



New
Edition

SCRIPTING LANGUAGE

Ramesh Singh Saud Basant Chapagain

Syllabus

Course Title: Scripting Language (3 Cr.)

Course No: CACS254

Nature of the Course: Theory + Lab

Year/Semester: II/IV

Class Load: 6 Hrs./Week (Theory: 3 Hrs, Practical: 3 Hrs.)

Course Description:

This course includes both theoretical as well as practical knowledge of scripting language (client side & server side) along with database connectivity. So that students can develop more effective and efficient web based application packages.

Course Objectives:

The general objectives of this course are to provide fundamental concepts of server side script programming and client script programming using JavaScript and PHP respectively along with database connectivity.

Course Content

Unit 1: Client Side Scripting

15 Hrs

JavaScript: Introduction, Need of Client Side Scripting Language, Formatting and Coding Conventions, JavaScript Files, Comments, Embedding JavaScript in HTML, Using Script Tag, No script Tag, Operators, Control Structures, Array and for each Loop, Defining and Invoking Functions, Built in Objects, Date Objects, Interacting with the Browser, Windows & Frames, Document Object Model, Event Handling, Forms, Cookies, Handling Regular Expression, Client Side Validations

Unit 2: Server Side Scripting with Database Connectivity

15 Hrs

PHP (8 hrs): Introduction to Server Side Scripting Language, PHP introduction, Basic PHP Syntax, Comments in PHP, Variables, PHP Operators. Control Structures (If else, switch, all loops), Arrays, For Each Loop, Functions, Form Handling, PHP \$_GET, PHP \$_POST, PHP \$_REQUEST, PHP date () Function, PHP include File, File Handling, File Uploading, PHP Sessions, Sending Emails, PHP Cookies.

MySQL (7 hrs): Introduction to MySQL, PHP MySQL Connect to a Database, Closing a Connection, MySQL Data Types. MySQL Insert. MySQL Select. MySQL Where Clause, MySQL Delete, MySQL Update, MySQL Aggregate Functions (sum, avg, count etc); MySQL Order by and Group by Clause, MySQL Subqueries, MySQL Joins,

Unit 3: Advanced Server Side Scripting

15 Hrs

Object Oriented Programming in PHP: Classes and Objects, Defining and Using properties and methods, Constructors and Destructors, Method Overriding, Encapsulation, Inheritance, Polymorphism, Static Members, Exception Handling

AJAX (Asynchronous JavaScript and XML): Using PHP, Using PHP + MySQL.

JQuery: Playing with elements, Hiding and Unhiding Images, JQuery UI

JOOMLA: Introduction To CMS, Installation, Handling Joomla Back End. Customization In Joomla, Introduction to Extensions, Installation and Uses Of Extensions in Joomla, Template Development in Joomla, Artiseer (IDE), Module Development in Joomla, Component Development in Joomla, Introduction to MVC (Model, View and Controller)

Word Press Administrator Level: Theme Integration, Creating Pages, Managing Posts, Managing Widgets.

Laboratory Works:

Laboratory works should be done covering all the topics listed above and a small project work should be carried out using the concept learnt in this course. Project should be assigned on Individual Basis.

Teaching Methods

The general teaching pedagogy includes class lectures, group discussions, case studies, guest lectures, research work, project work, assignments (theoretical and practical), and examinations (written and verbal), depending upon the nature of the topics. The teaching faculty will determine the choice of teaching pedagogy as per the need of the topics.

Evaluation

Examination Scheme				
Internal Assessment		External Assessment		Total
Theory	Practical	Theory	Practical	
20	20 (3 Hrs.)	60 (3 Hrs.)	-	100

Text Books:

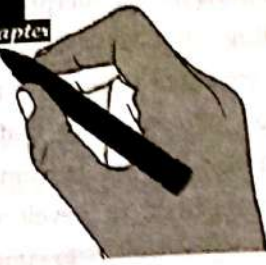
1. Harvey M. Deitel, Paul J. Deitel & Abbey Deitel, "Internet and World Wide Web: How to Program", 5th Edition, Pearson Education, 2012, ISBN: 9780273764021
2. Robin Nixon, "Learning PHP MySQL JavaScript and CSS", 2nd Edition, O' Reilly Media, 2012, ISBN 978-1449319267

Reference Books:

1. David Flanagan, "JavaScript: The Definitive Guide", O' Reilly Media, 2011, ISBN: 139780596805524
2. David Sawyer McFarland, "JavaScript & jQuery: The Missing", 2nd Edition, Pogue press, 2011, ISBN: 978-1449399023
3. Luke Welling & Laura Thomson, "PHP and MySQL Web Development", 5th Edition, Developer's Library, 2014, ISBN: 978-0321833891

1

Chapter



CLIENT SIDE SCRIPTING



CHAPTER OUTLINE

After comprehensive study of this chapter, you will be able to:

- **JavaScript:** Introduction, Need of Client Side Scripting Language, Formatting and Coding Conventions, JavaScript Files, Comments, Embedding JavaScript in HTML, Using Script Tag, No script Tag, Operators, Control Structures, Array and for each Loop, Defining and Invoking Functions, Built in Objects, Date Objects, Interacting with the Browser, Windows & Frames, Document Object Model, Event Handling, Forms, Cookies, Handling Regular Expression, Client Side Validations

SCRIPTING LANGUAGE

In computer programming, a script is a piece of program or sequence of instructions that is interpreted or carried out by another program rather than by the computer processor. Scripting language is defined as a programming language which contains series of commands that are able to be executed without the need for compiling. It is a language is used to write scripts and contain a series of commands that are interpreted one by one at runtime unlike programming languages that are compiled first before running. Scripting language are designed for gluing already existing components together to develop an application and uses type less approach to achieve a higher level of programming as well as to simplify connections between components to provide rapid application development. It brings new functions to applications and glue complex system together.

Scripting languages assume that there already exist a collection of useful components written in another language. They are not intended to write applications from scratch but intended primarily for plugging components together. Some of the most widely used scripting languages are JavaScript, VBScript, PHP, Perl, Python, Ruby, and ASP. Since a scripting language is normally used in conjunction with another programming language, they are often found alongside HTML, Java or C++.

Difference between Scripting Language and Programming Language:

Programming Language	Scripting Language
A programming language is a formal language that specifies a set of instruction that can be used to produce various kinds of output. Programming languages generally consist of instructions for a computer.	A scripting or script language is a programming language that supports scripts: programs written for a special runtime environment that automate the execution of tasks that could alternatively be executed one-by-one by a human operator. Scripting languages are often interpreted (rather than compiled).
It's a Compiler-based language	It an Interpreter based language
It is used for developing anything needful from scratch	It is used to combine existing components.
Unlike scripted language the programming language are compiled into a more compact form that does not need to be interpreted by another application in the same way. The compiled result is stand-alone.	Scripting language are interpreted within another program like JavaScript is put within HTML and then interpreted by the browser.
They run independent of a parent program.	These run inside another program.
They are designed to give full usage of a language.	They are designed to make coding fast and simple.
One shot conversion.	Line by line conversion.
It create Executable file like .exe.	It does not create executable file.
It takes longer to develop as more code or whole program needs to be written.	It takes less time to code as it needs less coding.
It does not require a host, is self-executable.	It requires a host.
Example: C, C++, C#, Java, Basic.	Example: JavaScript, VBScript, Perl, Python, Ruby, PHP

Basic working process of websites.

All websites run on three components: the server, the database, and the client. The client is simply the browser a person is using to view a site, and it's where client-side technology is unpacked and processed. The server is at a remote location anywhere in the world – housing data, running a site's back-end architecture, processing requests, and sending pages to the browser. The client is anywhere your users are viewing your site: mobile devices, laptops, or desktop computers. Server-side scripting is executed by a web server; client-side scripting is executed by a browser.

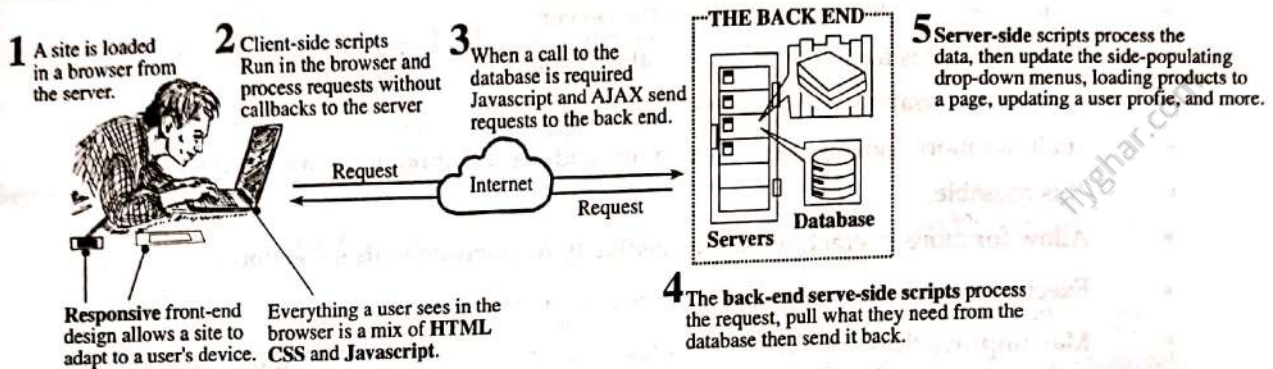


Fig: Working process of web application. (Redraw)

CLIENT SIDE SCRIPTING LANGUAGE

Client side scripting can be defined as a code that is present in a client's HTML page and usually attached to the browser in a language that is compatible with the browser. Client-side Scripting is used when the client's browser has all the code and the page is modified on the basis of the client's information. The Web Browser executes the client-side scripting that locates within the user's computer. Client-side scripts are also known as the embedded script (as they are often embedded within an HTML or XHTML document). The browser temporarily downloads that code, and then, apart from the server, processes it. If it needs to request additional information in response to user clicks, mouse-overs, etc. (called "events"), a request is sent back to the server.

Client-side scripting is always evolving—it's growing simpler, more nimble, and easier to use. As a result, sites are faster, more efficient, and less work is left up to the server.

NEEDS OF CLIENT SIDE SCRIPTING

Using a client-side script for interactivity in a web application usually provides the appearance of better performance than using a server-side script to implement the same functionality because the client-side script does not incur the delay of a round trip between the client and the server, making the web application more responsive. This is crucial for a dynamic user experience in a web application. As an extreme example, imagine that a developer wants to limit input to a text box to only numbers. Each time the user hits a key, the character entered must be examined and thrown away if it's not a number or cursor movement key. Imagine the delay if the application had to wait for a round trip between the client and server and redraw the page for each keystroke. Obviously, such a feature would never be practical in a Web application if it were not for client-side scripting. Client-side scripting allows many features to be easily incorporated in Web applications that would otherwise be impractical.

In addition, client-side scripting offers the benefit of off-loading part of the computational demands of an application from the server to the client. The client-side computer is usually wasting most of its processing potential, idly waiting for the user to do something. Web servers, on the other hand, are usually taxed to their limit, serving hundreds or thousands of concurrent users.

Advantages of Client-side Scripting

- It Immediately response to users
- Through this More responsive design and interaction with the user is possible.
- It does not need to send requests to the server.
- Loading time is faster compared to other models.
- It improves usability in browsers.
- It allows more control over the behavior and presentation of the web widget tools.
- It is reusable.
- Allow for more interactivity by immediately responding to users' actions.
- Execute quickly because they do not require a trip to the server.
- May improve the usability of web sites for users whose browsers support scripts.
- Can give developers more control over the look and behavior of their web widgets.
- Can be substituted with alternatives (for example, HTML) if users' browsers do not support scripts
- Are reusable and obtainable from many free resources.

Disadvantages of Client-side Scripting

- Not all browsers support scripts, therefore, users might experience errors if no alternatives have been provided.
- Different browsers and browser versions support scripts differently, thus more quality assurance testing is required.
- More development time and effort might be required (if the scripts are not already available through other resources).
- Developers have more control over the look and behavior of their web widgets; however usability problems can arise if a Web widget looks like a standard control but behave differently or vice-versa.

SERVER SIDE SCRIPTING LANGUAGE

Server-side scripting is a method of designing websites so that the process or user request is run on the originating server. Server-side scripting languages run on a web server and they returns generally HTML page to the client. When a client sends a request, the server responds by sending content via HTTP. It provide an interface to the user and are used to limit access to proprietary data and help keep control of the script source code. one significant benefit of server-side scripts is they are not viewable by the public like client-side scripts. It is used to make web site dynamic and allow to access database and do many other database related tasks. Examples for server side scripting languages: PHP, JSP, ASP, Asp. Net, Ruby Perl etc.

COMPARISON BETWEEN CLIENT-SIDE AND SERVER-SIDE SCRIPTING

Client-side scripting

- It Works at the front end and script are visible among the users.
- Does not need interaction with the server.
- Can reduce the load to the server.
- Response from a client-side script is faster because the scripts are processed on the local computer.
- Cannot connect to the databases on the web server.
- It is relatively insecure.
- Client side scripting languages: JavaScript, VB script, etc.

Server-side Scripting

- It works in the back end which could not be visible at the client end.
- It requires server interaction.
- It could effectively customize the web pages and provide dynamic websites.
- Response from a server-side script is slower because the scripts are processed remotely.
- Server side scripting is used to connect to the databases that is on web server.
- Relatively secure.
- Examples for server side scripting languages: PHP, JSP, ASP, Asp. Net, Ruby, Perl etc.

JavaScript

JavaScript is a programming language used primarily by web browsers to create a dynamic and interactive experience for the user. It is a lightweight, interpreted programming language. It is designed for creating network-centric applications. Most of the functions and applications that make the Internet indispensable to modern life are coded in some form of JavaScript. It is complimentary to and integrated with Java. It is very easy to implement because it is integrated with HTML. It is open and cross-platform.

JavaScript is a scripting or programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc.

When JavaScript was created, it initially had another name: "LiveScript". But Java was very popular at that time, so it was decided that positioning a new language as a "younger brother" of Java would help.

But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript, and now it has no relation to Java at all.

Today, JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.

A JavaScript engine is a computer program that executes JavaScript (JS) code. The first JavaScript engines were mere interpreters, but all relevant modern engines utilize just-in-time compilation for improved performance.

JavaScript engines are typically developed by web browser vendors, and every major browser has one. In a browser, the JavaScript engine runs in concert with the rendering engine via the Document Object Model.

The use of JavaScript engines is not limited to browsers. For example, the Chrome V8 engine is a core component of the popular Node.js runtime system.

The browser has an embedded engine sometimes called a "JavaScript virtual machine".

Different engines have different "codenames". For example:

- V8 - in Chrome and Opera.
- SpiderMonkey - in Firefox.
- There are other codenames like "Trident" and "Chakra" for different versions of IE, "ChakraCore" for Microsoft Edge, "Nitro" and "SquirrelFish" for Safari, etc.

The earliest incarnations of JavaScript were developed by Brendan Eich in 1995 for the Netscape Navigator Web browser. At the time, Web pages were static, offering little user interaction beyond clicking links and loading new pages. For the first time, JavaScript enabled animation, adaptive content and form validation on the page.

For many years, JavaScript only functioned on a limited number of browsers. Microsoft's Internet Explorer, the largest browser base, did not support JavaScript until much later. Instead, Microsoft created its own proprietary client-side script called JScript. In the early days of Web development, programmers who wished to create dynamic websites were often forced to choose one browser family over the other. This was less than ideal because it made the Internet less universally accessible.

JavaScript did not become standardized and widely adopted until 1999. Even after standardization, browser compatibility remained an issue for over a decade.

Uses of JavaScript

Adding interactivity to website: If you want a website to be more than a static page of text, you'll need to do some JavaScripting.

Developing mobile applications: JavaScript isn't just for websites, it is used to create those apps you have on your phone and tablet as well.

Creating web browser based games: Ever played a game directly from your web browser? JavaScript probably helped make that happen.

Back end web development: JavaScript is mostly used on the front end of applications, but it is a versatile scripting language so, it can be used on back end infrastructure, too.

Some specific uses of JavaScript:

- Client-side validation
- Dynamic drop-down menus
- Displaying date and time
- Validate user input in an HTML form before sending the data to a server.
- Build forms that respond to user input without accessing a server.
- Change the appearance of HTML documents and dynamically write HTML into separate Windows.
- Open and close new windows or frames.
- Manipulate HTML "layers" including hiding, moving, and allowing the user to drag them around a browser window.
- Build small but complete client side programs.

- Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)
- Displaying clocks etc.

Features of JavaScript

- JavaScript is an object-based scripting language.
- Giving the user more control over the browser.
- It Handling dates and time.
- It Detecting the user's browser and OS,
- It is light weighted.
- JavaScript is a scripting language and it is not java.
- JavaScript is interpreter based scripting language.
- JavaScript is case sensitive.
- JavaScript is object based language as it provides predefined objects.
- Every statement in JavaScript must be terminated with semicolon (;).
- Most of the JavaScript control statements syntax is same as syntax of control statements in C language.
- An important part of JavaScript is the ability to create new functions within scripts. Declare a function in JavaScript using **function** keyword.

Advantages of JavaScript

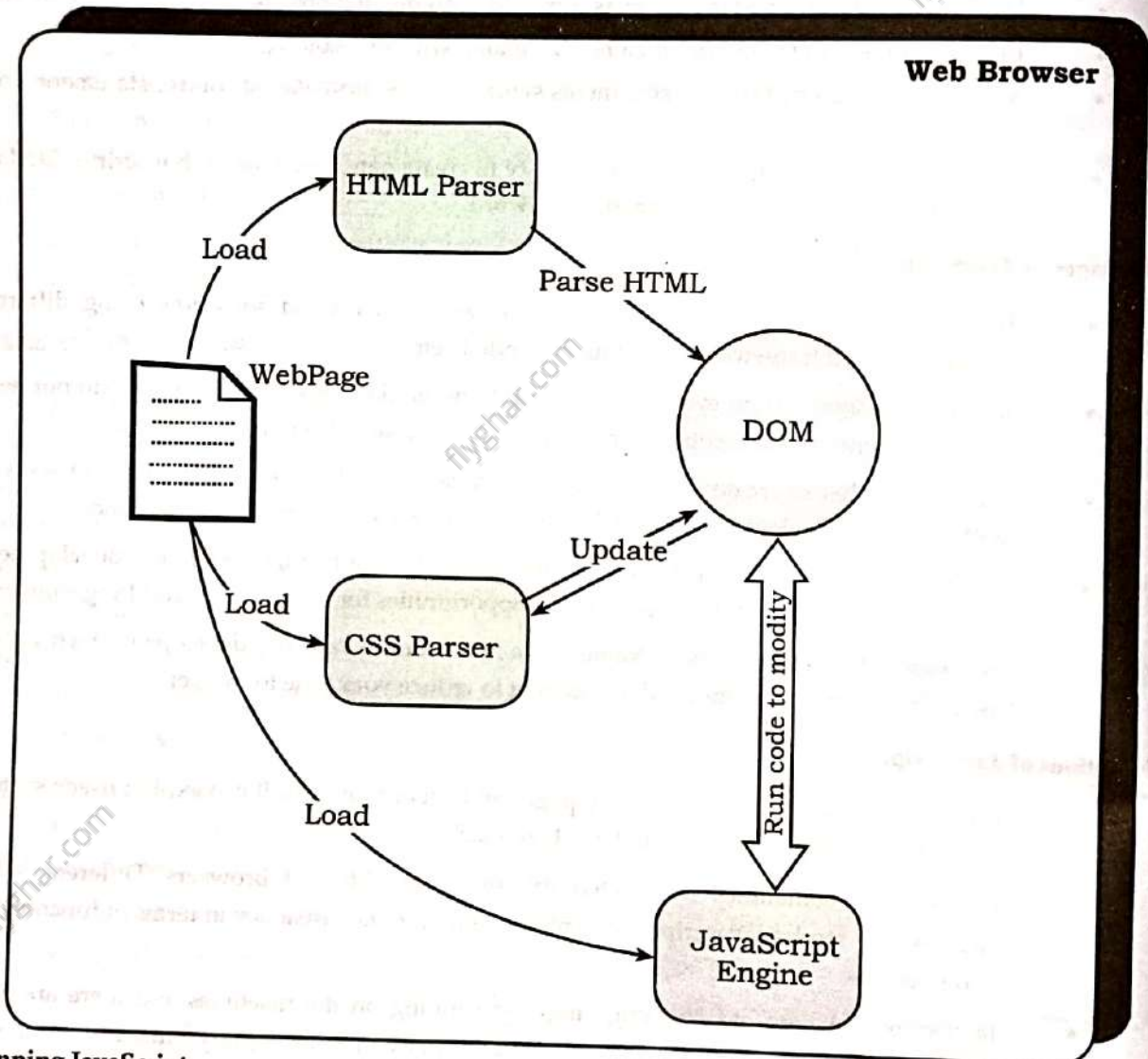
- It helps you developing great front-end as well as back-end softwares using different Javascript based frameworks like jQuery, Node.js etc.
- Javascript is installed on every modern web browser. So to learn Javascript we do not need any special environment setup. For example Chrome, Mozilla Firefox, Safari etc
- Javascript helps to create beautiful and crazy fast websites. we can develop our website with a console like look and feel and give users the best Graphical User Experience.
- JavaScript usage has now extended to mobile app development, desktop app development, and game development. This opens many opportunities for you as Javascript Programmer.
- In Javascript there are tons of frameworks and Libraries already developed which can be used directly in your software development to reduce your time to market.

Limitations of JavaScript

- JavaScript is explicitly added to web pages and client browsers, it can exploit user's system, so malicious code can be executed on client machine.
- JavaScript is sometimes interpreted differently by different browsers. Different layout engines may render Javascript differently resulting in inconsistency in terms of functionality and interface.
- JavaScript is a very old scripting language running on the machines and there are other technologies which are doing the same thing in place of it (ex. jQuery) in a better and easy way.
- If we disable JavaScript in browser, the entire JavaScript code is not run.
- JavaScript file is download on client machine so anyone can read the code and reuse it.

- Client-side JavaScript does not allow the reading or writing of files.
- It cannot be used for networking applications because there is no such support available.
- It doesn't have any multithreading or multiprocessor capabilities.
- Cannot access web pages hosted on a different domain.
- Cannot access databases.
- Depends a lot on the browser.
- Inability to use local devices.
- JavaScript can be disabled.
- Not Search Engine Friendly.
- JavaScript cannot protect your page source or images.

How Does JavaScript inside Web Browser



Running JavaScript

Being a scripting language, JavaScript cannot run on its own. In fact, the browser is responsible for running JavaScript code. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it is up to the browser to execute it.

FORMATTING AND CODING CONVENTIONS

Coding conventions are style guidelines for programming. The JavaScript style guidelines are directions to regulate programming in JavaScript. These coding prescription are beneficiary to write code in JavaScript without any syntactical errors. It covers the naming of variables, direction to put whitespaces and semicolons and various statement guidelines.

Main benefit of coding conventions is we can improve the quality, readability & make code maintenance easier.

Some coding conventions are:

Indentation:

We always indent our code using two spaces & our code must not end with trailing whitespaces.

Note: Do not use tabs (tabulators) for indentation. Different editors interpret tabs differently.

Example:

```
function getSum(number1,number2){  
    return number1+number2;  
}
```

Spaces, all brackets:

Blank spaces should always be used in the following circumstances:

- A keyword followed by (or) must provide whitespaces before and after it.
- Example:if (condition) { }
- All keywords like 'function', 'typeof' etc needs extra attention for whitespaces & a space provided after it.

```
function getPercentage(total,item) {  
    return total/item;  
}
```

Variable declaration:

All variables must be declared with var before they are used. We mostly declare the variable at the beginning of the function.

```
var name = "Basanta Chapagain";  
var programName = "BCA";  
var subjectName = "Scripting Lanauge";
```

Object guidelines:

General rules for object definitions:

There are various types of objects in JavaScript as array, function, dates, user define objects. Basically, every object follows a simple coding style written below.

In JavaScript, we put an equal (=) & opening bracket after declaring the name of the object. We define property as using a colon: and put string values inside quotes "xyz" but not on numeric value. We put a semicolon at the end of one property and do not put at the end of last property.


```
var student = {
  firstName: "Ramesh",
  middleName: "Singh",
  lastName: "Saud",
  age: 38
};
```

Same code can be written as compressed like below.

```
var student = { firstName: "Ramesh", middleName: "Singh", lastName: "Saud", age: 38};
```

Camel casing for identifier names

Variable and function names should be full words, using camel case with a lowercase first letter. Names should be descriptive, but not excessively so. Exceptions are allowed for iterators, such as the use of `i` to represent the index in a loop. We mostly use camel case for the identifier. In camel casing, the first letter of each function should be lowercase and the first letter of subsequent words should be capitalized.

```
actorName = "John Abraham";
totalInvestment = 500000000;
totalCollection = 594500023
profit = totalCollection - totalInverstment
```

Loops and Control Structures

We use same convention of parenthesis & brackets for control structures as follows,

- Always follow whitespace after an identifier of control statements.
- Always provide a whitespace & line break after the semicolon.
- Always provide a whitespace after every comma.

```
if (condition) {
  actionName();
} else {
  defaultAction();
}
```

Statement Guidelines:

- **Simple statement:** The statement which consists only single line. Always end with a semicolon.
- **Compound statement:** In JavaScript, we put the space & opening bracket { and have a list of statements with closing bracket } at the end. For example: - at the end of function name declaration before defining it. We put the closing bracket on a new line without following whitespace trail. We do not put the semicolon at the end of these type of statements.
- **Multiline statements:** In JavaScript Whenever a statement is not small enough to fit on one line, line breaks must occur after an operator. To improve readability lines should break into multiple lines.


```
//simple statement
var books = ["Scripting", "DBMS", "NumericalMethod", "Operating System"];
//compound statement
for (number = 0; number < 56; i++) {
total += number;
}
//multiline statement
if (
age === 123
&& name === 'abc'
) {
```

Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable. JavaScript comments can also be used to prevent execution, when testing alternative code.

Types:

- Single Line Comments
- Multi Line Comments

It is most common to use single line comments. Block comments are often used for formal documentation.

Single Line Comments

Single line comments start with //. Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

```
//var name = "Basanta Chapagain";
```

Single line comment also use at the end of each line to explain the code

```
var firstNumber = 80; // Declare firstNumber, give it the value of 80
var secondNumber = 92; // Declare secondNumber, give it the value of 92
// Declare total and assign sum of firstNumber and secondNumber
var total = firstNumber + secondNumber
```

Multi-line Comments

Multi-line comments start with /* and end with */. Any text between /* and */ will be ignored by JavaScript.

```
/*
Declare firstNumber, give it the value of 80
Declare secondNumber, give it the value of 92
Declare total and assign sum of firstNumber and secondNumber
*/
var firstNumber = 80;
var secondNumber = 92;
var total = firstNumber + secondNumber;
```


Embedding JavaScript in HTML

JavaScript is applied to your HTML page in a similar manner to CSS. Whereas CSS uses `<link>` elements to apply external stylesheets and `<style>` elements to apply internal stylesheets to HTML, JavaScript only needs one friend in the world of HTML the `<script>` element.

Different way of using JavaScript into your web page are:

- Internal JavaScript
- External JavaScript
- Inline JavaScript

Inline JavaScript

When JavaScript is written within the html element using attributes related to events of the element then it is called as inline JavaScript.

Steps:

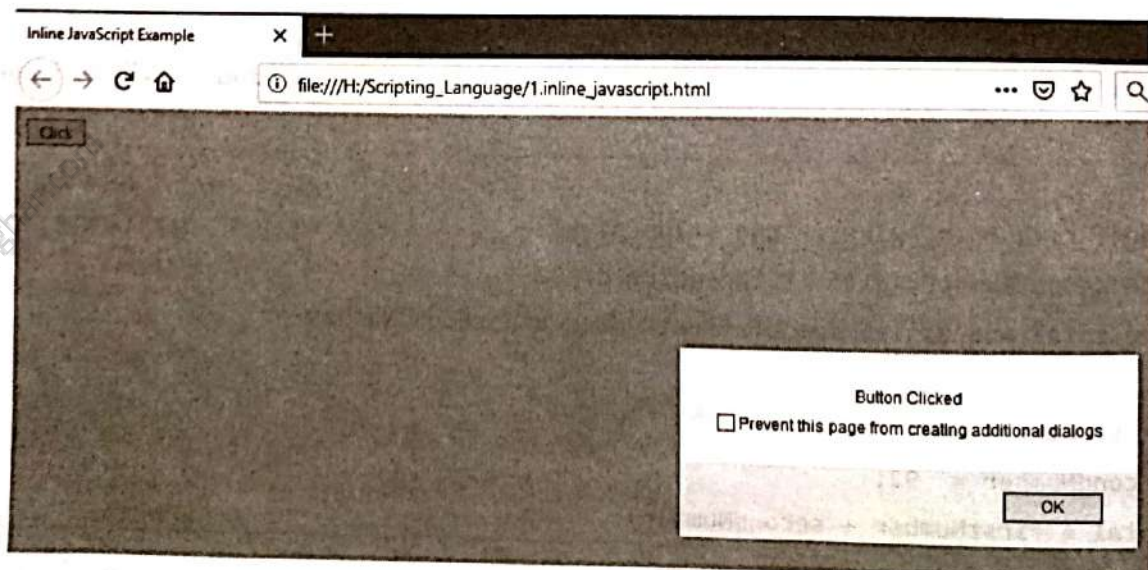
- Create your html file. (1.inline_javascript.html)
- Write following code and save it.
- Go to your file location and run it with your browser.

Example

```
<html>
<head>
<title>Inline JavaScript Example</title>
</head>
<body>
<form>
<input type="button" value="Click" onclick="alert('Button Clicked')"/>
</form>
</body>
</html>
```

For above example onclick is html attribute for JavaScript event and `onclick="alert('Button Clicked')"` is the inline JavaScript code. When user press click button on the web browser the event is captured and display alert popup with button clicked message.

Output



Internal JavaScript

When JavaScript is written within the <head> section using element then it is called as internal JavaScript with <script> tag. document.write function prints the specified text to the page.

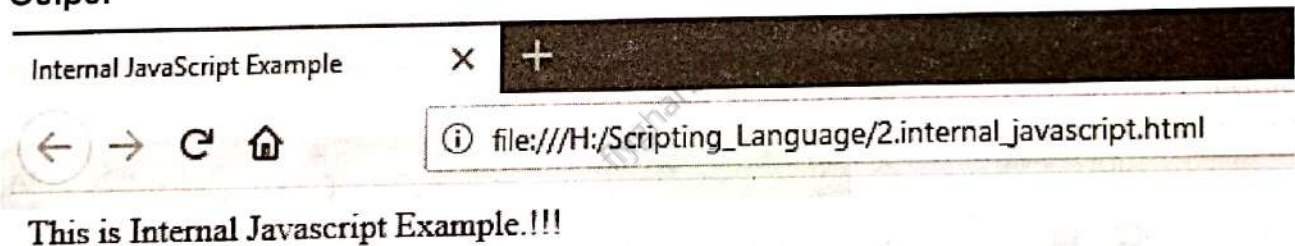
Steps:

- Create your html file. (2.internal_javascript.html)
- Write following code and save it.
- Go to your file location and run it with your browser.

Example

```
<html>
<head>
<title>Internal JavaScript Example</title>
<script type="text/javascript">
    document.write("This is Internal Javascript Example.!!!");
</script>
</head>
<body>
</body>
</html>
```

Output



Internal JavaScript is also can be written within body tag.

```
<html>
<head>
<title>Internal JavaScript Example</title>
</head>
<body>
<script type="text/javascript">
    document.write("This is Internal Javascript Example.!!!");
</script>
</body>
</html>
```

External JavaScript

JavaScript code are stored in separate external file using the .js extension (Ex: script.js). In the HTML file, the <script> tag can also be used to indicate the location of a JavaScript file. The src attribute is assigned the path and filename of the file. If you are using the same routine in several places, or you have a larger script, it can be an advantage to have the script externally. The way to do this, is by creating a separate file with the extension .js, i.e. a JavaScript file.

Steps:

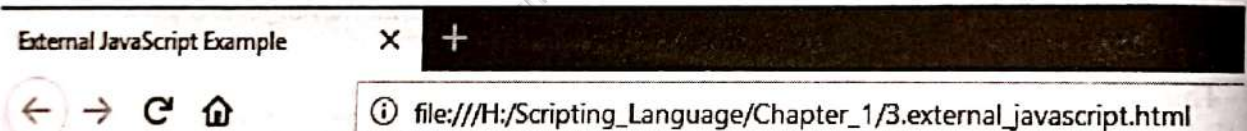
- Create your html file. (3.external_javascript.html)
- Create your JavaScript file inside js folder. (3.external_javascript.js)
- Write following code into related file and save it.
- Go to your file location and run it with your browser.

JavaScript File

```
//external javascript file to print information into web page
document.write("This is external Javascript Example.!!!");
```

HTML File**Example**

```
<html>
<head>
<title>External JavaScript Example</title>
<script type="text/javascript" src="js/3.external_javascript.js"></script>
</head>
<body>
</body>
</html>
```

Output

This is external Javascript Example.!!!

When to Use Internal and External JavaScript Code?

If you have only a few lines of code that is specific to a particular webpage, then it is better to keep your JavaScript code internally within your HTML document.

On the other hand, if your JavaScript code is used in many web pages, then you should consider keeping your code in a separate file. In that case, if you wish to make some changes to your code, you just have to change only one file which makes code maintenance easy. If your code is too long, then also it is better to keep it in a separate file. This helps in easy debugging.

JAVASCRIPT FILES

JavaScript code are stored in separate external file using the .js extension (Ex: script.js). In the HTML file, the <script> tag can also be used to indicate the location of a JavaScript file. The src attribute is assigned the path and filename of the file. If you are using the same routine in several places, or you have a larger script, it can be an advantage to have the script externally. The way to do this, is by creating a separate file with the extension .js, i.e. a JavaScript file.

Example Steps:

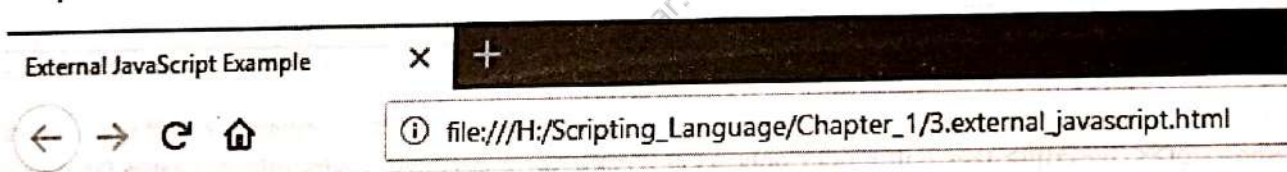
- Create your html file. (3.external_javascript.html)
- Create your JavaScript file inside js folder. (3.external_javascript.js)
- Write following code into related file and save it.
- Go to your file location and run it with your browser.

JavaScript File

```
//external javascript file to print information into web page  
document.write("This is external Javascript Example.!!!");
```

HTML File**Example**

```
<html>  
<head>  
<title>External JavaScript Example</title>  
<script type="text/javascript" src="js/3.external_javascript.js"></script>  
</head>  
<body>  
</body>  
</html>
```

Output

This is external Javascript Example.!!!

JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable. These are a meaningful way to deliver message. Comments add information about the code, warnings or suggestions so that end user can easily interpret the code. The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser. Comments are used to prevent the execution of statements.

- Comments in JavaScript are used to explain the code and make the program more readable for developers.
- In programming, comments can also be used to prevent some code lines from being executed. This is useful when testing.
- There are single-line comments (which comment out one line or a part of one line) and multi-line comments (which comment out a block of code).
- Multi-line comments are more often used for formal documentations.

We can use comments in JavaScript in two ways:

- Single Line Comments
- Multi Line Comments

Single Line Comments

It is represented by double forward slashes (//). It can be used before and after the statement. Single line comments start with //. Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

```
//var name = "Basanta Chapagain";
```

Single line comment also use at the end of each line to explain the code

```
var firstNumber = 80;    // Declare firstNumber, give it the value of 80
var secondNumber = 92;  // Declare secondNumber, give it the value of 92
// Declare total and assign sum of firstNumber and secondNumber
var total = firstNumber + secondNumber
```

Multi-line Comments

It can be used to add single as well as multi line comments. So, it is more convenient. It is represented by forward slash with asterisk then asterisk with forward slash (/* and end with */). Any text between /* and */ will be ignored by JavaScript. It can be used before, after and middle of the statement.

```
/*
Declare firstNumber, give it the value of 80
Declare secondNumber, give it the value of 92
Declare total and assign sum of firstNumber and secondNumber
*/
var firstNumber = 80;
var secondNumber = 92;
var total = firstNumber + secondNumber;
```

NoScriptTag

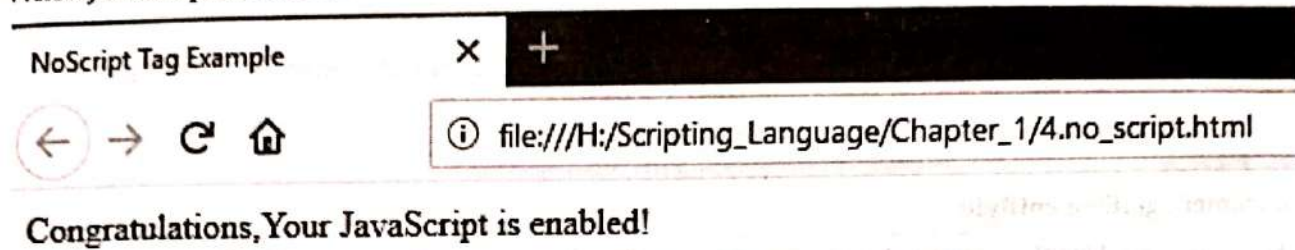
As the name implies, we can use no script to render alternate content. Anything within <noscript></noscript> tags will render only when JavaScript is disabled. Users might disable JavaScript for a number of reasons. A handful of people even install browser extensions like NoScript to prevent the browser from running JavaScript. It's better to have the default be non-JavaScript, and then let a JavaScript code overwrite with a JavaScript enabled page.

Example

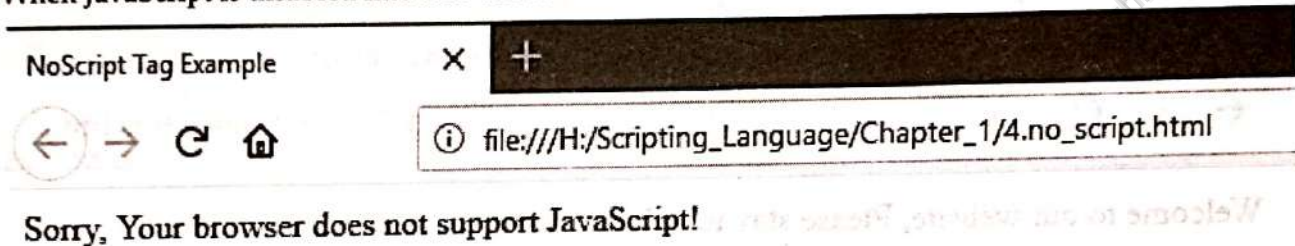
```
<!DOCTYPE html>
<html>
<head>
<title>NoScript Tag Example</title>
</head>
<body>
<script type = "text/JavaScript">
    document.write("Congratulations,Your JavaScript is enabled!")
</script>
<noscript>
Sorry,Your browser does not support JavaScript!
</noscript>
</body>
</html>
```


Output

When JavaScript is enabled



When JavaScript is disabled into web browser



Identifier in JavaScript

An identifier is a sequence of characters in the code that identifies a variable, function/method, or property. In JavaScript, identifiers are case-sensitive and can contain Unicode letters, \$, _ and digits (0-9), but may not start with a digit. An identifier differs from a string in that a string is data, while an identifier is part of the code. In JavaScript, there is no way to convert identifiers to strings, but sometimes it is possible to parse strings into identifiers.

1. **Variable:** A variable is a named location for storing a value. That way an unpredictable value can be accessed through a predetermined name.
2. **Function:** A function is a code snippet that can be called by other code or by itself, or a variable that refers to the function. When a function is called, arguments are passed to the function as input, and the function can optionally return an output.
3. **Property:** The term property can have several meanings depending on the context.

Basic Display function in JavaScript

There are multiple ways to display text on a web page in JavaScript. Some of them are,

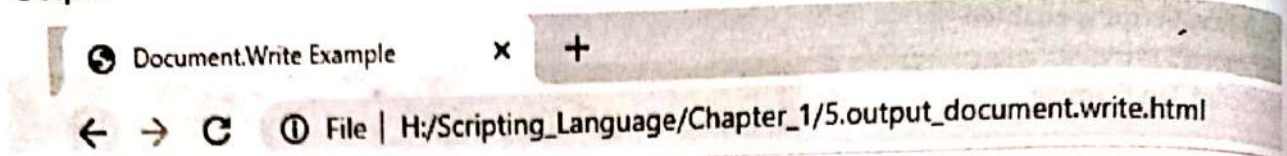
document.write

This is used to display a given text in a web page. This is mainly used for testing purpose. When document.write() is executed, after the page load it will erase all the contents in the web page and only show the text inside it.

Example

```
<script type="text/javascript">
    document.write("document.write is basic output method for JavaScript");
</script>
```

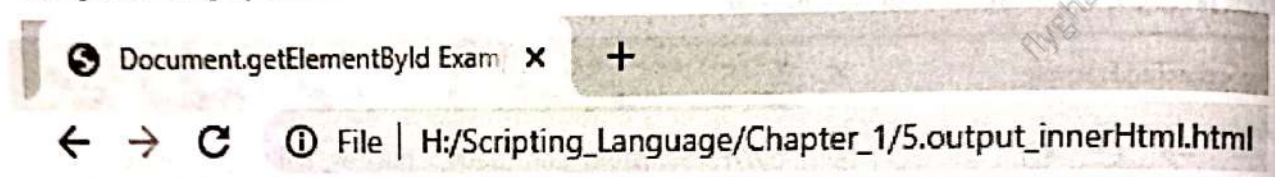
Output



document.write is basic output method for JavaScript

document.getElementById

To access an HTML element, JavaScript can use the document.getElementById(id) method. The id attribute defines the HTML element. The innerHTML property defines the HTML content. Following example will display content to element name welcome.



Welcome to our website, Please stay tuned us.

```
<!DOCTYPE html>
<html>
<head>
  <title>Document.getElementById Example</title>
</head>
<body>
  <p id="welcome"></p>
  <!-- Script to uses getElementById().innerHTML -->
  <script>
    document.getElementById("welcome").innerHTML = "Welcome to our
    website, Please stay tuned us.";
  </script>
</body>
</html>
```

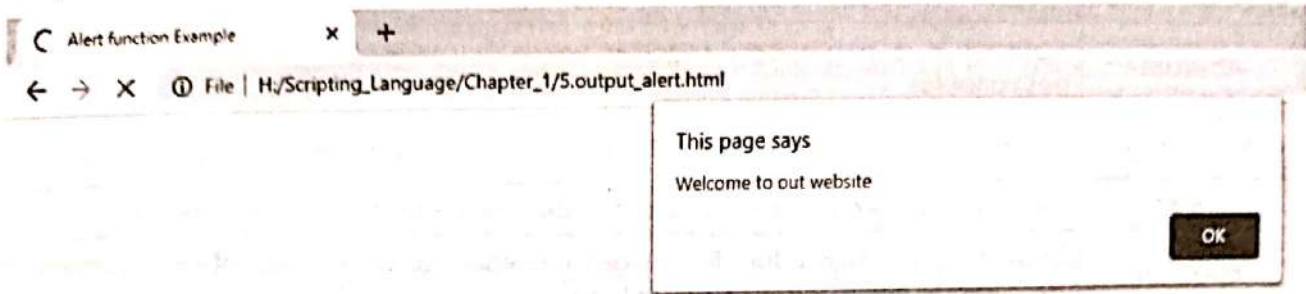
window.alert

This is one of the popup function defined by JavaScript. It displays the content using an alert box. We can simply write alert instead of window.alert.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Alert function Example</title>
</head>
<body>
  <script type="text/javascript">
    alert('Welcome to out website');
  </script>
</body>
</html>
```


Output



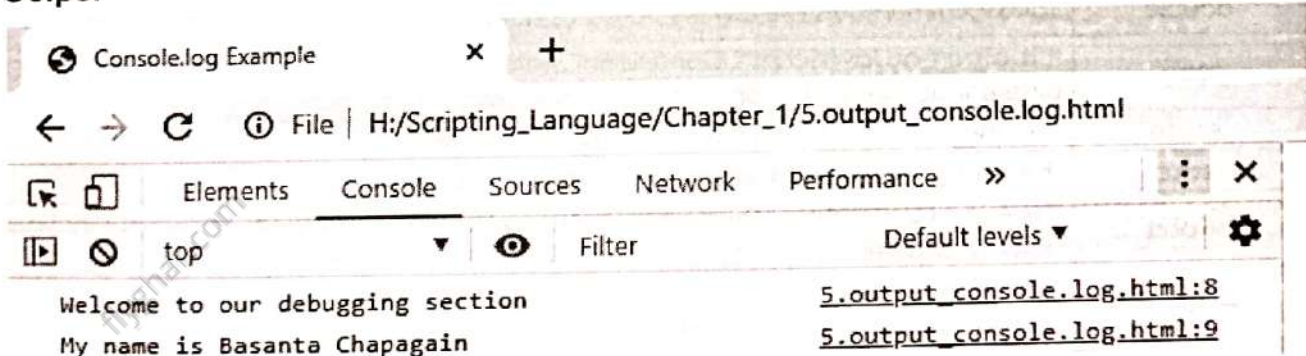
Console.log

The `console.log()` is a function in JavaScript which is used to print any kind of variables defined before in it or to just print any message that needs to be displayed to the user. It accepts a parameter which can be an array, an object or any message. It returns the value of the parameter given. To view output we have to go to console of browser. Most of the browser support `ctrl+shift+i` for console shortcut.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Console.log Example</title>
</head>
<body>
  <script type="text/javascript">
    console.log("Welcome to our debugging section");
    console.log("My name is Basanta Chapagain");
  </script>
</body>
</html>
```

Output



Keywords and reserve words

In any programming language, a reserved word or a reserved identifier is a word that cannot be used as an identifier, such as the name of a variable, function, or label. Thus, it is reserved and cannot be used for defining any of these. Keywords have special use in JavaScript and cannot be used as identifiers. Outside the keywords JavaScript defines reserved keywords which do not have special functionality now, but these may be used as future keywords. It should be noted that there are several different situations when these words are reserved.

Reserved Words	Description
Abstract	Abstract is a non-access modifier which is applicable for classes, methods but not variables.
Arguments	Argument is an array-like object lists the arguments and a few other properties.
Await	The keyword Await makes JavaScript wait until the promise returns a result.
Boolean	Boolean is used as a function to get the value of a variable, object, conditions, expressions, etc. in terms of true or false.
break	It breaks the loop and continues executing the code after the loop.
byte	This uses the javascript typeof() operator to detect what kind of type is variable x and y.
case	The switch statement evaluates an expression. The value of the expression is then compared with the values of each case in the structure.
catch	This statement allows you to define a block of code to be executed if an error occurs in the try block.
char	The char keyword is a data type that is used to store a single character.
class	Classes are another way of writing constructor functions which can be used in order to create a new object using the new keyword.
const	The const keyword is used to declare the constant in the function scope.
continue	Continue statement jumps over one iteration in the loop.
debugger	This statement stops the execution of JavaScript and calls the debugging function.
default	The default keyword can be within a switch statement, or with an export statement.
delete	You can use the delete keyword to delete the value of the property and the property itself.
do	The do/while statement is used when you want to run a loop at least one time.
double	JavaScript numbers are always stored as double precision floating point numbers
else	It is a part of JavaScript's Conditional Statements and you can use this to perform different actions based on different conditions.
enum	The enum keyword declares an enumerated type.
eval	The eval() function evaluates or executes an argument.
export	The export statement is used when creating JavaScript modules to export functions, objects, or primitive values from the module.
extends	The extends keyword is used in class declarations or class expressions to create a class which is a child of another class.
false	JavaScript includes a Boolean object to represent true or false. It considers null as false.
final	Final keyword is used to state that a value, class, or method cannot be changed.
finally	The finally statement lets you execute code, after try and catch, regardless of the result.
float	JavaScript numbers are always stored as double-precision floating-point numbers.
for	The for statement creates a loop that is executed as long as a condition is true.
function	A function is a JavaScript procedure that performs a task or calculates a value.
goto	The goto is accomplished in JavaScript using the break and continue keywords.

if	You can use if to specify a block of code to be executed if a specified condition is true.
implements	The implements keyword is used to implement an interface.
import	The import keyword is used to call the function to import the module dynamically.
in	The in operator returns true if the specified property is in the specified object or its prototype chain.
instance of	The instanceof operator in JavaScript is used to check the type of an object at run time.
int	It converts a primitive datatype, string, or array to its integer representation.
interface	The interface keyword is used to declare a special type of class that only contains abstract methods.
let	Let allows you to declare variables that are limited to a scope of a block statement, or expression on which it is used.
long	Long represents a 64 bit two's-complement integer value
native	The native keyword is applied to a method to indicate that the method is implemented in native code.
new	The new keyword constructs and returns an object of a constructor function.
null	Null in JavaScript that signifies 'no value' or nonexistence of any value.
package	A package is a directory with one or more modules inside of it and a package.json file which has metadata about the package.
private	A private function can only be used inside of its parent function or module.
protected	A public function can be used inside or outside of its parent function or module.
public	The protected keyword is an access modifier used for attributes, methods and constructors, making them accessible in the same package and subclasses.
return	Return statements cease execution in a function and return a value to the caller.
short	It is used to represent an integer value in JavaScript.
static	The static keyword defines a static method for a class.
super	The super keyword is used to access and call functions on an object's parent.
switch	The switch statement evaluates an expression.
synchronized	Synchronous code is executed in a sequence where each statement waits for the previous statement to finish before executing.
this	The JavaScript this keyword refers to the object it belongs to.
throw	Throw is the keyword that is used to raise an exception.
throws	The throws keyword indicates what exception type may be thrown by a method.
transient	Transient is a variables modifier used in serialization.
true	Boolean datatype returns either of two values i.e. true or false.
try	The try statement allows you to define a block of code to be tested for errors while it is being executed.
typeof	typeof is a JavaScript keyword that will return the type of a variable when you call it.
var	var is used to declare the variable in JavaScript.
void	The JavaScript void operator evaluates the given expression and then returns a value of undefined.
volatile	Volatile keyword is a qualifier that is applied to a variable when it is declared.
while	The while statement creates a loop that is executed while a specified condition is true.
with	The with statement extends the scope chain for a statement.
yield	The yield keyword returns an IteratorResult object with two properties and value.

Variable Declaration in JavaScript

Similar to other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container. JavaScript did not provide any data types for declaring variables and a variable in JavaScript can store any type of value. Hence JavaScript is loosely typed language. We can use a variable directly without declaring it. var keyword are use before variable name to declare any variable.

JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

Variable Declaration

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

```
<script type = "text/javascript">
    // variable decleration
    var name;
    var address;
    // declare multiple variables with the same var keyword
    var email, phone;
</script>
```

Storing a value in a variable is called variable initialization

You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

```
<script type = "text/javascript">
<!--
    var name = "Ram Kumar Chhetri";
    var age;
    age = 25;
    //-->
</script>
```

Declare a Variable without var Keyword

JavaScript allows variable declaration without var keyword. You must assign a value when you declare a variable without var keyword. It is not recommended to declare a variable without var keyword. It can accidently overwrite an existing global variable.

```
<script type = "text/javascript">
<!--
    name = "Ram Kumar Chhetri";
    age = 25;
    //-->
</script>
```


JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes. Scope of the variables declared without var keyword become global irrespective of wherever it is declared. Global variables can be accessed from anywhere in the web page.

Scope in JavaScript defines accessibility of variables, objects and functions.

There are two types of scope in JavaScript.

- Global scope
- Local scope

Global Scope

Variables declared outside of any function become global variables. Global variables can be accessed and modified from any function.

```
<script>
  var userName = "Ramesh";
  function modifyUserName() {
    userName = "RajKumar";
  };
  function showUserName() {
    alert(userName);
  };
  alert(userName); // display Ramesh
  modifyUserName();
  showUserName();// display RajKumar
</script>
```

In the above example, the variable userName becomes a global variable because it is declared outside of any function. A modifyUserName() function modifies userName as userName is a global variable and can be accessed inside any function. The same way, showUserName() function displays current value of userName variable. Changing value of global variable in any function will reflect throughout the program.

Local Scope

Variables declared inside any function with var keyword are called local variables. Local variables cannot be accessed or modified outside the function declaration. Function parameters are considered as local variables.

```
<script>
  function createUserName() {
    var userName = "Ramesh";
  }
  function showUserName() {
    alert(userName);
  }
  createUserName();
  showUserName(); // throws error: userName is not defined
</script>
```

In the above example, userName is local to createUserName() function. It cannot be accessed in showUserName() function or any other functions. It will throw an error if you try to access a variable which is not in the local or global scope. Use try catch block for exception handling.

Statements

JavaScript applications consist of statements with an appropriate syntax. A single statement may span multiple lines. Multiple statements may occur on a single line if each statement is separated by a semicolon. This isn't a keyword, but a group of keywords. A statement can set a variable equal to a value. A statement can also be a function call, i.e. `document.write()`. Statements define what the script will do and how it will be done.

Different types of JavaScript Statements

Conditional Statements

Conditional statements are used to control your scripts so that different actions can be taken depending on the situation. You may want to display a special image on your home page during the time of world cup game. This condition would depend on what day it was, and if it was a tournament period, then a special image would be displayed to your visitors. Without proper knowledge of the conditional statement, your scripts will not be as lively or dynamic as they could possibly be.

Conditional statements let you perform certain actions based on a logical condition. You specify a condition to test and the commands to execute if the condition is true. Example of one conditional statement is if statement.

If statement

If statement is an either/or switch. If a specified condition is true, JavaScript performs certain statements. If the condition is false, JavaScript might perform other statements. If statement looks as follows:

```
if (condition) {
    statements1 }
[else {
    statements2}]
```

The condition can be any JavaScript expression that evaluates to true or false. The conditional statements can be any JavaScript statements, including further nested if statements. Multiple statements must be enclosed in braces.

You may also compound the statements using else if to have multiple conditions tested in sequence, as follows:

```
if (condition)
    statement_1
[else if (condition_2)
    statement_2]
...
[else if (condition_n_1)
    statement_n_1]
[else
    statement_n]
```

To execute multiple statements, use a block statement (`{ ... }`) to group those statements. In general, it is a good practice to always use block statements, especially in code involving nested if statements:

Switch Statement

A switch statement allows a program to evaluate an expression and attempt to match the expression's value to a case label. If a match is found, the program executes the associated statement. A switch statement looks as follows:

```
switch (expression) {  
  case label_1:  
    statements_1  
    [break;]  
  case label_2:  
    statements_2  
    [break;]  
  ...  
  default:  
    statements_def  
    [break;]  
}
```

The program first looks for a case clause with a label matching the value of expression and then transfers control to that clause, executing the associated statements. If no matching label is found, the program looks for the optional default clause, and if found, transfers control to that clause, executing the associated statements. If no default clause is found, the program continues execution at the statement following the end of switch. By convention, the default clause is the last clause, but it does not need to be so.

The optional break statement associated with each case clause ensures that the program breaks out of switch once the matched statement is executed and continues execution at the statement following switch. If break is omitted, the program continues execution at the next statement in the switch statement.

Loop Statements

A loop is a set of commands that executes repeatedly until a specified condition is met. A loop statement checks to see if some condition is true, and if that condition is true, it executes a chunk of code. After the code is executed, the condition is checked again. If it is true, the process starts over again; if it is false, the loop stops and the rest of the code continues along.

JavaScript supports two loop structures: for and while. In addition, the break and continue statements are used specifically with loops.

for statement

A for loop repeats a loop until a specified condition evaluates to false. The JavaScript for loop is similar to the Java for loop and the traditional for loop in C. A for statement looks as follows:

```
for ([initial-expression;] [condition;] [increment-expression]) {  
  statements  
}
```

When for loop is encountered, the initial-expression is executed. The statements are executed as long as condition evaluates to true. The increment-expression is performed on each pass through the loop. The sequence of execution is as follows:

1. The initializing expression initial-expression, if any, is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity.
2. The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, for loop terminates.

3. The update expression increment-expression executes.
4. The statements execute and control returns to step 2.

While statement

A while statement repeats a loop as long as a specified condition evaluates to true. A while statement looks as follows:

```
while (condition) {
    statements
}
```

If the condition becomes false, the statements within the loop stop executing and control passes to the statement following the loop.

The condition test occurs only when the statements in the loop have been executed and the loop is about to be repeated. That is, the condition test is not continuous, but is performed once at the beginning of the loop and again just following the last statement in statements, each time control passes through the loop.

do...while Statement

The do...while statement repeats until a specified condition evaluates to false. A do...while statement looks as follows:

```
do
    statement
while (condition);
```

statement executes once before the condition is checked. To execute multiple statements, use a block statement ([...]) to group those statements. If condition is true, the statement executes again. At the end of every execution, the condition is checked. When the condition is false, execution stops and control passes to the statement following do...while.

Break statement

The break statement terminates the current while or for loop and transfers program control to the statement following the terminated loop.

Continue statement

A continue statement terminates execution of the block of statements in a while or for loop, and continues execution of the loop with the next iteration.

Object Manipulation Statements

JavaScript uses the for...in, for each...in, and with statements to manipulate objects.

for...in Statement

The for-in statement iterates a specified variable over all the properties of an object. For each distinct property, JavaScript executes the specified statements. A for...in statement looks as follows:

```
for (variable in object) {
    statements
}
```


Example

The following function takes as its argument an object and the object's name. It then iterates over all the object's properties and returns a string that lists the property names and their values.

```
function dump_props(obj, obj_name) {  
    var result = "";  
    for (var i in obj) {  
        result += obj_name + "." + i + " = " + obj[i] + "<br>";  
    }  
    result += "<hr>";  
    return result;  
}
```

For an object car with properties make and model, result would be:

car.make = Ford

car.model = Mustang

Arrays

Although it may be tempting to use this as a way to iterate array elements, because the for...in statement iterates over user-defined properties in addition to the array elements, if you modify the Array object, such as adding custom properties or methods, the for...in statement will return the name of your user-defined properties in addition to the numeric indexes. Thus it is better to use a traditional for loop with a numeric index when iterating over arrays.

for each...in Statement

for each ...in is a loop statement introduced in JavaScript 1.6. It is similar to for...in, but iterates over the values of object's properties, not their names.

```
var sum = 0;  
var obj = {prop1: 50, prop2: 90, prop3: 60, prop4: 56};  
for each (var item in obj) {  
    sum += item;  
}  
print(sum); // prints "256", which is 50+90+60+56
```

Comment Statements

Comments are author notations that explain what a script does. Comments are ignored by the interpreter. JavaScript supports Java and C++ style comments:

Comments on a single line are preceded by a double-slash (//).

Comments that span multiple lines are preceded by /* and followed by */:

Example

The following example shows two comments:

```
// This is a single-line comment.  
/* This is a multiple-line comment. It can be of any length, and  
you can put whatever you want here. */
```

Exception Handling Statements

You can throw exceptions using the `throw` statement and handle them using the `try...catch` statements.

- `throw` Statement
- `try...catch` Statement

`throw` Statement

Use the `throw` statement to throw an exception. When you throw an exception, you specify the expression containing the value to be thrown:

```
throw expression;
```

You may throw any expression, not just expressions of a specific type. The following code throws several exceptions of varying types:

```
throw "Error2";
throw 42;
throw true;
throw {toString: function() { return "I'm an object!"; } };
```

`try...catch` Statement

The `try...catch` statement marks a block of statements to try, and specifies one or more responses should an exception be thrown. If an exception is thrown, the `try...catch` statement catches it.

The `try...catch` statement consists of a `try` block, which contains one or more statements, and zero or more `catch` blocks, containing statements that specify what to do if an exception is thrown in the `try` block. That is, you want the `try` block to succeed, and if it does not succeed, you want control to pass to the `catch` block. If any statement within the `try` block (or in a function called from within the `try` block) throws an exception, control immediately shifts to the `catch` block. If no exception is thrown in the `try` block, the `catch` block is skipped. The `finally` block executes after the `try` and `catch` blocks execute but before the statements following the `try...catch` statement.

```
try {
    a = "ram";
    if ((typeof a) !== 'number') {
        throw new TypeError('Data type mismatch');
    }
}
catch (e){
    if(e instanceof Error){
        console.log('instance of general Error constructor');
    }
    if(e instanceof TypeError) {
        console.log(e.message);
    }
}
```


Expressions

An expression is any valid set of literals, variables, operators, and expressions that evaluates to a single value. The value may be a number, a string, or a logical value. Conceptually, there are two types of expressions: those that assign a value to a variable, and those that simply have a value. For example, the expression

$$x = 7$$

is an expression that assigns x the value 7. This expression itself evaluates to 7. Such expressions use assignment operators. On the other hand, the expression

$$3 + 4$$

Simply evaluates to 7; it does not perform an assignment. The operators used in such expressions are referred to simply as operators.

JavaScript has the following kinds of expressions:

- Arithmetic: evaluates to a number, for example
- String: evaluates to a character string, for example "Fred" or "234"
- Logical: evaluates to true or false

The special keyword null denotes a null value. In contrast, variables that have not been assigned a value are undefined, and cannot be used without a run-time error.

JavaScript Operators

An operator is a sign or symbol that performs some operation on single or multiple operands (data value) and produces a result. These are symbols or keywords that tell the JavaScript engine to perform some sort of actions. JavaScript Operators use either value or variable to compute some task. For example $x + y$, where $+$ sign is an operator and x is left operand and y is right operand. $+$ Operator adds two numeric values and produces a result.

Types of operator's in JavaScript are:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operator (Ternary Operator)

Arithmetic Operators

Arithmetic operators take numerical values (either literals or variables) as their operands and return a single numerical value. Arithmetic operators perform basic computations on their operands (the variables they operate on). Arithmetic operators are used to perform mathematical operations between numeric operands.

JavaScript supports the following arithmetic operators:

Operator	Description
+	Add two numeric operands). (Exception is addition operator behaves concatenate with string data type
-	Subtract right operand from left operand
*	Multiply two numeric operands
/	Divide left operand by right operand and return quotient
%	Modulus operator – Returns remainder of two operands
++	Increment operator, Increase operand value by one. It behaves as Pre-Increment and Post-Increment according to its uses.
--	Decrement operator. Decrease value by one. It behaves as Pre-Increment and Post-Increment according to its uses.

Example of arithmetic operator

Example

```

<!DOCTYPE html>
<html>
<head>
    <title>Arithmetic Operator</title>
</head>
<body>
<script type="text/javascript">
var numberVar1 = 10;
var numberVar2 = 3;
console.log('Sum of ' + numberVar1 + ' + ' + numberVar2 + ' is ' + (numberVar1 +
numberVar2));
console.log('Difference of ' + numberVar1 + ' - ' + numberVar2 + ' is ' + (numberVar1
- numberVar2));
console.log('Multiplication of ' + numberVar1 + ' * ' + numberVar2 + ' is ' +
(numberVar1 * numberVar2));
console.log('Quotient of ' + numberVar1 + ' / ' + numberVar2 + ' is ' + (numberVar1 /
numberVar2));
console.log('Reminder of ' + numberVar1 + ' % ' + numberVar2 + ' is ' + (numberVar1 %
numberVar2));
iValue = 5;
iValue1 = iValue++;
console.log('Value returned by iValue post increment is ' + iValue1);
console.log('Value of iValue after post increment is ' + iValue);

iValue = 5;
iValue1 = ++iValue;
console.log('Value returned by iValue pre increment is ' + iValue1);
console.log('Value of iValue after pre increment is ' + iValue);

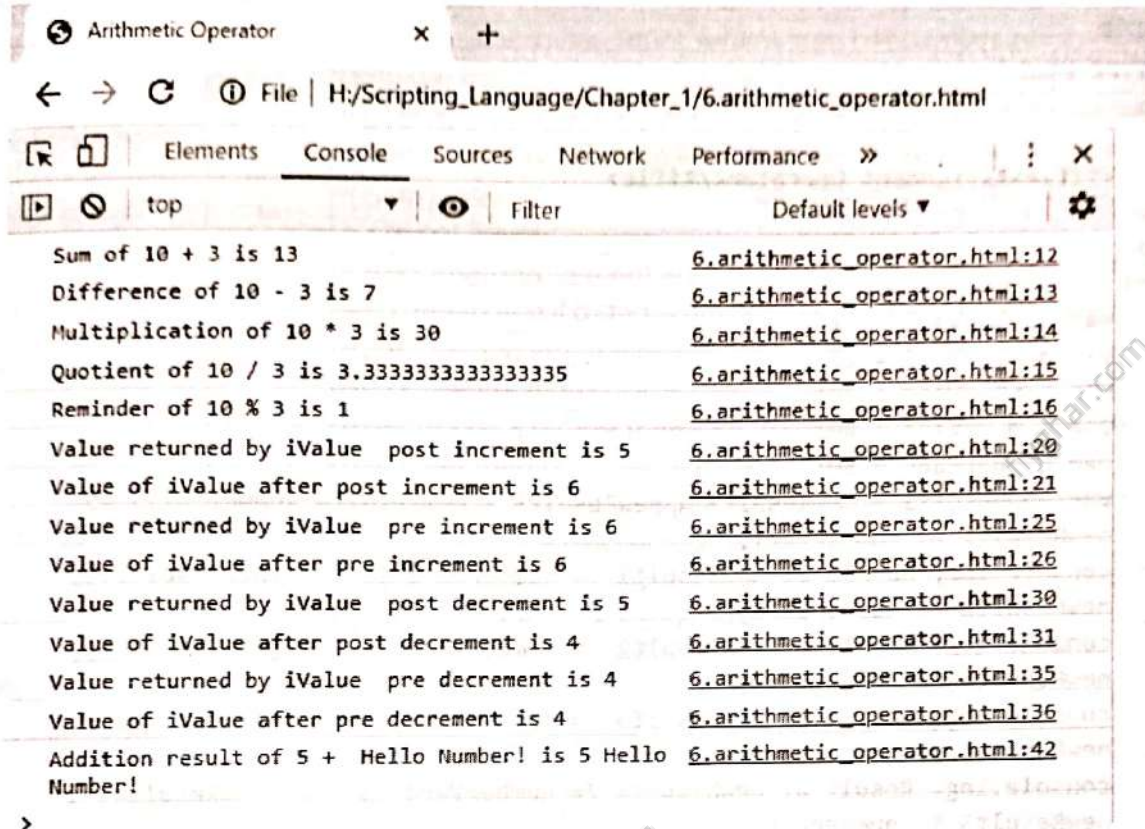
iValue = 5;
iValue1 = iValue--;
console.log('Value returned by iValue post decrement is ' + iValue1);
console.log('Value of iValue after post decrement is ' + iValue);

iValue = 5;
iValue1 = --iValue;
console.log('Value returned by iValue pre decrement is ' + iValue1);
console.log('Value of iValue after pre decrement is ' + iValue);

// Addition (+) operator performs different operation on operands with strings values
var firstNumber = 5;
var firstString = " Hello Number!";
console.log('Addition result of ' + firstNumber + ' + ' + firstString + ' is ' +
(firstNumber + firstString));
</script>
</body>
</html>

```


Output



Assignment Operators

In an assignment operation the value of a variable is computed from the expression that lies to the right of an assignment operator. That value is assigned to the variable or property that is on the left side of the operator.

Operator	Description
=	Assign right operand value to the left operand. For example, $a = 15$; or $a = b$;
+=	Sum of two right operand values is assigned to the left operand. $a += 5$; is equivalent to $a = a + 5$;
-=	-=Subtract right operand value from the left operand and assign the result to the left operand. $a -= 12$; is equivalent to $a = a - 12$;
*=	*=Multiply right and left operand and assign the result to the left operand. $a *= 25$; is equivalent to $a = a * 25$;
/=	/=Divideleft operand value with the rightoperand value and assign the result to the left operand. $a /= 31$; is equivalent to $a = a / 31$;
%=	%=Get the modulus of left operand, divide it by right operand and assign the result to the left operand. $a \% = 102$; is equivalent to $a = a \% 102$;
&=	&=Bitwise AND,AND operator compare two bits values return a results of 1, If both bits are 1. Otherwise return 0. $a \&= y$ $a = a \& y$ $= 2 \& 5$ $= 0000\ 0010 \& 0000\ 0101$ $= 0000\ 0000 = 0$
=	=Bitwise OR,OR operator compare two bits values return a results of 1, If both bits are 1. Otherwise return 0. $a = y$ $a = a y$ $= 2 5$ $= 0000\ 0010 0000\ 0101$ $= 0000\ 0111 = 7$

Example for assignment operator

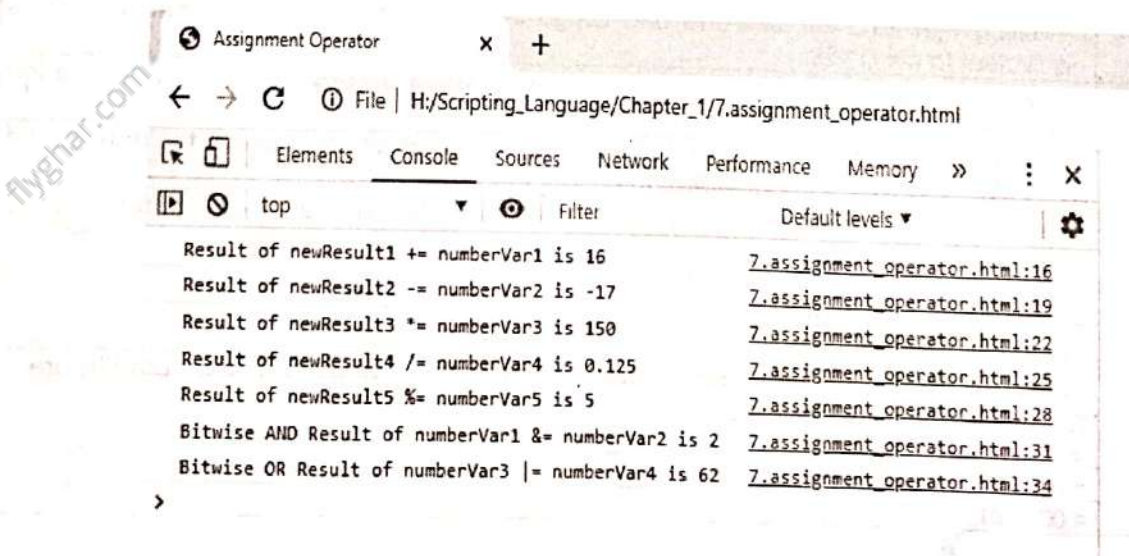
Example

```

<!DOCTYPE html>
<html>
<head>
    <title>Assignment Operator</title>
</head>
<body>
<script type="text/javascript">
    var numberVar1 = 11;
    var numberVar2 = 22;
    var numberVar3 = 30;
    var numberVar4 = 40;
    var numberVar5 = 50;
    var newResult1 = newResult2 = newResult3 = newResult4 = newResult5 = 5;
    newResult1 += numberVar1;
    console.log('Result of newResult1 += numberVar1 is ' + (newResult1));
    newResult2 -= numberVar2;
    console.log('Result of newResult2 -= numberVar2 is ' + (newResult2));
    newResult3 *= numberVar3;
    console.log('Result of newResult3 *= numberVar3 is ' + (newResult3));
    newResult4 /= numberVar4;
    console.log('Result of newResult4 /= numberVar4 is ' + (newResult4));
    newResult5 %= numberVar5;
    console.log('Result of newResult5 %= numberVar5 is ' + (newResult5));
    numberVar1 &= numberVar2;
    console.log('Bitwise AND Result of numberVar1 &= numberVar2 is ' +
    (numberVar1));
    numberVar3 |= numberVar4;
    console.log('Bitwise OR Result of numberVar3 |= numberVar4 is ' +
    (numberVar3));
</script>
</body>
</html>

```

Output



Comparison Operators

Name of Operator	Symbol	Description
Equal	==	If both operands are equal, returns true. a==b return true if a and b both are equal
Identical equal	===	If both operands are equal and/or same data type, returns true. a===b return true if a and b both are equal and have same data type.
Not equal	!=	If both operands are not equal, returns true. a != b return true if a and b are not equal.
Identical not equal	!==	If both operands are not equal and/or same data type, returns true. a !== b return true if a and b are not equal and same data type.
Greater than	>	If left operand larger than right operand, return true. a > b return true of a is greater than b.
Less then	<	If left operand smaller than right operand, return true. a < b return true of a is less than b.
Greater than, equal	>=	If left operand larger or equal than right operand, return true. a < b return true of a is greater than equal to b.
Less than, equal	<=	If left operand smaller or equal than right operand, return true. a < b return true of a is less than equal to b.

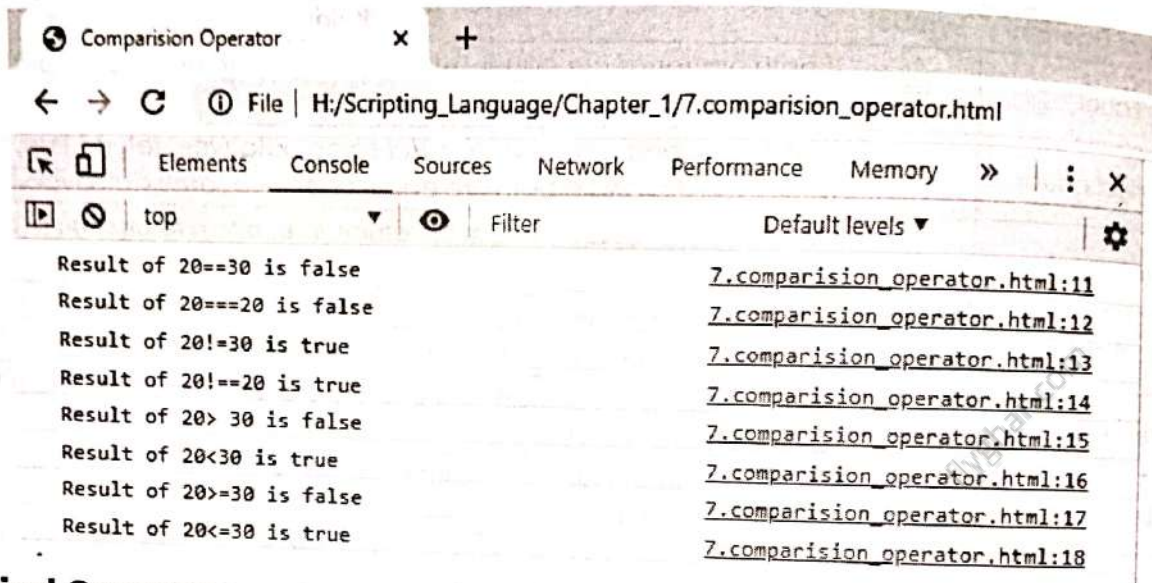
Example

```

<!DOCTYPE html>
<html>
<head><title>Comparision Operator</title></head>
<body>
<script type="text/javascript">
    var firstNumber = 20;
    var secondNumber = 30;
    var firstString = '20';
    console.log('Result of ' + firstNumber + '==' + secondNumber + ' is ' +
(firstNumber == secondNumber) );
    console.log('Result of ' + firstNumber + '===' + firstString + ' is ' +
(firstNumber == secondNumber) );
    console.log('Result of ' + firstNumber + '!=' + secondNumber + ' is ' +
(firstNumber != secondNumber) );
    console.log('Result of ' + firstNumber + '!== ' + firstString + ' is ' +
(firstNumber !== secondNumber) );
    console.log('Result of ' + firstNumber + '>' + secondNumber + ' is ' +
(firstNumber > secondNumber) );
    console.log('Result of ' + firstNumber + '<' + secondNumber + ' is ' +
(firstNumber < secondNumber) );
    console.log('Result of ' + firstNumber + '>=' + secondNumber + ' is ' +
(firstNumber >= secondNumber) );
    console.log('Result of ' + firstNumber + '<=' + secondNumber + ' is ' +
(firstNumber <= secondNumber) );
</script>
</body>
</html>

```

Output



Logical Operators

Logical or Boolean operators compare Boolean expressions and then return true or false. The && and || ('and' and 'or') operators take two operands. The ! ('not') operator takes a single operand. The logical operators are used to compare two or more conditions

Operator	Symbol	Description
Logical AND	&&	If first operand evaluate and return a true only that evaluate the second operand otherwise skips. Return true if both are must be true, otherwise return false.
Logical OR		Evaluate both operands, Return true if either both or any one operand true. Return false if both are false.
Logical NOT	!	Return the inverse of the given value result true become false, and false become true.

```
<!DOCTYPE html>
<html>
<head>
  <title>Logical Operator Example</title>
</head>
<body>
  <script type="text/javascript">
    var firstNumber = 10;
    var secondNumber = 20;
    var thirdNumber = 30;
    console.log('Result of logical operator AND: (' + firstNumber + ') < (' + secondNumber
    + ') && (' + firstNumber + ') < (' + thirdNumber + ') is : ' + ((firstNumber <
    secondNumber) && (firstNumber < thirdNumber)));
    console.log('Result of logical operator OR: (' + firstNumber + ') > (' + secondNumber
    + ') || (' + firstNumber + ') > (' + thirdNumber + ') is : ' + ((firstNumber >
    secondNumber) || (firstNumber > thirdNumber)));
    console.log('Result of logical operator NOT: !(((' + firstNumber + ') > (' +
    secondNumber + ') || (' + firstNumber + ') > (' + thirdNumber + ')) is : ' +
    !((firstNumber > secondNumber) || (firstNumber > thirdNumber)));
  </script>
</body>
</html>
```


Conditional Operator (Ternary Operator)

A Ternary Operator is an alternative to if, else conditions in JavaScript. Unlike other operators that usually take two expressions, a Ternary operator takes three expressions. The expressions are in the form of a condition (first value), followed by two more expressions, which returns the result.

