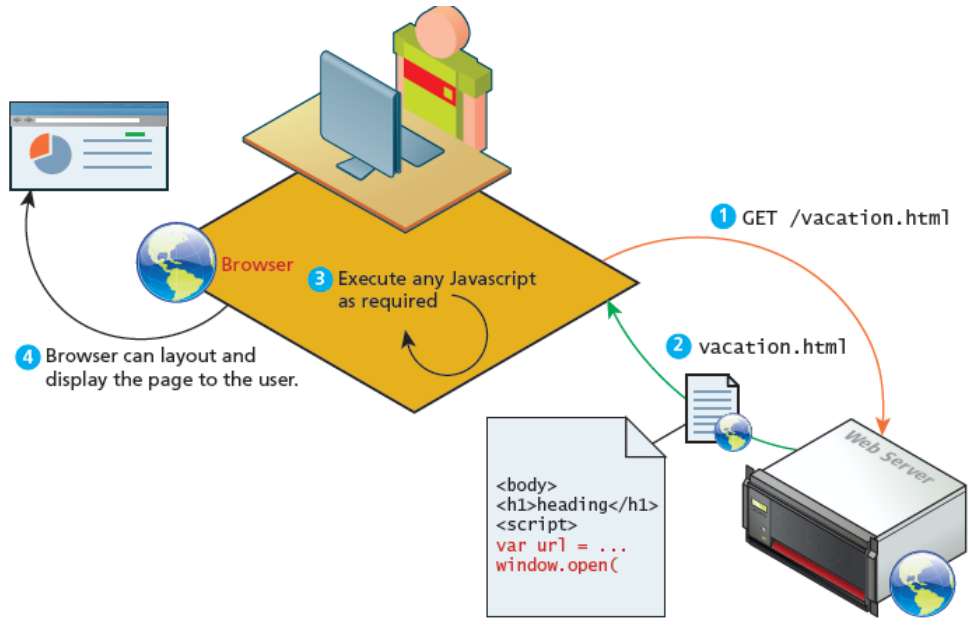
Module III
JavaScript & PHP

3.1 What is JavaScript and What can it do?

- Javascript is an object-oriented, dynamically typed, scripting language.
- It is primarily a **client-side scripting** language. It is completely different from the programming language *Java*.
- JavaScript is an object based language, with only few object-oriented features.
- It runs directly inside the browser, without the need for the JVM. Whereas, Java is fully object oriented language, which runs on any platform with a Java Virtual Machine
- JavaScript is dynamically typed (also called weakly typed) - variables can be easily converted from one data type to another. The data type of a variable can be changed during run time. Java is statically typed, the data type of a variable is defined by the programmer (e.g., int abc) and enforced by the compiler.

Client-Side Scripting

- Client-side scripting is important one in web development. It refers to the client machine (i.e., the browser), which runs code locally.
- It doesn't depend on the server to execute code and return the result. However, if required the client machine can download and execute JavaScript code received from server. The execution of script, takes place at the client.
- There are many client-side languages like Flash, VBScript, Java, and JavaScript.



Advantages of client-side scripting:

- Processing can be done on client machines

- Reduces the load on the server
- Faster response, the browser responds more rapidly to user events than a request to a remote server
- JavaScript can interact with the HTML

Disadvantages of client-side scripting-

- Required functionality might be housed on the server
- The script that works in one browser, may generate an error in another.
- Heavy web applications can be complicated to debug and maintain. JavaScript often uses inline HTML that are embedded into the HTML of a web page. This has the distinct disadvantage of blending HTML and JavaScript together. This methodology decreases code readability, and increases the difficulty of web development.

There are other client-side approaches to web programming,– **Action script & Java applets.**

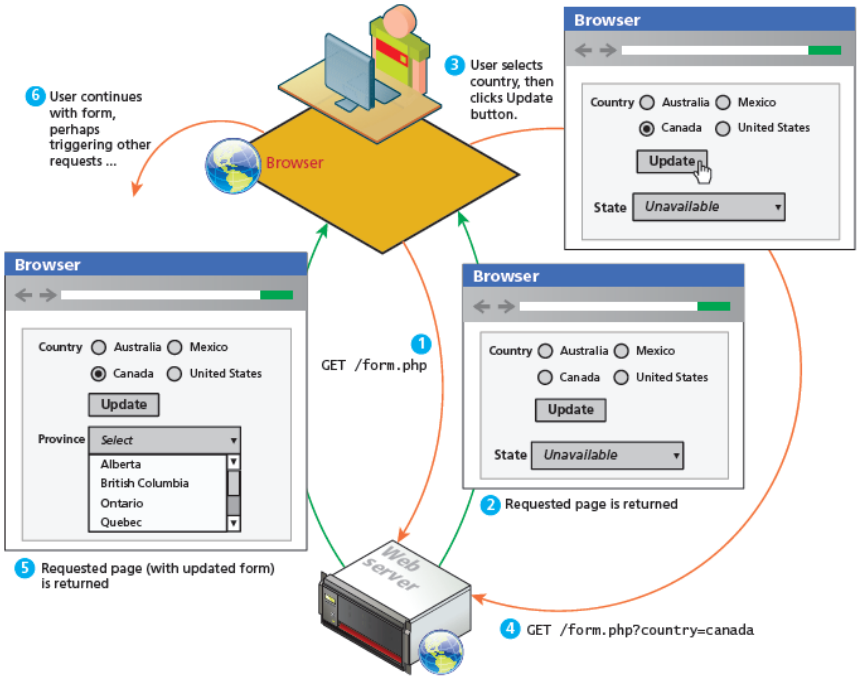
- **Adobe Flash** is a vector based drawing and animation program, a video file format, and a software platform that has its own JavaScript-like programming language called **ActionScript.** Flash is often used for animated advertisements and online games, and can also be used to construct web interfaces.
- The second alternative to JavaScript is **Java applets.** An applet is a term that refers to a small application that performs a relatively small task. Java applets are written using the Java programming language and are separate objects that are included within an HTML document via the <applet> tag. They are downloaded, and then passed on to a Java plug-in. This plug-in then passes on the execution of the applet outside the browser to the Java Runtime Environment (JRE) that is installed on the client's machine.

JavaScript's History and Uses

- JavaScript was introduced by Netscape in their Navigator browser back in 1996. It originally was called LiveScript, because one of its original purposes was to provide control over Java applets. JavaScript is an implementation of a standardized scripting language called ECMAScript.

- In 2000s, JavaScript became a much more important part of web development, with the emergence of AJAX. AJAX is an acronym of Asynchronous JavaScript and XML. AJAX is a technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

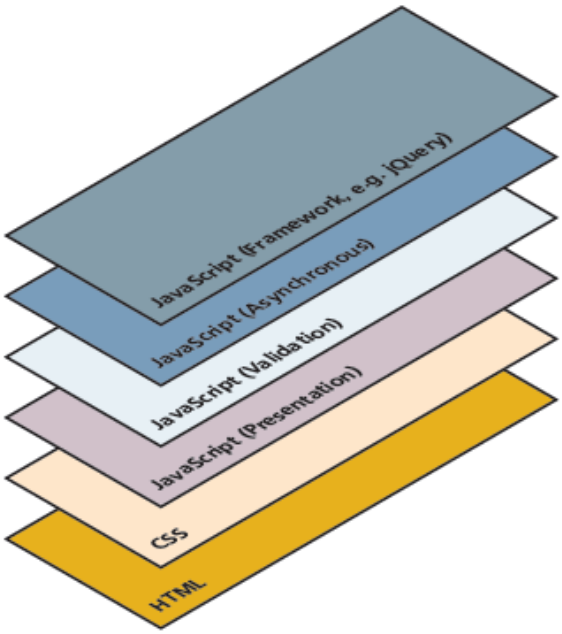
- The most important way that this responsiveness is created is via asynchronous data requests via JavaScript and the XMLHttpRequest object. This object was added to JavaScript by Microsoft as an ActiveX control.



- The above diagram shows the processing flow for a page that requires updates based on user input using the normal synchronous non-AJAX page request-response loop.
- **AJAX** provides web authors with a way to avoid the visual and temporal deficiencies of normal HTTP interactions. With AJAX web pages, it is possible to update sections of a page by making special requests of the server in the background, creating the illusion of continuity.
- The other key developments in the history of JavaScript are jQuery, Prototype, ASP.NET AJAX, and MooTools. These JavaScript frameworks reduce the amount of JavaScript code required to perform typical AJAX tasks.
- Recently introduced MVC JavaScript frameworks such as AngularJS, Backbone, and Knockout have gained a lot of interest from developers wanting to move more data processing and handling from server-side scripts to HTML pages.

3.2 JavaScript Design Principles

JavaScript is conceptualized to be built in different layers having different capabilities and responsibilities. The layer concept is considered optional, except in some special circumstances like online games.



The common layers conceptualized in javascript are -

Presentation Layer

This type of programming focuses on the display of information. JavaScript can alter the HTML elements of a page, which results in a change, visible to the user. These presentation layer applications include common things like creating, hiding, and showing etc. This layer is most closely related to the user experience and the most visible to the end user.

Validation Layer

JavaScript can be also used to validate logical aspects of the user's experience. This includes, validating a form to make sure the email entered is valid before sending it along. It is often used in conjunction with the presentation layer. The intention of this layer is to prevalidate forms before transmitting the input to the server. Both presentation layer and validation layer exist on the client machine.

Asynchronous Layers

Normally, JavaScript operates in a synchronous manner where a request sent to the server requires a response before the next lines of code can be executed. During the wait between request and response the browser is in a loading state and only updates upon receiving the response.

