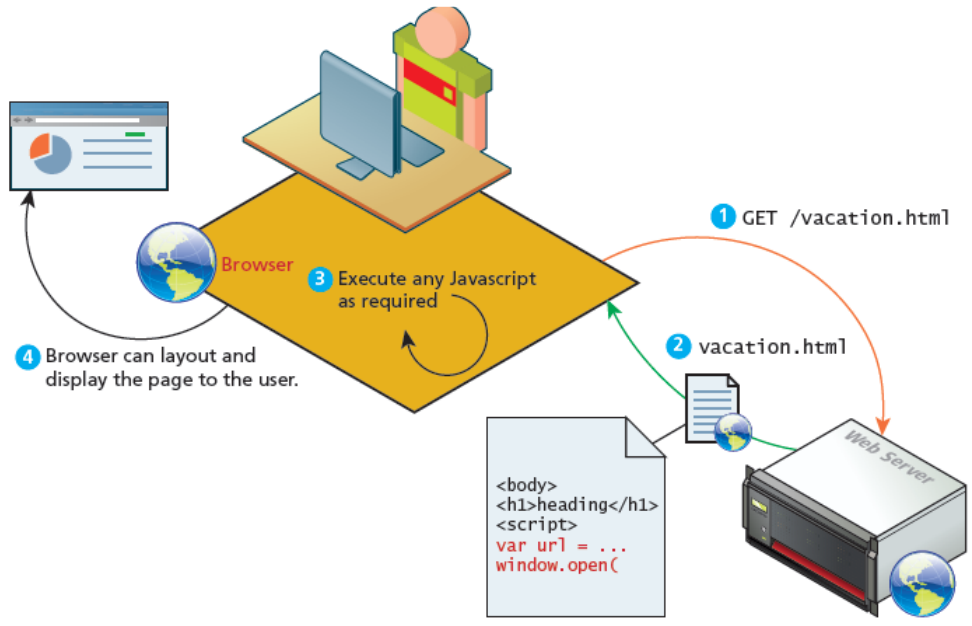
Module III
JavaScript & PHP

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.**

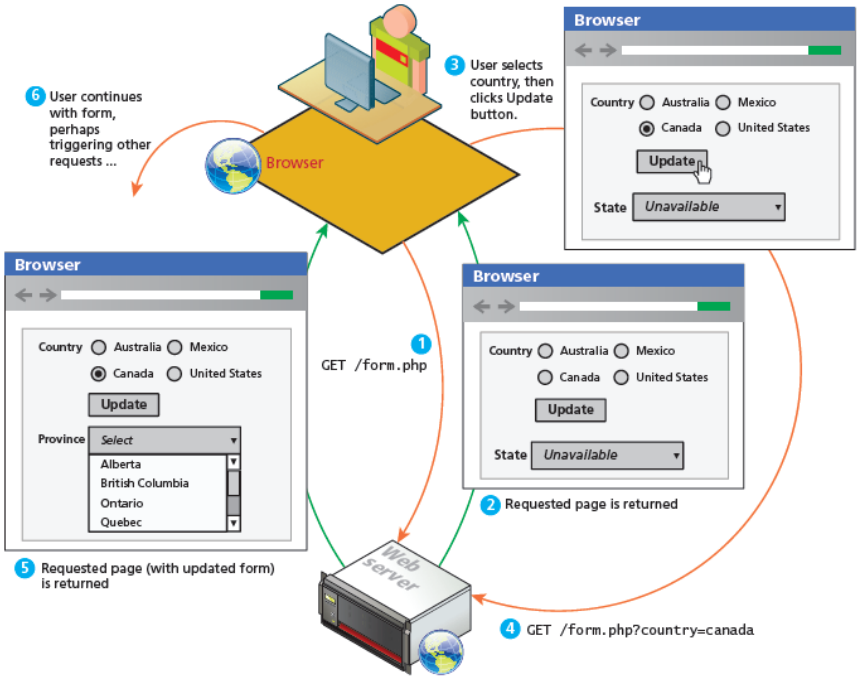
- **Adobe Flash** is a vector based drawing and animation program, a video file format, and a software platform that has its own JavaScript-like programming language called **ActionScript.** Flash is often used for animated advertisements and online games, and can also be used to construct web interfaces.
- 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.

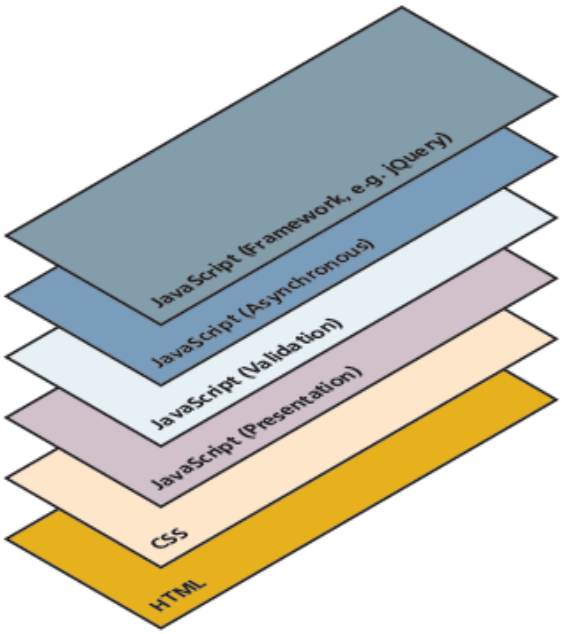
- The most important way that this responsiveness is created is via asynchronous data requests via JavaScript and the XMLHttpRequest object. This object was added to JavaScript by Microsoft as an ActiveX control.



- 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.

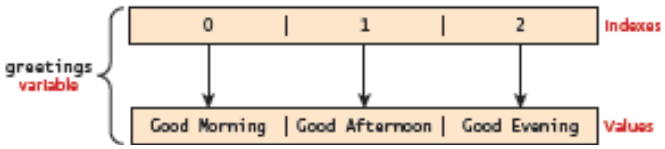
Objects Included in JavaScript
A number of useful objects are included with JavaScript. These include Array, Boolean, Date, Math, String, and others. In addition to these, JavaScript can also access Document Object Model (DOM) objects that correspond to the content of a page's HTML. These DOM objects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

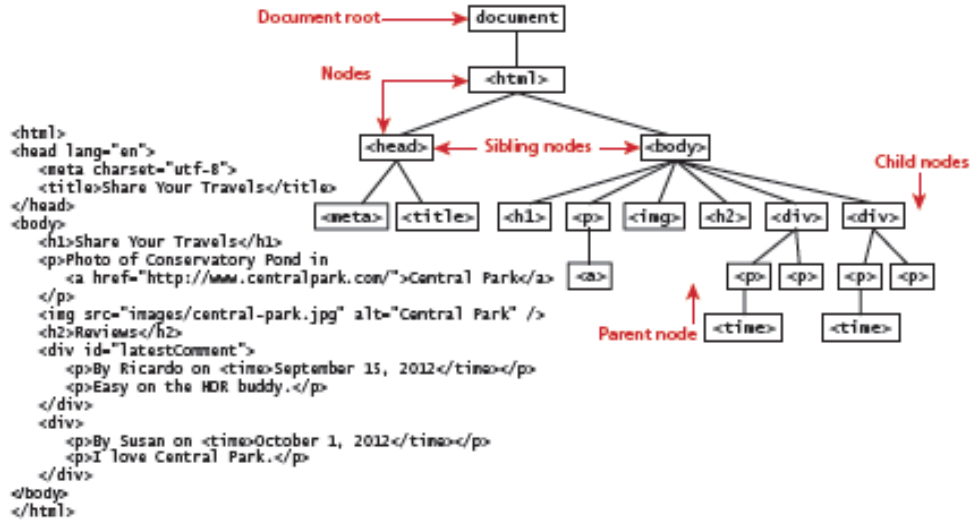
3.6 The Document Object Model (DOM)

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)**.

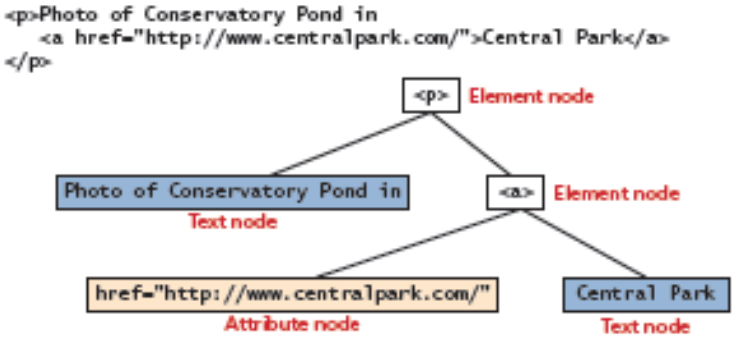
DOM is an API using which the javascript can dynamically access and modify html elements, its attributes and styles associated with it. Javascript can access, modify, add or delete the html elements.

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.

```
<div id="div01">
<p>By Ricardo on <time>September 15, 2012</time></p>
<p>Easy on the HDR buddy.</p>
<p>Updated this div with JS</p>
</div>
```

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:

```
<div id="example1" onclick="alert('hello')">Click for pop-up</div>
```

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:

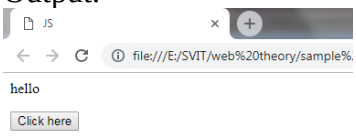
```
var formExample = document.getElementById("loginForm");
formExample.submit();
```

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>
```

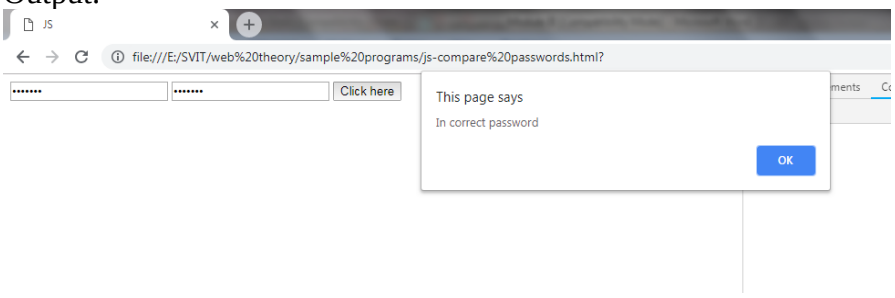
Output:



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:

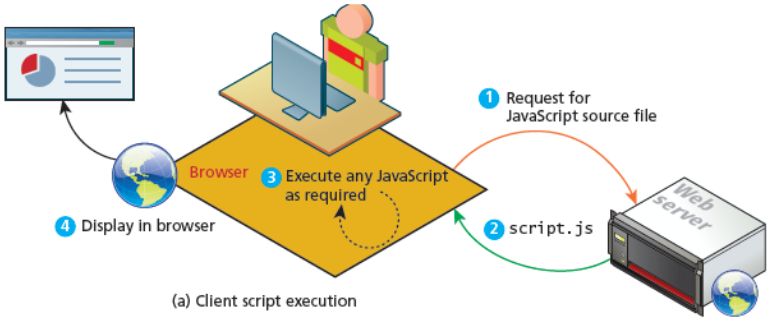


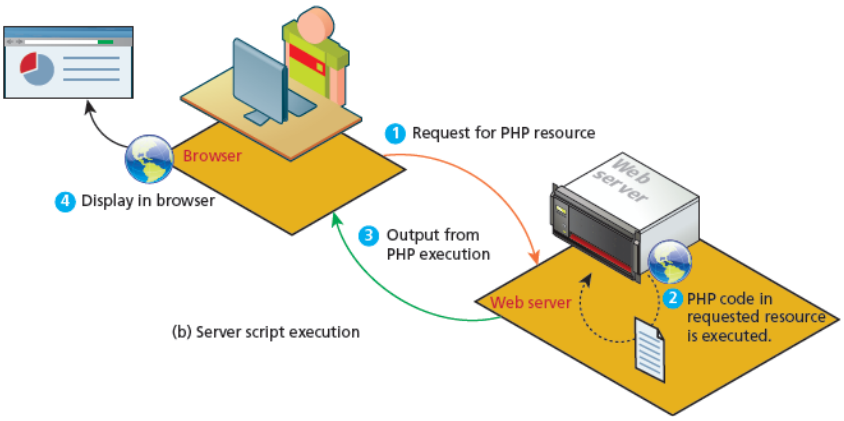
3.9 What Is Server-Side Development?

- 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.

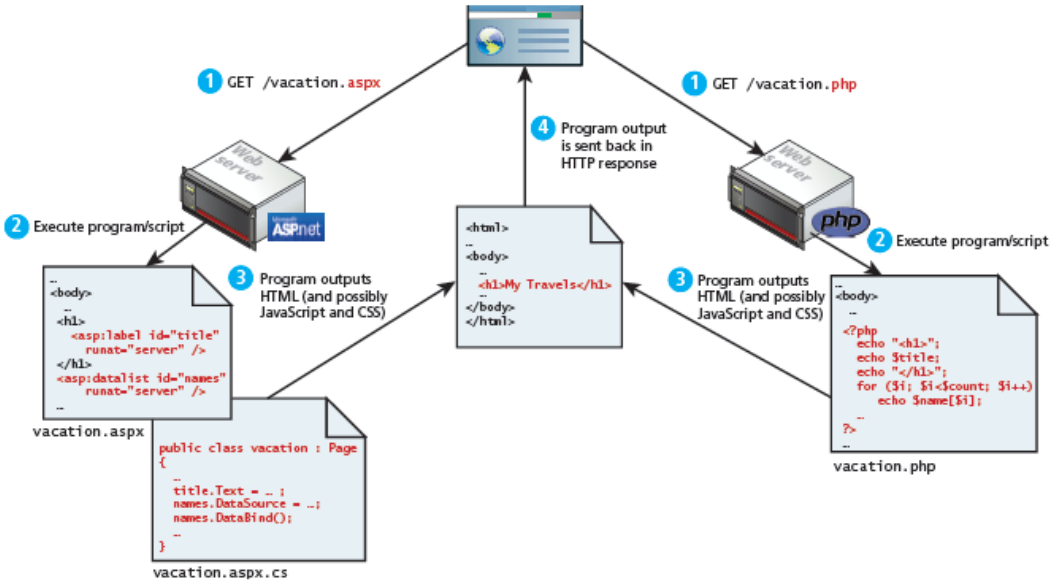
■ **Perl**. Until the development and popularization of ASP, PHP, and JSP, Perl was the language typically used for early server-side web development. It excels in the manipulation of text. It was commonly used in conjunction with the **Common Gateway Interface (CGI)**, an early standard API for communication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

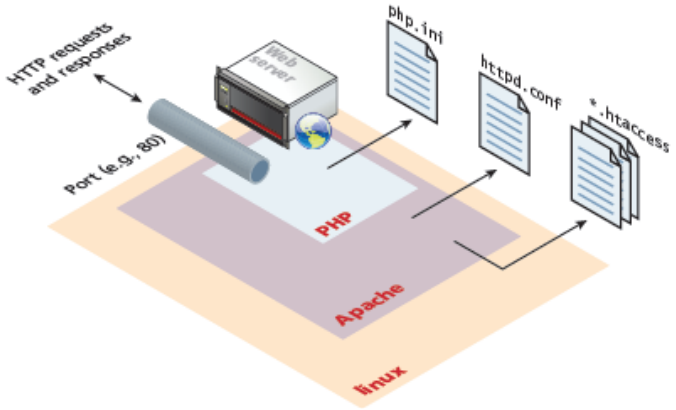
- In the very simplest case the server is hosting static HTML files, and in response to a request sends the content of the file back to the requester.

- 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.

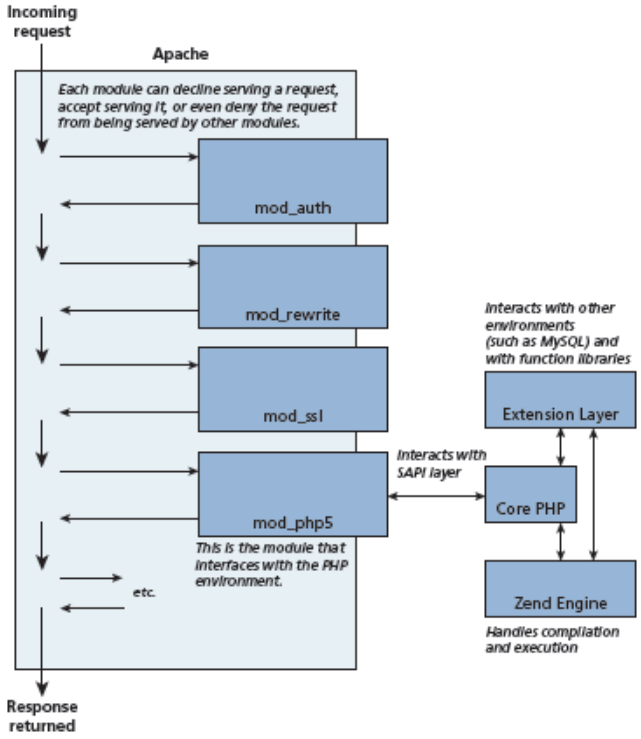
Apache and Linux

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

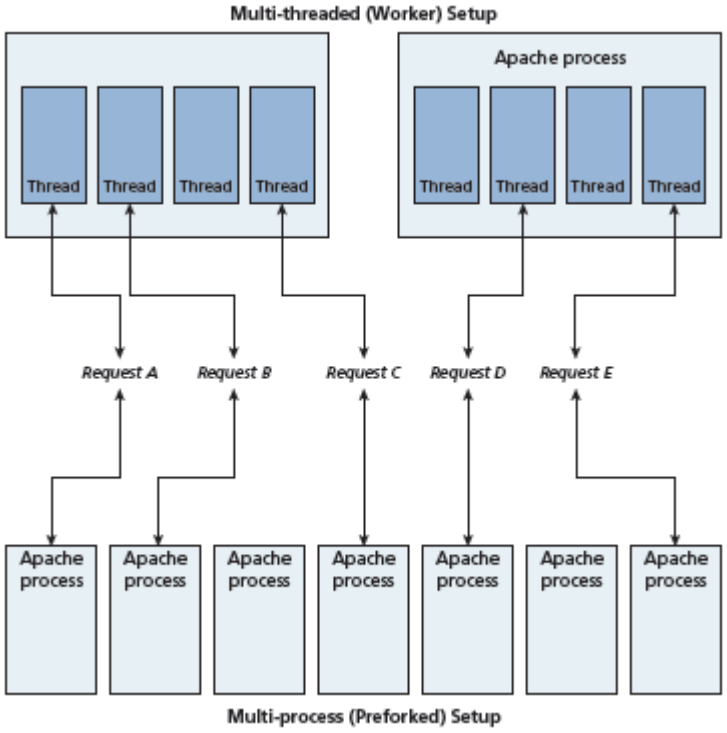


In Apache, a **module** is a compiled extension that helps to *handle* requests. These modules are also sometimes referred to as **handlers**. The below figure, illustrates that when a request comes into Apache, each module is given an opportunity to handle some aspect of the request.
Some modules handle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

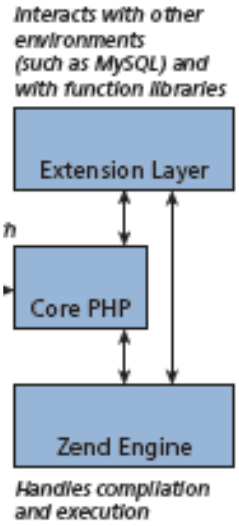
PHP Internals

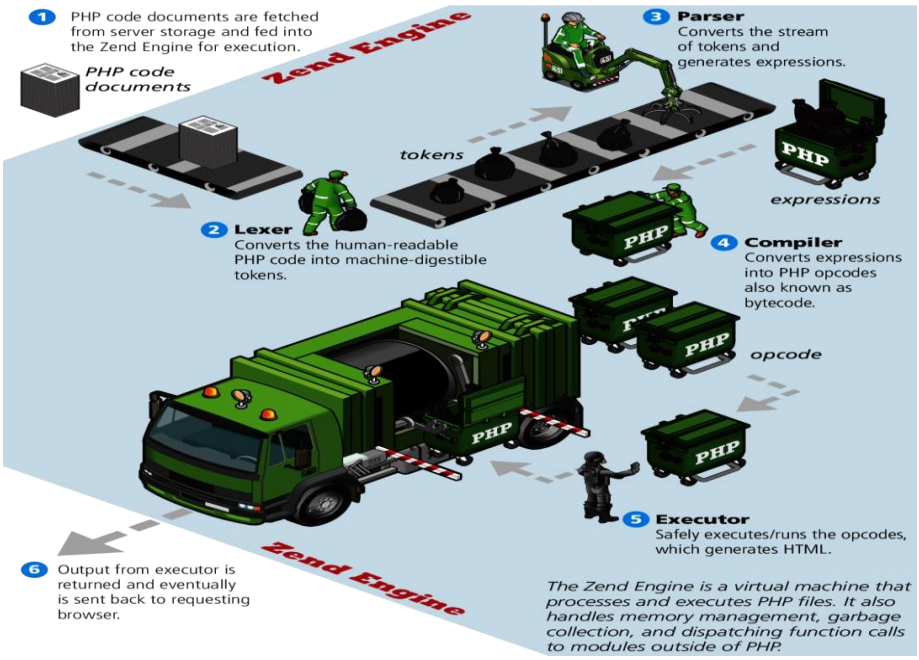
PHP is written in the C programming language and is composed of three main modules:

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.

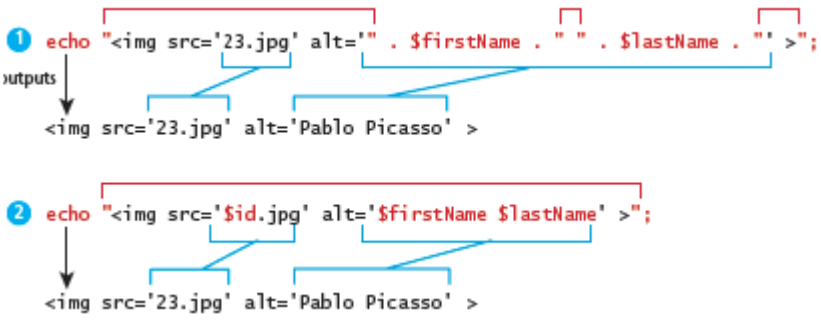
The variable references can appear within string literals (but only if the literal is defined using double quotes).