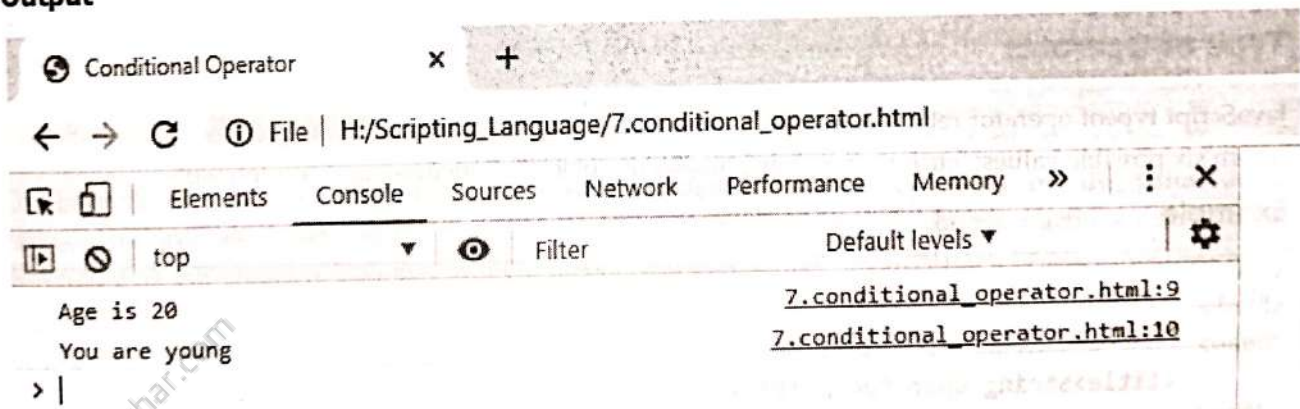
Condition ? first_expression : second_expression

It has two operators, the "?" (Question mark) and ":" (Colon), which separates the condition from the result. If the condition is true, it will evaluate the first expression and it becomes the result. Else, it will look for the result evaluating the second expression.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Conditional Operator</title>
</head>
<body>
<script type="text/javascript">
  var age = 20;
  console.log('Age is ' + age);
  console.log('You are ' + (age < 16 ? 'children' : 'young'));
</script>
</body>
</html>
```

Output



String Operators

In addition to the comparison operators, which may be used on string values, the concatenation operator (+) concatenates two string values together, returning another string that is the union of the two operand strings. For example,

"Ram " + "Dhami" returns the string "Ram Dhami"

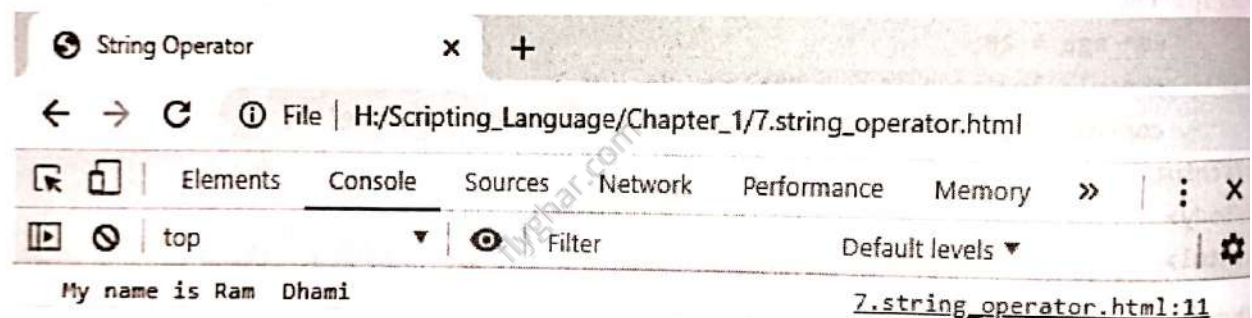
The shorthand assignment operator += can also be used to concatenate strings. For example, if the variable myString is a string that has the value "Ram ", then the expression myString += "Dhami" evaluates to "Ram Dhami" and assigns this value to myString.

Example

```

<!DOCTYPE html>
<html>
<head>
  <title>String Operator</title>
</head>
<body>
<script type="text/javascript">
  var firstName = "Ram ";
  var secondName = "Dhami";
  var name = firstName + ' ' + secondName;
  console.log('My name is ' + name);
</script>
</body>
</html>

```

Output**Type of Operator**

JavaScript type of operator return valid data type identifiers as a string of given expression. type of operator return six possible values: "string", "number", "boolean", "object", "function", and "undefined".

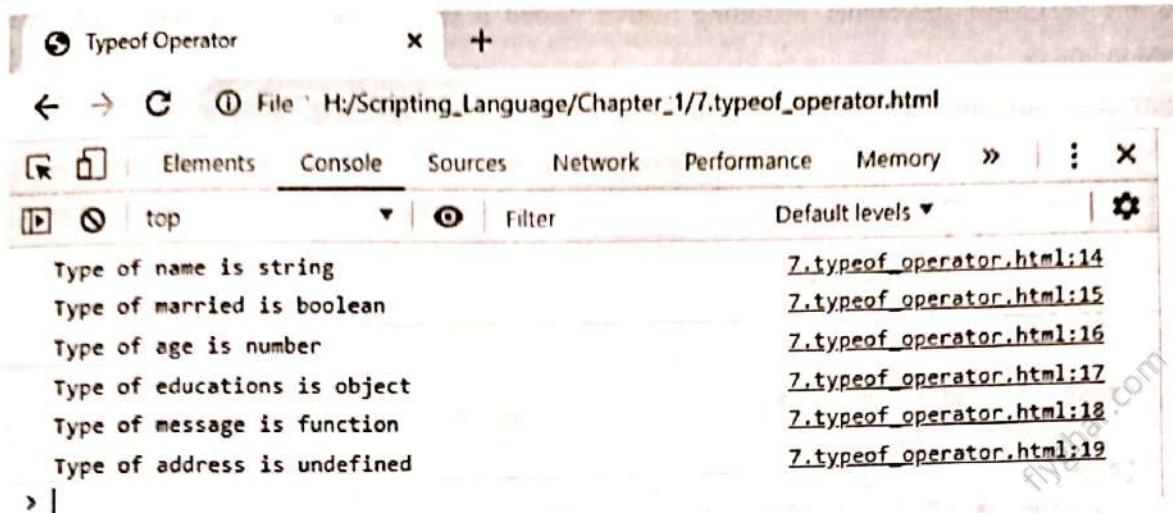
Example

```

<!DOCTYPE html>
<html>
<head>
  <title>String Operator</title>
</head>
<body>
<script type="text/javascript">
  var firstName = "Ram ";
  var secondName = "Dhami";
  var name = firstName + ' ' + secondName;
  console.log('My name is ' + name);
</script>
</body>
</html>

```


Output



CONTROL STRUCTURE

JavaScript control structure, as the term implies, refers to the flow of execution of the program. There are two possible control structures: linear and non-linear. The linear execution (also called sequential execution) refers to the execution of the statements in the order they are listed. In comparison, the non-linear execution refers to the execution of statements regardless of the order they are listed. JavaScript has number of statements that control flow of the program.

- Conditionals (if-else, switch) that perform different actions depending on the value of an expression,
- Loops (while, do-while, for, for-in, for-of), that execute other statements repetitively,
- Jumps (break, continue, labeled statement) that cause a jump to another part of the program.

Conditionals are control structures that perform different actions depending on the value of an expression.

Conditional Statements

Conditionals are control structures that perform different actions depending on the value of an expression. Conditional statements are used to control your scripts so that different actions can be taken depending on the situation. You may want to display a special image on your home page during the time of world cup game. This condition would depend on what day it was, and if it was a tournament period, then a special image would be displayed to your visitors. Without proper knowledge of the conditional statement, your scripts will not be as lively or dynamic as they could possibly be.

Conditional statements let you perform certain actions based on a logical condition. You specify a condition to test and the commands to execute if the condition is true. Example of one conditional statement is: if statement.

If statement

If statement is an either/or switch. If a specified condition is true, JavaScript performs certain statements. If the condition is false, JavaScript might perform other statements. If statement looks as follows:

```
if (condition) {
    statements1
}
[else {
    statements2}]
```

The condition can be any JavaScript expression that evaluates to true or false. The conditional statements can be any JavaScript statements, including further nested if statements. Multiple statements must be enclosed in braces.

You may also compound the statements using else if to have multiple conditions tested in sequence, as follows:

```
if (condition)
    statement_1
[else if (condition_2)
    statement_2]
...
[else if (condition_n_1)
    statement_n_1]
[else
    statement_n]
```

To execute multiple statements, use a block statement ({ ... }) to group those statements. In general, it is a good practice to always use block statements, especially in code involving nested if statements:

Switch Statement

A switch statement allows a program to evaluate an expression and attempt to match the expression's value to a case label. If a match is found, the program executes the associated statement. The optional break keyword associated with each case ensures that the program breaks out of the switch statement once the matched statement is executed. If the break is omitted all subsequent statements will also be executed. A switch statement looks as follows:

```
switch (expression) {
    case label_1:
        statements_1
        [break;]
    case label_2:
        statements_2
        [break;]
    ...
    default:
        statements_def
        [break;]
}
```

The program first looks for a case clause with a label matching the value of expression and then transfers control to that clause, executing the associated statements. If no matching label is found, the program looks for the optional default clause, and if found, transfers control to that clause, executing the associated statements. If no default clause is found, the program continues execution at the statement following the end of switch. By convention, the default clause is the last clause, but it does not need to be so.

The optional break statement associated with each case clause ensures that the program breaks out of switch once the matched statement is executed and continues execution at the statement following switch. If break is omitted, the program continues execution at the next statement in the switch statement.

Loop Statements

Loops are control structures that execute other statements repetitively. A loop is a set of commands that executes repeatedly until a specified condition is met. A loop statement checks to see if some condition is true, and if that condition is true, it executes a chunk of code. After the code is executed, the condition is checked again. If it is true, the process starts over again; if it is false, the loop stops and the rest of the code continues along.

JavaScript supports two loop structures: **for** and **while**. In addition, the **break** and **continue** statements are used specifically with loops.

for statement

A **for** loop repeats a loop until a specified condition evaluates to false. The JavaScript **for** loop is similar to the Java **for** loop and the traditional **for** loop in C. A **for** statement looks as follows:

```
for ([initial-expression;] [condition;] [increment-expression]) {  
    statements  
}
```

When **for** loop is encountered, the initial-expression is executed. The statements are executed as long as condition evaluates to true. The increment-expression is performed on each pass through the loop. The sequence of execution is as follows:

1. The initializing expression initial-expression, if any, is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity.
2. The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, **for** loop terminates.
3. The update expression increment-expression executes.
4. The statements execute and control returns to step 2.

While statement

A **while** statement repeats a loop as long as a specified condition evaluates to true. A **while** statement looks as follows:

```
while (condition) {  
    statements  
}
```

If the condition becomes false, the statements within the loop stop executing and control passes to the statement following the loop.

The condition test occurs only when the statements in the loop have been executed and the loop is about to be repeated. That is, the condition test is not continuous, but is performed once at the beginning of the loop and again just following the last statement in statements, each time control passes through the loop.

do...while Statement

The **do...while** statement repeats until a specified condition evaluates to false. A **do...while** statement looks as follows:

```
do  
    statement  
while (condition);
```

Statement executes once before the condition is checked. To execute multiple statements, use a block statement (`{ ... }`) to group those statements. If condition is true, the statement executes again. At the end of every execution, the condition is checked. When the condition is false, execution stops and control passes to the statement following `do...while`.

In the following example, the `do` loop iterates at least once and reiterates until `i` is no longer less than 5.

Example:

```
do {
  i += 1;
  document.write(i);
} while (i < 5);
```

JavaScript uses the `for...in`, `for each...in`, and `with` statements to manipulate objects.

for...in Statement

The `for-in` statement iterates a specified variable over all the properties of an object. For each distinct property, JavaScript executes the specified statements. A `for...in` statement looks as follows:

```
for (variable in object) {
  statements
}
```

Example:

The following function takes as its argument an object and the object's name. It then iterates over all the object's properties and returns a string that lists the property names and their values.

```
function dump_props(obj, obj_name) {
  var result = "";
  for (var i in obj) {
    result += obj_name + "." + i + " = " + obj[i] + "<br>";
  }
  result += "<hr>";
  return result;
}
```

For an object `car` with properties `make` and `model`, result would be:

`car.make = Ford`

`car.model = Mustang`

Arrays

Although it may be tempting to use this as a way to iterate array elements, because the `for...in` statement iterates over user-defined properties in addition to the array elements, if you modify the Array object, such as adding custom properties or methods, the `for...in` statement will return the name of your user-defined properties in addition to the numeric indexes. Thus it is better to use a traditional `for` loop with a numeric index when iterating over arrays.

for each...in Statement

for each ...in is a loop statement introduced in JavaScript 1.6. It is similar to for...in, but iterates over the values of object's properties, not their names.

```
var sum = 0;
var obj = {prop1: 50, prop2: 90, prop3: 60, prop4: 56};
for each (var item in obj) {
    sum += item;
}
print(sum); // prints "256", which is 50+90+60+56
```

for-of loop

The for-of is a new loop introduced in ECMAScript 2015. It iterates over iterable objects (arrays, maps, sets and others). The syntax is as follow:

```
for (variable of object)
    statement;
```

In relation to the for-in loop, the for-of loop is a more convenient and iterates over property values:

```
var elements = ['x', 'y', 'z', 16, 48, 65];
for (var element of elements) {
    console.log(element);
}
```

Jump Statements

Jumps are control structures that cause a jump to another part of the program.

break statement

The break statement terminates the current while or for loop and transfers program control to the statement following the terminated loop. A break statement looks as follows:

```
break
```

Example

The following function has a break statement that terminates the while loop when i is 3 and then returns the value 3 * x.

```
function testBreak(x) {
    var i = 0
    while (i < 6) {
        if (i == 3)
            break
        i++
    }
    return i*x
}
```

continue statement

A continue statement terminates execution of the block of statements in a while or for loop, and continues execution of the loop with the next iteration. A continue statement looks as follows:

```
continue
```

In contrast to the break statement, continue does not terminate the execution of the loop entirely. Instead,

- In a while loop, it jumps back to the condition.
- In a for loop, it jumps to the increment-expression.

Example. The following example shows a while loop that has a continue statement that executes when the value of i is 3. Thus, n takes on the values 1, 3, 7, and 12.

```
i = 0
n = 0
while (i < 5) {
    i++
    if (i == 3)
        continue
    n += i
}
```

labeled statement

The labeled statement provides an identifier for a statement that you can use to refer to it elsewhere in the program.

The syntax is as follow:

```
label:
statement
```

The break and continue statements are the only statements that use labels to specify the statement to which they apply.

Example:

```
var i, j;
outerLoop:
for (i = 0; i < 3; i++) {
    j = 0;
    while(j < 3) {
        if (i === 1 && j === 1) {
            continue outerLoop;
        }
        console.log('i = ' + i + ', j = ' + j);
        j++;
    }
}
```

Using continue with the outerLoop label, two pairs of values have been skipped.

ARRAY & FOR EACH LOOP

An array is an ordered collection of data. Arrays are complex variables that allow us to store more than one value or a group of values under a single variable name. It is a container that holds multiple values. Each element of the array has a fixed position called an index and stores a value. All indexes are numeric, the first element has index 0, second 1, etc.

Arrays in JavaScript are dynamic and increase or decrease as needed. Their size do not have to be predetermined. JavaScript arrays may be sparse - there can be gaps in the indexes.

Another feature of arrays in JavaScript is that they are untyped, elements of an array can store any valid value, including strings, numbers, objects, functions, and even other arrays, thus making it possible to create more complex data structures such as an array of objects or an array of arrays.

In fact JavaScript arrays are specialized form of objects. So indexes are property names that are integers. JavaScript provides predefined Array object that you can use to work with arrays.

To create an empty array simply use empty square brackets:

```
var emptyArray = [];
```

To create array of elements enclose them in square brackets:

```
var ages = [10,25, 56, 23];  
var subjects = ["DBMS", "NM", "Scripting", "OS","Project"];
```

In one array you can enclose values of different types:

```
var mixedArray = [10,25,"DBMS", "NM", "Scripting"];
```

An array can contain objects and other arrays:

```
var complexArray = [{age: 20, name: 'Rajesh'}, ["Kathmandu", "Pokhara"]];
```

Another way of creating arrays is to use the Array constructor.

Creating an empty array:

```
var emptyArray = new Array();
```

To create an array of elements pass more than one numeric value or non-numeric value as constructor arguments:

```
var arrayOne = new Array ('n'); // single element array  
var arrayTwo = new Array(4, 8, 12); // array of numbers  
var arrayThree = new Array(89,1.69, 5.9, true, "apple");
```

If you pass a single numeric value as an argument, it is treated as an array length specification.

```
var newArray = new Array(5);
```

This creates an empty array with the specified length.

Note that calling the constructor Array with a single numeric argument, which is not an integer, causes an error like:

```
var newArray = new Array(9.5); // RangeError: invalid array length
```

Arrays cannot use strings as element indexes (as in an associative array) but must use integers. Setting or accessing via non-integers using bracket notation (or dot notation) will not set or retrieve an element from the array list itself, but will set or access a variable associated with that array's object property collection. The array's object properties and list of array elements are separate, and the array's traversal and mutation operations cannot be applied to these named properties.

Writing and Reading Arrays

You can access an array element using the [] operator following a reference to the array. Inside the brackets, give the index of the element, which should be a non-negative integer. This way you can use for both reading and writing elements.

```
//create empty array
var myArray = [];
// write first element of myArray
myArray[0] = 2;
// write second element of myArray
myArray[1] = 3.14
// write third element of myArray
myArray[2] = "Apple";
```

You can also write values to an array when you create it:

```
var myArray1 = [2, 3.14, "Apple"];
```

To print second index of array

```
console.log(myArray1[2]);
```

To print second element of array

```
console.log(myArray1[1]);
```

Adding an Item to an Array

```
var subjects = ["NM", "Scripting", "OS"];
```

In our subjects variable we had three items, which consisted of the indices from 0 to 2. If we want to add a new item to the array, we can assign a value to the next index.

```
subjects[3] = "Project I";
```

If we add an item and accidentally skip an index, it will create an undefined item in the array.

```
subjects[5] = "SE";
```

forEach in JavaScript

While working with JavaScript, all of us must have surely run into the case where we need to loop through an array and break the running loop if a certain condition is met. There are many methods how we can loop through an array but we are always looking for the most efficient way to do that. A great way to traverse an array has been for loop, and that is exactly what the forEach loop improves upon. You can simply use the array's forEach() method to loop through all the items in an array using JavaScript. The forEach() method

executes a provided function once for each array element. JavaScript `forEach` only be used on the Arrays, Maps, and Sets.

Use Cases of JavaScript `forEach`

The `forEach` loops can be used in a number of situations. If you need to run the same function on every element of the array, a `forEach` call function can be useful. For example, if you've got an index of 500 names and you want to print them all to your console, you can use `forEach`.

Syntax of `forEach`

`array.forEach(function(currentValue, index, arr), thisValue)`

Parameter	Description								
<code>function(currentValue, index, arr)</code>	Required. A function to be run for each element in the array. Function arguments: <table border="1"> <thead> <tr> <th>Argument</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>currentValue</code></td><td>Required. The value of the current element</td></tr> <tr> <td><code>index</code></td><td>Optional. The array index of the current element</td></tr> <tr> <td><code>arr</code></td><td>Optional. The array object the current element belongs to</td></tr> </tbody> </table>	Argument	Description	<code>currentValue</code>	Required. The value of the current element	<code>index</code>	Optional. The array index of the current element	<code>arr</code>	Optional. The array object the current element belongs to
Argument	Description								
<code>currentValue</code>	Required. The value of the current element								
<code>index</code>	Optional. The array index of the current element								
<code>arr</code>	Optional. The array object the current element belongs to								
<code>thisValue</code>	Optional. A value to be passed to the function to be used as its "this" value. If this parameter is empty, the value "undefined" will be passed as its "this" value								

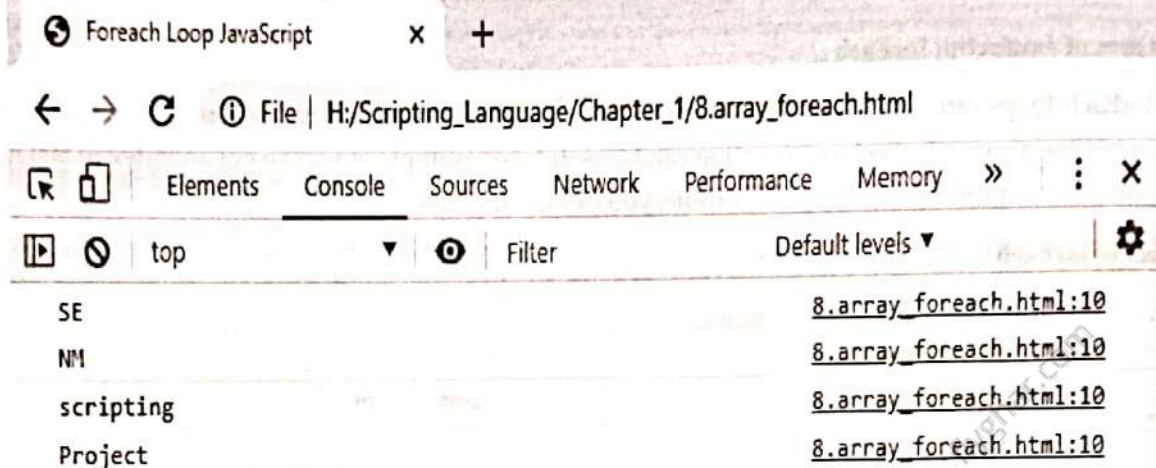
The best use cases for JavaScript `forEach` loops are where you need to do something with every item in an array in JavaScript, not just a few.

The `forEach` calls are part of JavaScript 1.6, so it has browser support across most of the main browsers.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Foreach Loop JavaScript</title>
</head>
<body>
  <script type="text/javascript">
    var mySubjects = ["SE", "NM", "scripting", "Project"];
    mySubjects.forEach(subject => console.log(subject));
  </script>
</body>
</html>
```

Output



Different JavaScript array properties

Property	Description
constructor	Returns the function that created the Array object's prototype
length	Sets or returns the number of elements in an array
prototype	Allows you to add properties and methods to an Array object

Constructor Property

The constructor property returns an array's constructor function. The return value is a reference to the function, not the name of the function.

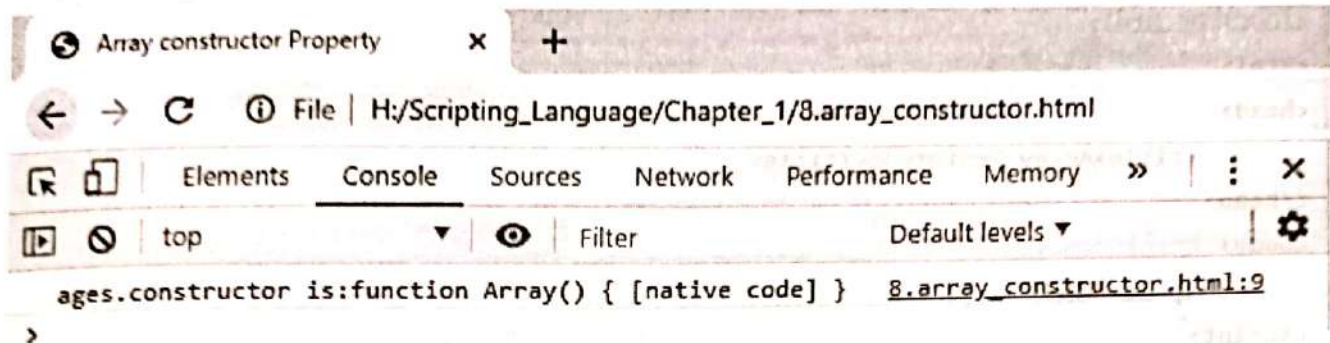
- For JavaScript arrays the constructor property returns function Array() { [native code] }.
- For JavaScript numbers the constructor property returns function Number() { [native code] }.
- For JavaScript strings the constructor property returns function String() { [native code] }.

We can use the constructor property to check if a variable is an array.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Array constructor Property</title>
</head>
<body>
<script type = "text/javascript">
    var ages = new Array( 63,98,45);
    console.log("ages.constructor is:" + ages.constructor);
</script>
</body>
</html>
```


Output



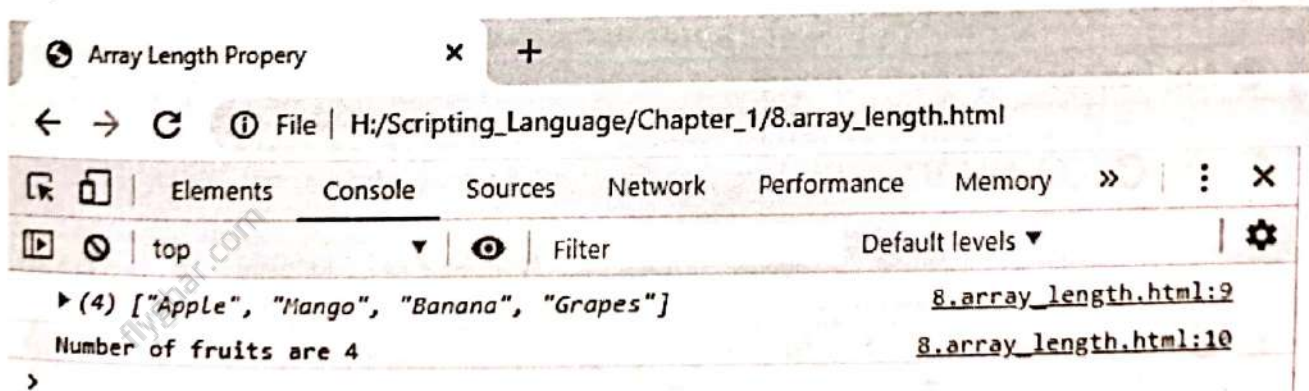
Array length Property

The length property sets or returns the number of elements in an array. Length is a property of arrays in JavaScript that returns or sets the number of elements in a given array.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Array Length Property</title>
</head>
<body>
<script type="text/javascript">
  var fruits = ["Apple", "Mango", "Banana", "Grapes"];
  console.log(fruits);
  console.log('Number of fruits are ' + fruits.length);
</script>
</body>
</html>
```

Output



Array prototype Constructor

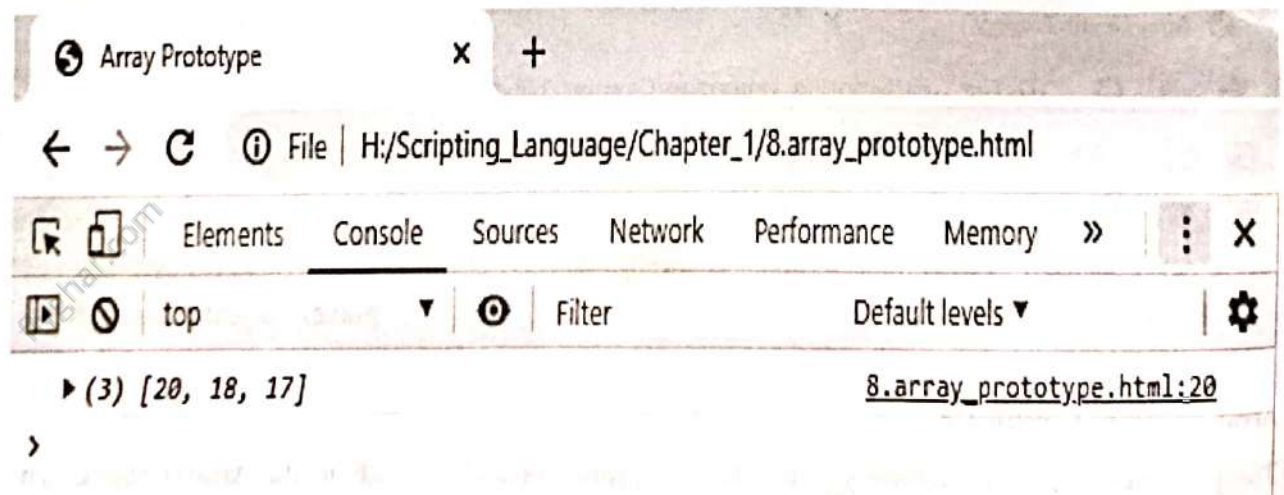
The prototype constructor allows you to add new properties and methods to the Array() object. When constructing a property, ALL arrays will be given the property, and its value, as default. When constructing a method, ALL arrays will have this method available. Prototype is a global object constructor which is available for all JavaScript objects.

Example

```

<!DOCTYPE html>
<html>
<head>
    <title>Array Prototype</title>
</head>
<body>
    <p id="demo"></p>
<script>
    Array.prototype.stringLength = function() {
        var i;
        for (i = 0; i < this.length; i++) {
            this[i] = this[i].length;
        }
    };
    function lengthFunction() {
        var subjects = ["Software Engineering", "Scripting Language", "Numerical
Methods"];
        subjects.stringLength();
        console.log(subjects);
    }
    lengthFunction();
</script>
</body>
</html>

```

Output**Array Methods**

The following table lists the standard methods of the Array object.

Method	Description
concat()	Merge two or more arrays, and returns a new array.
copyWithin()	Copies part of an array to another location in the same array and returns it.
entries()	Returns a key/value pair Array Iteration Object.
every()	Checks if every element in an array pass a test in a testing function.
fill()	Fill the elements in an array with a static value.
filter()	Creates a new array with all elements that pass the test in a testing function.
find()	Returns the value of the first element in an array that pass the test in a testing function.
findIndex()	Returns the index of the first element in an array that pass the test in a testing function.
forEach()	Calls a function once for each array element.
from()	Creates an array from an object.
includes()	Determines whether an array includes a certain element.
indexOf()	Search the array for an element and returns its first index.
isArray()	Determines whether the passed value is an array.
join()	Joins all elements of an array into a string.
keys()	Returns an Array Iteration Object, containing the keys of the original array.
lastIndexOf()	Search the array for an element, starting at the end, and returns its last index.
map()	Creates a new array with the results of calling a function for each array element.
pop()	Removes the last element from an array, and returns that element.
push()	Adds one or more elements to the end of an array, and returns the array's new length.
reduce()	Reduce the values of an array to a single value (from left-to-right).
reduceRight()	Reduce the values of an array to a single value (from right-to-left).
reverse()	Reverses the order of the elements in an array.
shift()	Removes the first element from an array, and returns that element.
slice()	Selects a part of an array, and returns the new array.
some()	Checks if any of the elements in an array passes the test in a testing function.
sort()	Sorts the elements of an array.
splice()	Adds/Removes elements from an array.
toString()	Converts an array to a string, and returns the result.
unshift()	Adds new elements to the beginning of an array, and returns the array's new length.
values()	Returns an Array Iteration Object, containing the values of the original array.

push()

Adding Element at the end of an Array. As array in JavaScript are mutable object, we can easily add or remove elements from the Array. And it dynamically changes as we modify the elements from the array. This method is used add elements at the end of an array.

Syntax: `Array.push(item1, item2 ...)`

Parameters: Items to be added to an array.

unshift()

Adding elements at the front of an Array. This method is used to add elements at the beginning of an array.

`Array.unshift(item1, item2 ...)`

Parameters: Items to be added to the array

pop()

Removing elements from the end of an array. It is used to remove array elements from the end of an array.

`Array.pop()`

Parameters: It takes no parameter

shift()

Removing elements at the beginning of an array. It is used to remove array at the beginning of an array.

`Array.shift()`

it takes no parameter

splice()

Insertion and Removal in between an Array. Splice is very useful method as it can remove and add elements from the particular location.

`Array.splice(start, deleteCount, item 1, item 2....)`

Parameters:

Start : Location at which to perform operation

deleteCount: Number of element to be deleted,

if no element is to be deleted pass 0.

Item1, item2- this is an optional parameter .

These are the elements to be inserted from location start

map()

The `map()` is used to modify each element of the array according to the callback function. `Array.prototype.map()` calls the callback function once for each element in the array in order. The point to note is that callback function is called on indexes of elements who has assigned value including undefined.

`arr.map(callback[, thisArg])`

Parameters:

callback : Callback function to be called for each element of the array arr

- **currentValue**: The value of the elements being processed currently
- **index(optional)**: Index of the currentValue element in the array starting from 0
- **array(optional)**: The complete array on which `Array.prototype.map()` is called

thisArg: Context to be passed as this when used while executing the callback function. If no context is passed, undefined will be used as default value.

concat()

```
NewArray = array.concat(value1[, value2[, ...[, valueN]]])
```

Parameters:

valueN : New array or value to be concatenated to array.

The `Array.prototype.concat()` is used to concatenate an array with another array/value. It does not change any existing array, instead, it returns a modified array. It accepts both new arrays and values as an argument which it concatenates with the array calling the function and returns the resultant array.

indexOf()

The `indexOf()` function is used to find the index of the first occurrence of the search element provided as the argument to the function.

The syntax of the function is as follows:

```
arr.indexOf(searchElement[,index])
```

Parameters:

The first argument to this function is the `searchElement` which is the value to be searched in the array. Second argument to this function is the optional index argument which defines the starting index in the array from where the element is to be searched. If this argument is not provided then index 0 is taken as the starting index to begin the search as it is the default value.

Return value:

This function returns the index of the first occurrence of the `searchElement`. If the element cannot be found in the array, then this function returns -1

Starting from 1st index:

```
var array = [2, 9, 9];  
document.write(array.indexOf(2));
```

Starting form 2nd Index;

```
var array = [2, 98, 12, 45];  
document.write(array.indexOf(98,2));
```

join()

The `join()` function is used to join the elements of the array together into a string.

Syntax:

```
Array.join([separator])
```

Parameters: (separator)

A string to separate each elements of the array. If leave it by default array element separate by comma (,).

Return Value:

This function returns a string created by joining all the elements of the array using the separator. If the separator is not provided then the array elements are joined using comma (,) as it is the default separator for this function. If an empty string is provided as the separator then the elements are joined directly without any character in between them. If array is empty then this function returns an empty string.

Join using default separator:

```
var a = [ 1, 2, 3, 4, 5, 6 ];
document.write(a.join());
```

Join using custom separator, for now (+):

```
var a = [ 1, 2, 3, 4, 5, 6 ];
document.write(a.join('+'));
```

toString()

arr.toString() returns the string representation of the array elements. The syntax of this function is as follows:

arr.toString()

Parameters:

This function does not take any arguments.

Return value:

The function returns the string representation of the array elements. If the array is empty, then it returns an empty string.

```
var arr = [2, 5, 8, 1, 4]
print(arr.toString());
```

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Array methods</title>
</head>
<body>
<!DOCTYPE html>
<html>
<head>
    <title>Array Example</title>
</head>
<body>
<script type="text/javascript">

    //initilize subject list
    var subjects = ['Scripting','NM','SE','OS','DBMS']
    console.log('Initial subject lists are:')
    console.log(subjects);

    //add new data into last index of array
    subjects.push('Project I');
    console.log('Subject lists after adding data into last index:')
    console.log(subjects);
```



```
//add new data into first index of array
subjects.unshift('Project Work I');
console.log('Subject lists after adding data into first index:')

console.log(subjects);

//remove data from last index of array
subjects.pop();
console.log('Subject lists after removing data from last index:')
console.log(subjects);

//remove data from first index of array
subjects.shift();
console.log('Subject lists after removing data from first index:');
console.log(subjects);

//add and remove from any index
// subjects.splice(startindex,remove item)
subjects.splice(2,1)
console.log('Subject lists after removing one data from 2nd index:')
console.log(subjects);

// add into index
subjects.splice(2,0,"Cryptography")
console.log('Subject lists after adding one data into 2nd index:')
console.log(subjects);

// remove add into index
subjects.splice(0,1,"Web Technology","Project Work II")
console.log('Subject lists after removing one data from 0 index and add two
data into same index:')
console.log(subjects);

//array concat
var boys = ['Kumar','Ram','Thakur'];
var girls = ['Gita','Rita','Sita'];
console.log('Initial array of boys');
console.log(boys);
console.log('Initial array of girls');
console.log(girls);

var students = boys.concat(girls);
console.log('Array of students after concatenation both array');
console.log(students);

students.sort();
console.log('Sorting list of students');
console.log(students);

students.reverse();
console.log('Sorting list of students into descending order');
console.log(students);
</script>
</body>
</html>
</body>
</html>
```

Output

Initial subject lists are:

- ▶ (5) ["Scripting", "NM", "SE", "OS", "DBMS"]

Subject lists after adding data into last index:

- ▶ (6) ["Scripting", "NM", "SE", "OS", "DBMS", "Project I"]

Subject lists after adding data into first index:

- ▶ (7) ["Project Work I", "Scripting", "NM", "SE", "OS", "DBMS", "Project I"]

Subject lists after removing data from last index:

- ▶ (6) ["Project Work I", "Scripting", "NM", "SE", "OS", "DBMS"]

Subject lists after removing data from first index:

- ▶ (5) ["Scripting", "NM", "SE", "OS", "DBMS"]

Subject lists after removing one data from 2nd index:

- ▶ (4) ["Scripting", "NM", "OS", "DBMS"]

Subject lists after adding one data into 2nd index:

- ▶ (5) ["Scripting", "NM", "Cryptography", "OS", "DBMS"]

Subject lists after removing one data from 0 index and add two data into same index:

- ▶ (6) ["Web Technology", "Project Work II", "NM", "Cryptography", "OS", "DBMS"]

Initial array of boys

- ▶ (3) ["Kumar", "Ram", "Thakur"]

Initial array of girls

- ▶ (3) ["Gita", "Rita", "Sita"]

Array of students after concatenation both array

- ▶ (6) ["Kumar", "Ram", "Thakur", "Gita", "Rita", "Sita"]

Sorting list of students

- ▶ (6) ["Gita", "Kumar", "Ram", "Rita", "Sita", "Thakur"]

Sorting list of students into descending order

- ▶ (6) ["Thakur", "Sita", "Rita", "Ram", "Kumar", "Gita"]

8.array_methods.html:18
8.array_methods.html:19
8.array_methods.html:23
8.array_methods.html:24
8.array_methods.html:28
8.array_methods.html:30
8.array_methods.html:34
8.array_methods.html:35
8.array_methods.html:39
8.array_methods.html:40
8.array_methods.html:45
8.array_methods.html:46
8.array_methods.html:50
8.array_methods.html:51
8.array_methods.html:55
8.array_methods.html:56
8.array_methods.html:61
8.array_methods.html:62
8.array_methods.html:63
8.array_methods.html:64
8.array_methods.html:68
8.array_methods.html:69
8.array_methods.html:72
8.array_methods.html:73
8.array_methods.html:76
8.array_methods.html:77

Multidimensional Arrays

JavaScript does not have true multidimensional arrays. But you can create the equivalent as array of arrays access the array element, use the appropriate number of times the [] operator and enter the appropriate indexes.

```
var gamesWithPlayer = [
  ['Football', 11],
  ['Cricket', 11],
  ['Basketball', 5],
  ['Hockey', 11],
  ['Volleyball', 6]
];
console.log(gameWithPlayer.join('\n'));
```


Adding an element to the JavaScript multidimensional array

You can use the array methods to manipulate a multidimensional array as normal. For example, to add a new element to a multidimensional array, you use the `push()` method as in the following example.

```
gameWithPlayer.push(['Badminton', 2]);  
console.log(gameWithPlayer.join('\n'));
```

Removing an element from the JavaScript multidimensional array

To remove an element from an array, you use the `pop()` method

```
gameWithPlayer.pop();  
console.log(gameWithPlayer.join('\n'));
```

DEFINING AND INVOKING FUNCTIONS

A function is a block of code that performs an action or returns a value. Functions are custom code defined by programmers that are reusable, and can therefore make your programs more modular and efficient.

The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can call that function. Like other programming languages, JavaScript also supports the use of functions. You must already have seen some commonly used functions in JavaScript like `alert()`, this is a built-in function in JavaScript. But JavaScript allows us to create user-defined functions also.

How to Create a Function in JavaScript

- Use the keyword `function` followed by the name of the function.
- After the function name, open and close parentheses.
- After parenthesis, open and close curly braces.
- Within curly braces, write your lines of code.

Syntax to define function

```
//defining a function  
function <function-name>()  
{  
    // code to be executed  
};
```

```
function welcomeMessage() {  
    alert("Hello Guest, Welcom to our website");  
}
```

Function Parameters

Parameters are additional information passed to a function. For example, in the above example, the task of the function `welcomeMessage` is to display message to user. By default user is guest, if you want to change guest name accordingly then the parameters are passed to the function within parentheses after the function name and separated by commas. A function in JavaScript can have any number of parameters and also at the same time a function in JavaScript can not have a single parameter.

Return Statement

There are some situations when we want to return some values from a function after performing some operations. In such cases, we can make use of the return statement in JavaScript. This is an optional statement and most of the times the last statement in a JavaScript function. The most basic syntax of using the return statement is:

return value;

The return statement begins with the keyword return separated by the value which we want to return from it. We can use an expression also instead of directly returning the value.

Calling Functions

After defining a function, the next step is to call them to make use of the function. We can call a function by using the function name separated by the value of parameters enclosed between parenthesis and a semicolon at the end.

Below syntax shows how to call functions in JavaScript:

<function-name>(Value1, Value2, ..);

Example: welcomeMessage();

Example

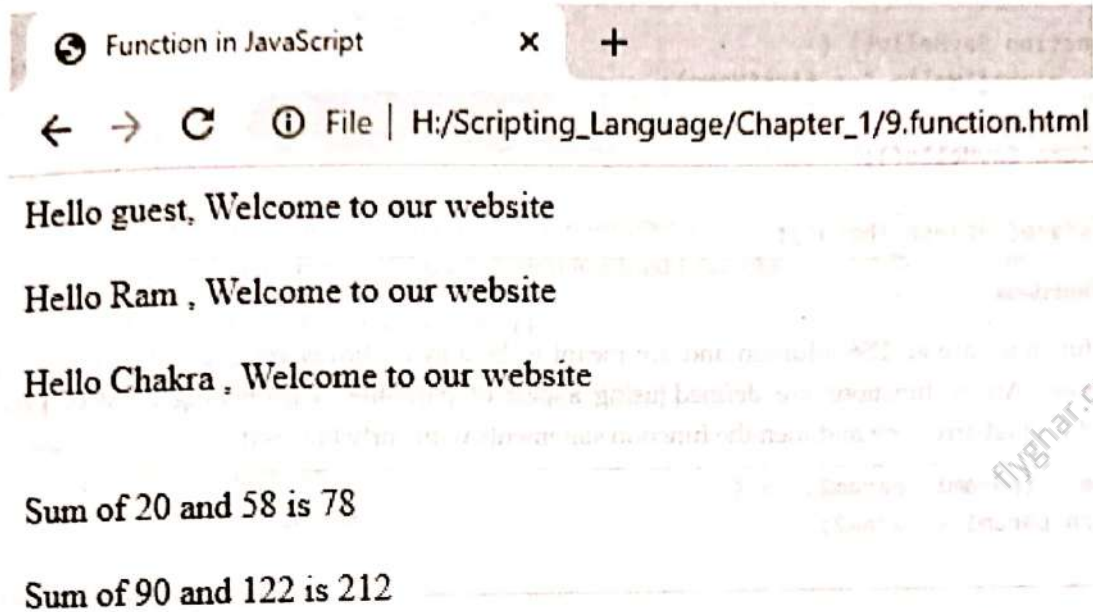
```
<!DOCTYPE html>
<html>
<head>
    <title>Function in JavaScript</title>
</head>
<body>
<script type="text/javascript">

    //default function without parameter and no return type
    function welcomeMessage(){
        document.write('Hello guest, Welcome to our website<br/><br/>');
    }
    welcomeMessage();

    // function with parameter and no return type
    function welcomeMessageWithParameter(guestName){
        document.write('Hello ' + guestName + ' , Welcome to our
website<br/><br/>');
    }
    welcomeMessageWithParameter('Ram');
    welcomeMessageWithParameter('Chakra');

    // function with parameter and return type
    function findSum(first,second){
        return (first+second);
    }

    document.write('<br/>Sum of 20 and 58 is ' + findSum(20,58));
    document.write('<br/><br/>Sum of 90 and 122 is ' + findSum(90,122));
</script>
</body>
</html>
```


Output**The Arguments Object**

All the functions in JavaScript can use arguments object by default. An arguments object includes value of each parameter. The arguments object is an array like object. You can access its values using index similar to array. However, it does not support array methods. An arguments object is still valid even if function does not include any parameters.

```
function welcomeMessage(firstName, lastName) {
    alert("Hello " + arguments[0] + " " + arguments[1]);
}
welcomeMessage("Rajesh", "Thapa");
welcomeMessage(105, 658);
```

we can remove two parameters and use arguments object.

```
function ShowMessage() {
    alert("Hello " + arguments[0] + " " + arguments[1]);
}
ShowMessage("Ram", "Thapa"); // display Hello Ram Thapa
```

An arguments object can be iterated using for loop.

```
function ShowMessage() {
    for(var i = 0; i < arguments.length; i++){
        alert(arguments[i]);
    }
}
ShowMessage("Ram", "Thapa");
```

Nested Functions

In JavaScript, a function can have one or more inner functions. These nested functions are in the scope of outer function. Inner function can access variables and parameters of outer function. However, outer function cannot access variables defined inside inner functions.

```
function ShowMessage(firstName)
{
    function SayHello() {
        alert("Hello " + firstName);
    }
    return SayHello();
}
ShowMessage("Rajesh Thapa");
```

Arrow functions

Arrow functions are an ES6 addition and are meant to be a syntactically compact alternative to function expressions. Arrow functions are defined using a pair of parentheses containing a list of parameters, followed by a fat arrow => and then the function statements with curly braces {}.

```
let sum = (param1, param2) => {
    return param1 + param2;
};
```

JavaScript Object

A JavaScript object is a collection of properties where each property has a name and a value, similar to Hash, Map, or Dictionary in other languages. The name of a property can be any string, including an empty string. The value can be any value, such as a string, Boolean, number, and null, but it cannot be undefined. The object's properties can be defined even after we start using the object.

Creating objects with object literals

The easiest way to create a new object is with object literal notation which is bracketed by a pair of curly braces: {}. Properties and their values can be added as we go.

```
var objectname = {};
```

Accessing properties

Property values can be retrieved in one of two ways; dot notation and bracket notation. Below are examples of each:

```
Objectname.property;
```

```
// creates an empty object
var rectangle = {};
//assign properties
rectangle.width = 20;
rectangle.height = 10;
//access properties
alert(rectangle.width);
alert(rectangle.height);
```

As an alternative you can immediately assign properties and their values in the literal notation.

```
var rectangle = { width: 20, height: 10 };
alert(rectangle.width);
alert(rectangle.height);
```


JavaScript objects are mutable, meaning you can modify their values.

```
var rectangle = { width: 20, height: 10 };
// => modify value
rectangle.width = 30;
alert(rectangle.width);
alert(rectangle.height);
```

Property values are not limited to primitive types, like number or string; you can also add properties that are other objects, including functions. When a function is added to an object it is called a method.

```
var rectangle = { width: 20, height: 10 };
// add new object as property
rectangle.color = { red: 0, green: 255, blue: 128 };
// add new method
rectangle.getArea = function() {
    return this.width * this.height;
};
alert(rectangle.color.red);
alert(rectangle.color.green);
alert(rectangle.color.blue);
alert(rectangle.getArea());
```

Example

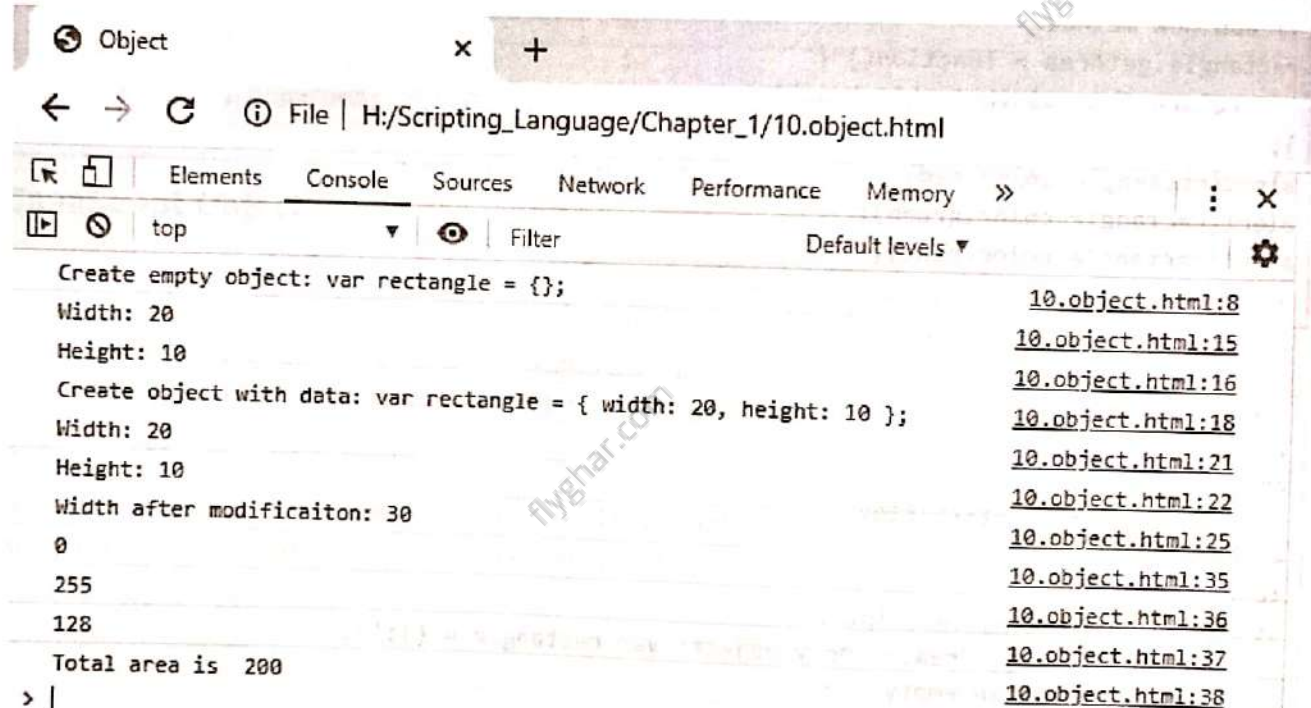
```
<!DOCTYPE html>
<html>
<head>
    <title>Object</title>
</head>
<body>
<script type="text/javascript">
    console.log('Create empty object: var rectangle = {};');
    // creates an empty object
    var rectangle = {};
    //assign properties
    rectangle.width = 20;
    rectangle.height = 10;
    //access properties
    console.log('Width: ' + rectangle.width);
    console.log('Height: ' + rectangle.height);

    console.log('Create object with data: var rectangle = { width: 20, height: 10
};')

    //creating object with data
    var rectangle = { width: 20, height: 10 };
    console.log('Width: ' + rectangle.width);
    console.log('Height: ' + rectangle.height);
    // modifying
    rectangle.width= 30;
    console.log('Width after modificaiton: ' + rectangle.width);
    //create object with other object and method
    var rectangle = { width: 20, height: 10 };
```

```
// add new object as property
rectangle.color = { red: 0, green: 255, blue: 128 };
// add new method
rectangle.getArea = function() {
    return this.width * this.height;
};
console.log(rectangle.color.red);
console.log(rectangle.color.green);
console.log(rectangle.color.blue);
console.log('Total area is ' + rectangle.getArea());
</script>
</body>
</html>
```

Output



Creating objects with Object () constructor

Another way to create objects in JavaScript involves using the "Object" constructor. The Object constructor creates an object wrapper for the given value. This, used in conjunction with the "new" keyword allows us to initialize new objects.

```
const student = new Object();
student.name = 'Rajesh Shah';
student.address = 'Kathmandu';
student.birth_year = 2000;

student.displayInfo = function(){
    console.log(`${student.name} was born on ${student.birth_year} at
    ${student.address}`);
}
student.displayInfo();
```


The two methods mentioned above are not well suited to programs that require the creation of multiple objects of the same kind, as it would involve repeatedly writing the above lines of code for each such object. To deal with this problem, we can make use of two other methods of object creation in JavaScript that reduces this burden significantly

Creating object with function constructors

Constructors in JavaScript, like in most other OOP languages, provides a template for creation of objects. In other words, it defines a set of properties and methods that would be common to all objects initialized using the constructor.

```
function Student(name, address, birthyear) {  
    this.name = name;  
    this.address = address;  
    this.birthyear = birthyear;  
}  
  
let student1 = new Student('Rajesh', 'Kathmandu', 2000);  
let student2 = new Student('Ramit', 'Pokhara', 1998)  
console.log(student1.name);  
console.log(student2.name);
```

BUILT-IN OBJECT JAVASCRIPT

All programming languages have built-in objects that create the essential functionality of the language. Built-in objects are the foundation of the language in which you write custom code that powers custom functionality based on your imagination. JavaScript has many built-in objects that define it as a language.

Some built-in objects

- Boolean & Number Objects
- String
- Date
- Array
- Math
- RegExp Object

Boolean and Number Objects

The JavaScript Boolean Object is a reference to Boolean Values, to create a boolean object using the Boolean Constructor and then pass either true or false as values. JavaScript provides the Boolean and Number objects as object wrappers for boolean true/false values and numbers, respectively. These wrappers define methods and properties useful in manipulating boolean values and numbers.

When a JavaScript program requires a boolean value, JavaScript automatically creates a Boolean object to store the value. JavaScript programmers can create Boolean objects explicitly with the statement.

```
var b = new Boolean( booleanValue );
```

Boolean-object methods

The boolean Value specifies whether the Boolean object should contain true or false. If boolean Value is false, 0, null, Number.NaN or an empty string (""), or if no argument is supplied, the new Boolean object contains false. Otherwise, the new Boolean object contains true.

toString() :

The boolean.toString() is an inbuilt function in JavaScript which is used to return a string either "true" or "false" depending upon the value of the specified boolean object.

boolean.toString()

Parameter: This function does not accept any single parameter.

Return Values: It returns a string either "true" or "false" depending upon the value of the specified boolean object.

valueOf() :

The Boolean.valueOf() is an inbuilt function in JavaScript which is used to return a boolean value either "true" or "false" depending upon the value of the specified boolean object.

It returns false if the string argument is null otherwise it returns true.

boolean.valueOf()

Parameter: Here nothing as a parameter is passed.

Return Values: It returns a boolean value either "true" or "false" depending upon the value of the specified boolean object.

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Boolean Object</title>
</head>
<body>
<script type="text/javascript">
    //boolean object with true value
    var myBoolean = new Boolean(true);
    console.log('Boolean object');
    console.log(myBoolean);
    console.log('Boolean object created with ' + myBoolean);
    console.log('Value of myBoolean Object is');
    console.log(myBoolean.valueOf());
    console.log('String value of myBoolean object is ');
    console.log(myBoolean.toString());

    var myBoolean1 = new Boolean(-1);
    console.log('Boolean object');
    console.log(myBoolean1);
    console.log('Boolean object created with ' + myBoolean1);
    console.log('Value of myBoolean1 Object is');
    console.log(myBoolean1.valueOf());
    console.log('String value of myBoolean1 object is ');
```

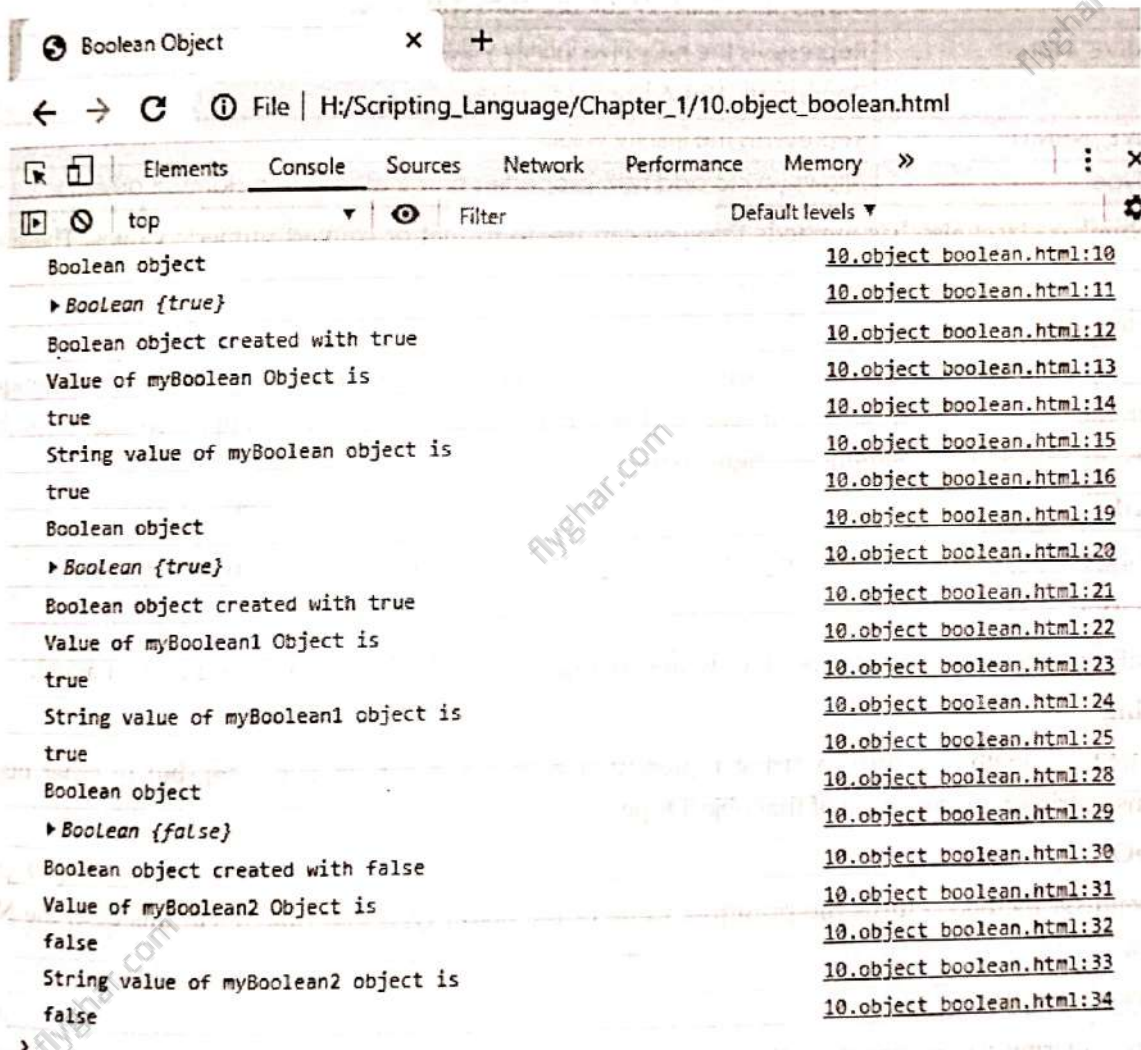


```

console.log(myBoolean1.toString());
var myBoolean2 = new Boolean(false);
console.log('Boolean object');
console.log(myBoolean2);
console.log('Boolean object created with ' + myBoolean2)
console.log('Value of myBoolean2 Object is');
console.log(myBoolean2.valueOf());
console.log('String value of myBoolean2 object is ');
console.log(myBoolean2.toString());
</script>
</body>
</html>

```

Output



Number Object

JavaScript automatically creates Number objects to store numeric values in a script. You can create a Number object with the statement.

```
var n = new Number( value );
```

The constructor argument value is the number to store in the object. Although you can explicitly create Number objects, normally the JavaScript interpreter creates them as needed.

Number object includes various properties and methods for manipulating or retrieving information about numbers. All of the properties available to the Number object are read-only constants, meaning that their values always remain the same and cannot be changed.

Four properties are included in the Number object:

Property	Description
MIN_SAFE_INTEGER	Represents the maximum safe integer in JavaScript (253 - 1).
MAX_VALUE	Returns the largest numeric value representable in JavaScript, approximately 1.79E+308. Values larger than MAX_VALUE are represented as Infinity.
MIN_SAFE_INTEGER	Represents the minimum safe integer in JavaScript (-(253 - 1)).
MIN_VALUE	Returns the smallest positive numeric value representable in JavaScript, approximately 5e-324. It is closest to 0, not the most negative number. Values smaller than MIN_VALUE are converted to 0.
NEGATIVE_INFINITY	Represents the negative infinity value.
NaN	Represents "Not-A-Number" value.
POSITIVE_INFINITY	Represents the infinity value.
Prototype	Allows you to add new properties and methods to a Number object.

The Number object also has methods that you can use to format or convert numeric values. The methods are:

toExponential

The toExponential method returns a string representation of a number in exponential form. The uniqueness of each method is in the arguments it accepts. The toExponential method has an optional argument that can be used to set how many significant digits to use.

toFixed

The toFixed method determines the post-decimal precision based on the argument passed.

toPrecision

The toPrecision method determines the significant digits to display based on the argument passed.

toString

The toString method returns a string representation of a number (in this case), but in other objects, it returns a string representation of that object type.

valueOf

The valueOf method returns the primitive value of the object type that calls it—in this case, the Number object.

toLocaleString()

Returns a string value version of the current number in a format that may vary according to a browser's locale settings.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Number Object</title>
</head>
<body>
```

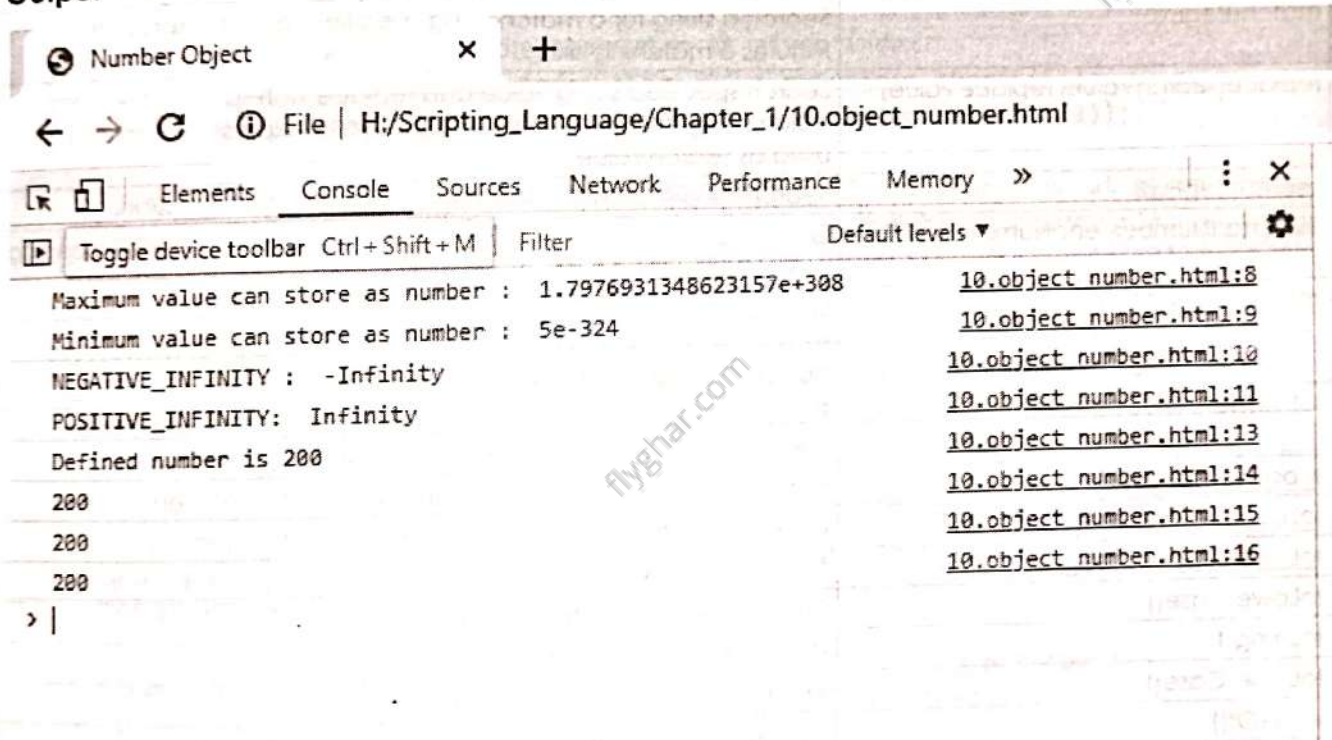


```

<script type="text/javascript">
  console.log('Maximum value can store as number : ' + Number.MAX_VALUE);
  console.log('Minimum value can store as number : ' + Number.MIN_VALUE);
  console.log('NEGATIVE_INFINITY : ' + Number.NEGATIVE_INFINITY);
  console.log('POSITIVE_INFINITY: ' + Number.POSITIVE_INFINITY);
  var myNumber = new Number(200);
  console.log('Defined number is ' + myNumber);
  console.log(myNumber.toString());
  console.log(myNumber.toLocaleString())
  console.log(myNumber.valueOf());
</script>
</body>
</html>

```

Output



String Object

There are two types of string data types: primitive strings and String objects. String objects have many methods for manipulating and parsing strings of text. Because these methods are available to primitive strings as well, in practice, there is no need to differentiate between the two types of strings.

String properties

length

Read-only value containing the number of characters in the string. We use the length property to return the length of a string only: We cannot change it externally.

```

var myString = "This is test String";
console.log(myString.length);

```


Some useful method of string object

Method	Description
charAt(position)	Returns the character at the specified position (in Number).
charCodeAt(position)	Returns a number indicating the Unicode value of the character at the given position (in Number).
concat([string..])	Joins specified string literal values (specify multiple strings separated by comma) and returns a new string.
indexOf(SearchString, Position)	Returns the index of first occurrence of specified String starting from specified number index. Returns -1 if not found.
lastIndexOf(SearchString, Position)	Returns the last occurrence index of specified SearchString, starting from specified position. Returns -1 if not found.
localeCompare(compareString, position)	Compares two strings in the current locale. Return -1 if the reference string is sorted before the compareString0 if the two strings are equal1 if the reference string is sorted after the compareString
match(RegExp)	Search a string for a match using specified regular expression. Returns a matching array.
replace(searchValue, replaceValue)	Search specified string value and replace with specified replace Value string and return new string. Regular expression can also be used as searchValue.
search(RegExp)	Search for a match based on specified regular expression.
slice(startNumber, endNumber)	Extracts a section of a string based on specified starting and ending index and returns a new string.
split(separatorString, limitNumber)	Splits a String into an array of strings by separating the string into substrings based on specified separator. Regular expression can also be used as separator.
substr(start, length)	Returns the characters in a string from specified starting position through the specified number of characters (length).
substring(start, end)	Returns the characters in a string between start and end indexes.
toLocaleLowerCase()	Converts a string to lower case according to current locale.
toLocaleUpperCase()	Converts a sting to upper case according to current locale.
toLowerCase()	Returns lower case string value.
toString()	Returns the value of String object.
toUpperCase()	Returns upper case string value.
valueOf()	Returns the primitive value of the specified string object.

String Methods for HTML

Method	Description
anchor()	Creates an HTML anchor <a>element around string value.
big()	Wraps string in <big> element.
blink()	Wraps a string in <blink> tag.
bold()	Wraps string in tag to make it bold in HTML.
fixed()	Wraps a string in <tt> tag.
fontcolor()	Wraps a string in a tag.
fontsize()	Wraps a string in a tag.
italics()	Wraps a string in <i> tag.
link()	Wraps a string in <a>tag where href attribute value is set to specify string.
small()	Wraps a string in a <small>tag.
strike()	Wraps a string in a <strike> tag.
sub()	Wraps a string in a <sub>tag
sup()	Wraps a string in a <sup>tag

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>String Object</title>
</head>
<body>
<script type="text/javascript">
  var mySubject = "Scripting Language";
  var mySubjectObjective = "To understand the use of scripting language for web
development";
  //length property
  console.log('Total length of mySubject string "' + mySubject + '" is ' +
mySubject.length);

  //charAt method to find the character at given index
  console.log('Character at index 5 is : ' + mySubject.charAt(5));
  console.log('Character at index 20 is : ' + mySubject.charAt(12));

  //charCodeAt method to find the Unicode value of the character at given index
  console.log('Character code at index 5 is : ' + mySubject.charCodeAt(5));
  console.log('Character code at index 20 is : ' + mySubject.charCodeAt(12));

  //charCodeAt method to find the Unicode value of the character at given index
  console.log('Character at index 5 is : ' + mySubject.charCodeAt(5));
  console.log('Character at index 20 is : ' + mySubject.charCodeAt(12));

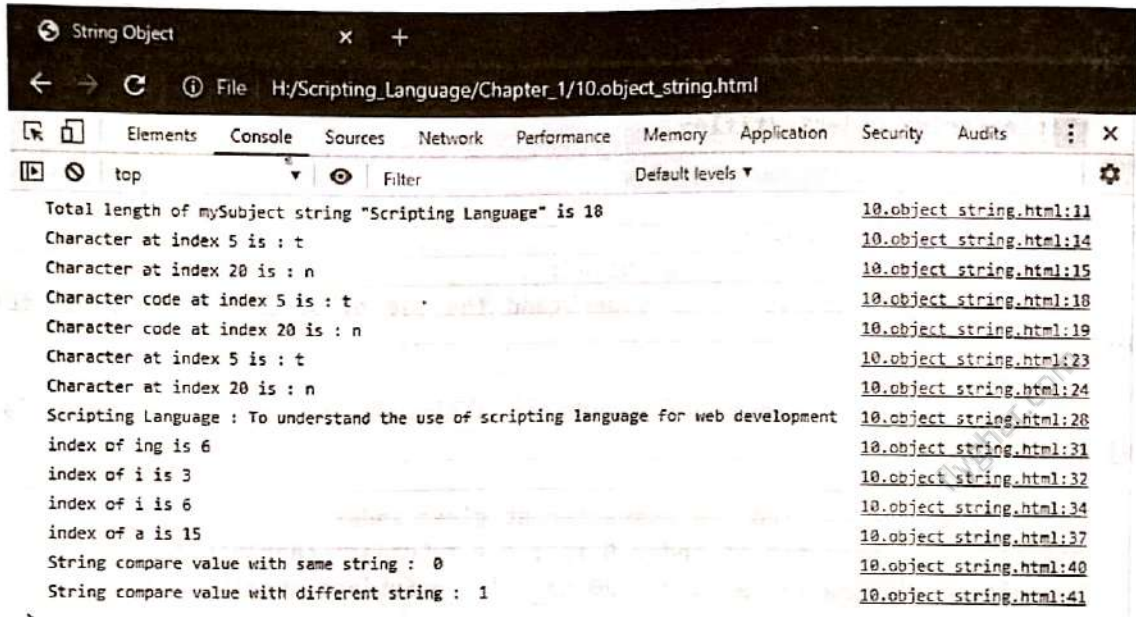
  //concat method is used to concatenation multiple string value
  var newString = mySubject.concat(" : ", mySubjectObjective);
  console.log(newString);

  //indexOf give string or character, return -1 if string not found
  console.log('index of ing is ' + mySubject.indexOf('ing'));
  console.log('index of i is ' + mySubject.indexOf('i'));
  //indexOf also give string or character from specified position
  console.log('index of i is ' + mySubject.indexOf('i',6));

  //lastIndexOf give string or character from last index, return -1 if string not
found
  console.log('index of a is ' + mySubject.lastIndexOf('a'));

  var myName = "Rajesh";
  console.log('String compare value with same string : ' +
myName.localeCompare('Rajesh'));
  console.log('String compare value with different string : ' +
myName.localeCompare('rajesh'));
</script>
</body>
</html>
```

Output



DATE OBJECT

JavaScript has the built in Date object and related methods. It is used to work with dates and times. The Date object is created by using new keyword, i.e. new Date(). The Date object can be used date and time in terms of millisecond precision within 100 million days before or after 1/1/1970. But using another methods we can only get and set the year, month, day, hour, minute, second, and millisecond fields using local or UTC or GMT time. So we can represent date and time till the year 275755 using Date object.

We can declare date in different ways, the basic things is that the date objects are created by the new Date() operator.

- new Date()
- new Date(year, month, day, hours, minutes, seconds, milliseconds)
- new Date(milliseconds)
- new Date(date string)

new Date()

The Date() constructor creates a Date object which sets the current date and time depend on the browser's time zone. It does not accepts any value.

new Date(milliseconds)

This method accepts single parameter milliseconds which indicates any numeric value. This argument is taken as the internal numeric representation of the date in milliseconds.

new Date(datastring)

This method accepts single parameter dataString which indicates any string value. It is a string representation of a date and return the datastring with the day.

new Date(year, month, date, hour, minute, second, millisecond)

This method accepts seven parameters as mentioned above and described below:

year: Integer value which represents the year. This should always specify the year in full, i.e. use 2018, instead of use 18.

month: Integer value which represents the month. The integer values are starts from 0 for January to 11 for December.

date: Integer value which represents the date.

hour: Integer value which represents the hour in 24-hours scale.

minute: Integer value which represents the minute.

second: Integer value which represents the second.

millisecond: Integer value which represents the millisecond.

Example:

```
<!DOCTYPE html>
<html>
<head>
    <title>Date Object</title>
</head>
<body>
<script>
    // Prints today's date
    var today = new Date();
    console.log('Today is: ' + today);
    // Prints date by given milliseconds
    var millisecondDate = new Date(10000);
    console.log('Date of 10000 milliseconds is: ' + millisecondDate);
    var millisecondDate = new Date(999999999999);
    console.log('Date of 10000 milliseconds is: ' + millisecondDate);
    //date with datestring
    var dateString = new Date("May 9, 2020 09:33:12");
    console.log('Date of given datestring is: ' + dateString);
    //date with given all separate parameter
    var givenDate = new Date(2018, 2, 13, 14, 52, 17, 456);
    console.log('Date of given Date is: ' + givenDate);
</script>
</body>
</html>
```

Epoch time

Epoch time, also referred to as zero time, is represented by the date string 01 January, 1970 00:00:00 Universal Time (UTC), and by the 0 timestamp. We can test this in the browser by creating a new variable and assigning to it a new Date instance based on a timestamp of 0.

Epoch time was chosen as a standard for computers to measure time by in earlier days of programming, and it is the method that JavaScript uses. It is important to understand the concept of both the timestamp and the date string, as both may be used depending on the settings and purpose of an application.

```
var epochTime = new Date(0);
console.log(epochTime);
```

Methods of Date object

Below methods are returns all the values according to local time:

METHOD	DESCRIPTION
Date()	It returns presents day's date and time.
getDate()	It returns the day for the specified date.
getDay()	It returns the day of the week for the specified date.
getFullYear()	It returns the year of the specified date.
getYear()	This method returns the year in the specified date.
getHours()	It returns the hour in a specified date.
getMilliseconds()	It returns the milliseconds in the specified date.
getMinutes()	It returns the minutes in the specified date.
getMonth()	It returns the month in the specified date. This also find the month.
getSeconds()	This method returns the seconds in the specified date.
getTime()	This method returns the date in terms of numeric value as milliseconds.
setDate()	This method sets the day of the month for a specified date.
setFullYear()	This method sets the full year for a specified date.

Below methods are returns all the values according to universal time:

METHODS	DESCRIPTION
getUTCDate()	It returns the day of a month for a specified date.
getUTCDay()	It returns the day of the week for a specified date.
getUTCFullYear()	This method returns the year for a specified date.
getUTCHours()	It returns the hours in a specified date.
getUTCMilliseconds()	This method returns the milliseconds form for a specified date.
getUTCMinutes()	This method returns the minutes in a specified date.
getUTCMonth()	This method returns the month for a specified date.