In contrast, an asynchronous layer sends the requests to the server in the **background**. In this model, as certain events are triggered, the JavaScript sends the HTTP requests to the server, but while waiting for the response, the rest of the application works normally, and the browser isn't in a loading state. When the response arrives JavaScript will update a portion of the page. Asynchronous layers are considered advanced versions of the presentation and validation layers.

Users without JavaScript

There are users who are not using JavaScript due to various reasons. This includes some of the most important clients, like web crawler, use of browser plug-in, using a text browser, or are visually impaired.

- **Web crawler** - A web crawler is a client running on behalf of a search engine to download your requested site, so that it can eventually be featured in their search results. These automated software agents do not interpret JavaScript, since it is costly, and the crawler cannot view the enhanced look.

- **Browser plug-in** - A browser plug-in is a piece of software that works within the browser, that interferes with JavaScript.
  There are many uses of JavaScript that are not desirable to the end user. Many malicious sites use JavaScript to compromise a user's computer, and many ad networks deploy advertisements using JavaScript. This motivates some users to install plug-ins that stop JavaScript execution.

- **Text-based client** - Some clients are using a text-based browser. Text-based browsers are widely deployed on web servers, which are often accessed using a command-line interface.

- **Visually disabled client** - A visually disabled client will use special web browsing software to read out the contents of a web page to them. These specialized browsers do not interpret JavaScript. Designing for these users requires some extra considerations.

The <NoScript> Tag

- There are many users who don't use Javascript, there is a simple mechanism to show them special HTML content that will not be seen by those with JavaScript. That mechanism is the HTML tag <noscript>.
- Any text between the <noscript> opening and closing tags will only be displayed to users without the ability to load JavaScript.
- The web site should be created with all the basic functionality enabled using regular HTML, as it doesnot support Javascript.
- This approach of adding functional replacements for web sites without JavaScript is referred as **fail-safe design**. It means that when a plan (such as displaying a fancy JavaScript popup calendar widget) fails, then the system's design will still work.

Graceful Degradation and Progressive Enhancement

The principle of fail-safe design (system works even when there is a failure) can still apply even to browsers that have enabled JavaScript. Some functionalities that works in the current version of Chrome might not work in IE version 8 and that works in a desktop browser might not work in a mobile browser. In such cases, web application developers can take up any of the two policies – graceful degradation or progressive enhancement.

In graceful degradation, a web site is developed for the abilities of current browsers. For those users who are not using current browsers, an **alternate site or pages** are developed.

The alternate strategy is progressive enhancement, which takes the opposite approach to the problem. In this case, the developer creates the site using CSS, JavaScript, and HTML features that are supported by all browsers of a certain age or newer. To that baseline site, the developers can now "progressively" (i.e., for each browser) "enhance" (add functionality) to their site based on the capabilities of the users' browsers.

3.3 Where Does JavaScript Go?

JavaScript and java

- JavaScript and java is only related through syntax.
- JavaScript is object based language and Java is object oriented.
- JavaScript is dynamically typed.
- Java is strongly typed language. Types are all known at compile time and operand types are checked for compatibility. But variables in JavaScript need not be declared and are dynamically typed, making compile time type checking impossible.
- Objects in Java are static -> their collection of data members and methods are fixed at compile time.
- JavaScript objects are dynamic : The number of data members and methods of an object can change during execution

JavaScript can be linked to an HTML page in different ways.

1) Inline JavaScript

Inline JavaScript refers to the practice of including JavaScript code directly within certain HTML attributes. Use of inline javascript is general a bad practice and should be avoided.

Eg:

```
<input type="button" onclick="alert('Are you sure?');" />
```

2) Embedded JavaScript

Embedded JavaScript refers to the practice of placing JavaScript code within a <script> element Use of embedded JavaScript is done for quick testing and for learning scenarios, but is not accepted in real world web pages. Like with inline JavaScript, embedded scripts can be difficult to maintain.

Eg:

```
<script type="text/javascript">
        /* A JavaScript Comment */
        alert ("Hello World!");
</script>
```

3) External JavaScript

Since writing code is a different than designing HTML and CSS, it is often advantageous to separate the two into different files. An external JavaScript file is linked to the html file as shown below.

```
<head>
        <script type="text/JavaScript" src="greeting.js">
        </script>
</head>
```

The link to the external JavaScript file is placed within the <head> element, just as was the case with links to external CSS files. It can also be placed anywhere within the <body> element. It is recommended to be placed either in the <head> element or the very bottom of the <body> element.

JavaScript external files have the extension .js. The file "greeting.js" is linked to the required html file. Here the linked is placed in the <head> tag. These external files typically contain function definitions, data definitions, and other blocks of JavaScript code. Any number of webpages can use the same external file.

3.4 Syntax

Some of the features of javascript are:

- Everything is type sensitive, including function, class, and variable names.
- The scope of variables in blocks is not supported. This means variables declared inside a loop may be accessible outside of the loop.
- There is a === operator, which tests not only for equality but type equivalence.
- Null and undefined are two distinctly different states for a variable.
- Semicolons are not required, but are permitted.
- There is no integer type, only number, which means floating-point rounding errors are prevalent even with values intended to be integers.

Variables

Variables in JavaScript are dynamically typed, it means a variable can be an integer, and then later a string, then later an object. The variable type can be changed dynamically. The variable declarations do not require the type fields like *int*, *char*, and *String*.

The 'var' keyword is used to declare the variable, Eg: **var count;**
As the value is not defined the default value is undefined.

Assignment can happen at declaration-time by appending the value to the declaration, or at run time with a simple right-to-left assignment.
Eg: **var count = 0;**
Var initial=2;
Var b="Abhilash";

In addition, the conditional assignment operator, can also be used to assign based on condition.
Eg: x= (y==1)? "yes": "no";

Variable x is assigned "yes", if value of y is 1, otherwise the value of x is "no".

Comparison Operators

The expressions upon which statement flow control can be based include primitive values, relational expression, and compound expressions. Result of evaluating a control expression is boolean value true or false.

In Javascript the comparison of two values are done by using the following operators-

Operator	Description	Matches (for x=9)
==	Equals	(x==9) is true (x=="9") is true
===	Exactly equals, including type	(x==="9") is false (x===9) is true
< , >	Less than, greater than	(x<5) is false
<= , >=	Less than or equal, greater than or equal	(x<=9) is true
!=	Not equal	(4!=x) is true
!==	Not equal in either value or type	(x!=="9") is true (x!==9) is false

Logical Operators

Many comparisons are combined together, using logical operators. Using logical operators, two or more comparisons are done in a single conditional checking operation. The logical operators used in Javascript are - && (and), || (or), and ! (not).

A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

AND Truth Table

A	B	A || B
T	T	T
T	F	T
F	T	T
F	F	F

OR Truth Table

A	! A
T	F
F	T

NOT Truth Table

Conditionals

JavaScript's syntax is almost identical to that of other programming languages when it comes to conditional structures such as if and if else statements. In this syntax the condition to test is contained within ( ) brackets with the body contained in { } blocks.

Optional else if statements can follow, with an else ending the branch. The below snippet uses a conditional statement to set a greeting variable, depending on the hour of the day

```
var hourOfDay;
var greeting;
if (hourOfDay > 4 && hourOfDay < 12)
{
        greeting = "Good Morning";
}
else if (hourOfDay >= 12 && hourOfDay < 20)
```

```
{
        greeting = "Good Afternoon";
}
else{
        greeting = "Good Evening";
}
```

Loops

Like conditionals, loops use the ( ) and { } blocks to define the condition and the body of the loop respectively.

While Loops

The most basic loop is the while loop, which loops until the condition is not met.

Loops normally initialize a loop control variable before the loop, use it in the condition, and modify it within the loop. One must be sure that the variables that make up the condition are updated inside the loop (or elsewhere) to avoid an infinite loop.

```
while(control expression)
{
        statement or compound statement
}
```

Eg:

```
var i=0;
while(i < 10)
{
        document.write(i);
        i++;
}
```

Do while Loops

It is similar to while loop, but the condition checking is done after the execution of the loop. The loop statements are executed at least ones. The syntax is:

```
        do
        {
                statement or compound statement
        }
        while(control expression);
```

Eg.

```
        var i=0;
        do {
                document.write (i);
           }while (i<10);
```

For Loops

A for loop combines the common components of a loop: initialization, condition, and post-loop operation into one statement. This statement begins with the for keyword and has the components placed between ( ) brackets, semicolon (;) separated. The syntax is as follows:

```
for(initial expression; control expression; increment expression)
{
        statement or compound statement
}
```

```
for (var i = 0; i < 10; i++)
{
        document.write(i);
}
```

Functions

- A function definition consists of the function's header and a compound statement that describes the actions of the function. This compound statement is called the body of the function.
- A function header consists of the **reserved word function**, the function's name, and a parenthesized list of parameters (optional).
- The parentheses are required even if there are no parameters.
- Since JavaScript is dynamically typed, functions do not require a return type, nor do the parameters require type.
- A return statement returns control from the function in which it appears to the function's caller. Optionally, it includes an expression, whose value is returned to the caller. A function body may include one or more return statements. If there are no return statements in a function or if the specific return that is executed does not include an expression, the value returned is undefined. This is also the case if execution reaches the end of the function body without executing a return statement (an action that is valid).
- Syntactically, a call to a function with no parameters states the function's name followed by an empty pair of parentheses. A call to a function that returns undefined is a standalone statement. A call to a function that returns a useful value appears as an operand in an expression

Therefore a function to raise x to the yth power is defined as:

```
function power(x,y)
{
        var pow=1;
        for (var i=0;i<y;i++)
        {
                pow = pow*x;
        }
        return pow;
}
```

It is called as

power(2,10);

Alert

The alert() function makes the browser show a pop-up to the user, with whatever is passed being the message displayed.