```
$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 –

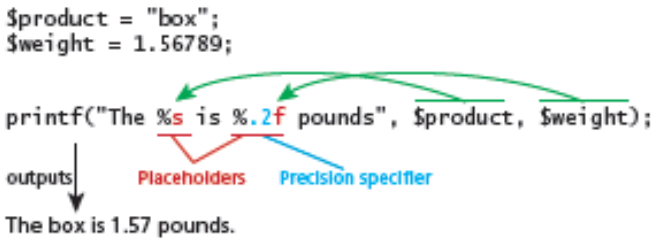
```
<?php
$id = 23;
$firstName = "Pablo";
$lastName = "Picasso";
```

```
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; ?>
```

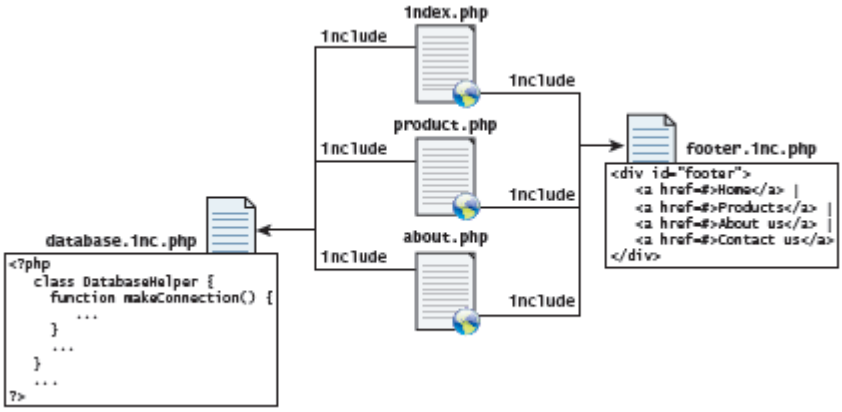
Include Files

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.

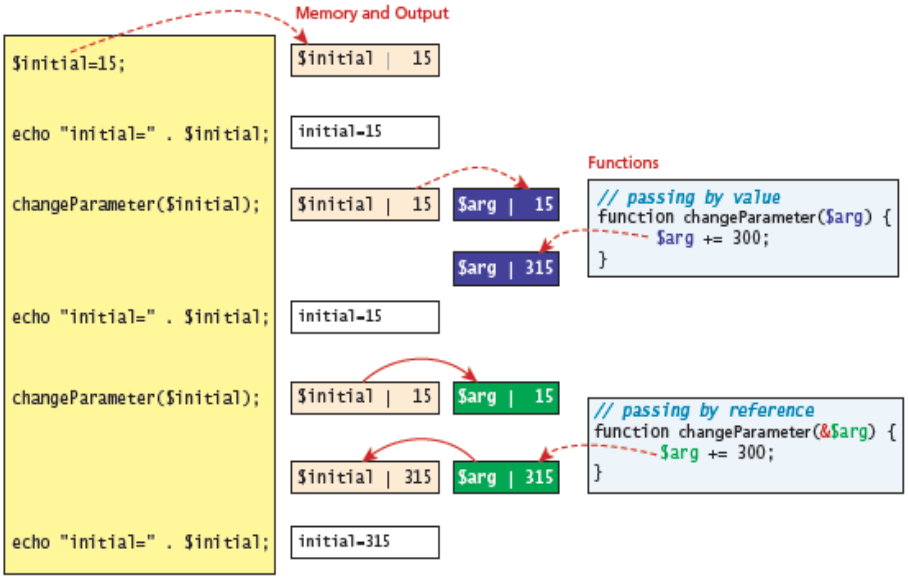
Eg:

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

```
changeParameter($initial);
echo "<br/>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;
?>
```

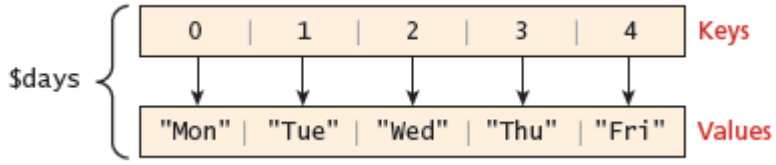
Output:
56
56

Module IV
Arrays, Classes & Objects, Exception Handling in PHP

4.1 Arrays

PHP supports arrays, like most other programming languages. An array is a data structure that allows the programmer to collect a number of related elements together in a single variable. In PHP an array is an **ordered map**, which associates each value in the array with a key.

An array of key-value pair, containing the days of the week can be visualized as follows –



Array keys in most programming languages are limited to integers, start at 0, and go up by 1. In PHP, keys can be either integers or strings.

Array values are not restricted to integers and strings. They can be any object, type, or primitive supported in PHP.

Defining and Accessing an Array

Declaration of an empty array named days:
$days = array()

Two different ways of defining an array –
Arrayvariable = array(elements separated by comma);
Arrayvariable = [elements separated by comma];

Eg -
$days = array(“Mon”,”Tue”,”Wed”,”Thu”,”Fri”);
$days = [“Mon”,”Tue”,”Wed”,”Thu”,”Fri”]; *// alternate syntax*

$count=array(“one”,”two”,3,4);
$count = [“one”,”two”,3,4];

In these examples, because no keys are explicitly defined for the array, the default key values are 0, 1, 2, . . . , n.

The array elements can also be defined individually using the square bracket notation:
$days = array();
$days[0] = "Mon";

```
$days[1] = "Tue";
$days[2] = "Wed";
// also alternate approach
$daysB = array();
$daysB[] = "Mon";
$daysB[] = "Tue";
$daysB[] = "Wed";
```

Arrays are dynamically sized - elements are added to or deleted from the array during run-time. Elements within an array are accessed using the familiar square bracket notation.
Arrayvariable[index];

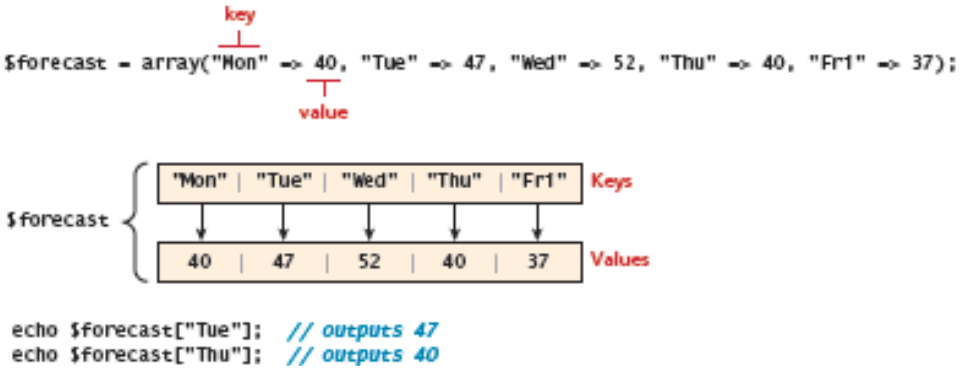
The code below echoes the value of our $days array for the key=1, which results in output of Tue.
echo "Value at index 1 is ". $days[1]; *// index starts at zero*

In PHP, keys can be explicitly defined along with the values. This allows us to use keys other than the classic 0, 1, 2, . . . , n to define the indexes of an array. The array
$days = array("Mon","Tue","Wed","Thu","Fri"); can be defined more explicitly by specifying the keys and values as

```
$days = array(0 => "Mon", 1 => "Tue", 2 => "Wed", 3 => "Thu", 4=> "Fri");
```

Where 0,1,2,… are the keys and "Mon","Tue",… are the values.

These types of arrays in PHP are generally referred to as **associative arrays**. Keys must be either integer or string values, but the values can be any type of PHP data type, including other arrays.



In the above example, the keys are strings (for the weekdays) and the values are weather forecasts for the specified day in integer degrees. To access an element in an associative array, simply use the key value rather than an index:
echo $forecast["Wed"]; *// this will output 52*

Multidimensional Arrays
PHP also supports multidimensional arrays, an array within another array. It can be defined as -

```
$month = array
(
```

```
array("Mon","Tue","Wed","Thu","Fri"),
array("Mon","Tue","Wed","Thu","Fri"),
array("Mon","Tue","Wed","Thu","Fri"),
array("Mon","Tue","Wed","Thu","Fri")
);
echo $month[0][3]; // outputs Thu
```

OR

```
$cart = array();
$cart[] = array("id" => 37, "title" => "Burial at Ornans", "quantity" => 1);
$cart[] = array("id" => 345, "title" => "The Death of Marat", "quantity" => 1);
$cart[] = array("id" => 63, "title" => "Starry Night", "quantity" => 1);

echo $cart[2]["title"]; // outputs Starry Night
```

Iterating through an Array

Iteration through the array contents is performed using the loops. Count() function is used to check the size of the array.

When there is nonsequential integer keys or strings as keys (i.e., an associative array), the accessing of elements can't be done using a $i, in such cases foreach loop is used. Foreach loop iterates till the end of array and stops. Each value can be accessed during the iterations, or both the key and value can be accessed, as shown below.

```
// while loop
$i=0;
while ($i < count($days)) {
        echo $days[$i] . "<br>";
        $i++;
}
```

```
// do while loop
$i=0;
do {
        echo $days[$i] . "<br>";
        $i++;
} while ($i < count($days));
```

```
// for loop
for ($i=0; $i<count($days); $i++) {
        echo $days[$i] . "<br>";
}
```

```
// foreach: iterating through the values
foreach ($forecast as $value) {
        echo $value . "<br>";
}

// foreach: iterating through both the values and the keys
foreach ($forecast as $key => $value) {
        echo "day" . $key . "=" . $value;
}
```

Adding and Deleting Elements

In PHP, arrays are dynamic. The arrays can grow or shrink in size. An element can be added to an array simply by using a key/index that hasn't been used.

```
$days[5] = "Sat";
```

Since there is no current value for key 5, the array grows by one, with the new key/value pair added to the end of our array. If the key had a value already, the new value replaces the value at that key.