Date Example for Specified Format:

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Date Object</title>
</head>
<body>
  <h1>Date Object</h1>
  <script type="text/javascript">
    var date = new Date();
    document.write(date);
```



```
var year = date.getFullYear();
// document.write(year);

var month = date.getMonth();
var months =
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'June', 'JULY', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec'];
// document.write(months[month]);

var month_date = date.getDate();
// document.write(month_date);

var day = date.getDay();
var days = ['sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat'];
// document.write(day);

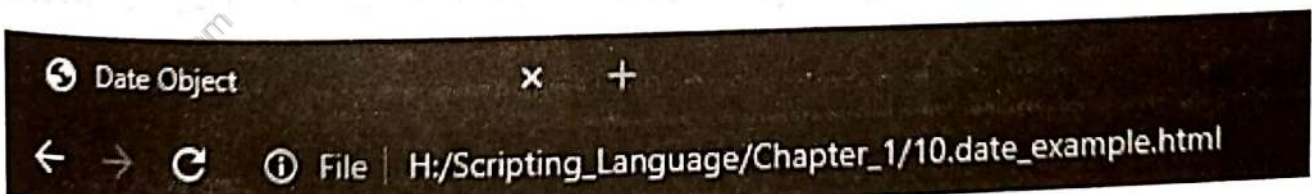
var hours = date.getHours();
// document.write(hours);

var minutes = date.getMinutes();
// document.write(minutes);

var seconds = date.getSeconds();
// document.write(seconds);

var am_pm = 'AM';
if (hours > 12) {
    hours = hours-12;
    am_pm = 'PM';
}
var today = year + ' ' + months[month] + ' ' + month_date + ' ' +
days[day] + ' ' + hours + ':' + minutes + ':' + seconds + ' ' + am_pm;
document.write(today);
</script>
</body>
</html>
```

Output



Date Object

Sat May 09 2020 14:35:48 GMT+0545 (Nepal Time)2020 May 9 sat 2:35:48 PM

Running clock example:

```

<!DOCTYPE html>
<html>
<head>
    <title>Date Object</title>
</head>
<body>
    <h1>Date Object</h1>
    <div id="clock"></div>
    <script type="text/javascript">
        function clock(){
            var date = new Date();
            document.getElementById('clock').innerHTML = date;
            setTimeout(clock,1000);
        }
        clock();
    </script>
</body>
</html>

```

Output

Date Object

Sat May 09 2020 14:40:09 GMT+0545 (Nepal Time)

JavaScript Math Objects

JavaScript math object is a top-level, a predefined object for mathematical constants and functions. This object cannot be created by the user. It is a predefined object. Mathematical properties and functions can be calculated by `math.property` or `math.method`.

JavaScript Math Objects Property

Name	Description
E	Returns the mathematical constant E, the base of natural logarithms, approximately 2.718.
LN10	Returns the natural logarithm of 10, approximately 2.302.
LN2	Returns the natural logarithm of 2, approximately 0.693.
LOG10E	Returns the base 10 logarithm of E (approximately equal to 0.434).
LOG2E	Returns the base 2 logarithm of E (approximately equal to 1.442).
PI	Returns the ratio of the circumference of a circle to its diameter (approximately 3.14159).
SQRT1_2	Returns the square root of 1/2 i.e. 0.5, approximately 0.707.
SQRT2	Returns the square root of 2 (approximately 1.414).

Some methods of Math Object

Methods	Description
<u>abs()</u>	It returns the absolute value of the given number.
<u>acos()</u>	It returns the arccosine of the given number in radians.
<u>asin()</u>	It returns the arcsine of the given number in radians.
<u>atan()</u>	It returns the arc-tangent of the given number in radians.
<u>cbrt()</u>	It returns the cube root of the given number.
<u>ceil()</u>	It returns a smallest integer value, greater than or equal to the given number.
<u>cos()</u>	It returns the cosine of the given number.
<u>cosh()</u>	It returns the hyperbolic cosine of the given number.
<u>exp()</u>	It returns the exponential form of the given number.
<u>floor()</u>	It returns largest integer value, lower than or equal to the given number.
<u>hypot()</u>	It returns square root of sum of the squares of given numbers.
<u>log()</u>	It returns natural logarithm of a number.
<u>max()</u>	It returns maximum value of the given numbers.
<u>min()</u>	It returns minimum value of the given numbers.
<u>pow()</u>	It returns value of base to the power of exponent.
<u>random()</u>	It returns random number between 0 (inclusive) and 1 (exclusive).
<u>round()</u>	It returns closest integer value of the given number.
<u>sign()</u>	It returns the sign of the given number
<u>sin()</u>	It returns the sine of the given number.
<u>sinh()</u>	It returns the hyperbolic sine of the given number.
<u>sqrt()</u>	It returns the square root of the given number
<u>tan()</u>	It returns the tangent of the given number.
<u>tanh()</u>	It returns the hyperbolic tangent of the given number.
<u>trunc()</u>	It returns an integer part of the given number.

Example

```

<!DOCTYPE html>
<html>
<head>
    <title>Math Object</title>
</head>
<body>
<script type="text/javascript">
    console.log('Absolute value of 2.5 is ' + Math.abs(2.5));
    console.log('Absolute value of -2.5 is ' + Math.abs(-2.5));
    console.log('arccosine value of 0.9 is ' + Math.acos(0.9));
    console.log('arcsine value of 0.9 is ' + Math.asin(0.9));
    console.log('arctangent value of 0.9 is ' + Math.atan(0.9));
    console.log('Ceiling value of 0.9 is ' + Math.ceil(0.9));
    console.log('Ceiling value of 0.2 is ' + Math.ceil(0.2));
    console.log('Floor value of 0.9 is ' + Math.floor(0.9));
    console.log('Floor value of 0.2 is ' + Math.floor(0.2));
    console.log('Exponent value of 0.2 is ' + Math.exp(0.2));
    console.log('Exponent value of 2.5 is ' + Math.exp(2.5));
    console.log('Exponent value of 3 is ' + Math.exp(3));
    console.log('(Natural) logarithm of a number 5 is' + Math.log(5));
  </script>
</body>
</html>

```

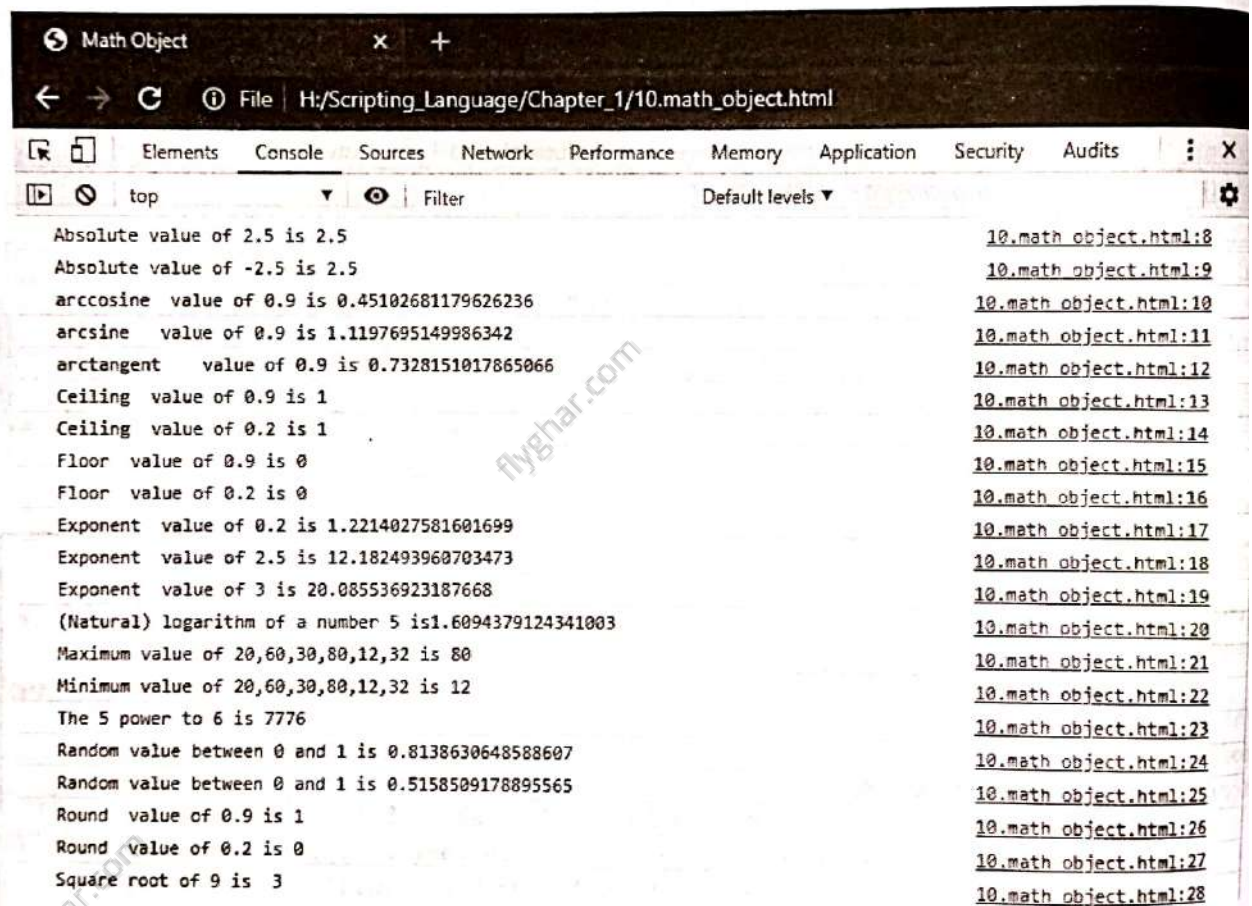


```

        console.log('Maximum value of 20,60,30,80,12,32 is ' +
Math.max(20,60,30,80,12,32));
        console.log('Minimum value of 20,60,30,80,12,32 is ' +
Math.min(20,60,30,80,12,32));
        console.log('The 5 power to 6 is ' + Math.pow(6,5));
        console.log('Random value between 0 and 1 is ' + Math.random());
        console.log('Random value between 0 and 1 is ' + Math.random());
        console.log('Round value of 0.9 is ' + Math.round(0.9));
        console.log('Round value of 0.2 is ' + Math.round(0.2));
        console.log('Square root of 9 is ' + Math.sqrt(9));
</script>
</body>
</html>

```

Output



The JavaScript RegExp Object

Regular expressions are an integral part of JavaScript. The RegExp object offers all the necessary methods to test and match regular expression patterns against strings of characters.

Regular expressions are a built-in data type. The RegExp object is a wrapper for the regular expression primitive.

Regular expressions are built directly into the syntax. As a result, regular expression primitives have access to properties and methods of the RegExp object. The conversion between regular expression primitives and RegExp objects is done automatically by the JavaScript engine.

A regular expression object can be created using the new RegExp() constructor:

Syntax:

```
var re1 = new RegExp(pattern [, flags]);
```

The pattern argument represents the regular expression pattern, while the flags argument is a sequence of modifiers.

```
var re_match_positive = new RegExp("[1-9]\\d*$");
```

RegExp Properties

constructor : Returns a reference to the function that created the RegExp object

global : The value of this property is true if the regular expression g flag was set

ignoreCase : This property is true if the regular expression was compiled with the i flag

lastIndex : Indicates the index position of the last match

multiline : The value of multiline is true if the m flag was used

prototype : Allows the addition of properties and methods to all objects

source : The text pattern of the regular expression

RegExp Methods

exec(text) : Executes the specified pattern on the input string and returns an array of matched strings

test(text) : Tests for a match in the input string and returns true if the pattern matches the string or false otherwise

toString() : Returns a string representation of the RegExp object

valueOf() : Returns the primitive value of the RegExp object

INTERACTING WITH THE BROWSER

The Browser Object Model (BOM) is used to interact with the browser. The default object of browser is window means you can call all the functions of window by specifying window or directly.

The Browser objects are automatically created by a browser at the time of loading a Web page. When an HTML document is opened in a browser window, the browser interprets the document as a collection of hierarchical objects and accordingly displays the data contained in these objects. The browser parses the document and creates a collection of objects, which defines the document and its details.

Browser object is further divided into five sections

- Window Object
- Navigator Object
- History Object
- Screen Object
- Location Object

Window Object

The window object is used to open a window in a browser to display the web page.

All data and information about any browser is attached to the window object as properties and the frames property in the window object returns all the frames in the current window.

Properties of the window object in JavaScript.

Property	Description
closed	returns a Boolean value that specifies whether a window has been closed or not
defaultStatus	specifies the default message that has to be appeared in the status bar of a window
document	specifies a document object in the window
frames	specifies an array of all the frames in the current window
history	specifies a history object for the window
innerHeight	specifies the inner height of a window's content area
innerWidth	specifies the inner width of a window's content area
length	specifies the number of frames contained in a window
location	specifies a location object for the window
name	specifies the name of a window
outerHeight	specifies the height of the outside boundary of a window in pixels
outerWidth	specifies the width of the outside boundary of a window in pixels
parent	returns the parent frame or window of the current window
screenLeft	specifies the x coordinate of the window relative to a user's monitor screen
screenTop	specifies the y coordinate of the window relative to a user's monitor screen
screenX	specifies the x coordinate of the window relative to a user's monitor screen
screenY	specifies the y coordinate of the window relative to a user's monitor screen
self	returns the reference of the current active frame or window
status	specifies the message that is displayed in the status bar of a window, when an activity is performed on the window
top	specifies the reference of the topmost browser window

JavaScript Window Object Methods

The methods of the window object enable you to perform various tasks such as open a url in a new window or to close a window. The following table describes the methods of the Window object in JavaScript.

Method	Description
alert()	displays an alert box with a message and an OK button
blur()	removes the focus from the current window
clearInterval()	clears the timer, which is set by using the setInterval() method
clearTimeout()	clears the timer, which is set by using the setTimeout() method
close()	closes the current window
confirm()	displays a dialog box with a message and two buttons, OK and Cancel
createPopup()	creates a pop-up window
focus()	sets focus on the current window
moveBy()	moves a window relative to its current position
moveTo()	moves a window to an specified position
open()	opens a new browser window
print()	sends a print command to print the content of the current window
prompt()	prompts for input
resizeBy()	resizes a window with the specified pixels
resizeTo()	resizes a window with the specified width and height
scrollBy()	scrolls the content of a window by the specified number of pixels
scrollTo()	scrolls the content of a window up to the specified coordinates
setInterval()	evaluates an expression at specified time intervals in milliseconds
setTimeout()	evaluates an expression after a specified number of milliseconds

Example of window objects

```
<!DOCTYPE html>
<html>
<head>
  <title>Window Object</title>
</head>
<body>
<h1>Window Object</h1>
<input type="button" value="Open New Window"
onclick="openNewWin('WindowObjectMethod.htm')" />
<script type="text/javascript">
  function openNewWin(url)
  {
    var wid = 400;
    var hei = 800;
    var winFeat = "width = " + wid + ", height = " + hei + " , status,
resizable";
    myWin = window.open(url, "subWind", winFeat);
  }
  function disp_alert()
  {
    alert("Hi, This is an alert box.");
  }
</script>
</body>
</html>
```

Navigator Object

The navigator object in JavaScript is used to display information about the version and type of the browser.

JavaScript Navigator Object Properties

The navigator object in JavaScript is used to retrieve properties of client browser. The following table describes properties of the navigator object in JavaScript.

Property	Description
appcodename	specifies the code name of the browser
appname	specifies the name of the browser
appversion	specifies the version of the browser being used
cookieenabled	specifies whether the cookies are enabled or not in the browser
platform	contains a string indicating the machine type for which the browser was compiled
useragent	contains a string representing the value of the user-agent header sent by the client for the server in the http protocol

JavaScript Navigator Object Methods

The methods in the navigator object specify the action related to the browser. The table given below describes the methods of the navigator object in JavaScript.

Method	Description
javaEnabled()	tests whether Java is enabled or not
taintEnabled()	determines whether data tainting is enabled or not

Tainting prevents other scripts from passing information that should be secure and private, such as directory structures or session history.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Navigator Object</title>
</head>
<body>
  <p id="java_enabled"></p>
  <p id="language"></p>
  <p id="platform"></p>
  <p id="app_version"></p>
  <p id="user_agent"></p>
  <p id="product"></p>
  <p id="app_code_name"></p>

  <script type="text/javascript">
    document.getElementById("java_enabled").innerHTML =
      "javaEnabled is " + navigator.javaEnabled();

    document.getElementById("java_enabled").innerHTML =
      "javaEnabled is " + navigator.javaEnabled();

    document.getElementById("language").innerHTML =
      "navigator.language is " + navigator.language;

    document.getElementById("platform").innerHTML =
      "navigator.platform is " + navigator.platform;

    document.getElementById("user_agent").innerHTML =
      "User agent is " + navigator.userAgent;

    document.getElementById("app_version").innerHTML =
      "App version is " + navigator.appVersion;

    document.getElementById("product").innerHTML =
      "navigator.product is " + navigator.product;

    document.getElementById("app_code_name").innerHTML =
      "navigator.appCodeName is " + navigator.appCodeName;
  </script>
</body>
</html>
```


Output

```

javaEnabled is false
navigator language is en-US
navigator platform is Win32
App version is 5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
User agent is Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
navigator product is Gecko
navigator appName is Mozilla
  
```

The History object

The History object is a predefined object in JavaScript that consists of an array of URLs, which are visited by a user in the browser.

JavaScript History Object Properties

Property	Description
length	specifies the number of elements contained in the object
current	specifies the URL of the current entry in the object
next	specifies the URL of the next element in the History list
previous	specifies the URL of the previous element in the History list

JavaScript History Object Methods

The table given below describes the methods of the history object in JavaScript.

Methods	Description
back()	specifies a method that loads the previous URL from the history list
forward()	specifies a method that loads the next URL from the history list
go()	specifies a method that loads a specific URL from the history list

Example

```

<!DOCTYPE html>
<html>
<head>
  <title>History Object</title>
</head>
<body>
  <p id="history_length"></p>
  <button onclick="goBack()">Go back</button>
  <button onclick="goForward()">Go forward</button>
  <button onclick="goBackWithStep()">Go back with certain step</button>
  <script type="text/javascript">
  
```

```

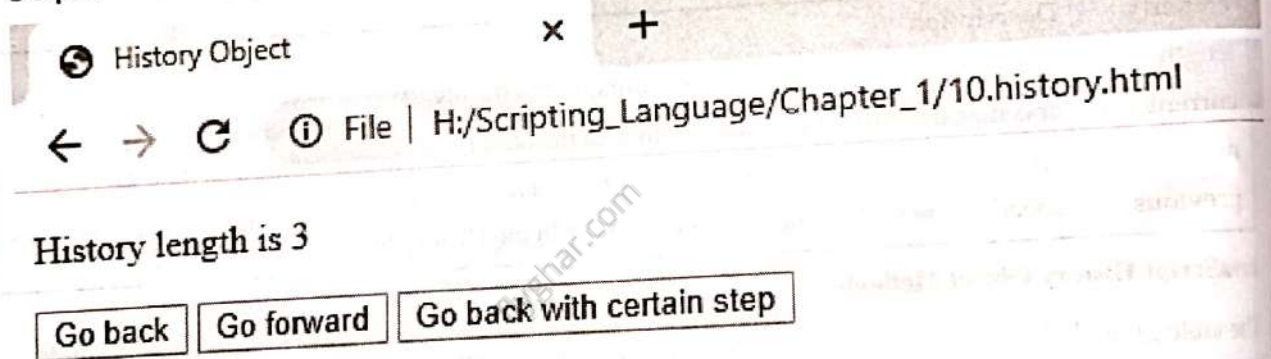
document.getElementById('history_length').innerHTML = "History length is " +
history.length;
function goBack() {
    window.history.back()
}

function goForward() {
    window.history.forward()
}

function goBackWithStep() {
    window.history.go(-4)
}
</script>
</body>
</html>

```

Output



The screen object

The screen object in JavaScript represents the current display screen in your browser. Javascript Screen is a built-in Interface (object type) that is used to fetch information related to the browser screen on which the current window is rendered. It provides information about the dimensions of the display screen such as its height, width, color bits, etc. Here is the general form to access the screen object property:

JavaScript Screen Object Properties

Property	Description
availHeight	specifies the height of the screen, excluding the Windows Taskbar
availWidth	specifies the width of the screen, excluding the Windows Taskbar
colorDepth	specifies the depth of the color palette, in bits, to display images
height	specifies the total height of the screen
pixelDepth	specifies the color resolution, in bits per pixel, of the screen
width	specifies the total width of the screen

Example


```
<!DOCTYPE html>
<html>
<head>
  <title>Screen Object</title>
</head>
<body>
  <h1>Screen Object</h1>
  <p id="pixel_depth"></p>
  <p id="color_depth"></p>
  <p id="avai_screen_height"></p>
  <p id="avai_screen_width"></p>
  <p id="screen_height"></p>
  <p id="screen_width"></p>
  <script type="text/javascript">
    document.getElementById("pixel_depth").innerHTML =
    "Screen pixel depth is " + screen.pixelDepth;

    document.getElementById("color_depth").innerHTML =
    "Screen color depth is " + screen.colorDepth;

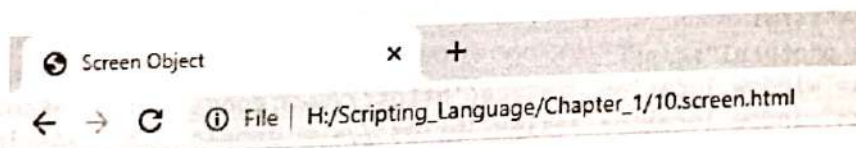
    document.getElementById("avai_screen_height").innerHTML =
    "Available screen height is " + screen.availHeight;

    document.getElementById("avai_screen_width").innerHTML =
    "Available screen width is " + screen.availWidth;

    document.getElementById("screen_height").innerHTML =
    "Screen height is " + screen.height;

    document.getElementById("screen_width").innerHTML =
    "Screen width is " + screen.width;
  </script>
</body>
</html>
```

Output



Screen Object

Screen pixel depth is 24

Screen color depth is 24

Available screen height is 728

Available screen width is 1366

Screen height is 768

Screen width is 1366

The location object in JavaScript helps in storing the information of current URL of the window object. It is a child object of the window object.

JavaScript Location Object Properties

Property	Description
href	represents a string specifying the entire URL
protocol	represents a string at the beginning of a URL up to the first colon (:), which specifies the method of access to the URL
host	represents a string consisting of the hostname and port strings
hostname	represents the server name, subdomain, and domain name of a URL
port	represents a string specifying the communications port that the server uses
pathname	represents a string portion of a URL specifying how a particular resource can be accessed
search	represents a string beginning with a question mark that specifies any query information in an HTTP URL
hash	represents a string beginning with a # specifies an anchor name in an HTTP URL

JavaScript Location Object Methods

The following table lists the methods of the Location object in JavaScript.

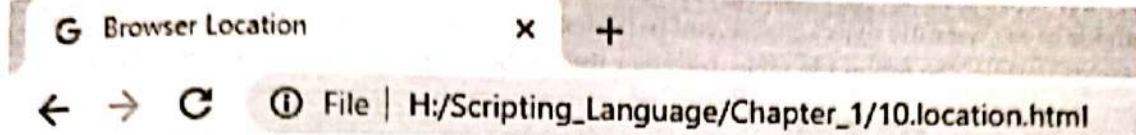
Method	Description
assign()	Loads a new document in the browser
reload()	reloads the current document that is contained in the location.href property
replace()	replaces the current document with the specified new one, you cannot navigate back to the previous document using the browser's Back button

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Browser Location</title>
</head>
<body>
<p id="hostname"></p>
<p id="location_href"></p>
<p id="path_name"></p>
<p id="location_protocol"></p>
<button onclick="window.location.replace('https://www.google.com')">Replace</button>
<button onclick="window.location.assign('https://www.google.com')">Assign</button>

<script>
document.getElementById("hostname").innerHTML =
"Page hostname is: " + window.location.hostname;
document.getElementById("location_href").innerHTML =
"The full URL of this page is:<br>" + window.location.href;
document.getElementById("path_name").innerHTML =
"Page path is: " + window.location.pathname;
document.getElementById("location_protocol").innerHTML =
"The page protocol is: " + window.location.protocol;
</script>
</body>
</html>
```


Output



Page hostname is:

The full URL of this page is:

file:///H:/Scripting_Language/Chapter_1/10.location.html

Page path is: /H:/Scripting_Language/Chapter_1/10.location.html

The page protocol is: file:

Replace

Assign

WINDOWS AND FRAMES

The window object is used to open a window in a browser to display the web page.

All data and information about any browser is attached to the window object as properties and the frames property in the window object returns all the frames in the current window.

Properties of the window object in JavaScript.

Property	Description
closed	returns a Boolean value that specifies whether a window has been closed or not
defaultStatus	specifies the default message that has to be appeared in the status bar of a window
document	specifies a document object in the window
frames	specifies an array of all the frames in the current window
history	specifies a history object for the window
innerHeight	specifies the inner height of a window's content area
innerWidth	specifies the inner width of a window's content area
length	specifies the number of frames contained in a window
location	specifies a location object for the window
name	specifies the name of a window
outerHeight	specifies the height of the outside boundary of a window in pixels
outerWidth	specifies the width of the outside boundary of a window in pixels
parent	returns the parent frame or window of the current window
screenLeft	specifies the x coordinate of the window relative to a user's monitor screen
screenTop	specifies the y coordinate of the window relative to a user's monitor screen
screenX	specifies the x coordinate of the window relative to a user's monitor screen
screenY	specifies the y coordinate of the window relative to a user's monitor screen
self	returns the reference of the current active frame or window
status	specifies the message that is displayed in the status bar of a window, when an activity is performed on the window
top	specifies the reference of the topmost browser window

JavaScript Window Object Methods

The methods of the window object enable you to perform various tasks such as open a url in a new window or to close a window. The following table describes the methods of the Window object in JavaScript.

Method	Description
alert()	displays an alert box with a message and an OK button
blur()	removes the focus from the current window
clearInterval()	clears the timer, which is set by using the setInterval() method
clearTimeout()	clears the timer, which is set by using the setTimeout() method
close()	closes the current window
confirm()	displays a dialog box with a message and two buttons, OK and Cancel
createPopup()	creates a pop-up window
focus()	sets focus on the current window
moveBy()	moves a window relative to its current position
moveTo()	moves a window to an specified position
open()	opens a new browser window
print()	sends a print command to print the content of the current window
prompt()	prompts for input
resizeBy()	resizes a window with the specified pixels
resizeTo()	resizes a window with the specified width and height
scrollBy()	scrolls the content of a window by the specified number of pixels
scrollTo()	scrolls the content of a window up to the specified coordinates
setInterval()	evaluates an expression at specified time intervals in milliseconds
setTimeout()	evaluates an expression after a specified number of milliseconds

Example

```

<!DOCTYPE html>
<html>
<head>
    <title>Window Object</title>
</head>
<body>
<h1>Window Object</h1>
<input type="button" value="Open Window" onclick="openWin()">
<input type="button" value="Open Window New Tab" onclick="openWinNewWindow()">
<input type="button" value="Blur Window" onclick="blurWindow()">
<input type="button" value="Focus Window New Tab" onclick="focusWindow()">
<button onclick="checkWindow()">Check Window Status</button>
<button onclick="printPage()">Print</button>
<button onclick="resizeWindow()">Resize Window</button>
<button onclick="scrollWindow()">Scroll Window</button>
<script type="text/javascript">
    var newWindow;
    function openWin() {
        newWindow= window.open("http://www.google.com");
    }
    function openWinNewWindow() {

```



```
newWindow = window.open("http://www.google.com", "_blank", "toolbar=yes,
location=yes, directories=no, status=no, menubar=yes, scrollbars=yes, resizable=no,
copyhistory=yes, width=800, height=500");
```

```
}

function blurWindow() {
    newWindow.blur();
}

function focusWindow() {
    newWindow.focus();
}

function closeWindow() {
    newWindow.close();
}

function checkWindow() {
    msg = ""
    if (!newWindow) {
        msg = "was never opened";
    } else {
        if (newWindow.closed) {
            msg = "is closed";
        } else {
            msg = "is open";
        }
    }
    alert("My New Window" + msg);
}
```

```
function printPage() {
    window.print();
}
```

```
function resizeWindow() {
    newWindow.resizeBy(50, 50);
    newWindow.focus();
}
```

```
function scrollWindow() {
    window.scrollBy(0, 10);
}
```

```
</script>
</body>
</html>
```

Output

← → C File | H:\Scripting_Language\Chapter_1\10.window.html

Window Object

Open Window Open Window New Tab Blur Window Focus Window New Tab Close Window Check Window Status Print Resize Window Scroll Window

DOCUMENT OBJECT MODEL

The Document Object Model (DOM) is an application programming interface (API) for manipulating HTML and XML documents. The DOM represents a document as a tree of nodes. It defines the logical structure of documents and the way a document is accessed and manipulated. DOM is cross-platform and language-independent ways of manipulating HTML and XML documents.

As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications.

We can create and build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model

A Web page is a document. This document can be either displayed in the browser window or as the HTML source. But it is the same document in both cases. The Document Object Model (DOM) represents that same document so it can be manipulated. The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript.

The different DOM standards like (W3C DOM and WHATWG DOM) standards are implemented in most modern browsers. Many browsers extend the standard, so care must be exercised when using them on the web where documents may be accessed by various browsers with different DOMs.

DOM Structure

When you open any web page in the browser, the HTML of the page is loaded and presented to the screen. JavaScript can be used to dynamically add a page's DOM to add, delete, and modify elements. The DOM represents a document as a tree structure of simple HTML Document.

DOM Nodes

All HTML Elements are treated as nodes or objects. The nodes are attributes, elements, text, link, document etc....

Document - Document node

HTML element - Element node

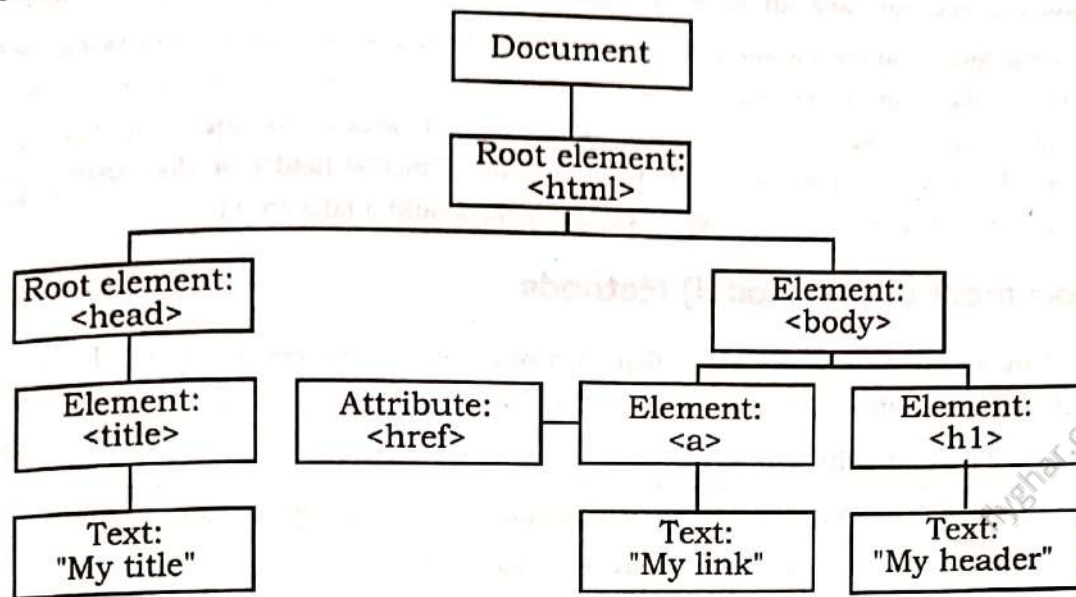
HTML attribute - Attribute Nodes

Text inside HTML elements - Text nodes

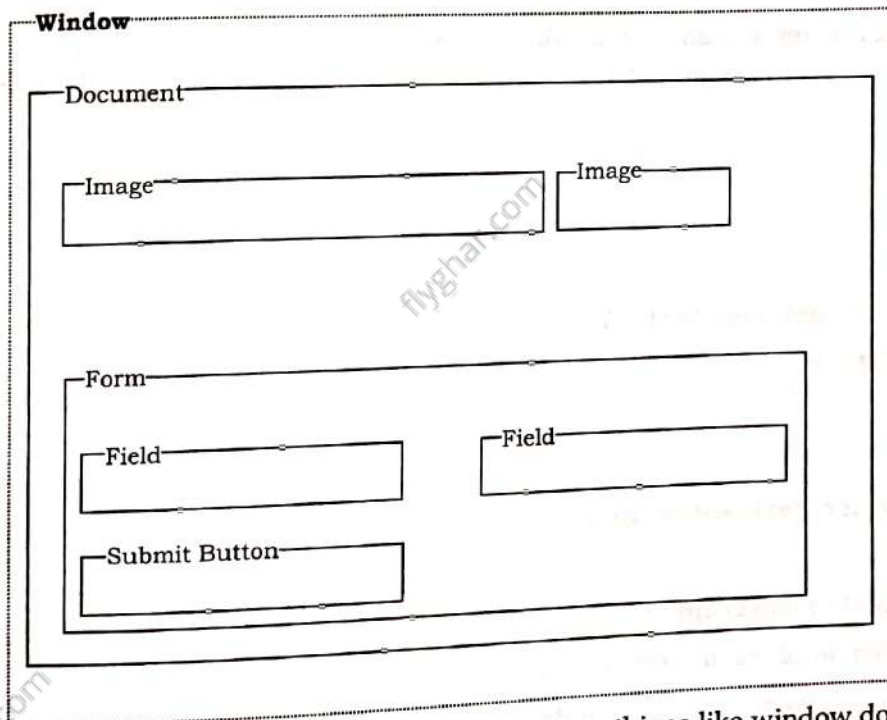
Simple example of DOM tree of following HTML code

```
<!DOCTYPE html>
<html>
<head>
  <title>My title</title>
</head>
<body>
  <h1>My header</h1>
  <a href="mylink.html">My link</a>
</body>
</html>
```


DOM tree of above HTML code.



Accessing DOM Objects



An object nesting is denoted by a dot between them, so we have things like `window.document.image` and `document.form.field`. Remember, the window level of nesting can usually be omitted unless you've created more than one.

When the document is fully loaded, your browser creates arrays for the images, forms, links and anchors (and the rest). It then places all the objects of each type into these arrays, indexed as they appear in the source code. So, the reference to the first image to appear on the page will be `document.images[0]`, and the third form will be `document.forms[2]`. Remember, the first index in an array is zero.

You can also give textual references to each element using the name attribute. If you have `` in your code, you can get access to this object with `document.images["cat1"]`, which makes things a little easier. Remember the quotes around the name.

In recent browsers, you can leave out the array notation entirely and refer to it as `document.cat1`.

Each entry in the `forms[]` array has another array inside it, `elements[]`, with each index being the fields and buttons that make the form up. So, you have an extra level of nested objects. Using the diagram above with a form named `form` and a field named `field`, to get access to the first field, you write `document.forms[0].elements[0]` or `document.forms["form"].elements["field"]` or `document.form.field`. All three methods are equivalent. You can even mix and match, should it take your fancy.

DOM (Document Object Model) Methods

All HTML elements are objects. We know that each object has properties and methods. The Document object HTML contains methods that allow you to select the desired HTML element.

- `getElementById()`: returns the element having the given id value.
- `getElementsByName()`: returns all the elements having the given name value.
- `getElementsByTagName()`: returns all the elements having the given tag name.
- `getElementsByClassName()`: returns all the elements having the given class name.

`getElementById()`

This is used to find and retrieve an HTML using a specific ID. The specific element ID must be unique. In the example below, the `getElementById()` method is used to select the element with `id="myid"` and change its content.

```
<!DOCTYPE html>
<html>
<head>
<title>Document object Model</title>
<meta charset="utf-8">
</head>
<body>
<h2>Document Object getElementById </h2>
<p id="myid"></p>
<script type="text/javascript">
//using getElementById find element by id
    var elm = document.getElementById("myid");
    elm.innerHTML = "Hi, This is Scripting Language";
</script>
</body>
</html>
```

`getElementsByClassName()`

This is used to find and retrieve a collection of all elements with the specific class name. An array index is used to access the elements with the specific tag. HTML collection is an array of objects. Length property is used to find the number of elements. It searches the entire document and returns only the elements that match the specified or given class name.


```
<!DOCTYPE html>
<html>
<head>
<title>DOM Methods</title>
<meta charset="utf-8">
</head>
<body>
<h2>DOM object ClassName</h2>
<h2>JavaScript</h2>
<p class="basic">
```

JavaScript is used along with HTML and CSS. It is one of the core technologies of the World Wide Web. Most powerful websites and browsers support JavaScript, like Google, Amazon, Facebook, etc.</p>

```
<p class="objective">
```

Uses of JavaScript can be used in multiple applications. It can be used for Client-Side Validation, which is for creating web pages. It is used to create web applications and server applications. It can be used to show the error message on the popup menu on websites. You can use JavaScript to create interactive web elements.

```
</p>
```

```
<script type="text/javascript">
```

```
var para = document.getElementsByClassName("basic");
```

```
var len1 = para.length;
```

```
alert("Basic Count :"+len1);
```

```
for (var i=0; i <len1; i++)
```

```
{
```

```
    para[i].style.background="red";
```

```
    para[i].style.color="white";
```

```
}
```

```
var para1 = document.getElementsByClassName("objective");
```

```
var len2 = para1.length;
```

```
alert("Objective Count :"+len2);
```

```
for (var i=0; i <len2; i++)
```

```
{
```

```
    para1[i].style.background="green";
```

```
    para1[i].style.color="white";
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

getElementsByName()

This is used to find and retrieve all HTML elements using the specific name, access array. The element accessed is in an array. If you give the same element name, there is no problem accessing it, the first name is assigned a zero index and the second name the first index. Look at the example below, and you'll clearly understand.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Element By Name</title>
</head>
<body>
<h2>DOM element by Name</h2>
<script type="text/javascript">
    function checkFruit() {
        var frt = document.getElementsByName("fruit")
        if (frt[1].checked)
        {
            alert("It is a Fruit");
        }
        else
        {
            alert("Not a fruit");
        }
    }
</script>
<h2>Select the Fruit</h2>
<br>
Mango<input type="radio" name="fruit">
<br>
    Apple<input type="radio" name="fruit">
<br>
<button onclick="checkFruit()">Check</button>
</body>
</html>
```

getElement ByTagName()

This is used to find and retrieve a collection of elements using the specific tag. An array index is used to access the elements with the specific tag, and HTML collection is an array of objects. You can change the specific tag (color, size, and font).


```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Element by Tag Name</title>
</head>
<body>
<h2>JavaScript </h2>
<h3>Select Element By Tag Name</h3>
<div id="div1">
<p id="basic">
```

JavaScript is used along with HTML and CSS. It is one of the core technologies of the World Wide Web. Most powerful websites and browsers support JavaScript, like Google, Amazon, Facebook, etc.</p>

```
<p id="objective">
```

Uses of JavaScript can be used in multiple applications. It can be used for Client-Side Validation, which is for creating web pages. It is used to create web applications and server applications. It can be used to show the error message on the popup menu on websites. You can use JavaScript to create interactive web elements.</p>

```
</div>
```

```
<p> Most powerful websites and browsers support JavaScript, like Google, Amazon, Facebook, etc.</p></p>
```

```
<script type="text/javascript">
```

```
var tags = document.getElementsByTagName('p');
tags[0].style.background="skyblue";
tags[1].style.color="blue";
tags[2].style.background="yellow";
```

```
</script>
```

```
</body>
```

```
</html>
```

DOM (Document Object Model) Properties

- InnerHtml
- InnerText

InnerHTML

The innerHTML is used to create a dynamic and static web content. It is used mostly in the web pages to generate dynamic HTML such as a registration form, comment form, and links. This content is created using innerHTML Properties.

```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>DOM Property</title>
</head>
<body>
<h2>DOM property innerHTML</h2>
<p id="message"></p>
<ul id="Items">
<li>List 1</li>
<li>List 2</li>
<li>List 3</li>
<li>List 4</li>
<li>List 5</li>
</ul>
<button type="button" onclick="show()">Message</button>
<button type="button" onclick="list()">List</button>
<script type="text/javascript">
    function show()
    {
        var msg = document.getElementById("message"); //message include the button
        using innerHTML
        msg.innerHTML = "Welcome to our website";
    }
    function list() {
        var first =document.getElementById("Items");//list includes the button
        innerHTML
        first.innerHTML =
        "<li>HTML</li><li>CSS</li><li>JavaScript</li><li>PHP</li><li>Angular</li>"
    }
</script>
</body>
</html>

```

Inner Text

The innerHTML returns the normal text of the specific element without HTML interpretation. It is used to create dynamic web content, show validation messages, and password strength.


```
<!DOCTYPE html>
<html>
<head>
<title>DOM property</title>
</head>
<body>
<h2>DOM innerText property</h2>
<p id="para1">JavaScript is used along with <b>HTML</b> and CSS.</p>
<form name="form1">
<br>Name:
<input type="text">
<br><br>Password:
<input type="Password" value="" name="pass" onkeyup="msgs()">
    strength: <span id="str">poor</span><!--below five character the password strength
will be show poor -->
<br>
<input type="button" value="submit" onclick="show1()">
</form>
<script type="text/javascript">
    var = document.getElementById("para1")
    function msgs()
    {
        var char1;
        var len = document.form1.pass.value.length;
        if (len > 5)
        {
            char1="good";
        }
        else
        {
            char1="poor";
        }
        document.getElementById("str").innerText="char";
    }
</script>
</body>
</html>
```

EVENT HANDLING

Events are the actions that result from activities, such as clicking a link or filling a form, by the user. Event handling is the mechanism that controls the event and decides what should happen if an event occurs. An event handler is required to manage proper execution of all these events. Event handlers are an extra attribute of the object. This attribute includes an event's name and the action taken if the event takes place. The window object has access to such information as the toolbar, height and width of the window, prompts, and alerts. Let's see how we can add an event listener (mousedown) to the window object and analyze some of its properties.

To work with events, there are two things we need to do:

- Listen for events
- React to events

Listening for Events

Almost everything we do inside an application results in an event getting fired. Sometimes, our application will fire events automatically, such as when it loads. Sometimes, our application will fire events as a reaction to us actually interacting with it. The thing to note is that our application is attacked by events constantly whether we intended to have them get fired or not. Our task is to tell our application to listen only to the events we care about.

The thankless job of listening to the right event is handled entirely by a function called `addEventListener`. This function is responsible for being eternally vigilant so that it can notify another part of our application when an interesting event gets fired.

The way we use this function looks as follows:

```
source.addEventListener(eventName, eventHandler, false);
```

Some event list

Event	Event is fired
click	When you press down and release the primary mouse button, trackpad, etc.
mousemove	Whenever you move the mouse cursor
mouseover	When you move the mouse cursor over an element. This is the event you would use for detecting a hover!
mouseout	When your mouse cursor moves outside the boundaries of an element.
dblclick	When you quickly click twice.
DOMContentLoaded	When your document's DOM has fully loaded.
load	When your entire document (DOM, external stuff like images, scripts, etc.) has fully loaded.
keydown	When you press down on a key on your keyboard
keyup	When you stop pressing down on a key on your keyboard
scroll	When an element is scrolled around
wheel & DOMMouseScroll	Everytime you use your mousewheel to scroll up or down

Reacting to Events

As we know listening to events is handled by `addEventListener`. What to do after an event is overheard is handled by the event handler. The only distinction between a typical function and one that is designated as the event handler is that our event handler function is specifically called out by name in an `addEventListener` call (and receives an Event object as its argument):

```
document.addEventListener("click", changeColor, false);
```

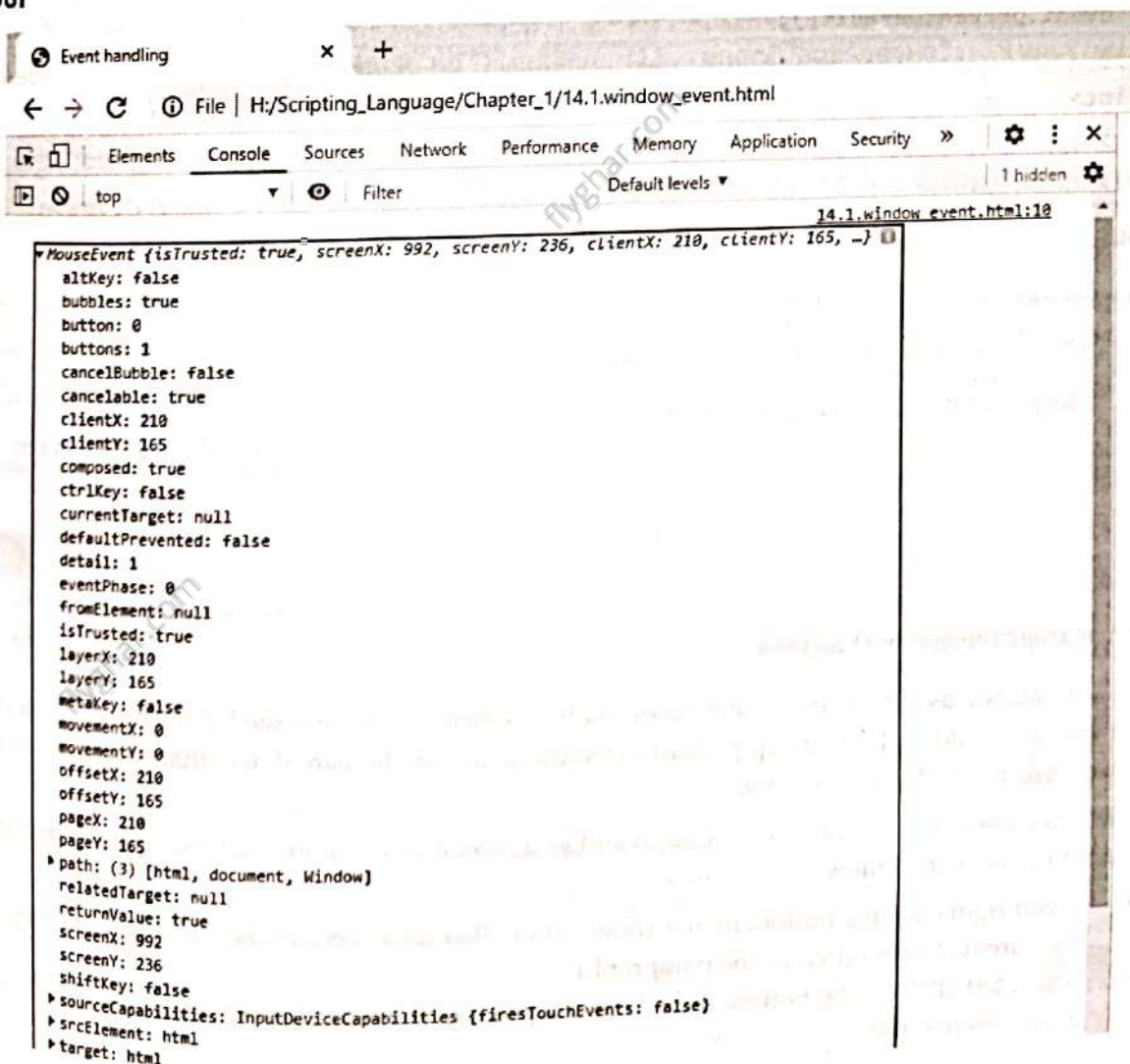
Example: How to add the listener on the window object

The `addEventListener` method is the most preferred way to add an event listener to window, document or any other element in the DOM. There is one more way called "on" property onclick, onmouseover, and so on. But is not as useful, as it does not allow us to add multiple event listeners on the same element. The other methods allow it. An event object is passed as an argument (optional) to the handler which contains all the information related to the event (in our case, mousedown) on the window.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Event handling</title>
</head>
<body>
<script type="text/javascript">
  window.addEventListener("mousedown",function(event){
    alert("Mouse down event captured");
    console.log(event);
  });
</script>
</body>
</html>
```

Output



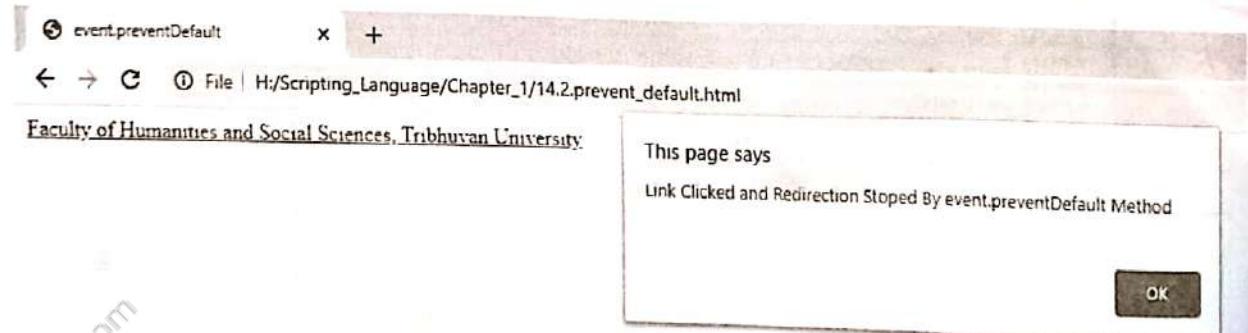
The Event.preventDefault() method

Sometime we don't want an HTML element to behave in the way it is supposed to behave in default. In such a case, we can use this method.

Example: Clicking the anchor element will make the browser redirect to that page by default. Let's try to avoid that.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>event.preventDefault</title>
</head>
<body>
<a href="https://www.tufohss.edu.np/">Faculty of Humanities and Social Sciences,
Tribhuvan University</a>
<script>
  let link = document.querySelector("a"); // It is the method to access the first
  matched element
  link.addEventListener("click", function(event) {
    alert("Link Clicked and Redirection Stopped By event.preventDefault Method");
    event.preventDefault();
  });
</script>
</body>
</html>
```

Output**The Event.stopPropagation() method**

Events flow outwards. There are certain cases, such as when you have nested elements and you perform some event on a child and it ends up performing some action on the parent, too, that you want to avoid. In such cases, this method is a useful one.

Imagine you have a button inside a paragraph and you have attached a mousedown event to both of them. You want to achieve the following use cases:

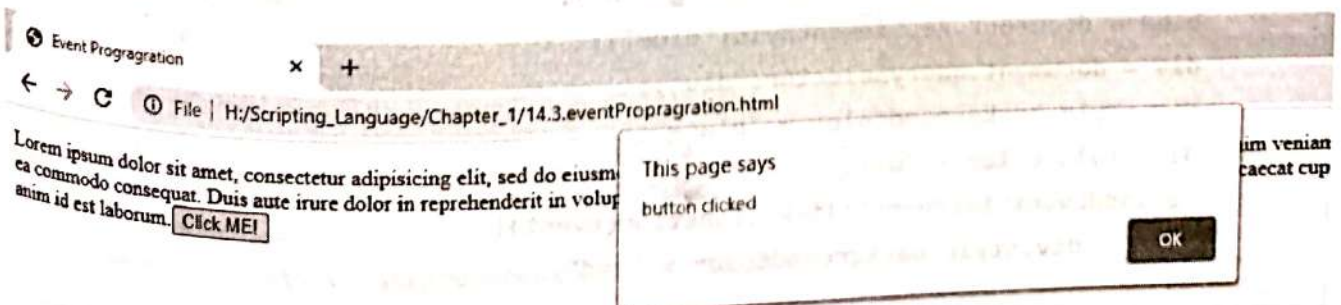
- If you right click the button, then it should show that it has been clicked and does not propagate to the parent element (that is, the paragraph).
- If you left click on the button, then it should propagate outwards normally and fire the paragraph event listener, too.

You can check using left and right click on mouse buttons to see the difference.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Event Progragation</title>
</head>
<body>
<p id="sample">Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.<button
id="click_me"> Click ME! </button></p>
<script>
// Event Listener on the Button and it's logic
document.getElementById("click_me").addEventListener("mousedown", function(event)
{
  alert("button clicked");
  if (event.button == 2) // Right Click
    event.stopPropagation();
});
// Event Listener on the paragraph element with it's logic:
document.getElementById("sample").addEventListener("mousedown", function(event) {
  alert("Paragraph clicked");
});
</script>
</body>
</html>
```

Output



Example with multiple event handling example.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    <button>Click Me!</button>
    <button id="red">Red</button>
    <button id="green">Green</button>
    <button id="blue">Blue</button>
    <div style="font-size: 25px">Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</div>
<script type="text/javascript">
    function myFunction(){
        alert('test form my function')
    }

    document.addEventListener("click",function(event){
        alert("document is clicked")
    });
    //get button as an object
    btn = document.querySelector("button");
    //add eventlistener to button's object
    btn.addEventListener("click",function(event){
        alert("Hi, I am clicked")
    });
    //get red button
    red = document.getElementById("red");
    green = document.getElementById("green");
    blue = document.getElementById("blue");
    div = document.querySelector("div");
    div.style.backgroundColor = "black";
    div.style.color = "white";
    red.addEventListener("click",function(event){
        div.style.backgroundColor = "red";
    })
```

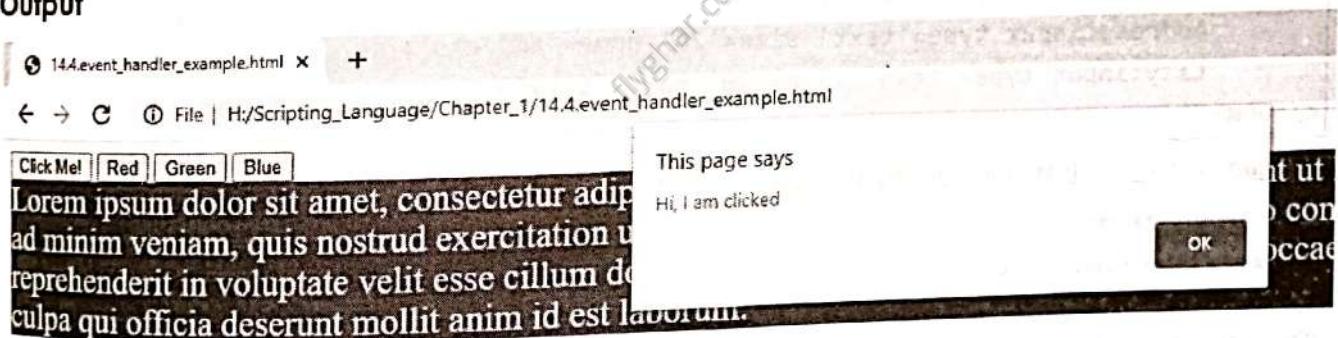


```

green.addEventListener("click",function(event){
    div.style.backgroundColor = "green";
})
blue.addEventListener("click",function(event){
    div.style.backgroundColor = "blue";
})
// uncomment for mouse specific event
/*
button = document.querySelector("button");
button.addEventListener("mousedown", function(event) {
    if (event.button == 0) {
        alert("Left button");
    } else if (event.button == 1) {
        alert("Middle button");
    } else if (event.button == 2) {
        alert("Right button");
    }
});*/
</script>
</body>
</html>

```

Output



Removing an Event Listener

Sometimes, we will need to remove an event listener from an element. The way we do that is by using `addEventListener`'s arch-nemesis, the `removeEventListener` function:

```
something.removeEventListener(eventName, eventHandler, false);
```

As we can see, this function takes the exact type of arguments as an `addEventListener` function. The reason for that is simple. When we are listening for an event on an element or object, JavaScript uses the `eventName`, `eventHandler`, and the `true/false` value to uniquely identify that event listener. To remove this event listener, we need to specify the exact same arguments.

```

document.addEventListener("click", changeColor, false);
document.removeEventListener("click", changeColor, false);
function changeColor() {
    document.body.style.backgroundColor = "#FFC926";
}

```

FORMS

JavaScript has a few built-in methods and properties for dealing with forms.

- You can submit a form by using the `submit()` method.
`document.forms[0].submit()`
- You can reset a form by using the `reset()` method.
`document.forms[0].reset()`

Accessing the form elements

The form validation script needs to access the form in the HTML page to determine what values the user has filled in. So first we have to enter the form by means of the Level 0 DOM.

The general syntax for accessing a form element is:

```
document.forms[number].elements[number]
```

When the page is loaded, JavaScript makes an array `forms` in which it puts all the forms that are on the page. The first form is `forms[0]`, the second is `forms[1]` etc.

Each form has another array in which JavaScript puts all the elements in the form. The first elements `elements[0]`, the second `elements[1]` etc. For every `<input>`, `<select>` and `<textarea>` is an element.

Let's take example

```
<form name="personal" action="" onsubmit="return checkValidation()">
  Name<input type='text' size='20' name='name'>
  Address<input type='text' size='20' name='address'>
  City<input type='text' size='20' name='city'>
</form>
```

We can also access form element by using form & field name.

```
document.personal.name
document.personal.address
document.personal.city
```

The advantage of using names is that you can put all elements somewhere else in the page and still have a working script, while a script using numbers will have to be changed. After all, the input box city is `document.forms[0].elements[2]` in the example above, but when you suddenly put it at the top of the form, it becomes `document.forms[0].elements[0]` and you have to change the script.

Texts, textareas and hidden fields

```
user_input = document.forms[0].text.value;
or
user_input = document.formname.fieldname.value;
```

In above example 'text' is the name of the text field, textarea or hidden field. The value of this element gives the text, so we transfer it to `user_input`.

Assigning value to form element

```
document.forms[0].text.value = 'New content';
```


Select boxes

```
user_input = document.forms[0].select.value;
```

To change the selected option in a select box, you have to change its selectedIndex, like

```
document.forms[0].select.selectedIndex = 2;
```

Now the third option in the box is selected.

Checkboxes

Checkboxes need a slightly different approach. We already know their values, but want to know whether the user has checked them. The checked property tells us. It can have two values: true or false.

```
if (document.forms[0].checkbox.checked) {
    user_input = document.forms[0].checkbox.name
}
```

In above example checkbox is the name of the checkbox. If the checkbox is checked, we take its name (or its value, if you need that bit of data) and transfer it to user_input.

To check a checkbox, set its property checked to true:

```
document.forms[0].checkbox.checked = true
```

Radio buttons

It's not possible to see at once which radio button in a group the user has checked. You need to go through all radios and see which one's checked property is true.

```
for (i=0;i<document.forms[0].radios.length;i++) {
    if (document.forms[0].radios[i].checked) {
        user_input = document.forms[0].radios[i].value;
    }
}
```

In above code radios is the name of the group of radio buttons. The document.forms[0].radios is an array filled with all radio buttons. Loop through all of them and see if it is checked. If one is, transfer the value of that radio button to user_input.

document.forms[0].radios.length gives the number of radio buttons. For each radio button, to check a radio button or set its property checked to true:

```
document.forms[0].radios[i].checked = true;
```

COOKIES

Cookies were originally invented by Netscape to give 'memory' to web servers and browsers. It is a piece of data that is stored on your computer to be accessed by your browser. A cookie is an amount of information that persists between a server-side and a client-side. A web browser stores this information at the time of browsing. A cookie contains the information as a string generally in the form of a name-value pair separated by semi-colons. It maintains the state of a user and remembers the user's information among all the web pages.

The HTTP protocol, which arranges for the transfer of web pages to your browser and browser requests for pages to servers, is state-less, which means that once the server has sent a page to a browser requesting it, it doesn't remember a thing about it. So if you come to the same web page a second, third, hundredth or millionth time, the server once again considers it the very first time you ever came there. This can be annoying in a number of ways. The server cannot remember if you identified yourself when you want to access protected pages, it cannot remember your user preferences, and it cannot remember anything. As soon as personalization was invented, this became a major problem. Cookies were invented to solve this problem. There are other ways to solve it, but cookies are easy to maintain and very versatile.

How cookies work

A cookie is nothing but a small text file that's stored in your browser. It contains some data:

- A name-value pair containing the actual data
- An expiry date after which it is no longer valid
- The domain and path of the server it should be sent to

As soon as you request a page from a server to which a cookie should be sent, the cookie is added to the HTTP header. Server side programs can then read out the information and decide that you have the right to view the page you requested or that you want your links to be yellow on a green background.

So every time you visit the site the cookie comes from, information about you is available. This is very nice sometimes, at other times it may somewhat endanger your privacy. Fortunately more and more browsers give you the opportunity to manage your cookies (deleting the one from the big ad site, for example).

Cookies can be read by JavaScript too. They're mostly used for storing user preferences.

name-value pair

Each cookie has a name-value pair that contains the actual information. The name of the cookie is for your benefit, you will search for this name when reading out the cookie information.

If you want to read out the cookie you search for the name and see what value is attached to it. Read out this value. Of course you yourself have to decide which value(s) the cookie can have and to write the scripts to deal with these value(s).

Expiry date

Each cookie has an expiry date after which it is trashed. If you don't specify the expiry date the cookie is trashed when you close the browser. This expiry date should be in UTC (Greenwich) time.

Domain and path

Each cookie also has a domain and a path. The domain tells the browser to which domain the cookie should be sent. If you don't specify it, it becomes the domain of the page that sets the cookie.

The path gives you the chance to specify a directory where the cookie is active. So if you want the cookie to be only sent to pages in the directory user, set the path to /user. Usually the path is set to /, which means the cookie is valid throughout the entire domain.

document.cookie

Cookies can be created, read and erased by JavaScript. They are accessible through the property document.cookie. Though you can treat document.cookie as if it's a string, it isn't really, and you have only access to the name-value pairs.

```
document.cookie = 'username=abcd; expires=sun, 18 Mar 2020 09:03:58 UTC; path=/'
```


First the name-value pair ('username=abcd') then a semicolon and a space then the expiry date in the correct format ('expires=18 Mar 2020 09:03:58 UTC') again a semicolon and a space then the path (path=/).

```
<!DOCTYPE html>
<html>
<head>
  <title>Cookie</title>
</head>
<body>
  <script type="text/javascript">
    createCookie('username', 'abcd', 7);
    document.write(readCookie('username'));
    remove('username');

    function createCookie(name, value, days) {
      if (days) {
        var date = new Date();
        date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
        var expires = "; expires=" + date.toGMTString();
      } else {
        var expires = "";
      }
      document.cookie = name + "=" + value + expires + "; path=/";
    }

    function readCookie(name) {
      var nameData = name + "=";
      var ca = document.cookie.split(';');
      for (var i = 0; i < ca.length; i++) {
        var c = ca[i];
        while (c.charAt(0) == ' ') c = c.substring(1, c.length);
        if (c.indexOf(nameData) == 0)
          return c.substring(nameData.length, c.length);
      }
      return null;
    }

    function remove(name) {
      createCookie(name, "", -1);
    }
  </script>
</body>
</html>
```

HANDLING REGULAR EXPRESSIONS

Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects. These patterns are used with the `exec()` and `test()` methods of `RegExp`, and with the `match()`, `matchAll()`, `replace()`, `replaceAll()`, `search()`, and `split()` methods of `String`.

Creating a regular expression you construct a regular expression in one of two ways:

- Using a regular expression literal, which consists of a pattern enclosed between slashes, as follows:

```
let re = /ab+c/;
```

Regular expression literals provide compilation of the regular expression when the script is loaded. If the regular expression remains constant, using this can improve performance.

- Calling the constructor function of the `RegExp` object, as follows:

```
let re = new RegExp('ab+c');
```

Using the constructor function provides runtime compilation of the regular expression. Use the constructor function when you know the regular expression pattern will be changing, or you don't know the pattern and are getting it from another source, such as user input.

Regular Expressions Methods

There are mainly two methods for testing regular expressions.

`RegExp.prototype.test()`

This method is used to test whether a match has been found or not. It accepts a string which we have to test against regular expression and returns `true` or `false` depending upon if the match is found or not.

```
var regex = /hello/;
var str = 'hello world';
var result = regex.test(str);
console.log(result);
// returns true
```

`RegExp.prototype.exec()`

This method returns an array containing all the matched groups. It accepts a string that we have to test against a regular expression.

```
var regex = /hello/;
var str = 'hello world';
var result = regex.exec(str);
console.log(result);
// returns [ 'hello', index: 0, input: 'hello world', groups: undefined ]
// 'hello' -> is the matched pattern.
// index: -> Is where the regular expression starts.
// input: -> Is the actual string passed.
```


Simple Regex Patterns

It is the most basic pattern, which simply matches the literal text with the test string.

```
var regex = /hello/;
console.log(regex.test('hello world'));
// true
```

Special Characters

Up until now we've created simple regular expression patterns. Now, let's tap into the full power of regular expressions when handling more complex cases. For example, instead of matching a specific email address let's say we'd like to match a number of email addresses. That's where special characters come into play. There are special symbols and characters that you have to memorize in order to fully understand the regular expressions.

Flags:

Regular expressions have five optional flags or modifiers. Let's discuss the two most important flags:

g – Global search, don't return after the first match

i – Case-insensitive search

You can also combine the flags in a single regular expression. Note that their order doesn't have any effect on the result.

Regular Expression Literal

```
Syntax => /pattern/flags
var regexGlobal = /abc/g;
console.log(regexGlobal.test('abc abc'));
// it will match all the occurrence of 'abc', so it won't return
// after first match.
var regexInsensitive = /abc/i;
console.log(regexInsensitive.test('Abc'));
// returns true, because the case of string characters don't matter
// in case-insensitive search.
```

Meta-characters

Meta-characters are characters with a special meaning. There are many Meta character but I am going to cover the most important ones here.

\d – Match any digit character (same as [0-9]).

\w – Match any word character. A word character is any letter, digit, and underscore. (Same as [a-zA-Z0-9_]) i.e alphanumeric character.

\s – Match a whitespace character (spaces, tabs etc).

\t – Match a tab character only.

\b – Find a match at beginning or ending of a word. Also known as word boundary.

.(period) matches any character except for newline.

`\D` – Match any non-digit character (same as `^[^0-9]`).

`\W` – Match any non-word character (Same as `^[^a-zA-Z0-9_]`).

`\S` – Match a non-whitespaces character.

Quantifiers:

Quantifiers are symbols that have a special meaning in a regular expression.

`+` – Matches the preceding expression 1 or more times.

`*` – Matches the preceding expression 0 or more times.

`?` – Matches the preceding expression 0 or 1 time, which is preceding pattern is optional.

`^` – Matches the beginning of the string, the regular expression that follows it should be at the start of the test string. i.e the caret (^) matches the start of string.

`$` – Matches the end of the string, that is the regular expression that precedes it should be at the end of the test string. The dollar (\$) sign matches the end of the string.

`[N]` – Matches exactly N occurrences of the preceding regular expression.

`[N,]` – Matches at least N occurrences of the preceding regular expression.

`[N,M]` – Matches at least N occurrences and at most M occurrences of the preceding regular expression (where $M > N$).

Alternation `X|Y` – Matches either X or Y.

Examples

Match any 10 digit number:

```
var regex = /^d{10}$/;
console.log(regex.test('9995484545'));
// true
```

Explanation:

- If we want to enforce that the match must span the whole string, we can add the quantifiers `^` and `$`. The caret `^` matches the start of the input string, whereas the dollar sign `$` matches the end. So it would not match if string contain more than 10 digits.
- `\d` matches any digit character.
- `{10}` matches the previous expression, in this case `\d` exactly 10 times. So if the test string contains less than or more than 10 digits, the result will be false.

Match a date with following format DD-MM-YYYY or DD-MM-YY

```
var regex = /^(\d{1,2}-){2}\d{2}(\d{2})?$/;
console.log(regex.test('01-01-1990'));
// true
console.log(regex.test('01-01-90'));
// true
console.log(regex.test('01-01-190'));
// false
```


Explanation

- We have wrapped the entire regular expression inside ^ and \$, so that the match spans entire string.
- (start of first subexpression.
- \d{1,2} matches at least 1 digit and at most 2 digits. - matches the literal hyphen character.
-) end of first subexpression.
- {2} match the first subexpression exactly two times.
- \d{2} matches exactly two digits.
- (\d{2})? Matches exactly two digits. But it's optional, so either year contains 2 digits or 4 digits.

CLIENT SIDE VALIDATION

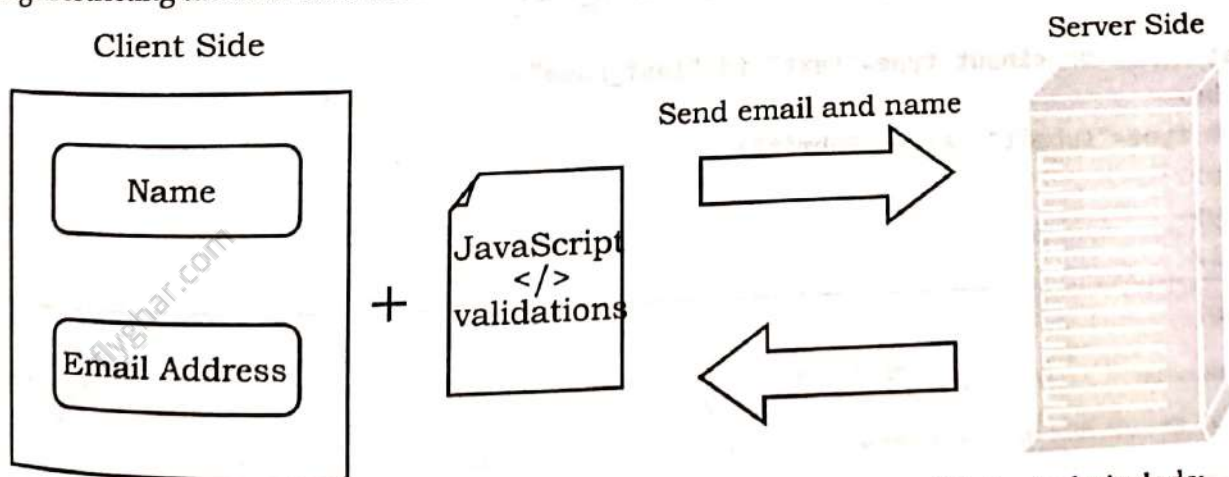
Validation is the process of testing and ensuring that the user has entered required and properly formatted information through the web form. Client side validation is the process in which all the validation and error recovery is carried out on the client side or users browser. It is also known as form Validation, is a process of verifying whether the data inputted by the users on the web page is complete or not.

The data given by the user will be sent to the server to check accuracy. If any mistake persists, that has to be sent back to the user for correction. As this is a time-taking process, the data will be checked by JavaScript before submitting. This will save a lot of time.

Efficient web-programming is possible only by making the information secure. JavaScript Forms authentication is defined as a process of validating the user credentials against some authority.

Form Authentication will be possible only when JavaScript is used along ASP.net or PHP.

When the form is submitted without providing the data in the form field, JavaScript will show a pop-up message restricting the form submission.



It is a process of checking the input given by the user in an application. The validation tasks include:

- Field check
- Date check
- Numeric text field check
- Pattern check

We use the `getElementById` method to get the form input text box and evaluate if the user entered a value. The input can be any value, but there must be at least one character input. We want to trigger the JavaScript check when the user clicks the submit button. Your first thought might be to fire the event when the "onclick" event happens. However, HTML forms have their own event trigger when the user clicks the submit button to send form values to the server. This event is called the "onsubmit" trigger.

Let's add the "onsubmit" event handler and hook into the current HTML page. Sample code for form processing.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Form validation</title>
<script>
function validateData()
{
    var firstName = document.getElementById("first_name");
    if (firstName.value == "")
    {
        alert('Enter firstName');
        return false;
    } else {
        return true;
    }
}
</script>
</head>
<body >
<form action="validation.php" onsubmit="return validateData();">
    First name:<br><input type="text" id="first_name">
<br>
    Last name:<br><input type="text" id="last_name">
<br><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Output

16.form.html

File | H:/Scripting_Language/Chapter_1/16.form.html

First name:

Last name:

Submit

This page says
 Enter firstName

OK

Registration form validation using JavaScript

- Name is required and must be 8 character
- Password and Confirm Password is required and should match
- Phone is required and must be number with 10 digit
- Email is required and must be email format
- DOB is required with format of year(4 digit),month(2 digit) and day(2digit)
- Country is required
- Gender must be selected
- Should accept term and condition to submit form.

Example

```

<!DOCTYPE html>
<html>
<head>
  <title>HTML Validation</title>
  <style type="text/css">
    fieldset{
      border:1px solid #f5a;
      margin-top: 20px;
    }

    legend{
      border:1px solid #5a5;
    }

    .group{
      margin-bottom: 10px;
      border:1px solid #fa2;
      padding: 10px;
      width: 70%
    }

    .group label{
      width: 100px;
      display: inline-block;
    }

    .group#dob_group input{
      width: 16%;
    }

    .group input,select{
      width: 75%;
      height: 20px;

```

```

    }
    .group input[type=radio],input[type=checkbox]{
        width: 5%;
    }

    input[type=submit]{
        border:none;
        background-color: #23a;
        padding: 20px;
        color: #fff;
        width: 15%
    }

    input[type=reset],input[type=submit]{
        border:none;
        background-color: #f50;
        padding: 10px;
        color: #fff;
        width: 100px;
        height: 50px;
    }

    .error_message{
        color: #f00;
    }
</style>
<script type="text/javascript">
function validateForm(){
    var error = 0;
    var name = document.signup.name.value;
    var username = document.signup.username.value;
    var password = document.signup.password.value;
    var repassword = document.signup.repassword.value;
    var phone = document.signup.phone.value;
    var email = document.signup.email.value;
    var year = document.signup.year.value;
    var month = document.signup.month.value;
    var day = document.signup.day.value;
    var country = document.signup.country.value;
    var gender = document.signup.gender.value;

    var msg = '';
    /* start validation for name */
    if (name.trim() == '') {
        document.getElementById('err_name').innerHTML = "Enter name";
        error++;
    } else {

```



```
document.getElementById('err_name').innerHTML = "";
msg += 'Name is ' + name + '<br>';

}

if (username.trim() == '') {
document.getElementById('err_username').innerHTML = "Enter username";
error++;
} else {
if (username.length < 8 ) {
document.getElementById('err_username').innerHTML = "username
should be minimum 8 character";
error++;
} else {
document.getElementById('err_username').innerHTML = "";
msg += 'username is ' + username + '<br>';
}
}

/* end validation for name */

/* start validation for password */
if (password.trim() == '') {
document.getElementById('err_password').innerHTML = "Enter password";
error++;
} else {
document.getElementById('err_password').innerHTML = "";
}

if (repassword.trim() == '') {
document.getElementById('err_repassword').innerHTML = "Enter re
password";
error++;
} else {
document.getElementById('err_repassword').innerHTML = "";

if (password != repassword ) {
document.getElementById('err_repassword').innerHTML = "Password not
matched";
error++;
} else {
document.getElementById('err_repassword').innerHTML = "";
msg += 'Password is ' + password + '<br>';
}
}

/* end validation for password */
```

```

/* Start validation for phone */
var phonepattern = /^[0-9]{10}$/;
if (phone.trim() == '') {
    document.getElementById('err_phone').innerHTML = "Enter phone";
    error++;
} else if (phone.length < 10 ) {
    document.getElementById('err_phone').innerHTML = "Phone must be
10 digit";
    error++;
} else if (!phonepattern.test(phone)) {
    document.getElementById('err_phone').innerHTML = "Phone must be in number
format";
    error++;
} else {
    document.getElementById('err_phone').innerHTML = "";
    msg += 'Phone is ' + phone + '<br>';
}
/* end validation for Phone */

/* Start validation for Email */
var emailpattern = /^[a-zA-Z0-9_\.\\-]+\@(([a-zA-Z0-9\\-])+\.)+([a-zA-
Z0-9]{2,4})+$/;
if (email.trim() == '') {
    document.getElementById('err_email').innerHTML = "Enter email";
    error++;
} else if (!emailpattern.test(email)) {
    document.getElementById('err_email').innerHTML = "Enter valid
email";
    error++;
} else {
    document.getElementById('err_email').innerHTML = "";
    msg += 'Email is ' + email + '<br>';
}
/* end validation for Email */

/* Start validation for DOB */
var yearpattern = /^[0-9]{4}$/;
var monthdaypattern = /^[0-9]{2}$/;
if (year.trim() == '' || month.trim() == '' || day.trim() == '' ) {
    document.getElementById('err_dob').innerHTML = "Enter full date
of birth";
    error++;
} else if (!yearpattern.test(year)) {

```



```

year";
    document.getElementById('err_dob').innerHTML = "Enter valid
    error++;
} else if (!monthdaypattern.test(month)) {
    document.getElementById('err_dob').innerHTML = "Enter valid
    error++;
} else if (!monthdaypattern.test(day)) {
    document.getElementById('err_dob').innerHTML = "Enter valid
    error++;
} else {
    document.getElementById('err_dob').innerHTML = "";
    msg += 'DOB is ' + year + '/' + month + '/' + day + '<br>';
}
/* end validation for DOB */

/* Start validation for Country */
if (document.signup.country.selectedIndex == 0) {
    document.getElementById('err_country').innerHTML = "Select country";
    error++;
} else {
    document.getElementById('err_country').innerHTML = "";
    msg += 'Country is ' + country + '<br>';
}
/* End validation for Country */

/* start validation for Gender */
if (document.signup.gender[0].checked == false &&
    document.signup.gender[1].checked == false ) {
    document.getElementById('err_gender').innerHTML = "Select
    error++;
} else {
    document.getElementById('err_gender').innerHTML = "";
    msg += 'Gender is ' + gender + '<br>';
}
/* end validation for Gender */

/* Start validation for Checkbox */
if (document.signup.term.checked == false){
    document.getElementById('err_term').innerHTML = "Please click on
    accept Term and Condition";
    error++;
}

if (error > 0) {
    return false;
} else {
    document.write(msg);
}
}
</script>
</head>

```

```

<body>
  <h1>Form Validation Using JavaScript</h1>
  <form action="#" method="post" onsubmit="return validateForm()" name="signup">
    <fieldset>
      <legend>Registration form</legend>
      <div class="group">
        <label>Name</label>
        <input type="text" name="name">
        <span id="err_name" class="error_message"></span>
      </div>
      <div class="group">
        <label>Username</label>
        <input type="text" name="username">
        <span id="err_username" class="error_message"></span>
      </div>
      <div class="group">
        <label>Password</label>
        <input type="password" name="password">
        <span id="err_password" class="error_message"></span>
      </div>
      <div class="group">
        <label>Confirm Password</label>
        <input type="password" name="repassword">
        <span id="err_repassword" class="error_message"></span>
      </div>
      <div class="group">
        <label>Phone</label>
        <input type="text" name="phone">
        <span id="err_phone" class="error_message"></span>
      </div>
      <div class="group">
        <label>Email</label>
        <input type="text" name="email">
        <span id="err_email" class="error_message"></span>
      </div>
      <div class="group" id="dob_group">
        <label>DOB</label>
        <input type="text" name="year">
        <input type="text" name="month">
        <input type="text" name="day">
        <span id="err_dob" class="error_message"></span>
      </div>
      <div class="group">
        <label>Country</label>
        <select name="country">
          <option value="">Select Country</option>
          <option value="Nepal">Nepal</option>
          <option value="India">India</option>
          <option value="China">China</option>
        </select>
        <span id="err_country" class="error_message"></span>
      </div>
      <div class="group">
        <label>Gender</label>
        <input type="radio" name="gender" value="Male">Male
        <input type="radio" name="gender" value="Female">Female
        <span id="err_gender" class="error_message"></span>
      </div>
    </fieldset>
  </form>

```