Eg:

alert ("Good Morning");

alert("The sum is:" + sum + "\n");

Confirm() method opens a dialog window in which it displays its string parameter, along with two buttons OK and Cancel. Confirm returns a Boolean value that indicates the users button input.

True -> for OK

False-> for cancel.

Eg.

var question = confirm("Do you want to continue this download?");

Prompt() method creates a dialog window that contains a text box which is used to collect a string of input from the user, which prompt returns as its value. The window also includes two buttons, OK and Cancel, prompt takes two parameters the string that prompts the user for input and a default string in case the user does not type a string before pressing one of the two buttons. In many cases an empty string is used for the default input.

Eg.

Var name=prompt("What is your name"."");

Errors Using Try and Catch (Exception Handling)

When the browser's JavaScript engine encounters an error, it will *throw* an exception. These exceptions interrupt the regular, sequential execution of the program and can stop the JavaScript engine altogether. These errors can optionally be caught, preventing disruption of the program using the try–catch block.

```
try
{
        nonexistantfunction("hello");
}
catch(err)
{
        alert("An exception was caught:" + err);
}
```

Throwing Your Own Exceptions

Although try-catch can be used exclusively to catch built-in JavaScript errors, it can also be used by your programs, to throw your own messages. The throw keyword stops normal sequential execution.

Try-catch and throw statements are used for *abnormal* or *exceptional* cases in the program. They should not be used as a normal way of controlling flow. The use of try-catch statements are generally avoided in code unless illustrative of some particular point.

The below example shows the throwing of a user-defined exception as a string.

```
try
{
        var x = -1;
        if (x<0)
        throw "smallerthan0Error";
}
catch(err)
{
        alert (err + "was thrown");
}
```

It should be noted that throwing an exception disrupts the sequential execution of a program. That is, when the exception is thrown all subsequent code is not executed until the catch statement is reached. This reinforces why try-catch is for exceptional cases.

3.5 JavaScript Objects

- JavaScript is not a full-fledged object-oriented programming language. It does not support many of the features of object-oriented language like inheritance and polymorphism. It contains some built-in classes and objects.

- Objects can have **constructors**, **properties**, and **methods** associated with them, and are used very much like objects in other object-oriented languages. There are objects that are included in the JavaScript language; you can also define your own kind of objects.

Constructors
Normally to create a new object we use the new keyword, the class name, and ( ) brackets with *n* optional parameters:
var someObject = new ObjectName(parameter 1,param 2,..., parameter n);

shortcut constructors are defined without using 'new' keyword.
var greeting = "Good Morning";
Instead of the formal definition ----- var greeting = new String("Good Morning");

Properties
Each object might have properties that can be accessed, depending on its definition. When a property exists, it can be accessed using **dot notation** where a dot between the instance name and the property references that property.

alert(someObject.property); *//show someObject.property to the user*

Methods
Objects can also have methods, which are functions associated with an instance of an object. These methods are called using the same dot notation as for properties, but instead of accessing a variable, we are calling a method.
someObject.doSomething();
Methods may produce different output depending on the object they are associated with because they can utilize the internal properties of the object.

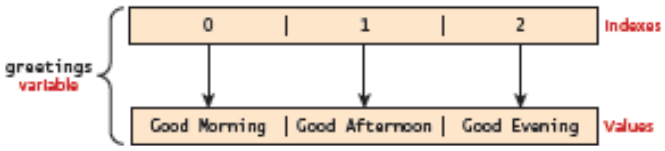
Objects Included in JavaScript
A number of useful objects are included with JavaScript. These include Array, Boolean, Date, Math, String, and others. In addition to these, JavaScript can also access Document Object Model (DOM) objects that correspond to the content of a page's HTML. These DOM objects let JavaScript code access and modify HTML and CSS properties of a page dynamically.

Arrays

Arrays are one of the most used data structures, and they have been included in JavaScript as well.

Objects can be created using the new syntax and calling the object constructor. The following code creates a new, empty array named greetings:
var greetings = new Array();
To initialize the array with values, the variable declaration would look like the following:
var greetings = new Array("Good Morning", "Good Afternoon");

or, using the square bracket notation:
var greetings = ["Good Morning", "Good Afternoon"];



Accessing and Traversing an Array

To access an element in the array use the familiar square bracket notation from Java and C-style languages, with the index you wish to access inside the brackets.

```
alert ( greetings[0] );
```

One of the most common actions on an array is to traverse through the items sequentially. The following for loop quickly loops through an array, accessing the ith element each time using the Array object's **length property** to determine the maximum valid index. It will alert "Good Morning" and "Good Afternoon" to the user.

```
for (var i = 0; i < greetings.length; i++)
{
        alert(greetings[i]);
}
```

Modifying an Array

To add an item to an existing array, you can use the push method.

```
greetings.push("Good Evening");
```

The pop method can be used to remove an item from the back of an array. Additional methods that modify arrays include concat(), slice(), join(), reverse(), shift(), and sort().

Array Methods

Array objects have a collection of useful methods, most of which are described in this section. The **join method** converts all of the elements of an array to strings and concatenates them into a single string. If no parameter is provided to join, the values in the new string are separated by commas. If a string parameter is provided, it is used as the element separator. Consider the following example:

```
var names = new Array["Mary", "Murray", "Murphy", "Max"];
...
var name_string = names.join(" : ");
```

The value of name_string is now "Mary : Murray : Murphy : Max".

The **reverse method** reverses the order of the elements of the Array object through which it is called.

The **sort method** coerces the elements of the array to become strings if they are not already strings and sorts them alphabetically. For example, consider the following statement:

```
names.sort();
```

```
var list = [2, 4, 6, 8, 10];
...
var list2 = list.slice(1, 3);
...
var new_names = names.concat("Moo", "Meow");
```

The value of names is now "Murphy", "Max"];

[“Mary”, “Max”, “Murphy”, “Murray”]

The **concat method** concatenates its actual parameters to the end of the Array object on which it is called. Thus, in the code

The new_names array now has length 6, with the elements of names, along with “Moo” and “Meow”, as its fifth and sixth elements.

The **slice method** does for arrays what the substring method does for strings, returning the part of the Array object specified by its parameters, which are used as subscripts. The array returned has the elements of the Array object through which it is called, from the first parameter up to, but not including, the second parameter. For example, consider the following code:

The value of list2 is now [4, 6]. If slice is given just one parameter, the array that is returned has all of the elements of the object, starting with the specified index. In the code

```
var list = ["Bill", "Will", "Jill", "dill"];
...
var listette = list.slice(2);
```

the value of listette is [“Jill”, “dill”]

Math

The **Math class** allows one to access common mathematic functions and common values quickly in one place. This static class contains methods such as max(), min(), pow(), sqrt(), and exp(), and trigonometric functions such as sin(), cos(), and arctan(). In addition, many mathematical constants are defined such as PI, E (Euler’s number), SQRT2 etc.

Eg: Math.PI *// 3.141592657*
Math.sqrt(4); *// square root of 4 is 2.*
Math.random(); *// random number between 0 and 1*

String

The **String class** is used to create string objects. The string objects are created using ‘new’ keyword or without using new keyword (shortcut constructor)

var greet = new String("Good"); *// long form constructor*
var greet = "Good"; *// shortcut constructor*

A common need is to get the length of a string. This is achieved through the length property
alert (greet.length); *// will display "4"*

Another common way to use strings is to concatenate them together using '+' operator.
var str = greet.concat("Morning"); *// Long form concatenation*
var str = greet + "Morning"; *// + operator concatenation*

Many other useful methods exist within the String class, such as accessing a single character using charAt(), or searching for one using indexOf(). Strings allow splitting a string into an array, searching and matching with split(), search(), and match() methods.

Date
Date allows us to quickly calculate the current date or create date objects for particular dates. To display today's date as a string, simply create a new object and use the toString() method.
var d = new Date();
// This outputs Today is Mon Nov 12 2012 15:40:19 GMT-0700
alert ("Today is "+ d.toString());

Window Object
The window object in JavaScript corresponds to the browser itself. Accessing the current page's URL, the browser's history, and what's being displayed in the status bar, as well as opening new browser windows is possible using this object.

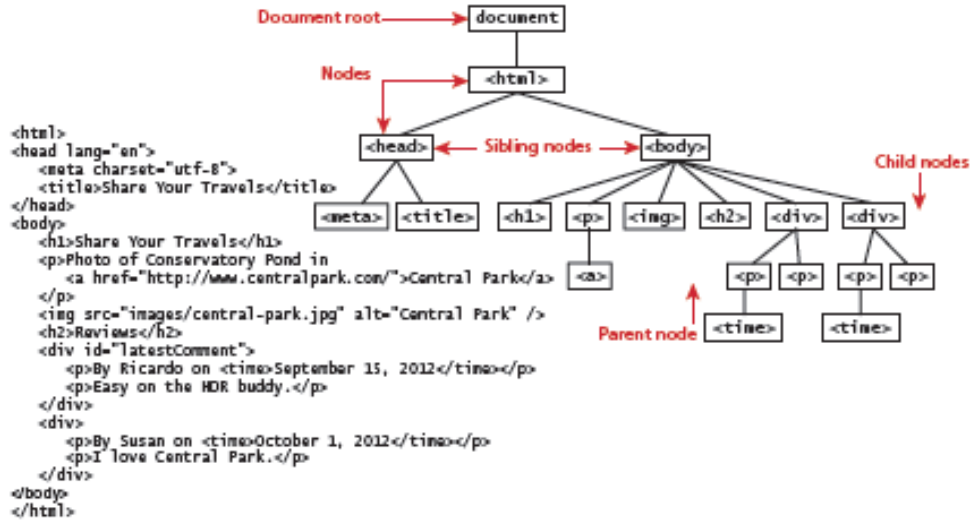
3.6 The Document Object Model (DOM)

JavaScript is used to interact with the HTML document elements in which it is contained. The elements and attributes of HTML can be programmatically accessed, through an API called the **Document Object Model (DOM)**.