As an alternative to specifying the index, a new element can be added to the end of any existing array by -

```
$days[ ] = "Sun";
```

The advantage to this approach is that you don't have to remember the last index key used.

```
$days = array("Mon","Tue","Wed","Thu","Fri");
$days[7] = "Sat";
print_r($days);
```

The print_r() - will display the non-NULL elements of the array, as shown below -

```
Array ([0] => Mon  [1] => Tue  [2] => Wed  [3] => Thu  [4] => Fri  [7] => Sat)
```

The **NULL** value are not displayed.

Elements can be deleted from the array explicitly by using the **unset**() function –

```
$days = array("Mon","Tue","Wed","Thu","Fri");
unset($days[2]);
unset($days[3]);
print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [4] => Fri )
```

The Null values in arrays can be removed and array is **reindexed by using the array_values**() function.

```
$b = array_values($days);
print_r($b); // outputs: Array ( [0] => Mon [1] => Tue [2] => Fri )
// Array is reindexed.
print_r($days) // outputs: Array ( [0] => Mon [1] => Tue [4] => Fri )
```

Checking If a Value Exists
It is possible to check if an array index contains a value or NULL by using the **isset()** function. The returns true if a value has been set, and false otherwise.

```
$days = array (1 => "Tue", 3 => "Wed", 5 => "Thurs");
if (isset($days[0]))
{
        // The code below will never be reached since $days[0] is not set!
        echo "Monday";
}
if (isset($days[1]))
{
        echo "Tuesday";
}
```

Array Sorting – Array Operations

There are many built-in sort functions, which sort by key or by value in PHP.

sort(arrayname); - sort the array in ascending order **by its values**. The array itself is sorted.

```
$days = array("Mon","Tue","Wed","Thu","Fri");
sort($days);
print_r($days); // outputs: Array ([0] => Fri [1] => Mon [2] => Thu [3] => Tue [4] => Wed)
```

sort() function loses the association between the values and the keys.

asort(arrayname) – sort the array in ascending order **by its values**, association is maintained.
Eg –

```
asort($days);
print_r($days); // outputs: Array ([4] => Fri [0] => Mon [3] => Thu [1] => Tue [2] => Wed)
```

More Array Operations
some key array functions are –

```
array_keys($Arrayname)
array_values($Arrayname)
array_rand($Arrayname, $num=1)
array_reverse($Arrayname)
array_walk($Arrayname, $callback, $optionalParam)
in_array($value, $Arrayname)
shuffle($Arrayname)
```

array_keys($Arrayname): This method returns an indexed array with the values being the *keys* of $Arrayname.
Eg : print_r(array_keys($days))
outputs - Array ([0] => 0 [1] => 1 [2] => 2 [3] => 3 [4] => 4)

array_values($Arrayname): This method returns an indexed array with the values being the *values* of $ Arrayname.
Eg: print_r(array_values($days))
Outputs - Array ([0] => Mon [1] => Tue [2] => Wed [3] => Thu [4] => Fri)

array_rand($Arrayname, $num=1): This function returns an array of, as many **random keys** as are requested. If you only want one, the key itself is returned;
otherwise, an array of keys is returned.
For example, print_r(array_rand($days,2))
might output: Array (3, 0)

array_reverse($Arrayname): This method returns $Arrayname in reverse order. The passed $Arrayname is not altered, a new array of reversed elements is returned.
For example, print_r(array_reverse($days))
outputs: Array ([0] => Fri [1] => Thu [2] => Wed [3] => Tue [4] => Mon)

array_walk($Arrayname, $callback, $optionalParam): This method allows to call a method ($callback), for each value in $Arrayname. One or more additional parameters can also be sent to the function.

The $callback function typically takes two parameters by default, the value, and the key of the array.
Eg - prints the value of each element in the array is shown below.

```
$someA = array("hello", "world");
array_walk($someA, "doPrint");

function doPrint($value,$key)
{
        echo $key . ": " . $value . “<br />”;
}
```

Output:
0 : hello
1 : world

Additional parameters can be sent to the function as shown below –

```
$someA = array("hello", "world");
array_walk($someA, "doPrint",”one”,”two”,3);

function doPrint($value,$key,$t1,$t2,$t3)
{
```

```
        echo $key . ": " . $value . "<br />";
        echo "$t1  $t2  $t3 <br />",
}
```

Output:
0 : hello
one two 3
1 : world
one two 3

in_array($value, $arrayname): This method search the array $ arrayname for a value ($value). It returns true if value is found, and false otherwise.

```
if(in_array("Wed",$days))
    echo "Element found";
else
    echo "Element not found";
```

shuffle($arrayname): This method shuffles $arrayname in random order and makes it an indexed array.

```
shuffle($days);
print_r($days);  // Array ( [0] => Thu [1] => Fri [2] => Wed [3] => Mon [4] => Tue)
shuffle($days);
print_r($days);  // Array ( [0] => Tue [1] => Mon [2] => Thu [3] => Wed [4] => Fri )
```

Superglobal Arrays

PHP uses special predefined associative arrays called **superglobal variables.** It allows the programmer to easily access HTTP headers, query string parameters, and other commonly needed information. They are called superglobal because these arrays are always accessible, from a function , class or file, without using the global keyword.

Some of the superglobal variables are –

NAME	DESCRIPTION
$GLOBALS	Array for storing user data that needs superglobal scope
$_COOKIES	Array of cookie data passed to page via HTTP request
$_ENV	Array of server environment data
$_FILES	Array of file items uploaded to the server
$_GET	Array of query string data passed to the server via the URL
$_POST	Array of query string data passed to the server via the HTTP header
$_REQUEST	Array containing the contents of $_GET, $_POST, and $_COOKIES
$_SESSION	Array that contains session data

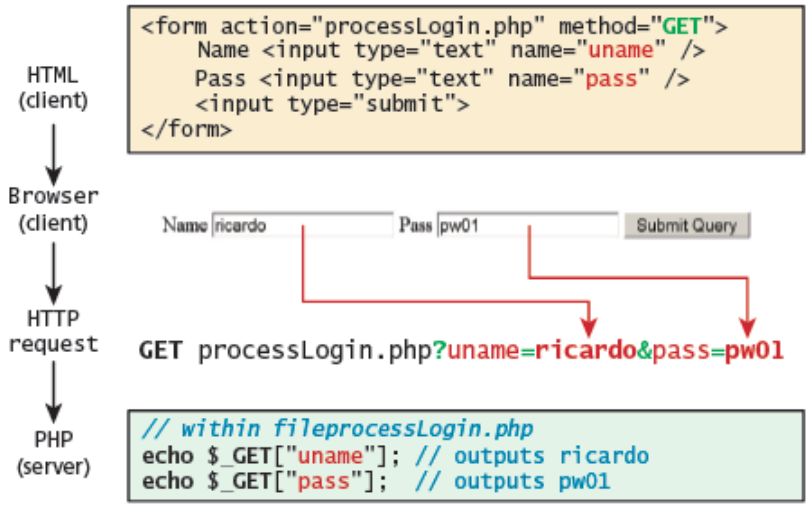
$_SERVER	Array containing information about the request and the server

4.2 $_GET and $_POST Superglobal Arrays

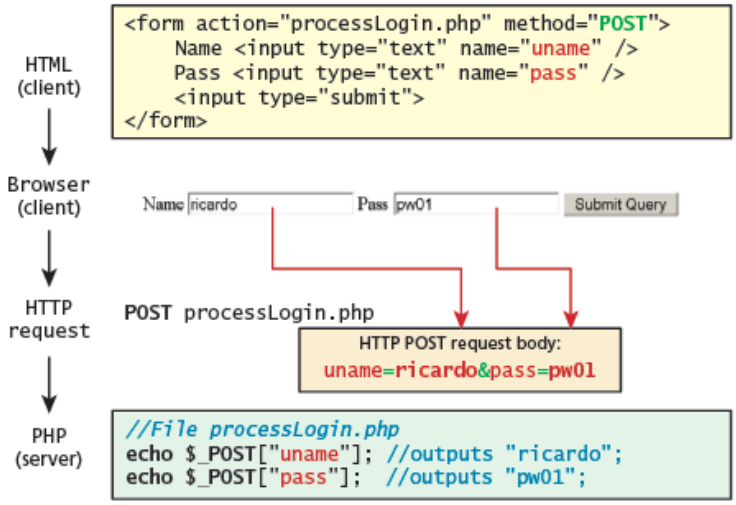
The $_GET and $_POST arrays allows the programmer to access data sent by the client in a query string. They are the most important superglobal variables in PHP.

An HTML form allows a client to send data to the server. That data is formatted such that each value is associated with a name of control defined in the form. If the form was submitted using an HTTP GET request ie. <form method="get" action="*server file path*" >, then the resulting URL will contain the data in the query string.
Eg-
https://name-processing.php?firstname=amith&lastname=kumar%20singh
retrieve the values sent through the URL using $_GET["firstname"] and $_GET["lastname"]

PHP will populate the superglobal $_GET array using the contents of this query string in the URL.



If the form was sent using HTTP POST, then the values will be sent through HTTP POST request body. The values and keys are stored in the $_POST array.



Thus the values passed by the user can easily be accessed at the server side using the global arrays $_GET[] and $_POST[] for 'get' and 'post' methods respectively.

Determining If Any Data Sent

Forms are used to send the input to the server. At the server side, to check if the user is sending the data by a POST or GET method the global variable $_SERVER['REQUEST_METHOD'] is used.

The variable $_SERVER['REQUEST_METHOD'] contains as a string, indicating the type of HTTP request as GET, POST, etc.

isset() function is used to check if any of the fields are set. It returns true if the value is sent otherwise returns false.

Syntax – isset($POST[fieldname]); or isset($GET[fieldname]);

isset($POST['name']);

Program to display the name and password using php script.

```
//PHP file named as samplePage.php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST")
{
        if ( isset($_POST["uname"]) && isset($_POST["pass"]) )
        {
                echo "handling user login now ...";
                $a=$_POST["uname"];
                $b=$_POST["pass"];
```

```
                echo "username is  $a <br /> password is $b";
        }
}
?>
```

HTML file

```
<!DOCTYPE html>
<html>
<body>
<h1>Some page that has a login form</h1>
<form action="samplePage.php" method="POST">
Name <input type="text" name="uname"/><br/>
Pass <input type="password" name="pass"/><br/>
<input type="submit">
</form>
</body>
</html>
```

Accessing Form Array Data

Sometimes it is required to access multiple values associated with a single name from few controls in form, like checkboxes.

A group of checkboxes will have the same name eg :(name="day"), and multiple values. $_GET['day'] value in the superglobal array *will only contain a single value - the last value from the list.*

To overcome this limitation, change the HTML in the form, ie. change the name attribute for each checkbox from day to day[].

```
<form method="get">
Please select days of the week you are free.<br />
Monday      <input type="checkbox" name="day[ ]" value="Monday" /> <br />
Tuesday     <input type="checkbox" name="day[ ]" value="Tuesday" /> <br />
Wednesday   <input type="checkbox" name="day[ ]" value="Wednesday" /> <br />
Thursday    <input type="checkbox" name="day[ ]" value="Thursday" /> <br />
Friday      <input type="checkbox" name="day[ ]" value="Friday" /> <br />
<input type="submit" value="Submit">
</form>
```

After making this change in the HTML, the corresponding variable $_GET['day'] will now have a value that is of type array. The values are accessed using the foreach loop and count() function returns the length of the array.

```
<?php
echo "You submitted " . count($_GET['day']) . "values";
foreach ($_GET['day'] as $d)
```

```
{
        echo $d . ", ";
}
?>
```

The output is the number of days selected and their values.
Eg

You submitted 3 values Monday , Thursday, Friday

Sanitizing Query Strings

The process of checking user input for incorrect or missing information is referred to as the process of **sanitizing user inputs**.

The user can give input in any way – wrong unexpected input, different data type, empty fields etc. The developer should be able to handle all such issues. He should always develop the web by distrusting user input.

The developer must be able to handle the following cases for *every* query string or form value:

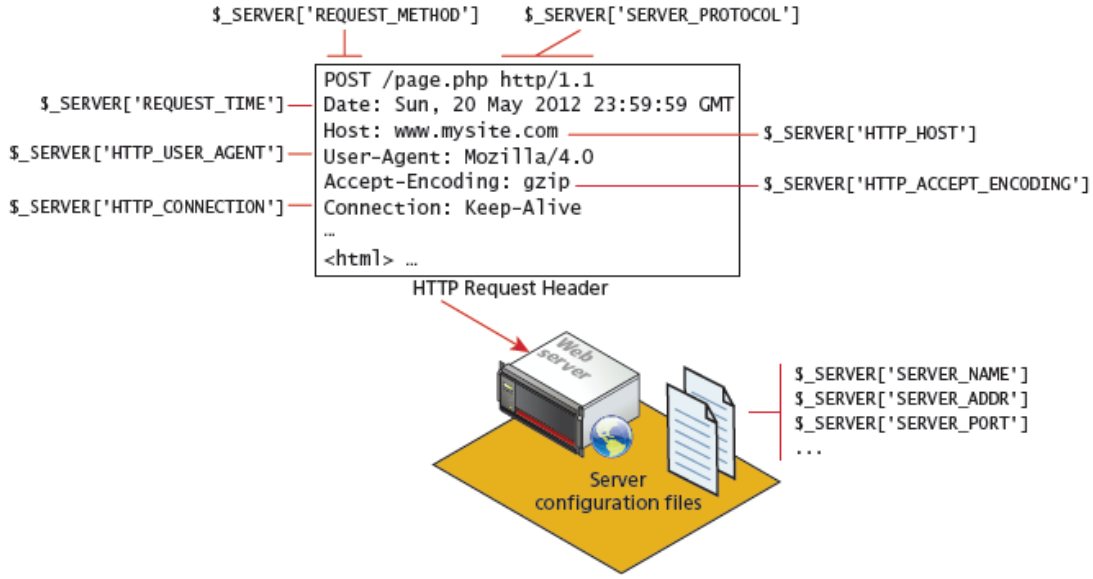
- Query string parameter doesn't exist.
- Query string parameter doesn't contain a value.
- Query string parameter value isn't the correct type.
- Value is required for a database lookup, but provided value doesn't exist in the database table.

4.3 $_SERVER Array

The $_SERVER associative array contains a variety of information. It contains information contained within HTTP request headers sent by the client. It also contains many configuration options for PHP.

Eg. of $_SERVER array and accessing the defined values -

```
echo $_SERVER["SERVER_NAME"] . "<br/>";
echo $_SERVER["SERVER_SOFTWARE"] . "<br/>";
echo $_SERVER["REMOTE_ADDR"] . "<br/>";
```



The $_SERVER array can be classified into keys containing information about the server settings and keys with request header information.

Server Information Keys
In the $_SERVER array,
SERVER_NAME key returns the name of the site that was requested.
SERVER_ADDR key returns the IP of the server.
DOCUMENT_ROOT key returns the file location from which the script is currently running.
SCRIPT_NAME key returns the name of actual script being executed.

Eg: $_SERVER['SERVER_NAME'] ;
$_SERVER['DOCUMENT_ROOT']; etc.

Request Header Information Keys
The web server responds to HTTP requests, and each request contains a request header. The following keys provide programmatic access to the data in the request header-
In the $_SERVER array,
REQUEST_METHOD key returns the request method that was used to access the page: GET, HEAD, POST, PUT.

REMOTE_ADDR key returns the IP address of the requestor.

HTTP_USER_AGENT key returns the operating system and browser used by the client.

HTTP_REFERER key returns the address of the page that referred (linked) us to this page.

Eg: $_SERVER['REQUEST_METHOD'] ;
$_SERVER['REMOTE_ADDR']; etc.

4.4 $_FILES Array

The $_FILES associative array contains information about the file that have been uploaded by the client to the current script. The <input type="file"> element is used to create the user interface for uploading a file from the client to the server. The user interface is only one part of the uploading process. A server script processes the upload file(s) using the $_FILES array.