```

<div class="group">
    <label>Faculty</label>
    <input type="checkbox" name="faculty[]" value="BIM">BIM
    <input type="checkbox" name="faculty[]" value="CSIT">CSIT
    <input type="checkbox" name="faculty[]" value="BSW">BSW
    <span id="err_faculty" class="error_message"></span>
</div>
<div class="group">
    <input type="checkbox" name="term" value="I accept Term
and Condition">I accept Term
    and Condition
    <span id="err_term" class="error_message"></span>
</div>
<div class="group" id="btn_group">
    <input type="submit" value="Signup">
    <input type="reset" value="Clear Form">
</div>
</fieldset>
</form>
</body>
</html>

```

Output

HTML Validation x +

File | H:/Scripting_Language/Chapter_1/16.register.html

Form Validation Using JavaScript

Registration form

Name Enter name

Username Enter username

Password Enter password

Confirm Password

Phone Enter phone

Email Enter email

DOB Enter full date of birth

Country Select Country

Gender ☐ Male ☐ Female Select gender

Faculty ☐ BIM ☐ CSIT ☐ BSW

☐ I accept Term and Condition Please click on I accept Term and Condition

Signup Clear Form



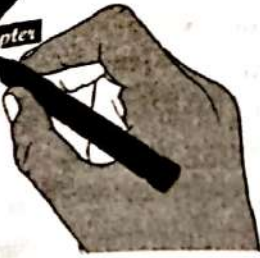
DISCUSSION EXERCISE

1. What is scripting language? Differentiate between scripting and programming language.
2. Describe process of web application works.
3. Why we need scripting language? Describe different type of scripting language.
4. Difference between server side and client side scripting language.
5. What is JavaScript? Explain JavaScript history.
6. Explain feature and limitation of JavaScript.
7. Describe use of script and noscript tag.
8. What is variable, describe how we can create variable in JavaScript.
9. Describe JavaScript coding convention with example.
10. What is the importance of comment? Describe different type of comment in JavaScript.
11. Describe how we can embed JavaScript into your html code.
12. What is identifier? Describe use of identifier with JavaScript.
13. Describe different output functions with example.
14. Describe different keyword and reserve words into JavaScript.
15. Explain different conditional statement with example.
16. Describe different types of loop uses into JavaScript.
17. Explain use of break and continue statement.
18. Describe how to create and traverse JavaScript array.
19. What is exception handling? Describe exception handling in JavaScript.
20. Describe different JavaScript operators with example.
21. Describe use of ForEach loop in JavaScript.
22. Describe different array properties with example.
23. Describe array methods with example.
24. Describe different type of array you use into JavaScript.
25. What is function? Describe different type of function in JavaScript.
26. Describe about nested function with example.
27. What is JavaScript object? Describe the different way to create object in JavaScript.
28. Describe boolean object's properties and method with example.
29. Describe number object's properties and method with example.
30. Describe string object's properties and method with example.
31. Describe math object's properties and method with example.
32. Describe RegExp object's properties and method with example.
33. Describe browser object model with example.
34. Describe properties and method of window object with example.
35. Describe properties and method of navigator object with example.
36. Describe properties and method of history object with example.
37. Describe properties and method of screen object with example.
38. Describe properties and method of location object with example.
39. What is document object model? Describe DOM tree with example.
40. Describe different DOM properties and methods with example.
41. Differentiate between event and event handler.
42. Describe how event handler work in JavaScript.
43. Describe how we can access web form objects in JavaScript with example.
44. Difference between client side and server side form validation.
45. Describe different JavaScript method and properties associated into form validation.
46. What is regular expressions? Describe quantifiers and Meta character with example.

□□□

2

Chapter



SERVER SIDE SCRIPTING WITH DATABASE CONNECTIVITY



CHAPTER OUTLINE

After comprehensive study of this chapter, you will be able to:

- PHP: Introduction to Server Side Scripting Language, PHP introduction, Basic PHP Syntax, Comments in PHP, Variables, PHP Operators. Control Structures(if else, switch, all loops), Arrays, For Each Loop, Functions, Form Handling, PHP \$_GET, PHP \$_POST, PHP \$_REQUEST, PHP date () Function, PHP include File, File Handling, File Uploading, PHP Sessions, Sending Emails, PHP Cookies.
- MySQL: Introduction to MySQL, PHP MySQL Connect to a Database, Closing a Connection, MySQL Data Types. MySQL Insert. MySQL Select. MySQL Where Clause, MySQL Delete, MySQL Update, MySQL Aggregate Functions(sum, avg, count etc); MySQL Order by and Group by Clause, MySQL Subqueries, MySQL Joins,

INTRODUCTION TO SERVER SIDE SCRIPTING LANGUAGE

Server-side scripting is a method of designing websites so that the process or user request is run on the originating server. It is a technique used in web development which involves employing scripts on a web server which produce a response customized for each user's (client's) request to the website. Server-side scripts provide an interface to the user and are used to limit access to proprietary data and help keep control of the script source code. Server side web development uses specific tools and frameworks to build dynamic programs that generate dynamic web sites. This scripting is often used to provide a customized interface for the user. These scripts may assemble client characteristics for use in customizing the response based on those characteristics, the user's requirements, access rights, etc. Server-side scripting also enables the website owner to hide the source code that generates the interface, whereas with client-side scripting, the user has access to all the code received by the client.

It is designed such that user requests are run on the primary server. It provides an interface to users to make requests, give instructions to the server to process. Server side scripting is also responsible for user authentication, and data security.

Server side technology are a set of tools and programming languages that are used to design, build and maintain server side operations of a web application.

Server-side scripting is inevitable in today's world. However, to combat security attacks, all server-side scripts have to be updated to the latest secure versions, and also utilize application firewalls, such as ModSecurity.

Many languages may be used to create these scripts, some of them are:

- ASP
- ASP.NET
- Java
- JavaScript (using SSJS (Server-side JavaScript) e.g., node.js)
- Perl
- PHP
- Python
- R
- Ruby

Advantages of server side scripting

- Server-side scripting prevents increasing of the load as it does not require plugins or browser scripting technology (such as JavaScript). Overloading leads to problems like slow loading, high CPU usage, and even freezing.
- Server side scripting executes scripts on server, it reduces the load on user's computer.
- Server side scripting can be used to create database web application.
- Server side scripting are used to hide scripts from users even if they view the source, only client side scripts can be seen.
- Server side scripting are used to generate dynamic website easily in which site administrator can alter the content at any time.
- Server side scripts do not depend on browser therefore, we don't need to worry about version of browsers.

- It is used to create pages dynamically on the fly. Based on the user interaction, new pages can instantly be created.
- Server-side scripting is necessary to run dynamic pages on browsers that do not fully support JavaScript.
- Server-side scripting does not depend on browser processing as all the processing is performed on the server side.
- Website owners can create their own applications and make use of CMS (content management system) to easily create and update content on the web without coding as the Server-side scripting languages like PHP can be configured to run CMS applications, like WordPress and Joomla.
- As the scripting is done on the server, it is not sent back to the browser, which prevents it from being cloned, copied, or scrutinized for hacking vulnerabilities.
- Loading time of the web pages is often reduced with Server-side scripting which helps to improve your site's Google ranking.
- An increased security is ensured for user privacy and is the preferred choice for e-commerce, membership and social media sites.

Disadvantages of server side scripting

- Server-side scripting is slow at times like: If the user disconnects from the internet, or the web hosting is down, then the script may take a long time to execute.
- The scripting software has to be installed on the server.
- New security concerns are associated with the dynamic scripts as in some cases hackers exploit the code flaws to gain access to servers.
- Server side scripts are difficult to debug
- Server side scripting is vulnerability to hacking.
- A database is required to store the dynamic data that needs a regular backing up and has to be kept secure.
- Here the pages have to be refreshed often in order to show the dynamic content. Ajax, an innovative method of exchanging data with a server is used by the developers which resolve the problem.
- Websites using large applications and with heavy traffic have to make use of more powerful hosting methods like dedicated servers or cloud hosting to cope with demand.

PHP INTRODUCTION

PHP is a widely-used open source general-purpose Server side scripting language that is especially suited for web development and can be embedded into HTML. PHP (recursive acronym for PHP: Hypertext Preprocessor) that earlier stood for Personal Home Pages is a widely-used open source general-purpose scripting language. PHP is especially suited for web development and can be embedded into HTML. PHP is typically used as a server-side language (as opposed to a language like JavaScript that's generally executed on the client-side).

PHP is used to develop dynamic websites or web applications. PHP scripts can only be interpreted on a server that has PHP installed. The client computers accessing the PHP scripts require a web browser only. A PHP file contains PHP tags and ends with the extension ".php".

PHP is known as a server-sided language. That's because the PHP doesn't get executed on your computer, but on the computer you requested the page from. The results are then handed over to you, and displayed in your browser.

Typically PHP is used for Server Side Scripting, Command Line Scripting and even for creating desktop applications.

PHP is compatible with all major Operating Systems including Linux, Windows and MAC OS. It is also compatible with most popular Web Servers such as Apache and LiteSpeed. PHP can be used to create dynamic content such as PDFs, images and is compatible with the most popular types of database tools such as MySQL.

PHP is used to power some of the most popular software such as blogging tools like WordPress and Joomla as well as Ecommerce Shopping carts like Open Cart and Prestashop and Bulletin Board tools like PHPBB and Vbulletin.

PHP can also be used to add extra functionality to your website by integrating applications built for 3rd party websites such as Facebook or by creating PHP based mailing lists or you can even create WordPress Plugins for commercial use.

History of PHP

Version	Release date	Supported until	Description
1.0	8 June 1995		Officially called "Personal Home Page Tools (PHP Tools)". This is the first use of the name "PHP". Development of PHP began in 1994 by Rasmus Lerdorf.
2.0	1 November 1997		Officially called "PHP/FI 2.0". This is the first release that could actually be characterized as PHP, being a standalone language with many features that have endured to the present day.
3.0	6 June 1998	20 October 2000	Development moves from one person to multiple developers. Zeev Suraski and Andi Gutmans rewrite the base for this version.
4.0	22 May 2000	23 June 2001	Added more advanced two-stage parse/execute tag-parsing system called the Zend engine.
4.1	10 December 2001	12 March 2002	Introduced "superglobals" (\$_GET, \$_POST, \$_SESSION, etc.)
4.2	22 April 2002	6 September 2002	Disabled register_globals by default. Data received over the network is not inserted directly into the global namespace anymore, closing possible security holes in applications.
4.3	27 December 2002	31 March 2005	Introduced the command-line interface (CLI), to supplement the CGI.
4.4	11 July 2005	7 August 2008	Fixed a memory corruption bug, which required breaking binary compatibility with extensions compiled against PHP version 4.3.x.
5.0	13 July 2004	5 September 2005	Zend Engine II with a new object model.
5.1	24 November 2005	24 August 2006	Performance improvements with introduction of compiler variables in re-engineered PHP Engine. Added PHP Data Objects (PDO) as a consistent interface for accessing databases.
5.2	2 November 2006	6 January 2011	Enabled the filter extension by default. Native JSON support.

5.3	30 June 2009	14 August 2014	Namespace support; late static bindings, jump label (limited goto), closures, PHP archives (phar), garbage collection for circular references, improved Windows support, sqlite3, mysqlnd as a replacement for libmysql as underlying library for the extensions that work with MySQL, fileinfo as a replacement for mime_magic for better MIME support, the Internationalization extension, and deprecation of ereg extension.
5.4	1 March 2012	3 September 2015	Trait support, short array syntax support. Removed items: allow_call_time_pass_reference, session_register(), session_unregister() and session_is_registered(). Built-in web server. Several improvements to existing features, performance and reduced memory requirements.
5.5	20 June 2013	10 July 2016	Support for generators, finally blocks for exceptions handling, OpCache (based on Zend Optimizer+) bundled in official distribution.
5.6	28 August 2014	31 December 2018	Constant scalar expressions, variadic functions, argument unpacking, new exponentiation operator, extensions of the use statement for functions and constants, new phpdbg debugger as a SAPI module, and other smaller improvements.
6.x	Not released	N/A	Abandoned version of PHP that planned to include native Unicode support.
7.0	3 December 2015	3 December 2018	Zend Engine 3 (performance improvements and 64-bit integer support on Windows), uniform variable syntax, AST-based compilation process, added Closure::call(), bitwise shift consistency across platforms, ?? (null coalesce) operator, Unicodecode point escape syntax, return type declarations, scalar type (integer, float, string and boolean) declarations, <=> "spaceship" three-way comparison operator, generator delegation, anonymous classes, simpler and more consistently available CSPRNG API, replacement of many remaining internal PHP "errors" with the more modern exceptions, and shorthand syntax for importing multiple items from a namespace.
7.1	1 December 2016	1 December 2019	void return type, class constant visibility modifiers
7.2	30 November 2017	30 November 2020	Object parameter and return type hint, Libsodium extension, Abstract method overriding, Parameter type widening
7.3	6 December 2018	6 December 2021	Flexible Heredoc and Nowdoc syntax, support for reference assignment and array deconstruction with list(), PCRE2 support, hrtime() function
7.4	28 November 2019	28 November 2022	Typed Properties 2.0, Preloading, Null Coalescing Assignment Operator, Improve openssl_random_pseudo_bytes, Weak References, FFI – Foreign Function Interface Always available hash extension, Password Hash Registry, Split multibyte string, Reflection for references, Unbundle ext/wddx, New custom object serialization mechanism
8.0	December 2020	December 2023	Just In Time compilation (JIT), arrays starting with a negative index, consistent type errors for internal functions, fatal error for incompatible method signatures

Basic component for PHP Web Development: AMPP Package

AMPP package is combination for "Apache MySQL PHP Perl" package.

Web Server

File servers, database servers, mail servers, and web servers use different kinds of server software. Each of these applications can access files stored on a physical server and use them for various purposes.

A web server is server software or hardware dedicated to running server software that can satisfy client requests on the World Wide Web. The job of a web server is to serve websites on the internet. To achieve that goal, it acts as a middleman between the server and client machines. It pulls content from the server on each user request and delivers it to the web. The biggest challenge of a web server is to serve many different web users at the same time — each of whom is requesting different pages. Web servers process files written in different programming languages such as PHP, Python, Java, and others. They turn them to static HTML files and serve these files in the browser of web users. When you hear the word web server, think of it as the tool responsible for the proper client - server communication.

Apache

The Apache HTTP Server, colloquially called Apache, is free and open-source cross-platform web server software, released under the terms of Apache License 2.0. Apache is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation. It allows website owners to serve content on the web — hence the name "web server." It's one of the oldest and most reliable web servers, with the first version released in 1995.

When someone wants to visit a website, they enter a domain name into the address bar of their browser. Then, the web server delivers the requested files by acting as a virtual delivery man.

How Does Apache Web Server Work?

Although we call Apache a web server, it is not a physical server, but rather software that runs on a server. Its job is to establish a connection between a server and the browsers of website visitors (Firefox, Google Chrome, Safari, etc.) while delivering files back and forth between them (client-server structure). Apache is cross-platform software; therefore it works on both UNIX and Windows servers.

When a visitor wants to load a page on your website, for instance, the "Home" or your "About Us" page, their browser sends a request to your server, and Apache returns a response with all the requested files (text, images, etc.). The server and the client communicate through the HTTP protocol, and the Apache software is responsible for the smooth and secure communication between the two machines.

Apache is highly customizable because it has a module-based structure. These modules allow server administrators to turn additional functionalities on and off. Apache has modules for security, caching, URL rewriting, password authentication, and more. You can also set up your own server configurations through a file called ".htaccess", which is an Apache configuration file supported with all hosting plans.

Apache Pros

- Open-source and free, even for commercial use.
- Reliable, stable software.
- Frequently updated, regular security patches.
- Flexible due to its module-based structure.

- Easy to configure, beginner-friendly.
- Cross-platform (works on both UNIX and Windows servers).
- Works out of the box with WordPress sites.
- Huge community and easily available support in case of any problem.

Apache Cons

- Performance problems on extremely traffic-heavy websites.
- Too many configuration options can lead to security vulnerabilities.

Nginx

Nginx, pronounced Engine-X, is a newer web server application first released in 2004. As of today, it has gained quite a popularity among website owners. Nginx was created to solve the so-called c10k problem, meaning that a web server that uses threads to handle user requests is unable to manage more than 10,000 connections at the same time.

Since Apache uses the thread-based structure, owners of traffic-heavy websites may encounter performance problems. Nginx is one of the web servers that address the c10k problem and probably the most successful one.

Nginx has an event-driven architecture that doesn't create a new process for each request. Instead, it handles every incoming request in a single thread. This master process manages several worker processes that perform the actual processing of requests. The event-based model of Nginx distributes user requests among worker processes in an efficient way, therefore leading to much better scalability.

If you need to manage a high-traffic website, Nginx is an excellent choice, as it can do that by using minimal resources. It cannot be a coincidence that it's used by many high-visibility websites such as Netflix, Hulu, Pinterest, and Airbnb.

However, for small and medium players, Apache comes with a handful of advantages over Nginx, such as its easy configuration, lots of modules, and a beginner-friendly environment.

Tomcat

Tomcat is a web server also developed by the Apache Software Foundation, thus its official name is Apache Tomcat. It's an HTTP server as well, however it powers Java applications instead of static websites. Tomcat can run several different Java specifications such as Java Servlet, JavaServer Pages (JSP), Java EL, and WebSocket.

Tomcat has been created specifically for Java apps, while Apache is a general-purpose HTTP server. You can use Apache together with different programming languages (PHP, Python, Perl, etc.) with the help of the appropriate Apache module (mod_php, mod_python, mod_perl, etc.).

Although you can use a Tomcat server to serve static web pages as well, it's less efficient for that purpose compared to the Apache server. For instance, Tomcat pre-loads the Java Virtual Machine and other Java-related libraries you won't need on most websites.

Tomcat is also less configurable compared to other web servers. For example, to run WordPress, the best choice is a general-purpose HTTP server such as Apache or NGINX.

MySQL

MySQL is an open-source relational database management system that allows you to manage relational databases. Its name is a combination of "My", the name of co-founders Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language. It is open source software backed by Oracle. MySQL can run on various platforms UNIX, Linux, Windows, etc. You can install it on a server or even in a desktop. Besides, MySQL is reliable, scalable, and fast. MySQL is an essential component of the AMP stack, which includes Apache, MySQL, and PHP.

The MySQL server provides a database management system with querying and connectivity capabilities, as well as the ability to have excellent data structure and integration with many different platforms. It can handle large databases reliably and quickly in high-demanding production environments. The MySQL server also provides rich function such as its connectivity, speed, and security that make it suitable for accessing databases.

PHP

PHP is quite a simple language with roots in C and Perl, yet it looks more like Java. It is also very flexible, but there are a few rules that you need to learn about its syntax and structure.

PHP can be embedded in HTML, and it's well suited for web development and the creation of dynamic web pages for web applications, e-commerce applications, and database applications. It's considered a friendly language with abilities to easily connect with MySQL, Oracle, and other databases.

PHP scripts can be used on most of the well-known operating systems like Linux, Unix, Solaris, Microsoft Windows, MAC OS and many others. It also supports most web servers including Apache and IIS. Using PHP affords web developers the freedom to choose their operating system and web server.

The Uses for PHP Are

- It is utilized to make a dynamic site.
- To Interacting with a web server (Apache and so forth)
- To collaborate with any back-end/database server for example MySQL
- To communicate with the local record arrangement of the OS
- To actualize the business sensible layers (at least one)
- It can Encrypt Data
- Access Cookies variable and set treats
- Utilizing PHP you can limit the client to get to explicit page
- PHP typically used to yield HTML code to the program
- Utilized for interface web application with Database
- It is utilized to send and get E-Mails.
- You can utilize PHP to discover the present date, and afterward assemble a schedule for the month.
- On the off chance that you have standard commercials on your site, you can utilize PHP to pivot them arbitrarily.
- Utilizing PHP you can tally your guests on your site.
- You can utilize PHP to make a unique zone of your site for individuals.
- Utilizing PHP you can make a login page for your client.

- Utilizing PHP you can include, erase, and alter components inside your database through PHP. Access treats factors and set treats.
- Utilizing PHP, you can confine clients to get to certain pages of your site.
- It can encode information.

Features of PHP

Simple

It is very simple and easy to use, compare to other scripting language it is very simple and easy, this is widely used all over the world.

Interpreted

It is an interpreted language, i.e. there is no need for compilation.

Faster

It is faster than other scripting language e.g. asp and jsp.

Open Source

PHP is an interpreted scripting language (actually precompiled opcodes), free, open source and distributed under a license authorizing the modification and redistribution.

Platform Independent

PHP code will be run on every platform, Linux, UNIX, Mac OS X, and Windows.

Case Sensitive

PHP is case sensitive scripting language at time of variable declaration. In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive.

Error Reporting

PHP have some predefined error reporting constants to generate a warning or error notice.

Real-Time Access Monitoring

PHP provides access logging by creating the summary of recent accesses for the user.

Loosely Typed Language

PHP supports variable usage without declaring its data type. It will be taken at the time of the execution based on the type of data it has on its value.

Originally stood for Personal Home Page but now it stands for the recursive acronym PHP i.e. Hypertext Preprocessor

PHP place in the world

PHP is a programming language used mostly for building websites. Instead of a PHP program running on a desktop computer for the use of one person, it typically runs on a web server and is accessed by lots of people using web browsers on their own computer

PHP is a programming language. Something in the web server computer reads your PHP programs, which instructions written in this are programming language and figures out what to do. The PHP engine follows the instruction.

PHP is called sever side scripting language because it runs on a web server. A language such as JavaScript can be used as a client-side language because, embedded in a web browser, it can cause that browser, while running on your PC to do something such as pop up a new window.

Advantages of PHP

The following are the advantages of PHP:

PHP is Free

You don't have to pay anyone to use PHP. Whether you run the PHP engine on a beat-up 10-year-old PC in your basement or in a room full of million-dollar "enterprise-class" servers, there are no licensing fees, support fees, maintenance fees, upgrade fees, or any other kind of charge.

PHP is Cross Platform

PHP can be used with a web server computer that runs Windows, Mac OS, Linux and many other version of UNIX.

PHP is widely used

PHP is used on different websites from countless tiny personal homepage to giant's websites

PHP hides its complexity

Powerful ecommerce websites can be build using PHP as well as simple sites. Simpler and complex features can be used simultaneously which doesn't get in the way with each other's

PHP is built for Web Programming

Unlike other programming languages, PHP was created for generating web pages.

BASIC PHP SYNTAX

- PHP code is start with <?php and ends with ?>
- Every PHP statements end with a semicolon (;).
- PHP code save with .php extension.
- PHP contain some HTML tag and PHP code.
- You can place PHP code anywhere in your document.

Unlike other programming language do have its own syntax PHP also has its own syntax

Start Tag: The less than sign(<) followed by ? symbol and the word 'php' i.e : <?php anything written inside this tag is considered as PHP code

End Tag: The ? symbol followed by greater than sign(>) i.e ?>

So the syntax for PHP is:

```
<?php
//statements or blocks
?>
```

PHP can support multiple start and end tags. Each start tag must be closed by its own end tags.

Basic Rules of PHP Programs

1. Start and End Tags
2. Whitespace and Case-Sensitivity
3. Comments

COMMENTS IN PHP

There are two ways in which you can add comments to your PHP code.

1. Single Line comment

The two backslash (//) or the hash (#) is used to post a single line comment in PHP

2. Multiline comments

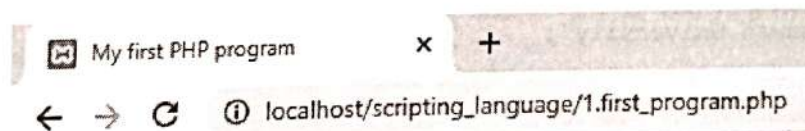
For a multiline comment, start the comment with/* and end it with*/. Everything between the /*and */ is treated as a comment by the PHP engine. Multiline comments are useful for temporarily turning off a small block of code.

Let's run first PHP program.

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>My first PHP program</title>
</head>
<body>
<h1>This is my first php program</h1>
<?php
    echo "This is my first text printed from php code";
?>
</body>
</html>
```

Output



This is my first php program

This is my first text printed from php code

Printing statement in PHP

There are two different function we can use to print or display content .To print the statement in PHP we use **print** or **echo**.

In order to output one or more strings, we can use echo statement. Anything that can be displayed to the browser using echo statement, such as string, numbers, variables values, the results of expressions, etc. It is required to use parenthesis if you want to use more than one parameter. Unlike some other language constructs echo does not behave like a function, so it cannot always be used in the context of a function. The end of echo statement is identified by the semi-colon (;).

	echo	print
Type	Echo is a type of output string	Print is also a type of output string
Parenthesis	Basically, it is written with parenthesis	It can and cannot be written with parenthesis
argument	echo (\$arg1[, \$arg2..])	print(\$arg)
Functionality	It is faster than print	It is slower than echo
Strings	It can output one or more strings	It can output only one string at a time and returns 1

Example

```

<!DOCTYPE html>
<html>
<head>
    <title>echo vs print</title>
</head>
<body>
<?php
echo "Example of echo statement<br/>";
$subject = "Scripting Language";
$page = 305;
echo $subject."<br/>";
echo "Total no of page is : ". $page;
echo "<br/>" , "Total no of page is " , $page;
//invalid
// print "Total no of page into book is " , $page;
print "<br/>Total no of page is $page ";
print "<br/>Tribhuwan University";
?>
</body>
</html>

```

Output

echo vs print

localhost/scripting_language/2.echo_print.php

Example of echo statement

Scripting Language

Total no of page is : 305

Total no of page is 305

Total no of page is 305

Tribhuwan University

PHP Short-Tags

For less motivated typists, an even shorter delimiter syntax is available, known as short-tags. However, to use this feature, you need to enable PHP's `short_open_tag` directive into `php.ini` file.

An example follows:

```
<?print "This is another PHP Syntax.;"?>
```

When short-tags syntax is enabled and you want to quickly escape to and from PHP to output a bit of dynamic text, you can omit these statements using an output variation known as short circuit syntax:

```
<?="This is another PHP Syntax.;"?>
```

This is functionally equivalent to both of the following variations:

```
<? echo "This is another PHP Syntax.;" ?>
```

```
<?php echo "This is another PHP Syntax.;"?>
```

VARIABLES

Variables is an identifier that hold the data of your program manipulates while it runs, such as user information that you've loaded from a database or entries that have been typed into an HTML form. In PHP, variables are denoted by a (dollar sign) \$ followed by the variable's name. To assign a value to a variable, use an equals sign (=). This is known as the assignment operator.

Rules to declare variable in PHP

- A variable starts with the \$ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name can't start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (\$txt and \$TXT both are two different)

Variable names may only include:

- Uppercase or lowercase Basic Latin letters (A-Z and a-z)
- Digits (0-9)
- Underscore (_)
- Any non-Basic Latin character (such as ç), if you're using a character encoding such as UTF-8 for your program file
- Additionally, the first character of a variable name is not allowed to be a digit.

Following are the list of some allowed variable names:

- \$book
- \$myBook
- \$MY_FIRST_BOOK
- \$_m_y_b_o_o_k
- \$myBook2myFriend

The following are the list of some invalid variable names and reason

Variable name	Reason
\$2myFriend	Begins with a number
\$my-book	Unacceptable character
\$mybook@example.com	Unacceptable characters: @ and .
\$hello!book	Unacceptable character: !
\$my+book	Unacceptable character: +

Variable names are case-sensitive. This means that variables named \$book, \$Book, and \$BOOK are separate and distinct. In practice, you should avoid using variable names that differ only by letter case. They make programs difficult to read and debug.

Example

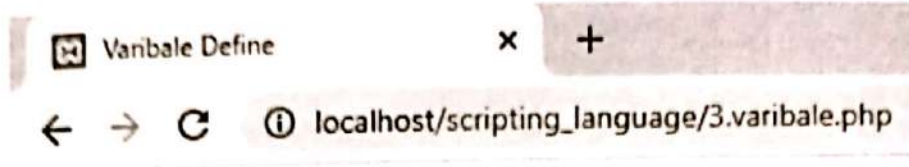
```
<!DOCTYPE html>
<html>
<head>
    <title>Variable Define</title>
</head>
<body>
<h1>Define variable</h1>
<?php
//declaring variable
$name;

//assigning value to variable
$address = "Koteshwor, Kathmandu-31, Kathmandu";

//assign value to predefined variable
$name = "Ram Kumar Gautam";

//print variable
echo $name;
echo '<br/>';
// print variable with string concatenation
echo 'My name is ' . $name;
echo '<br/>';
echo 'I am from ' . $address;
?>
</body>
</html>
```


Output



Define variable

Ram Kumar Gautam

My name is Ram Kumar Gautam

I am from Koteshtwor, Kathmandu-31, Kathmandu

Destroying PHP Variables

To destroy a variable, pass the variable to PHP's `unset()` function. If `unset()` is called inside a user-defined function, it unsets the local variables. If a user wants to unset the global variable inside the function, then he/she has to use `$GLOBALS` array to do so. The `unset()` function has no return value.

Example

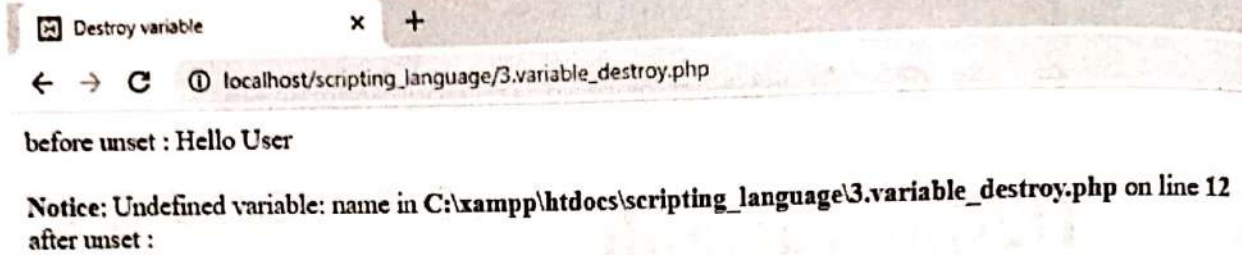
```
<!DOCTYPE html>
<html>
<head>
    <title>Destroy variable</title>
</head>
<body>
<?php
$name='Hello User';
echo " before unset : ".$name;
echo"<br>";
unset($name);
echo "after unset : " . $name;
?>
</body>
</html>
```

After using `unset` function our program throw notice error "Undefined variable".

Difference Between `$var` and `$$var` in PHP

PHP `$$var` uses the value of the variable whose name is the value of `$var`. It means `$$var` is known as reference variable whereas `$var` is normal variable. It allows you to have a "variable's variable" - the program can create the variable name the same way it can create any other string.

Output



Example

```
<!DOCTYPE html>
<html>
<head>
    <title>$ vs $$</title>
</head>
<body>
<?php
$name="Rajesh";
$Rajesh="Rajesh Kumar Shah";
echo $name."<br/>";
echo $$name."<br/>";
?>
</body>
</html>
```

You first assign the value of a variable, (\$name) as the name of another variable.

When you set \$Rajesh to a value, it will replace that variable name with the value of the variable you provide.

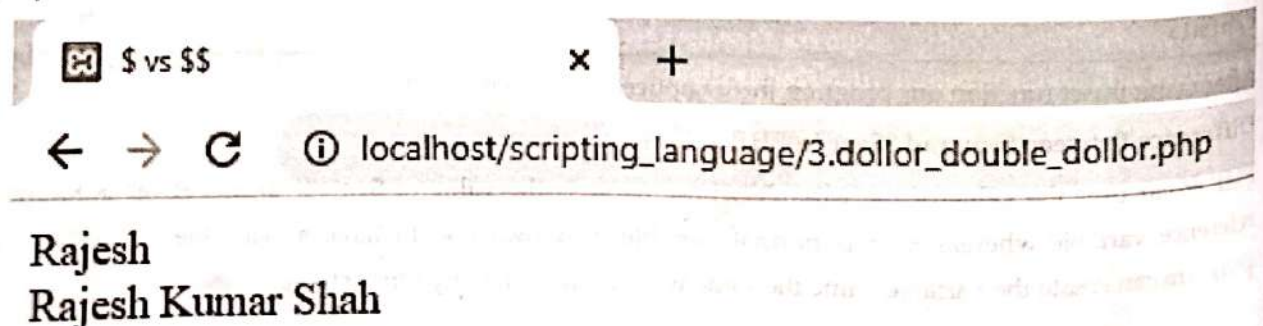
Variable \$name hold value = Rajesh.

\$\$name(reference variable) hold value = Rajesh Kumar Shah. Now we want to print the value.

echo \$name gives output:Rajesh

echo \$\$name gives output:Rajesh Kumar Shah.

Output



Super Global Variable in PHP

PHP super global variable is used to access global variables from anywhere in the PHP script. PHP super global variables are accessible inside the same page that defines it, as well as outside the page. While local variable's scope is within the page that defines it. But there is no guarantee that each web server recognizes every super global.

The PHP super global variables are:

- `$GLOBALS`
- `$_SERVER`
- `$_REQUEST`
- `$_POST`
- `$_GET`
- `$_FILES`
- `$_ENV`
- `$_COOKIE`
- `$_SESSION`

`$GLOBALS` Variable

`$GLOBAL` is a PHP super global variable which can be used instead of 'global' keyword to access variables from global scope, i.e. the variables which can be accessed from anywhere in a PHP script even within functions or methods.

PHP stores all global variables in an array called `$GLOBALS [index]`. The index holds the name of the variable.

```
<?php
$a = 75;
$b = 25;
function sum() {
    $GLOBALS['c'] = $GLOBALS['a'] + $GLOBALS['b'];
}
sum();
echo $c;
?>
```

`$_SERVER` Variable

`$_SERVER` is one of the PHP global variables - termed super global, which contain information about server and execution environments. These are pre-defined variables so they are always accessible from any class, function or file.

`$_SERVER` is a super global variable containing information such as headers, paths, and script locations. The entries here are recognized by web servers, but there is no guarantee that each web server recognizes every super global. These three PHP `$_SERVER` arrays all behave in similar way they return information about the file in use. When exposed to different scenarios, in some cases they behave differently. These examples may help you decide which is best for what you need.

List of \$_SERVER arrays is available at the PHP website.

Element/Code	Description
\$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
\$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
\$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
\$_SERVER['SERVER_NAME']	Returns the name of the host server
\$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache 1.6.27)
\$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
\$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
\$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
\$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
\$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
\$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept Charset header from the current request (such as utf-8,ISO-8859-1)
\$_SERVER['HTTP_HOST']	Returns the Host header from the current request
\$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
\$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
\$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
\$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
\$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
\$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
\$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as user@domain.com)
\$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
\$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
\$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
\$_SERVER['SCRIPT_NAME']	Returns the path of the current script
\$_SERVER['SCRIPT_URI']	Returns the URI of the current page


```
<?php
foreach($_SERVER as $key_name => $key_value) {
    print $key_name . " = " . $key_value . "<br>";
}
?>
```

\$_REQUEST Variable

\$_REQUEST can be used to collect data with both post and get method. \$_REQUEST contains values passed by post, get and/or cookies. As get is easy to hack so safer mechanism would be to use post when send data from one html/php file to another. Then you need to use \$_POST to get the data.

```
<html>
<head>
<title>$_REQUEST Example</title>
</head>
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>"
    Username: <input type="text" name="username">
    <input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $username = $_REQUEST['username'];
    if (empty($username)) {
        echo "Username is empty";
    } else {
        echo $username;
    }
}
?>
</body>
</html>
```

\$_POST Variable

\$_POST is a super global variable which is widely used to pass variables. This super global variable is widely used to handle form data.

```

<html>
<head>
<title>$_POST Example</title>
</head>
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Username: <input type="text" name="username">
  <input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  // collect value of input field
  $username = $_POST['username'];
  if (empty($username)) {
    echo "Username is empty";
  } else {
    echo $username;
  }
}
?>
</body>
</html>

```

\$_GET Variable

\$_GET is a super global variable which can be used to do the same job done by POST. But besides, \$_GET can do some other wonderful jobs as far as passing data is concerned.

It is used to collect value from a form (HTML script) sent with method='get'. Information sent from a form with the method='get' is visible to everyone (it display on the browser URL bar).

```

<html>
<head>
<title>$_REQUEST Example</title>
</head>
<body>
<form method="get" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Username: <input type="text" name="username">
  <input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "GET") {
  // collect value of input field
  $username = $_GET['username'];
  if (empty($username)) {
    echo "Username is empty";
  } else {
    echo $username;
  }
}
?>
</body>
</html>

```


\$_FILES

\$_FILES is a super global variable which can be used to upload files. **\$_FILES** is a two dimensional associative global array of items which are being uploaded by via HTTP POST method and holds the attributes of files.

ATTRIBUTE	DESCRIPTION
name	Name of file which is uploading
size	Size of the file
type	Type of the file (like .pdf, .zip, .jpeg.....etc)
tmp_name	A temporary address where the file is located before processing the upload request
error	Types of error occurred when the file is uploading

\$_ENV

\$_ENV is used to return the environment variables from the web server. It's a super global variable ... an array of environment variables. That environment may be bash (or some other shell running the php-cli interpreter) or Apache (or some other server related environment). I've only used them with Apache, so my knowledge is a bit limited.

For example in Magento you would define ENV variables in .htaccess (or some other web server configuration file) and they would be imported in the global namespace of PHP as part of the super global variable mentioned.

\$_COOKIE

Cookies are small text files loaded from a server to a client computer storing some information regarding the client computer, so that when the same page from the server is visited by the user, necessary information can be collected from the cookie itself, decreasing the latency to open the page.

\$_SESSION

Sessions are wonderful ways to pass variables. All you need to do is start a session by `session_start()`; Then all the variables you store within a **\$_SESSION**, you can access it from anywhere on the server.

Some examples of super global variable:

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Super Global Example</title>
</head>
<body>
<h4>Example of $GLOBALS</h4>
<?php
$a = 566;
```

```

$b = 200;
function sum() {
    $GLOBALS['c'] = $GLOBALS['a'] + $GLOBALS['b'];
}
sum();
echo "Sum of 566 and 200 is : " . $c;
?>
<h4>Example of $_REQUEST of given URL:
http://localhost/scripting_language/4.superglobal.php?my_user=Ramji&&age=50</h4>
<pre>
<?php
print_r($_REQUEST);
?>
</pre>
<h4>Example of $_SESSION</h4>
<pre>
<?php
session_start();
$_SESSION['username'] = 'admin';
print_r($_SESSION);
?>
</pre>
<h4>Example of $_COOKIE</h4>
<pre>
<?php
setcookie('name','Scripting Language',time() + 7*24*60*60);
print_r($_COOKIE);
?>
</pre>
<h4>Example of $_SERVER</h4>
<pre>
<?php
print_r($_SERVER);
?>
</pre>
</body>
</html>

```


PHP's Supported Data Types

A datatype is the generic name assigned to any data sharing a common set of characteristics. Data Types defines the type of data a variable can store. PHP allows eight different types of data types. Common data types include Boolean, integer, float, string, and array. All of them are discussed below. The first five are called simple data types and the last three are compound data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- NULL
- Array
- Object
- Resource

String

Simply put, a string is a sequence of characters treated as a contiguous group. Strings are delimited by single or double quotes.

A string literal can be specified in four different ways:

- single quoted
- double quoted
- heredoc syntax
- nowdoc syntax (since PHP 5.3.0)

Single quoted

The simplest way to specify a string is to enclose it in single quotes (the character `'`). Variables and escape sequences for special characters will not be expanded when they occur in single quoted strings.

To specify a literal single quote, escape it with a backslash (`\`). To specify a literal backslash, double it (`\\`). All other instances of backslash will be treated as a literal backslash: this means that the other escape sequences you might be used to, such as `\r` or `\n`, will be output literally as specified rather than having any special meaning.

The following are all examples of valid strings:

- `'PHP is a most popular scripting language'`
- `'Scripting-Language'`
- `'*9subway\n'`
- `'123$%^789'`
- `'I\'ll be back';`
- `'Variables do not $expand $either';`

Double quoted:

If the string is enclosed in double-quotes ("), PHP will interpret the following escape sequences for special characters:

Escaped Character:

Sequence	Meaning
\n	linefeed (LF or 0x0A (10) in ASCII)
\r	carriage return (CR or 0x0D (13) in ASCII)
\t	horizontal tab (HT or 0x09 (9) in ASCII)
\v	vertical tab (VT or 0x0B (11) in ASCII) (since PHP 5.2.5)
\e	escape (ESC or 0x1B (27) in ASCII) (since PHP 5.4.4)
\f	form feed (FF or 0x0C (12) in ASCII) (since PHP 5.2.5)
\\	backslash
\\$	dollar sign
\"	double-quote
\[0-7]{1,3}	the sequence of characters matching the regular expression is a character in octal notation, which silently overflows to fit in a byte (e.g. "\400" === "\000")
\x[0-9A-Fa-f]{1,2}	the sequence of characters matching the regular expression is a character in hexadecimal notation
\u[0-9A-Fa-f]{1,6}	the sequence of characters matching the regular expression is a Unicode codepoint, which will be output to the string as that codepoint's UTF-8 representation (added in PHP 7.0.0)

As in single quoted strings, escaping any other character will result in the backslash being printed too.

Heredoc

A third way to delimit strings is the heredoc syntax: <<<. After this operator, an identifier is provided, then a newline. The string itself follows, and then the same identifier again to close the quotation.

```
<?php
echo <<<"MY"
Hello World!
MY;
?>
```

Nowdoc

Nowdocs are to single-quoted strings what heredocs are to double-quoted strings. A nowdoc is specified similarly to a heredoc, but no parsing is done inside a nowdoc. The construct is ideal for embedding PHP code or other large blocks of text without the need for escaping.

```
<?php
echo <<<'EOD'
Example of string spanning multiple lines
using nowdoc syntax. Backslashes are always treated literally,
e.g. \\ and \'.
EOD;
?>
```

For normal string value, you could retrieve a particular character of the string by treating the string as an array. For example

```
$address = "Kathmandu";
$newValue = $address [2]; // Assigns to $newValue
```

Integer

An integer is representative of any whole number or, in other words, a number that does not contain fractional parts. PHP supports integer values represented in base 10 (decimal), base 8 (octal), and base 16 (hexadecimal) numbering systems. To use octal notation, precede the number with a 0 (zero). To use hexadecimal notation precede the number with 0x. To use binary notation precede the number with 0b. As of PHP 7.4.0, integer literals may contain underscores (_) between digits, for better readability of literals. These underscores are removed by PHP's scanner.

Several examples follow:

```
$a = 1234; // decimal number
$a = 0123; // octal number (equivalent to 83 decimal)
$a = 0x1A; // hexadecimal number (equivalent to 26 decimal)
$a = 0b11111111; // binary number (equivalent to 255 decimal)
$a = 1_234_567; // decimal number (as of PHP 7.4.0)
```

The size of an integer is platform-dependent, although a maximum value of about two billion is the usual value (that's 32 bits signed). PHP 6 introduced a 64-bit integer value, meaning PHP will support integer values up to positive or negative 2^{63} in size. 64-bit platforms usually have a maximum value of about 9E18, except on Windows prior to PHP 7, where it was always 32 bit. PHP does not support unsigned integers. Integer size can be determined using the constant `PHP_INT_SIZE`, maximum value using the constant `PHP_INT_MAX` since PHP 5.0.5, and minimum value using the constant `PHP_INT_MIN` since PHP 7.0.0. If PHP encounters a number beyond the bounds of the integer type, it will be interpreted as a float instead.

Float

Floating-point numbers, also referred to as floats, doubles, or real numbers, allow you to specify numbers that contain fractional parts. Floats are used to represent monetary values, weights, distances, and a whole host of other representations in which a simple integer value won't suffice.

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
$d = 1_234.567; // as of PHP 7.4.0
?>
```

Floating point numbers have limited precision. Although it depends on the system, PHP typically uses the IEEE 754 double precision format, which will give a maximum relative error due to rounding in the order of $1.11e-16$. Non elementary arithmetic operations may give larger errors, and, of course, error propagation must be considered when several operations are compounded.

Boolean:

This is the simplest type. A Boolean expresses a truth value. It can be either TRUE or FALSE. To specify a Boolean literal, use the constants TRUE or FALSE. Both are case-insensitive.

```
<?php
$martialStatus = True; // assign the value TRUE to $martialStatus
?>
```

NULL

The special NULL value represents a variable with no value. NULL is the only possible value of type null.

A variable is considered to be null if:

- It has been assigned the constant NULL.
- It has not been set to any value yet.
- It has been unset().

There is only one value of type null, and that is the case-insensitive constant NULL.

```
<?php
$var = NULL;
?>
```

Compound Data Types

Compound data types allow for multiple items of the same type to be aggregated under a single representative entity. The array and the object fall into this category.

Array

It's often useful to aggregate a series of multiple items together, arranging and referencing them in some specific way. This data structure, known as an array, is formally defined as an indexed collection of data values. Each member of the array index (also known as the key) references a corresponding value and can be a simple numerical reference to the value's position in the series, or it could have some direct correlation to the value.

For example, if you were interested in creating a list of Nepal's province, you could use a numerically:

Indexed array can be represent as follow:

```
$province[0] = "Province 1";
```

```
$province[1] = "Province 2";
```

```
$province[2] = "Bagmati";
```

```
$province[3] = "Gandaki";
```

```
$province[4] = "Province 5";
```

```
$province[5] = "Karnali";
```

```
$province[6] = "Sudur Paschim";
```

But what if the project required correlating Nepal's province to their capitals? Rather than base the keys on a numerical index, you might instead use an associative index, like this:

```

$province["province_1"] = "Biratnagar";
$province["province_2"] = "Janakpur";
$province["bagmati"] = "Hetauda";
$province["gandaki"] = "Pokhara";
$province["province_5"] = "Butwal";
$province["karnali"] = "Birendranagar";
$province["sudur_paschim"] = "Godawari";

```

Object

The other compound datatype supported by PHP is the object. The object is a central concept of the object-oriented programming paradigm. Unlike the other data types contained in the PHP language, an object must be explicitly declared. This declaration of an object's characteristics and behavior takes place within something called a class.

Here's a general example of a class definition and subsequent invocation:

```

class Student {
    private $name;
    function setName($name) {
        $this->name = $name;
    }
}
$ram = new Student;

```

A class definition creates several attributes and functions pertinent to a data structure, in this case a data structure named Appliance. There is only one attribute, power, which can be modified by using the method setName().

Remember, however, that a class definition is a template and cannot itself be manipulated. Instead, objects are created based on this template. This is accomplished via the new keyword. Therefore, in the last line of the previous listing, an object of class Student named ram is created.

The ram object's name attribute can then be set by making use of the method setName():

```
$ram->setName("Ram Bahadur Tamang");
```

Resource:

But, unlike other data types, it is acting as a reference or an identifier to access resource data. For example, when we attempt to access the database or a file resource, we can have a resource identifier. This identifier will be used to hook the resource to access data.

The following code shows how to get the database resource identifier by requesting the MySQL connection. In this code, the database information is specified for the mysqli_connect() and it returns MySQL connection object as a resource identifier.

```
$conn = mysqli_connect(localhost,"root","","db_scripting_language");
```


While working with files, we need to get the file resource identifier. With the reference to this identifier, we can perform the write, append and more file manipulation operations. The following PHP code is used to create a file resource object reference.

```
$fp = fopen("index.php", 'r');
```

After finishing all the operations with the resource data, the connection or the hook we created with the external resources will be cleared. The following PHP functions are used to clear the resource object reference created for the database and the file resource.

```
mysqli_close($conn);
```

```
fclose($fp);
```

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Variable Type</title>
</head>
<body>
<pre>
<table border="1">
<tr>
<td>
<h3>1. Array</h3>
<h5>Numeric Array</h5>
<?php
    $province[0] = "Province 1";
    $province[1] = "Province 2";
    $province[2] = "Bagmati";
    $province[3] = "Gandaki";
    $province[4] = "Province 5";
    $province[5] = "Karnali";
    $province[6] = "Sudur Paschim";
    var_dump($province);
?>
</td>
<td>
<h5>Associative Array</h5>
<?php
    $province["province_1"] = "Biratnagar";
    $province["province_2"] = "Janakpur";
    $province["bagmati"] = "Hetauda";
    $province["gandaki"] = "Pokhara";
    $province["province_5"] = "Butwal";
    $province["karnali"] = "Birendranagar";
    $province["sudur_paschim"] = "Godawari";
    var_dump($province);
?>
```

```

</td>
<td>
<h3>2. Class</h3>
<?php
    class Student {
        private $name;
        function setName($name) {
            $this->name = $name;
        }
    }
    $ram = new Student;
    $ram->setName('Rajesh');
    print_r($ram);
?>
<h3>3. Resource </h3>
<?php
    $fp = fopen("4.superglobal.php", 'r');
    print_r($fp);
?>
<h3>4. Null</h3>
<?php
    $var = NULL;
    var_dump($var);
?>
<h3>5. Float type</h3>
<?php
    $a = 1.234;
    $b = 1.2e3;
    $c = 7E-10;
    var_dump($a);
    var_dump($b);
    var_dump($c);
?>
</td>
<td>
<h3>6. Integer</h3>
<?php
    $a = 1234; // decimal number
    $b = 0123; // octal number (equivalent to 83 decimal)
    $c = 0x1A; // hexadecimal number (equivalent to 26 decimal)
    $d = 0b11111111; // binary number (equivalent to 255 decimal)
    var_dump($a);
    var_dump($b);
    var_dump($c);
    var_dump($d);
?>
<h3>7. String </h3>

```



```

<?php
    echo 'PHP is a most popular scripting language';
    echo 'Scripting-Language';
    echo "9subway\n";
    echo '123$%^789';
    echo 'I\'ll be back';
    echo 'Variables do not $expand $either';

?>
<h3>8. Boolean</h3>
<?php
    $maritalStatus = True; // assign the value TRUE to $maritalStatus
    var_dump($maritalStatus);
?>
</td>
</tr>
</table>
</pre>
</body>
</html>

```

Output

1. Array <pre> array(7) { [0] => string(10) "Province 1" [1] => string(10) "Province 2" [2] => string(7) "Bagnati" [3] => string(7) "Gandaki" [4] => string(10) "Province 5" [5] => string(7) "Karnali" [6] => string(13) "Sudur Paschim" } </pre>	Associative Array <pre> array(14) { [0] => string(10) "Province 1" [1] => string(10) "Province 2" [2] => string(7) "Bagnati" [3] => string(7) "Gandaki" [4] => string(10) "Province 5" [5] => string(7) "Karnali" [6] => string(13) "Sudur Paschim" ["province_1"] => string(10) "Biratnagar" ["province_2"] => string(8) "Janakpur" ["bagnati"] => string(7) "Metsuda" ["gandaki"] => string(7) "Pokhara" ["province_5"] => string(6) "Butwal" ["karnali"] => string(13) "Birendranagar" ["sudur_paschim"] => string(8) "Godawari" } </pre>	2. Class <pre> Student Object ([name:Student:private] => Rajesh) </pre> 3. Resource <pre> Resource id #3 </pre> 4. Null <pre> NULL </pre> 5. Float type <pre> float(1.234) float(1200) float(7.0E-10) </pre>	6. Integer <pre> int(1234) int(83) int(26) int(255) </pre> 7. String <pre> PHP is a most popular scripting languageScripting-Language9subway 123\$%^789I'll be backVariables do not \$expand \$either </pre> 8. Boolean <pre> bool(true) </pre>
--	---	---	--

PHP Constant

Define constant- A constant is a variable whose value cannot be changed at runtime. Suppose we are developing a program that uses the value of PI 3.14, we can use a constant to store its value.

Let's now look at an example that defines a constant. `define('PI',3.14);` //creates a constant with a value of 3.14 Once you define PI as 3.14 , writing a code like below will generate an error `PI = 4;` //PI has been defined as a constant therefore assigning a value is not permissible.

Name of constants must follow the same rules as variable names, which means a valid constant name must start with a letter or underscore, followed by any number of letters, numbers or underscores with one exception: the \$ prefix is not required for constant names.

An E_WARNING level error is generated if the constant is not defined.

Example

```
<?php
// Defining constant
define("SITE_URL", "https://127.0.0.1");
// Using constant
echo 'Thank you for visiting - ' . SITE_URL;
?>
```

Output

Constant Example

localhost/scripting_language/6.constant.php

Constant Example

Thank you for visiting - http://127.0.0.1

Converting Between Data Types Using Type Casting

Converting values from one datatype to another is known as type casting. A variable can be evaluated once as a different type by casting it to another. This is accomplished by placing the intended type in front of the variable to be cast.

Cast Operators => Conversion

- (array) => Array
- (bool) or (boolean) => Boolean
- (int) or (integer) => Integer
- (object) => Object
- (real) or (double) or (float) => Float
- (string) => String

Cast an integer as a double:

```
$weight = (double) 13;
var_dump($weight);
```


Cast double to an integer

Type casting a double to an integer will result in the integer value being rounded down, regardless of the decimal value.

```
$weight = (int) 14.8;  
var_dump($weight);
```

Cast string datatype to integer;

```
$sentence = "This is test string";  
echo (int) $sentence;
```

Simple datatype to Array type:

You can also cast a datatype to be a member of an array. The value being cast simply becomes the first element of the array:

```
$age = 112;  
$ageList = (array) $age;  
echo $ageList[0]; // Outputs 112
```

Other type to object type

Any data type can be cast as an object. The result is that the variable becomes an attribute of the object, the attribute having the name scalar:

```
$gender = "male";  
$obj = (object) $gender;  
var_dump($obj);
```

The value can then be referenced as follows:

```
print $obj->scalar; // returns "male"
```

Example

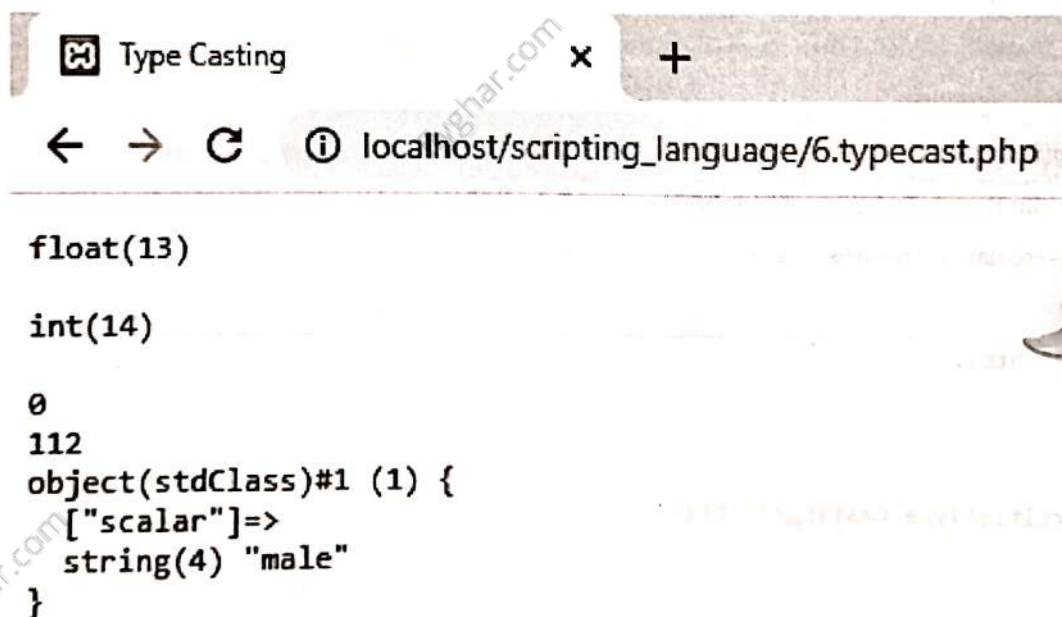
```
<!DOCTYPE html>  
<html>  
<head>  
    <title>Type Casting</title>  
</head>  
<body>  
    <pre>  
        <?php  
            $weight = (double) 13;  
            var_dump($weight);  
            echo '<br>';  
            $weight = (int) 14.8;  
            var_dump($weight);
```

```

echo '<br>';
$sentence = "This is test string";
echo (int) $sentence;
echo '<br>';
$age = 112;
$ageList = (array) $age;
echo $ageList[0];
echo '<br>';
$gender = "male";
$obj = (object) $gender;
var_dump($obj);
?>
</pre>
</body>
</html>

```

Output



PHP OPERATORS

An operator is a symbol that specifies a particular action in an expression. Many operators may be familiar to you. Regardless, you should remember that PHP's automatic type conversion will convert types based on the type of operator placed between the two operands, which is not always the case in other programming languages. The precedence and associativity of operators are significant characteristics of a programming language.

Operator associativity

In programming languages, the associativity of an operator is a property that determines how operators of the same precedence are grouped in the absence of parentheses.

Operator	Associativity	Purpose
new	NA	Object instantiation
()	NA	Expression subgrouping
[]	Right	Index enclosure
! -- ++ --	Right	Boolean Not, Bitwise Not, Increment, Decrement
@	Right	Error Suppression
/ * %	Left	Division, Multiplication, Modulus
+ - .	Left	Addition, Subtraction, Concatenation
<< >>	Left	Shift left, Shift right (bitwise)
<<= >>=	NA	Less than, less than or equal to, greater than, greater than or equal to
== != === <>	NA	Is equal to, is not equal to, is identical to, is not equal to
& ^	Left	Bitwise AND, bitwise XOR, bitwise OR
&&	Left	Boolean AND, Boolean OR
?:	Right	Ternary operator
= += *= /= .= %= &= = ^= <<= >>=	Right	Assignment operators
AND XOR OR	Left	Boolean AND, Boolean XOR, Boolean OR
.	Left	Expression separation

The associativity characteristic of an operator specifies how operations of the same precedence are evaluated as they are executed. Associativity can be performed in two directions, left-to-right or right-to-left. Left-to-right associativity means that the various operations making up the expression are evaluated from left to right.

Consider the following example:

```
$value = 3 * 4 * 5 * 7 * 2;
```

The preceding example is the same as the following:

```
$value = (((3 * 4) * 5) * 7) * 2;
```

Operator Precedence

Operator Precedence Operator precedence is a characteristic of operators that determines the order in which they evaluate the operands surrounding them. PHP follows the standard precedence rules used in elementary school math class.

Consider a few examples:

```
$total_cost = $cost + $cost * 0.06;
```

This is the same as writing

```
$total_cost = $cost + ($cost * 0.06);
```

Because the multiplication operator has higher precedence than the addition operator.

PHP Operators can be categorized in following forms:

- Arithmetic Operators
- Comparison Operators
- Bitwise Operators
- Logical Operators

- String Operators
- Incrementing/Decrementing Operators
- Array Operators
- Type Operators
- Execution Operators
- Error Control Operators
- Assignment Operators

We can also categorize operators on behalf of operands. They can be categorized in 3 forms:

- Unary Operators: works on single operands such as ++, -- etc.
- Binary Operators: works on two operands such as binary +, -, *, / etc.
- Ternary Operators: works on three operands such as "?:".

Arithmetic Operators

There are following arithmetic operators supported by PHP language.

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Assignment Operators

There are following assignment operators supported by PHP language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

String Operators

PHP's string operators provide a convenient way in which to concatenate strings together.

There are two such operators, including the concatenation operator (.) and the concatenation assignment operator (.=).

Operator	Description	Example
.	Concatenation	<code>\$a = "abc"."def";</code> \$a is assigned the string "abcdef"
<code>.=</code>	Concatenation-assignment	<code>\$a .= "ghijkl";</code> \$a equals its current value concatenated with "ghijkl"

Logical Operators

There are following logical operators supported by PHP language.

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
And	Called Logical AND operator. If both the operands are true then condition becomes true.	(A and B) is true.
Or	Called Logical OR Operator. If any of the two operands are non-zero then condition becomes true.	(A or B) is true.
<code>&&</code>	Called Logical AND operator. If both the operands are non-zero then condition becomes true.	(A <code>&&</code> B) is true
<code> </code>	Called Logical OR Operator. If any of the two operands are non-zero then condition becomes true.	(A <code> </code> B) is true
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A <code>&&</code> B) is false.

Comparison Operators

There are following comparison operators supported by PHP language.

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
<code>==</code>	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A <code>==</code> B) is not true.
<code>!=</code>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A <code>!=</code> B) is true.s
<code>></code>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A <code>></code> B) is not true.
<code><</code>	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A <code><</code> B) is true.
<code>>=</code>	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A <code>>=</code> B) is not true.
<code><=</code>	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <code><=</code> B) is true.
<code>===</code>	Checks if the value of two operands are equal and or not, also datatype of given operand are equal if yes then condition becomes true.	(A <code>===</code> B) is not true.

Ternary operator

The ternary operator is a conditional operator that decreases the length of code while performing comparisons and conditionals. This method is an alternative for using if-else and nested if-else statements. The order of execution for this operator is from left to right. Obviously, it is the best case for a time-saving option.

It is called a ternary operator because it takes three operands – a condition, a result for true, and a result for false.

Syntax:

(Condition) ? (Statement1) : (Statement2);

- Condition: It is the expression to be evaluated which returns a boolean value.
- Statement 1: it is the statement to be executed if the condition results in a true state.
- Statement 2: It is the statement to be executed if the condition results in a false state.

```
<?php
$marks=90;
echo ($marks>=35) ? "pass" : "Fail";
?>
```

Control Structures (if else, switch, all loops)

Code execution can be grouped into following categories:

Sequential – this one involves executing all the codes in the order in which they have been written.

Decision – this one involves making a choice given a number of options. The code executed depends on the value of the condition.

Control structures determine the flow of code within an application, defining execution characteristics such as whether and how many times a particular code statement will execute, as well as when a code block will relinquish execution control.

Conditional Statements Conditional statements make it possible for your program to respond accordingly to a wide variety of inputs, using logic to discern between various conditions based on input value.

You can use conditional statements in your code to make your decisions.

PHP supports following three decision making statements –

PHP if statement

PHP if statement is executed if condition is true. Use this statement if you want to execute a set of code when a condition is only true.

Syntax:

```
if(condition){
//code to be executed
}
```

Example:

```
<?php
$num = 50;
if($num>0)
{
echo $num." is positive number";
}
?>
```


PHP If-else Statement

PHP if-else statement is executed whether condition is true or false. Use this statement if you want to execute a set of code when a condition is true and another if the condition is not true.

Syntax:

```
if(condition){
//code to be executed if true
}
else{
//code to be executed if false
}
```

Example:

```
<?php
$num = 55;
if($num>0)
{
echo $num." is positive number";
}
else
{
echo $num." is negative number";
}
?>
```

PHP If-else if else Statement

PHP if-else if else statement is executed with more than two condition. Used with the if...else statement to execute a set of code if one of the several condition is true

Syntax:

```
if(condition1){
//code to be executed if true
}
else if (condition2){
//code to be executed if true
}
.....
.....
else if (condition n-1){
//code to be executed if true
} else {
//code to be executed if false
}
```

Example:

```
<?php
$char = "a";
if($char=="a")
{
echo $char." is vowel";
}
else if($char=="e")
{
echo $char." is vowel";
}
else if($char=="i")
{
echo $char." is vowel";
}
else if($char=="o")
{
echo $char." is vowel";
}
else if($char=="u")
{
echo $char." is vowel";
}
else
{
echo $char." is consonent";
}
?>
```

Switch Statement

Used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if .. else if .. else code. The switch statement first it evaluates given expression then seeks a label to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the label matches then statement will execute any specified default code.

```
switch (expression){
case label1:
    code to be executed if expression = label1;
    break;
case label2:
    code to be executed if expression = label2;
    break;
.....
.....
default:
    code to be executed if expression is different from both label1 and label2;
}
```


Example:

```
<?php
switch ($i)
{
case 0:
    echo "$i equals 0 ";
    break;
case 1:
    echo "$i equals 1 ";
    break;
case 2:
    echo "$i equals 2 ";
    break;
}
?>
```

LOOPING STATEMENTS

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

for – loops through a block of code a specified number of times.

while – loops through a block of code if and as long as a specified condition is true.

do...while – loops through a block of code once, and then repeats the loop as long as a special condition is true.

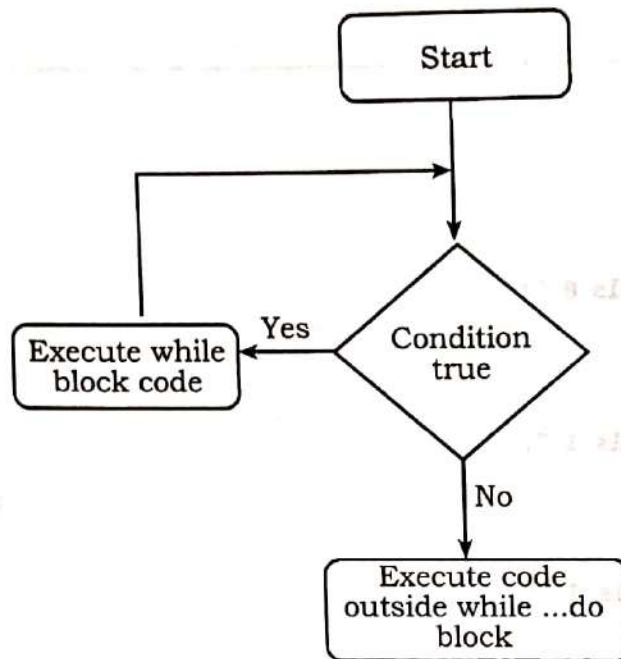
foreach – loops through a block of code for each element in an array.

For loop

For loop is used when you know how many times you want to execute a statement or a block of statements.

```
for (initialization; condition; increment){
    //code to be executed;
}
```

- "for...{...}" is the loop block
- "Initialize" usually an integer; it is used to set the counter's initial value.
- "Condition" the condition that is evaluated for each php execution. If it evaluates to true then execution of the for... loop is terminated. If it evaluates to false, the execution of the for... loop continues.
- "Increment" is used to increment the initial value of counter integer.



```

<?php
$number = 16;
for ($i = 1; $i <= 10; $i++){
    $product = $number * $i;
    echo "The product of $number * $i is $product <br/>";
}
?>

```

PHP While loop

They are used to execute a block of code repeatedly until the set condition gets satisfied. While loops are used to execute a block of code until a certain condition becomes true. While loop checks the condition first. If it evaluates to true, the block of code is executed as long as the condition is true. If it evaluates to false, the execution of the while loop is terminated. You can use a while loop to read records returned from a database query.

Example

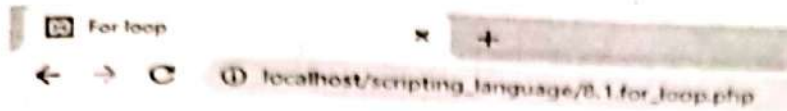
```

<?php
while (condition){
    block of code to be executed;
}
?>

```

- "while(...){...}" is the while loop block code
- "condition" is the condition to be evaluated by the while loop
- "block of code..." is the code to be executed if the condition gets satisfied

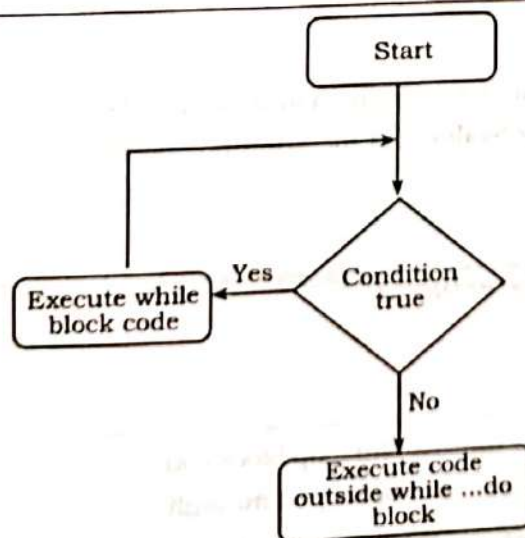
Output



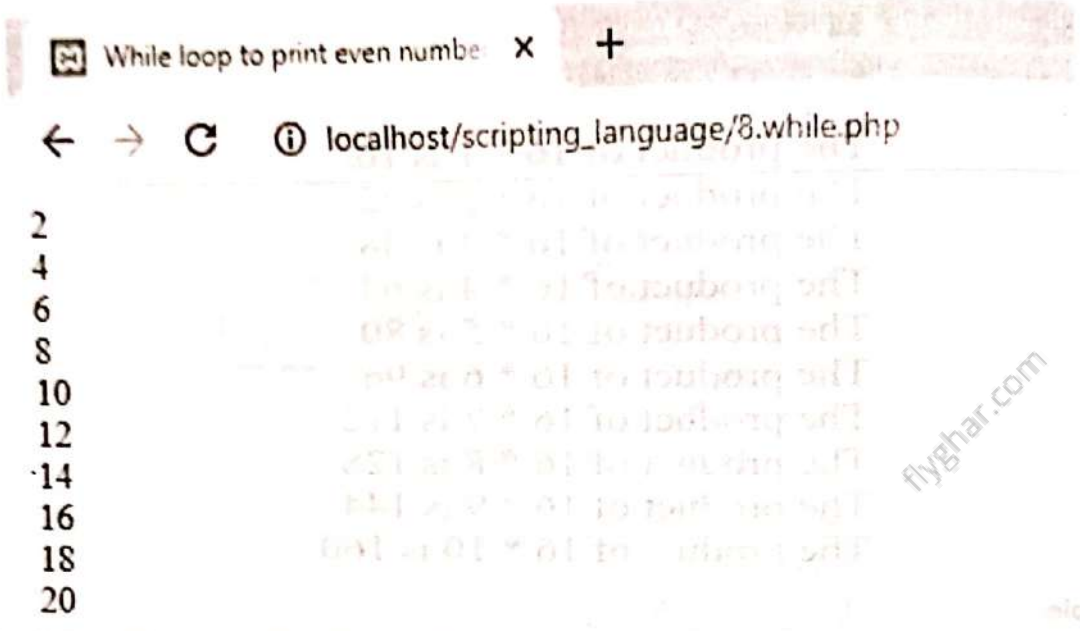
The product of 16 * 1 is 16
The product of 16 * 2 is 32
The product of 16 * 3 is 48
The product of 16 * 4 is 64
The product of 16 * 5 is 80
The product of 16 * 6 is 96
The product of 16 * 7 is 112
The product of 16 * 8 is 128
The product of 16 * 9 is 144
The product of 16 * 10 is 160

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>While loop to print even number upto 20</title>
</head>
<body>
<?php
$i = 2;
while ($i <= 20){
    echo $i . "<br>";
    $i=$i+2;
}
?>
</body>
</html>
```



Output



localhost/scripting_language/8.while.php

2
4
6
8
10
12
14
16
18
20

PHP Do While

The difference between While... loop and Do... while loop is do... while is executed at-least once before the condition is evaluated.

Syntax:

```
<?php
do{
    block of code to be executed
}while(condition);
?>
```

- "do{...} while(...)" is the do... while loop block code
- "condition" is the condition to be evaluated by the while loop
- "block of code..." is the code that is executed at least once by the do... while loop

PHP for-each loop

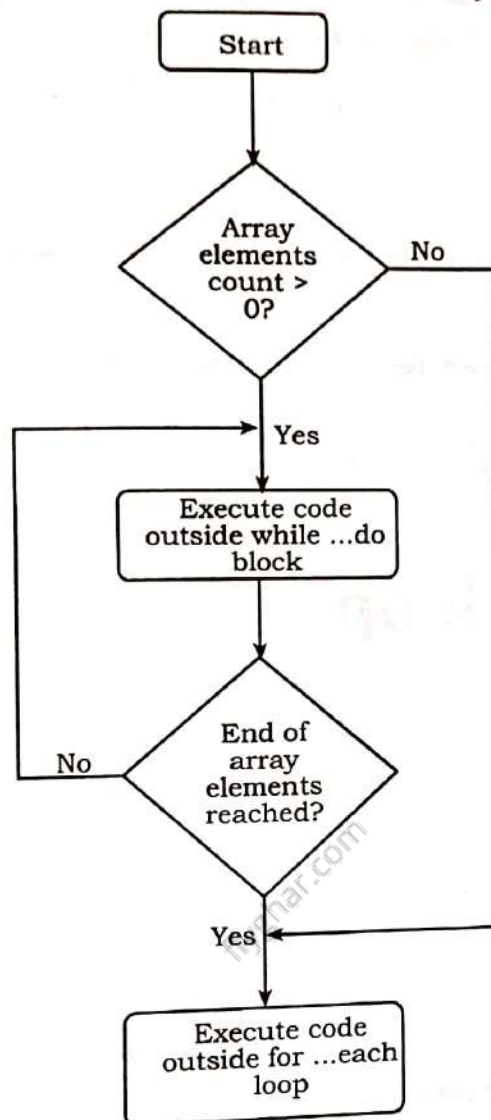
The php for each loop is used to iterate through array values. For each pass the value of the current array element is assigned to \$key => \$value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

```
foreach ($array as $key => value) {
    code to be executed;
}
```

- "foreach(...){...}" is the foreach php loop block code
- "\$array" is the array variable to be looped through
- "\$key" is the temporary variable that holds the current array item key.

- "\$value" is the temporary variable that holds the current array item key.
- "block of code..." is the piece of code that operates on the array values



Example

```

<!DOCTYPE html>
<html>
<head>
    <title>Foreach loop example</title>
</head>
<body>
<h1>Foreach loop</h1>
<?php
$fruits = array("Apple", "Mango", "Banana", "Papaya", "Watermelon", "Grape");
foreach($fruits as $fruit){
    echo $fruit . "<br>";
}
?>
<br>
<?php
  
```

```

$student = array("name" => "Ramesh", "address" => "Kalinchowk", "education" => "BCA");
foreach($student as $key => $value){
    echo ucfirst($key) . " is " . $value . "<br>";
}
?>
</body>

</html>

```

Output

Foreach loop example

x

+

localhost/scripting_language/8.3.foreach.php

Foreach loop

Apple
 Mango
 Banana
 Papaya
 Watermelon
 Grape

Name is Ramesh
 Address is Kalinchowk
 Education is BCA

ARRAYS

A PHP array is a variable that stores more than one piece of related data in a single variable. Array is one of the most vital part in every programming languages. It is also important to know what facilities and built-in functionality a programming language provides. PHP provides a very strong set of such efficient ways to deal with array data type. Think of an array as a box of noodle with slots inside.

The box represents the array itself while the spaces containing noodle packet represent the values stored in the arrays.

Types of PHP Arrays:

- Numeric index array
- Associative index array
- Multidimensional array

Numeric index array

Numeric arrays use number as access keys. An access key is a reference to a memory slot in an array variable. The access key is used whenever we want to read or assign a new value an array element. Arrays with sequential numeric index, such as 0,1,2 etc.

Different way of creating numeric index array

```
$array_name[0] = value1;
```

```
$array_name[1] = value2;
```

```
$array_name[2] = value3;
```

```
$array_name[3] = value4;
```

Or

```
$array_name = array(value1,value2,value3,value4);
```

Or

```
$array_name = [value1,value2,value3,value4];
```

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Numneric Array</title>
</head>
<body>
<?php
$animals = array("Buffalo","Lion", "Tiger", "Wolf");
// loop through the array
for($i = 0; $i < count($animals); $i++) {
    echo $animals[$i];
    echo "<br>";
}
?>
</body>
</html>
```

Output

Numneric Array

localhost/scripting_language/9.numeric_array.php

Buffalo
Lion
Tiger
Wolf

Associative array

This is the most frequently used type of PHP arrays whose elements are defined in key/value pair. Associative array differ from numeric array in the sense that associative arrays use descriptive names for id keys. Associative array are also very useful when retrieving data from the database.

Different way of creating numeric index array.

```
$array_name["key1"] = value1;
```

```
$array_name["key2"] = value2;
```

```
$array_name["key3"] = value3;
```

```
$array_name["key4"] = value4;
```

Or

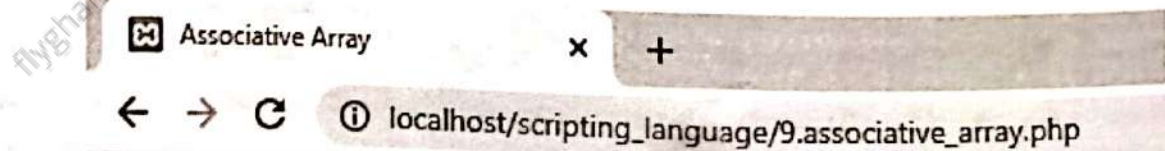
```
$array_name = array ("key1" =>value1, "key2" =>value2, "key3" =>value3, "key4" =>value4);
```

Or

```
$array_name = ["key1" => value1, "key2" => value2, "key3" => value3, "key4" => value4];
```

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Associative Array</title>
</head>
<body>
<?php
$animals = array("Lion"=>"Wild", "Sheep"=>"Domestic", "Tiger"=>"Wild", "Buffalo" =>
"Domestic");
// loop through associative array and get key value pairs
foreach($animals as $key => $value) {
    echo "Key=" . $key . ", Value=" . $value;
    echo "<br>";
}
?>
</body>
</html>
```

Output

```
Key=Lion, Value=Wild
Key=Sheep, Value=Domestic
Key=Tiger, Value=Wild
Key=Buffalo, Value=Domestic
```


PHP Multi-dimensional array

A multidimensional array is an array which stores another array at each index rather than storing a single value. In simple words, a multidimensional array is an array of arrays. These are arrays that contain other nested arrays. The advantage of multidimensional arrays is that they allow us to group related data together. There is no limit in the level of dimensions.

Different way of creating multidimensional array:

Example

```
$array = array(  
    array(value-1,value-2,.....,value-n),  
    array(value-1,value-2,.....,value-n),  
    array(value-1,value-2,.....,value-n)  
);
```

```
$array = [  
    [value-1,value-2,.....,value-n],  
    [value-1,value-2,.....,value-n],  
    [value-1,value-2,.....,value-n]  
];
```

```
$array = array(  
    key => array(  
        key => value-1,  
        key => value-2,  
        .....,  
        key => value-n  
    ),  
    key => array(  
        key => value-1,  
        key => value-2,  
        .....,  
        key => value-n  
    ),  
);
```

```
$array = [  
    key => [  
        key => value-1,  
        key => value-2,  
        .....,  
        key => value-n  
    ],  
    key => [  
        key => value-1,  
        key => value-2,  
        .....,  
        key => value-n  
    ],  
];
```

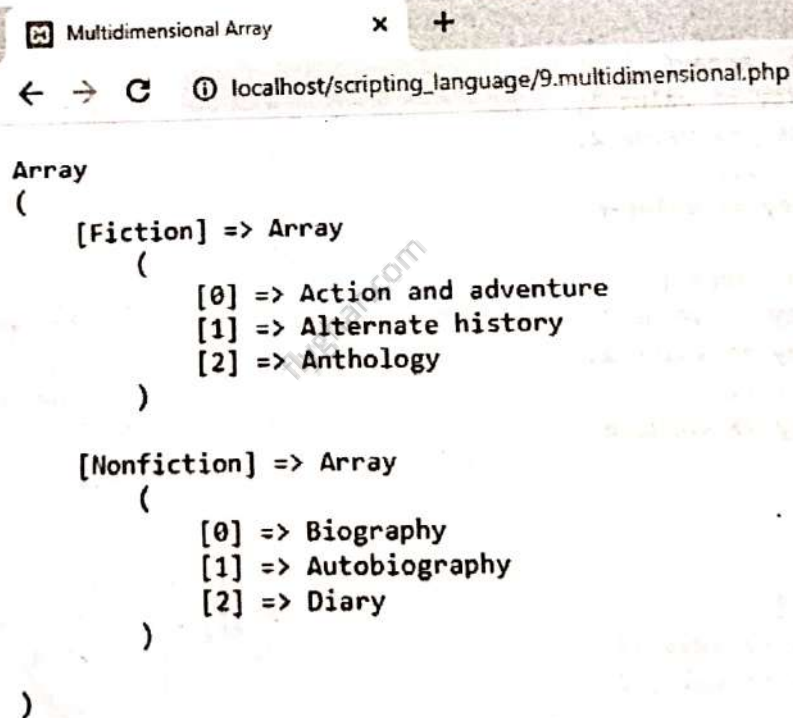
<!DOCTYPE html>

```

<html>
<head>
    <title>Multidimensional Array</title>
</head>
<body>
<?php
$movies =array(
    "Fiction" => array("Action and adventure", "Alternate history", "Anthology"),
    "Nonfiction" => array("Biography", "Autobiography", "Diary")
);
echo '<pre>';
print_r($movies);
echo '</pre>';
?>
</body>
</html>

```

Output



```

Array
(
    [Fiction] => Array
        (
            [0] => Action and adventure
            [1] => Alternate history
            [2] => Anthology
        )
    [Nonfiction] => Array
        (
            [0] => Biography
            [1] => Autobiography
            [2] => Diary
        )
)

```

FUNCTIONS

A Function is nothing but a 'block of statements' which generally performs a specific task and can be used repeatedly in our program. This 'block of statements' is also given a name so that whenever we want to use it in our program/script, we can call it by its name. Key benefit of using functions is that they are reusable; if you have a task that needs to be performed a number of times, a function is an ideal solution. A function can be either defined by user or by PHP (PHP has a rich collection of built-in functions). A function doesn't execute when it's defined, it executed when it is called.

The advantages of using functions are:

- Reducing duplication of code
- Decomposing complex problems into simpler pieces
- Improving clarity of the code
- Reuse of code
- Information hiding

The built-in functions are part of the PHP language. Examples are: `print_r()`, `var_dump()`, `phpinfo()`, `isset()` or `empty()`. The user defined functions are created by application programmers as per their needs. They are created with the function keyword.

User Defined Function

The actual syntax for creating a function is pretty self-explanatory. First, you must tell PHP that you want to create a function. You do this by typing the keyword `function` followed by your function name.

Naming rule of PHP function.