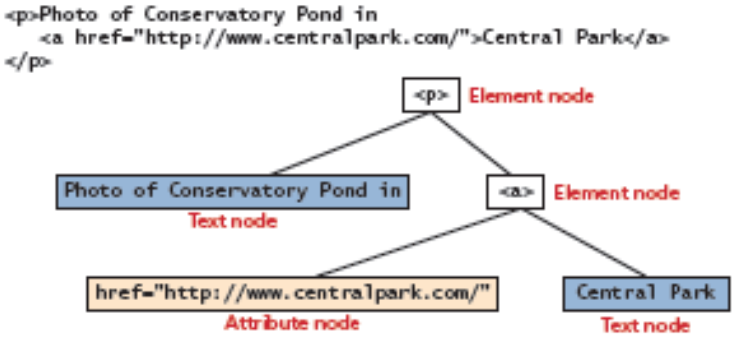
DOM is an API using which the javascript can dynamically access and modify html elements, its attributes and styles associated with it. Javascript can access, modify, add or delete the html elements.

Nodes
The html document with elements is considered as a tree structure called the DOM tree. Here each element of HTML document is called a **node**. Each node is an individual branch, with text nodes, and attribute nodes. Most of the tasks that we typically perform in JavaScript involve finding a node, and then accessing or modifying it via those properties and methods.

The DOM tree



Example of node and its attribute and text nodes:



Properties of a node object

Property	Description
attributes	Collection of node attributes
childNodes	A NodeList of child nodes for this node
firstChild	First child node of this node
lastChild	Last child of this node
nextSibling	Next sibling node for this node
nodeName	Name of the node
nodeType	Type of the node
nodeValue	Value of the node
parentNode	Parent node for this node
previousSibling	Previous sibling node for this node.

Document Object

The **DOM document object** is the root JavaScript object representing the entire HTML document. It contains some properties and methods that are used extensively. Some essential methods like getElementById() , getElementByTagName() are available, the former function returns the object(control) of the single DOM elements whose id is passed as parameter and later function returns a list of elements.

Some essential DOM methods -

Method	Description
createAttribute()	Creates an attribute node
createElement()	Creates an element node
createTextNode()	Creates a text node
getElementById(id)	Returns the element node whose id attribute matches the passed id parameter
getElementsByTagName(name)	Returns a NodeList of elements whose tag name matches the passed name parameter

Element Node Object and its modification

The document.getElementById() method returns an object of **element node**. The method takes the 'ID' of element as parameter,whose control is returned. Since IDs must be unique in an HTML document, getElementByID() returns a single node.

Some properties can be used on these element nodes to modify, create or alter the node properties. They are

Property	Description
id	The current value for the id of this element.
innerHTML	Represents all the things inside of the tags. This can be read or written to and is the primary way in which we update particular , <p> elements using JavaScript.
style	The style attribute of an element. We can read and modify this property.
tagName	The tag name for the element.

Some of the properties applied on particular element nodes are –

Property	Description	Tags
href	The href attribute used in a tag to specify a URL to link to when the context is clicked.	a
name	The name property is a bookmark to identify this tag. Unlike id, which is available to all tags, name is limited to certain form-related tags.	a, input, textarea, form
src	Links to an external URL that should be loaded into the page (as opposed to href, which is a link to follow when clicked)	img, input, iframe, script
value	The value is related to the value attribute of input tags. To retrieve the user input data.	input, textarea, submit

Eg:

Suppose the document contains these elements –

```
<div id="div01">
        <p>By Ricardo on <time>September 15, 2012</time></p>
        <p>Easy on the HDR buddy.</p>
</div>
/******IN JavaScript **********/
var latest = document.getElementById("div01");
var oldMessage = latest.innerHTML;
latest.innerHTML = oldMessage + "<p>Updated this div with JS</p>";
```

/******IN JavaScript **********/

Now the document has been modified to reflect that change.

By Ricardo on <time>September 15, 2012</time>
Easy on the HDR buddy.
Updated this div with JS

2)To get the password out of the following input field and alert the user
<input type='password' name='pw' id='pw' />
We would use the following JavaScript code:
var pass = document.getelementById("pw");
alert (pass.value); // value of the password input tag is displayed.

Changing an Element's Style
The CSS style associated with a particular block of elements can be modified by using 'style' or className property of the Element node,

Eg:

1)To change a node's background color
var x= document.getElementById("specificTag");
x.style.backgroundColour = "#FFFF00";

2) To add a three-pixel border.
var x= document.getElementById("specificTag");
commentTag.style.borderWidth="3px";

3.7 JavaScript Events

A JavaScript event is an event (function) that takes place when an action is detected by JavaScript. The action is done by the user or by the browser. Such as, a button click action of the user triggers its events in the javascript.

Inline Event Handler Approach
JavaScript events allow the programmer to react to user interactions. In early web development, it made sense to weave code and HTML together.
Eg: To pop-up an alert when is clicked:
Click for pop-up

Here the HTML attribute onclick is used to attach a handler to that event. When the user clicks the , the event is triggered and the alert is executed.

Listener Approach

The disadvantage of inline approach is that, it reduces the ability of designers to work separately from programmers and it complicates maintenance of applications.

The HTML element that will trigger the event is accessed to the javascript by using different DOM methods such as getElementById(id), getElementByTagName(tagname) etc. Then the element's event is used to handle the event.

```
var greetingBox = document.getElementById('div01');
greetingBox.onclick = alert('Good Morning');
```

The first line creates a temporary variable for the HTML element that will trigger the event. The next line attaches the element's onclick event to the event handler, which invokes the JavaScript alert() method.

The main advantage of this approach is that this code can be written anywhere, including an external file. However, one limitation is that only one handler can respond to any given element event.

Another approach is to use addEventListener(). This approach has the additional advantage that multiple handlers can be assigned to a single object's event.

```
var greetingBox = document.getElementById('div01');
greetingBox.addEventListener('click', alert('Good Morning')); //without using 'on' for the event.
greetingBox.addEventListener('mouseOut', alert('Goodbye')); //without using 'on' for the event.
```

Functions can be invoked when an event occurs by using the following steps:

```
function displayTheDate()
{
        var d = new Date();
        alert ("You clicked this on "+ d.toString());
}
var element = document.getElementById('div01');
element.onclick = displayTheDate;

// or using the other approach

element.addEventListener('click',displayTheDate);
```

An anonymous function is invoked when the event occurs by the following method –

```
var element = document.getElementById('div01');
element.onclick = function()
{
        var d = new Date();
        alert ("You clicked this on " + d.toString());
};
```

Here, there is no function name mentioned. The keyword 'function' is used and the function is defined directly.

Event Object

The events can be passed to the function handler as a parameter(*e).*

```
function someHandler(e)
{
        // e is the event that triggered this handler.
}
```

Bubbles. The bubbles property is a Boolean value. If an event's bubbles property is set to true then there must be an event handler in place to handle the event or it will bubble up to its parent and trigger an event handler there.

Cancelable. The Cancelable property is also a Boolean value that indicates whether or not the event can be cancelled. If an event is cancelable, then the default action associated with it can be canceled.

preventDefault. A cancelable default action for an event can be stopped using the preventDefault() method.

Event Types

There are several classes of event, with several types of event within each class. The classes are mouse events, keyboard events, form events, and frame events.

Mouse Events

Mouse events are defined to capture a range of interactions driven by the mouse. These can be further categorized as mouse click and mouse move events. The possible mouse events are –

Events	Description
onclick	The mouse was clicked on an element
ondblclick	The mouse was double clicked on an element
onmousedown	The mouse was pressed down over an element
onmouseup	The mouse was released over an element
onmouseover	The mouse was moved (not clicked) over an element
onmouseout	The mouse was moved off of an element
onmousemove	The mouse was moved while over an element

Keyboard Events

Keyboard events occur when taking inputs from the keyboard. These events are most useful within input fields. The possible keyboard events

Events	Description
onkeydown	The user is pressing a key (this happens first)
onkeypress	The user presses a key (this happens after onkeydown)
onkeyup	The user releases a key that was down (this happens last)

Eg: to validate an email address, on pressing any key

```
<input type="text" id="keyExample">
```

The input box above, for example, could be listened to and each key pressed echoed back to the user as an alert.

```
document.getElementById("keyExample").onkeydown = function
myFunction(e){
var keyPressed=e.keyCode; //get the raw key code
var character=String.fromCharCode(keyPressed); //convert to string
alert("Key " + character + " was pressed");
}
```

Form Events

Forms are the main means by which user input is collected and transmitted to the server. The events triggered by forms allow us to do some timely processing in response to user input. The most common JavaScript listener for forms is the onsubmit event. The different form events are -