HTML Requirements for File Uploading-

- The HTML form must use the HTTP POST method, since transmitting a file through the URL is not possible.
- Must add the enctype="multipart/form-data" attribute to the HTML form that is performing the upload so that the HTTP request can submit data in multiple pieces.
- Must include an input tag of type file in the form. This will show up with a browse button beside it so the user can select a file from their computer to be uploaded.

A simple form tag to upload a file to the server -

```
<form enctype='multipart/form-data' method='post'>
<input type='file' name='file1' id='file1' />
<input type='submit' />
</form>
```

Handling the File Upload in PHP

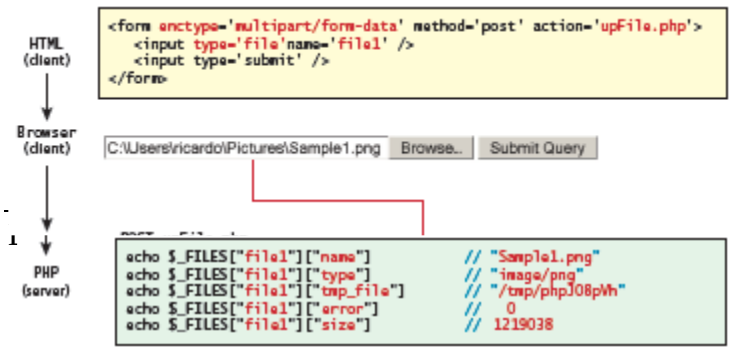
The superglobal $_FILES array is utilized at the server to handle the file uploaded by the client. This array will contain a key=value pair for each file uploaded in the post. The key for each element will be the name attribute from the HTML form, while the value will be an array containing information about the file as well as the file itself. The keys in that array are the name, type, tmp_name, error, and size.

$_FILES[name of file control][other keys]
Different files, uploaded will have a different file control names.

The values for each of the keys are -

- **name** is a string containing the full file name of file, including any file extension.
- **type** defines the MIME type of the file.
- **tmp_name** is the full path to the location on your server where the file is being temporarily stored. The file should be copied to another location if it is required.
- **error** is an integer that encodes many possible errors and is set to UPLOAD_ERR_OK (integer value 0) if the file was uploaded successfully.
- **size** is an integer representing the size in bytes of the uploaded file.

The below figure illustrates the process of uploading a file to the server and how the corresponding upload information is contained in the $_FILES array.



Checking for Errors

For every uploaded file, there is an error value associated with it in the $_FILES array. The error values are specified using constant values, which resolve to integers. The value for a successful upload is UPLOAD_ERR_OK. The other errors are –

Error Code	Integer	Meaning
UPLOAD_ERR_OK	0	Upload was successful.
UPLOAD_ERR_INI_SIZE	1	The uploaded file exceeds the `upload_max_filesize` directive in php.ini.
UPLOAD_ERR_FORM_SIZE	2	The uploaded file exceeds the `max_file_size` directive that was specified in the HTML form.
UPLOAD_ERR_PARTIAL	3	The file was only partially uploaded.
UPLOAD_ERR_NO_FILE	4	No file was uploaded. Not always an error, since the user may have simply not chosen a file for this field.
UPLOAD_ERR_NO_TMP_DIR	6	Missing the temporary folder.
UPLOAD_ERR_CANT_WRITE	7	Failed to write to disk.
UPLOAD_ERR_EXTENSION	8	A PHP extension stopped the upload.

File Size Restrictions

Some scripts limit the file size of each upload. There are three main mechanisms for maintaining uploaded file size restrictions: **via HTML in the input form, via JavaScript in the input form, and via PHP coding**.

To restrict the filesize **using HTML form**, add a **hidden** input field before any other input fields in your HTML form with a name of MAX_FILE_SIZE. This technique allows **php.ini** (a file used for configuring PHP settings) maximum file size to be large, while letting files with smaller size. The checking of file size is done at the server side.

```
<form enctype='multipart/form-data' method='post'>
<input type="hidden" name="MAX_FILE_SIZE" value="1000000" />
<input type='file' name='file1' />
<input type='submit' />
</form>
```

The more complete client-side mechanism to prevent a large file from uploading is to prevalidate the form **using JavaScript**.

```
<script>
var file = document.getElementById('file1');
var max_size = document.getElementById("max_file_size").value;
```

```
if (file.files && file.files.length ==1){
        if (file.files[0].size > max_size) {
        alert("The file must be less than " + (max_size/1024) + "KB");
        e.preventDefault();
        }
}
</script>
```

The third mechanism for limiting the uploaded file size is to add a simple check on the server side **using PHP**. This technique checks the file size on the server by simply checking the size field in the $_FILES array.

```
$max_file_size = 10000000;
foreach($_FILES as $fileKey => $fileArray)
{
        if ($fileArray["size"] > $max_file_size) {
                echo "Error: " . $fileKey . " is too big";
        }
        printf("%s is %.2f KB", $fileKey, $fileArray["size"]/1024);
}
```

Limiting the Type of File Upload

To check the type of file, check the file extension and the type of file using type key of $_FILES array.

```
$validExt = array("jpg", "png");
$validMime = array("image/jpeg","image/png");
foreach($_FILES as $fileKey => $fileArray )
{
        $extension = end(explode(".", $fileArray["name"]));
        if (in_array($fileArray["type"],$validMime) && in_array($extension, $validExt))
        {
                echo "all is well. Extension and mime types valid";
        }
        else
        {
                echo $fileKey." Has an invalid mime type or extension";
        }
}
```

Moving the File

PHP function move_uploaded_file() is used to move the file from the temporary file location to the file's final destination. This function will only work if the source file exists and if the destination location is writable by the web server (Apache). If there is a problem the function will return false.

```
$fileToMove = $_FILES['file1']['tmp_name'];
```

```
$destination = "./upload/" . $_FILES["file1"]["name"];
if (move_uploaded_file($fileToMove,$destination))
{
        echo "The file was uploaded and moved successfully!";
}
else
{
        echo "there was a problem moving the file";
}
```

4.5 Reading/Writing Files

There are two basic techniques for read/writing files in PHP:

- **Stream access** In this technique, just a small portion of the file is read at a time. It is the most memory-efficient approach when reading very large files.
- **All-In-Memory access** In this technique, the entire file is read into memory (i.e., into a PHP variable). While not appropriate for large files, it does make processing of the file extremely easy.

Stream Access

The functions used in this technique are similar to that of C programming.
The function fopen() takes a file location or URL and access mode as parameters. The returned value is a **stream resource (file handle)**.
Some of the common modes are "r" for read, "w" for write, and "c" creates a new file for writing, "rw" for read and write.

The other functions are –

fclose(file handle)- closing the file

fgets(file handle)- To read a single line, returns 0 if no more data
fread(file handle, no.of bytes) - To read an specified amount of data
fwrite(file handle, string)- To write the string to a file

fgetc(file handle) – To read a single character
feof(file handle) –checks for EOF

A program to read from a file and display it -

```
<?php
$f = fopen("sample.txt", "r");
while ($line = fgets($f))
{
        echo $line . "<br>";
}
fclose($f);
?>
```

In-Memory File Access

While the previous approach to reading/writing files requires more care in dealing with the streams, file handles etc. The alternative simpler approach is to read/write the entire contents into the memory with the help of variable.

The alternative functions to process the file are –

Function	Description
file()	Reads the entire file into an array, with each array element corresponding to one line in the file
file_get_contents	Reads the entire file into a string variable
file_put_contents	Writes the contents of a string variable out to a file

The file_get_contents() and file_put_contents() functions allow you to read or write an entire file in one function call. To read an entire file into a variable you can simply use:
$fileAsString = file_get_contents(FILENAME);

To write the contents of a string $writeme to a file, use
file_put_contents(FILENAME, $writeme);

Write a PHP script to retrieve the fields from the text file (EmpID, EmpName, Department). The fields are delimited by comma(,). Display the fields separately.

```
<?php
$filearr = file("Emp.txt");
Foreach($filearr as $empdata)
{
        $EmpFields = explode(',', $empdata);
        $id= $EmpFields[0];
        $name = $EmpFields[1];
        $dept = $EmpFields[2];
        Echo "ID of Employee is $id <br /> Name is $name <br /> Department is $dept<br />";
}
```

PHP Classes and Objects

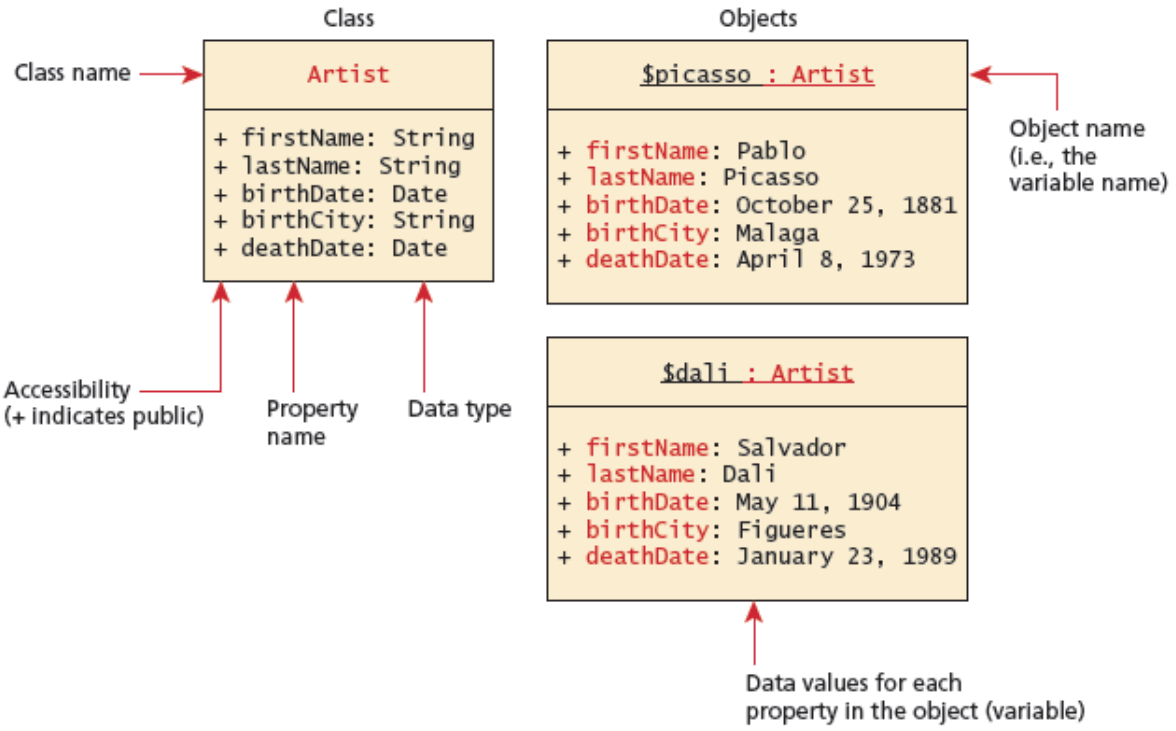
4.6 Object-Oriented Overview

PHP is a full-fledged object-oriented language like Java and C++.
Class is a user defined data-type which has data members and member functions.
Each variable created from a class is called an **object** or **instance**, and each object maintains its own set of variables.

The Unified Modeling Language

The standard diagramming notation for object-oriented design is **UML (Unified Modeling Language)**. UML is a set of diagramatic notations for visualizing, specifying, constructing and documenting the components of software system.
It was created by OMG(Object Management Group) in 1997.

Consider the 'Artist' class, with firstname, lastname, birthdate, birthcity and deathdate as members.



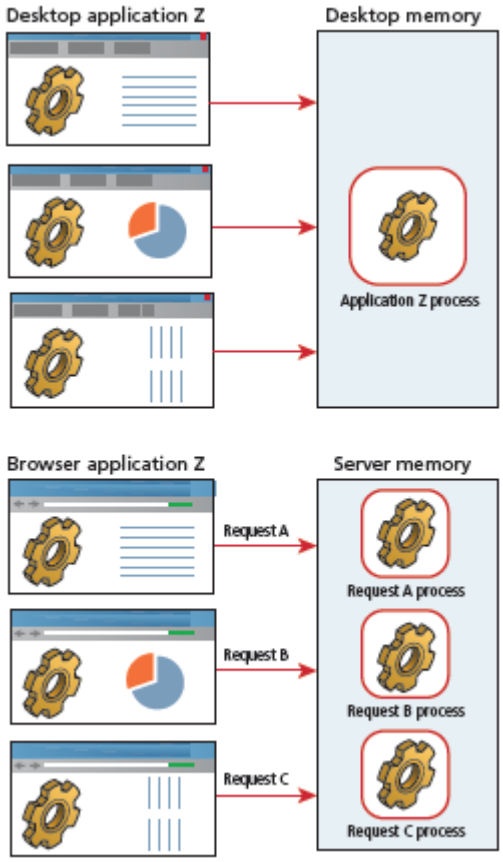
Differences between Server and Desktop Objects

One important distinction between web programming and desktop application programming is that the objects created exists only until a web script is terminated. While desktop software can load an object into memory and make use of it for several user interactions. Object must be recreated and loaded into memory for each request in web applications.

In a server, there are thousands of users making requests at once, so objects are destroyed upon responding to each request and also the memory is shared between many simultaneous requests.

It is possible to have objects persist between multiple requests in a server using serialization, which rapidly stores and retrieves object .

The below figure shows the lifetimes of objects in memory between a desktop and a browser application



4.7 Classes and Objects in PHP

Developers can create their own classes in PHP. Classes should be defined in their own files so they can be imported into multiple scripts. A class file is named as **classname.class.php**. Any PHP script can make use of an external class by including the class file in the script.

Defining Classes

The PHP syntax for defining a class uses the class keyword followed by the classname and { } braces. The properties and methods of the class are defined within the braces.

Eg –

```
class Employee{
public $EmpName;
public $EmpID;
public $Dept;
public $Designation;
public $salary;
}
```

Each property in the class is declared using one of the keywords public, protected, or private access specifiers followed by the property or variable name.

Instantiating Objects
Once a class has been defined, any number of instances of that class can be created, using the **new** keyword.

To create two new instances of the Employee class called $emp1 and $emp2, you instantiate two new objects using the new keyword as follows:
$emp1 = new Employee();
$emp2 = new Employee ();

Properties
The objects can access and modify the properties using an arrow (->).
$emp1 = new Employee();
$emp2 = new Employee ();
$emp1->EmpName = “Pablo”;
$emp1->EmpID = “M123”;
$emp1->Dept = “Testing”;
$emp1-> Designation = “Manager”;
$emp1->salary = “35K”;

Constructors
A **constructor** is a member function of a class which initializes objects of a class. Objects can be done by using above method also, but if there are many objects then it is difficult to initialize.

In PHP, constructors are defined as functions with the name __construct() (*two* underscores __ before the word construct.).

```
class Employee{
        public $EmpName;
        public $EmpID;
        public $Dept;
        public $Designation;
        public $salary;
        function __construct($name,$id,$dept,$desig,$sal)
        {
                $this->EmpName = $name;
                $this->EmpID = $id;
                $this->Dept = $dept;
                $this-> Designation = $desig;
                $this->salary = $sal;
        }
}
```

In the constructor each parameter is assigned to an internal class variable using the $this-> syntax. Inside a class $this syntax must be used to reference all properties and methods associated with this particular instance of a class.
The object is created as follows –

```
$emp1 = new Employee("Pablo","M123","Testing","Manager","35K");
$emp2 = new Employee ("Salvador","M124","Testing","Trainee",”10K”);
```

Methods

In object-oriented concept, the operations on properties of objects are performed within the **methods** defined in a class. Methods are like functions, except they are associated with a class. They define the tasks each instance of a class can perform and are useful since they associate behavior with objects.