- Function names must start with a letter or an underscore but not a number
- The function name must be unique
- The function name must not contain spaces
- It is considered a good practice to use descriptive function names.
- Functions can optionally accept parameters and return values too.

Creating and Invoking Functions

The basic syntax of creating a user defined function can be given with:

```
<?php
function functionName(){
    // Code to be executed
}
?>
```

The declaration of a user-defined function start with the word `function`, followed by the name of the function you want to create followed by parentheses i.e. `()` and finally place your function's code between curly brackets `{}`.

```
<?php
// Defining function
function sayHello(){
    echo "Hello User, Welcome to our website!!!";
}
// Calling function
sayHello();
?>
```

Functions with Parameters

You can specify parameters when you define your function to accept input values at run time. The parameters work like placeholder variables within a function; they're replaced at run time by the values (known as argument) provided to the function at the time of invocation.

```
<?php
function functionName($parameter1, $parameter2){
    // Code to be executed
}
?>
```

You can define as many parameters as you like. However for each parameter you specify, a corresponding argument needs to be passed to the function when it is called.

The sayHello() function take single string value as argument and replace with "user" keyword inside function description.

```
<?php
// Defining function
function sayHello($user){
    echo "Hello $user, Welcome to our website!!!";
}
// Calling function
sayHello("Rajesh");
sayHello("Harish");
?>
```

Let's take one more example:

The sum() function in following example takes two integer values as arguments, simply add them together and then display the result in the browser.

```
<?php
// Defining function
function sum($num1, $num2){
    $sum = $num1 + $num2;
    echo "Sum of the two numbers $num1 and $num2 is : $sum";
}
// Calling function
sum(11, 31);
sum(90, 50);
?>
```

Functions with Optional Parameters and Default Values

You can also create functions with optional parameters for that, just insert the parameter name, followed by an equals (=) sign, followed by a default value.


```
<?php
// Defining function
function sum($num1, $num2=5){
    $sum = $num1 + $num2;
    echo "Sum of the two numbers $num1 and $num2 is : $sum";
}
// Calling function
getSum(10);
getSum(22,65);
?>
```

In above example we passed only single value at calling part so function will use value 5 in case of default value for num2. If we pass value instead of default value, function will take newly passed value.

Returning Values from a Function

A function can return a value back to the script that called the function using the return statement. The value may be of any type, including arrays and objects.

```
<?php
// Defining function
function sum($num1, $num2){
    $total = $num1 + $num2;
    return $total;
}
// Printing returned value
echo sum(5, 10);
$total = sum(65,4);
echo "Sum of 64 and 4 is $total";
?>
```

A function cannot return multiple values. However, you can obtain similar results by returning an array

```
<?php
// Defining function
function divideNumbers($dividend, $divisor){
    $quotient = $dividend / $divisor;
    $array = array($dividend, $divisor, $quotient);
    return $array;
}
// Assign variables as if they were an array
list($dividend, $divisor, $quotient) = divideNumbers(10, 2);
echo $dividend;
echo $divisor;
echo $quotient;
?>
```

PHP Function Overloading

Function overloading allows you to have multiple different variants of one function, differentiated by the number and type of arguments they take.

For example, we defined the function `sum()` which takes two arguments, and return the sum of those two. What if we wish to provide support for adding 3 numbers?

To tackle such situations, what we can do is, we can define two different variants of the function `sum()`, one which takes in 2 arguments and another which accepts 3 arguments. This is called Function Overloading. Unfortunately, PHP does not support Function Overloading. In PHP, function signatures are only based on their names and do not include the argument lists, hence we cannot have two functions with same name.

General example for the concept of function overloading in PHP.

```
<?php
// sum function with 2 arguments
function sum($a, $b)
{
    $sum = $a + $b;
    // returning the result
    return $sum;
}
// overloaded sum function with 3 arguments
function sum($a, $b, $c)
{
    $sum = $a + $b + $c;
    // returning the result
    return $sum;
}
?>
```

Extracting function argument using built-in PHP function:

func_num_args

Returns the number of arguments passed to the function

```
<?php
function getNumberOfArguments()
{
    $numargs = func_num_args();
    echo "Number of arguments: $numargs ";
}
getNumberOfArguments(61, 65, 38, 25, 98);
?>
```


func_get_arg

Gets the specified argument from a user defined function's argument list. Start with 0

```
<?php
function getSpecifiedParameter()
{
    $numargs = func_num_args();
    echo "Number of arguments: $numargs<br /> ";
    if ($numargs >= 2) {
        echo "Second argument is: " . func_get_arg(1) . "<br /> ";
    }
}
getSpecifiedParameter (14, 26, 73,45,56);
?>
```

func_get_args

Returns an array in which each element is a copy of the corresponding member of the current user-defined function's argument list. `func_get_args()` will generate a warning if called from outside of a function definition. This function cannot be used directly as a function parameter. Instead, its result may be assigned to a variable, which can then be passed to the function.

This function may be used in conjunction with `func_get_arg()` and `func_num_args()` to allow user-defined functions to accept variable-length argument lists.

```
<?php
function sumAllParam()
{
    $numargs = func_num_args();
    $arg_list = func_get_args();
    $sum = 0;
    for ($i = 0; $i < $numargs; $i++) {
        $sum += $arg_list[$i];
        echo "Argument $i is: " . $arg_list[$i] . "<br />\n";
    }
    echo "Final sum is " . $sum;
}
sumAllParam(11, 45, 56);
?>
```

PHP variable and function scope

Scope can be defined as the range of availability a variable has to the program in which it is declared. The scope of a variable in PHP is the context in which the variable was created, and in which it can be accessed.

PHP variables can be one of four scope types.

- Local variables
- Function parameters
- Global variables
- Static variables

Local Variables

A variable declared in a function is considered local. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function:

```
<?php
$marks = 40;

function assignMarks () {
    $x = 60;
    print "\$Marks inside function is $marks";
}
assignMarks();
print "\$marks outside of function is $marks";
?>
```

Output

\$marks inside function is 60.

\$ marks outside of function is 40.

Function Parameters

Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be:

```
<?php
// find reminder a value by divided by 3 and return it to the caller
function reminder ($value) {
    $value = $value % 3;
    return $value;
}
$retval = reminder (11);
Print "Return value is $retval\n";
?>
```

Output

Return value is 2

Global Variables

In contrast to local variables and function parameter, a global variable can be accessed in any part of the program. However, in order to be modified, a global variable must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword GLOBAL or global in front of the variable that should be recognized as global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name.

```
<?php
$first_num = 15;
function add() {
GLOBAL $first_num;
$first_num++;
print "Somevar is $first_num";
}
add();
$a = 1;
$b = 2;
function Sum()
{
    global $a, $b;
    $b = $a + $b;
}
Sum();
echo $b;
?>
```

The above script will output 3. By declaring \$a and \$b global within the function, all references to either variable will refer to the global version. There is no limit to the number of global variables that can be manipulated by a function. PHP provides a special set of predefined global arrays containing various useful nuggets of information. These arrays are known as superglobal because they're accessible from anywhere in your script, including inside functions, and you don't need to declare them as global using the global keyword.

We have already studied about super global variable available in PHP.

Static Variables

In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.

You can declare a variable to be static simply by placing the keyword **STATIC** in front of the variable name.

```
<?php
function keep_track() {
    STATIC $count = 0;
    $count++;
    print $count;
    print "<br/>";
}
keep_track();
keep_track();
keep_track();
?>
```

When you create a local variable inside a function, that variable only hangs around while the function is being run. As soon as the function exits, the local variable vanishes. When the function is called again, a new local variable is created. While static variables remember their values across function calls, they only last as long as the running script. Once the script exits, a static variable gets destroyed, just like local and global variables do.

Some PHP Array Functions

Merge multiple array into single one

The "array_merge" function provides the easiest way to merge a list of arrays. It can take as much parameters as you give to it and it will merge all of them into a single one and return back to you.

```
array_merge ( array $array1 [, array $array2 [, array $array3...]] );
```

```
<?php
$boys = array('Ram','Shyam','John');
$girls = array('Sita','Rita','Nita');
//now merge array of boys and girls into single array and call then student of single
class
$students = array_merge($boys,$girls);
print_r($students);
?>
```

Merge array recursively

Merges the elements of one or more arrays together so that the values of one are appended to the end of the previous one. If the input arrays have the same string keys, then the values for these keys are merged together into an array, and this is done recursively, so that if one of the values is an array itself, the function will merge it with a corresponding entry in another array too.

```
array_merge_recursive ( array $array1 [, array $array2...])
```



```
<?php
$array1=array("a"=>"Horse","b"=>"Cat","c"=>"Dog");
$array2=array("d"=>"Cow","a"=>"Cat","e"=>"elephant");
print_r(array_merge_recursive($array1,$array2));
?>
```

Sorting array

There are two ways to sort elements of array. If you want a simple default ascending order sorting with non-object array elements, then you can go for "sort" function. You just pass the array as parameter and it will sort the array and returns true if succeed.

```
sort($array [, $sort_flags] );
```

sort_flag => Optional. Specifies how to sort the array values. Possible values:

- SORT_REGULAR - Default. Treat values as they are (don't change types)
- SORT_NUMERIC - Treat values numerically
- SORT_STRING - Treat values as strings
- SORT_LOCALE_STRING - Treat values as strings, based on local settings

```
<?php
$students = array('Ram','Shyam','John','Sita','Rita','Nita');
sort($students);
?>
```

Reverse sort

Sorts the array in a descending order. The rsort() function returns true on success or false on failure.

```
rsort($array [, $sort_flags] );
```

\$array => array to sort

sort_flag => Optional. Specifies how to sort the array values. Possible values:

- SORT_REGULAR - Default. Treat values as they are (don't change types)
- SORT_NUMERIC - Treat values numerically
- SORT_STRING - Treat values as strings
- SORT_LOCALE_STRING - Treat values as strings, based on local settings

```
<?php
$students = array('Ram','Shyam','John','Sita','Rita','Nita');
rsort($students);
?>
```

Sort array with key ascending

The PHP ksort() sorts an associative array in ascending order, according to the key. It is mainly used for an associative array. This function was introduced in PHP 4.0.

```
bool ksort ( array &$array [, int $sort_flags = SORT_REGULAR ] );
```

array => It's compulsory parameter and specifies the array to use.

sort_flag It's a optional parameter and specifies how to sort the array values.

Possible values -

SORT_REGULAR - Default. Treat values as they are (don't change types)

SORT_NUMERIC - Treat values numerically

SORT_STRING - Treat values as strings

SORT_LOCALE_STRING - Treat values as strings, based on local settings.

```
<?php
$info = array('name' => 'Rajesh Kumar Chapagain','address' => 'Kathmandu-32','email'
=> 'info@rajesh.com.np','website' => 'rajesh.com.np');
ksort($info);
print_r($info);
?>
```

Sort array with key descending

The PHP `ksort()` sorts an associative array in descending order, according to the key. It is mainly used for an associative array.

bool ksort (array &\$array [, int \$sort_flags = SORT_REGULAR]);

array => It's compulsory parameter and specifies the array to use.

sort_flag It's a optional parameter and specifies how to sort the array values.

Possible values -

SORT_REGULAR - Default. Treat values as they are (don't change types)

SORT_NUMERIC - Treat values numerically

SORT_STRING - Treat values as strings

SORT_LOCALE_STRING - Treat values as strings, based on local settings.

```
<?php
$info = array('name' => 'Rajesh Kumar Chapagain','address' => 'Kathmandu-32','email'
=> 'info@rajesh.com.np','website' => 'rajesh.com.np');
ksort($info);
print_r($info);
?>
```

Difference between two arrays

If you are trying to exclude few elements from an array which are common to another array's elements, then "`array_diff`" is the perfect function for you.

array_diff (array \$array1, array \$array2 [, array \$...])

```
<?php
$myarray1 = array(51,52,53,554,--15,--25);
$myarray2 = array(-15,-52,-15);
$filtered_array = array_diff($myarray1,$myarray2);
print_r($filtered_array);
?>
```


Get common elements among two arrays:

If you are trying to get only the common elements from two different arrays, "array_intersect" is the proper function for such purpose.

`array_intersect ( array $array1, array $array2 [, array $... ] )`

```
<?php
    $myarray1 = array(1,2,3,4,-1,-2);
    $myarray2 = array(-1,-2,3);
    $common_array = array_intersect($myarray1,$myarray2);
    print_r($common_array);
?>
```

Getting keys/values separately from associative array:

The `array_values()` function is an inbuilt function of PHP. It is used to get an array of values from another array that may contain key-value pairs.

The `array_keys()` function is an inbuilt function of PHP. It is used to get an array of keys from another array that may contain key-value pairs.

```
<?php
$info = array('name' => 'Rajesh Kumar Chapagain','address' => 'Kathmandu-32','email'
=> 'info@rajesh.com.np','website' => 'rajesh.com.np');
$keys = array_keys($info);
$values = array_values($info);
print_r($keys);
echo '<br>';
print_r($values);
?>
```

Get size of array

The `count()` or `sizeof()` function tells you the number of elements in an array

`count(array_name)` or `sizeof(array_name)`

```
<?php
$info = array('name' => 'Rajesh Kumar Chapagain','address' => 'Kathmandu-32','email'
=> 'info@rajesh.com.np','website' => 'rajesh.com.np');
echo count($info);
echo sizeof($info);
?>
```

Search array by key or value

To check for an element with certain key, use `array_key_exists()`. This function returns true if an element with the provided key exists in the provided array.

`array_key_exists(key,array_name)`

To check for an element with certain key, use `in_array()`. This function returns true if it finds an element with the given value. It is case-sensitive when it compares strings.

`in_array(value,array_name)`

```
<?php
$info = array('name' => 'Rajesh Kumar Chapagain','address' => 'Kathmandu-32','email'
=> 'info@rajesh.com.np','website' => 'rajesh.com.np');
if (array_key_exists('name', $info)) {
    echo "Record exists name key value pair";
}
?>

<?php
$info = array('name' => 'Rajesh Kumar Chapagain','address' => 'Kathmandu-32','email'
=> 'info@rajesh.com.np','website' => 'rajesh.com.np');
if (in_array('Kathmandu-32', $info)) {
    echo "Record exists key with value Kathmandu-32 value pair";
}
?>
```

Remove array element

To remove an element from an array, use `unset()`. Removing an element with `unset()` is different than just setting the element value to 0 or the empty string. When you use `unset()`, the element is no longer there when you iterate through the array or count the number of elements in the array. Using `unset()` on an array that represents a store's inventory is like saying that the store no longer carries a product. Setting the element's value to 0 or the empty string says that the item is temporarily out of stock.

`unset(array[key]).`

Convert array to string and vice-versa

The `explode` function is used to "Split a string by a specified string into pieces i.e. it breaks a string into an array". The `explode` function in PHP allows us to break a string into smaller text with each break occurring at the same symbol. This symbol is known as the delimiter. Using the `explode` command we will create an array from a string. The `explode()` function breaks a string into an array, but the `implode` function returns a string from the elements of an array.

The `implode` function in PHP is used to "join elements of an array with a string". The `implode()` function returns a string from elements of an array. It takes an array of strings and joins them together into one string using a delimiter (string to be used between the pieces) of your choice.

The `implode` function in PHP is easily remembered as "array to string", which simply means that it takes an array and returns a string. It rejoins any array elements and returns the resulting string, which may be put in a variable.

```
<?php
$subjects = array('DBMS','Scripting Language', 'Operating system', 'Numerical
Method','Project - I','Software Engineering');
$subjectString = implode('+', $subjects);
echo "String after conversion is : " . $subjectString;
$newSubjectArray = explode('+', $subjectString);
print_r($newSubjectArray);
?>
```


Final code for above functions:

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Built-in Array Functions</title>
</head>
<body>
<?php
ini_set('display_errors', 0);
$boys = array('Ram', 'Shyam', 'John');
$girls = array('Sita', 'Rita', 'Nita');

//now merge array of boys and girls into single array and call then student of single
class

$students = array_merge($boys,$girls);
print_r($students);
?>
<br/>
<?php
$array1=array("a"=>"Horse", "b"=>"Cat", "c"=>"Dog");
$array2=array("d"=>"Cow", "a"=>"Cat", "e"=>"elephant");
print_r(array_merge_recursive($array1,$array2));
?>
<br/>
<?php
//lets sort previously merged student array assending
sort($students);
print_r($students);
?>
<br/>
<?php
//lets sort previously merged student array decending
rsort($students);
print_r($students);
?>
<br/>
<?php
$info = array('name' => 'Rajesh Kumar Chapagain', 'address' => 'Kathmandu-32', 'email'
=> 'info@rajesh.com.np', 'website' => 'rajesh.com.np');
ksort($info);
print_r($info);
?>
```

```

<br/>
<?php
    $info = array('name' => 'Rajesh Kumar Chapagain','address' => 'Kathmandu-32','email'
=> 'info@rajesh.com.np','website' => 'rajesh.com.np');
    krsort($info);
    print_r($info);
?>
<br/>
<?php
$myarray1 = array(51,52,53,554,--15,--25);
$myarray2 = array(-15,-52,-15);
$filtered_array = array_diff($myarray1,$myarray2);
print_r($filtered_array);
?>
<br/>
<?php
    $myarray1 = array(1,2,3,4,-1,-2);
    $myarray2 = array(-1,-2,3);
    $common_array = array_intersect($myarray1,$myarray2);
    print_r($common_array);
?>

<?php
$info = array('name' => 'Rajesh Kumar Chapagain','address' => 'Kathmandu-32','email'
=> 'info@rajesh.com.np','website' => 'rajesh.com.np');
$keys = array_keys($info);
$values = array_values($info);
print_r($keys);
echo '<br>';
print_r($values);
?>
<br>
<?php
$subjects = array('DBMS','Scripting Language', 'Operating system', 'Numerical
Method','Project - I','Software Engineering');
$subjectString = implode('+', $subjects);
echo "String after conversion is : " . $subjectString;
$newSubjectArray = explode('+', $subjectString);
print_r($newSubjectArray);
?>
</body>
</html>

```


Output

```

Array ( [0] => Ram [1] => Shyam [2] => John [3] => Sita [4] => Rita [5] => Nita )
Array ( [a] => Array ( [0] => Horse [1] => Cat [2] => Cat [3] => Dog [4] => Cow [5] => elephant )
Array ( [0] => John [1] => Nita [2] => Ram [3] => Rita [4] => Shyam [5] => Sita )
Array ( [0] => Sita [1] => Shyam [2] => Rita [3] => Ram [4] => Nita [5] => John )
Array ( [address] => Kathmandu-32 [email] => info@rajesh.com.np [name] => Rajesh Kumar Chapagain [website] => rajesh.com.np )
Array ( [website] => rajesh.com.np [name] => Rajesh Kumar Chapagain [email] => info@rajesh.com.np [address] => Kathmandu-32 )
Array ( [0] => 51 [1] => 52 [2] => 53 [3] => 554 [5] => -25 )
Array ( [2] => 3 [4] => 1 [5] => 2 ) Array ( [0] => name [1] => address [2] => email [3] => website )
Array ( [0] => Rajesh Kumar Chapagain [1] => Kathmandu-32 [2] => info@rajesh.com.np [3] => rajesh.com.np )
String after conversion is : DBMS+Scripting Language+Operating system+Numerical Method+Project - I+Software EngineeringArray (
[0] => DBMS [1] => Scripting Language [2] => Operating system [3] => Numerical Method [4] => Project - I [5] => Software
Engineering )

```

FORM HANDLING IN PHP

Form processing is an essential component of almost any web application. Forms are how users communicate with your server: signing up for a new account, searching a forum for all the posts about a particular subject, retrieving a lost password, finding a nearby restaurant or shoemaker, or buying a book. Forms come in handy when developing flexible and dynamic applications that accept user input, edit already existing data from the database

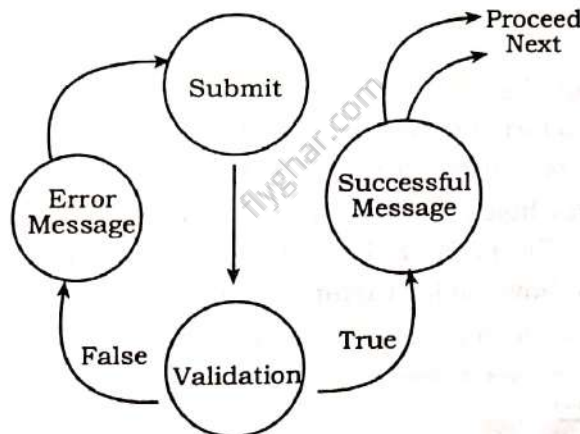


Fig: General Form Processing

Using a form in a PHP program is a two-step activity. Step one is to display the form. This involves constructing HTML that has tags for the appropriate user-interface elements in it, such as text boxes, checkboxes, and buttons.

When a user sees a page with a form in it, she inputs the requested information into the form and then clicks a button or hits enter to send the form information back to your server. Processing that submitted form information is step two of the operation.

Process of creating form

We will use HTML tags to create a form. Below is the minimal list of things you need to create a form.

- Opening and closing form tags <form>...</form>
- Form submission type POST or GET
- Submission URL that will process the submitted data
- Input fields such as input boxes, text areas, buttons, checkboxes etc.

```

<!DOCTYPE html>
<html>
<head>
    <title>Form Example</title>
</head>
<body>
    <h2>Registration Form</h2>
    <form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="get">
    First name: <input type="text" name="first_name"><br><br>
    Last name : <input type="text" name="last_name"><br>
    <input type="submit" value="Submit" name="Register">
    </form>
</body>
</html>

```

- <form...>...</form> are the opening and closing form tags
- action="" method="POST" specifies the destination URL and the submission type, if not specified will redirect to same page
- First/Last name: are labels for the input boxes
- <input type="text" ...> are input box tags
- <input type="submit" value="Submit" name="submit"> is the button that when clicked submits the form to the server for processing

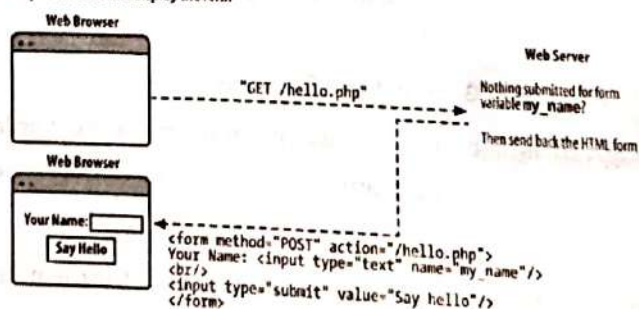
Accessing Form Elements

At the beginning of every request, the PHP engine sets up some super global arrays that contain the values of any parameters submitted in a form or passed in the URL. URL and form parameters from GET method forms are put into \$_GET. Form parameters from POST method forms are put into \$_POST.

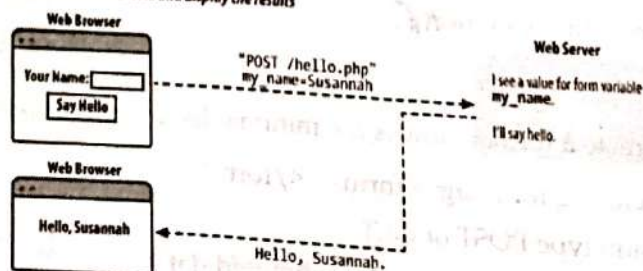
The \$_SERVER super global array holds a variety of information about your server and the current request the PHP engine is processing. The PHP_SELF element of \$_SERVER holds the pathname part of the current request's URL, which is shown in the example below.

The following figure indicates displaying and processing a simple form

Step 1: Retrieve and display the form



Step 2: Submit the form and display the results



There are major two ways the browser client can send information to the web server.

ACCESSING FORMS USING GET METHOD

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character.

Sample URL for GET: http://www.example.com/index.php?first_name=ram&last_name=thapa

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send up to 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides \$_GET associative array to access all the sent information using GET method.

```
<?php
if(isset($_GET['Register'])) {
    if( $_GET["first_name"] && $_GET["last_name"] ) {
        echo "Welcome ". $_GET['first_name']. ' ' . $_GET['last_name'];
        exit();
    }
}
?>

<!DOCTYPE html>
<html>
<head>
<title>Form Example</title>
</head>
<body>
<h2>Registration Form</h2>
<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="get">
    First name: <input type="text" name="first_name"><br><br>
    Last name : <input type="text" name="last_name"><br>
    <input type="submit" value="Submit" name="Register">
</form>
</body>
</html>
```

ACCESSING FORMS USING POST METHOD

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides \$_POST associative array to access all the sent information using POST method.

```
<?php
if(isset($_POST['Register'])) {
    if( $_POST["first_name"] && $_POST["last_name"] ) {
        echo "Welcome ". $_POST['first_name']. ' ' . $_POST['last_name'];
        exit();
    }
}
?>

<!DOCTYPE html>
<html>
<head>
<title>Form Example</title>
</head>
<body>
<h2>Registration Form</h2>
<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="post">
    First name: <input type="text" name="first_name"><br><br>
    Last name : <input type="text" name="last_name"><br>
    <input type="submit" value="Submit" name="Register">
</form>
</body>
</html>
```


Difference between get and post method.

POST	GET
Values not visible in the URL	Values visible in the URL
Has not limitation of the length of the values since they are submitted via the body of HTTP.	Has limitation on the length of the value usually 255 characters. This is because the values are displayed in the URL. Note the upper limit of the characters is dependent on the browser.
Has lower performance compared to GET method due to time spent encapsulation the POST values in the HTTP body.	Has high performance compared to POST method due to the simple nature of appending the values in the URL.
Supports many different data types such as string, numeric, binary etc.	Supports only string data types because the values are displayed in the URL
Results cannot be book marked	Results can be book marked due to the visibility of the values in the URL

FORM VALIDATION

Data validation is one of the most important parts of a web application. Weird, wrong, and damaging data shows up where you least expect it. Users can be careless, malicious, and fabulously more creative (often accidentally) than you may ever imagine when you are designing your application. Validation means check the input submitted by the user. There are two types of validation are available in PHP. They are as follows:

- Client-Side Validation – Validation is performed on the client machine web browsers.
- Server Side Validation – after submitted by data, the data has sent to a server and perform validation checks in server machine.

Server Side Validation**Text validation**

For validating text we use regular expression i.e **preg_match()**. This function searches string for pattern, returns true if pattern exists, otherwise returns false. Usually search starts from beginning of subject string. The optional parameter offset is used to specify the position from where to start the search.

Example 28: Text Validation with preg_match()

```
<?php
$name = $_POST['name'];
if (!preg_match("/^[a-zA-Z ]*$/", $name)) {
    echo "Only letters and white space allowed";
}
else{
    echo "The name is". $name;
}
?>
```

In the above example the `preg_match()` function searches in string given through a variable `$name` for the pattern of a-zA-z.

Email validation

To validate the email, use `FILTER_VALIDATE_EMAIL`

Example 29: Email validation using `FILTER_VALIDATE_EMAIL`

```
<?php
// Variable to check
$email = $_POST['email'];
// Validate email
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo("$email is a valid email address");
} else {
    echo("$email is not a valid email address");
}
```

Number Validation:

To validate the integer, use `FILTER_VALIDATE_INT` and to validate float, use `FILTER_VALIDATE_FLOAT` into `filter_validate` function.

Some useful validating functions

- **htmlspecialchars**: converts some predefined characters to HTML entities.
- **stripslashes**: function removes backslashes
- **trim**: removes the white spaces
- **isset**: check variable existence
- **empty**: check empty value of variable
- **addslashes**: used to add backslashes in front of the characters
- **htmlentities**: converts characters to HTML entities.
- **html_entity_decode**: converts HTML entities to characters.

PHP Date and Time function

What is a Timestamp?

A timestamp is a numeric value in seconds between the current time and value as at 1st January, 1970 00:00:00 Greenwich Mean Time (GMT). The value returned by the time function depends on the default time zone. The default time zone is set in the `php.ini` file. It can also be set programmatically using `date_default_timezone_set` function.

time() function:

The `time()` function returns the current time as Unix timestamp. The UNIX timestamps measures time as a number of seconds since midnight of Greenwich Mean Time on January 1, 1970.


```

<!DOCTYPE html>
<html>
<head>
    <title> PHP Date and Time Function</title>
</head>
<body>
<h1>Date Time Function</h1>
<?php
$currentTime = time();
$timeAfterOneWeek = time()+(7*24*60*60);
$timeAfterFiftyDays = time()+(50*24*60*60);
$timeBeforeOneMonth = time() - (30*24*60*60);
echo "<br>Current timestamp is : " . $currentTime;
echo "<br>Timestamp after one week is : " . $timeAfterOneWeek;
echo "<br>Timestamp after one fifty days is : " . $timeAfterFiftyDays;
echo "<br>Timestamp before one month is : " . $timeBeforeOneMonth;
?>
</body>
</html>

```

date() function:

The date() function displays a formatted local time/date.

date(format,[timestamp])

Here are some of the most common date format characters and their values.

Character	Description	Example
d	day of the month with leading zeros	07 or 17
j	day of the month without leading zeros	7 or 17
D	day of the week as a three-letter abbreviation	Tue
l	full day of the week	Tuesday
m	month as a number with leading zeros	06 or 12
n	month as a number without leading zeros	6 or 12
M	month as a three-letter abbreviation	Nov
F	full month	November
y	two-digit year	19
Y	full year	2020

Some of the most commonly used time format characters:

Character	Meaning	Example
g	hours in 12-hour format without leading zeros	1 or 12
h	hours in 12-hour format with leading zeros	01 or 12
G	hours in 24-hour format without leading zeros	1 or 13
H	hours in 24-hour format with leading zeros	01 or 13
a	am/pm in lowercase	am
A	am/pm in uppercase	AM
i	minutes with leading zeros	09 or 15
s	seconds with leading zeros	05 or 30

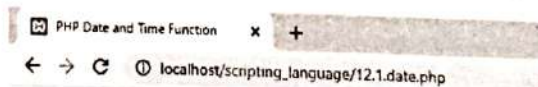
Example

```

<!DOCTYPE html>
<html>
<head>
    <title> PHP Date and Time Function</title>
</head>
<body>
<h1>Date Time Function</h1>
<?php
$currentTime = time();
$timeAfterOneWeek = time()+(7*24*60*60);
$timeAfterFiftyDays = time()+(50*24*60*60);
$timeBeforeOneMonth = time() - (30*24*60*60);
echo "<br>Today date is : " . date('Y-m-d',$currentTime);
echo "<br>Date after one week is : " . date('Y-m-d H:i:s' , $timeAfterOneWeek);
echo "<br>Date after one fifty days is : " . date('Y-m-d H:i:s' ,
$timeAfterFiftyDays);
echo "<br>Date before one month is : " . date('Y-m-d H:i:s' , $timeBeforeOneMonth);
echo "<br>";
echo date('h:i:s A');
echo "<br>";
echo date('l, h:i:s A');
echo "<br>";
echo date('d F Y, h:i:s A');
echo "<br>";
echo date('Y');
echo "<br>";
echo date('F Y');
echo "<br>";
echo date('d F, Y');
echo "<br>";
echo date('d F, Y (1)');
?>
</body>
</html>

```

Output



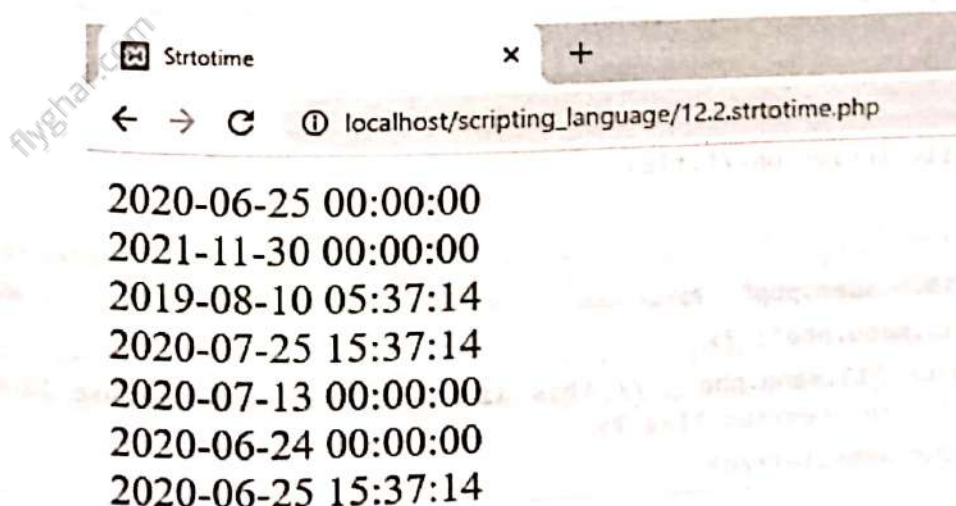
Date Time Function

Today date is : 2020-06-25
 Date after one week is : 2020-07-02 15:36:31
 Date after one fifty days is : 2020-08-14 15:36:31
 Date before one month is : 2020-05-26 15:36:31
 03:36:31 PM
 Thursday, 03:36:31 PM
 25 June 2020, 03:36:31 PM
 2020
 June 2020
 25 June, 2020
 25 June, 2020 (Thursday)

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Strtotime</title>
</head>
<body>
<?php
$time1 = strtotime("today");
$time2 = strtotime("last day of November 2021");
$time3 = strtotime("10 months 15 days 10 hours ago");
$time4 = strtotime("next month");
$time5 = strtotime("third monday");
$time6 = strtotime("last day of November 2021");
$time6 = strtotime("yesterday");
$time7 = strtotime("now");
echo date('Y-m-d H:i:s' , $time1);
echo "<br>";
echo date('Y-m-d H:i:s' , $time2);
echo "<br>";
echo date('Y-m-d H:i:s' , $time3);
echo "<br>";
echo date('Y-m-d H:i:s' , $time4);
echo "<br>";
echo date('Y-m-d H:i:s' , $time5);
echo "<br>";
echo date('Y-m-d H:i:s' , $time6);
echo "<br>";
echo date('Y-m-d H:i:s' , $time7);
echo "<br>";
?>
</body>
</html>
```

Output



PHP Include and Require Files

The `include()` and `require()` statement allow you to include the code contained in a PHP file within another PHP file. Including a file produces the same result as copying the script from the file specified and pasted into the location where it is called.

You can save a lot of time and work through including files — Just store a block of code in a separate file and include it wherever you want using the `include()` and `require()` statements instead of typing the entire block of code multiple times. A typical example is including the header, footer and menu file within all the pages of a website.

```
include("path/to/filename"); -Or- include "path/to/filename";
require("path/to/filename"); -Or- require "path/to/filename";
```

Difference Between include and require Statements

You might be thinking if we can include files using the `include()` statement then why we need `require()`. Typically the `require()` statement operates like `include()`.

The only difference is — the `include()` statement will only generate a PHP warning but allow script execution to continue if the file to be included can't be found, whereas the `require()` statement will generate a fatal error and stops the script execution.

The following example will show you how to include the common header, footer and menu codes which are stored in separate '13.header.php', '13.footer.php' and '13.menu.php' files respectively, within all the pages of your website. Using this technique you can update all pages of the website at once by making the changes to just one file, this saves a lot of repetitive work.

The include_once and require_once Statements

If you accidentally include the same file (typically functions or classes files) more than one time within your code using the `include` or `require` statements, it may cause conflicts. To prevent this situation, PHP provides `include_once` and `require_once` statements. These statements behave in the same way as `include` and `require` statements with one exception.

The `include_once` and `require_once` statements will only include the file once even if asked to include it a second time i.e. if the specified file has already been included in a previous statement, the file is not included again.

Following example will demonstrate above function:

File: 13.inclusion.php

```
<!DOCTYPE html>
<html>
<head>
  <title>File Inclusion</title>
</head>
<body>
  <?php include "13.header.php"; ?>
  <?php include "13.menu.php"; ?>
  <?php include_once "13.menu.php"; // this line will not work because 13.menu.php is
  already included into previous line ?>
  <h3>Welcome to Our Website!</h3>
```



```
<p>Here you will find lots of useful information.</p>
<?php //require "gallery.php"; => This line will throw fatal error if we uncomment
this line ?>
<?php include "13.footer.php"; ?>
</body>
</html>
```

File: 13.header.php

```
<h1>BCA 4th Websites</h1>
```

File: 13.menu.php

```
<a href="index.php">Home</a>
<a href="about.php">About</a>
<a href="service.php">Service</a>
<a href="gallery.php">Gallery</a>
<a href="contact.php">Contact</a>
<br>
```

File: 13.footer.php

```
<hr>
copyright &copy; <?php echo date('Y') ?>
```

Output

 File Inclusion

x +

 localhost/scripting_language/13.inclusion.php

BCA 4th Websites

[Home](#) [About](#) [Service](#) [Gallery](#) [Contact](#)**Welcome to Our Website!**

Here you will find lots of useful information.

copyright © 2020

FILE HANDLING

PHP is a server side programming language, it allows you to work with files and directories stored on the web server. We can create, access, and manipulate files on your web server using the PHP file system functions.

Opening file

The PHP `fopen()` function is used to open a file. To work with a file you first need to open the file. The first parameter passed to `fopen()` specifies the name of the file you want to open, and the second parameter specifies in which mode the file should be opened.

<code>fopen(filename, mode)</code>	
Modes	Description
<code>r</code>	Open the file for reading only.
<code>r+</code>	Open the file for reading and writing.
<code>w</code>	Open the file for writing only and clears the contents of file. If the file does not exist, PHP will attempt to create it.
<code>w+</code>	Open the file for reading and writing and clears the contents of file. If the file does not exist, PHP will attempt to create it.
<code>a</code>	Append. Opens the file for writing only. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it.
<code>a+</code>	Read/Append. Opens the file for reading and writing. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it.
<code>x</code>	Open the file for writing only. Return FALSE and generates an error if the file already exists. If the file does not exist, PHP will attempt to create it.
<code>x+</code>	Open the file for reading and writing; otherwise it has the same behavior as 'x'.

If you try to open a file that doesn't exist, PHP will generate a warning message. So, to avoid these error messages you should always implement a simple check whether a file or directory exists or not before trying to access it, with the PHP `file_exists()` function.

Closing a File

Once you've finished working with a file, it needs to be closed. The `fclose()` function is used to close the file.

```
fclose(handle);
```

Reading from Files

PHP has several functions for reading data from a file. You can read from just one character to the entire file with a single operation.

The `fread()` function can be used to read a specified number of characters from a file. It supports two parameters: a file handle and the number of bytes to read.

The `fread()` function can be used in conjunction with the `fseek()` function to read the entire file at once to read the entire content of a file. The `fseek()` function returns the size of the file in bytes.

`fread(file handle, length in bytes)`

```
<?php
$file = "14.data.txt";
// Check the existence of file
if(file_exists($file)){
    // Attempt to open the file
    $fp = fopen($file, "r") or die("ERROR: Cannot open the file.");
    // Read fixed number of bytes from the file
    $content = fread($fp, 20);
    //close file handle
    fclose($fp);
} else{
    echo "ERROR: File does not exist.";
}
?>
```

The easiest way to read the entire contents of a file in PHP is with the `readfile()` function. This function allows you to read the contents of a file without needing to open it.

```
<?php
$file = "14.data.txt";
// Check the existence of file
if(file_exists($file)){
    // Reads and outputs the entire file
    readfile($file) or die("ERROR: Cannot open the file.");
} else{
    echo "ERROR: File does not exist.";
}
?>
```

Another way to read the whole contents of a file without needing to open it is with the `file_get_contents()` function.

```
<?php
$file = "14.data.txt";
// Check the existence of file
if(file_exists($file)){
    // Reading the entire file into a string
    $content = file_get_contents($file) or die("ERROR: Cannot open the file.");
    // Display the file content
    echo $content;
} else{
    echo "ERROR: File does not exist.";
}
?>
```

One more method of reading the whole data from a file is the PHP's `file()` function. It does a similar job to `file_get_contents()` function, but it returns the file contents as an array of lines, rather than a single string. Each element of the returned array corresponds to a line in the file. To process the file data, you need to iterate over the array using a `foreach` loop.

```
<?php
$file = "14.data.txt";
// Check the existence of file
if(file_exists($file)){
    // Reading the entire file into an array
    $lines = file($file) or die("ERROR: Cannot open the file.");
    foreach($lines as $line){
        echo $line;
    }
} else{
    echo "ERROR: File does not exist.";
}
?>
```

Writing the Files

We can write data to a file or append to an existing file using the PHP `fwrite()` function. The `fwrite()` function takes two parameter: A file handle and the string of data that is to be written.

`fwrite(file handle, string)`

if the "14.data.txt" file doesn't exist PHP will automatically create it and write the data. But, if the "14.data.txt" file already exist, PHP will erase the contents of this file, if it has any, before writing the new data, however if you just want to append the file and preserve existing contents just use the mode append 'a' mode instead of 'w' mode.

```
<?php
$file = "14.data.txt";
// String of data to be written
$data = "This is writting example.";
// Open the file for writing
$handle = fopen($file, "w") or die("ERROR: Cannot open the file.");
// Write data to the file
fwrite($handle, $data) or die ("ERROR: Cannot write the file.");
// Closing the file handle
fclose($handle);
echo "Data written to the file successfully.";
?>
```


We can use the `file_put_contents()` function to write content into file. It is counterpart of `file_get_contents()` function and provides an easy method of writing the data to a file without needing to open it. This function accepts the name and path to a file together with the data to be written to the file.

```
file_put_contents(filename,data);
```

If the file specified in the `file_put_contents()` function already exists, PHP will overwrite it by default. If you would like to preserve the file's contents you can pass the special `FILE_APPEND` flag as a third parameter to the `file_put_contents()` function. It will simply append the new data to the file instead of overwriting it.

```
<?php
$file = "14.data.txt";
// String of data to be written
$data = "The quick brown fox jumps over the lazy dog.";
// Write data to the file
file_put_contents($file, $data) or die("ERROR: Cannot write the file.");
echo "Data written to the file successfully.";
?>
```

RENAMING FILES

You can rename a file or directory using the PHP's `rename()` function. `rename()` will take two parameter first as an old file name and second as a new file name.

```
rename(old file, new file)
```

```
<?php
$file = "14.old.txt";
// Check the existence of file
if(file_exists($file)){
    // Attempt to rename the file
    if(rename($file, "14.new.txt")){
        echo "File renamed successfully.";
    } else{
        echo "ERROR: File cannot be renamed.";
    }
} else{
    echo "ERROR: File does not exist.";
}
?>
```

Removing Files

You can delete files or directories using the PHP's `unlink()` function. `unlink()` will take single parameter as a file name to delete.

```
<?php
$file = "14.data.txt";
// Check the existence of file
if(file_exists($file)){
    // Attempt to delete the file
    if(unlink($file)){
        echo "File removed successfully.";
    } else{
        echo "ERROR: File cannot be removed.";
    }
} else{
    echo "ERROR: File does not exist.";
}
?>
```

ERROR IN PHP

Error is the fault or mistake in a program. It can be several types. Error can occur due to wrong syntax or wrong logic. It is a type of mistakes or condition of having incorrect knowledge of the code.

Basically there are 4 type of errors.

Parse error or Syntax Error: It is the type of error done by the programmer in the source code of the program. The syntax error is caught by the compiler. After fixing the syntax error the compiler compile the code and execute it. Parse errors can be caused dues to unclosed quotes, missing or extra parentheses, unclosed braces, Missing semicolon etc.

```
<?php
$name = 'sdfdsf';
address = "Kathmandu";
echo $name;
echo $address;
?>
```

In above program, \$ sign is missing in line 3 so it gives an error message.

Fatal Error: It is the type of error where PHP compiler understand the PHP code but it recognizes an undeclared function. This means that function is called without the definition of function.

```
<?php
function sum($x, $y)
{
    $sum = $x + $y;
    echo "sum = " . $sum;
}
$x = 0;
$y = 63;
sum($x, $y);
diff($x, $y);
?>
```


In above program, function diff is called but the definition of function is not available. So it gives error.

Warning Errors: The main reason of warning errors are including a missing file. This means that the PHP function call the missing file.

```
<?php
$x = "MY Test String";
include ("index.php");
echo $x . "Welcome to our website";
?>
```

This program call a missing file index.php which are not available. So it produces error.

Notice Error: It is similar to warning error. It means that the program contains something wrong but it allows the execution of script.

```
<?php
$name = "Pranab Sharma";
echo $name;
echo $address;
?>
```

In the above program, undeclared variable \$address so it gives error message.

Changing runtime configuration for PHP errors

We use ini_set function to set error parameter at run time which will try to override the configuration found in PHP ini file.

There are four ways to display errors in PHP.

- **error_reporting:** It does not display the E-STRICT, E-NOTICE and E_DEPRECATED level errors and display all the other level errors.
- **display_errors:** Its default value is "off". Set it to "on".
- **log_errors:** It's default value is "on" which indicates whether or not error logging should be done.
- **error_log string:** It sets the name of the file where scripts error should be logged.

```
<?php
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
?>
```

PHP ERROR HANDLING

Sometimes your application will not run as it supposed to do, resulting in an error. There are a number of reasons that may cause errors, like:

- The Web server might run out of disk space
- A user might have entered an invalid value in a form field
- The file or database record that you were trying to access may not exist

- The application might not have permission to write to a file on the disk
- A service that the application needs to access might be temporarily unavailable

These types of errors are known as runtime errors, because they occur at the time the script runs. They are distinct from syntax errors that need to be fixed before the script will run. A professional application must have the capabilities to handle such runtime error gracefully. Usually this means informing the user about the problem more clearly and precisely.

The PHP engine triggers an error whenever it encounters a problem with your script, but you can also trigger errors yourself to generate more user friendly error messages. This way you can make your application more sophisticated.

Error Levels

When there's a problem that prevents a script from running properly, the PHP engine triggers an error. Each error is represented by an integer value and an associated constant.

- 1 : E_ERROR :Fatal runtime error execution of script has been halted
- 2 : E_WARNING :Non-fatal runtime error execution of script has been halted
- 4 : E_PARSE :Compile time error it is generated by the parser
- 8 : E_NOTICE :The script found something that might be an error
- 16 : E_CORE_ERROR :Fatal errors that occurred during initial startup of script
- 32 : E_CORE_WARNING :Non-fatal errors that occurred during initial startup of script
- 8191 : E_ALL :All errors and warning

Error handling in PHP is almost similar to error handling in all programming languages. The default error handling in PHP will give file name line number and error type.

Ways to handle PHP Errors

die() method

Basic error handling: Using die() function The die() function print a message and exit from current script.

```
die( $message )
```

```
<?php
// If file is not present then exit from script
if( !file_exists("geeks.txt") ) {
    die("File is not present");
}
// If file is present
// then continue
else {
    $file = fopen("15.data.txt", "w");
}
?>
```


Custom Error Handling

Creating a custom error handler in PHP is quite simple. Create a function that can be called when error has been occurred in PHP.

```
error_function( $error_level, $error_message, $error_file, $error_line, $error_context);
```

Parameters:

- **\$error_level:** It is required parameter and it must be an integer. There are predefined error levels.
- **\$error_message:** It is required parameter and it is the message which user want to print.
- **\$error_file:** It is optional parameter and used to specify the file in which error has been occurred.
- **\$error_line:** It is optional parameter and used to specify the line number in which error has been occurred.
- **\$error_context:** It is optional parameter and used to specify an array containing every variable and their value when error has been occurred.

set_error_handler() Function: After creating customError() function need to set custom error handler because in normal way PHP handles it but if user doing custom error handling then user have to set it in place of argument and pass out customError function as a string.

```
<?php
// Creates my error function which prints message to user
function customError($errno, $errstr, $errfile, $errline, $errcontext){
    $message = date("Y-m-d H:i:s - ");
    $message .= "Error: [" . $errno ."], " . "$errstr in $errfile on line $errline, ";
    $message .= "Variables:" . print_r($errcontext, true) . "\r\n";
    echo $message;
    die("There was a problem, please try again.");
}
set_error_handler("customError"); // Setting set_error_handler
$a = 32;
$b = 0;
// This will generate error
echo($a / $b);
?>
```

FILE UPLOADING

Successfully managing file uploads via PHP is the result of cooperation between various configuration directives, the \$_FILES superglobal, and a properly coded web form.

PHP's File Upload/Resource Directives

1. file_uploads = On | Off
Scope: PHP_INI_SYSTEM; Default value: On
The file_uploads directive determines whether PHP scripts on the server can accept file uploads.

2. `max_input_time = integer`

Scope: `PHP_INI_ALL`; Default value: 60

The `max_input_time` directive determines the maximum amount of time, in seconds, that a PHP script will spend attempting to parse input before registering a fatal error.

This is relevant because particularly large files can take some time to upload, eclipsing the time limit set by this directive.

3. `max_file_uploads = integer`

Scope: `PHP_INI_SYSTEM`; Default value: 20

Available since PHP 5.2.12, the `max_file_uploads` directive sets an upper limit on the number of files which can be simultaneously uploaded.

4. `memory_limit = integerM`

Scope: `PHP_INI_ALL`; Default value: 16M

The `memory_limit` directive sets a maximum allowable amount of memory in megabytes that a script can allocate (note that the integer value must be followed by M for this setting to work properly). It prevents runaway scripts from monopolizing server memory and even crashing the server in certain situations.

5. `post_max_size = integerM`

Scope: `PHP_INI_PERDIR`; Default value: 8M

The `post_max_size` places an upper limit on the size of data submitted via the POST method. Because files are uploaded using POST, you may need to adjust this setting upwards along with `upload_max_filesize` when working with larger files.

6. `upload_max_filesize = integerM`

Scope: `PHP_INI_PERDIR`; Default value: 2M

The `upload_max_filesize` directive determines the maximum size in megabytes of an uploaded file. This directive should be smaller than `post_max_size` because it applies only to information passed via the file input type and not to all information passed via the POST instance.

The `$_FILES` Array

The `$_FILES` superglobal stores a variety of information pertinent to a file uploaded to the server via a PHP script. In total, five items are available in this array, each of which is described here:

- `$_FILES['userfile']['error']`: This array value specifies error or status code associated with the file upload, e.g. it will be 0, if there is no error.
- `$_FILES['userfile']['name']`: This variable specifies the original name of the file, including the extension, as declared on the client machine. Therefore, if you browse to a file named `my.png` and upload it via the form, this variable will be assigned the value `my.png`.
- `$_FILES['userfile']['size']`: This variable specifies the size, in bytes, of the file uploaded from the client machine. For example, in the case of the `my.png` file, this variable could plausibly be assigned a value such as 5253, or roughly 5KB.
- `$_FILES['userfile']['tmp_name']`: This variable specifies the temporary name assigned to the file once it has been uploaded to the server. This is the name of the file assigned to it while stored in the temporary directory (specified by the PHP directive `upload_tmp_dir`).

`$_FILES['userfile']['type']`: This variable specifies the MIME type of the file uploaded from the client machine. Therefore, in the case of the vacation.png image file, this variable would be assigned the value image/png. If a PDF was uploaded, the value application/pdf would be assigned. Because this variable sometimes produces unexpected results, you should explicitly verify it yourself from within the script.

PHP's File-Upload Functions

In addition to the number of file-handling functions made available via PHP's filesystem library, PHP offers two functions specifically intended to aid in the fileupload process, `is_uploaded_file()` and `move_uploaded_file()`. Determining Whether a File Was Uploaded The `is_uploaded_file()` function determines whether a file specified by the input parameter filename is uploaded using the POST method.

boolean `is_uploaded_file(string filename)`

This function is intended to prevent a potential attacker from manipulating files not intended for interaction via the script in question. For example, consider a scenario in which uploaded files are made immediately available for viewing via a public site repository.

Moving an Uploaded File

The `move_uploaded_file()` function provides a convenient means for moving an uploaded file from the temporary directory to a final location. Its prototype follows: `boolean move_uploaded_file(string filename, string destination)` although `copy()` works equally well, `move_uploaded_file()` offers one additional feature: it will check to ensure that the file denoted by the filename input parameter was in fact uploaded via PHP's HTTP POST upload mechanism. If the file has not been uploaded, the move will fail and a FALSE value will be returned. Because of this, you can forgo using `is_uploaded_file()` as a precursor condition to using `move_uploaded_file()`.

```
<?php
if (isset($_POST['btnUpload'])) {
    $err = [];
    //error check
    if (isset($_FILES['photo']['error']) && $_FILES['photo']['error'] == 0 ) {
        //size check
        if ($_FILES['photo']['size'] < 102400) {
            //type check
            $type = ['image/png','image/jpg','image/jpeg','image/gif'];
            if (in_array($_FILES['photo']['type'], $type)) {
                //upload file
                if (move_uploaded_file($_FILES['photo']['tmp_name'],
                    'photo/' . $_FILES['photo']['name'])) {
                    echo "File Upload Success";
                }else{
                    echo "File upload Failed";
                }
            }else{
                $err['photo'] = 'File type does not match';
            }
        }else {
            $err['photo'] = 'File size exceed, 100 kb allowed';
        }
    }else {
        //error message
    }
}
```



```

        $err['photo'] = 'File cannot upload';
    }
}

?>
<!DOCTYPE html>
<html>
<head>
<title>File Upload</title>
</head>
<body>
<h1>File Upload</h1>
<form action="" method="post" enctype="multipart/form-data">
<label for="photo">Photo</label>
<input type="file" name="photo" id="photo"/>
<?php if (isset($err['photo'])) {
    echo $err['photo'];
} ?>
<br/>
<br/>
<input type="submit" name="btnUpload">
</form>
</body>
</html>

```

SESSION AND COOKIE

What is a Cookie?

A cookie is small pieces of data which the webserver asks the client's web browser to store and each request back to the webserver will include these pieces of data. It is a small text file that lets you store a small amount of data (nearly 4KB) on the user's computer. They are typically used to keeping track of information such as username that the site can retrieve to personalize the page when user visit the website next time.

Setting a Cookie in PHP

The `setcookie()` function is used to set a cookie in PHP. Make sure you call the `setcookie()` function before any output generated by your script otherwise cookie will not set.

```
setcookie(name, value, expire, path, domain, secure);
```


All the arguments except the name are optional. You may also replace an argument with an empty string ("") in order to skip that argument, however to skip the expire argument use a zero (0) instead, since it is an integer.

- **Name:** The name of the cookie.
- **Value:** The value of the cookie. Do not store sensitive information since this value is stored on the user's computer.
- **Expires:** The expiry date in UNIX timestamp format. After this time cookie will become inaccessible. The default value is 0.
- **Path:** Specify the path on the server for which the cookie will be available. If set to /, the cookie will be available within the entire domain.
- **Domain:** Specify the domain for which the cookie is available to e.g www.example.com.
- **Secure:** This field, if present, indicates that the cookie should be sent only if a secure HTTPS connection exists.

```
<?php
// Setting a cookie
setcookie("username", "admin", time()+7*24*60*60);
?>
```

Accessing Cookies Values

The PHP \$_COOKIE superglobal variable is used to retrieve a cookie value. It typically an associative array that contains a list of all the cookies values sent by the browser in the current request, keyed by cookie name

```
<?php
// Verifying whether a cookie is set or not
if(isset($_COOKIE["username"])){
    echo "Hi " . $_COOKIE["username"];
} else{
    echo "Welcome Guest!";
}
?>
```

Removing Cookies

You can delete a cookie by calling the same setcookie() function with the cookie name and any value (such as an empty string) however this time you need to set the expiration date in the past. You should pass exactly the same path, domain, and other arguments that you have used when you first created the cookie in order to ensure that the correct cookie is deleted.

What is a Session?

A session is usually a file or database record which contains the small pieces of data which the webserver wants to store for each user. Instead of sending key/value pairs to the browser, these values are stored on the server, and only a reference identifier ("session ID") is sent to the user's browser as a cookie. This session ID needs to be a long and unique number. On each future request, the browser will send the session ID as a cookie and the webserver will locate the corresponding session file to allow access to the stored user data.

Since cookies are stored on user's computer it is possible for an attacker to easily modify a cookie content to insert potentially harmful data in your application that might break your application. You can solve both of these issues by using the PHP session. A PHP session stores data on the server rather than user's computer. In a session based environment, every user is identified through a unique number called session identifier or SID. This unique session ID is used to link each user with their own information on the server like emails, posts, etc. A session is a global variable stored on the server. Each session is assigned a unique id which is used to retrieve stored values. Whenever a session is created, a cookie containing the unique session id is stored on the user's computer/browser and returned with every request to the server. Sessions have the capacity to store relatively large data compared to cookies. The session values are automatically deleted when the browser is closed. If you want to store the values permanently, then you should store them in the database. Session variables are stored in the `$_SESSION` array variable. The session must be started before any HTML tags.

Why and when to use Sessions?

- You want to store important information such as the user id more securely on the server where malicious users cannot temper with them.
- You want to pass values from one page to another.
- You want the alternative to cookies on browsers that do not support cookies.
- You want to store global variables in an efficient and more secure way compared to passing them in the URL.
- You are developing an application such as a shopping cart, login page.

Creating a Session

In order to create a session, you must first call the PHP `session_start` function and then store your values in the `$_SESSION` array variable.

Let's suppose we want to know the number of times that a page has been loaded, we can use a session to do that.

The code below shows how to create and retrieve values from sessions.

```
<?php
session_start(); //start the PHP_session function
if(isset($_SESSION['page_count']))
{
    $_SESSION['page_count'] += 1;
}
else
{
    $_SESSION['page_count'] = 1;
}
echo 'You are visitor number ' . $_SESSION['page_count'];
?>
```


Destroying Session Variables

The `session_destroy()` function is used to destroy the whole PHP session variables.

If you want to destroy only a session single item, you use the `unset()` function.

The code below illustrates how to use both methods.

```
<?php
session_destroy(); //destroy entire session
?>
<?php
unset($_SESSION['username']); //destroy product session item
?>
```

`session_destroy` removes all the session data including cookies associated with the session where `Unset` only frees the individual session variables. Other data remains unchanged.

How Sessions Work

1. Server opens a session (sets a cookie via HTTP header)
2. Server sets a session variable.
3. Client changes page or refresh page.
4. Client sends all cookies, along with the session ID from step 1.
5. Server reads session ID from cookie.
6. Server matches session ID from a list in a database (or memory).
7. Server finds a match, reads variables which are now available on `$_SESSION` superglobal.
8. If PHP does not find a match with session ID, it will start a new session, and repeat the steps from 1-7.

SENDING E-MAIL

E-mail can be sent through a PHP script in amazingly easy fashion, using the `mail()` function.

```
boolean mail(string to, string subject, string message [, string addl_headers [,string
addl_params]])
```

The `mail()` function can send an e-mail with a subject and a message to one or several recipients. You can tailor many of the e-mail properties using the `addl_headers` parameter; you can even modify your SMTP server's behavior by passing extra flags via the `addl_params` parameter.

On the Unix platform, PHP's `mail()` function is dependent upon the `sendmail` MTA. If you're using an alternative MTA (e.g. `qmail`), you need to use that MTA's `sendmail` wrappers. PHP's Windows implementation of the function depends upon establishing a socket connection to an MTA designated by the SMTP configuration directive.

Sending a Plain-Text E-mail

Sending the simplest of e-mails is trivial using the `mail()` function, done using just the three required parameters, in addition to the fourth parameter which allows you to identify a sender.

```
<?php
mail("test@example.com", "This is a subject", "This is the mail
body", "From: admin@example.com\r\n");
?>
```

[Basic Project with session and cookie]

MySQL

MySQL is one of the most popular relational database systems being used on the Web today. It is freely available and easy to install. With PHP, you can connect to and manipulate databases. MySQL is the most popular database system used with PHP. The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows. Databases are useful for storing information categorically.

MySQL database server offers several advantages

- MySQL is easy to use, yet extremely powerful, fast, secure, and scalable.
- MySQL runs on a wide range of operating systems, including UNIX or Linux, Microsoft Windows, Apple Mac OS X, and others.
- MySQL supports standard SQL (Structured Query Language).
- MySQL is ideal database solution for both small and large applications.
- MySQL is developed, and distributed by Oracle Corporation.
- MySQL includes data security layers that protect sensitive data from intruders.

MySQL database stores data into tables like other relational database. A table is a collection of related data, and it is divided into rows and columns.

Each row in a table represents a data record that are inherently connected to each other such as information related to a particular product, whereas each column represents a specific field such as id, name, price etc.

Tbl_categories

ID	Name
1	Stationary
2	Electronic Item
3	Food Items

Tbl_products

ID	Category_id	Name	Price
101	1	Pen	25
102	1	Pencil	10
103	1	Copy	100
104	1	Marker	90
105	1	Masking Tape	120
106	2	Mobile Charger	300
107	2	USB Cable	100
108	2	Card Reader	50

There are three ways of working with MySQL and PHP

1. MySQLi (object-oriented)
2. MySQLi (procedural)
3. PDO

PHP MYSQL CONNECT TO A DATABASE

Database connection using MySQLi object oriented approach

Example

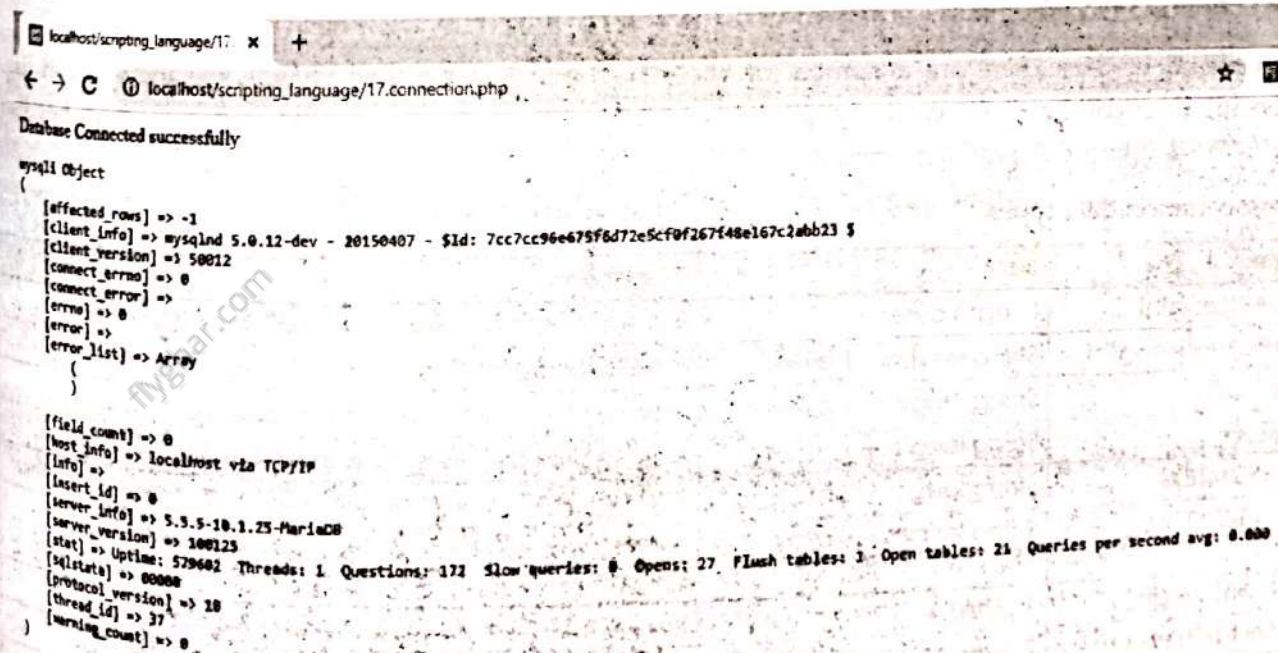
```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "db_scripting_language";

// Creating connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Checking connection
if ($conn->connect_errno != 0) {
    die("Connection failed: " . $conn->connect_error);
}

echo "Database Connected successfully";
echo '<pre>';
print_r($conn);
?>
```

Output



```
Database Connected successfully
mysqli Object
(
    [affected_rows] => -1
    [client_info] => mysqld 5.0.12-dev - 20150407 - $Id: 7cc7cc96e675f8d72e5cf0f267f48e167c2abb23 $
    [client_version] => 50012
    [connect_errno] => 0
    [connect_error] =>
    [errno] => 0
    [error] =>
    [error_list] => Array
        (
        )
    [field_count] => 0
    [host_info] => localhost via TCP/IP
    [info] =>
    [insert_id] => 0
    [server_info] => 5.5.5-10.1.25-MariaDB
    [server_version] => 100125
    [stat] => Uptime: 579602 Threads: 1 Questions: 172 Slow queries: 0 Opens: 27 Flush tables: 1 Open tables: 21 Queries per second avg: 0.000
    [sqlstate] => 00000
    [protocol_version] => 10
    [thread_id] => 37
    [warning_count] => 0
)
```

We can breakdown database connection into two step. First one to connect to MySQL and second one to select database for database operation.


```

<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbase = "db_scripting_language";
// Creating connection
$conn = new mysqli($servername, $username, $password);
// Checking connection
if ($conn->connect_errno != 0) {
    die("Connection failed: " . $conn->connect_error);
}
//Selecting database
$conn -> select_db($dbase);
echo "Database Connected successfully";
echo '<pre>';
print_r($conn);
?>

```

MYSQL DATA TYPES

Once you have identified all of the tables and columns that the database we need, you should determine each field's MySQL data type. When creating the database, MySQL requires that you define what sort of information each field will contain.

There are three primary categories, which is true for almost every database software:

- Text
- Numbers
- Dates and times

Within each of these, there are a number of variants, some of which are MySQL-specific you can use. Choosing your column types correctly not only dictates what information can be stored and how, but also affects the database's overall performance.

List of common data types.

Type	Size	Description
CHAR[Length]	Length bytes	A fixed-length field from 0 to 255 characters long.
VARCHAR[Length]	String length + 1 bytes	A fixed-length field from 0 to 255 characters long.
TINYTEXT	String length + 1 bytes	A string with a maximum length of 255 characters.
TEXT	String length + 2 bytes	A string with a maximum length of 65,535 characters.
MEDIUMTEXT	String length + 3 bytes	A string with a maximum length of 16,777,215 characters.
LONGTEXT	String length + 4 bytes	A string with a maximum length of 4,294,967,295 characters.
TINYINT[Length]	1 byte	Range of -128 to 127 or 0 to 255 unsigned.
SMALLINT[Length]	2 bytes	Range of -32,768 to 32,767 or 0 to 65535 unsigned.
MEDIUMINT[Length]	3 bytes	Range of -8,388,608 to 8,388,607 or 0 to 16,777,215 unsigned.

Type	Size	Description
INT[Length]	4 bytes	Range of -2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 unsigned.
BIGINT[Length]	8 bytes	Range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 or 0 to 18,446,744,073,709,551,615 unsigned.
FLOAT	4 bytes	A small number with a floating decimal point.
DOUBLE[Length, Decimals]	8 bytes	A large number with a floating decimal point.
DECIMAL[Length, Decimals]	Length + 1 or Length + 2 bytes	A DOUBLE stored as a string, allowing for a fixed decimal point.
DATE	3 bytes	In the format of YYYY-MM-DD.
DATETIME	8 bytes	In the format of YYYY-MM-DD HH:MM:SS.
TIMESTAMP	4 bytes	In the format of YYYYMMDDHHMMSS;
TIME	3 bytes	In the format of HH:MM:SS
ENUM	1 or 2 bytes	Short for enumeration, which means that each column can have one of several possible values.
SET	1, 2, 3, 4, or 8 bytes	Like ENUM except that each column can have more than one of several possible values.

Many of the types can take an optional Length attribute, limiting their size (the square brackets, [], indicate an optional parameter to be put in parentheses, while parentheses "()" themselves indicate required arguments). The number types can be UNSIGNED—limiting the column to positive numbers or zero—or be defined as ZEROFILL, which means that any extra room will be padded with zeroes (ZEROFILLs are also automatically UNSIGNED).

To choose your data types

- Identify whether a column should be a text, number, or date type.

This is normally an easy and obvious step. You will find that numbers such as ZIP codes and dollar amounts should be text fields if you include their corresponding punctuation (dollar signs, commas, and hyphens), but you'll get better results if you store them as numbers and address the formatting elsewhere.

- Choose the most appropriate subtype for each column.

For improved performance, keep in mind following considerations:

- Fixed-length fields (such as CHAR) are generally faster than variable-length fields (such as VARCHAR), but they also take up more disk space. See the side-bar for more information.
- The size of any field should be restricted to the smallest possible value, based upon what the largest possible input could be. For example, if the largest a number such as Client ID could be is in the hundreds, set the column as an unsigned three-digit SMALLINT (allowing for up to 999 values).
- You should keep in mind that if you insert a string five characters long into a CHAR (2) field, the final three characters will be truncated. This is true for any field in which the length is set (CHAR, VARCHAR, INT, etc.).

Set the maximum length for text and number columns as well as other attributes such as UNSIGNED

MySQL spatial data types

MySQL supports many spatial data types that contain various kinds of geometrical and geographical values.

Spatial Data Types	Description
GEOMETRY	A spatial value of any type
POINT	A point (a pair of X-Y coordinates)
LINESTRING	A curve (one or more POINT values)
POLYGON	A polygon
GEOMETRYCOLLECTION	A collection of GEOMETRY values
MULTILINESTRING	A collection of LINESTRING values
MULTIPOINT	A collection of POINT values
MULTIPOLYGON	A collection of POLYGON values

Difference between CHAR and VARCHAR**CHAR**

- It is used to store character string value of fixed length.
- The maximum no. of characters the data type can hold is 255 characters.
- It is 50% faster than VARCHAR.
- It uses static memory allocation.

VARCHAR

- It is used to store variable length alphanumeric data.
- The maximum this data type can hold is up to Pre-MySQL 5.0.3: 255 characters and Post-MySQL 5.0.3: 65,535 characters shared for the row.
- It is slower than CHAR.
- It uses dynamic memory allocation.

Creating database from PHP code.

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbase = "db_scripting_language";
// Creating connection
$conn = new mysqli($servername, $username, $password);
// Checking connection
if ($conn->connect_errno != 0) {
    die("Connection failed: " . $conn->connect_error);
}
```



```
// Create database
$sql = "CREATE DATABASE $database";
if ($connection->query($sql)) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . $connection->error;
}
//selecting database
$connection -> select_db($database);
//closing database connection
$connection->close();
?>
```

Creating Table in the database

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$database = "db_scripting_language";

// Creating connection
$connection = new mysqli($servername, $username, $password,$database);

// Checking connection
if ($connection->connect_errno != 0) {
    die("Connection failed: " . $connection->connect_error);
}

// Create table query
$sql = "CREATE TABLE tbl_products(
    ID INT(10) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30) NOT NULL,
    price decimal(10,2) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

//execute sql
if ($connection->query($sql)) {
    echo "Table created successfully";
} else {
    echo "Error creating table: " . $connection->error;
}
?>
```

Insert record into table

```

<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbase = "db_scripting_language";
// Creating connection
$connection = new mysqli($servername, $username, $password,$dbase);
// Checking connection
if ($connection->connect_errno != 0) {
    die("Connection failed: " . $connection->connect_error);
}
// insert into table
$sql = "INSERT INTO tbl_products (ID, name, price) VALUES (101, 'Pen', 25)";
//execute sql
if ($connection->query($sql)) {
    echo "Record inserted successfully";
} else {
    echo "Error creating database: " . $connection->error;
}
?>

```

Select list of record from table

```

<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbase = "db_scripting_language";
// Creating connection
$connection = new mysqli($servername, $username, $password,$dbase);
// Checking connection
if ($connection->connect_errno != 0) {
    die("Connection failed: " . $connection->connect_error);
}
// select all record from table
$sql = "SELECT * FROM tbl_products";
//execute sql and get result object in case of select query
$result = $connection->query($sql);
//check if result object contain any record
if ($result->num_rows > 0) {
    //initilize empty array to store data
    $data = [];
    //fetch single row from result object as an associative array
    while ($row = $result->fetch_assoc()) {
        //assign row to data array
        array_push($data, $row);
    }
}
echo '<pre>';
print_r($data);
//you can display this array as your requirement
?>

```


Select specific record with where clause.

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbase = "db_scripting_language";
// Creating connection
$conn = new mysqli($servername, $username, $password,$dbase);
// Checking connection
if ($conn->connect_errno != 0) {
    die("Connection failed: " . $conn->connect_error);
}
// select data from table
$sql = "SELECT * FROM tbl_products where ID=101";
//execute sql and get result object in case of select query
$result = $conn->query($sql);
//check if result object contain any record
if ($result->num_rows > 0) {
    $row = $result->fetch_assoc();
    print_r($row);
}
?>
```

Update record into table

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbase = "db_scripting_language";
// Creating connection
$conn = new mysqli($servername, $username, $password,$dbase);
// Checking connection
if ($conn->connect_errno != 0) {
    die("Connection failed: " . $conn->connect_error);
}
// update query
$sql = "UPDATE tbl_products SET name='Natraj Pencil',price=12.5 where id=102";
//execute sql and get result object in case of select query
$result = $conn->query($sql);
//check if result object contain any record
if ($result->affected_rows == 1) {
    echo "Record update successfully";
}
?>
```

Delete from table.

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "db_scripting_language";
// Creating connection
$conn = new mysqli($servername, $username, $password,$dbname);
// Checking connection
if ($conn->connect_errno != 0) {
    die("Connection failed: " . $conn->connect_error);
}
// delete table query
$sql = "DELETE FROM tbl_products WHERE id=102";
//execute sql and get result object in case of select query
$result = $conn->query($sql);
//check if result object contain any record

if ($result->affected_rows == 1) {
    echo "Record deleted successfully";
}
?>
```

MYSQL AGGREGATE FUNCTIONS (SUM, AVG, COUNT ETC)

MYSQL aggregate function retrieves a single value after performing a calculation on set of values. It typically operates on the values in columns so these functions are also called column functions. They perform a calculation on the values in a set of selected rows and a QUERY that contains one or more aggregate functions. It is typically referred to as a summary query.

Different organization levels have different information requirements. Top levels managers are usually interested in knowing whole figures and not necessary the individual details. So, aggregate functions facilitate to produce summarized data from our database.

The five (5) aggregate functions namely;

- COUNT
- SUM
- AVG
- MIN
- MAX

COUNT Function

The COUNT function returns the total number of values in the specified field. It works on both numeric and non-numeric data types. All aggregate functions by default exclude nulls values before working on the data.

COUNT (*) is a special implementation of the COUNT function that returns the count of all the rows in a specified table. COUNT (*) also considers Nulls and duplicates.

SUM function

The aggregate sum function is used to total the non-null values in the expression.

SUM([ALL | DISTINCT] expression)

MIN function

The MIN function returns the smallest value in the specified table field.

MIN(column_name)

MAX function

Just as the name suggests, the MAX function is the opposite of the MIN function. It returns the largest value from the specified table field.

MAX(column_name)

AVG function

MySQL AVG function returns the average of the values in a specified column. Just like the SUM function, it works only on numeric data types.

AVG(column_name)

Example

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$database = "db_scripting_language";
// Creating connection
$conn = new mysqli($servername, $username, $password, $database);
// Checking connection
if ($conn->connect_errno != 0) {
    die("Connection failed: " . $conn->connect_error);
}
// sql query to use aggregate function like count,sum,min,max,avg
$sql = "SELECT count(*) as total_product ,sum(price) as sum_price ,avg(price) as
avg_price,min(price) as min_price,max(price) as max_price FROM tbl_products";
//execute sql and get result object in case of select query
$result = $conn->query($sql);
//check if result object contain any record
if ($result->num_rows > 0) {
    $row = $result->fetch_assoc();
    print_r($row);
}
?>
```

Output

localhost/scripting_language/17 x localhost / 127.0.0.1 / db_scripting x +

localhost/scripting_language/17.aggregate.php

Array ([total_product] => 8 [sum_price] => 795.00 [avg_price] => 99.375000 [min_price] => 10.00 [max_price] => 300.00)

MYSQL ORDER BY AND GROUP BY CLAUSE

Group by clause

GROUP BY clause is added to SQL because aggregate function like SUM returns the aggregate of all column values every time they are called, and without the GROUP BY clause it is impossible to find the sum for each individual group of column values. In many cases, we want to apply the aggregate functions to subgroups of tuples in a relation rather than all record. Each subgroup of tuples consists of the set of tuples that have the same value for the grouping attribute(s). The function is applied to each subgroup independently. SQL has a GROUP BY-clause for specifying the grouping attributes, which must also appear in the SELECT clause.

Order by clause

The ORDER BY clause is used to sort the tuples in a query result based on the values of some attribute(s). The default order is in ascending order of values. We can specify the keyword DESC if we want a descending order; the keyword ASC can be used to explicitly specify ascending order, even though it is the default. When the ORDER BY clause is missing, the result rows from a query have no defined order. It defines the ordering of rows based on columns from the SELECT clause.

Example: To get no of item related to each category from products table, we can use group by clause

```
SELECT category_id,COUNT(name) FROM tbl_products GROUP BY category_id ORDER BY
category_id DESC
```

PHP Example

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbase = "db_scripting_language";
// Creating connection
$conn = new mysqli($servername, $username, $password,$dbase);
// Checking connection
if ($conn->connect_errno != 0) {
    die("Connection failed: " . $conn->connect_error);
}
// query
$sql = "SELECT category_id,COUNT(name) FROM tbl_products GROUP BY category_id ORDER
BY category_id DESC";
//execute sql and get result object in case of select query
$result = $conn->query($sql);
//check if result object contain any record
```



```

if ($result->num_rows > 0) {
    //initilize empty array to store data
    $data = [];
    //fetch single row from result object as an associative array
    while ($row = $result->fetch_assoc()) {
        //assign row to data array
        array_push($data, $row);
    }
}
echo '<pre>';
print_r($data);
//you can display this array as your requirement
?>

```

Output

localhost/scripting_language/17 x localhost / 127.0.0.1 / db_scriptin x +

localhost/scripting_language/17.9.order_group_by.php

Array

```

(
    [0] => Array
        (
            [category_id] => 2
            [COUNT(name)] => 3
        )

    [1] => Array
        (
            [category_id] => 1
            [COUNT(name)] => 5
        )

```

MYSQL SUBQUERIES

A sub query is a select query that is contained inside another query. The inner select query is usually used to determine the results of the outer select query.

Subqueries to find to category details of product we have.