Events	Description
onblur	A form element has lost focus (that is, control has moved to a different element), perhaps due to a click or Tab key press.
onchange	Some <input>, <textarea>, or <select> field had their value change. This could mean the user typed something, or selected a new choice
onfocus	This is triggered when an element gets focus (the user clicks in the field or tabs to it).
onreset	HTML forms have the ability to be reset. This event is triggered when reset (cleared).
onselect	When the users selects some text. This is often used to try and prevent copy/paste.
onsubmit	When the form is submitted this event is triggered. We can do some prevalidation when the user submits the form in JavaScript before sending the data on to the server

Eg: If the password field (with id pw) is blank, we prevent submitting to the server

```
document.getElementById("loginForm").onsubmit = function(e){
        var pass = document.getElementById("pw").value;
        if(pass=="")
        {
                alert ("enter a password");
                e.preventDefault();
        }
}
```

Frame Events

Frame events are the events related to the browser frame that contains the web page. The most important event is the onload event, which triggers when an object is loaded. The frame events are -

Events	Description
onabort	An object was stopped from loading
onerror	An object or image did not properly load
onload	When a document or object has been loaded
onresize	The document view was resized
onscroll	The document view was scrolled
onunload	The document has unloaded

3.8 Forms

form-related JavaScript concepts, consider the simple HTML form

```
<form action='login.php' method='post' id='loginForm'>
<input type='text' name='username' id='username'/>
<input type='password' name='password' id='password'/>
<input type='submit'></input>
</form>
```

Validating Forms

Form validation is one of the most common applications of JavaScript. Writing code to prevalidate forms on the client side will reduce the number of incorrect submissions to the server, thereby reducing server load.

Although validation must still happen on the server side, JavaScript prevalidation is a best practice. There are a number of common validation activities including email validation, number validation, and data validation.

Empty Field Validation

A common application of a client-side validation is to make sure the user entered something into a field. There's certainly no point sending a request to log in if the username was left blank.

The way to check for an empty field in JavaScript is to compare a value to both null and the empty string ("") to ensure it is not empty.

```
document.getElementById("loginForm").onsubmit = function(e)
{
        var fieldValue=document.getElementByID("username").value;
        if(fieldValue==null || fieldValue== "")
        {
                // the field was empty. Stop form submission
                // Now tell the user something went wrong
                alert("you must enter a username");
        }
}
```

Some additional things to consider are fields like checkboxes, whose value is always set to "on". The code to ensure a checkbox is ticked -

```
var inputField=document.getElementByID("license");
if (inputField.type=="checkbox")
{
        if (inputField.checked)
        //Now we know the box is checked, otherwise it isn't
}
```

Number Validation

Number validation can take many forms. The fields like age, must have only number, this can be checked by using the function like parseInt(), isNAN(), and isFinite() in an validation function.

```
function isNumeric(n) {
        return !isNaN(parseFloat(n)) && isFinite(n);
}
```

Submitting Forms

Submitting a form using JavaScript requires to access the form element into javascript. After accessing the element use the submit() method:

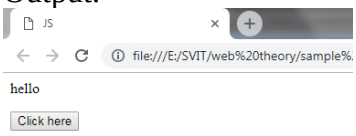
```
var formExample = document.getElementById("loginForm");
formExample.submit();
```

This can be used to submit a form when the user did not click the submit button, or to submit forms with no submit buttons at all in the form (use of an image instead). Also, this can allow JavaScript to do some processing before submitting a form, perhaps updating some values before submitting form.

Write a java script which displays a greeting message when the mouse button is pressed.

```
<html>
<head>
<title>JS</title>
<script type="text/javascript">
        function Fun()
        {
         document.getElementById("p01").innerHTML="hello";
         }
</script>
</head>
<body>
<p id="p01" >qqqqqqqq </p>
<button onclick="Fun()">Click here</button>
</body>
</html>
```

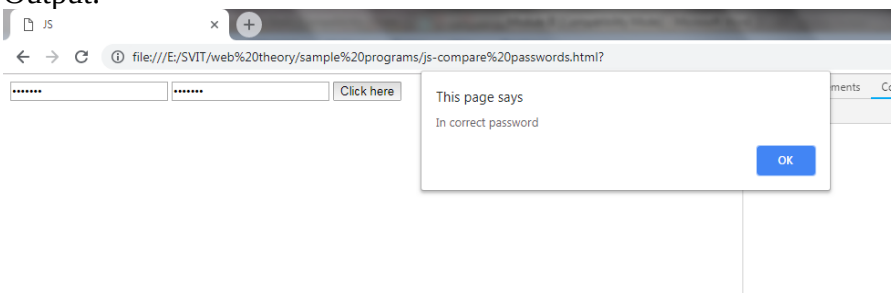
Output:



Create a javascript to compare two passwords.

```
<html>
<head>
<title>JS</title>
<script type="text/javascript">
        function Match()
        {
         var x = document.getElementById("p01").value;
         var y = document.getElementById("p02").value;
         if (x==y)
                alert ("password matches");
         else
                alert ("In correct password ");
        }
</script>
</head>
<body>
<form>
<input type ="password" id="p01"  />
<input type ="password" id="p02"  />
<button onclick="Match()">Click here</button>
</body>
</html>
```

Output:

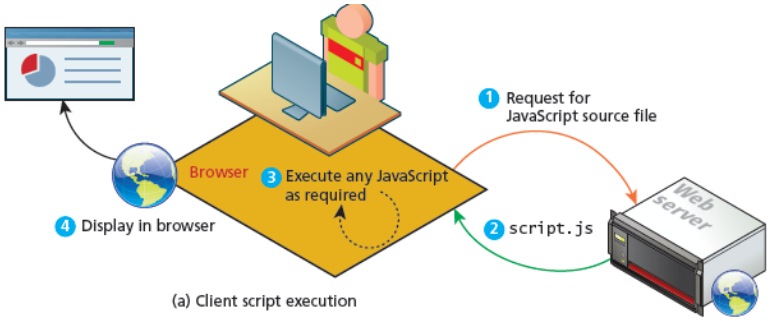


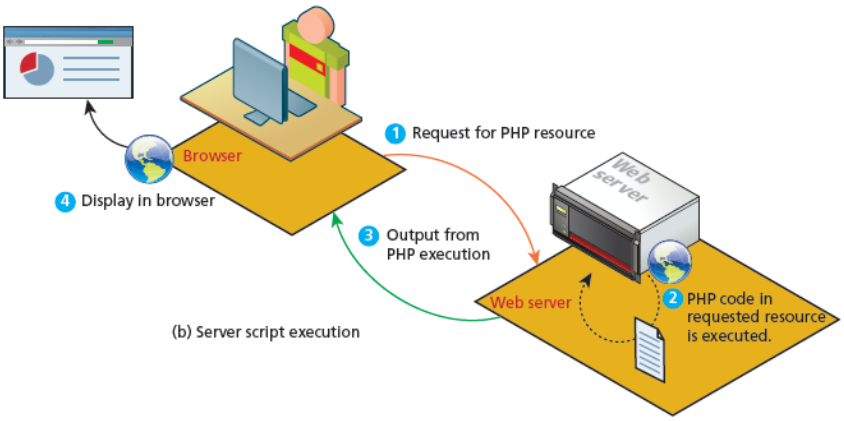
3.9 What Is Server-Side Development?

- The server is a system which is at a remote end and stores resources like files and databases to service the client. It involves the use of a programming technology like PHP or ASP.NET to create scripts that dynamically generate content.
- In server programming the software runs on a web server and uses the HTTP request response loop for most interactions with the clients.

Difference between Client and Server Scripts

Client-side Scripting	Server-side Scripting
The code is executed on the client browser	The code is executed on the web server
The requested script file is downloaded to the client.	The requested script file is hidden from the client as it is processed on the server
The requested file is downloaded at the client and executed.	The requested file is executed at the server and the output is sent to the client.
The client receives the script file.	The client will never see the script file, just the HTML output from the script, is sent to the client.
Client scripts can manipulate the HTML or DOM of a page in the client browser	Server scripts cannot manipulate the HTML or DOM of a page in the client browser
Client script cannot access resources(like database) present on the web server	Server script can access resources on the web server





Server-Side Script Resources

A server-side script can access any resources made available to it by the server. The resources are data storage resources, web services, and software applications. The most commonly used resource is **data storage**, stored in the form of database, and managed by DBMS.

Server-Side Technologies

There are several different server-side technologies for creating web applications. The most common include:

■ **ASP (Active Server Pages)**. This was Microsoft's first server-side technology (also called ASP Classic). Like PHP, ASP code (using the VBScript programming language) can be embedded within the HTML. ASP programming code is interpreted at run time, hence it can be slow in comparison to other technologies.

■ **ASP.NET**. This replaced Microsoft's older ASP technology. ASP.NET is part of Microsoft's .NET Framework and can use any .NET programming language (C# is the most commonly used). ASP.NET uses an explicitly object-oriented approach that typically takes longer to learn than ASP or PHP, and is often used in larger corporate web application systems. A recent extension of ASP.NET is ASP.NET MVC (Model-View-Controller).

■ **JSP (Java Server Pages)**. JSP uses Java as its programming language and like ASP.NET it uses an explicit object-oriented approach and is used in large enterprise web systems and is integrated into the J2EE environment. Since JSP uses the Java Runtime Engine, it also uses a JIT compiler for fast execution time and is cross-platform.

■ **Node.js**. This is a more recent server environment that uses JavaScript on the server side, thus allowing developers already familiar with JavaScript to use just a single language for both client-side and server-side development.