The below code uses the display function to display the details of an object.

```
<?php
Class Student
{
        public $name;
        public $USN;
        public $address;
        public $avg;

        function __construct($n,$usn,$add,$avg)
        {
                $this->name = $n;
                $this->USN= $usn;
                $this->address = $add;
                $this->avg = $avg;
        }
        function display()
        {
             echo “name is $this->name <br /> USN is $this->USN <br />”;
              echo “Address is $this->address <br /> Avg is $this->avg <br />”;
        }
}

$s1 = new Student("Sajeev", "1VA15CS013","Bangalore",66);
$s1->display();
?>
```

Class

Student
+ name: String + USN: String + address: String + avg: float
+ __construct(int, String, String, String) + display( ) : void

Object

$s1:Student
+ name: Sajeev + USN: 1VA15CS013 + address: Bangalore + avg: 66
+ __construct(int, String, String, String) + display( ) : void

_ _toString( ) method – Any function which returns a string, can be renamed as _ _toString( ). The function is called by just using the object as a variable.
Eg-

```
$s1 = new Student( );
Echo $s1;   //calls the _ _toString( ) function and displays the string returned.
```

Visibility
The **visibility** of a property or method determines the accessibility of a **class member** (i.e., a property or method) and can be set to public, private, or protected.
The public keyword means that the property or method is accessible to any code anywhere that has a reference to the object.
The private keyword sets a method or variable to only be accessible from within the class. This means that we cannot access or modify the property from outside of the class, even if it is referenced with an object.
The protected keyword sets a method or variable to be accessible from within the class and from its derived classes.

In UML, the "+" symbol is used to denote public properties and methods, the "–" symbol for private members, and the "#" symbol for protected members.

Specifiers	Within Same Class	In Derived Class	Outside the Class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

Static Members
A **static** member is a property or method that all instances of a class share. Unlike an instance property, where each object gets its own value for that property, there is only one value for a class's static property. It is common for all objects.

A variable is declared static by including the static keyword in the declaration:
public static $StudentCount = 0;

A static property is accessed within a class using self::syntax. This is an indication to programmer to understand that the variable is static and is not associated with an instance ($this). This static variable is accessed outside the class, without any instance of an Student object by using the class name, that is, Student::$StudentCount.

Static methods are similar to static properties in that they are globally accessible (if public) and are not associated with particular objects. Static methods cannot access instance members. Static methods are called using the same double colon syntax as static properties.

Static members are used when some data or operations are independent of the instances of the class.

```
<?php
Class Student
{
        public $name;
        public $USN;
        public $address;
        public $avg;
        public static $StudentCount = 0;

        function __construct($n,$usn,$add,$avg)
        {
                $this->name = $n;
                $this->USN= $usn;
                $this->address = $add;
                $this->avg = $avg;
                self::$StudentCount++;
        }
}

$s1 = new Student("Sajeev", "1VA15CS013","Bangalore",66);
Echo Student::$StudentCount;
?>
```

Output:

1

In the UML notation the shared property is underlined to indicate its static nature.

Class

Student
+ StudentCount: int + name: String + USN: String + address: String + avg: float
+ __construct(int, String, String, String)

Object

$s1:Student
+ self::$StudentCount + name: Sajeev + USN: 1VA15CS013 + address: Bangalore + avg: 66
+ __construct(int, String, String, String)

Class Constants

Constant values are stored more efficiently as class constants using 'const' keyword. There will be the constant values as long as they are not calculated or updated.

Eg - const EARLIEST_DATE = 'January 1, 1200';
Unlike all other variables, constants don't use the $ symbol when declaring or using them. They can be accessed both inside and outside the class using self::EARLIEST_DATE in the class and classReference::EARLIEST_DATE outside.

4.8 Object-Oriented Design

The object-oriented design of software offers many benefits in terms of modularity, testability, and reusability.

Objects can be reused across the program. The software is easier to maintain, as any changes in the structure need to change only the class. OO concepts enables faster development and easier maintenance of the software.

Data Encapsulation

Object-oriented design enables the possibility of **encapsulation** (hiding), that is, restricting access to an object's internal components (properties and methods). Another way of understanding encapsulation is: it is the hiding of an object's implementation details.

In properly encapsulated class, the properties and methods, are hidden using the private access specifier and these properties and methods are accessed to outside the class using the public methods. Thus we can restrict the usage of required properties and methods.

The hidden properties are accessed and modified using public methods commonly called **getters and setters** (or accessors and mutators). A getter returns a variable's value does not modify the property. It is normally called without parameters, and returns the property from within the class.
Eg -

```
public function getFirstName() {
        return $this->firstName;
}
```

Setter methods modify properties, and allow extra logic to be added to prevent properties from being set to strange values.

Eg –

```
public function setFirstName($name) {
        $this->firstName = $name;
}
```

The below example shows the modified Student class with getters and setters. Some of the properties are private. These properties cannot be accessed or assigned from outside the class, the getters and setters.

```
<?php
Class Student
{
        private $name;
        private $USN;
```

```
        public $address;
        public $avg;

function __construct($n,$usn,$add,$avg)
{
        $this->name = $n;
        $this->USN= $usn;
        $this->address = $add;
        $this->avg = $avg;
}

public function getname() { return $this->name; }
public function getUSN() { return $this->USN; }

public function setname($fullname) { $this->name=$fullname; }
public function setUSN($usn) { $this->USN=$usn; }
}

$s1 = new Student("Sajeev", "1VA15CS013","Bangalore",66);
$s1->setname("Adithya");
$s1->avg = 77;                  //setting avg directly , where as name is set using the function
$name = $s1->getname();
$usn= $s1->getUSN();
echo "name is $name <br /> USN is $usn <br />";
echo "address is $s1->address <br /> average is $s1->avg";

?>
```

Note: '$s1->address' and '$s1->avg' – can be accessed directly, without using any get function.

Two forms of the UML class diagram for our data encapsulated class are –

Class

Student
- name: String - USN: String + address: String + avg: float
+ __construct(int, String, String, String) + getname() : String + getUSN() : String + setname(String) : void + setUSN(String) : void

Class

Student
- name: String - USN: String + address: String + avg: float
+ __construct(int, String, String, String)

The first UML diagram includes, all the getter and setter methods. UML diagram may also exclude the getter and setter methods from a class as shown in the second diagram.

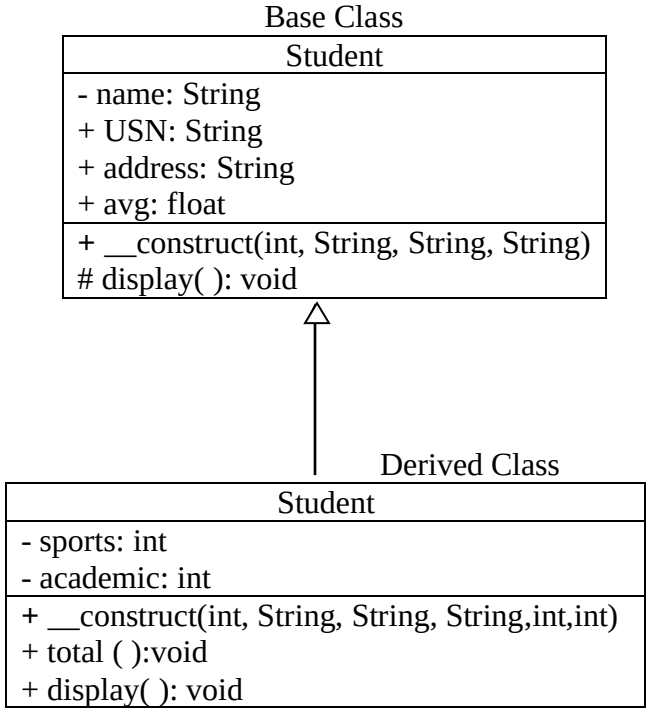
Inheritance

Inheritance enables you to create new PHP classes that reuse, extend, and modify the behavior that is defined in another PHP class. A class that is inheriting the properties and methods from another class is said to be a **subclass,** a **derived class** or a **child class**. The class that is being inherited is called a **superclass ,** a **base class or a parent class.**

When a class inherits from another class, it inherits all of its public and protected methods and properties. Just as in Java, a PHP class is defined as a subclass by using the extends keyword.

class Painting extends Art { . . . }

Inheritance in a UML class diagram -



<?php

```
Class Student
{
        private $name;
        public $USN;
        public $address;
        public $avg;

function __construct($n,$usn,$add,$avg)
{
        $this->name = $n;
        $this->USN= $usn;
        $this->address = $add;
        $this->avg = $avg;
}
protected function display()
{
    echo $this->name;
}
}

class FeeDetails extends Student
{
        private $sports;
        private $academic;

        function __construct($n,$usn,$add,$avg,$s,$ac)
        {
        parent::__construct($n,$usn,$add,$avg);
        $this->sports= $s;
        $this->academic= $ac;
        }

        public function total()
        {
                $sum= ($this->sports)+($this->academic);
                echo "total fee is $sum";
        }
        public function display()
        {
                parent::display();
                $this->total();
        }
}

$f1 = new FeeDetails("Sajeev", "1VA15CS013","Bangalore",66,25000,3000);
$f1->display();
?>
```

Referencing Base Class Members

A subclass inherits the public and protected members of the base class. In PHP any reference to a member in the base class requires the addition of the **parent::** prefix instead of the $this-> prefix. So within the FeeDetails class, a reference to the parent constructor or display() method would be done by using parent:: prefix.

It is important to note that private members in the base class are **not** available to its subclasses. Thus, within the FeeDetails class, a reference like the following would **not** work.

$abc = parent::name; *// would not work within the* FeeDetails *class*

If a member has to be available to subclasses but not anywhere else, it can be done by using the protected access modifier. 'name' member of base class can be accessed in base class only as it is private. It is displayed using '$this->name' in base class.

Inheriting Methods

Every method defined in the base/parent class can be overridden when extending a class, by declaring a function with the same name. A simple example of overriding is done in the previous program, the display( ) method of subclass FeeDetails overrides the display( ) method of parent class.

To access a public or protected method or property defined within a base class from within a subclass, use the member name with parent:: prefix.

Parent Constructors

To invoke a parent constructor from the derived class's constructor, use the parent:: syntax and call the constructor on the first line parent:: _ _construct(). This is similar to calling other parent methods, except that to use it we *must* call it at the beginning of constructor.

Polymorphism

Polymorphism is the OO concept, where the object can act differently at different times. Even if the same function name is used.

Overriding of methods in two or more derived classes takes place, the actual method called will depend on the type of the object. In the below example program, the display ( ) method is defined in both the derived classes. The method accessed depends on the object used to access the function.

```
<?php

Class Student
{
        private $name;
        public $USN;
        public $address;
```

```
        public $avg;

function __construct($n,$usn,$add,$avg)
{
        $this->name = $n;
        $this->USN= $usn;
        $this->address = $add;
        $this->avg = $avg;
}
protected function display()
{
    echo $this->name;
}
}

class FeeDetails extends Student
{
        private $sports;
        private $academic;

        function __construct($n,$usn,$add,$avg,$s,$ac)
        {
        parent::__construct($n,$usn,$add,$avg);
        $this->sports= $s;
        $this->academic= $ac;
        }

        public function total()
        {
                $sum= ($this->sports)+($this->academic);
                echo "total fee is $sum";
        }
        public function display()
        {
                parent::display();
                $this->total();
        }
}

class Scholarship extends Student
{

        function __construct($n,$usn,$add,$avg)
        {
        parent::__construct($n,$usn,$add,$avg);
        }

        public function display()
```

```
        {
                parent::display();
                echo "No fees";
        }
}

$f1 = new FeeDetails("Sajeev", "1VA15CS013","Bangalore",66,25000,3000);
$s1=new Scholarship("Amith", "1VA15CS013","Bangalore",66);
$f1->display();
$s1->display();

?>
```

OUTPUT:
Sajeev total fee is 28000 Amith No fees

The same display( ) method is invoked by two different objects. First the display( ) function of FeeDetails class is invoked and then the display( ) function of Scholarship class is invoked.

The run time decision of determining the function to be invoked at run time, is called **dynamic dispatching**. Just as each object can maintain its own properties, each object also manages its own table of methods. This means that two objects of the same type can have different implementations with the same name.

Object Interfaces
An object **interface** is a way of declaring a formal list of methods(only) without specifying their implementation. The class that uses the interface **must** implement the declared functions.

Interfaces provide a mechanism for defining what a class can do without specifying how it does it. Interfaces are defined using the interface keyword. It looks similar to PHP classes, except an interface contains no properties and its methods do not have method bodies defined.

Eg -

```
interface Viewable {
public function getmessage();
public function getData();
}
```

Notice that an interface contains only public methods, and instead of having a method body, each method is terminated with a semicolon.

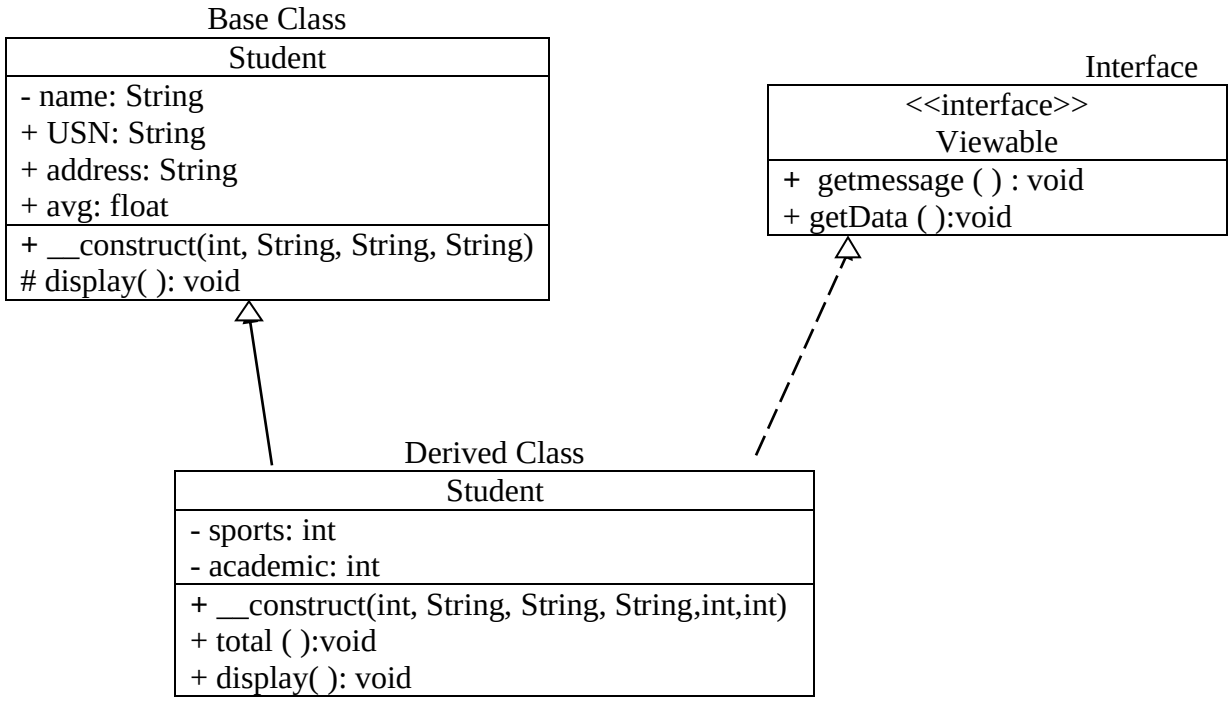
In PHP, a class can be said to *implement* an interface, using the **implements** keyword:

```
class FeeDetails extends Student implements Viewable
{ ... }
```

This means then that the class ‘FeeDetails’ must provide implementations for the getmessage() and getData() methods.

Implementing an interface allows a class to become more formal about the behavior it promises to provide. The same interface can be inherited by more than one class. It helps in the implementation of multiple inheritance.

PHP allows implementing two or more interfaces. The UML diagram to denote the interfaces include the <<interface>> stereotype. Classes that implement an interface are shown to implement using the same hollow triangles as inheritance but with dotted lines.



Runtime Class and Interface Determination

During the implementation, it might be required to know the class of a particular object or the interfaces used in the object and so on, this is possible in PHP using some of the built-in functions.

To display the class name of an object $x use the get_class() function:
echo get_class($x);

Similarly we can access the parent class with:
echo get_parent_class($x);

To determine what interfaces this class has implemented, use the function class_implements(), which returns an array of all the interfaces implemented by this class or its parents.
$allInterfaces = class_implements($x);

Error Handling and Validation

4.9 What Are Errors and Exceptions?

Even the best-written web application can suffer from runtime errors. Most complex web applications must interact with external systems such as databases, web services, email servers, file system, and other externalities that are beyond the developer's control. A failure in any one of these systems will mean that the web application will no longer run successfully. It is vitally important that web applications gracefully handle such problems.