```

SELECT * FROM tbl_categories WHERE ID IN (SELECT DISTINCT category_id FROM
tbl_products WHERE category IS NOT NULL );

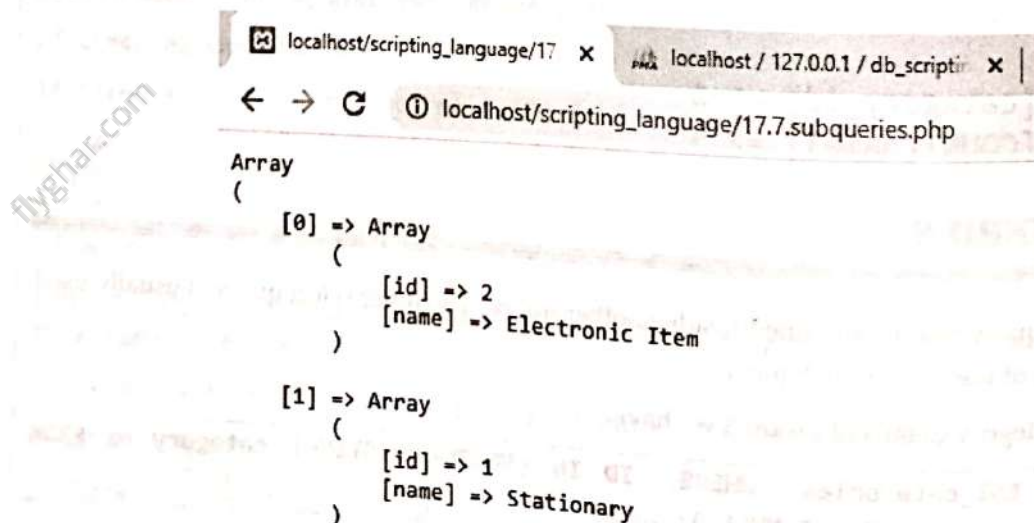
```

You can also implement with PHP.

Example

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbase = "db_scripting_language";
// Creating connection
$conn = new mysqli($servername, $username, $password, $dbase);
// Checking connection
if ($conn->connect_errno != 0) {
    die("Connection failed: " . $conn->connect_error);
}
// Subqueries to find to category details of product we have.
$sql = "SELECT * FROM tbl_categories WHERE ID IN (SELECT DISTINCT category_id
FROM tbl_products WHERE category_id IS NOT NULL)";
//execute sql and get result object in case of select query
$result = $conn->query($sql);
//check if result object contain any record
if ($result->num_rows > 0) {
    //initilize empty array to store data
    $data = [];
    //fetch single row from result object as an associative array
    while ($row = $result->fetch_assoc()) {
        //assign row to data array
        array_push($data, $row);
    }
}
echo '<pre>';
print_r($data);
//you can display this array as your requirement
?>
```

Output



```
localhost/scripting_language/17 x localhost / 127.0.0.1 / db_scripting x |
localhost/scripting_language/17.7.subqueries.php
Array
(
    [0] => Array
        (
            [id] => 2
            [name] => Electronic Item
        )
    [1] => Array
        (
            [id] => 1
            [name] => Stationary
        )
)
```


MYSQL JOINS

Joins help retrieving data from two or more database tables. The tables are mutually related using primary and foreign keys.

INNER JOIN

The inner JOIN is used to return rows from both tables that satisfy the given condition.

Suppose, you want to get list of product with their category name instead of category_id. You can simply use an INNER JOIN for that, which returns rows from both tables that satisfy with given conditions.

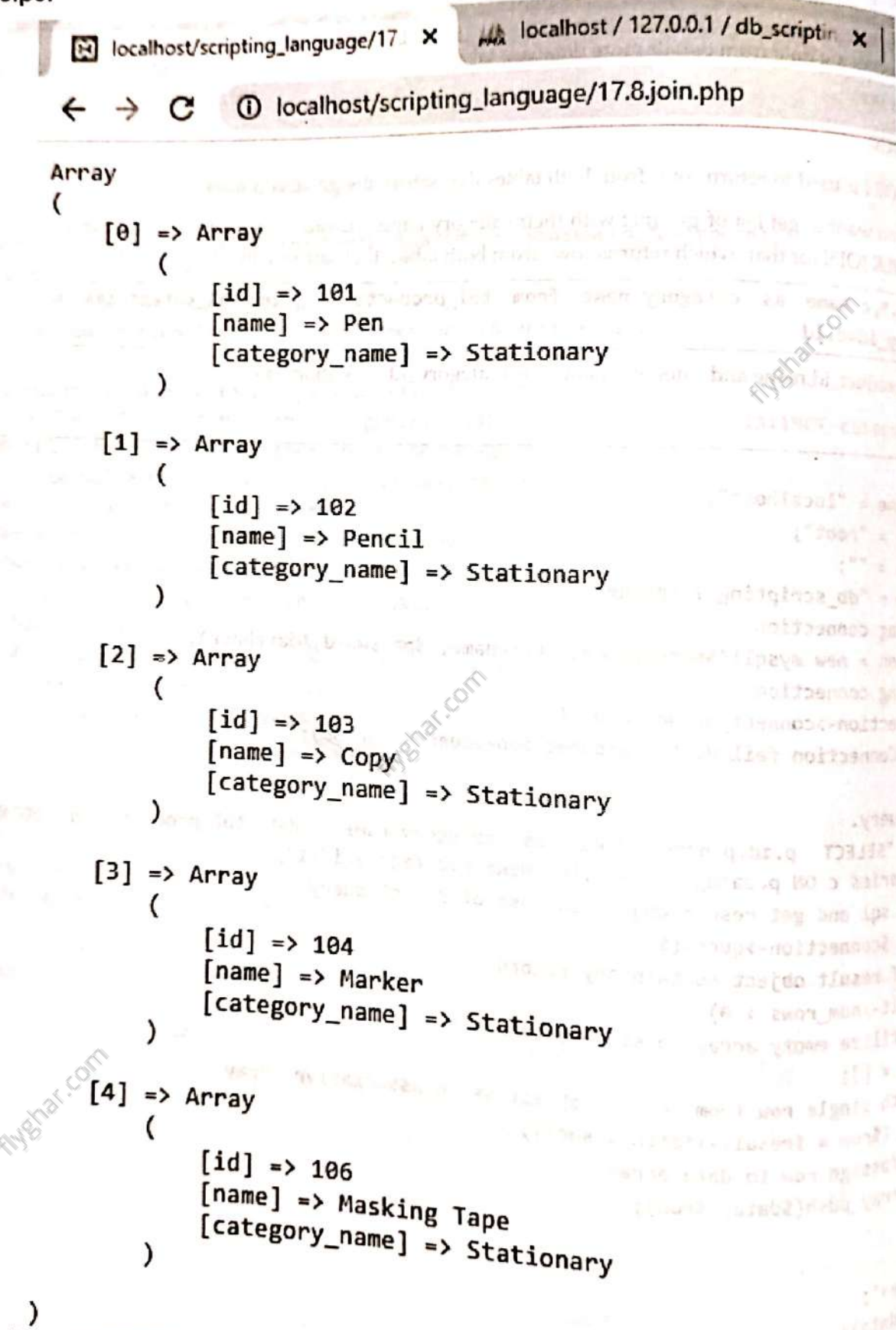
```
select p.*,c.name as category_name from tbl_products p join tbl_categories c on
p.category_id=c.id
```

Get list of product_id,name and category_name with category_id=1(stationary)

Example

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbase = "db_scripting_language";
// Creating connection
$conn = new mysqli($servername, $username, $password,$dbase);
// Checking connection
if ($conn->connect_errno != 0) {
    die("Connection failed: " . $conn->connect_error);
}
// Join query.
$sql = "SELECT p.id,p.name,c.name as category_name FROM tbl_products p JOIN
tbl_categories c ON p.category_id=c.id WHERE p.category_id=1";
//execute sql and get result object in case of select query
$result = $conn->query($sql);
//check if result object contain any record
if ($result->num_rows > 0) {
    //initilize empty array to store data
    $data = [];
    //fetch single row from result object as an associative array
    while ($row = $result->fetch_assoc()) {
        //assign row to data array
        array_push($data, $row);
    }
}
echo '<pre>';
print_r($data);
//you can display this array as your requirement
?>
```

Output



```

Array
(
    [0] => Array
        (
            [id] => 101
            [name] => Pen
            [category_name] => Stationary
        )

    [1] => Array
        (
            [id] => 102
            [name] => Pencil
            [category_name] => Stationary
        )

    [2] => Array
        (
            [id] => 103
            [name] => Copy
            [category_name] => Stationary
        )

    [3] => Array
        (
            [id] => 104
            [name] => Marker
            [category_name] => Stationary
        )

    [4] => Array
        (
            [id] => 106
            [name] => Masking Tape
            [category_name] => Stationary
        )

)

```




DISCUSSION EXERCISE

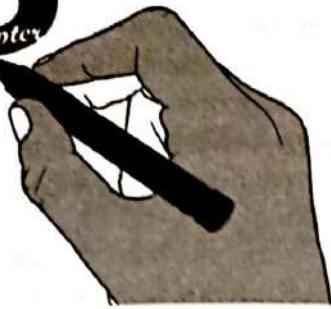
1. What is server side scripting language? What are the advantages of server side scripting language?
2. What is PHP? Explain history of PHP.
3. Describe is AMPP with example.
4. Describe basic PHP syntax.
5. Describe comments in PHP with example.
6. Describe output statement with example in PHP.
7. Difference between echo and print.
8. What is variable? Write rules to variable in PHP.
9. What is super global variable? Describe various super global variable in PHP.
10. Describe various data types supported by PHP.
11. What is constant? How to create constant in PHP.
12. What is type casting? Describe different casting operator in PHP.
13. Describe different PHP operator with example.
14. What is ternary operator? Write a PHP program to demonstrate use of ternary operator.
15. Describe different control structure using PHP.
16. What is loop? Describe different types of loop in PHP.
17. What is use of foreach loop in PHP? Describe with example.
18. What is array, Describe different type of array in PHP?
19. Difference between numeric & associative array.
20. How to define function in PHP.
21. Describe different type of function in PHP.
22. Describe function overloading in case of PHP.
23. Write a PHP code to extract dynamic parameter from function.
24. Describe variable scope in PHP with example.
25. Describe sorting, merging, extracting function of array in PHP.
26. Describe form handling in PHP.
27. Differentiate between get and post method.
28. Describe different form validation technique in PHP.
29. What is UNIX timestamp? Describe time function with example.
30. Differentiate between time and strtotime in PHP.
31. Describe date function with different format specifier.
32. Describe file inclusion in PHP? Differentiate between require and include function.

33. Differentiate between `require_once` and `include_once`.
34. Describe `fopen` function with different opening mode.
35. What different file handling function in PHP with example.
36. What are different types of error in PHP describe with example.
37. How to change PHP configuration at runtime.
38. Describe error handling mechanism in PHP.
39. How to upload file in PHP, Write a PHP program to upload your CV document into server.
40. What is session? Describe how session works.
41. Write PHP program to sending email.
42. What is cookie? Describe different type of cookie with example.
43. Difference between session and cookie.
44. Describe different session function.
45. Describe `setcookie` function with proper example.
46. What is MySQL? Describe feature of MySQL database.
47. Describe different way of database connection using PHP.
48. Describe different MySQL datatypes.
49. Write a process of data insert into with example.
50. Write a process of data selection into with example.
51. Write a process of data update with example.
52. Write a process of data delete with example.
53. Describe different aggregate function with example.
54. Describe group by function with example.
55. Describe subqueries with example.
56. Describe MySQL join with example.

□□□

3

Chapter



ADVANCED SERVER SIDE SCRIPTING



CHAPTER OUTLINE

After comprehensive study of this chapter, you will be able to:

- **Object Oriented Programming in PHP:** Classes and Objects, Defining and Using properties and methods, Constructors and Destructors, Method Overriding, Encapsulation, Inheritance, Polymorphism, Static Members, Exception Handling
- **AJAX (Asynchronous JavaScript and XML):** Using PHP, Using PHP + MySQL.
- **JQuery:** Playing with elements, Hiding and Unhiding Images, JQuery UI
- **JOOMLA:** Introduction To CMS, Installation, Handling Joomla Back End. Customization In Joomla, Introduction to Extensions, Installation and Uses Of Extensions in Joomla, Template Development in Joomla, Artiseer (IDE), Module Development in Joomla, Component Development in Joomla, Introduction to MVC (Model, View and Controller)
- **Word Press Administrator Level:** Theme Integration, Creating Pages, Managing Posts, Managing Widgets.

OBJECT ORIENTED PROGRAMMING IN PHP (OOP IN PHP)

Object oriented programming is an approach to program design and development which attempts to eliminate pitfalls of conventional programming methods by incorporating best features of structured programming along with new features such as object, class, encapsulation etc. It is an approach that provides a way of modularizing program by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand. OOP allows us to decompose a problem in to number of entities called objects and then build data and functions known as methods around these entities. It ties data more closely to the functions that operate on it and protects it from modifications by other functions. It is used to develop software applications which could be any type like it could be web based application, windows based application. In object oriented programming, everything will be around the objects and class. By using OOP in PHP you can create modular web application. By using OOP in PHP we can perform any activity in the object model structure. OOP has following characteristics:

- It emphasize on data rather than procedure.
- It follows bottom up approach in application design.
- Application is decomposed into objects.
- Methods that operate on the data of an object are tied together in the data structure.
- Concept of encapsulation is used so that data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through methods message passing mechanism.
- It follows concepts of inheritance and polymorphism.

Advantage of Object Oriented Programming

There are various advantage of OOP over conventional or procedural programming. Following are some of the basic advantages of using OOP techniques.

Reusability of your code: If you use OOP technique for creating your application then it will gives you a greater reusability. For example, if you have created database class at one place then you can use the same database class in your application.

Easy to Maintain: Application developed using OOP techniques are easier to maintain than normal programming. For example in database class, if you need to change the username and password then we change username and password using connection class and this will automatically change all part of database connection.

Good Level of Abstraction: Abstraction means making something hidden. By using OOP technique you are abstracting your business logic from implementation. It will provide you greater ease. For example using OOP you will only concern about creating object of database not more details about it.

OBJECT & CLASS

Any real world entities or instances that have unique features are called objects. These are the basic runtime entities in an object oriented system. They may represent a person, place, laptop, mobile, car, account, bonus, premium or any item that a program can handle. In this world every object has two thing properties and behaviors. For example your television has property size, color, and has behavior like turn on, turn off. If you are able to find properties and behaviors of real object. Then it will be very easy for you

to work with Object Oriented Programming. If you observe carefully then you can find that every object has some property and behavior from other object.

Class is user defined data type that consists of entire set of data together with code of an object. It represent group of objects that share common properties, behavior and relationships. A class may be thought as a data type and an object as variable of that data type. Once class has been created, we can create any number of objects belonging to that class. So a class is thus a blue print or template which describes characteristics of objects of similar type. For example: desktop, laptop, palmtop are the members of class computer.

Creating class and object in PHP

You can use 'class' (small letter) keyword to define class in PHP.

```
classclassName {
    //Properties
    //Methods
}
```

So let us create a class for interest calculator and define its properties like rate, capital, duration and behavior like calculate interest.

```
classinterestCalculator
{
    var $rate;
    var $duration;
    var $capital;

    functioncalculateInterest()
    {
        return ($this>
            rate*$this>
            duration*$>
            capital)/100;
    }
}
```

Creating object of the class is very easy in PHP. You can create object of class with the help of new keyword. Following is very basic example of creation of object of your class interest calculator:

```
$calculator = new interestCalculator();
```

In above declaration you are creating object of your class interestCalculator in variable \$calculator. Now your variable \$calculator is an object of class interestCalculator. Next step is to set property or variable of object calculator and perform calculation of interest.

```
$calculator = new InterestCalculator()
$calculator->rate = 3;
$calculator->duration = 2;
$calculator->capital = 300;
echo $calculator->calculateInterest();
```

Here object of your class interestCalculator is your PHP variable \$calculator. In next 3 lines of above code you are setting properties of class. You can access property of class with >. So in above code rate property is set using \$calculator->rate = 3;

Let's take another example of student:

Example

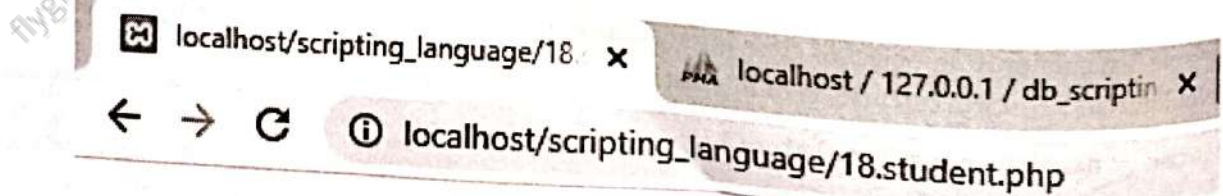
```
<?php
class Student{
//define properties/attribute
var $name,$section,$faculty;
//define behaviour/method
function welcomeMessage(){
return '<br>Hi, ' . $this->get('name') .
' Welcome to section ' . $this->get('section') . ' of ' . $this->get('faculty');
}
function set($key,$value){
    $this->$key = $value;
}
function get($key){
return $this->$key;
}
}

$ram = new Student();
$ram->set('name','Ram Bahadur');
$ram->set('section','A');
$ram->set('faculty','BCA');
echo $ram->welcomeMessage();

$hari = new Student();
$hari->set('name','Hari Krishna');
$hari->set('section','A');
$hari->set('faculty','BA');
echo $hari->welcomeMessage();
?>
```

Above code will display following output. We have created multiple object of same class. Here \$ram and \$hari are different object of student class.

Output



Hi, Ram Bahadur Welcome to section A of BCA
Hi, Hari Krishna Welcome to section A of BA

\$this keyword

The `$this` keyword indicates that if we use own methods and properties the class then allows to have access to them within the class's scope. The `$this` keyword is system defined object variable of the class. For the both object of Student class `$this` object is different.

PROPERTIES & METHODS

Properties

You could think of class properties as variables that are used to hold information about the object. In the above example, we've defined three properties – name, section, and faculty. In most cases, class properties are accessed via instantiated objects.

Methods

We can think of class methods as functions that perform specific actions associated with objects. In most cases, they are used to access and manipulate object properties and perform related operations. In the above example, we've defined the `welcomeMessage` method, which returns the welcome message associated with the object.

CONSTRUCTOR & DESTRUCTOR

Constructor

Constructor is a special member function defined in PHP class which initializes object of its class and it is automatically called when you create object of that class. Constructor enables an object to initialize itself when it is created. As soon as you will write `$object = new YourClass()` your constructor function of the class will be executed. You can't create constructor private. You can also create constructor by defining magic function `__construct`. In case of PHP we can also create constructor with class name too. But class name constructor will have less priority over default constructor.

```
<?php
class Student{
var $name,$section,$faculty;
//main constructor
function __construct($n,$s,$f){
    $this->name = $n;
    $this->section = $s;
    $this->faculty = $f;
}
function welcomeMessage(){
return '<br>Hi, ' . $this->name .
    ', Welcome to section ' . $this->section . ' of ' . $this->faculty;
}
public function __destruct() {
echo "Object is destroyed";
}
}
$ram = new Student('Ram Bahadur','A','BCA');
$hari = new Student('Hari Krishna','B','BA');
echo $ram->welcomeMessage();
echo $hari->welcomeMessage();
?>
```

Destructor

A destructor is a special member function of a class that is executed when an object of that class is destroyed. PHP destructor allows you to clean up resources before PHP releases the object from the memory. The `__destruct()` method is called when the object is destroyed. It accepts no arguments and is commonly used to perform any cleanup operations such as closing a database connection.

DESTROYING OBJECTS

By default objects have been automatically destroyed at the end of the script they were created in, thanks to PHP's automatic garbage collection. However, you will almost certainly want to arbitrarily delete objects at some point in time, and this is accomplished using `unset()` in the same way as you would delete an ordinary variable. Calling `unset()` on an object will call its destructor before deleting the object, as you would expect.

Example

```
<?php
class Student{

var $name,$section,$faculty;

function __construct($n,$s,$f){
    $this->name = $n;
    $this->section = $s;
    $this->faculty = $f;
}

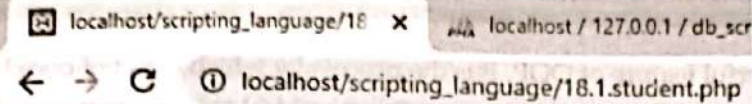
function Student(){
echo "This is function constructor";
}

function welcomeMessage(){
return '<br>Hi, ' . $this->name .
    ', Welcome to section ' . $this->section . ' of ' . $this->faculty;
}

public function __destruct() {
echo "<BR>Object is destroyed";
}

$ram = new Student('Ram Bahadur','A','BCA');
$hari = new Student('Hari Krishna','B','BA');
$sita = new Student('SitaRana','A','BCA');
echo $ram->welcomeMessage();
unset($ram);
echo $hari->welcomeMessage();
echo $sita->welcomeMessage();
?>
```


Output



Hi, Ram Bahadur, Welcome to section A of BCA

Object is destroyed

Hi, Hari Krishna, Welcome to section B of BA

Hi, Sita Rana, Welcome to section A of BCA

Object is destroyed

Object is destroyed

ENCAPSULATION

The wrapping up of data and methods into a single unit called class is known as encapsulation. It is an ability to bind data and functions together in a place and hide or prevent the data from unauthorized use from other parts of the program. Through encapsulation, data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it. These methods provide an interface between objects data and program. Encapsulation is used to safe data or information in an object from other it means encapsulation is mainly used for protection and data hiding purpose. Whenever data modified in one function affects the other functions, it causes a lot of problems in any software. To overcome this problem, object-oriented programming in PHP uses the concept of Encapsulation. Outside, Class can't access private method of other class. That's why encapsulation is known as data hiding is the main advantage for encapsulation. Second advantage of encapsulation is you can make the class read only or write only by providing setter or getter method.

Advantages of Encapsulation

- **Data Hiding and Abstraction:** Unnecessary details, internal representation and implementation are hidden from the end-users for protection of data structure and the Class. Data access is prohibited from members of other classes by creating private methods. It protects the internal state of any object by keeping member variables private and preventing any inconsistent state. It is the enclosing of data and related operations into that object.
- **Data security:** Encapsulation helps in making data very robust and secured, as the data and member functions are wrapped together to form an object. All the tasks are done inside without any external worry and it also makes life very easy.
- **Reduces complexity:** Encapsulation helps in reducing development complexity of the software by hiding the details of implementation and exposing the methods or operations.
- **Reusability:** There are instances, you don't have to re-write same functionality that you inherited from the parent class.
- **Reliability:** You can make the class read-only or write-only by writing SET or GET methods.
- **Easier testing of code:** Encapsulated PHP code is easy to test as the functions used for testing child class ensures the testing of parent class functions also.
- **Increased flexibility:** Class variables can be accessed by GET or SET methods increasing flexibility. It is easy to maintain as the internal implementation can be changed without changing the code.

INHERITANCE

Inheritance is the powerful feature of OOP. It is the process by which object of one class acquire the properties of objects of another class. It supports the concept of hierarchical classification as well as idea of reusability. It allows you to create a new class that reuses the properties and methods from an existing class. A class that inherits from another class is called subclass (also a child class or a derived class). The class from which the subclass inherits is known as parent class (also a super class or a based class). Besides inheriting properties and methods from the parent class, a subclass class can have additional properties and methods.

Inheritance is very useful if you want to create several similar classes. You put the common properties and methods in the parent class and specific properties and methods in the child classes. It helps to avoid duplicate code that implements the same properties and method in many classes. We use extends keyword to inherit from a parent class.

```
classParentClass {
// properties and methods here
}
classChildClass extends ParentClass {
// additional properties and methods here
}
```

Here we've created a class, ParentClass, then created another class, ChildClass, that inherits from ParentClass. ChildClass inherits all the properties and methods of ParentClass, and it can also add its own properties and methods.

In the following example we have the parent class Vehicle. We are creating a child class Car which is inherited from the parent class.

Example

```
<?php
class Vehicle {
private $color;
private $brand;

public function setColor($color) {
    $this->color = $color;
}

public function getColor() {
    return $this->color;
}

public function setBrand($brand) {
    $this->brand = $brand;
}

public function getBrand() {
    return $this->brand;
}
}
```



```

class Car extends Vehicle {
    private $steeringWheel;

    public function setSteeringWheel($steeringWheel) {
        $this->steeringWheel = $steeringWheel;
    }

    public function getSteeringWheel() {
        return $this->steeringWheel;
    }
}

//object of the child class
$obj = new Car();

//set property
$obj->setColor("Black");
$obj->setColor("KIA");
$obj->setSteeringWheel("Leather");

//get property
$color = $obj->getColor();
$brand = $obj->getBrand();
$steeringWheel = $obj->getSteeringWheel();

printf("Color = %s, Brand = %s and SteeringWheel = %s", $color,$brand,$steeringWheel);
?>

```

Output

localhost/scripting_language/18 x localhost / 127.0.0.1 / db_scriptin x

← → ↻ ⓘ localhost/scripting_language/18.inheritance.php

Color = KIA, Brand = and SteeringWheel = Leather

POLYMORPHISM

Polymorphism is derived from two Greek words. Poly (meaning many) and morph (meaning forms). Polymorphism is one of the PHP Object Oriented Programming (OOP) features. In general, polymorphism means the ability to have many forms. For example: an operation may exhibit different behavior in different instances. In other words, "Polymorphism describes a pattern in Object Oriented Programming in which a class has varying functionality while sharing common interfaces."

There are two types of Polymorphism; they are:

- Compile time (function overloading)
- Run time (function overriding)

Compile time Polymorphism

But PHP "does not support" compile time polymorphism, which means function overloading and operator overloading.

Runtime Polymorphism

The Runtime polymorphism means that a decision is made at runtime (not compile time) or we can say we can have multiple subtype implements for a super class, function overloading is an example of runtime polymorphism. I will first describe function overloading. When we create a function in a derived class with the same signature (in other words a function has the same name, the same number of arguments and the same type of arguments) as a function in its parent class then it is called method overriding.

METHOD OVERRIDING

When a child class has a method with the same name as the parent class it is called the overriding of the method of the parent class. It is the use of two or more methods having same name with same signature but defined one in base class and another in derived class. When that method is called by an object of the child class then, the child class method is executed instead of the parent class method.

Example

```
<?php
class Vehicle {
    private $color;
    private $brand;

    public function setColor($color) {
        $this->color = $color;
    }

    public function getColor() {
        return $this->color;
    }

    public function setBrand($brand) {
        $this->brand = $brand;
    }

    public function getBrand() {
        return $this->brand;
    }

    function displayInformation(){
        return "Parent :: This is vehical information with color and brand";
    }
}

class Car extends Vehicle {
    private $steeringWheel;
    public function setSteeringWheel($steeringWheel) {
        $this->steeringWheel = $steeringWheel;
    }
}
```



```

    }

    public function getSteeringWheel() {
        return $this->steeringWheel;
    }

    function displayInformation(){
        return "Child:: This is vehical information with steeringWheel<br> " .
        parent::displayInformation();
    }
}

//object of the child class
$obj = new Car();

//set property
$obj->setColor("Black");
$obj->setColor("KIA");
$obj->setSteeringWheel("Leather");

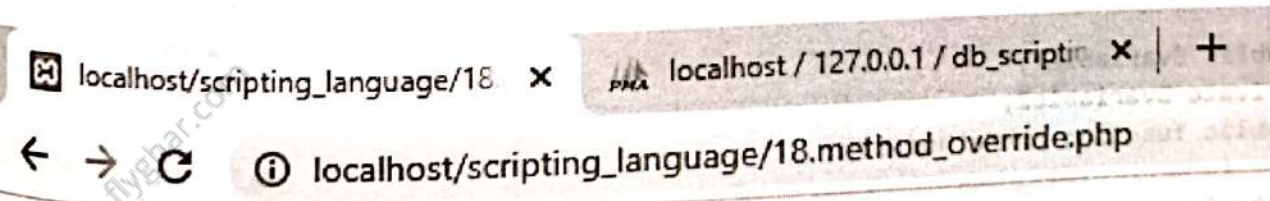
//get property
$color = $obj->getColor();
$brand = $obj->getBrand();
$steeringWheel = $obj->getSteeringWheel();

echo $obj->displayInformation();
echo '<br>';
printf("Color = %s, Brand = %s and SteeringWheel = %s", $color,$brand,
$steeringWheel);

?>

```

Output



Child:: This is vehical information with steeringWheel
 Parent :: This is vehical information with color and brand
 Color = KIA, Brand = and SteeringWheel = Leather

In this example we have created method `displayInformation()` into both parent and child class, which is method override. In case of method overriding we can directly call parent class method into child class method using 'parent' keyword. We call like: `parent::displayInformation()`; inside child method.

Visibility in PHP Classes

You are ever noticed that every method has their visibility in PHP. Public, private and protected are three type of visibility in php classes available for controlling access of your class variable and property. Every visibility level has it own rights.

Public Visibility

If we set any method as public then it means it can be accessible from anywhere either from inside the class, outside the class and in child class. There are no any limitation for public access.

```
<?php
classdemoClass
{
public $variable1;
public $variable2;

public function demoMethod()
{
}
}
$obj1 = new demoClass();
echo $obj1->variable1;//accessible from outside
$obj1->demoMethod();//public method of the class demoClass
?>
```

So in above example you are notice that everything is open, there is no any limitation. Variable and method are accessible from outside the class. By default every method and properties are public in PHP.

Private Visibility

If we set any method as private then it means it can be accessible within that class only it can't accessible from outside of class or in child class. You can access property and method within class by using \$this keyword. Private method or property in PHP are used to set restriction over class which means you don't want that outside of classes can access the method.

```
<?php
Class test
{
public $variable1;
private $variable2;
public function pubMethod($a)
{
echo $a;
}
private function privMethod($b)
{
echo $b;
}
public function pubPrivMethod()
{
$this->variable2 = 1;
}
```



```

$this->privMethod(1);
}
}
$objT = new test();
$objT->variable1 = 3;//Works fine due to public visibility
$objT->variable2 = 1;//Throw fatal error of visibility because variable is set to
private
$objT->pubMethod("test");//give output : "test"
$objT->privMethod(1);//Fatal error of visibility because method is set to private
$objT->pubPrivMethod();//Within this method private function privMethod and
variable variable2 is called using $this variable.
?>

```

Protected Visibility

Method or variable with protected visibility can only be access within the class itself and in the child class. Protected will be used in the process of inheritance.

```

<?php
class parent
{
protected $var1;
public $var2
protected function demoParent()
{
echo "this is demo";
}
}
class child extends parent
{
public function demoChild()
{
$this->demoParent(); //will work because it
}
}
$objParent = new parent();
$objParent->demoParent();//Throw error
?>

```

STATIC MEMBERS

Static methods and properties

In certain cases, it is better to approach methods and properties of a class without the need to create an object out of the class. This can be achieved by defining the methods and properties of a class as static. In the following example, the class "Utilis" has a static public property with the name of \$numCars and so the variable name is preceded by both the static and the public keywords.

```

classUtilis {
    // static methods and properties are defined with the static keyword.
    static public $numCars = 0;
}

```

In the same manner that we used the `$this` keyword to approach the class's own properties and methods from within the class, we use the reserved keyword `self` to approach static methods and properties.

In the example given below, the method `addToNumCars()` gets and sets the number of cars from the static `$numCars` property within the same class by using the "self" keyword.

Example

```

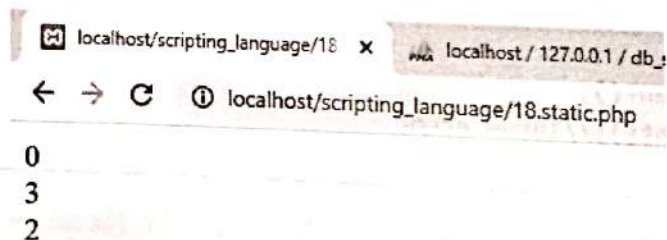
<?php
classUtilis {
    static public $numCars = 0;
    static public function addToNumCars($num)
    {
        $num = (int)$num;
        self::$numCars += $num;
    }
}
echoUtilis::$numCars;
echo '<br>';
Utilis::addToNumCars(3);
echoUtilis::$numCars;
echo '<br>';

Utilis::addToNumCars(-1);
echoUtilis::$numCars;
echo '<br>';

?>

```

Output



```

localhost/scripting_language/18 x localhost / 127.0.0.1 / db_
localhost/scripting_language/18.static.php
0
3
2

```

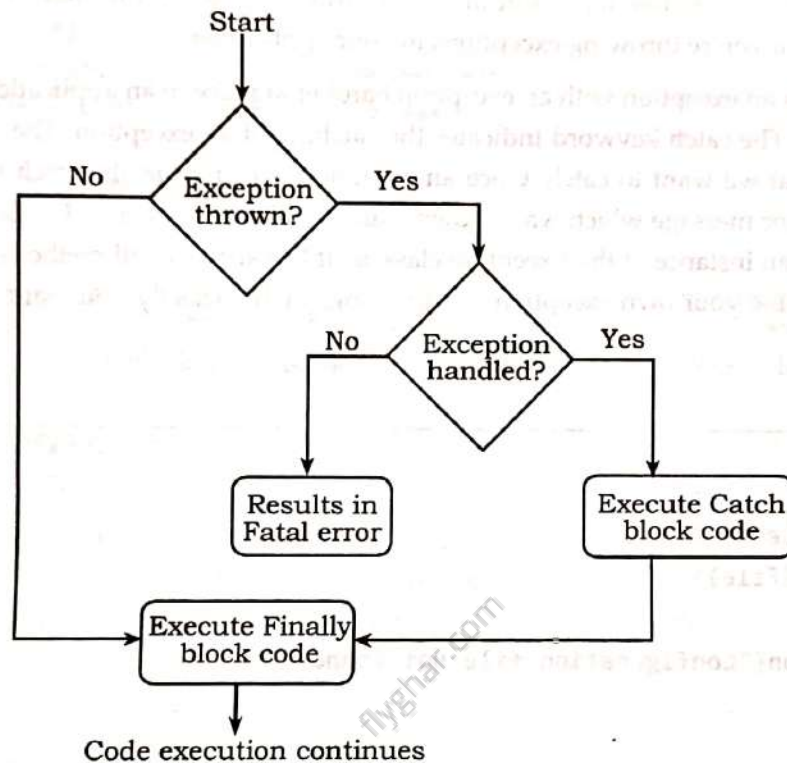
EXCEPTION HANDLING

An exception is an abnormal condition that may arise in a program at runtime. Thus it can be considered as runtime error. PHP introduces a new error model which allows to throw and catch exceptions in an application, which is a better way of handling errors than what we had in older versions of PHP.

It's important to note here that exception handling is different than error handling. In error handling, we can use the `set_error_handler` function to set our custom error handling function so that whenever an error

is triggered, it calls our custom error handling function. In that way, you can control errors. Generally, however, certain kinds of errors are unrecoverable and stop program execution.

On the other hand, exceptions are thrown deliberately by the code, and it's expected that they'll be caught at some point in your application. So we can say that exceptions are recoverable as opposed to certain errors which are unrecoverable. If an exception which is thrown is caught somewhere in your application, program execution continues from the point where the exception was caught. And an exception which is not caught anywhere in your application results in an error, thus halting program execution.



Exception handling is a key coding skill, and you should consider how exceptions will be handled while developing your applications. This will help you detect and recover from unexpected errors in your application.

Exceptions can be thrown and caught by using the PHP try and catch blocks. You are responsible for throwing exceptions when something occurs which is not expected.

```

// code before the try-catch block
try {
    // code

    // if something is not as expected
    // throw exception using the "throw" keyword

    // code, it won't be executed if the above exception is thrown
} catch (Exception $exception) {
    // exception is raised and it'll be handled here
    // $exception->getMessage() contains the error message
}
// code after the try-catch block, will always be executed
  
```

Most of the time, when you're dealing with exceptions, you'll end up using a pattern, as shown in the above example. You can also use the finally block along with the try and catch blocks.

THROWING AN EXCEPTION

An exception might be thrown by a function that you call, or you can use the `throw` keyword to throw an exception manually. For example, you might validate some input before performing any operation, and throw an exception if the data is not valid.

It's important to note here that if you throw an exception but you haven't defined the catch block which is supposed to handle that exception, it'll result in a fatal error. So you need to make sure that you always define the catch block if you're throwing exceptions in your application.

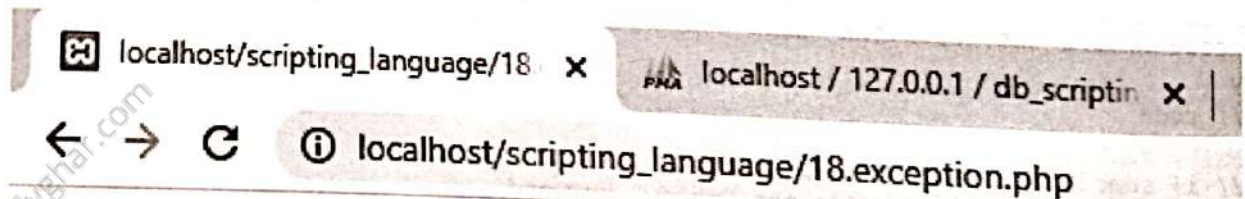
An application catches an exception with an exception handler at place in an application where we want to handle the exception. The `catch` keyword indicates the catching of an exception. The catch block specifies the exception type that we want to catch. Once an exception is caught in the catch block, the `Exception` object contains the error message which was thrown using the `throw` keyword. The `$exception` variable in the above example is an instance of the `Exception` class, so it has access to all methods of that class. In this block, you should define your own exception handling logic, what exactly you want to do with the error you catch.

Example

```
<?php
try {
    $file = "myfile.php";
    if (!file_exists($file))
    {
        throw new Exception("Configuration file not found.");
    }

} catch (Exception $e) {
    echo $e->getMessage();
    die();
}
?>
```

Output



Configuration file not found.

The Finally Block

If you want to execute a piece of code irrespective of whether an exception was thrown. That's where you can use the `finally` block, since the code you place in the `finally` block will always be executed after execution of the `try` and `catch` blocks, irrespective of whether or not an exception has been thrown.


```

<?php
try {
    // code

    // if something is not as expected
    // throw exception using the "throw" keyword

    // code, it won't be executed if the above exception is thrown
} catch (Exception $e) {
    // exception is raised and it'll be handled here
    // $e->getMessage() contains the error message
} finally {
    // code, it'll always be executed
}
?>

```

The code in the above example is pretty much the same, with the only exception being that we've added the finally block after the catch block. And, as we discussed, the code in this block will always be executed.

CUSTOM EXCEPTION HANDLING

To make a custom exception handler, you must create a class, containing functions meant for handling exceptions in PHP once they are triggered. This class has to be an exception class's extension. It will have all of the original properties of the exception class, but you can provide it with custom functions.

Example

```

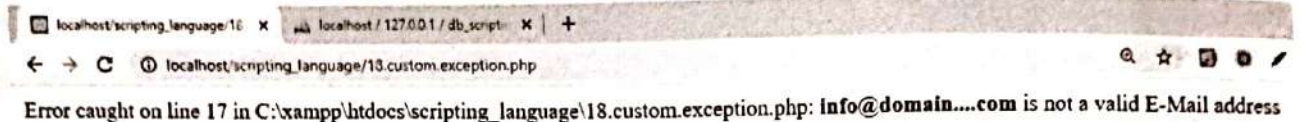
<?php
class customException extends Exception {
    public function error_message() {
        //defining the error message
        $error_msg = 'Error caught on line '.$this->getLine().' in '.$this->getFile()
            .': <b>'.$this->getMessage().'</b> is no valid E-Mail address';
        return $error_msg;
    }
}

$email = "info@domain.com";

try {
    //checking if
    if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
        //throwing an exception in case email is not valid
        throw new customException($email);
    }
} catch (customException $e) {
    //displaying a custom message
    echo $e->error_message();
}
?>

```

Output



localhost/scripting_language/16 x localhost/127.0.0.1/db_script x +
 localhost/scripting_language/13.custom.exception.php
 Error caught on line 17 in C:\xampp\htdocs\scripting_language\18.custom.exception.php: info@domain....com is not a valid E-Mail address

This new class is an almost identical copy of the original exception class. However, now it has a custom `error_message()` function. Since it has all the methods and properties of the original class, we can use every function contained in a normal exception class, such as `GetLine()`, `GetFile()`, `GetMessage()`.

MULTIPLE EXCEPTIONS

By using multiple exceptions, you can check multiple conditions at once. To do that, you should use either several `if...else` code blocks, a `switch`, or multiple exceptions. Keep in mind that each of these exceptions may use its own class and display their error messages:

```
<?php
class customException extends Exception {
    public function error_message() {
        //error message
        $error_msg = 'Error caught on line '.$this->getLine().' in '.$this->getFile()
            .': <b>'.$this->getMessage().'</b> is no valid E-Mail address';
        return $error_msg;
    }
}

$email = "info@domain.com";

try {
    //check if
    if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
        //throwing an exception in case email is not valid
        throw new customException($email);
    }
    //checking for "example" in mail address
    if(strpos($email, "example") !== FALSE) {
        throw new Exception("$email contains 'example'");
    }
} catch (customException $e) {
    echo $e->error_message();
}
catch (Exception $e) {
    echo $e->getMessage();
}
?>
```


Best Practice of Classes and Objects

- Instead of assigning variable of the classes after creating object it is good if you use constructor.
- Use visibility as required. Do not make your variable and method either more secure or completely open. Over security will affect your flexibility, under security will distrust your structure.
- Follow some convention in your classes and objects. Like start all public method with camel case, all protected method and variable prefix with _ etc. It will give you better visibility.
- Do not try to do everything in single class. Create class very specific to your requirement. It will save your time and execution.
- Always try to create every class in separate file and follow some naming convention.

AJAX: (ASYNCHRONOUS JAVASCRIPT AND XML)

AJAX is an acronym that stands for Asynchronous JavaScript and XML, which describes a set of development techniques used for building websites and web applications. Ajax is a technique for creating fast and dynamic web pages and allows data to be loaded into web pages without having to reload the whole web page. It is a Client side web technique. Using JavaScript with AJAX, it is possible to make background server calls for fetching additional data, updating some portion in web page without refreshing the whole page.

- AJAX is based on JavaScript and HTTP requests.
- AJAX is not a new programming language, but a new way to use existing standards with DOM manipulating.
- JavaScript is help to an AJAX to make background AJAX calls for fetching certain amount of data.
- Without AJAX, traditional web page takes a longer time to finishing round trip process for getting a data form the server. So it's time-consuming process even if small changes are in web page, entire web page reload.
- In a traditional web page you can't update small portion without reloading page. So here Ajax is help to background server calls for fetching data and update new content without reloading page.
- AJAX is a type of programming made popular in 2005 by Google (with Google Suggest).

Benefits of Ajax

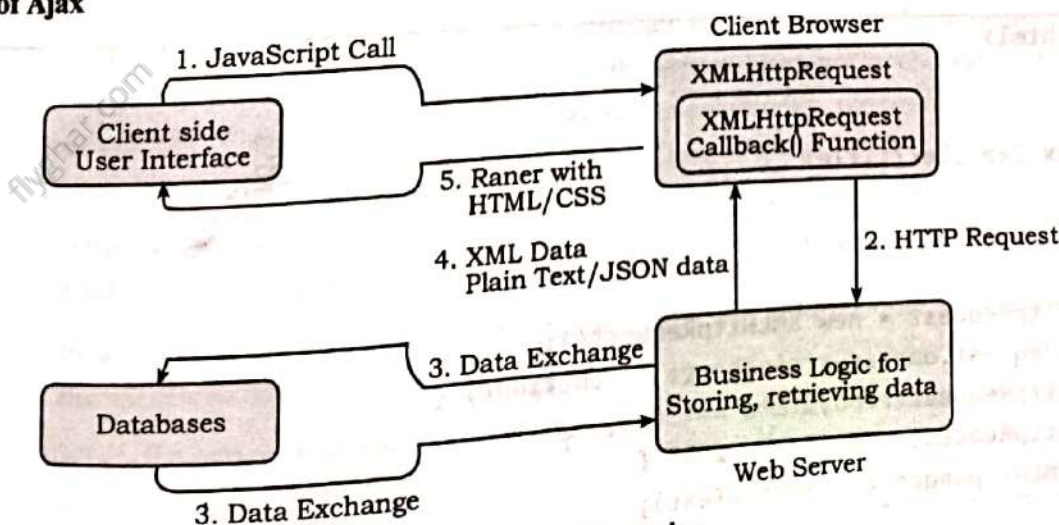


Fig: How AJAX works

Advantages of AJAX

- **Simplicity:** Using AJAX you can create better, faster, and more user-friendly web applications.
- **Versatile:** Ajax is based on JavaScript, CSS, HTML and XML etc. So you can easily learn.
- **Speed:** It reduces the server traffic in both side requests as well as reduces the time consuming on both side responses.
- **Interaction:** AJAX is much responsive; it is possible to transfer whole page (small amount of) data at a time.
- **XMLHttpRequest:** XMLHttpRequest has an important role in the Ajax web development technique. It is special JavaScript object that was designed by Microsoft. XMLHttpRequest object call as an asynchronous HTTP request to the Server for transferring data both side. It's used for making requests to the non-Ajax pages.
- **Asynchronous calls:** AJAX makes asynchronous calls to a web server. This means client browsers are avoid waiting for all data arrive before start the rendering.
- **Form Validation:** Forms are common element in web page and it should be validated instant and properly. AJAX gives the service of form validation and many more.
- **Bandwidth Usage:** In AJAX it is not required to completely reload page again and again so, it improve the speed and performance. Fetching data from database and storing data into database perform background without reloading page makes it fast.

Disadvantages of AJAX

- AJAX application would be a mistake because search engines would not be able to index an AJAX application.
- Since AJAX is open Source, anyone can view the code source written for AJAX.
- ActiveX requests are enabled only in Internet Explorer and newer latest browser.
- The last but not least disadvantage, XMLHttpRequest objects itself. For a security reason you can only use to access information from the web host that serves initial pages. If you need to fetching information from another server, it's is not possible within the AJAX.

Basic (XMLHttpRequest) AJAX request response using Vanilla JavaScript.

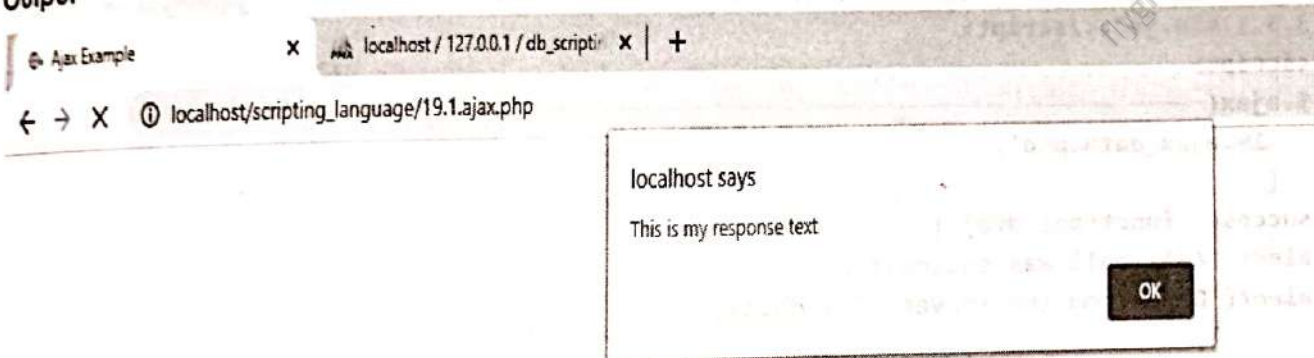
Example

```
<!DOCTYPE html>
<html>
<head>
<title>Ajax Example</title>
</head>
<body>
<script>
var objXMLHttpRequest = new XMLHttpRequest();
objXMLHttpRequest.onreadystatechange = function() {
if(objXMLHttpRequest.readyState === 4) {
if(objXMLHttpRequest.status === 200) {
alert(objXMLHttpRequest.responseText);
```



```
} else {  
    alert('Error Code: ' + objXMLHttpRequest.status);  
    alert('Error Message: ' + objXMLHttpRequest.statusText);  
}  
}  
objXMLHttpRequest.open('GET', '19.ajax_data.php');  
objXMLHttpRequest.send();  
</script>  
</body>  
</html>
```

Output



Execution steps

- First, we initialize the XMLHttpRequest object, which is responsible for making AJAX calls.
- The XMLHttpRequest object has a readyState property, and the value of that property changes during the request lifecycle. It can hold one of four values: OPENED, HEADERS_RECEIVED, LOADING, and DONE.
- We can set up a listener function for state changes using the onreadystatechange property. And that's what we've done in the above example: we've used a function which will be called every time the state property is changed.
- In that function, we've checked if the readyState value equals 4, which means the request is completed and we've got a response from the server. Next, we've checked if the status code equals 200, which means the request was successful. Finally, we fetch the response which is stored in the.responseText property of the XMLHttpRequest object.
- After setting up the listener, we initiate the request by calling the open method of the XMLHttpRequest object. The readyState property value will be set to 1 after this call.
- Finally, we've called the send method of the XMLHttpRequest object, which actually sends the request to the server. The readyState property value will be set to 2 after this call.
- When the server responds, it will eventually set the readyState value to 4, and you should see an alert box displaying the response from the server.

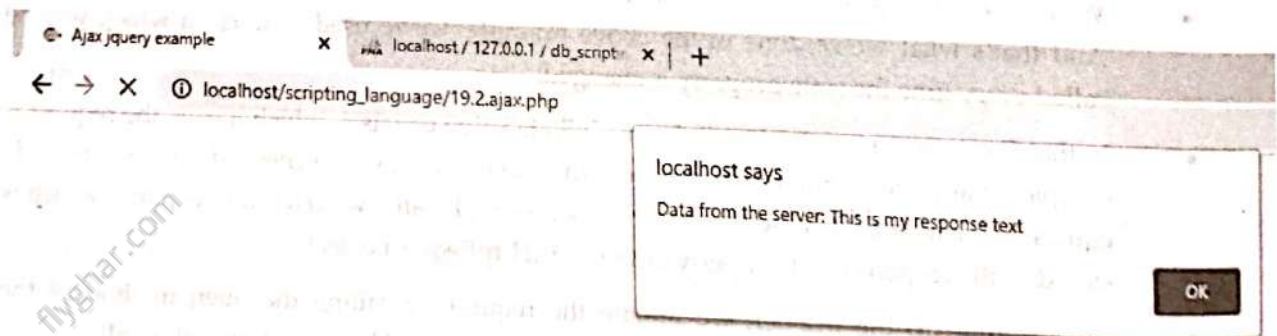
AJAX WITH JQUERY LIBRARY

The jQuery library provides a few different methods to perform AJAX calls, although here is the standard AJAX method, which is the most often used.

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Ajax jquery example</title>
</head>
<body>
    <script type="text/javascript" src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
    <script>
    $.ajax(
        '19.ajax_data.php',
        {
            success: function(data) {
                alert('AJAX call was successful!');
                alert('Data from the server: ' + data);
            },
            error: function() {
                alert('There was some error performing the AJAX call!');
            }
        }
    );
    </script>
</body>
</html>
```

Output



- The \$ sign is used to refer to a jQuery object. So we need to include JQuery library to process with ajax method.
- The first parameter of the ajax method is the URL that will be called in the background to fetch content from the server side.
- The second parameter is in JSON format and lets you specify values for some different options supported by the ajax method.

- In most cases, you will need to specify the success and error callbacks. The success callback will be called after the successful completion of the AJAX call.
- The response returned by the server will be passed along to the success callback. On the other hand, the failure callback will be called if something goes wrong and there was an issue performing the AJAX call.

So as you can see, it's easy to perform AJAX operations using the jQuery library. In fact, the process is more or less the same, irrespective of the JavaScript library with which you choose to perform AJAX calls.

AJAX WITH PHP & MYSQL

Let's take example of login system which will include login form, dashboard and backend login process file to check credential with database.

Database setup for tbl_users

SN	Name	Type
1	ID	integer
2	Username	Varchar(25)
3	Password	Varchar(25)

Please insert some data to this table to perform login operation.

Login Form:

```
<!doctype html>
<html>
<head>
<scriptsrc="https://code.jquery.com/jquery-3.3.1.js"></script>
</head>
<body>
<form id="loginform" method="post">
<div>
    Username: <input type="text" name="username" id="username" />
    Password: <input type="password" name="password" id="password" />
    <input type="submit" name="loginBtn" id="loginBtn" value="Login" />
</div>
</form>
<script type="text/javascript">
$(document).ready(function() {
    $('#loginform').submit(function(e) {
    e.preventDefault();
    $.ajax({
    type: "POST",
    url: '19.3.ajax_login.php',
    data: $(this).serialize(),
    success: function(response)
    {
    varjsonData = JSON.parse(response);
```

```

        // user is logged in successfully in the back-end
        // let's redirect
if (jsonData.success == "1")
{
    alert('Login success ! We are redirecting to dashboard.')
    location.href = '19.dashboard.php';
}
else
{
    alert('Invalid Credentials!');
}
});
});
});
</script>
</body>
</html>

```

Back logic to check username and password

```

<?php
if (isset($_POST['username']) && $_POST['username'] &&isset($_POST['password']) &&
$_POST['password']) {
    $username = $_POST['username'];
    $password = $_POST['password'];

    $servername = "localhost";
    $dbusername = "root";
    $dbpassword = "";
    $database = "db_scripting_language";
    // Creating connection
    $connection = new mysqli($servername, $dbusername, $dbpassword,$database);
    // Checking connection
    if ($connection->connect_errno != 0) {
        die("Connection failed: " . $connection->connect_error);
    }
    // Create table
    $sql = "SELECT * FROM tbl_users WHERE username='$username' and
password='$password'";
    //execute sql
    $result = $connection->query($sql);
    //check if result object contain any record
    if ($result->num_rows> 0) {
        echojson_encode(array('success' => 1));
    } else {
        echojson_encode(array('success' => 0));
    }
    $connection->close();
} else {
    echojson_encode(array('success' => 0));
}
?>

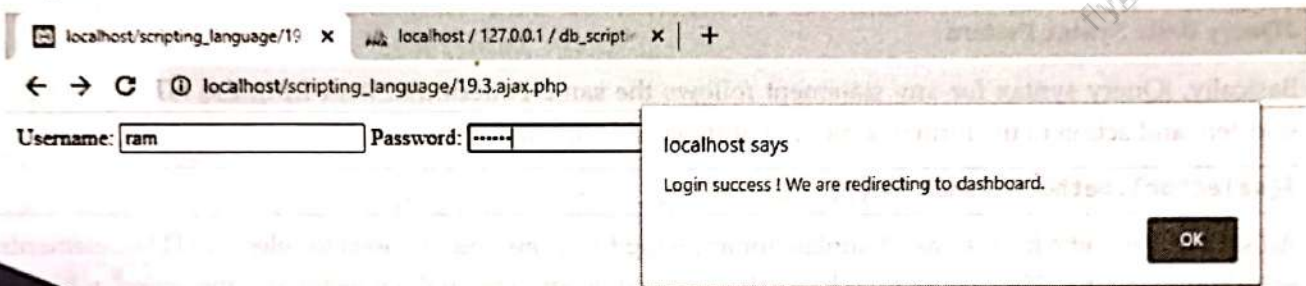
```


Dashboard after Login Success

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Dashbaord</title>
</head>
<body>
<h1>Welcome to Dashboard</h1>
</body>
</html>
```

Output



JQUERY

jQuery is a library that makes it quicker and easier to build JavaScript webpages and web apps. Often with jQuery you can write a single line of code to achieve what would have taken 10-20 lines of regular JavaScript code. jQuery allows web developers to plug routine JavaScript features into a web page so they can spend more time focusing on complicated features that are unique to their site. jQuery is itself written in JavaScript, and comes in the form of a single .js file that you link to from your webpage. Your JavaScript code then accesses the library by calling various jQuery functions.

The jQuery library actually comes in 2 forms:

- The uncompressed .js file is easy to read and modify, but it's around 160kb in size (at the time of writing).
- The minified .min.js file has all comments, whitespace, and other unnecessary characters removed from the file, squeezing the whole library into a mere 23kb. Although you can't easily read the code, this is the version you'll want to place on your site, as it's much quicker for visitors to download.

What can you do with jQuery?

- Adding animated effects to elements. jQuery lets you easily add effects such as fading in/out, sliding in/out, and expanding/contracting.
- Making XML (Ajax) requests. These use JavaScript to request additional data from the Web server without having to reload the page.

- Manipulating the DOM. You can easily add, remove, and reorder content in the Web page using just a couple of lines of code.
- Creating image slideshows. You can use jQuery effects to build nice animated slideshows and light boxes.
- Making drop-down menus. JQuery makes it easy to create multi-level dropdowns with animations.
- Creating drag-and-drop interfaces. Use jQuery to build a page with elements that can be repositioned or reordered simply by dragging and dropping.
- Adding power to forms. With jQuery you can easily add complex client-side form validation, create auto-complete Ajax text fields that pull data from a server-side database, and so on.

jQuery Basic Syntax Pattern

Basically, jQuery syntax for any statement follows the same pattern. An item from the HTML DOM is selected, and action in the form of a method or function is applied:

```
$(selector).methodOrFunction();
```

All statements come from a basic template for applying functions and methods to selected HTML elements and their attributes. This method can be easily modified depending on the selector and the action, which is the applied method or function.

Document Ready Event

Before jQuery is applied, it is important to make sure that all HTML elements have been loaded, and the CSS is fully computed. These conditions are necessary for jQuery to be applied properly. To make sure that is the case, we place all jQuery inside the document ready event. This means that the statements inside the block of code will only be applied once the web page has finished loading.

```
$(document).ready(() => {  
    // All jQuery statements go here  
});
```

There is one more way of writing the document ready event. It is shorter and therefore often used for convenience.

```
$(function(){  
    // All jQuery statements go here  
});
```

Playing with elements

To work with different elements of webpage with jQuery we need to understand basic selectors and methods. We will apply different method to selected selector to design website.

jQuery Selector

Selectors in jQuery are meant to specify a set of elements based on certain attributes, such as ID, class, or the type of tag itself. These elements can then be selected for applying the jQuery method or a function you define.

- Type
- Class
- ID
- Possession of Attribute
- Attribute Value
- Indexed Selector
- Pseudo-state

Let's take the following HTML for example:

```
<a href="index.html"></a><!-- 1 -->
<a id="second-link"></a><!-- 2 -->
<a class="example"></a><!-- 3 -->
<a class="example" href="about.html"></a><!-- 4 -->
<span class="example"></span><!-- 5 -->
```

Selecting by Type

The following jQuery selector will select all <a> elements, including 1, 2, 3 and 4.

```
$("a")
```

Selecting by Class

The following jQuery selector will select all elements of class example (including non-a elements), which are 3, 4 and 5.

```
$(".example")
```

Selecting by ID

The following jQuery selector will select the element with the given ID, which is 2.

```
$("#second-link")
```

Selecting by Possession of Attribute

The following jQuery selector will select all elements with a defined href attribute, including 1 and 4.

```
$("[href]")
```

Selecting by Attribute Value

The following jQuery selector will select all elements where the href attribute exists with a value of index.html, which is just 1.

```
$("[href='index.html']")
```

Selecting by Pseudo-state

You can also select in jQuery using pseudo-states, including :first-child, :last-child, :first-of-type, :last-of-type, etc. The following jQuery selector will only select the first <a> element: number 1.

```
$("a:first-of-type")
```

Combining jQuery selectors

You can also increase your specificity by combining multiple jQuery selectors; you can combine any number of them or combine all of them. You can also select multiple classes, attributes and states at the same time.

```
$("a.class1.class2.class3#someID[attr1][attr2='something'][attr3='something']:first-of-type:first-child")
```

This would select an <a> element that:

- Has the following classes: class1, class2, and class3
- Has the following ID: someID
- Has the following Attribute: attr1
- Has the following Attributes and values: attr2 with value something, attr3 with value something
- Has the following states: first-child and first-of-type

You can also separate different selectors with a comma:

```
$("a, .class1, #someID")
```

This would select:

- All <a> elements
- All elements that have the class class1
- An element with the id #someID

Child and Sibling selection

jQuery selectors generally conform to the same conventions as CSS, which allows you to select children and siblings in the same way.

- To select a non-direct child, use a space
- To select a direct child, use a >
- To select an adjacent sibling following the first, use a +
- To select a non-adjacent sibling following the first, use a ~

Wildcard selection (Universal)

There might be cases when we want to select all elements but there is not a common property to select upon (class, attribute etc.). In that case we can use the * selector that simply selects all the elements:

```
$('#wrapper *') // Select all elements inside #wrapper element
```

Hiding Unhiding Elements

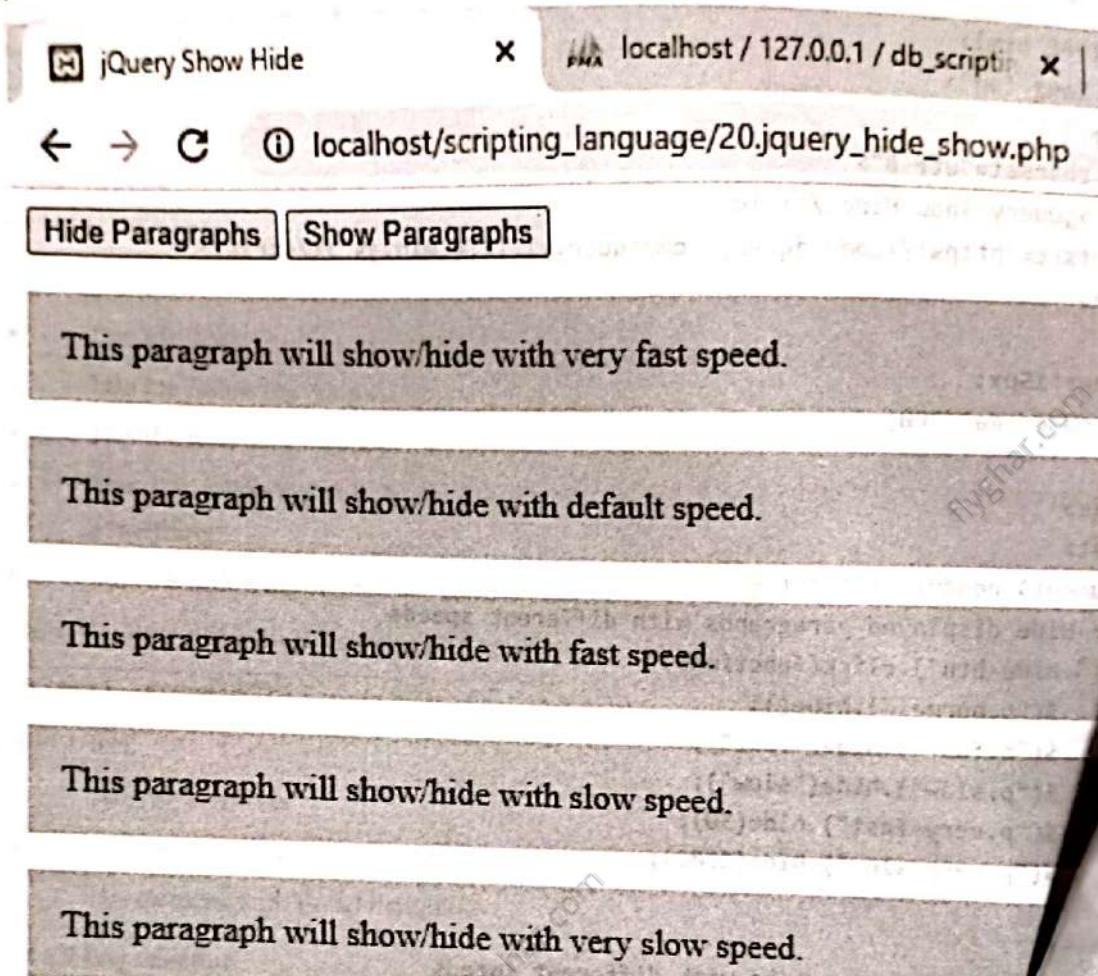
You can show and hide HTML elements using the jQuery show() and hide() methods. The hide() method simply sets the inline style display: none for the selected elements. Conversely, the show() method restores the display properties of the matched set of elements to whatever they initially were, typically block, inline, or inline-block. Before the inline style display: none was applied to them.

Example

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Show Hide</title>
<scriptsrc="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<style>
p{
padding: 15px;
background: #a2f200;
}
</style>
<script>
$(document).ready(function(){
    // Hide displayed paragraphs with different speeds
    $(".hide-btn").click(function(){
        $("p.normal").hide();
        $("p.fast").hide("fast");
        $("p.slow").hide("slow");
        $("p.very-fast").hide(50);
        $("p.very-slow").hide(2000);
    });

    // Show hidden paragraphs with different speeds
    $(".show-btn").click(function(){
        $("p.normal").show();
        $("p.fast").show("fast");
        $("p.slow").show("slow");
        $("p.very-fast").show(50);
        $("p.very-slow").show(2000);
    });
});
</script>
</head>
<body>
<button type="button" class="hide-btn">Hide Paragraphs</button>
<button type="button" class="show-btn">Show Paragraphs</button>
<p class="very-fast">This paragraph will show/hide with very fast speed.</p>
<p class="normal">This paragraph will show/hide with default speed.</p>
<p class="fast">This paragraph will show/hide with fast speed.</p>
<p class="slow">This paragraph will show/hide with slow speed.</p>
<p class="very-slow">This paragraph will show/hide with very slow speed.</p>
</body>
</html>
```

Output



JQUERY UI

jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library. Whether you're building highly interactive web applications or you just need to add a date picker to a form control, jQuery UI is the perfect choice. jQuery UI is built to work with jQuery.

jQuery UI contains many widgets that maintain state and therefore have a slightly different usage pattern than typical jQuery plugins. All of jQuery UI's widgets use the same patterns, so if you learn how to use one, then you'll know how to use all of them.

jQuery UI is a JavaScript library that can be used to add sophisticated controls such as a date picker, dialog boxes, slider and so on to a standard HTML page. It doesn't even have to be an HTML5 page as long as JavaScript is running then it works with most browser. You can also use it to add animation and other special effect.

To use jQuery UI you need to load

- jQuery
- jQuery UI
- jQuery UI style sheet.

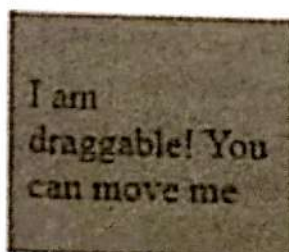
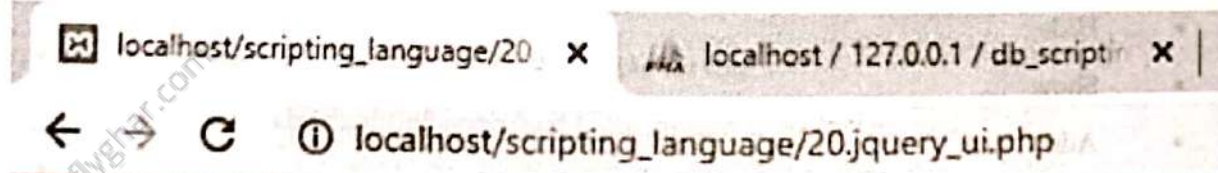
There is more to say about jQuery UI and its style sheet but for the moment a basic default style sheet will do. You can host all three files on your local server, but to get started the easiest thing to do is to use remote versions of the libraries.

Basic example of jQuery UI with draggable div.

Example

```
<!DOCTYPE html>
<head>
<link href="https://code.jquery.com/ui/1.10.4/themes/ui-lightness/jquery-ui.css"
rel="stylesheet">
<scriptsrc="https://code.jquery.com/jquery-1.10.2.js"></script>
<scriptsrc="https://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
<style>
#draggable{ width: 100px; height: 100px; padding: 0.5em; background:#22a2ff;}
</style>
<script>
$(function() {
$( "#draggable" ).draggable();
});
</script>
</head>
<body>
<div id="draggable" class="ui-widget-content">
<p>I am draggable! You can move me</p>
</div>
</body>
</html>
```

Output



JQUERY UI CATEGORIZATION

Interactions

Interactions are the set of plug-ins which facilitates users to interact with DOM elements. These are the mostly used interactions:

- Draggable
- Droppable
- Resizable
- Selectable
- Sortable

Widgets

Widgets are the jQuery plug-ins which makes you able to create user interface elements like date picker, progress bar etc. These are the mostly used widgets:

- Accordion
- Autocomplete
- Dialog
- Button
- Date Picker
- Menu
- Progress Bar
- Tabs
- Tooltip
- Slider
- Spinner

Effects

The internal jQuery effects contain a full suite of custom animation and transition for DOM elements.

- Hide
- Show
- Add Class
- Remove Class
- Switch Class
- Toggle Class
- Color Animation
- Effect
- Toggle

Utilities

Utilities are the modular tools, used by jQuery library internally.

- **Position:** It is used to set the position of the element according to the other element's alignment (position).

Introduction to CMS

A content management system is software that helps users create, manage, and modify content on a website without the need for specialized technical knowledge. CMS is software which stores all the data such as text, photos, music, documents, etc. and is made available on your website. It helps in editing, publishing and modifying the content of the website. Instead of building your own system for creating web pages, storing images, and other functions, the content management system handles all that basic infrastructure stuff for you so that you can focus on more forward-facing parts of your website.

Features of CMS

- **User Management:** It allows managing the user information such as changing the role of the users to (subscriber, contributor, author, editor or administrator), create or delete the user, change the password and user information. The main role of the user manager is Authentication.
- **Media Management:** It is the tool for managing the media files and folder, in which you can easily upload, organize and manage the media files on your website.
- **Theme System:** It allows modifying the site view and functionality. It includes images, stylesheet, template files and custom pages.
- **Extend with Plugins:** Several plugins are available which provides custom functions and features according to the users need.
- **Search Engine Optimization:** It provides several search engine optimization (SEO) tools which makes on-site SEO simple.
- **Multilingual:** It allows translating the entire content into the language preferred by the user.
- **Importers:** It allows importing data in the form of posts. It imports custom files, comments, post pages and tags.

Advantages

- It is an open source platform and available for free.
- CSS files can be modified according to the design as per users need.
- There are many plugins and templates available for free. Users can customize the various plugins as per their need.
- It is very easy to edit the content as it uses WYSIWYG editor (What You See Is What You Get is a user interface that allows the user to directly manipulate the layout of document without having a layout command).
- Media files can be uploaded easily and quickly.
- It offers several SEO tools which makes on-site SEO simple.

- Customization is easy according to the user's needs.
- It allows creating different roles for users for website such as admin, author, editor and contributor.

Disadvantages

- Using several plugins can make the website heavy to load and run.
- PHP knowledge is required to make modifications or changes in the WordPress website.
- Sometimes software needs to be updated to keep the WordPress up-to-date with the current browsers and mobile devices. Updating WordPress version leads to loss of data, so it a backup copy of the website is required.
- Modifying and formatting the graphic images and tables is difficult.

Joomla

Joomla is an open source content management system platform on which Web sites and applications can be created and connects your site to a database in order to make content management and delivery easier on both the site manager and visitor. It helps people without programming skills to set up their websites and gives the possibility to publish different kind of content (text, photos, videos, documents and more). Creating, editing and publishing content is simple! all the features and functionalities can be controlled from the management interface. It has been extremely successful long time and now and is popular with millions of users worldwide. Joomla is used all over the world for simple homepages and for complex corporate websites as well. It is easy to install, easy to manage and very reliable.

What can you do with Joomla?

All kinds of dynamic websites can be created with Joomla. These websites consists of web pages containing content such as text, images, video and audio. The pages changes or customizes themselves frequently and automatically, based on certain criteria. Take the frontpage as an example. Usually it displays the newest articles automatically on top.

Different kind of website we can develop with Joomla.

- Personal site
- Personal blog
- Government site
- News site
- Ecommerce site
- Gallery site
- Forum site
- Community site
- Blog
- Company site
- School College site

- Brand building site

Installation

Requirements for Joomla! 3.x

Software	Recommended	Minimum	More Information
PHP	7.3 +	5.3.10	https://www.php.net (Magic Quotes GPC, MB String Overload = off) (Zlib Compression Support, XML Support, INI Parser Support, JSON Support, MB Language = Default)
Supported Databases			
MySQL ^[2]	5.5.3 +	5.1	https://www.mysql.com (InnoDB support required)
SQL Server	10.50.1600.1 +	10.50.1600.1	https://www.microsoft.com/sql
PostgreSQL	9.1 +	8.3.18	https://www.postgresql.org/
Supported Web Servers			
Apache	2.4 +	2.0	https://www.apache.org (with mod_mysql, mod_xml, and mod_zlib)
Nginx	1.8 +	1.0	https://www.nginx.com/resources/wiki/
Microsoft IIS	7	7	https://www.iis.net

Recommended PHP.ini Settings

There are some PHP settings that need to be sufficient for Joomla to install. The settings are usually in a "php.ini" or "user.ini". Talk to your host about how to change these settings if it is possible to do so. If working on a localhost e.g. with XAMPP, you should not be restricted by these settings and VPS or dedicated hosting should also not be as restrictive.

The values for PHP.ini below are *suggested values* only.

memory_limit - **Minimum:** 64M, **recommended:** 128M or better

upload_max_filesize - **Minimum:** 20M

post_max_size - **Minimum:** 20M

max_execution_time: **Recommended:** 30

In order to install Joomla! On your local PC, it is necessary to set up your "own internet", for which you'll need a browser, a web server, a PHP environment and as well as Joomla supported database system. We call this a solution stack and a client server system.

A solution stack is a set of software subsystems or components to make a fully functional Joomla website possible.

The solution stack we need consists of

- The operating system (Linux, Windows, OSX)
- The web server (Apache, IIS)
- The database (MySQL, PostgreSQL)
- The script language (PHP)

Preparation to Install Joomla

- Download the current release of Joomla! 3.x from <https://downloads.joomla.org/>
- Move the downloaded Joomla! Installation package to the server(local or live). Use a FTP Client to transfer the Joomla! 3.x files to your server or if you want to install into your local server move to your document room folder as per your server configuration. Document root include public_html and htdocs. What your Host uses depends on them.
- Also create database into your database server to store CMS data.

Installation steps

1. After placing Joomla setup folder into your document root , browser installation folder from your browser using server address, for now we are using localhost server and htdocs folder.

URL: http://localhost/scripting_language/joomla/[scripting_language is folder where we place the joomla installation folder.] and you will get following interface and enter configuration as per form requirement.

The screenshot shows the Joomla! Web Installer interface in a browser window. The address bar shows the URL localhost/scripting_language/joomla/installation/index.php. The Joomla! logo is at the top, followed by the text "Joomla! is free software released under the GNU General Public License." Below this are three tabs: "1 Configuration", "2 Database", and "3 Overview". The "Configuration" tab is active. Under "Select Language", "English (United States)" is selected. A "Next" button is visible. The "Main Configuration" section contains several fields: "Site Name *" with the value "BCA", "Description" with the text "This website is for the purpose of blog for BCA 4th Semester Student", "Super User Account Details" with "Email *" as "instructor.basanta@gmail.com", "Username *" as "instructor", "Password *" and "Confirm Password *" fields, and a "Site Offline" section with "Yes" and "No" radio buttons. A "Next" button is at the bottom right.

Joomla! Web Installer x localhost / 127.0.0.1 / db_script x +

localhost/scripting_language/joomla/installation/index.php

Joomla!

Joomla! is free software released under the GNU General Public License.

1 Configuration 2 Database 3 Overview

Select Language English (United States) Next

Main Configuration

Site Name * BCA
Enter the name of your Joomla! site

Description This website is for the purpose of blog for BCA 4th Semester Student
Enter a description of the overall website that is to be used by search engines. Generally, a maximum of 20 words is best.

Super User Account Details

Email * instructor.basanta@gmail.com
Enter an email address. This will be the email address of the website Super User.

Username * instructor
Set the username for your Super User account.

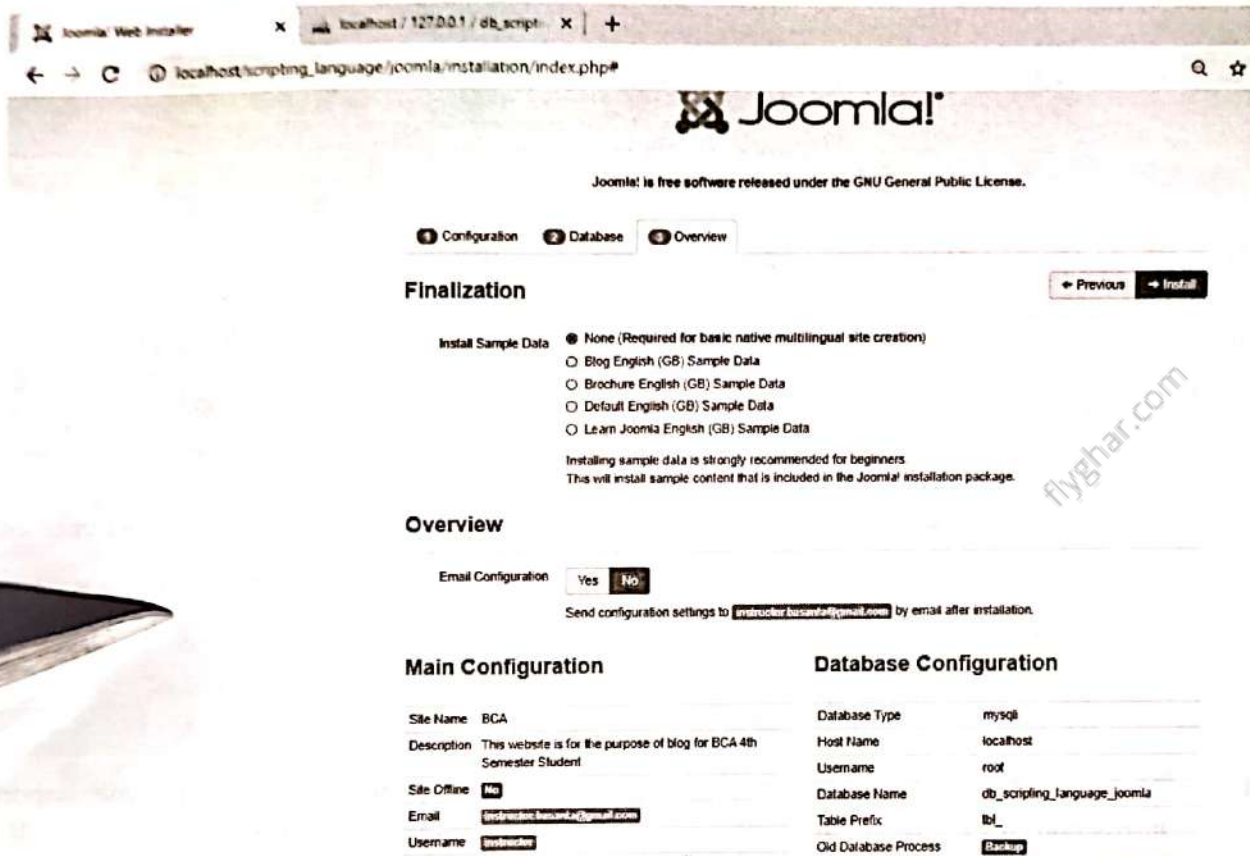
Password *
Set the password for your Super User account and confirm it in the field below.

Confirm Password *
Set the password for your Super User account and confirm it in the field below.