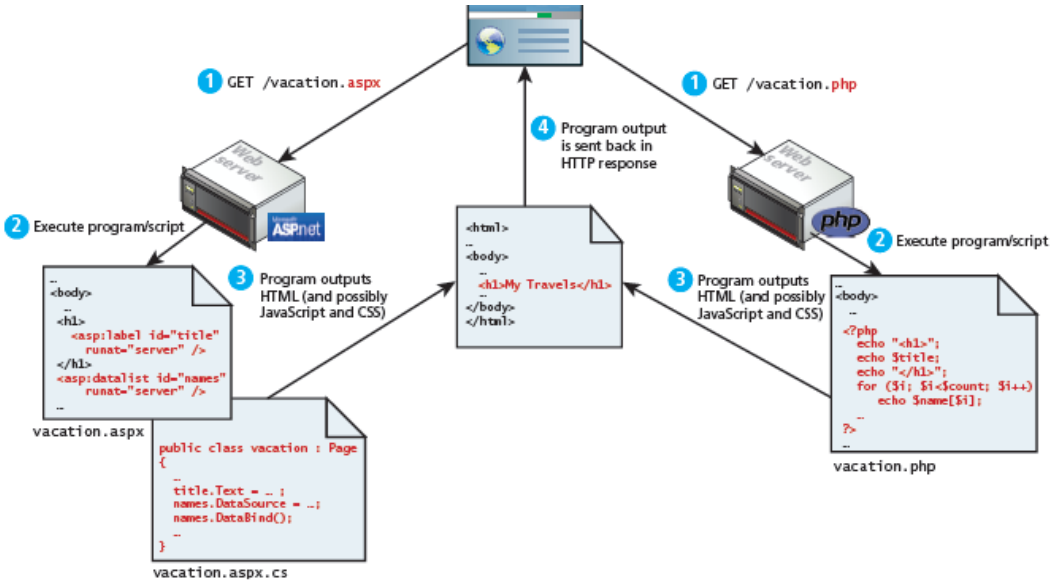
■ **Perl**. Until the development and popularization of ASP, PHP, and JSP, Perl was the language typically used for early server-side web development. It excels in the manipulation of text. It was commonly used in conjunction with the **Common Gateway Interface (CGI)**, an early standard API for communication between applications and web server software.

■ **PHP**. Like ASP, PHP is a dynamically typed language that can be embedded directly within the HTML. It supports most common object oriented features, such as classes and inheritance. By default, PHP pages are compiled into an intermediary representation called **opcodes** that are analogous to Java's byte-code. Originally, PHP stood for *personal home pages*, but now it's acronym is '*Hypertext Processor*'.

■ **Python**. It is an object-oriented programming language. It has many uses, including being used to create web applications.

■ **Ruby on Rails**. This is a web development framework that uses the Ruby programming language. Like ASP.NET and JSP, Ruby on Rails emphasizes the use of common software development approaches.

All of these technologies generate HTML and possibly CSS and JavaScript on the server and send it back to the requesting browser in HTTP response.



3.10 A Web Server's Responsibilities

- Server is responsible for answering all client requests. Even a simplest website must have a web server, to answer requests.

- Once a web server is configured and the IP address associated through a DNS server, it can then start listening for and answering HTTP requests.

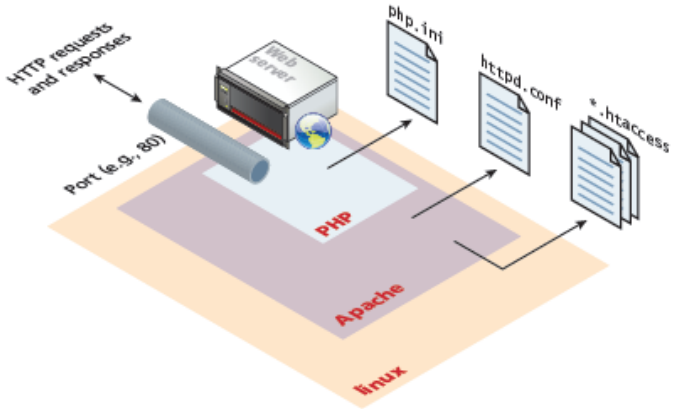
- In the very simplest case the server is hosting static HTML files, and in response to a request sends the content of the file back to the requester.

- A web server has many responsibilities beyond responding to requests for HTML files. These include handling HTTP connections, responding to requests for static and dynamic resources, managing permissions and access for certain resources, encrypting and compressing data, managing multiple domains and URLs, managing database connections, cookies, and state, and uploading and managing files.

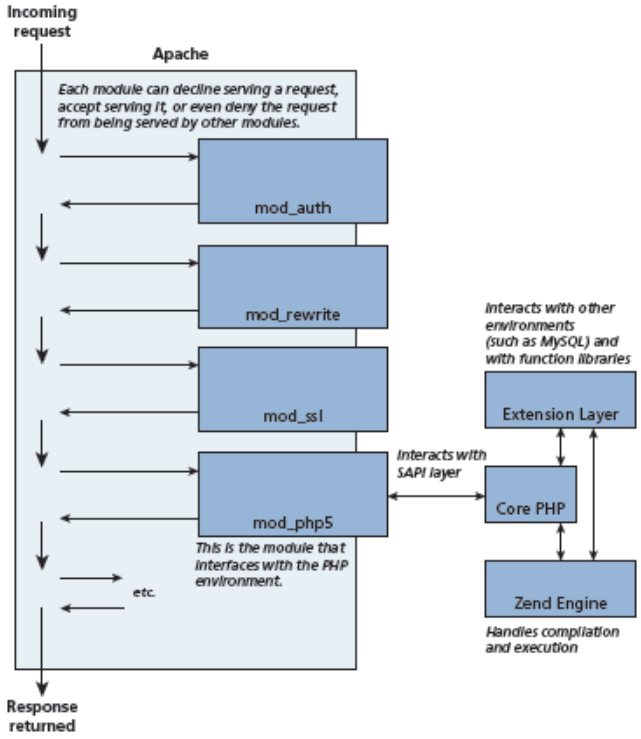
Apache and Linux

The Apache web server can be considered as an intermediary that interprets HTTP requests that arrive through a network port and decides how to handle the request, which often requires working in conjunction with PHP; both Apache and PHP make use of configuration files that determine exactly how requests are handled.

Apache runs as a daemon on the server. A **daemon** is an executing instance of a program (also called a **process**) that runs in the background, waiting for a specific event that will activate it. As a background process, the Apache waits for incoming HTTP requests. When a request arrives, Apache then uses modules to determine how to respond to the request

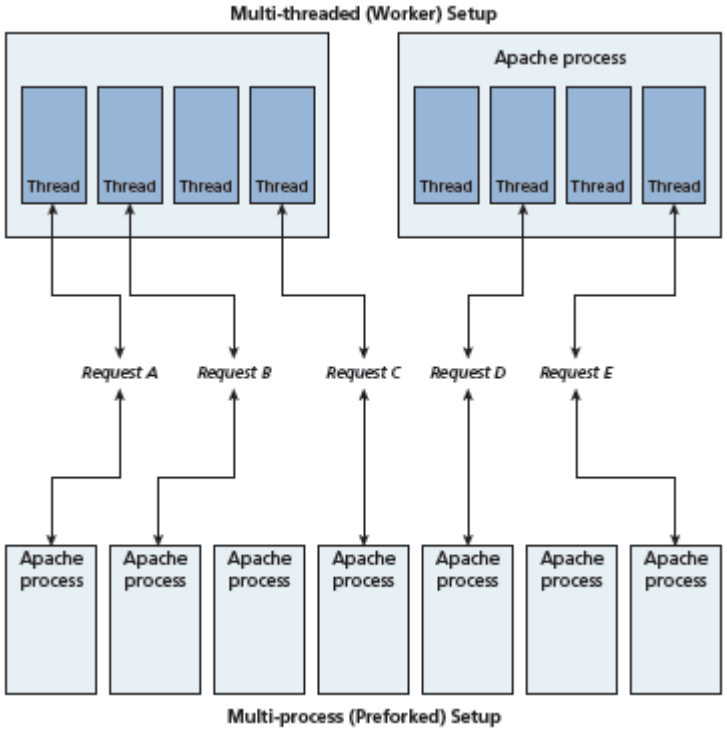


In Apache, a **module** is a compiled extension that helps to *handle* requests. These modules are also sometimes referred to as **handlers**. The below figure, illustrates that when a request comes into Apache, each module is given an opportunity to handle some aspect of the request.
Some modules handle authorization, others handle URL rewriting, while others handle specific extensions.



Apache and PHP
As shown in the above figure, PHP is usually installed as an Apache module. The PHP module mod_php5 is sometimes referred to as the **SAPI** (Server Application Programming Interface) layer since it handles the interaction between the PHP environment and the web server environment.

Apache runs in two possible modes: **multi-process** (also called **preforked**) or **multi-threaded** (also called **worker**).



The default installation of Apache runs using the multi-process mode. That is, each request is handled by a separate process of Apache; the term **fork** refers to the operating system creating a copy of an already running process. Since forking is time intensive, Apache will prefork a set number of additional processes in advance of their being needed. A key advantage of multi-processing mode is that each process is insulated from other processes, that is, problems in one process can't affect other processes

In the multi-threaded mode, a smaller number of Apache processes are forked. Each of the processes runs multiple threads. A **thread** is like a lightweight process that is contained within an operating system process. A thread uses less memory than a process, and typically threads share memory and code; as a consequence, the multi-threaded mode typically scales better to large loads.