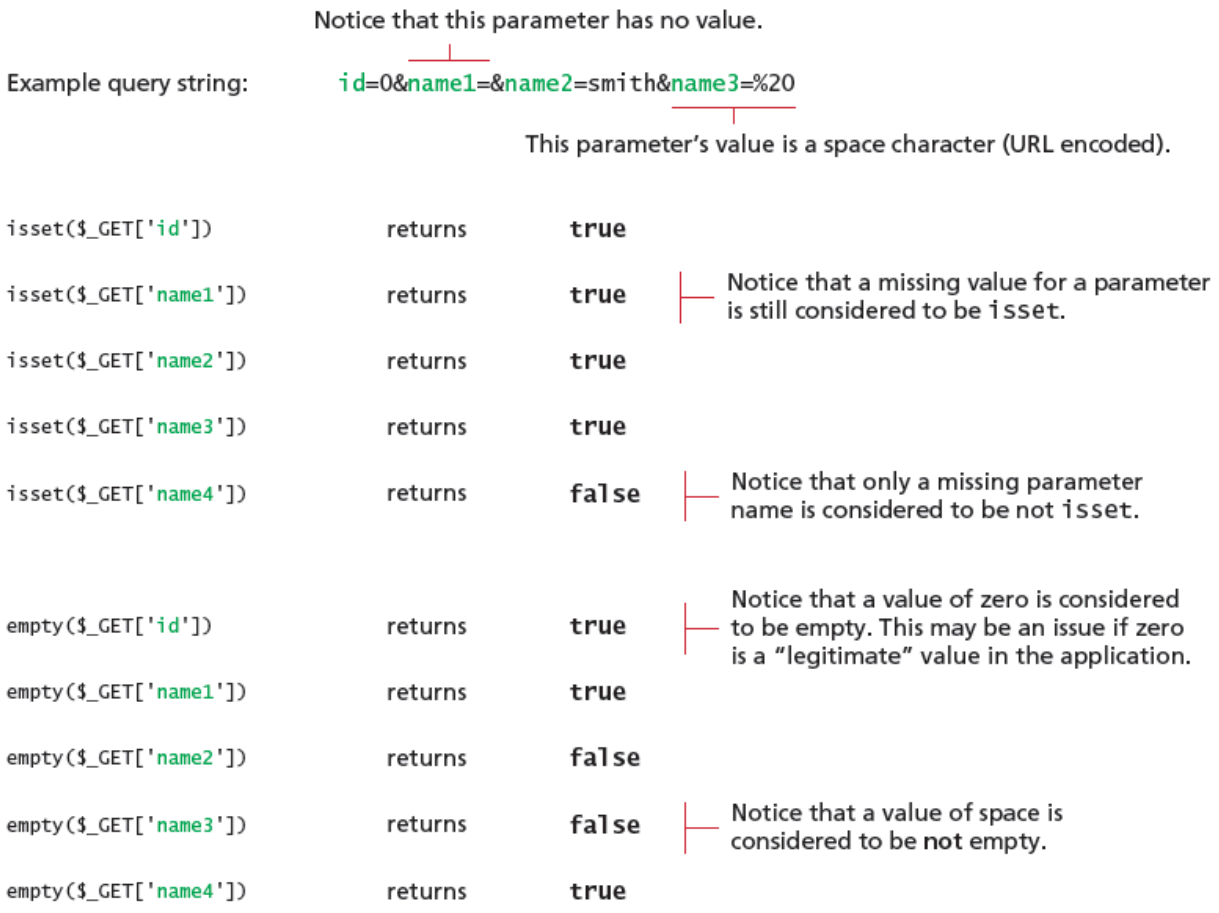
Types of Errors
There are three different types of website problems:

- Expected errors
- Warnings
- Fatal errors

An **expected error** is an error that routinely occurs during an application. The most common example of this type of error is as a result of user inputs, ie. , entering letters when numbers were expected. The web application should be developed such that, we expect the user to not always enter expected values. Users will leave fields blank, enter text when numbers were expected, type in too much or too little text, forget to click certain things, and click things they should not etc.

Not every expected error is the result of user input. Web applications that rely on connections to externalities such as database management systems, legacy software systems, or web services is expected to occasionally fail to connect. So there should be some type of logic that verifies the errors in code, check the user inputs and check if it contains the expected values.

PHP provides two functions for testing the value of a variable.
isset() function - which returns true if a variable is not null.
empty() function - which returns true if a variable is null, false, zero, or an empty string.



To check a numeric value, use the is_numeric() function, as shown

```
$id = $_GET['id'];
if (!empty($id) && is_numeric($id) ) {
// use the query string since it exists and is a numeric value
...
}
```

Another type of error is **warnings**, which are problems that generate a PHP warning message (which may or may not be displayed) but will not halt the execution of the page. For instance, calling a function without a required parameter will generate a warning message but not stop execution. While not as serious as expected errors, these types of incidental errors should be eliminated by the programmer.

The **fatal errors**, which are serious in that the execution of the page will terminate unless handled in some way. These types of errors are exceptional and unexpected, such as a required input file being missing or a database table or field disappearing. These types of errors need to be reported so that the developer can try to fix the problem, and also the page needs to recover gracefully from the error so that the user is not excessively puzzled or frustrated.

Exceptions

In the context of PHP, error and exception is not the same. An **error** is some type of problem that generates a nonfatal warning message or that generates an error message that terminates the program's execution. An **exception** refers to objects that are of type Exception and which are used in conjunction with the object-oriented try . . . catch language construct for dealing with runtime errors.

4.10 PHP Error and Exception Handling

When a fatal PHP error occurs, program execution will eventually terminate unless it is handled. The PHP documentation provides two mechanisms for handling runtime errors: procedural error handling and the more object-oriented exception handling.

Procedural Error Handling

In the procedural approach to error handling, the programmer needs to explicitly test for error conditions after performing a task that might generate an error. Eg - While connecting to the database, we may need to test for and deal with errors after each operation that might generate an error state -

```
$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);
$error = mysqli_connect_error();
if ($error != null) {
// handle the error
...
}
```

This type of coding requires the programmer to know ahead of time what code is going to generate an error condition and also result in a great deal of code duplication. The advantage of the try . . . catch mechanism is that it allows the developer to handle a wider variety of exceptions in a single catch block.

Object-Oriented Exception Handling

When a runtime error occurs, PHP *throws* an *exception*. This exception can be *caught* and handled either by the function, class, or page that generated the exception or by the code that called the function or class. If an exception is not caught, then eventually the PHP environment will handle it by terminating execution with an "Uncaught Exception" message.

PHP also uses the try . . . catch programming construct to programmatically deal with exceptions at runtime. The catch construct expects some type of parameter of type Exception. The Exception built-in class provides methods for accessing not only the exception message, but also the line number of the code that generated the exception and the stack trace, both of which can be helpful for understanding where and when the exception occurred.

```
// Exception throwing function
function throwException($message = null,$code = null)
{
        throw new Exception($message,$code);
}
try {
```

```
        // PHP code here
        $connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME)
        or throwException("error");
        //...
}
catch (Exception $e) {
        echo ' Caught exception: ' . $e->getMessage();
        echo ' On Line : ' . $e->getLine();
        echo ' Stack Trace: '; print_r($e->getTrace());
}

finally {
// PHP code here that will be executed after try or after catch
}
```

The finally block is optional. Any code within it will always be executed *after* the code in the try or in the catch blocks, even if that code contains a return statement. It is typically used if the developer wants certain things to do regardless of whether an exception occurred, such as closing a connection or removing temporary files.

It is also possible in PHP to programmatically throw an exception via the throw keyword. An exception is thrown when an expected programming assumption is not met. It also possible to rethrow an exception within a catch block.

```
function processArray($array)
{
// make sure the passed parameter is an array with values
if ( empty($array) ) {
        throw new Exception('Array with values expected');
}
// process the array code
...
}
```

Rethrowing the exception

```
try {
// PHP code here
}
catch (Exception $e) {
        // do some application-specific exception handling here
        ...
        // now rethrow exception
        throw $e;   // same exception is rethrown
}
```

4.11 PHP Error Reporting

PHP has a flexible and customizable system for reporting warnings and errors that can be set programmatically at runtime or declaratively at design-time within the **php.ini** file. There are three main error reporting flags:

- error_reporting
- display_errors
- log_errors

Note: PHP.ini is a configuration file, that is used to customize behavior of PHP at runtime. It consists of settings for different values such as register global variables, display errors, log errors, max uploading size setting, maximum time to execute a script and other configurations. When PHP Server starts up it looks for PHP.ini file first to load various values for settings.

The error_reporting Setting

The error_reporting setting specifies which type of errors are to be reported. It can be set programmatically inside any PHP file by using the error_reporting() function:

```
error_reporting(E_ALL);
```

It can also be set within the **php.ini** file:

```
error_reporting = E_ALL
```

The possible levels for error_reporting are defined by predefined constants(the default setting is zero, that is, no reporting).

Constant Name	Value	Description
E_ALL	8191	Report all errors and warnings
E_ERROR	1	Report all fatal runtime errors
E_WARNING	2	Report all nonfatal runtime errors (i.e., warnings)
	0	No reporting

The display_errors Setting

The display_error setting specifies whether error messages should or should not be displayed in the browser. It can be set programmatically via the ini_set() function:

```
ini_set('display_errors','0');
```

It can also be set within the **php.ini** file:

```
display_errors = Off
```

The log_error Setting

The log_error setting specifies whether error messages should or should not be sent to the server error log. It can be set programmatically via the ini_set() function:

```
ini_set('log_errors','1');
```

It can also be set within the **php.ini** file:

log_errors = On

When logging is turned on, error reporting will be sent to either the operating system's error log file or to a specified file in the site's directory. If error messages is to be saved in a log file in the user's directory, the file name and path can be set via the error_log setting.
ini_set('error_log', '/restricted/my-errors.log');

It can also be set within the **php.ini** file:
error_log = /restricted/my-errors.log

You can also programmatically send messages to the error log at any time via the error_log()

```
$msg = 'Some horrible error has occurred!';
// send message to system error log (default)
error_log($msg,0);
// email message
error_log($msg,1,'support@abc.com','From: somepage.php@abc.com');
// send message to file
error_log($msg,3, '/folder/somefile.log');
```

Important questions –

1. How is array defined in PHP? Explain the array operations of PHP.
2. What are superglobal arrays? List them
3. Explain the process of populating $_GET superglobal array.
4. Explain the process of populating $_POST superglobal array.
5. Explain the $_SERVER array
6. Explain the use of $_FILES array.
7. Explain how the files are uploaded from the browser and values are accessed at the server
8. How is the size of file restricted in PHP
9. Explain the stream access of file in PHP.
10. Explain the All-In-Memory access of file in PHP.
11. How are the classes declared in PHP
12. How to access methods and properties of class in PHP
13. Explain the use of static members
14. How are constructors defined in PHP. Explain with example.
15. Explain encapsulation with example.
16. Explain inheritance with example.
17. Explain polymorphism with example.
18. Explain interface with example.
19. How is exception handled in PHP.
20. What are the different types of errors
21. Explain the different error reporting flags of PHP.
22. Wrie a PHP script to move the uploaded file to the desired location.
23. Wrie a PHP script to limit the type of file.

Module I
Introduction to HTML

What Is HTML and Where Did It Come from?

- HTML is a markup language
- First public specification of the HTML was published by Tim Berners-Lee in 1991
- HTML's codification by the World Wide Web Consortium (better known as the W3C) started in 1997.
- The Netscape Navigator and Microsoft Internet Explorer of the early and mid-1990s, a time when intrepid developers working for the two browser manufacturers ignored the W3C and brought forward a variety of essential new tags (such as, for instance, the <table> tag), and features such as CSS and JavaScript, all of which have been essential to the growth and popularization of the web.
- In 1998 the W3C froze the HTML specification at version 4.01.

XHTML

- In 1990s, the W3C developed a new specification called XHTML 1.0, which was a version of HTML that used stricter **XML** (extensible markup language) syntax rules.
- The goal of XHTML with strict rules was to make page rendering more predictable by forcing web authors to create web pages without **syntax errors.**

There are two versions of XHTML.
A) **XHTML 1.0 Transitional** and B) **XHTML 1.0 Strict**

- The **Transitional** version of HTML is the most common type of HTML. It has a flexible syntax, or grammar. Over the years, transitional HTML has been used without syntax restrictions. If tags are misspelled, the browsers do not correct web developer's errors, and they display the content anyway. Browsers do not report HTML errors, they simply display what they can.

- The **strict** version of HTML is meant to specify rules into HTML and make it more reliable. As a clean and error-free code helps to load pages faster, it uses the tag support described by the W3C XHTML 1.0 Strict specification. For example, the strict type requires closing all tags for all opened tags.

HTML versus XHTML

- There are some commonly heard arguments for using HTML rather than XHTML, especially XHTML 1.0 Strict. First, because of its less syntax rules, HTML is much **easier to write**, whereas XHTML requires a level of discipline many of us naturally resist.
- Because of the huge number of HTML documents available on the Web, browsers will continue to **support HTML** as far as one can see into the future. Indeed, some older browsers have problems with some parts of XHTML.
- There are strong reasons that one should use XHTML. One of the most compelling is

that quality and **consistency** in any endeavour

- HTML has few syntactic rules, and HTML processors (e.g., browsers) do not enforce the rules it does have. Therefore, HTML authors have a high degree of freedom to use their own syntactic preferences to create documents. Because of this freedom, HTML documents **lack consistency**, both in low-level syntax and in overall structure. XHTML has strict syntactic rules that impose a consistent structure on all XHTML documents.
- Another significant reason for using XHTML is that when you create an XHTML document, its syntactic correctness can be checked, either by an XML browser or by a **validation tool**.

XML

XML is a textual markup language, the formal rules for XML were set by the W3C. The XML-syntax rules are pretty easy to follow. The main rules are:

- There must be a single root element.
- Element names are composed of any of the valid characters (most punctuation symbols and spaces are not allowed) in XML.
- Element names can't start with a number.
- Element and attribute names are case sensitive.
- Attributes must always be within quotes.
- All elements must have a closing element (or be self-closing).

XML also provides a mechanism for validating its content : for instance, whether the text inside an element called <date> is actually a valid date, or the text within an element called <year> is a valid integer and falls between, say, the numbers 1950 and 2010.

HTML5

- A group of developers at Opera and Mozilla formed the **WHATWG** (Web Hypertext Application Technology Working Group) group within the W3C, and worked to enhance the features of HTML to higher versions.
- The WHATWG group was very small group led by Ian Hickson. By 2009, the W3C supported the WHATWG group and the work done by them and named it as HTML5.

There are three main aims of HTML5:

1. Specify unambiguously how browsers should deal with invalid markup.
2. Provide an open, nonproprietary programming framework (via JavaScript) for creating rich web applications.
3. Be backwards compatible with the existing web programs.

Structure of HTML Documents

A simple html program

```
<!DOCTYPE html>                                  → Defines the type of document
<html>                                           → Root element – indicates that webpage is written in html
  <head>
      <title> A simple Program                   → Contains title & information about this web page
      </title>
  </head>
  <body>
      <p> A simple program to show the working of html documents</p>   → Contents to be displayed on webpage
  </body>
</html>                                          → Closing of html document
```

The DOCTYPE (**Document Type Definition**) element informs the browser, the type of document it is about to process.
<!DOCTYPE html > specifies that the web page contains HTML code.

Head and Body

- HTML5 does not require the use of the <html>, <head>, and <body> elements. However, most web authors continue to use them.
- The <html> element is sometimes called the **root element** as it contains all the other HTML elements in the document.
- It has a **'lang'** attribute - This attribute tells the browser the natural language that is being used for textual content in the HTML document. Eg: <head lang = "en">
- HTML pages are divided into **two sections**: the **head** and the **body**, which correspond to the <head> and <body> elements.