Site Offline Yes No
Set the site Frontend offline when installation is completed. The site can be set online later on through the Global Configuration.

Next

2. After clicking next from configuration screen website will move to Database configuration.



Joomla! Web Installer

localhost / 127.0.0.1 / db_script

localhost/scripting_language/joomla/installation/index.php#

Joomla!

Joomla! is free software released under the GNU General Public License.

1 Configuration 2 Database 3 Overview

Finalization Previous Install

Install Sample Data

- ☒ None (Required for basic native multilingual site creation)
- ☐ Blog English (GB) Sample Data
- ☐ Brochure English (GB) Sample Data
- ☐ Default English (GB) Sample Data
- ☐ Learn Joomla! English (GB) Sample Data

Installing sample data is strongly recommended for beginners.
This will install sample content that is included in the Joomla! installation package.

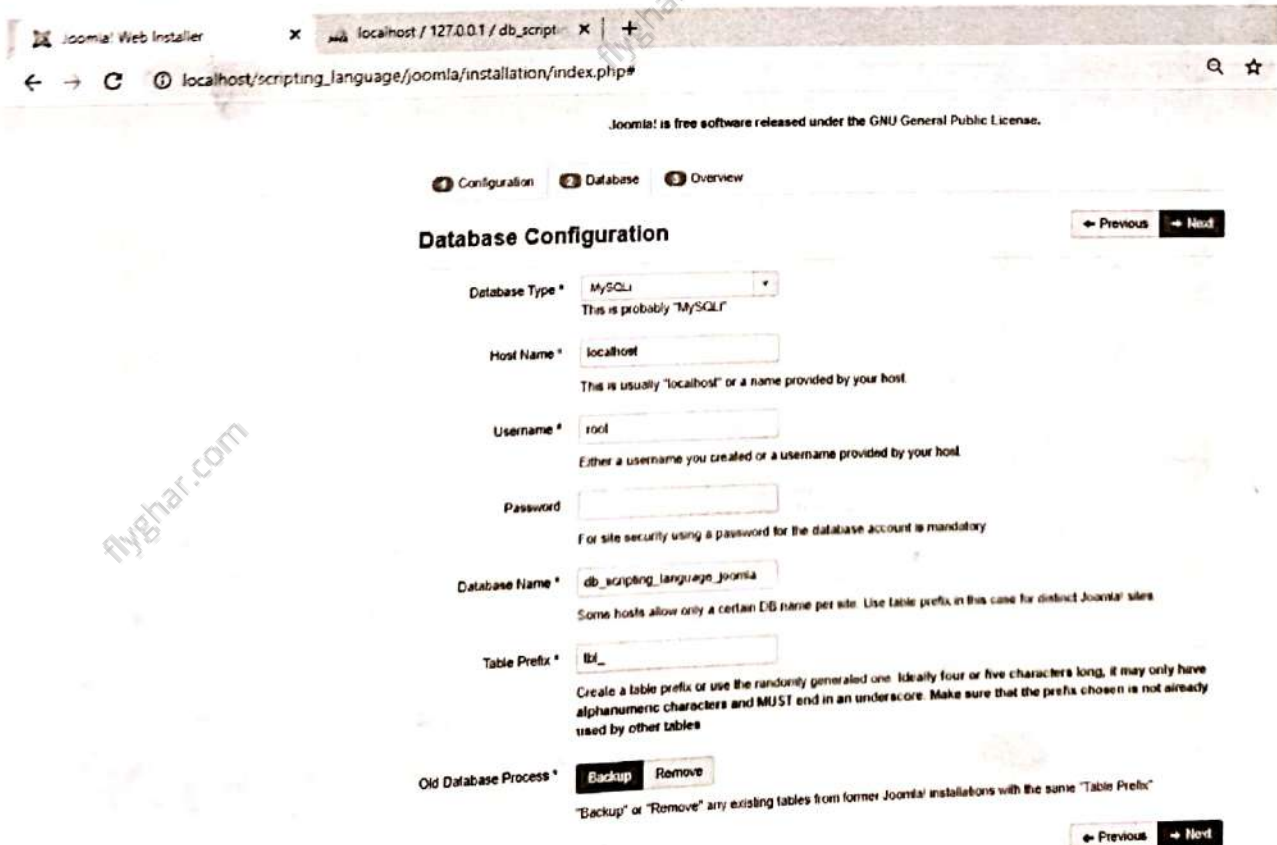
Overview

Email Configuration Yes No

Send configuration settings to instructor.kusanta@gmail.com by email after installation.

Main Configuration	Database Configuration
Site Name: BCA	Database Type: mysql
Description: This website is for the purpose of blog for BCA 4th Semester Student	Host Name: localhost
Site Offline: No	Username: root
Email: instructor.kusanta@gmail.com	Database Name: db_scripting_language_joomla
Username: instructor	Table Prefix: tbl_
	Old Database Process: Backup

Final overview screen will show all the configuration list.



Joomla! Web Installer

localhost / 127.0.0.1 / db_script

localhost/scripting_language/joomla/installation/index.php#

Joomla! is free software released under the GNU General Public License.

1 Configuration 2 Database 3 Overview

Database Configuration Previous Next

Database Type * MySQL
This is probably "MySQL"

Host Name * localhost
This is usually "localhost" or a name provided by your host.

Username * root
Either a username you created or a username provided by your host.

Password *
For site security using a password for the database account is mandatory.

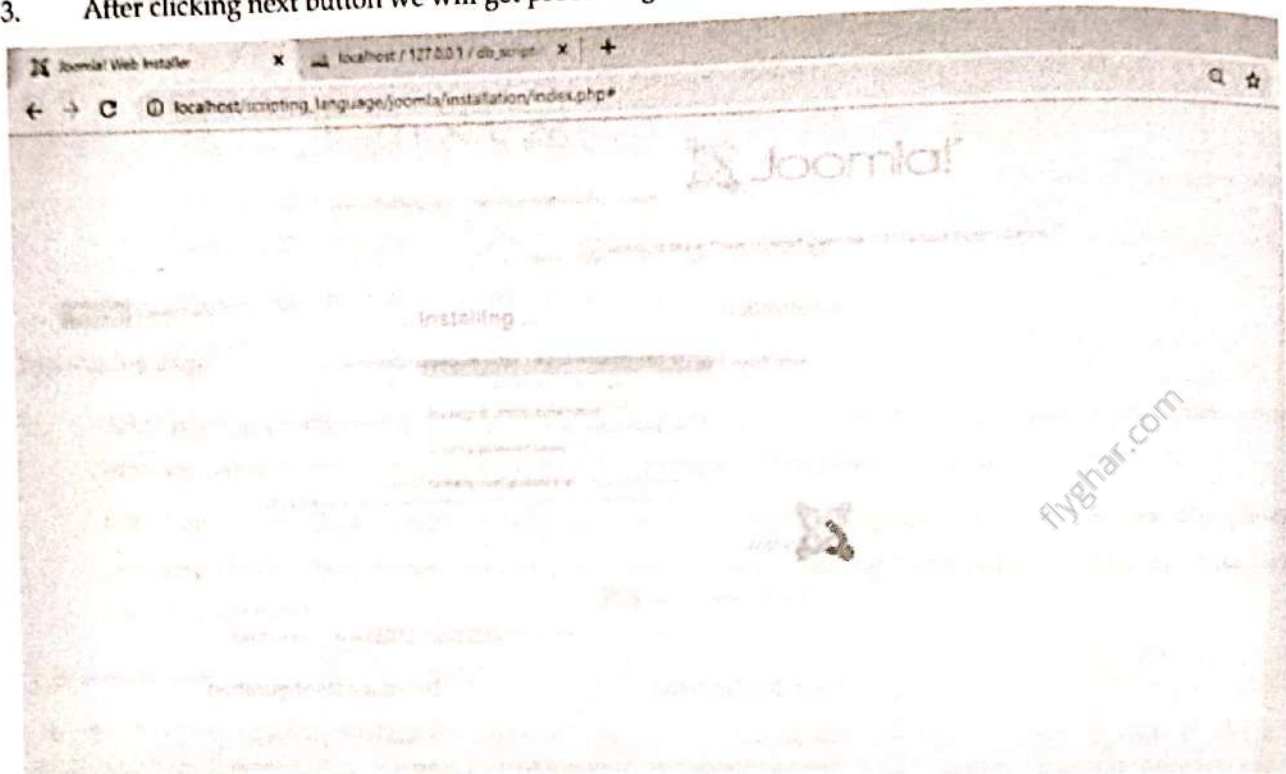
Database Name * db_scripting_language_joomla
Some hosts allow only a certain DB name per site. Use table prefix in this case for distinct Joomla! sites.

Table Prefix * tbl_
Create a table prefix or use the randomly generated one. Ideally four or five characters long, it may only have alphanumeric characters and MUST end in an underscore. Make sure that the prefix chosen is not already used by other tables.

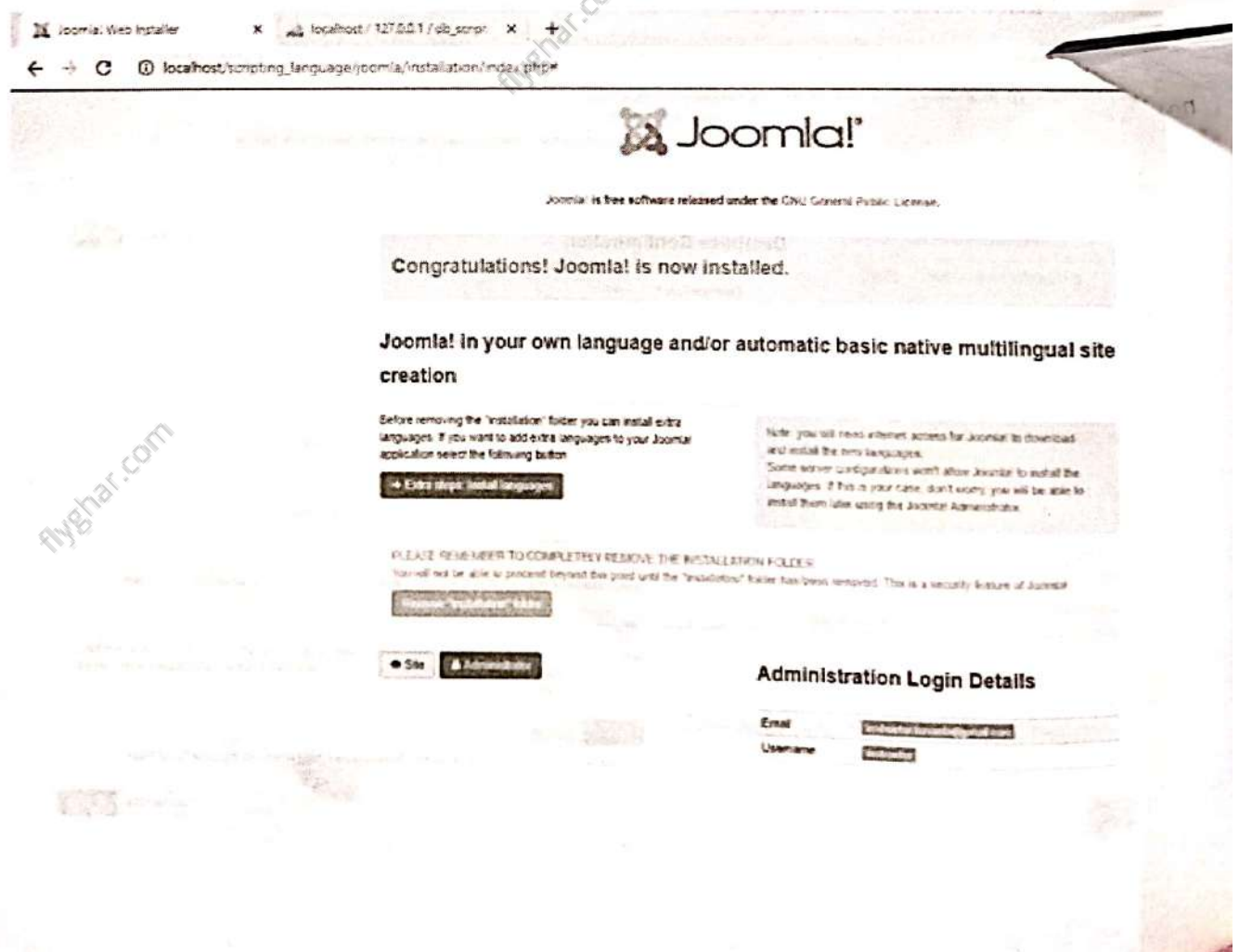
Old Database Process * Backup Remove
"Backup" or "Remove" any existing tables from former Joomla! installations with the same "Table Prefix"

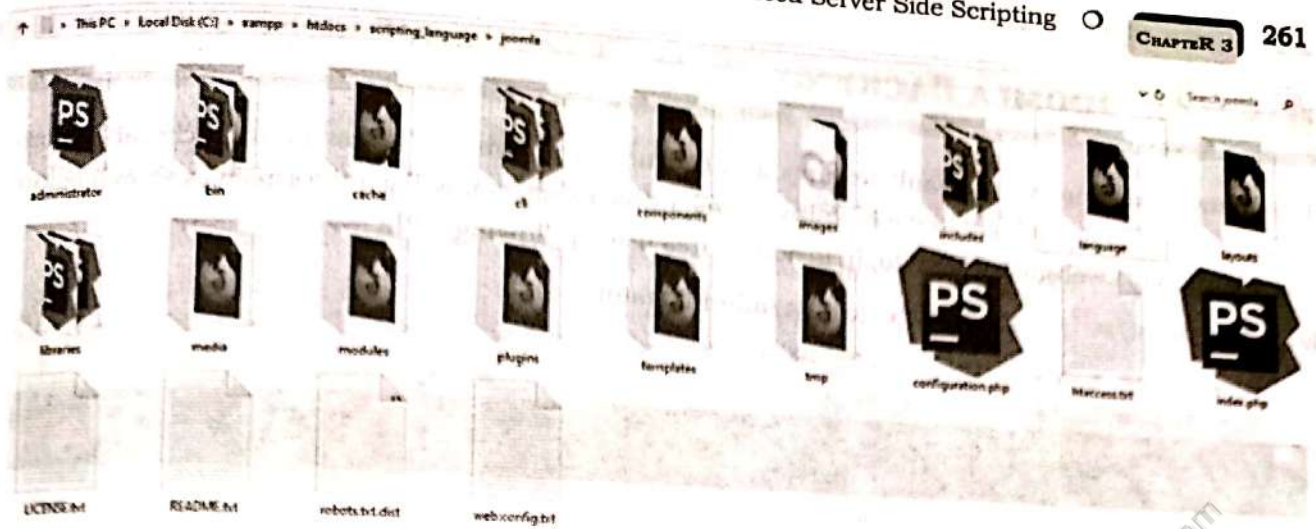
Previous Next

3. After clicking next button we will get processing screen.



4. After completion of installation process we will get following screen with some important instructions. It's important to remove "installation" folder from your website directory list. If you forgot to remove or rename "installation folder" installation process will repeat again.



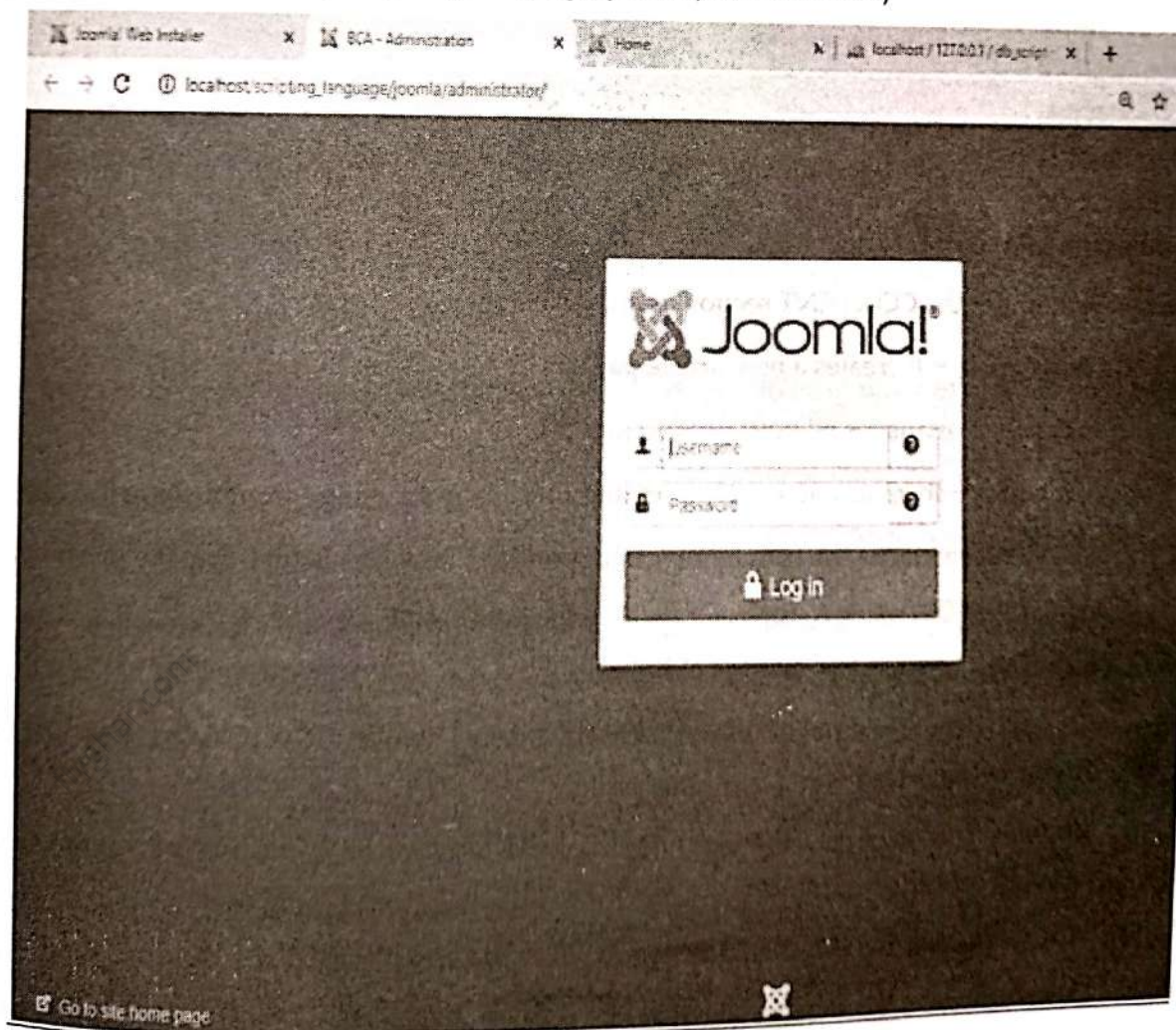


After completion of installation process you will get following folder structure for your website.

Two Areas of a Joomla Site

Every Joomla Web site has two areas: a public area for visitors and a private area for administrators.

- To access administrator's area use following URL:
http://localhost/scripting_language/joomla/administrator/

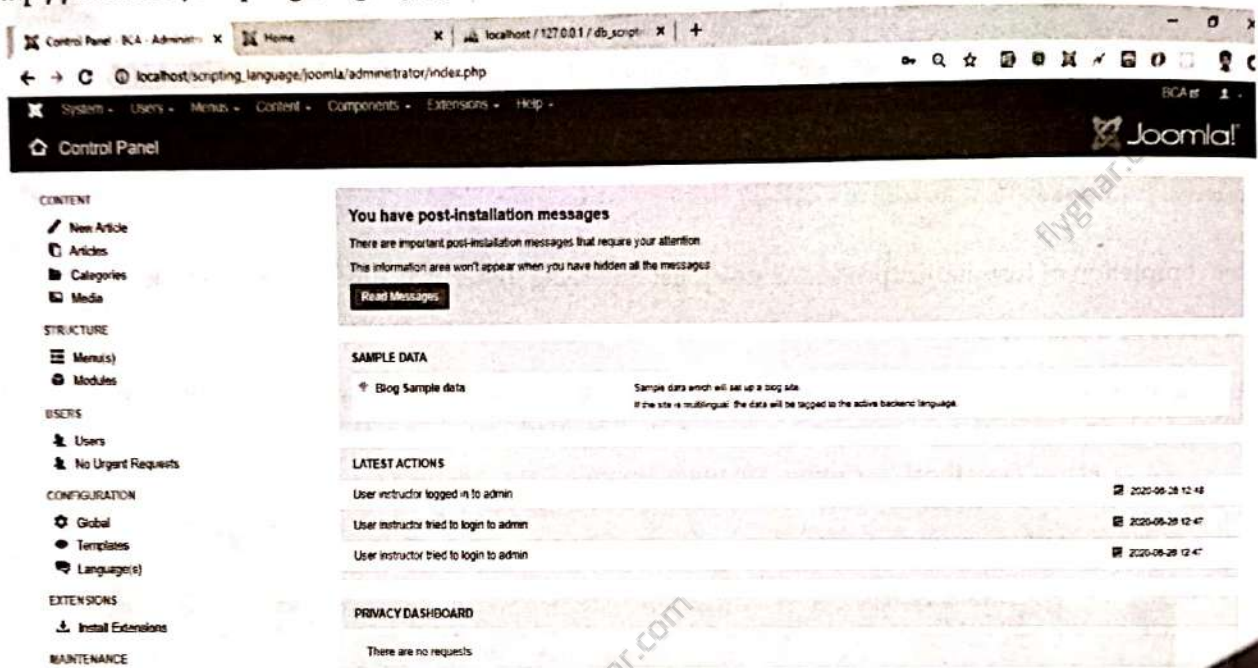


- To access visitors area use following URL:
http://localhost/scripting_language/joomla/

HANDLING JOOMLA BACKEND

The Control Panel provides default features and functions of Joomla to access through clickable icons, menu bar etc. When you login to the Joomla administrative panel, you will get the screen as shown below. To handle Joomla website backend login to admin area using following URL:

http://localhost/scripting_language/joomla/administrator/



Basic menus of Joomla control panel.

There are four link under the CONTENT section as shown below –

- New Article – It creates a new article page.
- Articles – It manages all your present articles.
- Categories – It creates new categories and helps in publishing/unpublishing the categories.
- Media – It manages the files by uploading various new files or deleting the existing ones on your web server.

In STRUCTURE section, there are two groups of links –

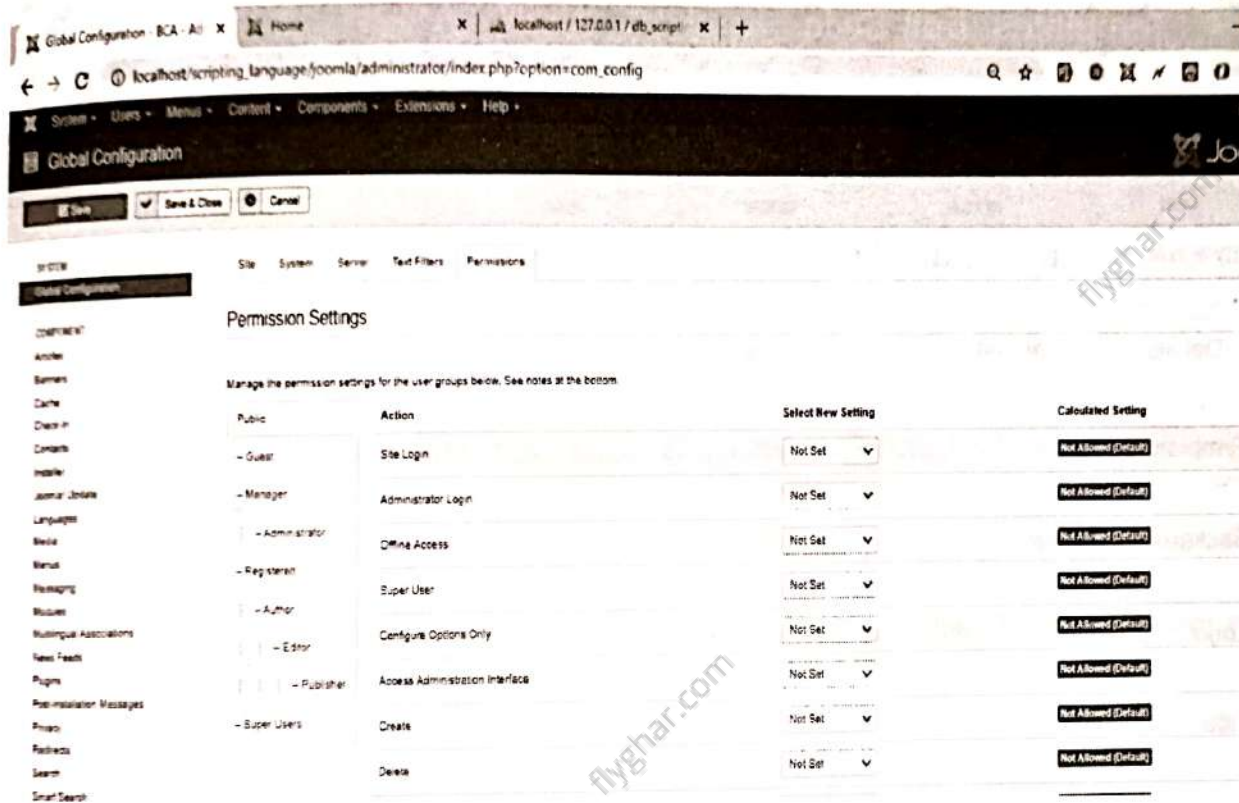
- Menu(s) – Menu manager allows creating custom menus for your website and navigating through your website.
- Modules – It manages the modules such as location and function of modules that are installed on site.

Under USERS section, there is two links are located –

- User Manager – It manages the user information, which allows creating or deleting the user, changing passwords, time and languages. You can also assign the user to User Groups.
- No Urgent Requests - It manages any user request as per their priority.

Three links are located under the CONFIGURATION section as given below –

- Global –this is an important part in the Joomla back-end. Any changes made in this configuration, will affect the entire website.
- Template – It manages the templates used in the website.
- Language – It manages installed language by setting the default language for your site.



Install Extension

There are many Extensions available in Joomla. You can install different types of extensions to extend the functionality of the site.

Maintenance

In MAINTENANCE section, there are two links located –

- Joomla is up-to-date – It views the current update status of the Joomla installation.
- All extensions are up-to-date – It views the current update status of the Joomla extension.

Customization in Joomla

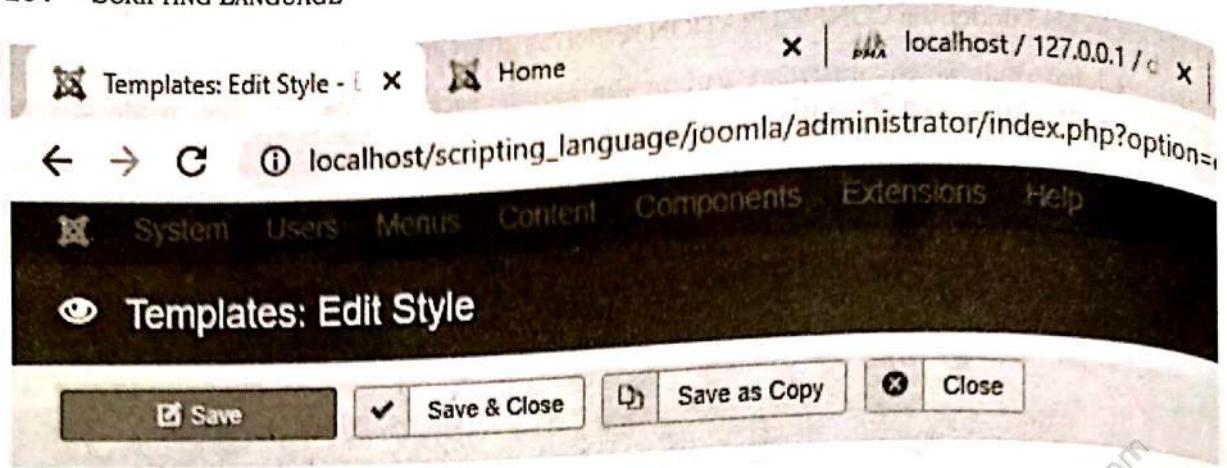
For customization of Joomla website we can modify template and global configuration for website from our top menus and left sidebar menu section.

To customize template follow following steps.

- Go to Template Manager from top Extensions menu.
- Choose your current template which you want to change.

To customize global configuration

Go to Global link under Configuration menu and change content as per your need.



Style Name *

Details

Advanced

Menu Assignment

Template Colour

#0088cc

Background Colour

#f4f6f7

Logo



Select



Title

Description

Google Font for Headings

Yes

No

Google Font Name

Open+Sans

Fluid Layout

Fluid

Static

Introduction to Extensions

Extensions are what enables you and your Joomla site visitors to perform some tasks which are not available in Joomla. An extension is a software package that extends your Joomla! Installation in some way. A small selection of extensions is included with the default Joomla! Installation but many more are available from the Joomla! Extensions Directory.

Joomla extension extends the functionality of your Joomla-powered website. Extensions can include templates, modules, components or plugins.

For example, you may want to add an e-commerce store to your Joomla site. Or a forum. Or a business directory. You then download an appropriate Joomla extension, for free or for a payment. Then you install it via Joomla extensions manager and start using it.


```

<?xml version="1.0" encoding="utf-8"?>
<extension version="2.5" type="template">
<name>customtemplate</name>
<creationDate>2020-06-01</creationDate>
<author>Author Name</author>
<authorEmail>email@domain.com</authorEmail>
<authorUrl>http://www.domainname.com</authorUrl>
<copyright>Author Name</copyright>
<license>GNU/GPL</license>
<version>1.0.2</version>
<description>My New Template</description>
<files>
<filename>index.php</filename>
<filename>templateDetails.xml</filename>
<folder>images</folder>
<folder>css</folder>
</files>
<positions>
<position>breadcrumb</position>
<position>left</position>
<position>right</position>
<position>top</position>
<position>user1</position>
<position>user2</position>
<position>user3</position>
<position>user4</position>
<position>footer</position>
</positions>
</extension>

```

So, as you can see, we have a set of information between markup tags (the <element>s). Your best approach is to copy and paste this into your templateDetails.xml file and change the relevant bits (such as <name> and <author>).

The <files> part should contain all the files that you use - you possibly don't know what they are called yet - don't worry, update it later. The <folder> element can be used to define an entire folder at once.

Leave the positions as they are - these are a common set so you will be able to switch easily from the standard templates.

3. Creating a basic index.php file

The index.php file becomes the core of every page that Joomla! delivers. Essentially, you make a page (like any HTML page) but place PHP code where the content of your site should go. The template works by adding Joomla code into module positions and the component section in your template. Anything added to the template will appear on all pages unless it is added to one of these sections via the Joomla CMS (or customized code).

A Joomla template begins with the following lines:

```
<?php defined( '_JEXEC' ) or die( 'Restricted access' );?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="<?php echo $this->language; ?>" lang="<?php echo $this->language; ?>" >
```

The first line stops people looking at your coding and getting up to bad things. The second line is the Document Type Declaration (DOCTYPE), which tells the browser (and web crawlers) which flavor of HTML the page is using. The doctype used above is HTML5, a newer version of HTML that is largely backwards compatible, but contains many new features. You should be aware that this will not work well in Internet Explorer 8 or earlier without a hack. You might want to investigate this situation and your clients' wishes before deciding on which doctype you want to use. However this is used in Joomla 3.0 and higher.

The third line begins our HTML document and describes what language the website is in. A html document is divided into two parts, head and body. The head will contain the information about the document and the body will contain the website code which controls the layout.

Head Section

```
<head>
<jdoc:include type="head" />
<link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/system/css/system.css"
type="text/css" />
<link rel="stylesheet" href="<?php echo $this->baseurl
?>/templates/system/css/general.css" type="text/css" />
<link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/<?php echo $this-
>template; ?>/css/template.css" type="text/css" />
</head>
```

The first line gets Joomla to put the correct header information in. This includes the page title, meta information as well as system JavaScript. The rest creates links to two system style sheets and to your own style sheet (if it's named template.css and is located in the css folder of your template directory. So if your template is in `http://localhost/scripting_language/joomla/templates/customtemplate/` then the css files will go in `http://localhost/scripting_language/joomla/templates/customtemplate/css/`).

Body Section

```
<body>
<jdoc:include type="modules" name="top" />
<jdoc:include type="component" />
<jdoc:include type="modules" name="bottom" />
</body>
```

These lines, usually called jdoc statements, tell Joomla to include output from certain parts of the Joomla system.

Custom Images

If you want to add any images to the template you can do so like this:

```
<imgsrc="<?php echo $this->baseurl; ?>/templates/<?php echo $this->template;
?>/images/myimage.png" alt="Custom image" class="customImage" />
```

Here the template variable will fill in the name of your template.

Custom CSS

You can add custom css like this:

```
<link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/<?php echo $this->template;?>/css/styles.css" type="text/css" />
```

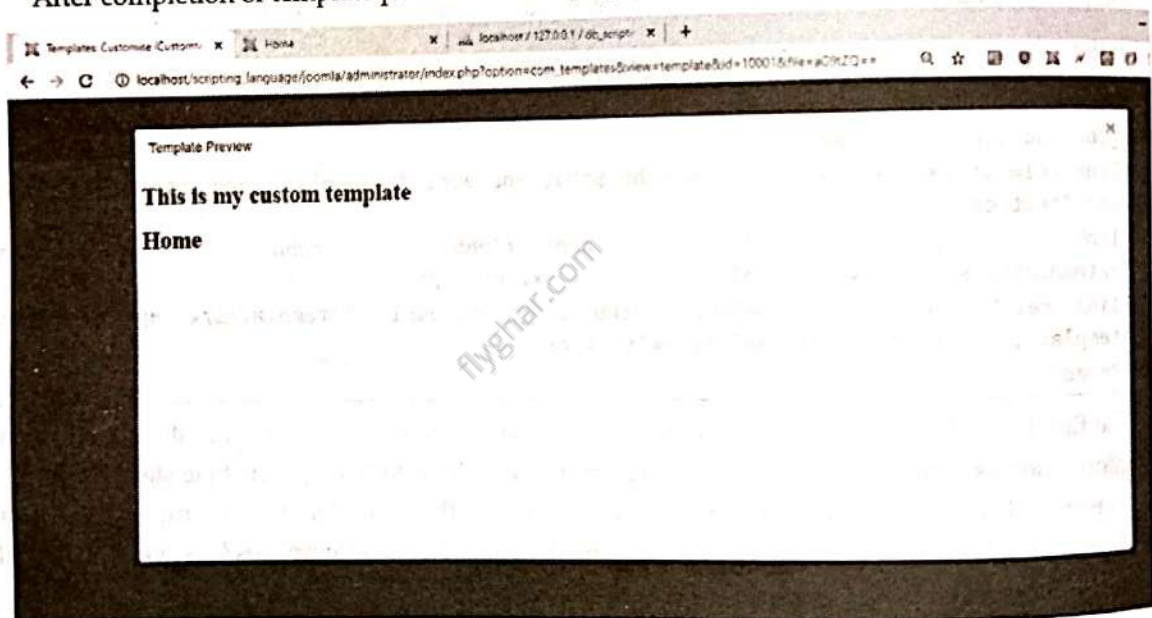
Every file which is added must have a line in the templateDetails.xml file for the template, unless it resides in a sub-folder (which can be included in a <folder> element).

4. Testing the template

Find the template in the Template Manager, select it and click Default to make it the default template.

This feature is called Discover Extensions and can be accessed via Extensions -> Extension Manager -> Discover (i.e. the Discover tab). Click Discover (i.e. the Discover button) to discover your template, then select it and click Install to install it. Now your template should show up in the Template Manager (Styles), accessible via Extensions -> Template Manager.

After completion of template preview will display below screen.

**ARTISEER (IDE)**

Artiseer is the first and only Web design automation product that instantly creates fantastic looking unique website templates and blog themes.

Feature of Artiseer IDE:

- Design awesome blogs and cool web templates in minutes
- Export to Blogger, Joomla, WordPress and other CMS products
- No need to learn Photoshop, CSS, HTML or other technologies
- Fun and easy to use!
- Preview, download and edit online website and template samples!

However you can your favorite IDE to create website.

MODULE DEVELOPMENT IN JOOMLA

Modules are lightweight and flexible extensions. They are used for small bits of the page that are generally less complex and are able to be seen across different components.

You can see many examples of modules in the standard Joomla! Install: - Menus - Latest News - Login form - and many more.

Create Hello World Module

File Structure

There are four basic files that are used in the standard pattern of module development:

- `mod_helloworld.php` - This file is the main entry point for the module. It will perform any necessary initialization routines, call helper routines to collect any necessary data, and include the template which will display the module output.
- `mod_helloworld.xml` - This file contains information about the module. It defines the files that need to be installed by the Joomla installer and specifies configuration parameters for the module.
- `helper.php` - This file contains the helper class which is used to do the actual work in retrieving the information to be displayed in the module (usually from the database or some other source).
- `tmpl/default.php` - This is the module template. This file will take the data collected by `mod_helloworld.php` and generate the HTML to be displayed on the page.

Creating `mod_helloworld.php`

The `mod_helloworld.php` file will perform three tasks: include the `helper.php` file which contains the class to be used to collect the necessary data invoke the appropriate helper class method to retrieve the data include the template to display the output:

```
<?php
/**
 * Hello World! Module Entry Point
 *
 * @package Joomla.Tutorials
 * @subpackage Modules
 * @license GNU/GPL, see LICENSE.php
 * @link http://docs.joomla.org/J3.x:Creating_a_simple_module/Developing_a_Basic_Module
 * mod_helloworld is free software. This version may have been modified pursuant
 * to the GNU General Public License, and as distributed it includes or
 * is derivative of works licensed under the GNU General Public License or
 * other free or open source software licenses.
 */
// No direct access
defined('_JEXEC') or die;
// Include the syndicate functions only once
require_once dirname(__FILE__) . '/helper.php';
$hello = modHelloWorldHelper::getHello($params);
requireJModuleHelper::getLayoutPath('mod_helloworld');
?>
```

Creating helper.php

The helper.php file contains the helper class that is used to retrieve the data to be displayed in the module output. As stated earlier, our helper class will have one method: `getHello()`. This method will return the 'Hello, World !! This is my first module' message.

```
<?php
/**
 * Helper class for Hello World! module
 *
 * @package Joomla.Tutorials
 * @subpackage Modules
 * @link http://docs.joomla.org/J3.x:Creating_a_simple_module/Developing_a_Basic_Module
 * @license GNU/GPL, see LICENSE.php
 * mod_helloworld is free software. This version may have been modified pursuant
 * to the GNU General Public License, and as distributed it includes or
 * is derivative of works licensed under the GNU General Public License or
 * other free or open source software licenses.
 */
class ModHelloWorldHelper
{
    /**
     * Retrieves the hello message
     *
     * @param array $params An object containing the module parameters
     *
     * @access public
     */
    public static function getHello($params)
    {
        return 'Hello, World! This is my first module';
    }
}
?>
```

There is no rule stating that we must name our helper class as we have, but it is helpful to do this so the class can be easily identified and located. Note that it is required to be in this format if you plan to use the `com_ajax` component. More advanced modules might include database requests or other functionality in the helper class method.

Creating tmpl/default.php

The default.php file is the template which displays the module output.

```
<?php
// No direct access
defined('_JEXEC') or die; ?>
<?php echo $hello; ?>
```


An important point to note is that the template file has the same scope as the `mod_helloworld.php` file. What this means is that the variable `$hello` can be defined in the `mod_helloworld.php` file and then used in the template file without any extra declarations or function calls.

Creating `mod_helloworld.xml`

The `mod_helloworld.xml` is used to specify which files the installer needs to copy and is used by the Module Manager to determine which parameters are used to configure the module. Other information about the module is also specified in this file.

```
<?xml version="1.0" encoding="utf-8"?>
<extension type="module" version="3.1.0" client="site" method="upgrade">
  <name>Hello, World!</name>
  <author>John Doe</author>
  <version>1.0.0</version>
  <description>A simple Hello, World! module.</description>
  <files>
    <filename>mod_helloworld.xml</filename>
    <filename module="mod_helloworld">mod_helloworld.php</filename>
    <filename>index.html</filename>
    <filename>helper.php</filename>
    <filename>tmpl/default.php</filename>
    <filename>tmpl/index.html</filename>
  </files>
  <config>
  </config>
</extension>
```

You will notice that there are two additional files that we have not yet mentioned: `index.html` and `tmpl/index.html`. These files are included so that these directories cannot be browsed. If a user attempts to point their browser to these folders, the `index.html` file will be displayed. These files can be left empty or can contain the simple line: which will display an empty page.

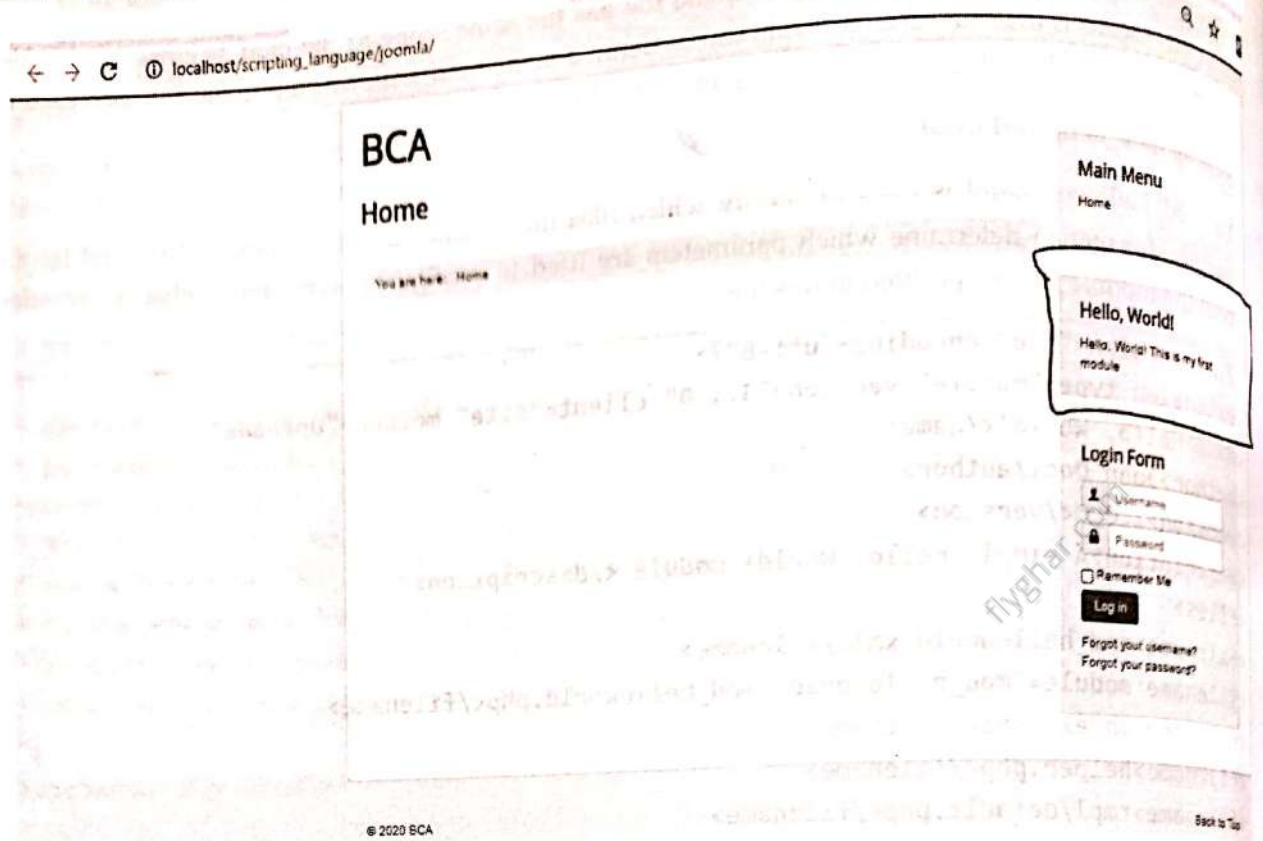
Installing and Viewing the Module

To install the module, zip up the parent directory of your source files, to create a file called `mod_helloworld.zip`. Go into Joomla admin, to the Extensions / Manage / Install form, click on the Update Package File tab, and upload your zip file. You should find that the module installs ok. If it doesn't, then compare carefully your code with the code above, and ensure your source code files are named correctly. To make the module visible on your site web pages go into the Extensions / Modules listing and click on the Hello World module to edit it.

Set the Status to Published

Select the Position where the module should appear (if you're unsure about which Position to set, then navigate to Extensions / Templates / Styles, check which template is assigned on your web pages, then follow this guide to finding the module positions). On the Menu Assignment tab set the Module Assignment to "On all pages"

Save and Close your edit form. Now if you navigate to your site you should see "Hello, World!" displayed in your chosen module position. Below figure show that hello world module at the right sidebar.



COMPONENT DEVELOPMENT IN JOOMLA

A component is a kind of Joomla! Extension. Components are the main functional units of Joomla!; they can be seen as mini-applications. An easy analogy would be that Joomla! is the operating system and the components are desktop applications. Created by a component, content is usually displayed in the center of the main content area of a template (depending on the template).

Most components have two main parts: an administrator part and a site part. The site part is what is used to render pages of your site when they are requested by your site visitors during normal site operation. The administrator part provides an interface to configure and manage different aspects of the component and is accessible through the Joomla! Administrator application. Joomla! comes with a number of core components, like the content management system, contact forms and Web Links.

Steps for component development

- Create the files for the component
- Install the component
- View the component on the front end of the website

Creating the files for the Hello World component

- Within your Components folder, create a folder named com_helloworld
- Within your new com_helloworld folder, create a blank file named helloworld.xml
- Within your new com_helloworld folder, create helloworld.php with the following

```
<?php echo "Hello World"; ?>
```


Installing our Hello World component

- Log into your Joomla admin dashboard
- In the top menu, click Extensions and then click Extension Manager
- In the left menu, click the Discover link
- In the top menu, click the Discover button
- You will see com_helloworld show up in the list of available extensions. Click the checkbox to the left of it, and then click the Install button in the top menu.

You will see a few Warnings and Messages at the top of the page:

JInstallerInstall: Cannot find Joomla XML setup file
Component Install: The XML file did not contain an administration element
Discover install failed: 10005

For now, we will ignore these messages. We're doing very barebones testing, and our component is missing a few things, which in turn is triggering these messages to show.

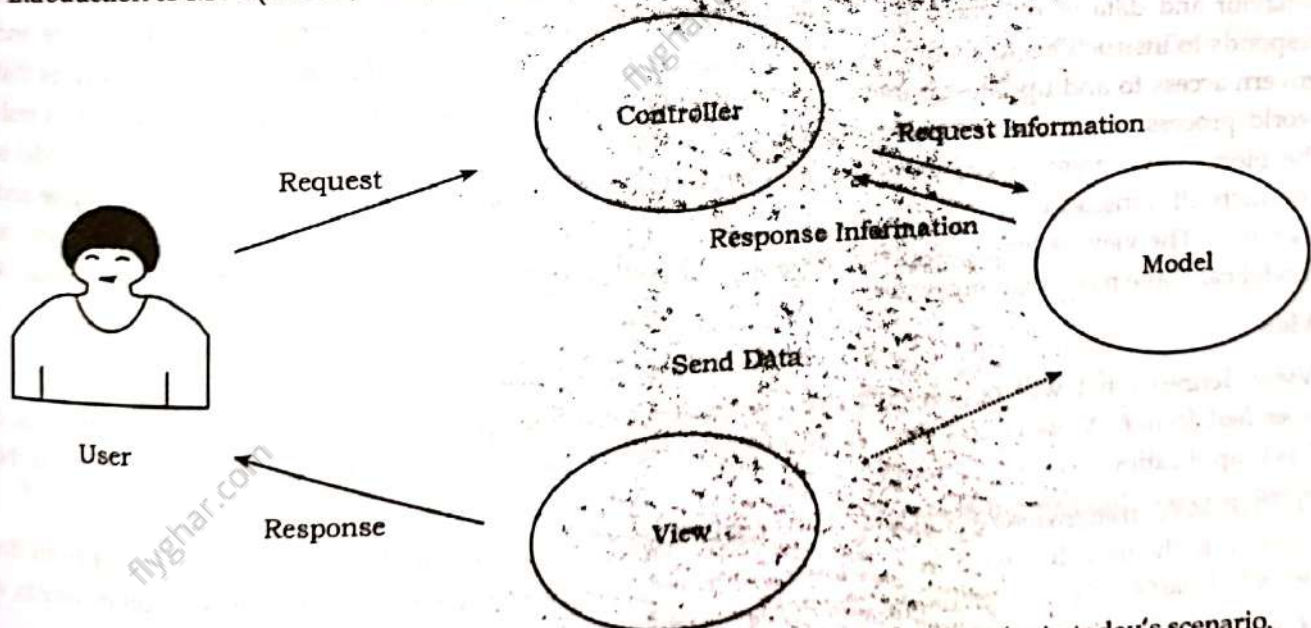
Viewing our Hello World component

To see the Hello World component in action, visit your Joomla website and add the following at the end of your URL:

`http://localhost/scripting_language/joomla/index.php?option=com_helloworld`

When going to this URL, Joomla will execute the helloworld component. It should look something similar to the screenshot to the right.

Introduction to MVC(Model,View,Controller)



MVC stands for Model-View-Controller. It is a standard in web application design in today's scenario. The Model View Controller model is the latest application development model for developing desktop, mobile, and web applications. It is extremely useful in large applications because in MVC, different parts (views, business logic and data storage) of a system are developed separately. That is the result of loosely coupled modular application. Because of this model maintenance, flexibility, and scalability of a system increases and code becomes shorter and easier to read. A lot of PHP frameworks are based on MVC patterns which already exist like laravel, zend, codigniter.

What is PHP MVC Framework?

In PHP MVC Framework, the 'html' views, business logics controllers and data handling models are developed separately, enabling the designers and developers to work with the views without any concern with the data and its handling. This makes PHP application components loosely coupled.

Advantages of MVC model:

- It is a simple solution for most web applications.
- It is flexible and extensible.
- It supports advance application.
- It is best to manage application workflow.
- MVC model separate the code into three categories: Presentation, Business logic and Data access.
- Presentation code can be consolidated in one part of application, business logic in another and the data access in another one.
- It is the best suited model, if there is more than one developer working on the same application.
- Easy to test and maintenance

Model

It is the part of the application that defines the basic functionality behind a set of abstraction. Data access routines and some business logic can be defined in the model. The model can provide the view stored user data for presentation.

A model is an object representing data or even activity, e.g. a database table. The model manages the behavior and data of the application domain, responds to requests for information about its state and responds to instructions to change state. The model represents enterprise data and the business rules that govern access to and updates of this data. Often the model serves as a software approximation to a real-world process, so simple real-world modeling techniques apply when defining the model. The model is the piece that represents the state and low-level behavior of the component. It manages the state and conducts all transformations on that state. The model has no specific knowledge of either its controllers or its views. The view is the piece that manages the visual display of the state represented by the model. A model can have more than one view.

View

View defines what will be presented to the user. Controllers pass data to each view, to render in a specified format. View collect data from the user. This is the part where we use HTML mark up in the MVC application.

In PHP MVC framework, the view is responsible for all the display logic used to present data from the 'model' to the user. It is used to pass the user input to a controller using control events. In other words, a view is a part of application that involves the generation of the html and interaction with user.

Controller

Controller binds the whole pattern together. It is responsible to manipulate data. It controls which view to display based on the request of user and other factors and pass along the data that each view will need.

In PHP MVC framework, the controller takes an input from user as a request, processes and analyzes this input. It also initializes and invokes the model, takes the model response and sends it back to the respected view. So controller is a part of application that involves the logical processes of the application also called the business logic. It acts as a middle layer for views and models.

A controller offers facilities to change the state of the model. The controller interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate. A controller is the means by which the user interacts with the application. A controller accepts input from the user and instructs the model and view to perform actions based on that input. In effect, the controller is responsible for mapping end-user action to application response.

The controller translates interactions with the view into actions to be performed by the model. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in a Web application they appear as HTTP GET and POST requests. The actions performed by the model include activating business processes or changing the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.

The controller is the piece that manages user interaction with the model. It provides the mechanism by which changes are made to the state of the model.

WORDPRESS

WordPress is the world's most popular tool for creating websites. It is an open-source content management system (CMS) and is a popular tool for individuals without any coding experience who want to build websites and blogs. WordPress is capable of creating any style of website, from a simple blog to a full-featured business website. You can even use WordPress to create an online store.

Key Features and Functionality of WordPress

- Moderating - Keep track of comments, users, roles, permissions, etc.
- Full User System - Allow newsletter users, admins, reviewers, etc.
- Integrated RSS feeds, APIs, and preset plugins.
- Maintaining Stability - WordPress is designed for non-technical maintenance. The large community extensively test core updates prior to release.
- It's generally simple to import data from multiple platforms.
- Dynamic generation of URLs and overall database structure.
- Drag-and-drop functionality in the backend to allow for easy customization.
- Variety of supported languages, typography, and content editors.
- Access revised version of posts and old content.
- Share a single backend among multiple WordPress sites.
- Easily publish content, pages, and media.

Advantages of Using WordPress

- The code and template structure available in WordPress allows to build custom themes around your exact needs.
- Search engine optimization can be integrated within custom themes from popular plugins. Google has indicated support for WordPress sites as crawl bots can easily index these sites.
- It generally makes content updates simple and efficient compared to other CMS platforms.
- WordPress was originally meant for blogging and still remains a top CMS for simplified blogging.
- Since, managing users is essential for larger websites, so, through WordPress, we can control users, roles and specific permissions.
- It provides an extensive amount of plugins and possibilities for functionality improvements.

Limitations of Using WordPress

1. Since WordPress is so popular it becomes a central target for hackers. The core and plugins are commonly exploited, and require fairly frequent updates to remain secure.
2. Some features like custom post types (content types) may become complex and require some training.
3. Scalability often becomes an issue with very large sites.
4. WordPress is not built around performance; therefore, extended plugins or coding may be required to improve overall performance.

WORDPRESS ADMINISTRATOR LEVEL**WordPress Version: 5.4.2****How to start WordPress:**

There are some basic requirements before starting WordPress development.

- Localhost (XAMPP/WAMPP/LAMPP/IIS)
- WordPress CMS Starter File (Download)

How to download WordPress starter file

- Go to <https://wordpress.org>
- Click on "Get WordPress" Button
- Download Latest Version of WordPress

Priceless, and also free

Download WordPress and use it on your site.

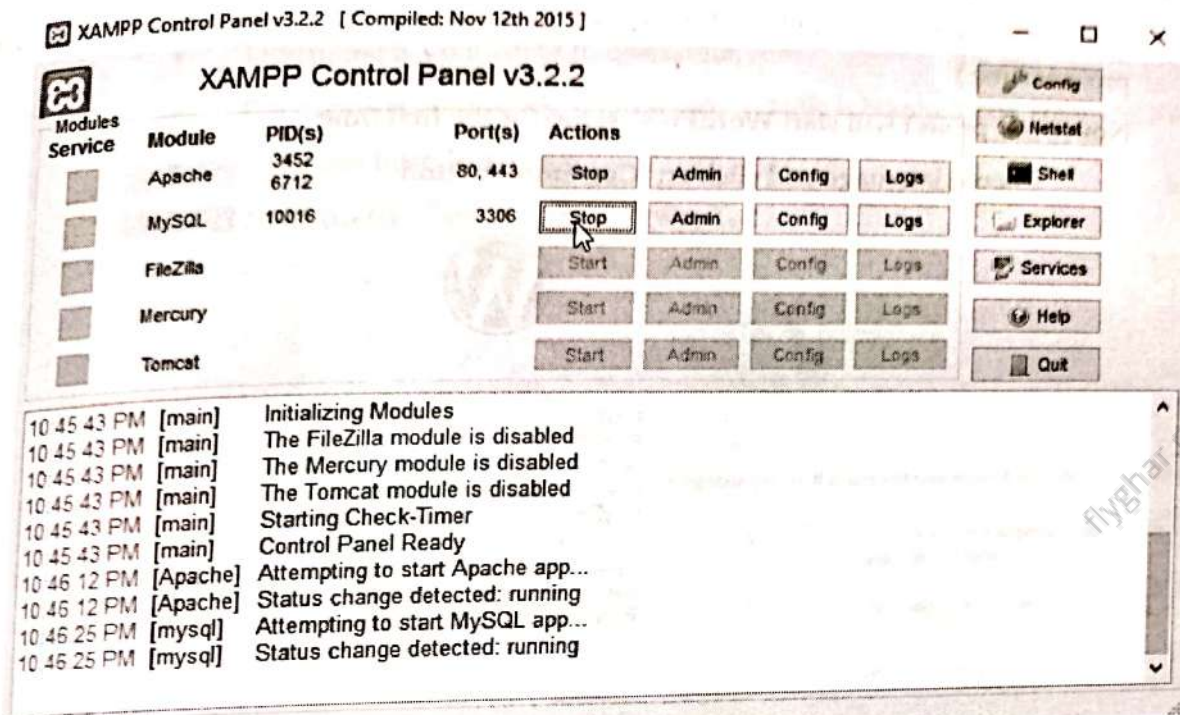
 **Download WordPress 5.3.2**

[Download .tar.gz](#)

How to start website with downloaded WordPress starter file**Start WordPress project using XAMPP****1. Start WordPress project using XAMPP**

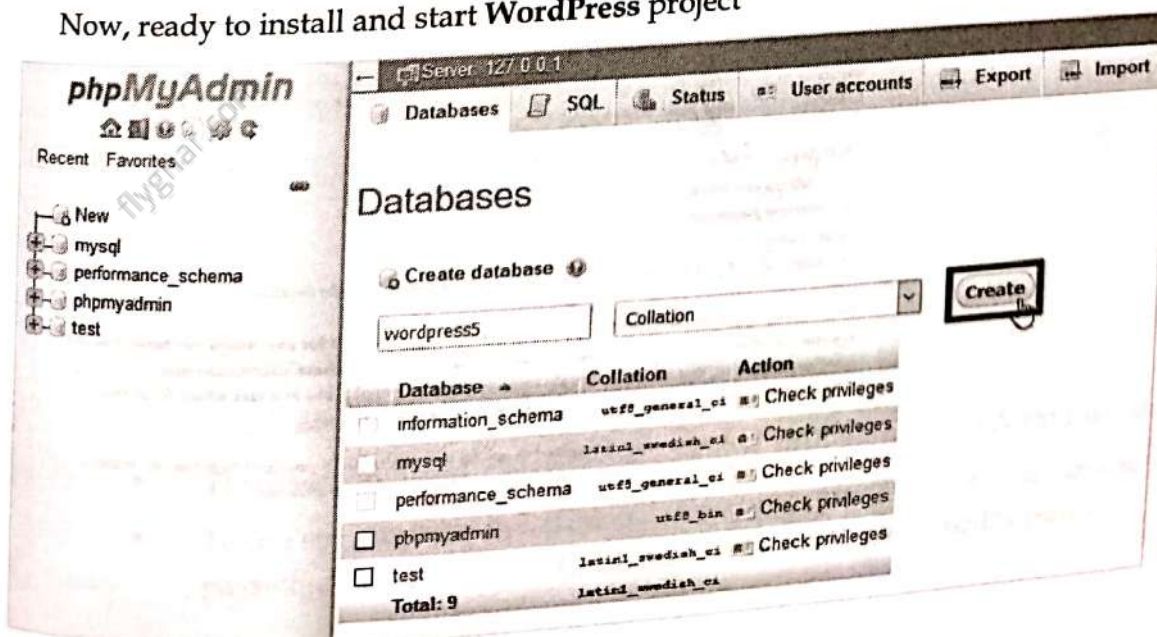
- You need have installed XAMPP
- Turn on XAMPP (Start Apache and MySQL inside XAMPP control panel)
- Extract downloaded WordPress starter file inside **htdocs** > **project** folder of XAMPP
 - o Steps to start project/website by extracting WordPress starter file
 - Navigate to ~C:\XAMPP\htdocs
 - Create project name inside **htdocs** (eg: wptraining) (Note: don't use any special characters on slug/folder name)
 - Extract WordPress root files and folder inside created project folder

Up next, you need to have database created to setup WordPress website.



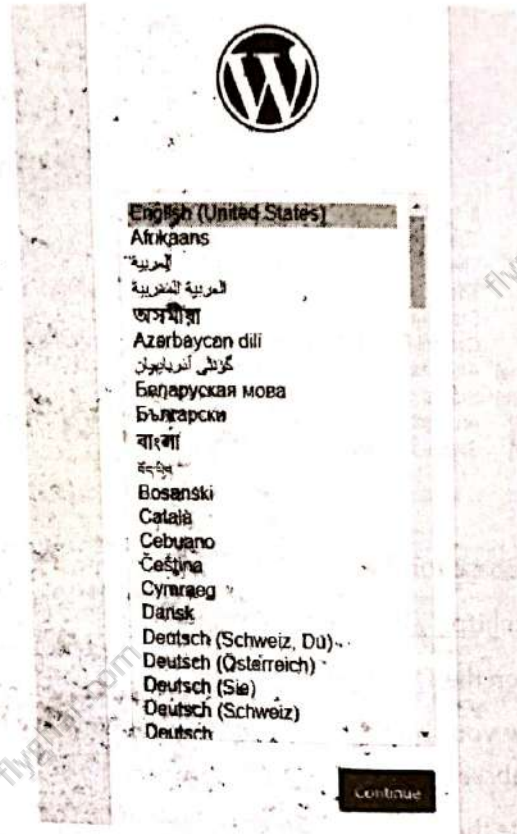
How to create database for the project in XAMPP?

- Navigate to <http://localhost/phpmyadmin>
- And, click on the **Database** tab
 - o Now you should see the option to Create a Database and input field to enter the database name
 - o Write the **database name** (eg: devproject_db) and hit the 'Create' button
 - o You will see a success message in a while
 - o From the list of tables, you can view your database. You are free to use this database wherever you like with default settings. By default the HostName is 'localhost', MySQL user is 'root' and have no password.
- Now, ready to install and start **WordPress** project

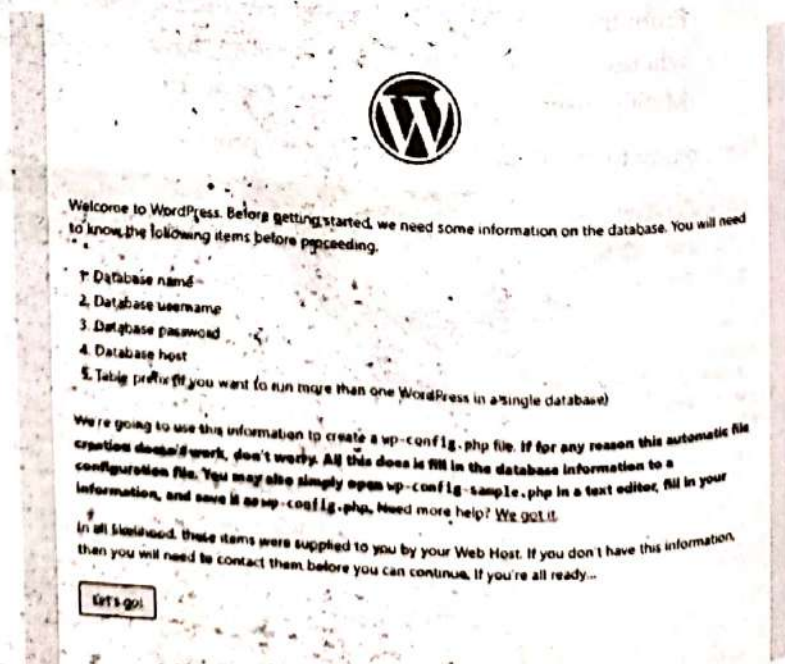


3. How to install/setup WordPress project?

- Open project on localhost (<http://localhost/projectname> --- for now : **wptraining** is your project name)
- Now, your project will start **WordPress** setup for the first time
 - o Choose language and click on **"Continue"** button



- o Read the document for next step and, Click on **"Let's go!"** button (Sometimes, this step will be escaped)



- o Fill all the required field to connect with database you created earlier, example:
 - Database name: devproject_db
 - Username is your MySQL username: root
 - Password is your MySQL password: (Leave it blank, If you are using XAMPP)
 - Database host: localhost
 - Table prefix: (Leave it default: wp_)



Below you should enter your database connection details. If you're not sure about these, contact your host.

Database Name	devproject_db	The name of the database you want to use with WordPress.
Username	root	Your database username.
Password		Your database password.
Database Host	localhost	You should be able to get this info from your web host. If localhost doesn't work.
Table Prefix	wp_	If you want to run multiple WordPress installations in a single database, change this.

- Click on "Submit" button
- o Click on "Run the installation" button



All right, sparky! You've made it through this part of the installation. WordPress can now communicate with your database. If you are ready, time now to...

- o Now, fill up all the required fields
 - Site Title
 - This Username and Password is to login dashboard as an Administrator.
 - To use strong password (Use complex type password: or You can use simple password and check the "Confirm use of weak password" option.

- Email: you can use your Gmail also, this is to get notifications.
- Check the "Discourage search engines to honor this request" option to avoid listing your local website on google.
- Click on "Install WordPress" button



Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title

Username
Username can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password
☒ Weak

Important: You will need this password to log in. Please store it in a secure location.

Confirm Password ☒ Confirm use of weak password

Your Email
Double-check your email address before continuing.

Search Engine Visibility ☒ Discourage search engines from indexing this site
It is up to search engines to honor this request.

- Now, click on "Login" button (You can navigate: <http://localhost/projectname/wp-admin> instead of clicking "Login" button)



Success!

WordPress has been installed. Thank you, and enjoy!

Username dev-admin

Password Your chosen password.

Log In

- Use your "Username and Password" created for dashboard



Username or Email Address

dev-admin

Password

•••••



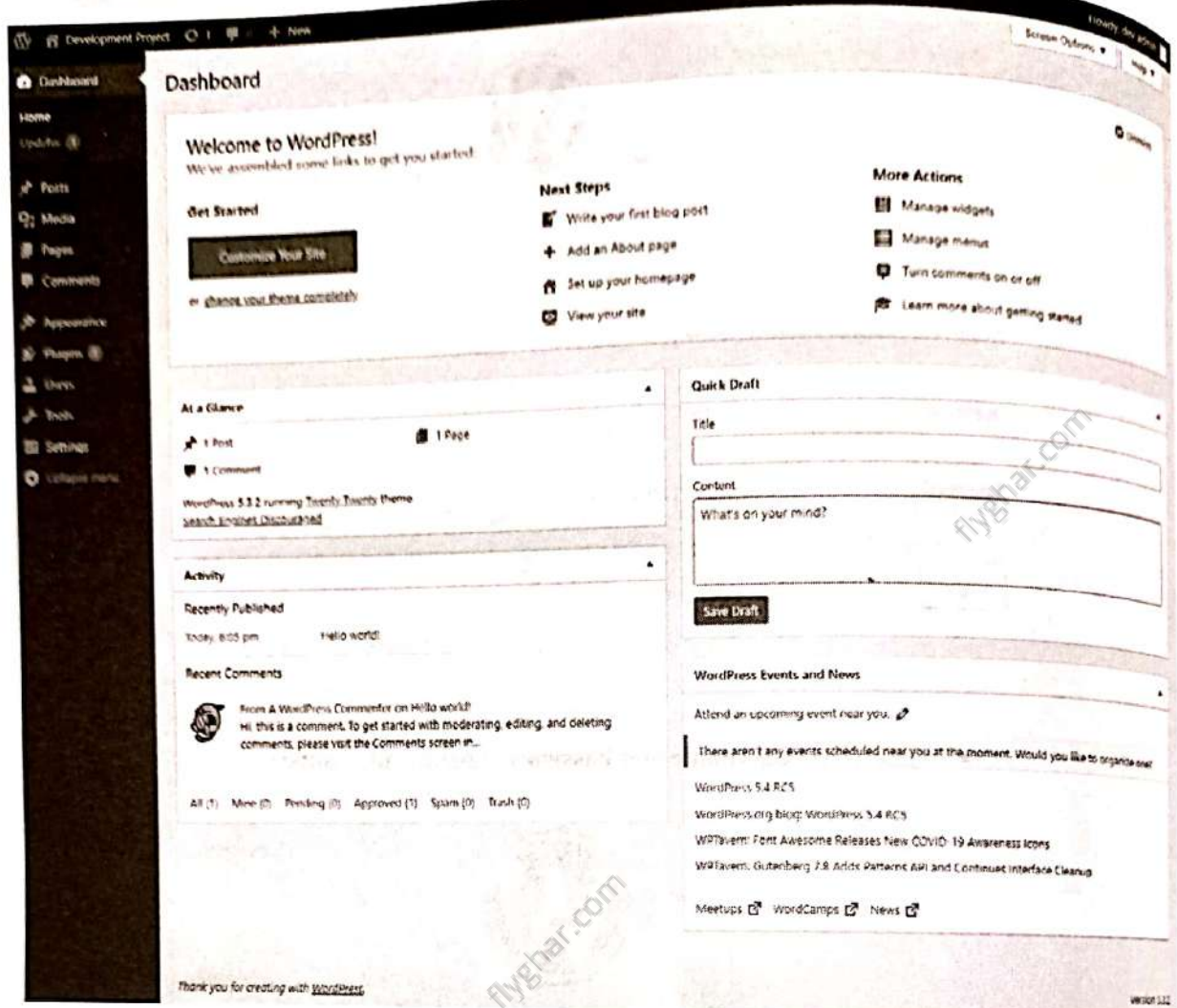
☒ Remember Me

Log In

Lost your password?

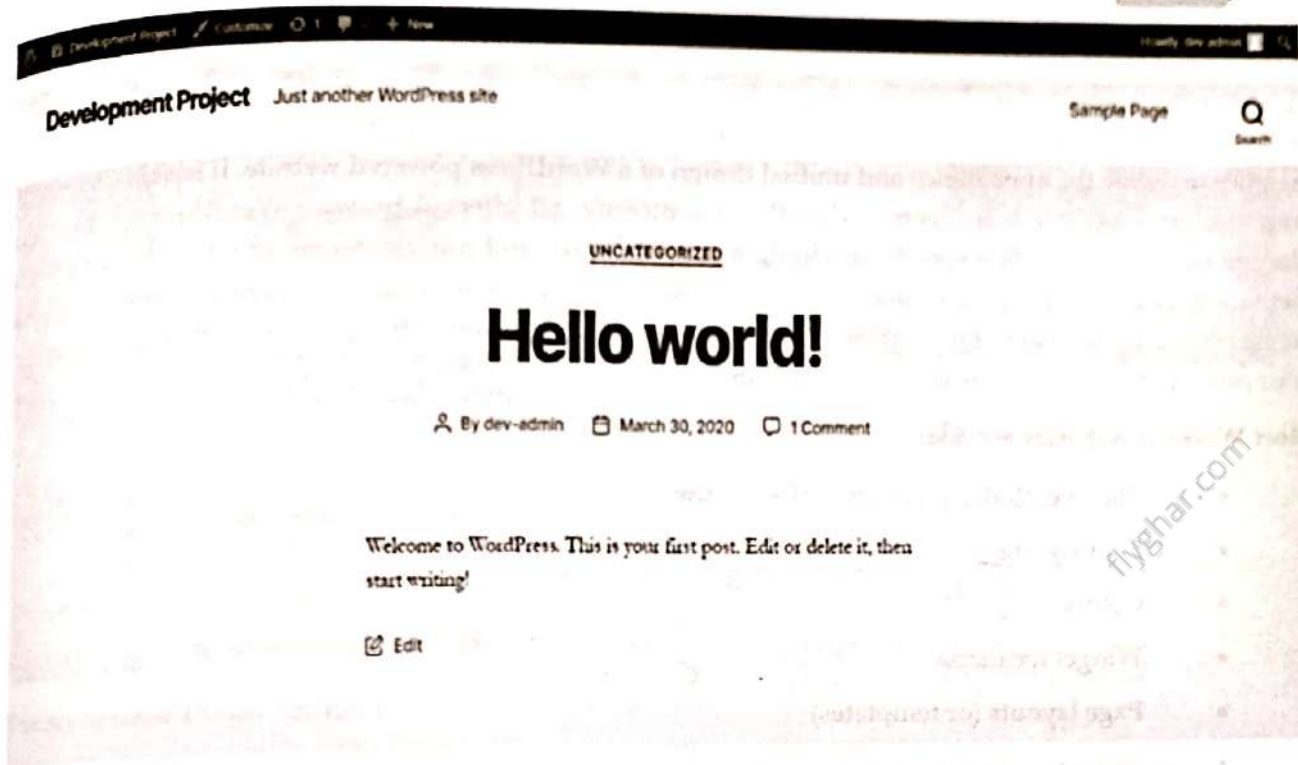
← Back to Development Project

- Now, your project is ready with Dashboard (CMS - Content Management System)



- To view your Project's first look
 - Hover on "Project Title" on top left corner in Dashboard
 - Click on "Visit Site" option





Search ...

SEARCH

Archives

March 2020

Recent Posts

Hello world!

Categories

Uncategorized

Recent Comments

A WordPress Commenter on Hello world!

Meta

Site Admin

- To logout dashboard
 - Hover on "Login User Name" on top right corner of the page
 - Click on "Logout" option
 - It will redirect to login page (You can remove login URL or open new tab to view Project as a User: <http://localhost/projectname>)

How to get inside Dashboard?

- Navigate to wp-admin url (eg: <http://localhost/projectname/wp-admin>)
- If you are already logged in and opened homepage:
 - o Click on Site Title or WordPress Logo on top left corner to get inside dashboard
- Now you are ready to Change theme and customize Active Theme Settings

THEME INTEGRATION

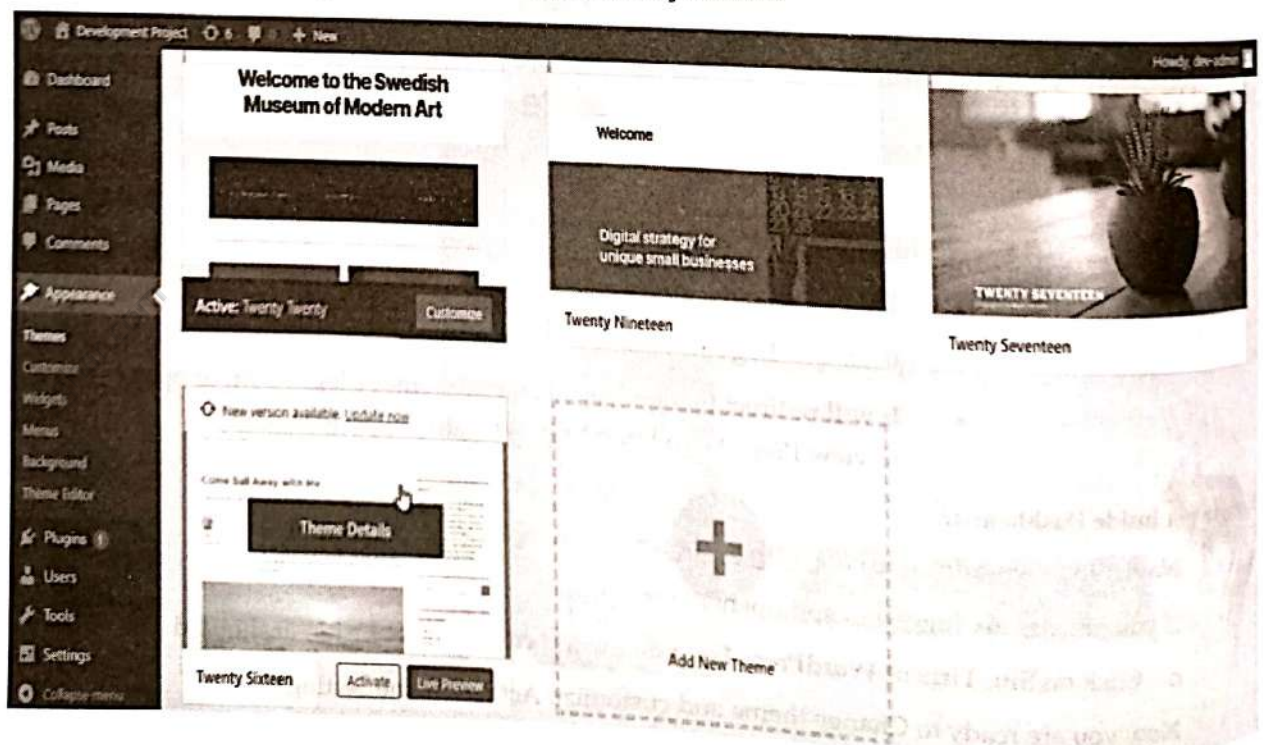
In WordPress, a theme is defined as a collection of files like the templates and stylesheets that work together to define the appearance and unified design of a WordPress powered website. It is outer layout of your site and the user who visits is directly or indirectly get affected by your WordPress theme. The placement of widgets, sliders and everything related to small and minute details about your WordPress theme will affect UI and your visitor's experience. Now it depends on what your niche you are going into before choosing the theme for yourself. There are specific genre theme that you can explore to maximize your output of whatever you are trying to desire.