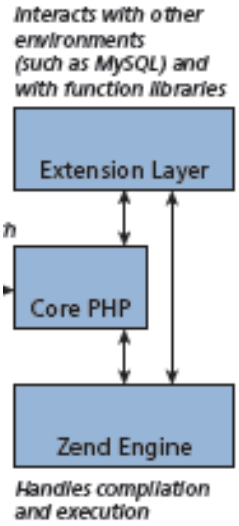
PHP Internals

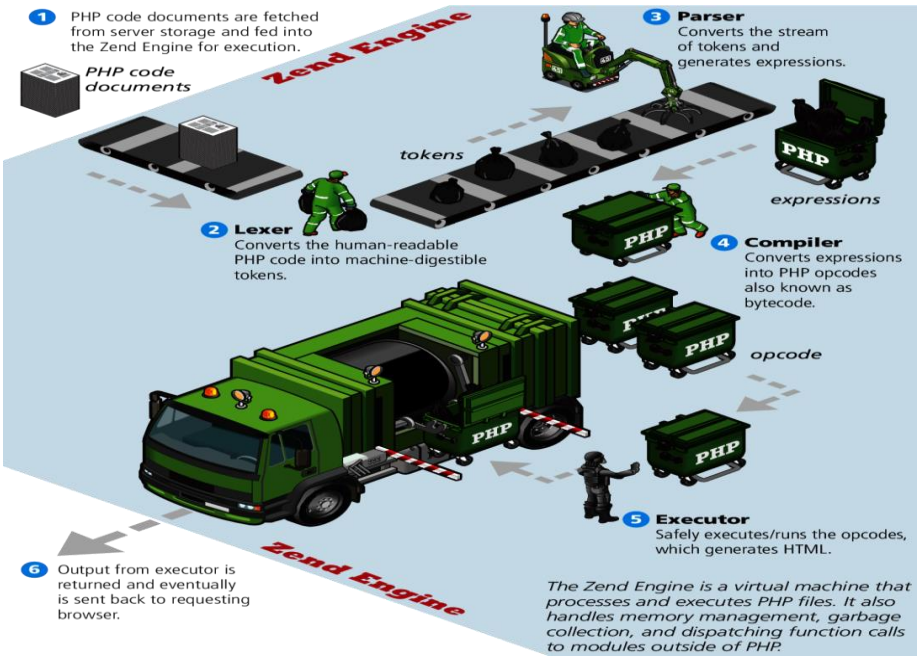
PHP is written in the C programming language and is composed of three main modules:

PHP core. The Core module defines the main features of the PHP environment, including essential functions for variable handling, arrays, strings, classes, math, and other core features.

Extension layer. This module defines functions for interacting with services outside of PHP. This includes libraries for MySQL, FTP, SOAP web services, and XML processing, among others.

Zend Engine. This module handles the reading in of a requested PHP file, compiling it, and executing it.





3.11 Quick tour of PHP

- PHP is a dynamically typed language, like JavaScript.
- It uses classes and functions in a way consistent with other object-oriented languages such as C++, C#, and Java, though with some minor exceptions.
- The syntax for loops, conditionals, and assignment is identical to JavaScript.
- All keywords, class names, function names(both built-in and user defined) are not case sensitive.
- Variables are case-sensitive.

PHP Tags

The PHP programming code can be embedded directly within an HTML file. But, PHP file will have the extension **.php**. The PHP programming code must be contained within an opening "<?php" tag and a matching closing "?>" tag in order to differentiate it from the HTML. The

programming code within the <?php and the ?> tags is interpreted and executed, while any code outside the tags is echoed directly out to the client (browser).
Eg:

```
<?php
        $user = "Randy";
?>
<!DOCTYPE html>
<html>
<body>
<h1><?php echo “Welcome $user”; ?></h1>
</body>
</html>
```

The HTML output from the PHP script sent to the browser is –

```
<!DOCTYPE html>
<html>
<body>
<h1>Welcome Randy</h1>
</body>
</html>
```

The combining of PHP file and HTML in the same file makes the file complex, doing so will make your PHP pages very difficult to understand and modify. Indeed, the authors have seen PHP files that are several thousands of lines long, which are quite a nightmare to maintain.

An alternative way is to keep much of the php script in a separate file with extension .php, and include it in the required file.

PHP Comments

Programmers are supposed to write documentation to provide other developers guidance on certain parts of a program. In PHP any writing that is a comment is ignored when the script is interpreted, but visible to developers who need to write and maintain the software. The types of comment styles in PHP are:

- **Single-line comments**. Lines that begin with a # are comment lines and will not be executed.
- **Multiline (block) comments**. The multiline comments begin with a /* and encompass everything that is encountered until a closing */ tag. These tags cannot be nested.
- **End-of-line comments**. Comments need not always be large blocks of natural language. Whenever // is encountered in code, everything up to the end of the line is considered a comment.

Variables, Data Types, and Constants

Variables in PHP are **dynamically typed**, which means that the data type of a variable is not declared. Instead the PHP engine makes a best guess as to the intended type based on what it is being assigned. To declare a variable you must preface the variable name with the dollar ($) symbol. Whenever you use(access) that variable, you must also include the $ symbol with it. You can assign a value to a variable as in JavaScript’s right-to-left assignment, so creating a variable named count and assigning it the value of 42 would be done with:

$count = 42;

Note:

- In PHP the name of a variable is case-sensitive, so $count and $Count are references to two different variables.
- In PHP, variable names can also contain the underscore character.
- The variable name can begin with an alphabet or underscore.
- The PHP engine determines the data type when the variable is assigned a value.

A **constant** is somewhat similar to a variable, except a constant's value never changes. A constant can be defined anywhere but is typically defined near the top of a PHP file via the define() function. The define() function generally takes two parameters: the name of the constant and its value.

```
<?php
        define("PI",3.142);
?>
```

Writing to Output

To output something that will be seen by the browser, you can use the echo() function.

```
echo ("hello");
```

There is also an equivalent **shortcut** version that does not require the parentheses.

```
echo "hello";
```

Strings can easily be appended together using the concatenate operator, which is the period (.) symbol. Consider the following code:

```
$username = "Ricardo";
echo "Hello". $username;
```

This code will output Hello Ricardo to the browser.

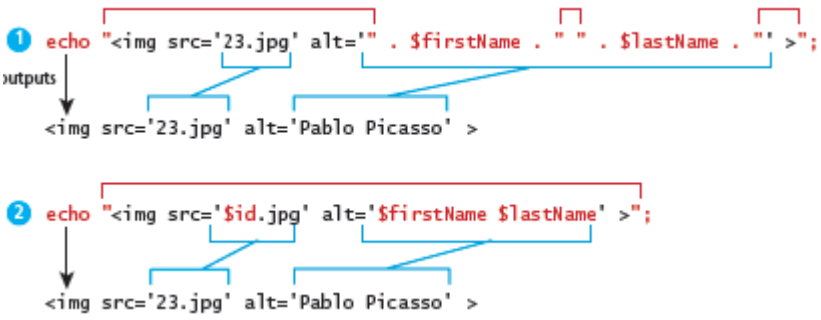
The variable references can appear within string literals (but only if the literal is defined using double quotes).

```
$firstName="Abhilash";
$lastName='NaiK';
echo "<b>" . $firstName . " ". $lastName. </b>";
echo "<b> $firstName $lastName </b>";
Echo '<b>' . $firstName . '  '. $lastName. '</b>';
```

Concatenation is an important part of almost any PHP program. Some of the sample concatenation statements are –

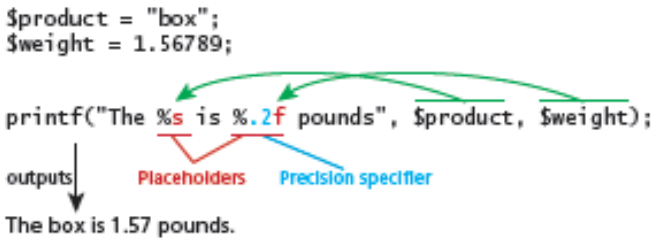
```
<?php
$id = 23;
$firstName = "Pablo";
$lastName = "Picasso";
```

```
echo "<img src='23.jpg' alt='". $firstName . " ". $lastName . "' >";
echo "<img src='$id.jpg' alt='$firstName $lastName' >";
?>
```



printf

The printf() function takes at least one parameter, which is a string, and that string optionally references parameters, which are then integrated into the first string by **placeholder substitution**. The printf() function also allows a developer to apply special formatting, for instance, number of decimal places.



Each placeholder requires the percent (%) symbol in the first parameter string followed by a type specifier.

Common type specifiers are b for binary, d for signed integer, f for float, o for octal, and x for hexadecimal. Precision is achieved in the string with a period (.) followed by a number specifying how many digits should be displayed for floating-point numbers.