The head contains descriptive elements *about* the document, such as its title, any style sheets or JavaScript files it uses, and other types of meta information used by search engines and other programs. The body contains content (both HTML elements and regular text) that will be displayed by the browser.
Eg: <head>

```
    <meta charset = "utf-8" />
    <link rel = "stylesheet" href = "main.css" />
    <script src = "new.js"> </script>
    <title> Student activity </title>
<head>
```

- The <meta> element declares that the character encoding for the document is UTF- 8. Character encoding refers to which character set standard, is used to encode the characters in the document. **UTF-8** is a more complete variable-width encoding system that can encode all 110,000 characters in the Unicode character set.

- The <link> element specifies an external CSS style sheet file by name 'main.css' is used with this document. The style sheets is used to define the visual display of the HTML elements in the document. The styles can also be included within the document.

- The <script> element references an external JavaScript file. The JavaScript code can also be written directly within the HTML document.

- The <title> element is used to provide a broad description of the content. It is displayed by the browser in its window and/or tab.

Uses of <title> element –

1. To provide a broad description of content
2. Used by the browser for its bookmarks
3. Used by the browser to store the history list
4. Search engines use it as the linked text in their result pages.

HTML Syntax

Elements and Attributes

- HTML documents are composed of textual content and HTML elements.
- The term **HTML element** is often used interchangeably with the term **tag**.
- However, an HTML element consists of the element name within angle brackets (i.e., the tag) and the content within the tag.
- A tag consists of the element name within angle brackets.
- The element name appears in both the
- beginning tag and the closing tag.
- The closing tag contains a forward slash followed by the element name, all enclosed within angle brackets.

In the above example, <p> is the tag and "Hello I am a paragraph" is the content. HTML elements can also contain attributes.

- An **HTML attribute** is a 'name = value' pair that provides more information about the HTML element. In the above example, style is an attribute.

- An element which does not contain any text or image content is called an **empty element.** It is an instruction to the browser to do something.

Eg: <img> (image element) is an empty element.

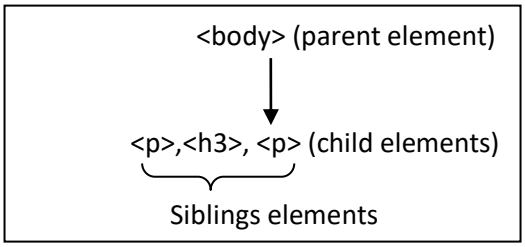
- The empty elements are terminated by a trailing slash.

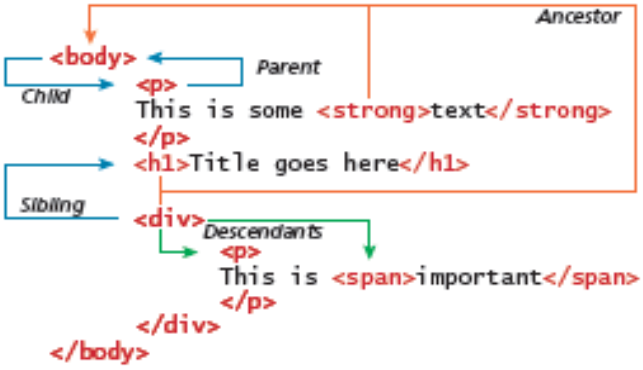
Eg: -------- empty element

Nesting HTML Elements

- An HTML element may contain other HTML elements. The container element is said to be a parent of the contained element (child).
- Any elements contained within the child are said to be **descendants** of the parent element.
- Any given child element, may have a variety of **ancestors**

```
<body>
    <p> This is some big <b> text </b>
     </p>
    <h3>A small title here </h3>
    <p> This is <span> important </span>
    </p>
</body>
```





- Each HTML element must be nested properly. That is, a child’s ending tag must occur before its parent’s ending tag. As shown in the above example, tag must be closed first and then the <p> tag is closed.

Semantic Markup

- HTML documents should focus on the structure of the document. Information about how the content should look when it is displayed in the browser is taken care by CSS(Cascading Style Sheets).
- HTML document should not describe how to visually present content, but only describe its content's structural semantics or meaning.
- Structure is a vital way of communicating information in paper and electronic documents. It makes it easier for the reader to quickly grasp the hierarchy of importance as well as broad meaning of information in the document.

Importance of writing HTML markup and its advantages:

- **Maintainability**. Semantic markup is easier to update and change than web pages that contain a great deal of presentation markup. More time is spent maintaining and modifying existing code than in writing the original code.
- **Faster**. Semantic web pages are typically quicker and faster to download.
- **Accessibility**. Not all web users are able to view the content on web pages. Users with sight disabilities experience the web using voice reading software. Visiting a web page using voice reading software can be a very frustrating experience if the site does not use semantic markup.
- **Search engine optimization**. The most important users of a website are the various search engine crawlers. These crawlers are automated programs that cross the web scanning sites for their content, which is then used for users' search queries. Semantic markup provides better instructions for these crawlers.

Some basic HTML Elements

HTML5 contains many structural and presentation elements, few elements are explained below -

Headings

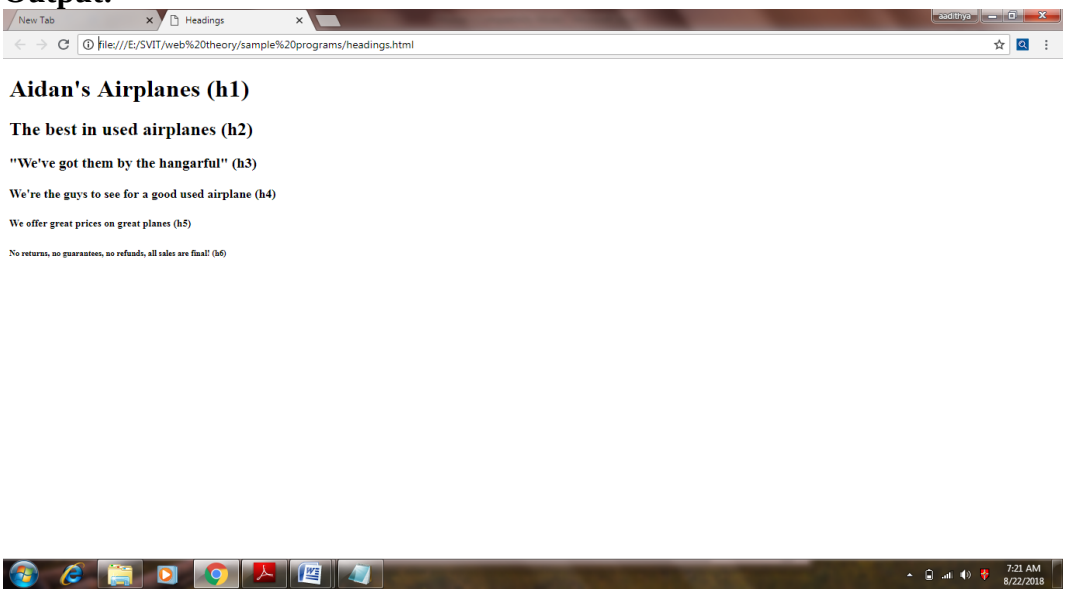
- Text is often separated into sections in documents by beginning each section with a heading.
- Larger sections sometimes have headings that appear more prominent than headings for sections nested inside them.
- In XHTML, there are six levels of headings, specified by the tags <h1>, <h2>, <h3>, <h4>, <h5>, and <h6>, where <h1> specifies the **highest-level heading**.
- Headings are usually displayed in a **boldface font** whose default size depends on the number in the heading tag.
- On most browsers, <h1>, <h2>, and <h3> use font sizes that are larger than that of the default size of text, <h4> uses the default size, and <h5> and <h6> use smaller sizes.

- The heading tags always break the current line, so their content always appears on a new line.

The following example illustrates the use of headings:

```
<!DOCTYPE html>
<!-- headings.html An example to illustrate headings-->
<head> <title> Headings </title>
</head>
<body>
        <h1> Aidan's Airplanes (h1) </h1>
        <h2> The best in used airplanes (h2) </h2>
        <h3> "We've got them by the hangarful" (h3) </h3>
        <h4> We're the guys to see for a good used airplane (h4) </h4>
        <h5> We offer great prices on great planes (h5) </h5>
        <h6> No returns, no guarantees, no refunds, all sales are final! (h6) </h6>
</body>
</html>
```

Output:

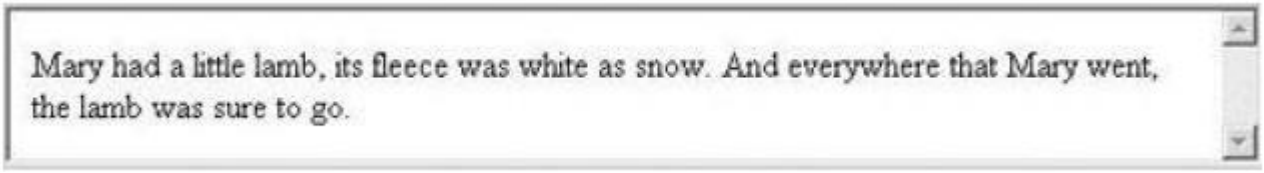


Paragraphs and Divisions

- Text is normally organized into paragraphs in the body of a document. T
- he HTML standard does not allow text to be placed directly in a document body. Instead, textual paragraphs appear as the content of a paragraph element, specified with the tag <p>.
- The <p> tag leaves a line before and after the tag. In displaying the content of a paragraph, the browser puts as many words as will fit on the lines in the browser window. The browser supplies a line break at the end of each line.

For example, the following paragraph might be displayed by a browser as shown below.

```
<p>
Mary had
        a
        little lamb, its fleece was white as
snow. And everywhere that
                Mary went, the lamb
was sure to go.
</p>
```



Notice that multiple spaces in the source paragraph element are replaced by single Spaces. If the content of a paragraph tag is displayed at a position other than the beginning of the line, the browser breaks the current line and inserts a blank line. For example, the following line would be displayed as shown below.

```
<p> Mary had a little lamb, </p> <p> its
        fleece was white as snow. </p>
```

The <p> tag is a container and can contain HTML and other **inline HTML elements** (the and <a> elements). Inline HTML elements are elements that do not cause a paragraph break but are part of the regular flow of the text.
The element is also a container element and is used to create a logical grouping of content

Links (anchor tag <a>)

Links are an essential feature of all web pages. Links are created using the <a> element (the "a" stands for anchor).
<a> is an inline tag. A link has two main parts: the destination and the label. The label of a link can be text or another HTML element such as an *image*.

destination label

 click here

<img src = “image1.jpg”>

- A link specifies the address of the destination. Such an address might be a file name, a directory path and a file name, or a complete URL.
- The document whose address is specified in a link is called the target of that link.
- The value assigned to href (hypertext reference) specifies the target of the link. If the target is in another document in the same directory, the target is just the document’s file name.

If the target file is in a subdirectory named ‘lab’, then the access to file is done by specifying the directory name. < a href = “lab/next.html”> clickhere.

```
<!DOCTYPE html >
<html>
<head>
    <title >sdfsfdfddddd</title>
</head>
<body>
    <p> Mary had a little
    <a href="lab/lamb.jpg"> lamb,</a>
    </p> <p> its fleece was white as snow. </p>
</body>
</html>
```

Output:

Mary had a little lamb,
Its fleece was white as snow.

Anchor element <a> can be used to create a wide range of links. These include:

- Links to external file (or to individual resources such as images or movies on an external site).
- Links to other pages or resources within the current file.
- Links to other places within the current page.

- Links to particular locations on another page (whether on the same site or on an external site).
- Links that are instructions to the browser to start the user's email program.
- Links that are instructions to the browser to execute a JavaScript function.
- Links that are instructions to the mobile browser to make a phone call.

Targets within Documents

If the target of a link is not at the beginning of a document, it must be some element within the document, in which case there must be some means of specifying it. The target element can include an id attribute, which can then be used to identify it in an href attribute. Consider the following example:

<h2 id = "avionics"> Avionics </h2>

Nearly all elements can include an id attribute. The value of an id attribute must be unique within the document. If the target is in the same document as the link, the target is specified in the href attribute value by preceding the id value with a pound sign (#), as in the following example:

* What about avionics? *

When the What about avionics? link is taken; the browser moves the display so that the h2 element whose id is *avionics* is at the top.

When the target is a part or fragment of another document, the name of the part is specified at the end of the URL, separated by a pound sign (#), as in this example:

* Avionics *

One common use of links to parts of the same document is to provide a table of contents in which each entry has a link. This technique provides a convenient way for the user to get to the various parts of the document simply and quickly

Link to external site

```
<a href="http://www.centralpark.com">Central Park</a>
```

Link to resource on external site

```
<a href="http://www.centralpark.com/logo.gif">Central Park</a>
```

Link to another page on same site as this page

```
<a href="index.html">Home</a>
```

Link to another place on the same page

```
<a href="#top">Go to Top of Document</a>
...
<a name="top">
```

Defines anchor for a link to another place on same page

Link to specific place on another page

```
<a href="productX.html#reviews">Reviews for product X</a>
```

Link to email

```
<a href="mailto:person@somewhere.com">Someone</a>
```

Link to JavaScript function

```
<a href="javascript:OpenAnnoyingPopup();">See This</a>
```

Link to telephone (automatically dials the number when user clicks on it using a smartphone browser)

```
<a href="tel:+18009220579">Call toll free (800) 922-0579</a>
```

URL Relative Referencing

- To construct links with the <a> element, reference images with the <img> element, or include external JavaScript or CSS files, the files should be successfully referred using **relative referencing** from the document.
- If the referred file is an external file then **absolute reference** is required. Absolute path contains the full path including the domain name, any paths, and then finally the file name of the desired resource.
- However, when referencing a resource that is on the same server as the HTML document, then **relative referencing** is done. Relative paths change depending upon where the links are present.

There are several rules to create a link using the relative path:

- links in the same directory as the current page have no path information listed **filename**
- sub-directories are listed without any preceding slashes **weekly/filename**
- links up one directory are listed as **../filename**

Sl. No.	Relative Link Type	Example
1	**Same Directory** – To link to a file within the same folder, simply use the file name.	<a href = "ex.html">
2	**Child Directory** – To link to a file within a subdirectory, use the name of subdirectory and a slash before the file name.	<a href = "images/ex.html">
3	**Grandchild/Descendant Directory** – To link to a file that is multiple subdirectories below the current one, construct the full path by including each subdirectory name before the file name.	<a href = "css/images/ex.html">
4	**Parent/Ancestor Directory** – use "../" to reference a folder above the current one. If trying to reference a file several levels above the current one, simply string together multiple "../".	<a href = "../ex.html"> <a href = "../../ex.html">
5	**Sibling Directory** – use "../" to move up to the appropriate level, and then use the same technique as for child or grandchild directories.	<a href = "../ex.html"> <a href = "../images/ex.html">
6	**Root Reference –** An alternative approach for ancestor and sibling references is to use the root reference approach. Ie. Begin the reference with the root reference ("/") and then use the same technique as for child or grandchild directories.	<a href = "/ex.html"> <a href = "/images/ex.html">
7	**Default Document** -Web servers allow references to directory names without file names. In such a case, the web server will serve the default document.	<a href = "members"> or <a href = "/members">

Inline Text Elements

Few HTML elements do not disrupt the flow of text (i.e., do not cause a line break), they are called inline elements. Eg: ,<u>,<i>,<b>, <a>, , <time>, <small> elements).