Most WordPress themes provide:

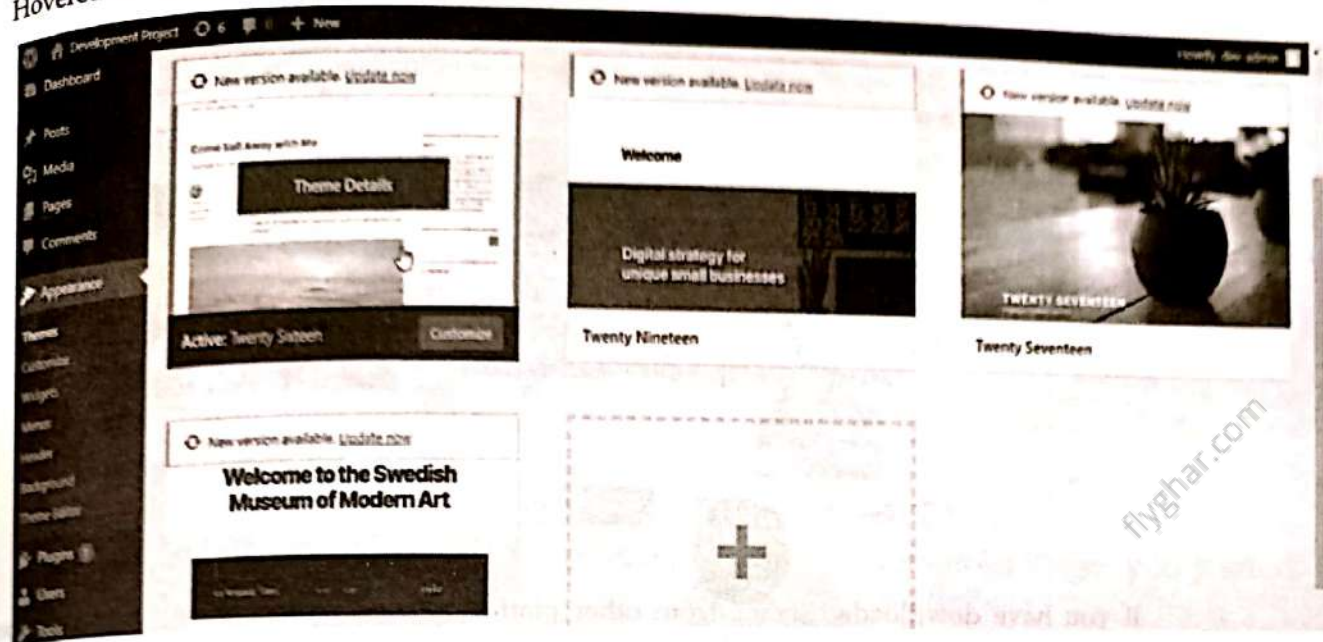
- The overall design or style of your site
- Font styling
- Colors
- Widget locations
- Page layouts (or templates)
- Styles for blog posts and blog archives
- Additional stylistic details

Changing WordPress Default Theme:

- Navigate to **Appearance** menu from left side vertical main bar
 - o It will redirect to **Theme Page** itself, or Click on **Theme** submenu inside **Appearance**
- Now, hover or click on wanted theme from the list and click on **Active**
 - o Example: activate the theme **Twenty Sixteen**



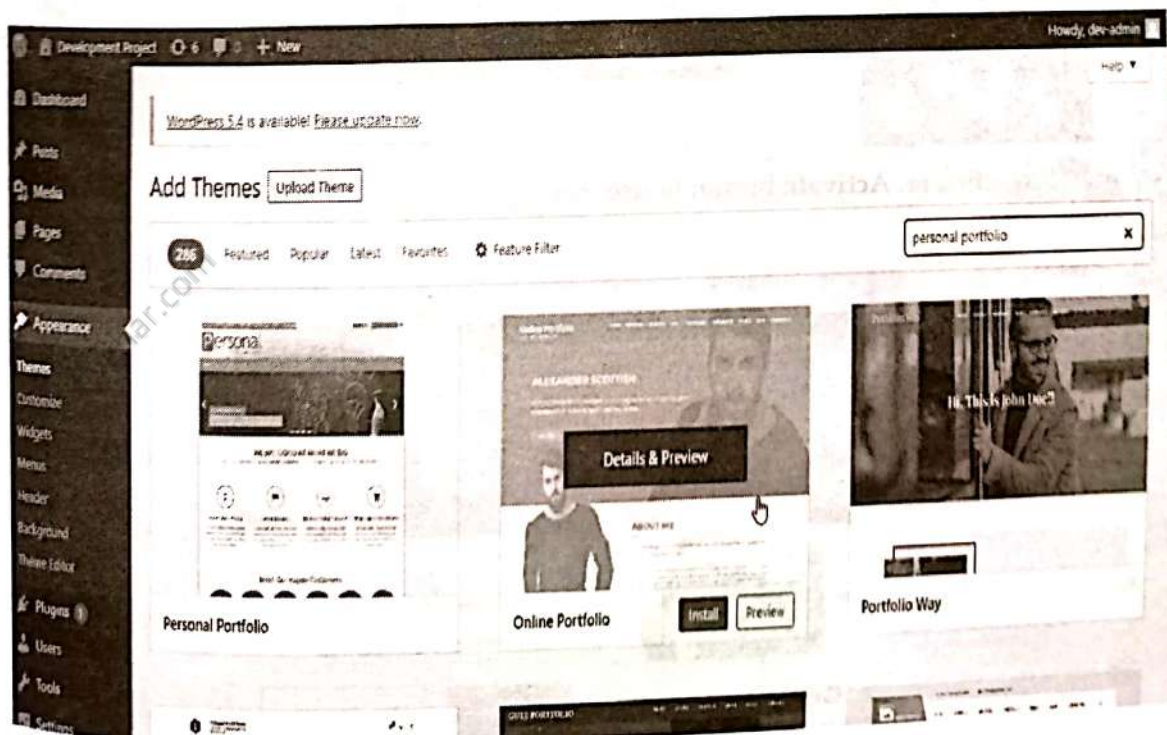
Hovered on Twenty Sixteen Theme



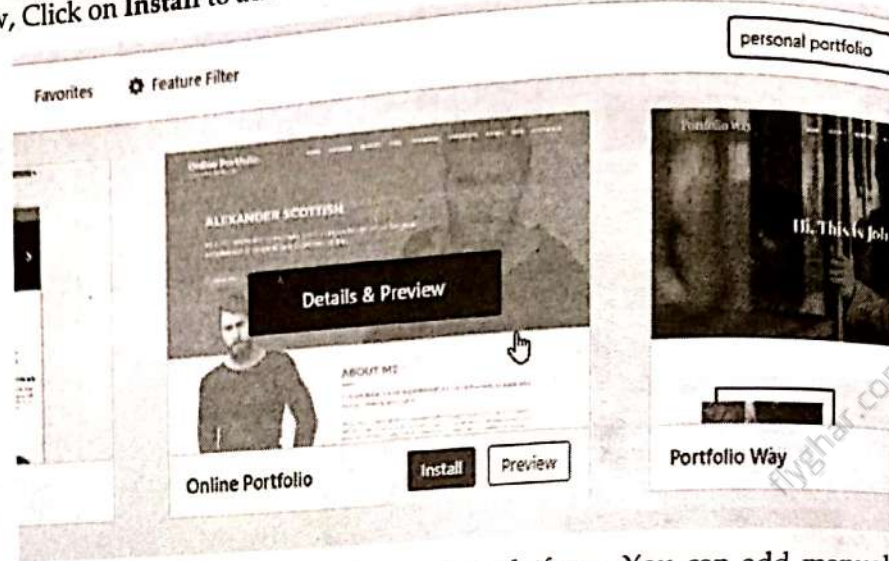
Twenty Sixteen Theme Activated

Adding and Activating WordPress Free Themes collected from online WordPress Official Site

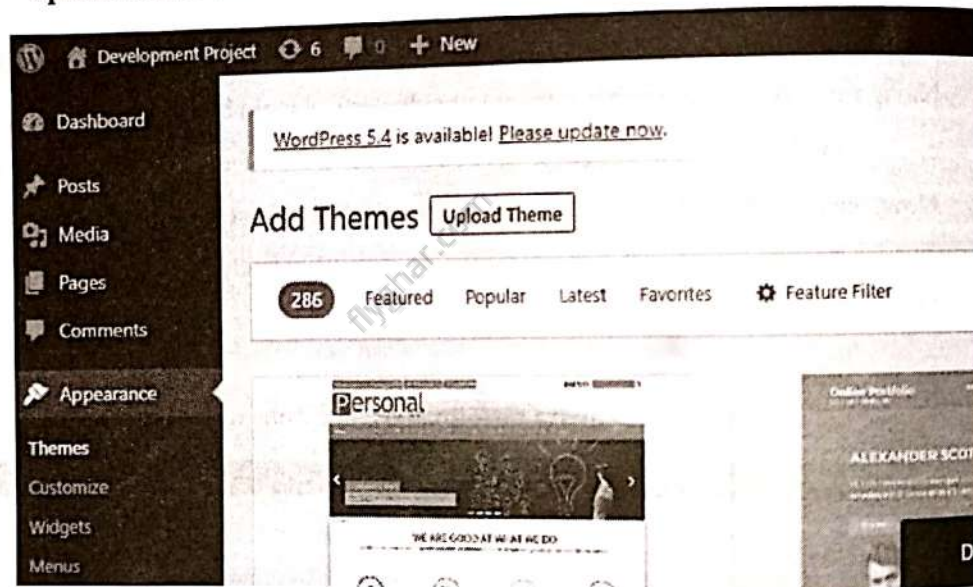
- Navigate to **Appearance** menu from left side vertical main bar
 - o It will redirect to **Theme Page** itself, or Click on **Theme** submenu inside **Appearance**
- Now, click on **Add New Theme** option from the Theme list (Last Option)
- Choose Theme from the list on your requirement (Featured/Popular/Latest/Favorites or Search Manually from Search box)
 - o Example
 - Let's search for **Personal Portfolio** and choose one to use from the list



- For example, here you can choose **Online Portfolio** one
- Now, Click on **Install** to add selected theme to use



- If you have downloaded theme from other platform, You can add manually clicking on **Upload Theme** button



- Now, click on **Activate** button to use it

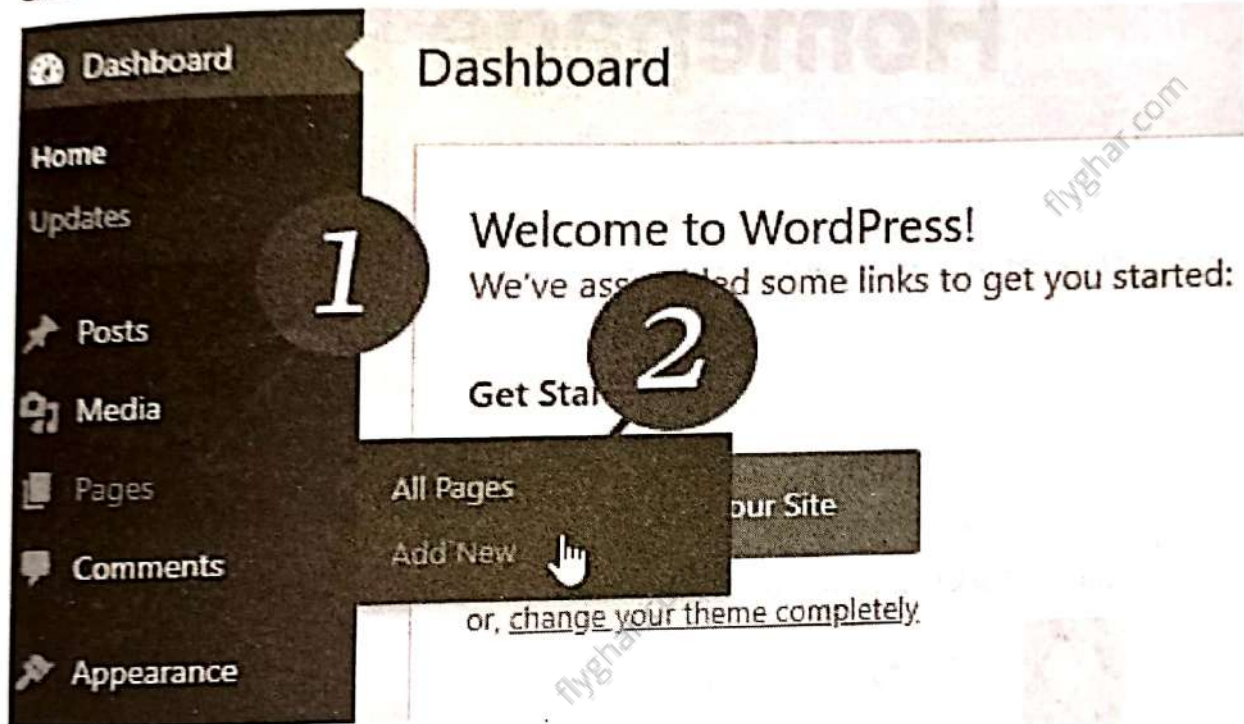


Manage Page

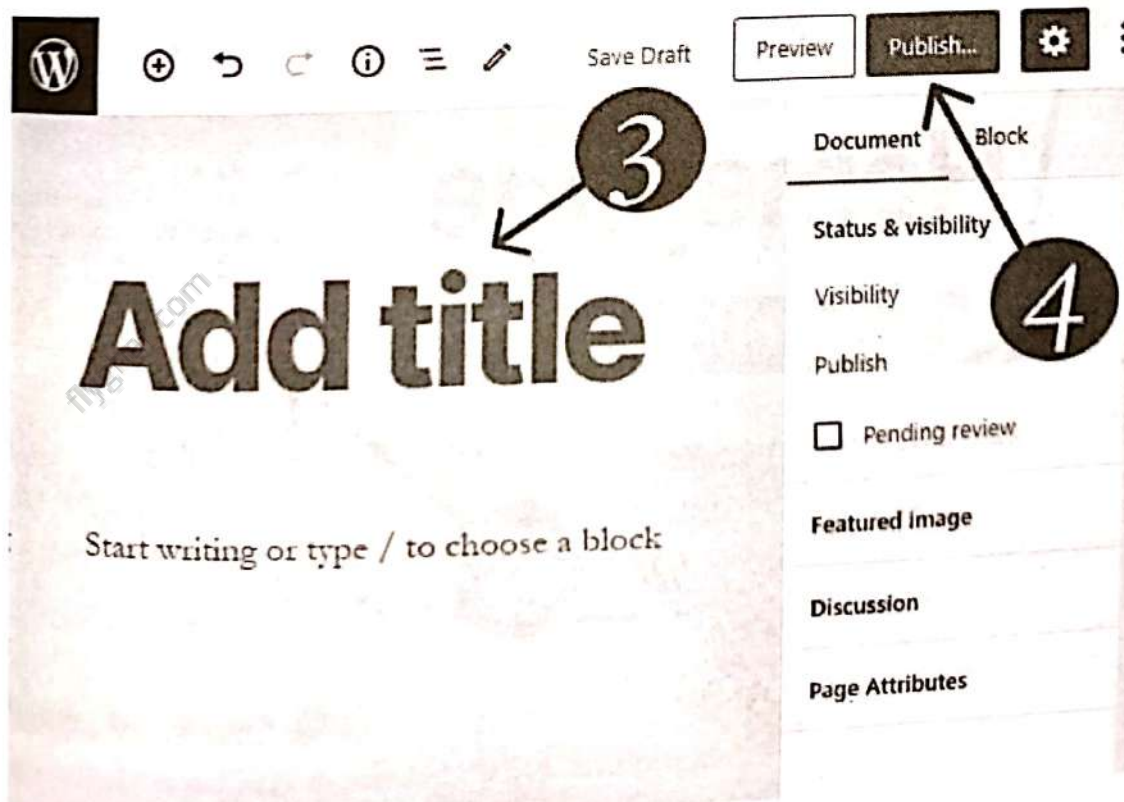
A page in WordPress usually refers to the **page** post type. It is one of the default pre-defined WordPress post types.

How to create a page?

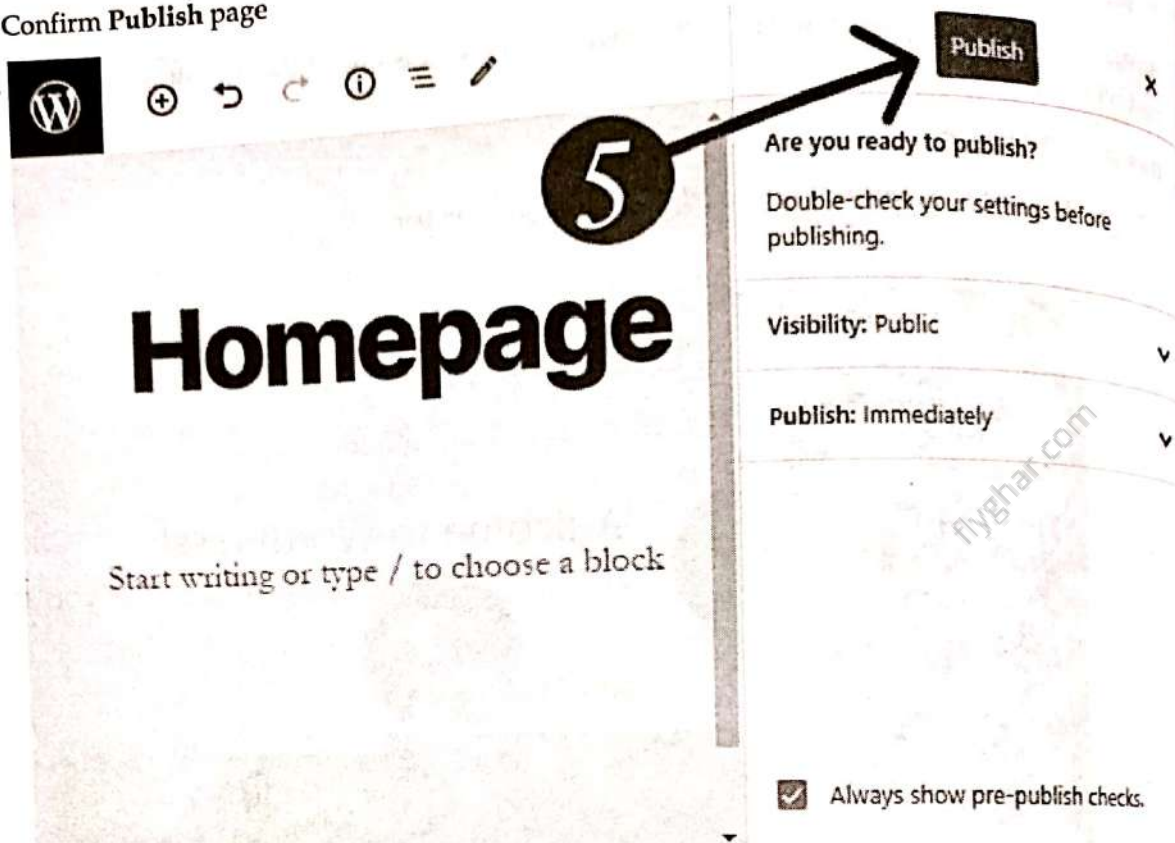
- Navigate to **Page** menu from left side vertical main bar
- Click on **Add New**



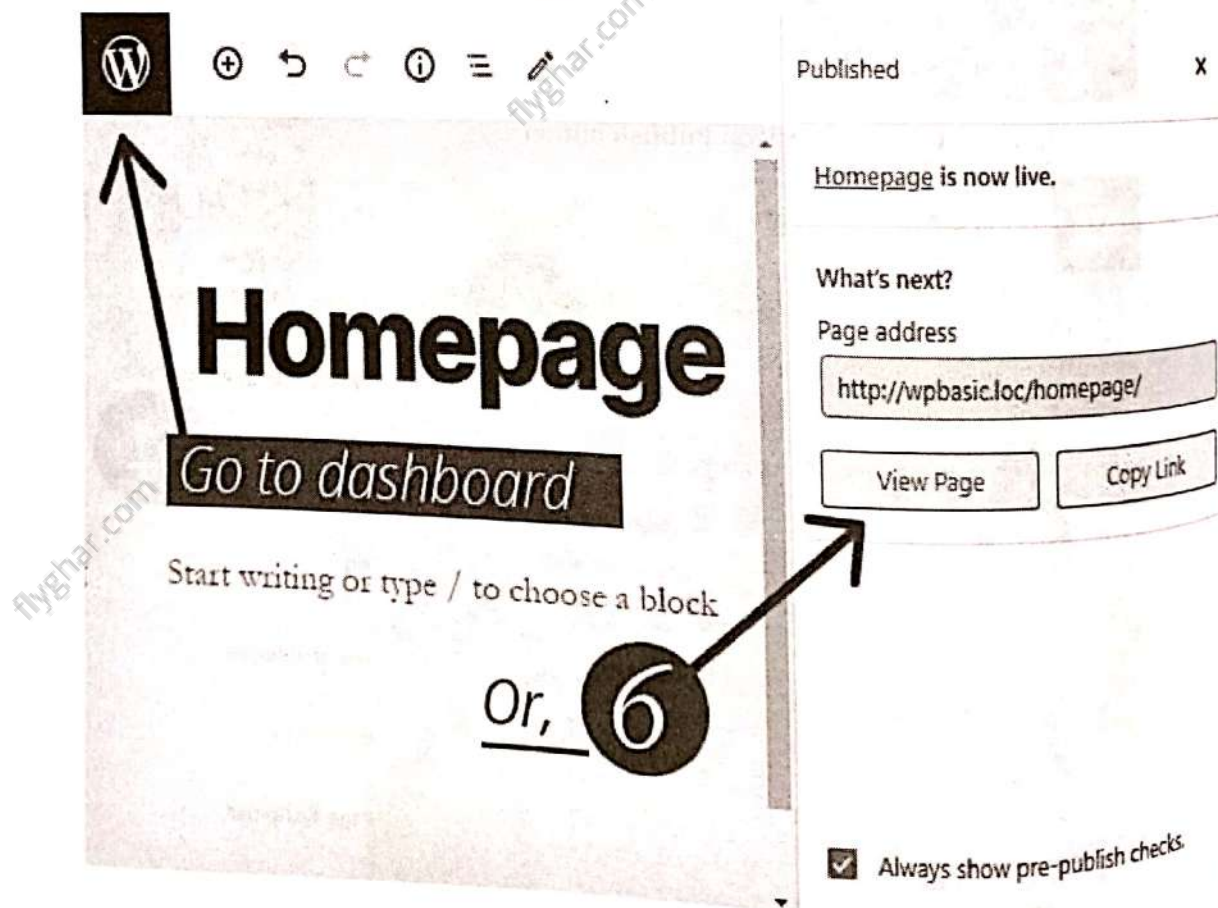
- Add Page Title and click on **Publish** button



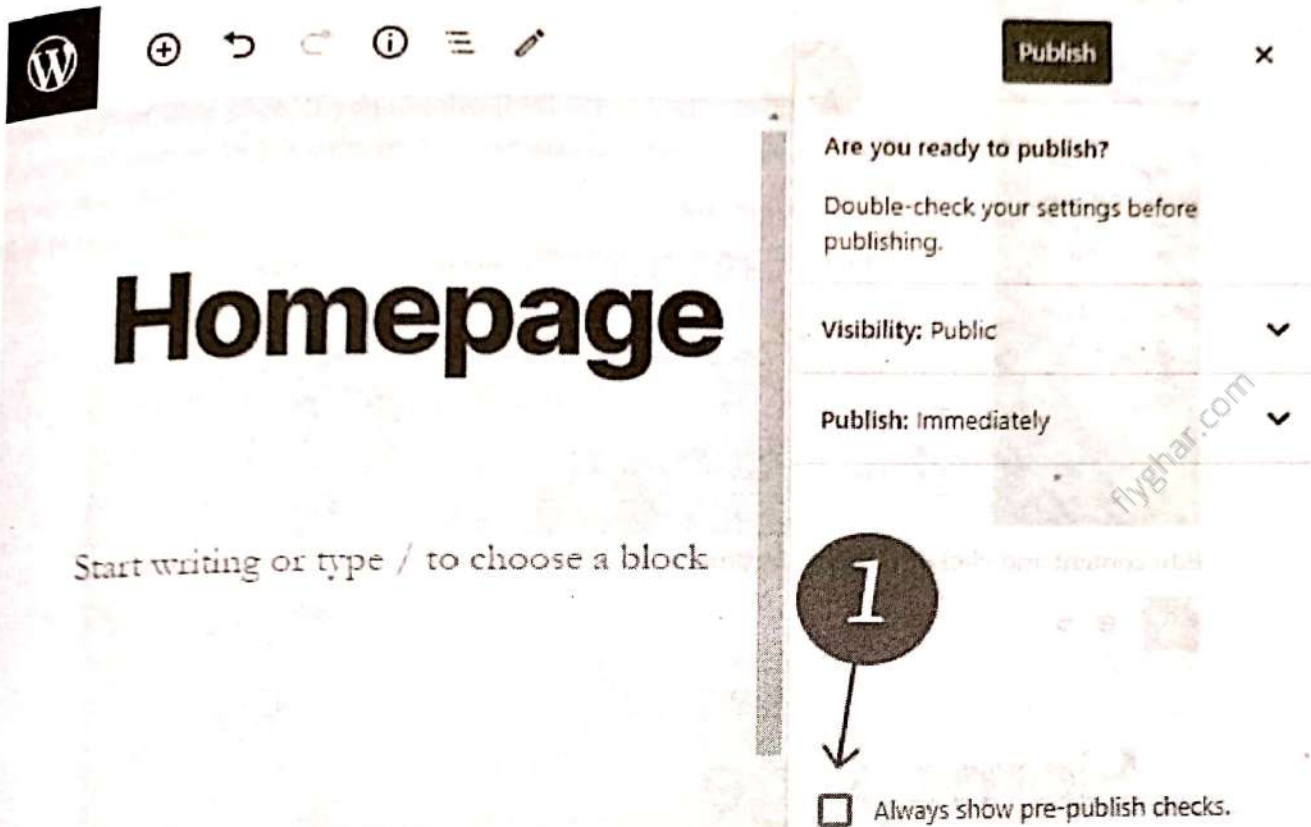
- Confirm Publish page



- Now, go back to Dashboard or View Page

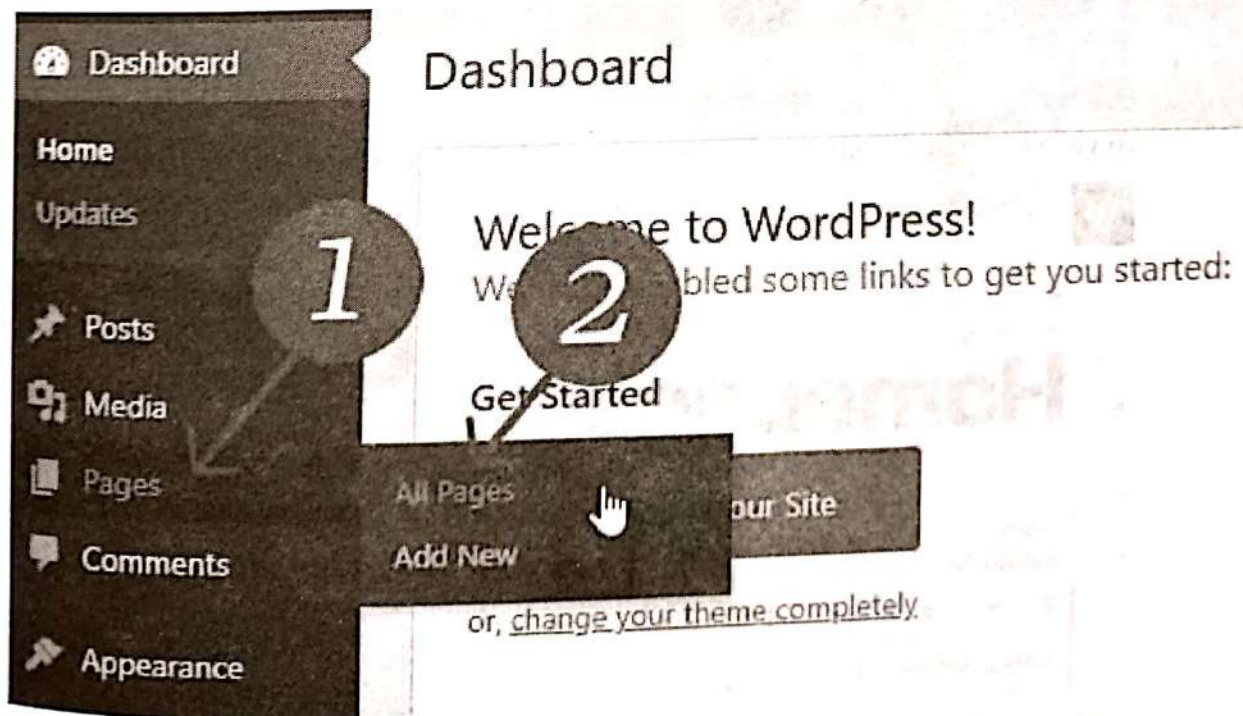


Optional, If you want to remove two step **Publish** action, remove **pre-publish check**

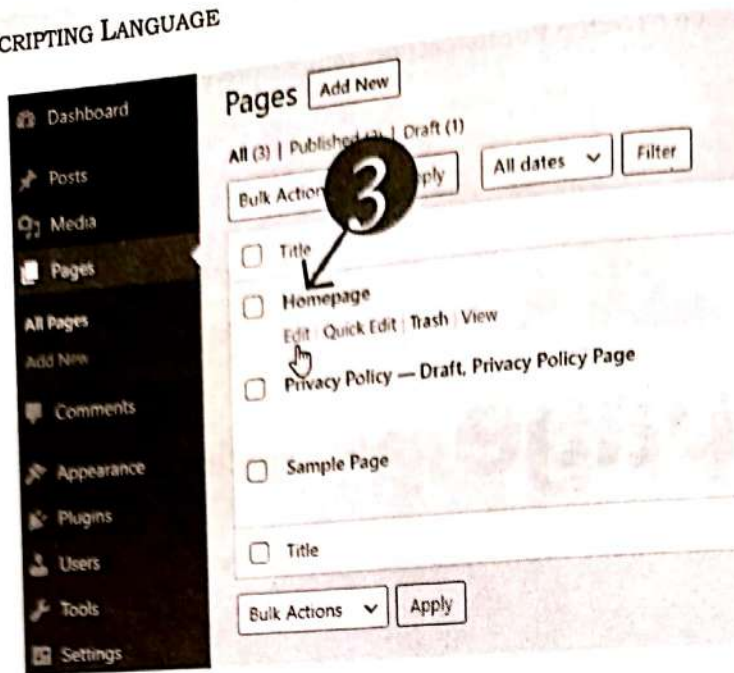


How to edit existing page?

- Navigate to **Page** menu from left side vertical main bar
- Navigate to **All Pages** submenu



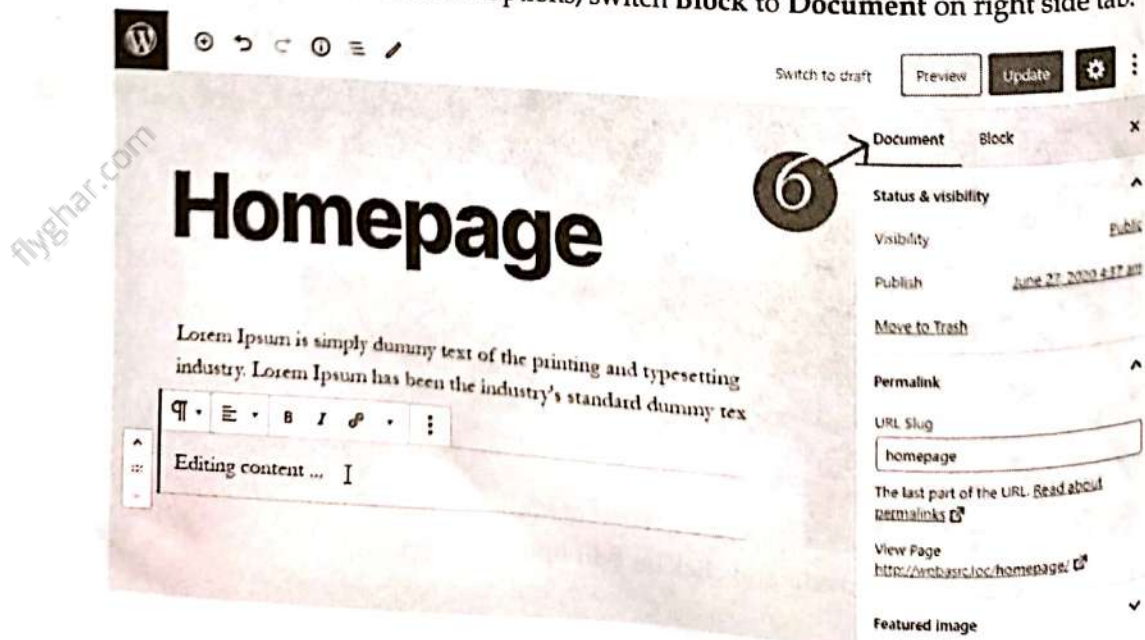
Hover on wanted page to edit and click on **Edit** option



- Edit content and click on Update button



- To view and edit document other options, switch Block to Document on right side tab.

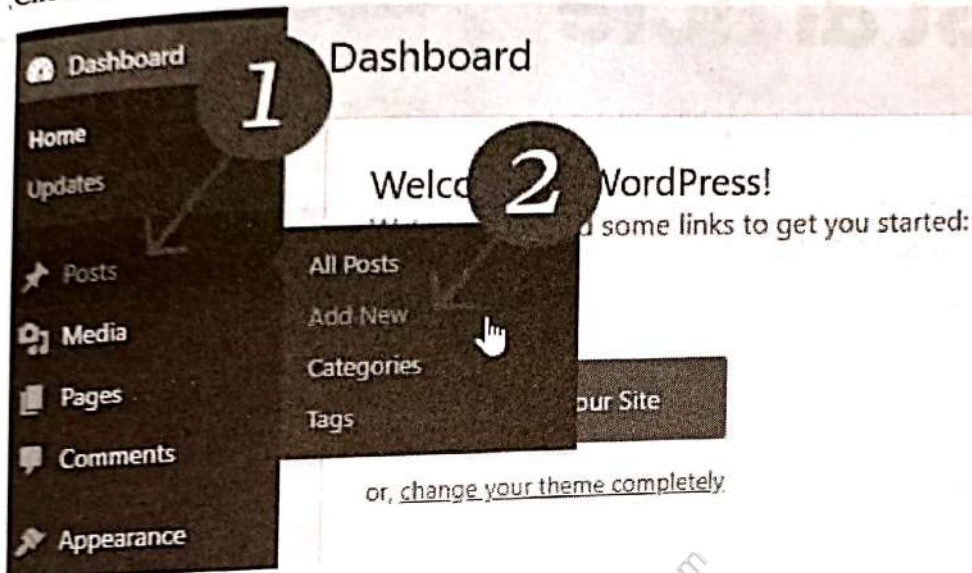


MANAGING POSTS

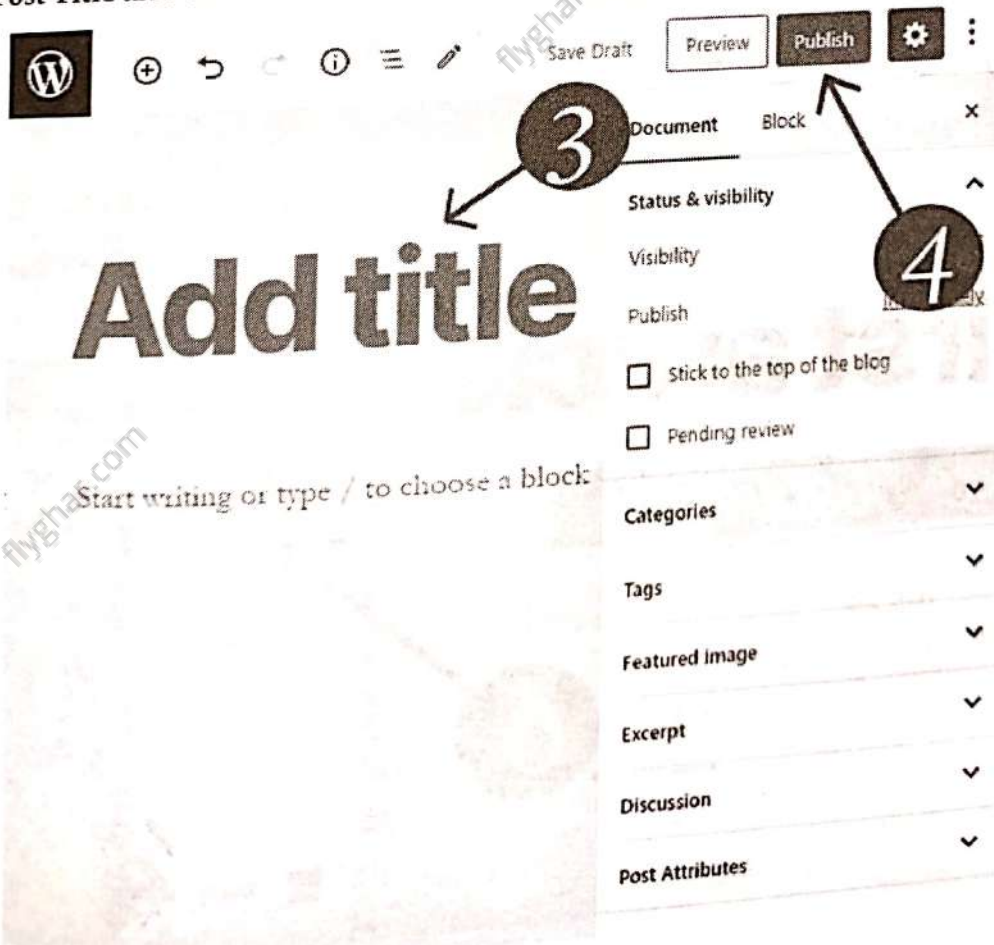
Posts are blog content listed in a reverse chronological order (newest content on top). You will see posts listed on your blog page. If you are using WordPress as a blog, then you will end up using posts for the majority of your website's content. You can also add and edit your WordPress posts from the 'Posts' menu in your dashboard.

How to create a post?

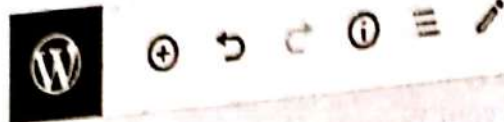
- Navigate to Posts menu from left side vertical main bar
- Click on Add New



- Add Post Title and click on **Publish** button



Confirm Publish page



5

First article

Start writing or type / to choose a block

Publish

Are you ready to publish?

Double-check your settings before publishing.

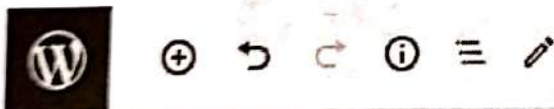
Visibility: Public

Publish: Immediately

Suggestion: Add tags

☒ Always show pre-publish checks.

Now, go back to Dashboard or View Post



First article

Go to dashboard

Start writing or type / to choose a block

Or,

6

Published

First article is now live.

What's next?

Post address

<http://wpbasic.loc/first-article/>

View Post

Copy Link

☒ Always show pre-publish checks.

Optional, If you want to remove two step **Publish** action, remove **pre-publish check**

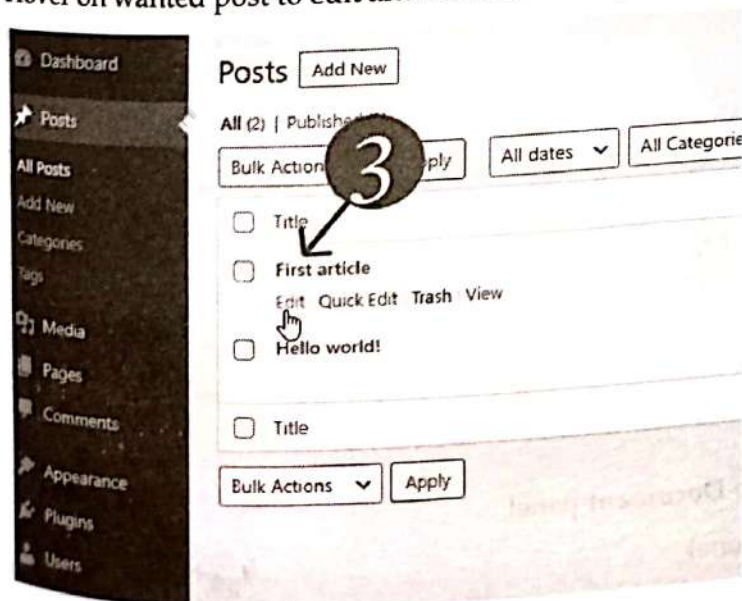


How to edit existing post?

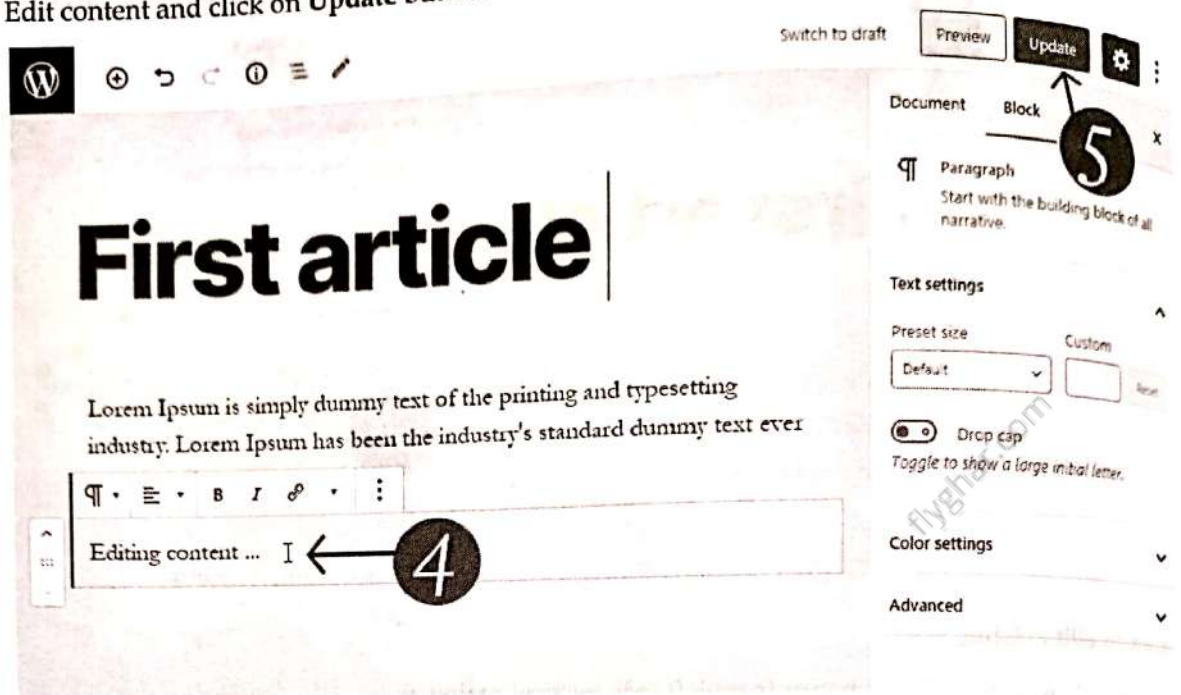
- Navigate to **Posts** menu from left side vertical main bar
- Navigate to **All Posts** submenu



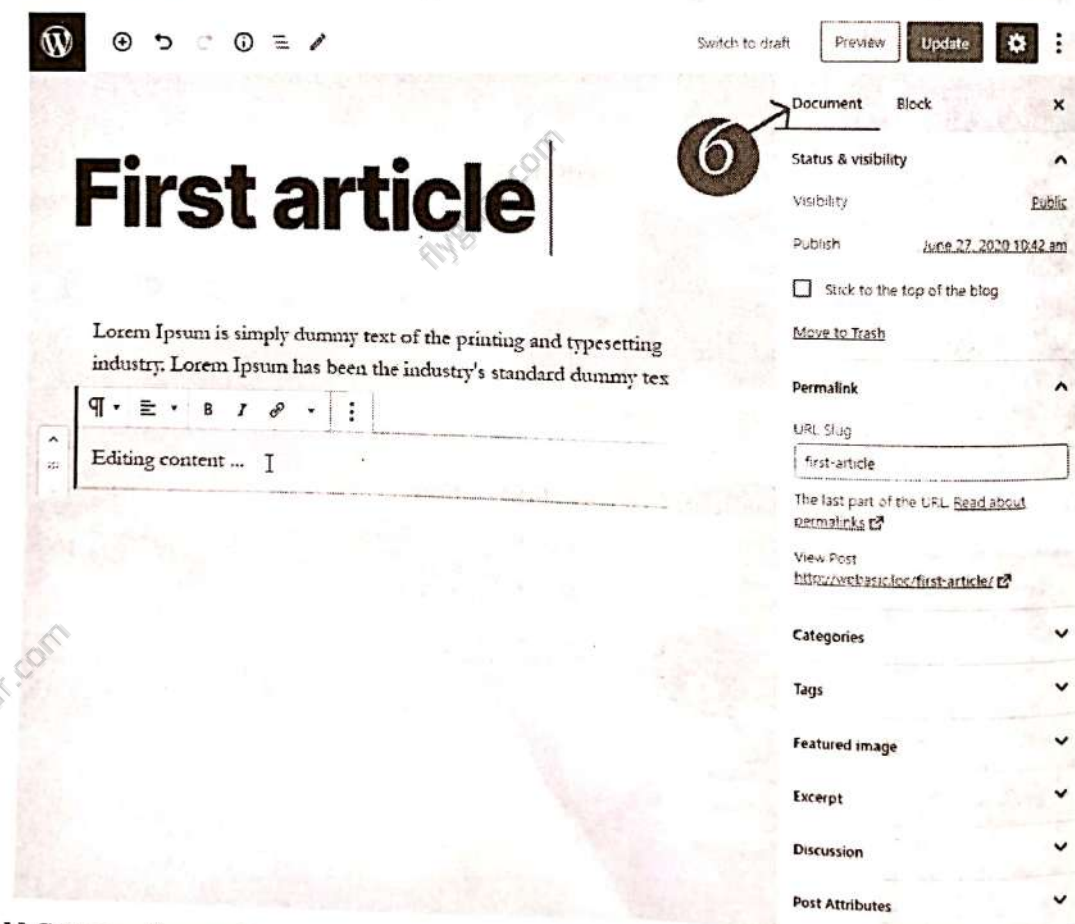
- Hover on wanted post to edit and click on **Edit** option



- Edit content and click on Update button



- To view and edit document other options, switch Block to Document on right side tab.



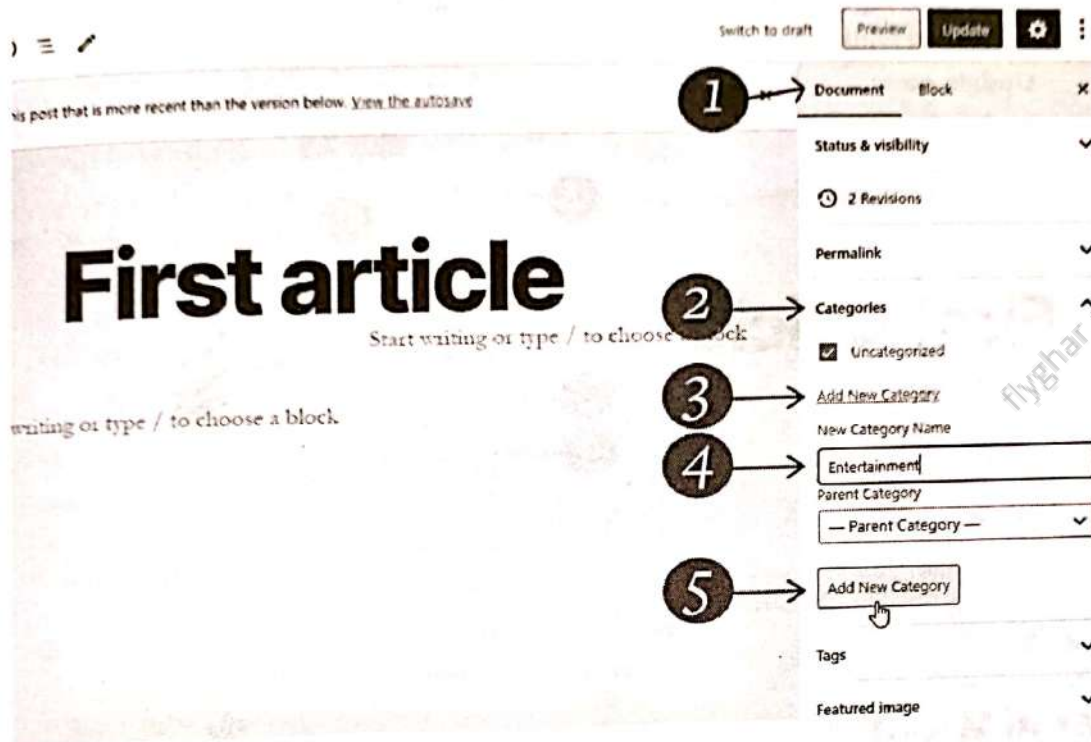
How to add Category for post?

- Add or edit post & Go to Document panel
- Expand Category sub-panel

Click on **Add New Category**

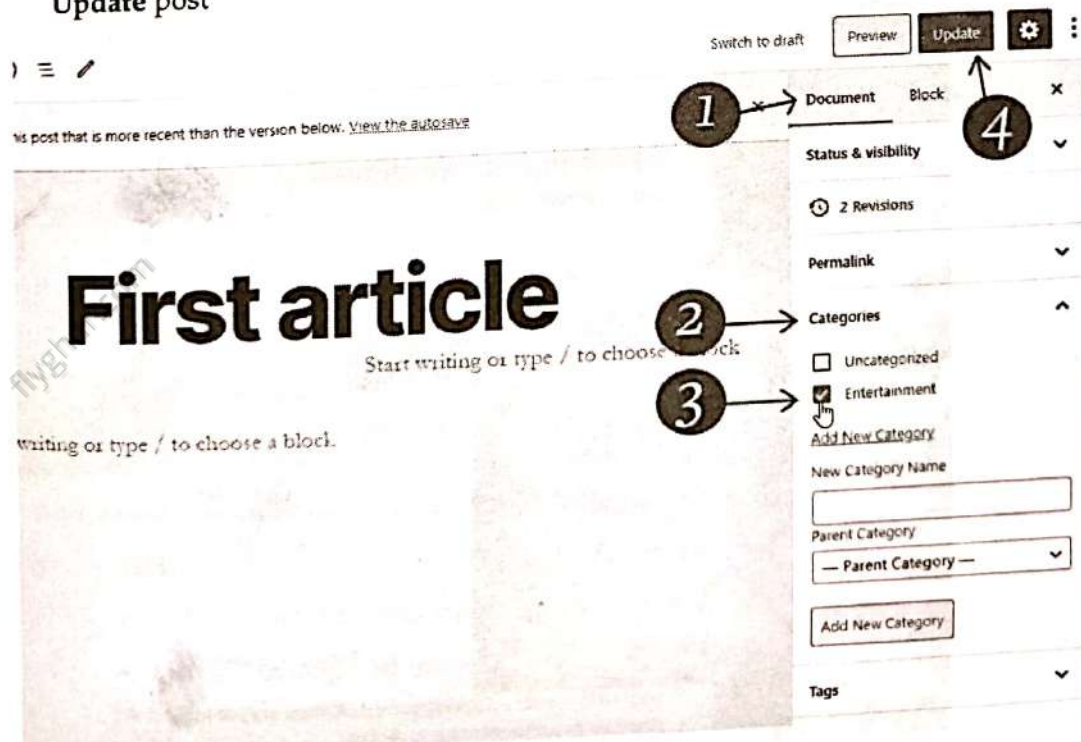
Write a category name inside **New Category Name** box

Click on **Add New Category** button



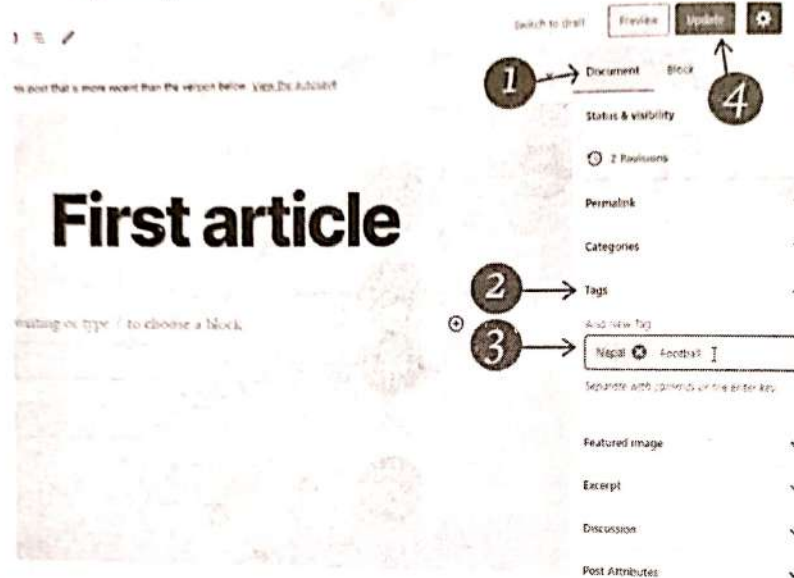
How to use existing Category for post?

- Add or edit post & Go to **Document** panel
- Expand **Category** sub-panel
- Choose and Check wanted category (eg: **Entertainment**)
- **Update** post



How to add and use Tags for post?

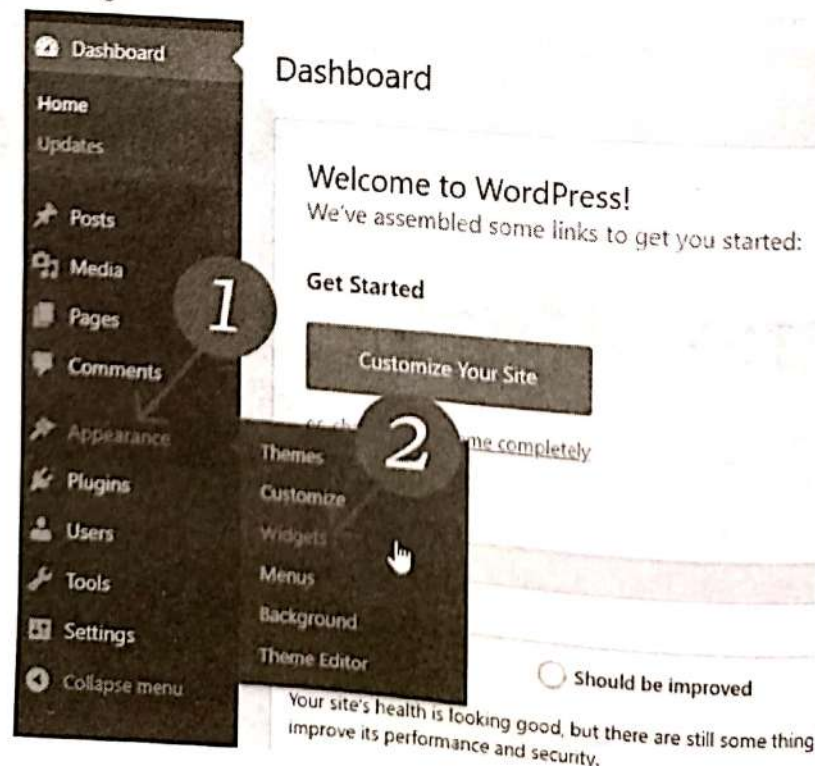
- Add or edit post & Go to Document panel
- Expand Tags sub-panel
- Write hash tag and place comma at the end of tag word. (You can use multiple tags separating with comma)
- Update post

**MANAGING WIDGETS**

In WordPress, widgets are blocks of content that you can add to your site's sidebars, footers, and other areas.

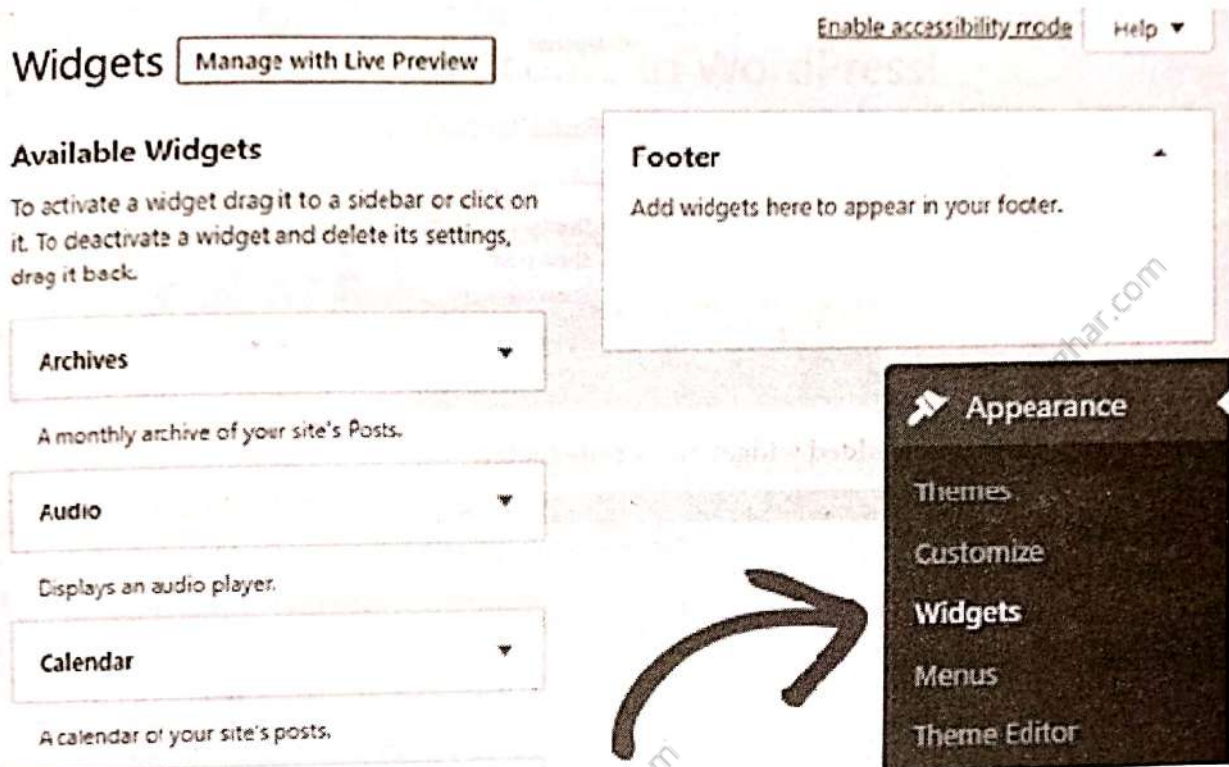
How to use widget?

- Navigate to **Appearance** menu from left side vertical main bar
- Click on **Widgets**

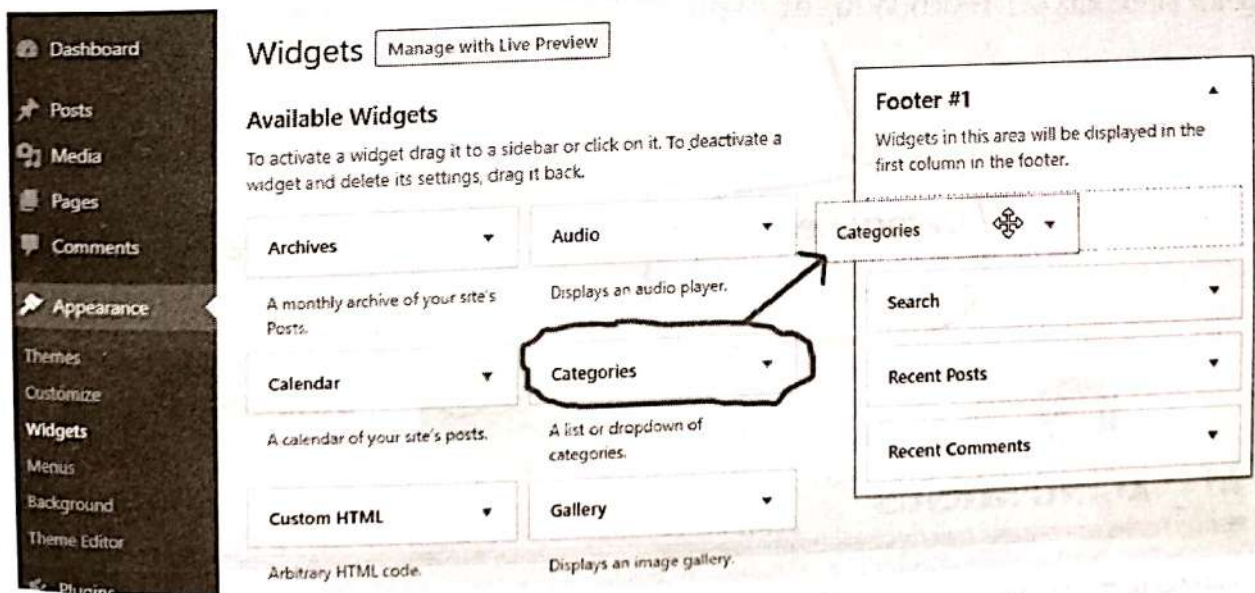


Now, you will see created **Widget Areas** on right side & **Widget Collections** on left side

- (eg: **Footer** widget area)
- (eg: **Archives, Categories** etc. widgets)



To use, drag and drop wanted widget on Widget area which is associated on theme required place.



And, manage options as your need (Title, and other options)

- Note:
 - If you just drag and place widget, you don't need to do save manually
 - If you manage other option inside widget after added inside widget area, you have save edited options.

Footer #1

Widgets in this area will be displayed in the first column in the footer.

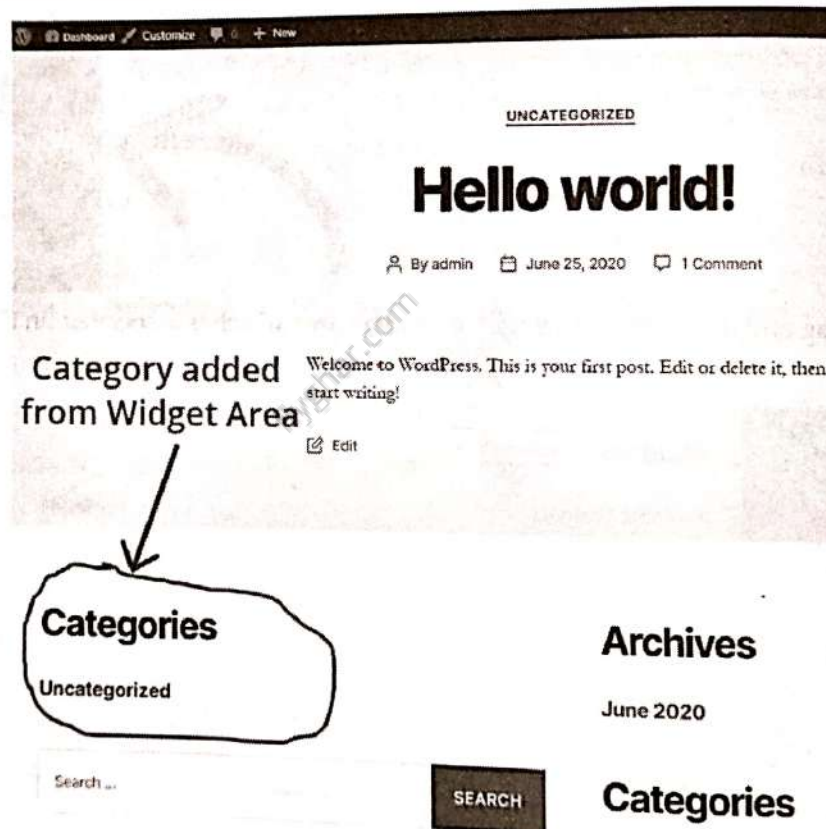
Categories

Title:

☐ Display as dropdown
☐ Show post counts
☐ Show hierarchy

[Delete](#) | [Done](#) [Save](#)

Now, check added widget on website footer

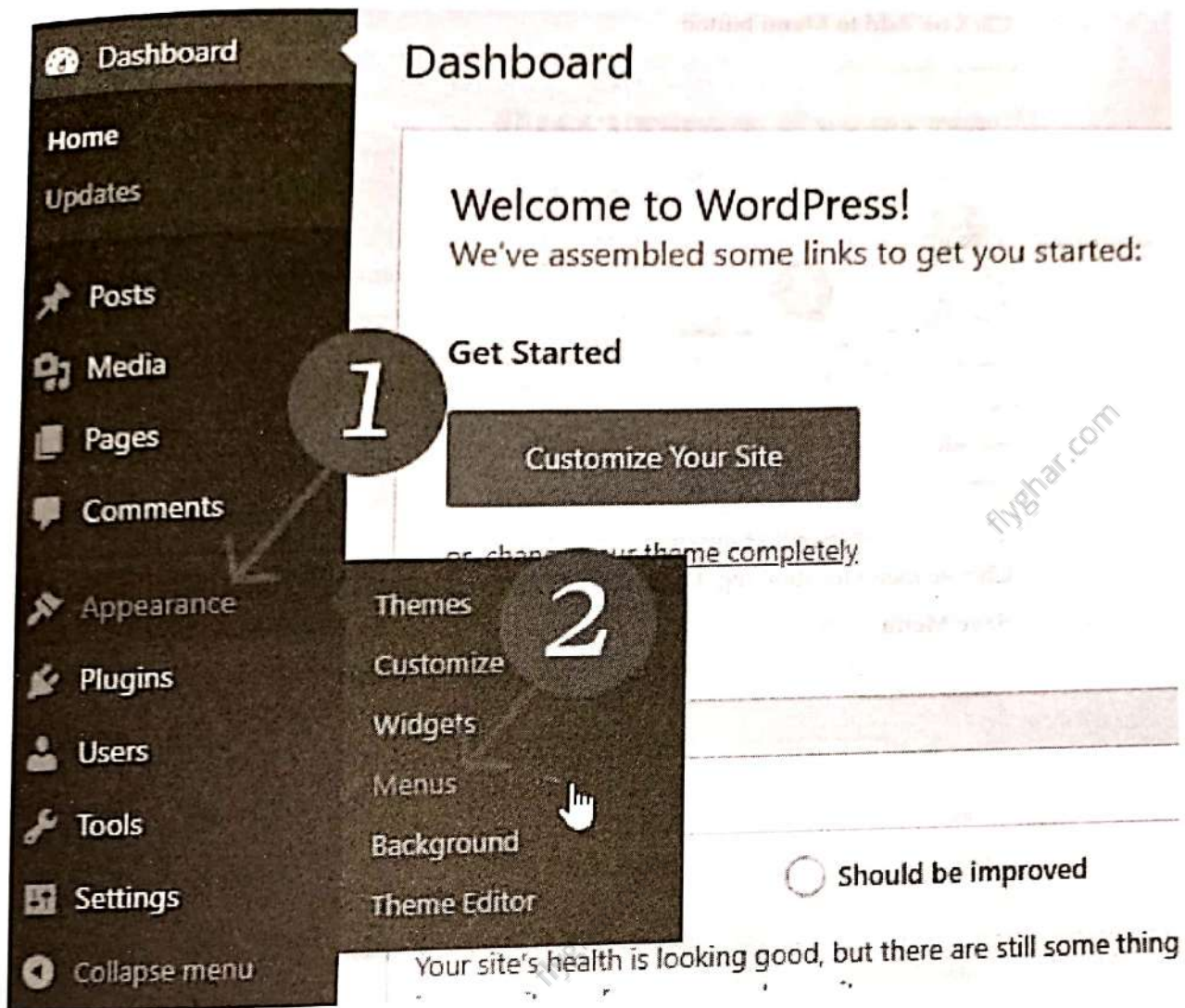


MANAGING MENUS

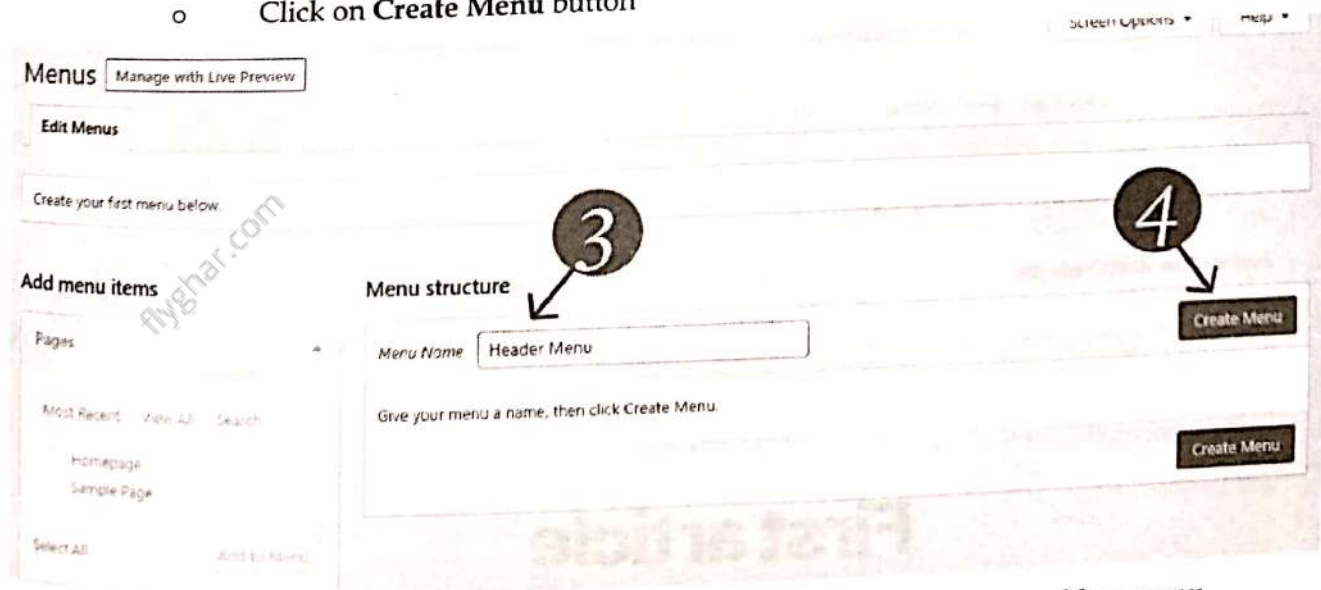
Navigation Menus, or simply Menus, are a WordPress theme feature which allows users to create navigation menus by using built-in Menu Editor located in WordPress admin area under Appearance » Menus.

How to create Menu?

- Navigate to **Appearance** menu from left side vertical main bar
- Click on **Menus**



- Now, **Menu structure**
 - o Write *Menu Name*
 - o Click on **Create Menu** button



- Now, to add menu items, from the left panel of **Pages**, Select page to add on menu

Click on Add to Menu button

- Choose menu location (eg: Desktop Horizontal Menu) under Menu Settings
- Save Menu.

Check on website page

Just another WordPress site





DISCUSSION EXERCISE

1. What is OOP, Describe advantage of OOP?
2. What is class and object? Describe how we can create object and class using PHP.
3. How to define properties and method in PHP?
4. Difference between constructor and destructor with example.
5. What is polymorphism? Describe different types of polymorphism with example.
6. What is encapsulation? Describe with proper example.
7. How inheritance can be achieved in PHP? Describe with proper example.
8. Describe use of static members with example.
9. What is exception handling, how can you achieve exception handling in PHP?
10. What is AJAX? Describe different method of implementing AJAX in PHP.
11. What is JQuery? Describe with proper example.
12. Describe different JQuery UI function with example.
13. Write a program to hide and unhide element using JQuery.
14. What is CMS? Describe feature of CMS.
15. What is Joomla? Describe different steps of Joomla installation.
16. What is extensions? Describe use of Joomla extensions.
17. What is template? Write different steps to create custom template in Joomla.
18. What is module? Write different steps to create module in Joomla.
19. What is component? Write different steps to create component in Joomla.
20. What is MVC? Describe how MVC framework works.
21. What is WordPress? Describe feature of WordPress.
22. Write down different steps of WordPress installation.
23. Write down different steps of WordPress theme integration.
24. Write down different steps to create page in WordPress.
25. Write down different steps to manage menu in WordPress.
26. Write down different steps to manage widgets in WordPress.
27. Write down different steps to manage post in WordPress.

□□□

PROJECT WORK

A Simple CRUD Operation with Session Cookie:

Using object oriented approaches for database Connectivity.

HTML, CSS code is based on Bootstrap Framework (v4.3.1).

- A single admin user should create using command line or phpmyadmin.
- The user should login into application, after successful login user will redirect to dashboard.
- The user can create category from category menu.
- The user can list all category list added into database table.
- The user can edit, view and delete category from list screen.
- The user will logout from application from account menu.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int(11)			No	None		AUTO_INCREMENT
2	name	varchar(50)	latin1_swedish_ci		No	None		
3	email	varchar(50)	latin1_swedish_ci		No	None		
4	username	varchar(50)	latin1_swedish_ci		No	None		
5	password	varchar(50)	latin1_swedish_ci		No	None		
6	status	tinyint(1)			Yes	0		
7	gender	char(6)	latin1_swedish_ci		No	None		
8	last_login	datetime			Yes	NULL		
9	created_at	datetime			No	None		

Database table structure for Project

Users Table: tbl_users

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int(11)			No	None		AUTO_INCREMENT
2	name	varchar(100)	utf8_general_ci		No	None		
3	rank	int(11)			No	None		
4	status	tinyint(1)			No	0		
5	created_by	int(100)			No	None		
6	updated_by	int(100)			Yes	NULL		
7	created_at	datetime			No	None		
8	updated_at	datetime			Yes	NULL		

Categories Table: categories

Database connection: connection.php

```
<?php
$connect = new mysqli('localhost','root','','db_sample_project');
if ($connect->connect_errno != 0) {
    die('Database Connection Error' . $connect->connect_error);
}
?>
```


Common Header Page: header.php

```

<?php @session_start(); ?>
<!DOCTYPE html>
<html lang="">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title><?php echo $title ?></title>

<link rel="stylesheet" href="../css/fontawesome.min.css">
<link href="css/bootstrap.min.css" rel="stylesheet">
<link rel="stylesheet" href="css/index.css">
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/3.7.2/animate.min.css">
</head>
<body>
<nav class="navbarnavbar-expand-lgnavbar-light bg ">
    Sample Project
    <button class="navbar-toggler hidden-lg-up" type="button" data-toggle="collapse"
data-target="#collapsibleNavId" aria-controls="collapsibleNavId"
aria-expanded="false" aria-label="Toggle navigation"></button>
    <div class="collapse navbar-collapse" id="collapsibleNavId">
    <ul class="navbar-navmr-auto mt-2 mt-lg-0">
    <li class="nav-item active">
    <a class="nav-link" href="home.php">Home <span class="sr-only">(current)</span></a>
    </li>
    <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" id="dropdownId" data-toggle="dropdown"
aria-haspopup="true" aria-expanded="false">Category</a>
    <div class="dropdown-menu" aria-labelledby="dropdownId">
    <a class="dropdown-item" href="create_category.php">Create</a>
    <a class="dropdown-item" href="list_category.php">List</a>
    </div>
    </li>
    <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" id="dropdownId" data-toggle="dropdown"
aria-haspopup="true" aria-expanded="false">Account</a>
    <div class="dropdown-menu" aria-labelledby="dropdownId">
    <?php if (isset($_SESSION['admin_username'])) { ?>
    <a class="dropdown-item" href="index.php">login</a>
    <a class="dropdown-item" href="register.php">register</a>
    <?php } else { ?>
    <a class="dropdown-item" href="logout.php">Logout</a>
    <?php } ?>
    </div>
    </li>
    </ul>
    </div>
</nav>

```

University
Faculty of Humanities & Social Sciences
OFFICE OF THE DEAN

Examination 2019

Bachelor in Computer applications

Course Title: Scripting Language

Code No: CACS 254

Semester: IV

Full Marks: 60
Pass Marks: 24
Time: 3 hours

Candidates are required to answer the questions in their own words as far as possible.

Group A

Attempt all the questions.

10×1=10

1. Circle (O) the correct answer.
 - i) \$.foo() is equivalent to
 - a) javascript.foo()
 - b) document.foo()
 - c) jQuery.foo()
 - d) None of the above
 - ii) What are the advantages of Ajax?
 - a) Bandwidth utilization
 - b) More interactive
 - c) Speeder retrieval of data
 - d) All of these
 - iii) Which option will you select to find the unpublished Articles in Joomla?
 - a) Component Manager
 - b) Media Manager
 - c) Article Manager
 - d) None of the above
 - iv) Which of the functions is used to sort an array in descending order in PHP?
 - a) sort()
 - b) asort()
 - c) rsort()
 - d) dsort()
 - v) Which two predefined variables are used to retrieve information from forms?
 - a) \$GET & \$SET
 - b) \$_GET & \$_SET
 - c) \$_GET & \$_SET
 - d) GET & SET
 - vi) What is the correct JavaScript syntax to write "Hello World"?
 - a) System.out.println("Hello World")
 - b) println ("Hello World")
 - c) response.write("Hello World")
 - d) document.write("Hello World")
 - vii) Which clause is used to "Modify the existing field of the table"?
 - a) ALTER
 - b) FROM
 - c) SELECT
 - d) MODIFY
 - viii) In MVC architecture, the model is referred to
 - a) Shape of data
 - b) Html content
 - c) Collection of data
 - d) Type of data
 - ix) Which Key is used to link two tables in Mysql?
 - a) Primary Key
 - b) Foreign Key
 - c) Both a) and b)
 - d) None of the mentioned
 - x) Which of the following provides the size of the uploaded file in PHP?
 - a) \$_FILES['file']['tmp_name']
 - b) \$_FILES['file']['name']
 - c) \$_FILES['file']['size']
 - d) \$_FILES['file']['type']

Group B

Attempt any SIX questions.

[6×5 = 30]

Explain the concept, structure and importance of Document Object Model (DOM).

What is JavaScript? Write a JavaScript to print largest & smallest among 10 elements of an array.

What is dialog box? Explain different dialog boxes with suitable example in JavaScript.

Explain two HTTP functions to accept the user values from interface with suitable PHP program.

What is the use of JQuery? Write the sample program to show and hide the certain <div> with use of JQuery.

Why Ajax is required in web development? Write an sample program to synchronize data between JavaScript and PHP.

Why is CMS? Write the steps to create sub menu in WordPress or Joomla.

Group C

Attempt any TWO questions.

[2×10 = 20]

Design the following form in HTML and display the value in the result box after calculating basic arithmetic operations based on user input with use of JavaScript.

Basic Arithmetic Functions

Operand

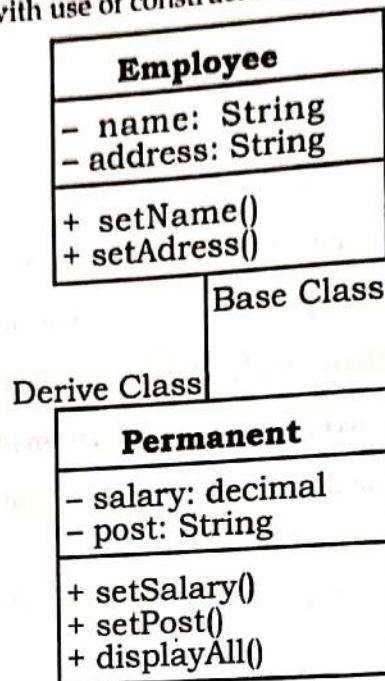
Operand

AddSubMul

DivMod

Result:

10. Write an Object Oriented PHP Program to implement the concept of inheritance in considering with following class diagram with use of constructor for 20 students.



11. Design following form in HTML and write corresponding PHP and SQL code to store the user's values after satisfying following validation rules.
- Length of full name up to 40 characters.
 - Email address must be valid email address.
 - Username must start with string followed by number.
 - Password length must be more than eight characters.

Assume your own database and database connectivity constraints if necessary.

Register

*required fields

Your Full Name*:

Email Address*:

User Name*:

Password*:

□□□