3.12 Program Control

Just as with most other programming languages there are a number of conditional and iteration constructs in PHP. There are if and switch, and while, do while, for, and foreach loops.

if . . . else

In this syntax the condition to test is contained within () brackets with the body contained in {} blocks. Optional else if statements can follow, with an else ending the branch. The below code use a condition to set a greeting variable, depending on the hour of the day.

```
if ( $hourOfDay > 6 && $hourOfDay < 12 ) {
        $greeting = "Good Morning";
}
else if ($hourOfDay == 12) {
                $greeting = "Good Noon Time";
}
```

```
else {
                $greeting = "Good Afternoon or Evening";
}
```

It is also possible to place the body of an if or an else outside of PHP. This approach will sometimes be used when the body of a conditional contains nothing but markup with no logic, though because it mixes markup and logic.

```
<?php if ($userStatus == "loggedin") { ?>
        <a href="account.php">Account</a>
<?php } else { ?>
        <a href="register.php">Register</a>
<?php } ?>
```

switch . . . case

The switch statement is similar to a series of if . . . else statements.

```
switch ($artType)
{
        case "PT":
        $output = "Painting";
        break;
        case "SC":
        $output = "Sculpture";
        break;
        default:
        $output = "Other";
}
```

while and do . . . while

The while loop and the do . . . while loop are quite similar. Both will execute nested statements repeatedly as long as the while expression evaluates to true. In the while loop, the condition is tested at the beginning of the loop; in the do ... while loop the condition is tested at the end of each iteration of the loop.

```
Eg : //while loop
$count = 0;
while ($count < 10)
{
        echo $count;
        $count++;
}

//do-while
$count = 0;
do
{
```

```
        echo $count;
        $count++;
} while ($count < 10);
```

for

The for loop in PHP has the same syntax as the for loop in JavaScript. The for loop contains the same loop initialization, condition, and post-loop operations as in JavaScript.

```
for ($count=0; $count < 10; $count++)
{
        echo $count.'<br />';
}
```

The foreach loop is used for iterating through arrays or list of elements.

Alternate Syntax for Control Structures

PHP has an alternative syntax for most of its control structures (namely, the if, while, for, foreach, and switch statements). In this alternate syntax, the colon (:) replaces the opening curly bracket, while the closing brace is replaced with endif;, endwhile;, endfor;, endforeach;, or endswitch;. It improves the readability of PHP code.

```
<?php if ($userStatus == "loggedin") : ?>
<a href="account.php">Account</a>
<?php else : ?>
<a href="login.php">Login</a>
<a href="register.php">Register</a>
<?php endif; ?>
```

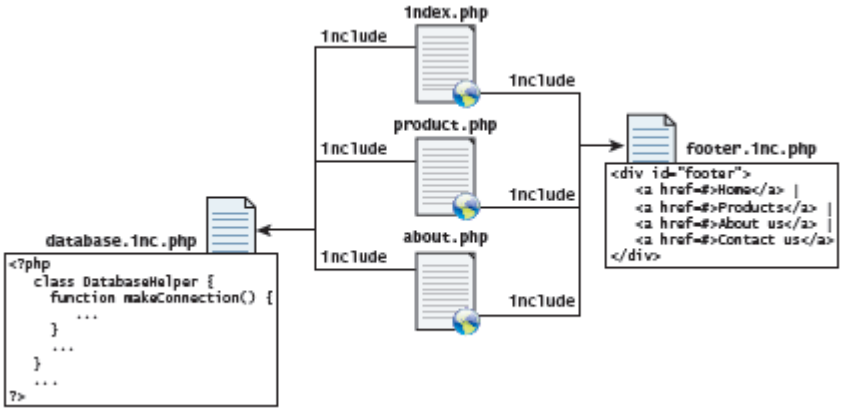
Include Files

PHP have facility to include or insert content from one file into another. Include files provide a mechanism for reusing both markup and PHP code.

PHP provides four different statements for including files, as shown below.

```
include "somefile.php";
include_once "somefile.php";
require "somefile.php";
require_once "somefile.php";
```

The difference between include and require lies in what happens when the specified file cannot be included. With include, a warning is displayed and then execution continues. With require, an error is displayed and **execution stops**. The include_once and require_once statements work just like include and require but if the requested file has already been included once, then it will not be included again.



3.13 Functions

A **function** in PHP contains a small bit of code that accomplishes a task. These functions can be made to behave differently based on the values of their parameters. It exist all on its own, and is called from anywhere that needs to make use of them.

In PHP there are two types of function: user-defined functions and built-in functions.
A **user-defined function** is one that the programmer define. A **built-in function** is one of the functions that come with the PHP environment. One of the real strengths of PHP is its rich library of built-in functions.

Function Syntax

The PHP function uses 'function' keyword followed by the function's name, round ( ) brackets for parameters, and then the body of the function inside curly { } brackets. Functions can return values to the caller, or not return a value.

Syntax:

```
function functionname(parameter list)
{
 }
```

Eg:

```
function getSum()
{
        $x=5;$y=6;
        $z= $x+$y;
        echo $z;
}
```

A function called getNiceTime(), which will return a formatted string containing the current server time

```
function getNiceTime() {
        return date("H:i:s");
}
```

Calling a Function

To call a function you must use its name with the () brackets. Since getNiceTime() returns a string, you can assign that return value to a variable, or echo that return value directly, as shown below.

```
$output = getNiceTime();
OR
echo getNiceTime();
```

If the function doesn't return a value, you can just call the function:

```
getSum();
```

```
<?php
function getSum($x,$y)
{
        $z= $x+$y;
        return $z;   // return $x+$y;
}
$ sum =getSum(5,6);
Echo $sum;
?>
```

Parameters

Parameters are the mechanism by which values are passed into functions, and there are some complexities that allow us to have multiple parameters, default values, and to pass objects by reference instead of value. Parameters, being a type of variable, must be prefaced with a $ symbol like any other PHP variable

```
<?php
function getSum($x,$y)
{
        $z= $x+$y;
        echo $z;
}
getSum(5,6);
?>
```

Parameter Default Values

In PHP, **parameter default values** can be set for any parameter in a function. However, once you start having default values, all subsequent parameters must also have defaults.

```
<?php
function getSum($x,$y=10)
{
        $z= $x+$y;
        return $z;   // return $x+$y;
}
Echo "sum of 5 and6 is ".getSum(5,6);
```

```
Echo “<br />sum of 15 and 10 is “.getSum(15);
Echo “<br />sum of 12 and 14 is “.getSum(12,14);
Echo “<br />sum of 25 and 10 is “.getSum(25);
?>
```

Output:
sum of 5 and6 is 11
sum of 15 and 10 is 25
sum of 12 and 14 is 26
sum of 25 and 10 is 35

Even if 2nd parameter value is not passed to the function, the default value 10 is taken. If the value is passed, the default will be overridden by whatever that value was.

Passing Parameters by Reference

By default, arguments passed to functions are **passed by value** in PHP. This means that PHP passes a copy of the variable so if the parameter is modified within the function, it does not change the original.

In the below example, notice that even though the function modifies the parameter value, the contents of the variable passed to the function remain unchanged after the function has been called.

```
function changeParameter($arg) {
$arg += 300;
echo "arg=" . $arg;
}
$initial = 15;
echo "<br/>initial=" . $initial;
changeParameter($initial);
echo "<br/>initial=" . $initial; // value not changed
```

Output:
initial=15
arg =315
initial=15

PHP allows arguments to functions to be **passed by reference**, which will allow a function to change the contents of a passed variable. A parameter passed by reference points the local variable to the same place as the original, so if the function changes it, the original variable is changed as well. The mechanism in PHP to specify that a parameter is passed by reference is to add an ampersand (&) symbol next to the parameter name in the function definition.

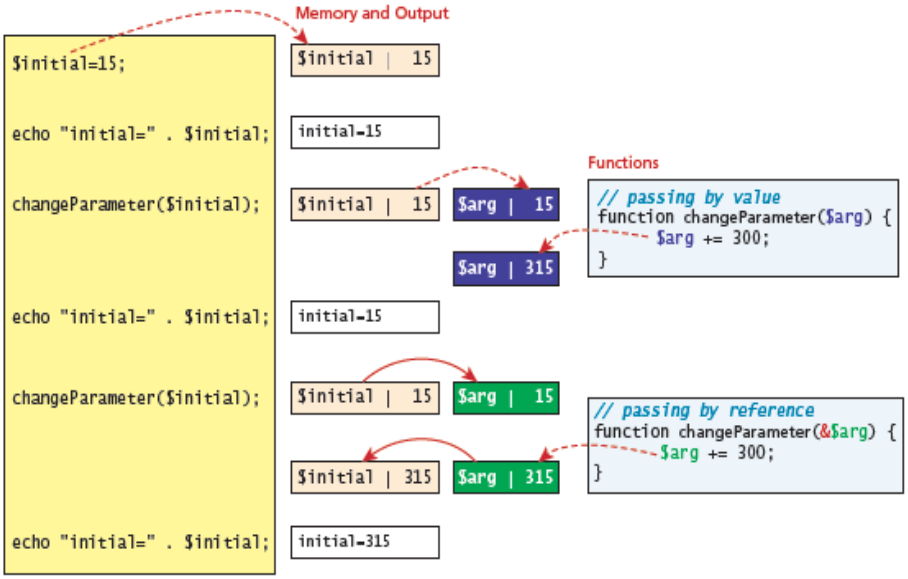
Eg:

```
function changeParameter(&$arg) {
$arg += 300;
echo "arg=" . $arg;
}
$initial = 15;
echo "<br/>initial=" . $initial;
```

changeParameter($initial);
echo "
initial=" . $initial; // value changed

Output:
initial=15
arg =315
initial=315

The memory differences between pass-by value and pass-by reference is illustrated below.



Variable Scope within Functions
All variables defined within a function have **function scope**, ie. they are only accessible within the function.
Unlike in other language, in PHP any variables created outside of the function in the main script are unavailable within a function. For instance, in the following example, the output of the echo within the function is 0 and not 56 since the reference to $count within the function is assumed to be a new
variable named $count with function scope.

$count= 56;
function testScope() {
    echo $count; *// outputs 0 or generates run-time warning/error*
}
testScope();
echo $count; *// outputs 56*

The program results in run – time error

PHP allow variables with global scope to be accessed within a function using the global keyword.

```
<?php
$count= 56;
function testScope() {
        global $count;
        echo $count;
}
testScope();
echo $count;
?>
```

Output:
56
56