Element	Description
<a>	Anchor used for hyperlinks.
<abbr>	An abbreviation
 	Line break
<cite>	Citation (i.e., a reference to another work).
<code>	Used for displaying code, such as markup or programming code.
	Emphasis
<mark>	For displaying highlighted text
<small>	For displaying the fine-print, i.e., "non-vital" text, such as copyright or legal notices.
	The inline equivalent of the element. It is generally used to mark text that will receive special formatting using CSS.
	For content that is strongly important.
<time>	For displaying time and date data

Images

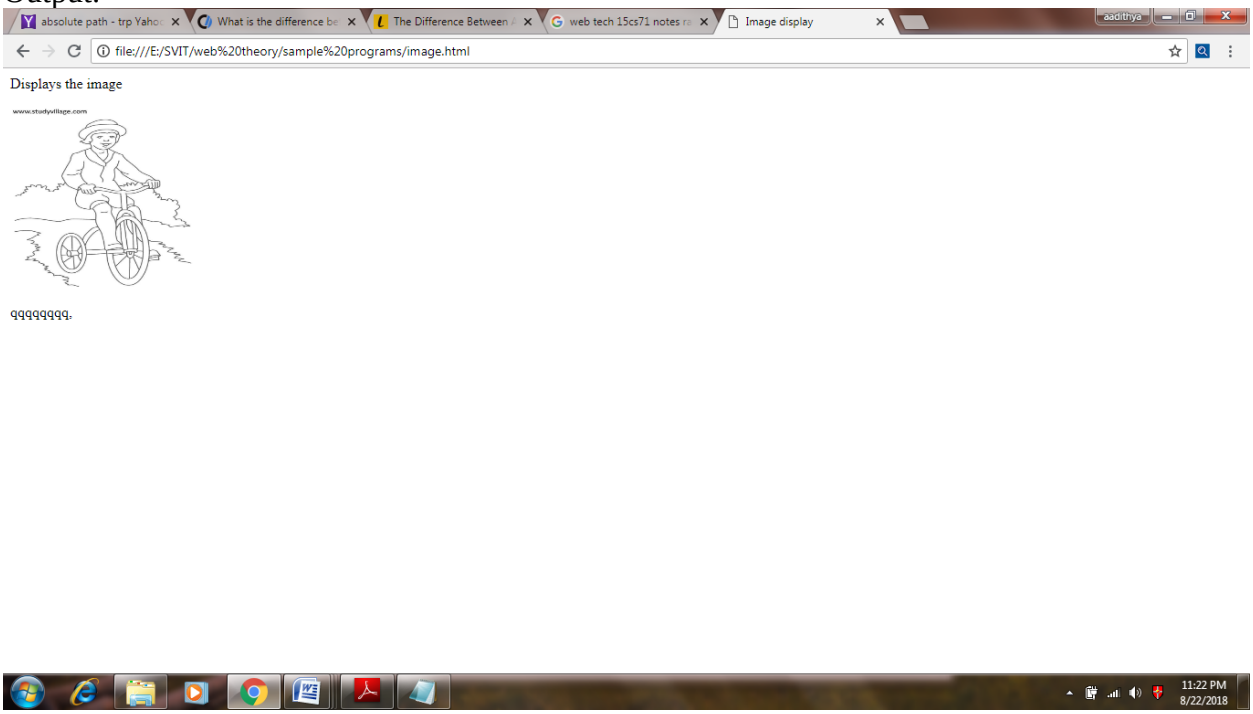
The <img> tag is the oldest method for displaying an image.

src attribute - specifies the file containing the image;
alt - specifies text to be displayed when it is not possible to display the image.
title – specifies the title to be displayed in pop-up tool tip when user moves mouse over image.
width and height - can be included to specify (in pixels) the size of the rectangle for the image

Eg:

```
<html>
<head>
<head> <title>Image display</title>
</head>
<body>
<p >Displays the image</p>
<img  src="cycle-riding.jpg" width="200" height="200">
<p >qqqqqqqq,</p>
</body>
</html>
```

Output:



Character entities

- These are special characters for symbols for which there is either no easy way to type them via a
- keyboard (such as the copyright symbol or accented characters) or which have a reserved meaning in HTML (for instance the "<" or ">" symbols).
- There are many HTML character entities. They can be used in an HTML document by using the entity name or the entity number.

Entity Name	Entity Number	Description
		Nonbreakable space. The browser ignores multiple spaces in the source HTML file. If you need to display multiple spaces, you can do so using the nonbreakable space entity.
<	<	Less than symbol ("<").
>	>	Greater than symbol (">").
©	©	The © copyright symbol
€	€	The € euro symbol.
™	™	The ™ trademark symbol.
ü	ü	The ü—i.e., small u with umlaut mark.

&amp;	Ampersand
&quot;	Double quote
&apos;	Single quote (apostrophe)
&deg;	Degree

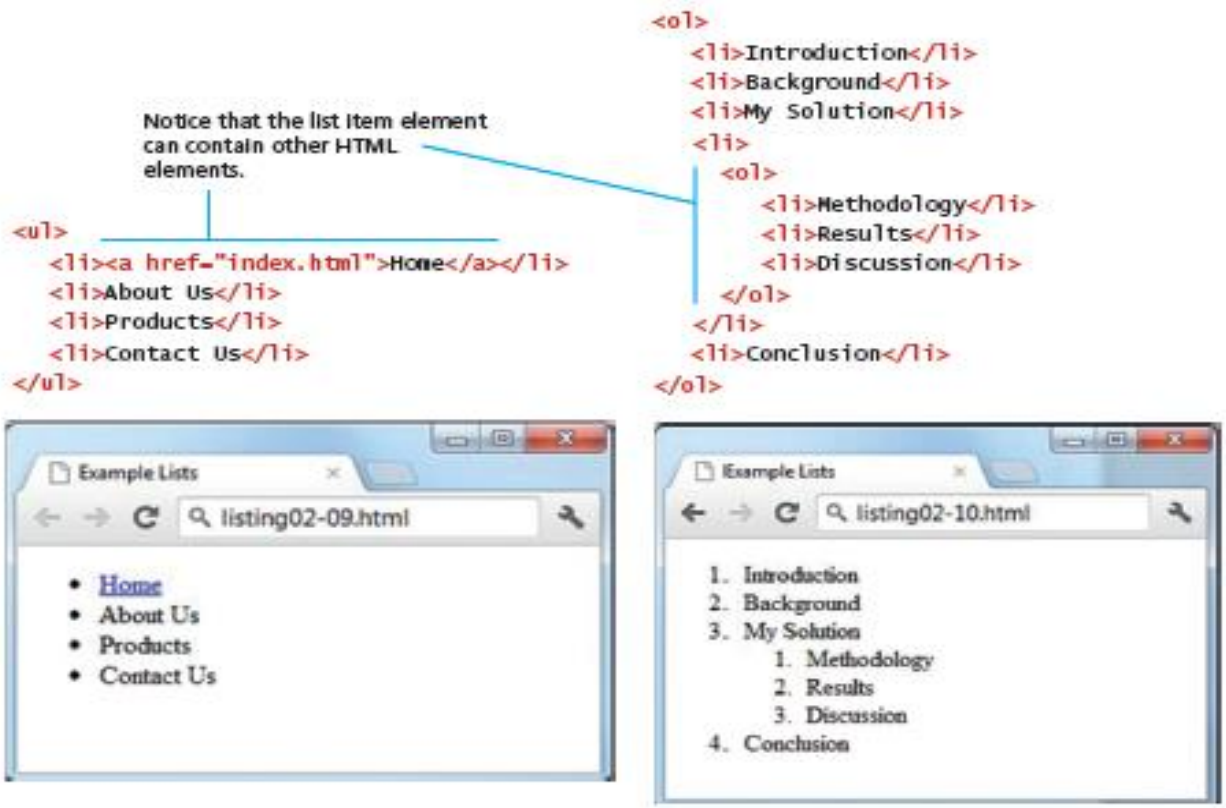
Lists

HTML provides simple and effective ways to specify lists in documents.
There are three types of lists:

■ **Unordered lists**. Collections of items in no particular order; these are by default rendered by the browser as a bulleted list. However, it is common in CSS to style unordered lists without the bullets. Unordered lists have become the conventional way to markup navigational menus.

■ **Ordered lists**. Collections of items that have a set order; these are by default rendered by the browser as a numbered list.

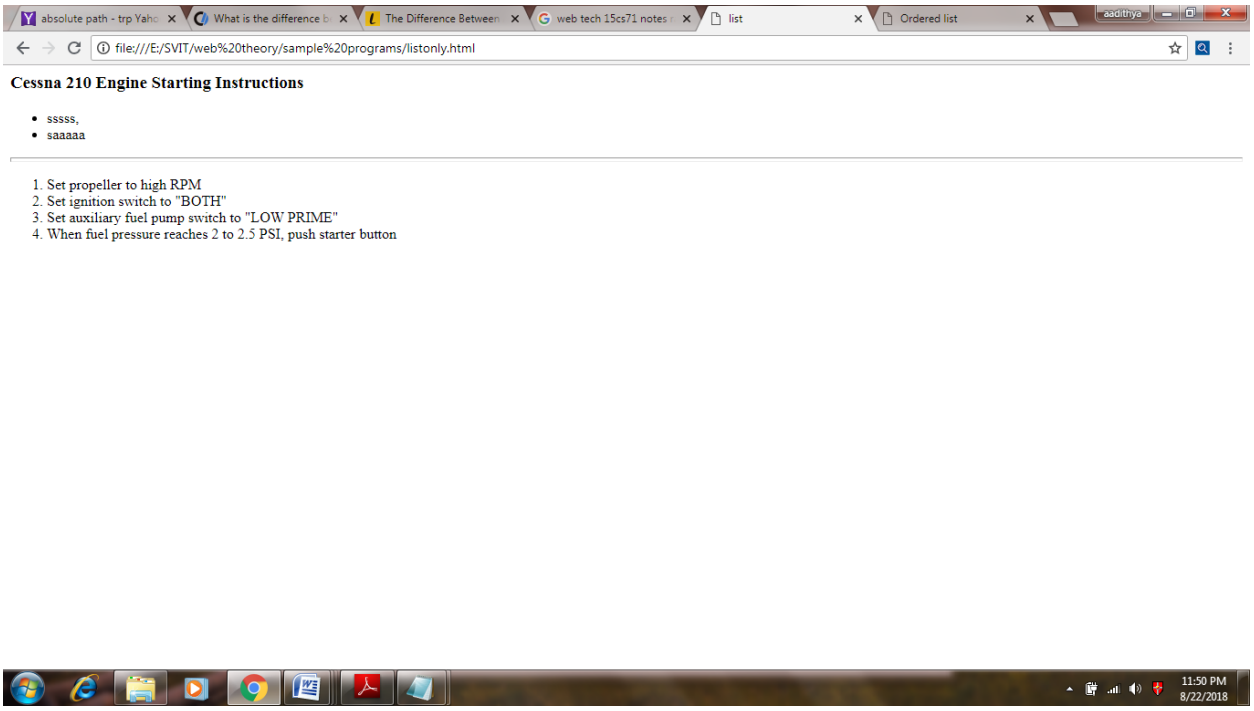
■ **Definition lists**. Collection of name and definition pairs. These tend to be used infrequently. Perhaps the most common example would be a FAQ list.




```
<html >
<head> <title> list </title>
</head>
<body>
    <h3> Cessna 210 Engine Starting Instructions </h3>
    <ul >
        <li>sssss,</li>
        <li>saaaaa</li>
```

```
    </ul>
    <hr size="5"  />
    <ol >
        <li> Set propeller to high RPM </li>
        <li> Set ignition switch to "BOTH" </li>
        <li> Set auxiliary fuel pump switch to "LOW PRIME" </li>
        <li> When fuel pressure reaches 2 to 2.5 PSI, push starter button </li>
    </ol>
</body>
</html>
```

Output:



HTML5 Semantic Structure Elements

- The different sections of a website are usually divided by using tag.
- Most complex websites are packed solid with elements. Most of these are marked with different id or class attributes, that are styled by using various CSS.
- But many elements make the markup confusing and hard to modify.
- Developers typically give suitable names to id or class of , so as to provide some clue to understand what that part of code means. The new elements help the bots to view sites and the codes more like people.
- The idea behind using new semantic block structuring elements in HTML5 is to make it easier to understand, what a particular part of document does.

- Some of tag is replaced with self-explanatory HTML5 elements. There is no predefined presentation for these new tags. The new semantic elements in HTML5 are listed below -
  1. Headers and Footer
  2. Heading Groups
  3. Navigation
  4. Articles and Sections
  5. Figure and Figure Captions
  6. Aside

Header and Footer

Most website pages have a recognizable header and footer section. Typically the header contains the site logo and title (and perhaps additional subtitles or taglines), horizontal navigation links, and perhaps one or two horizontal banners. The typical footer contains less important material, such as smaller text versions of the navigation, copyright notices, information about the site's privacy policy, and perhaps twitter feeds or links to other social sites.

Both the HTML5 <header> and <footer> element can be used not only for *page* headers and footers, but also for header and footer elements within other HTML5 containers, such as <article> or <section>.

```
<header>
<img src="logo.gif" alt="logo" />
<h1>Fundamentals of Web Development</h1>
...
</header>
<article>
   <header>
      <h2>HTML5 Semantic Structure Elements</h2>
      <p>By <em>Randy Connolly</em></p>
      <p><time>September 30, 2015</time></p>
   </header>
   ...
</article>
```

Heading Groups

A header may contain multiple headings <hgroup> element is usually used in such cases. The <hgroup> element can be used in contexts other than a header. For instance, one could also use an <hgroup> within an <article> or a <section> element. The <hgroup> element can *only* contain <h1>, <h2>, etc., elements.

```
<header>
   <hgroup>
   <h1>Chapter Two: HTML 1</h1>
   <h2>An Introduction</h2>
   </hgroup>
</header>
<article>
   <hgroup>
   <h2>HTML5 Semantic Structure Elements</h2>
   <h3>Overview</h3>
   </hgroup>
</article>
```

Navigation

The <nav> element represents a section of a page that contains links to other pages or to other parts within the same page. Like the other new HTML5 semantic elements, the browser does not apply any special presentation to the <nav> element. The <nav> element was intended to be used for major navigation blocks. However, like all the new HTML5 semantic elements, from the browser's perspective, there is no definite right or wrong way to use the <nav> element. Its sole purpose is to make the document easier to understand.

```
<header>
  <img src="logo.gif" alt="logo" />
  <h1>Fundamentals of Web Development</h1>
  <nav role="navigation">
    <ul>
       <li><a href="index.html">Home</a></li>
       <li><a href="about.html">About Us</a></li>
       <li><a href="browse.html">Browse</a></li>
    </ul>
  </nav>
</header>
```

Articles and Sections

The new HTML5 semantic elements <section> and <article> are used to group the tags. Suppose a book is divided into smaller blocks of content called chapters, which makes the book easier to read. If each chapter is further divided into sections (and these sections into even smaller subsections), this makes the content of the book easier to manage for both the reader and the authors.

The article element represents a section of content that forms an independent part of a document or site. The section element represents a section of a document, typically with a title or heading. It is a broader element.

Figure and Figure Captions
Prior to HTML5, web authors typically wrapped images and their related captions within a nonsemantic element. In HTML5 we can instead use the <figure> and <figcaption> elements. *The figure element represents some flow content, optionally with a caption, that is self-contained and is typically referenced as a single unit from the main flow of the document.*

```
<p>This photo was taken on October 22, 2011 with a Canon EOS 30D camera.</p>
<figure>
    <img src="images/central-park.jpg" alt="Central Park" /><br/>
    <figcaption>Conservatory Pond in Central Park</figcaption>
</figure>
</p>
```

The above tags illustrates a sample usage of the <figure> and <figcaption> element.

Aside
The <aside> element is similar to the <figure> element, the <aside> element "represents a section of a page that consists of content that is indirectly related to the content around the aside element". The <aside> element is be used for sidebars, pull quotes, groups of advertising images, or any other grouping of non-essential elements.

Cascading Style Sheets (CSS)

- CSS is a W3C standard for describing the appearance of HTML elements. It is used to define the **presentation** of HTML documents.
- Using CSS with HTML elements, we can assign font properties, colors, sizes, borders, background images, and even position elements on the page.
- In early 1990s, a variety of different style sheet standards were proposed, including JavaScript style sheets, which was proposed by Netscape in 1996.
- Netscape's proposal was one that required the use of JavaScript programming to perform style changes. Due to many nonprogrammers everywhere, the W3C decided to adopt CSS, and by the end of 1996 the CSS Level 1 Recommendation was published. The latest version is CSS Level 4.
- CSS can be added directly to any HTML element (via the style attribute), or within the <head> element, or in a separate text file that contains only CSS.

Benefits of CSS
■ **Improved control over formatting -** CSS gives website developers fine-grained control over the appearance of their web content. The contents are better formatted using CSS.

■ **Improved site maintainability -** Websites is easily maintainable, as all formatting can be put in a single CSS file.

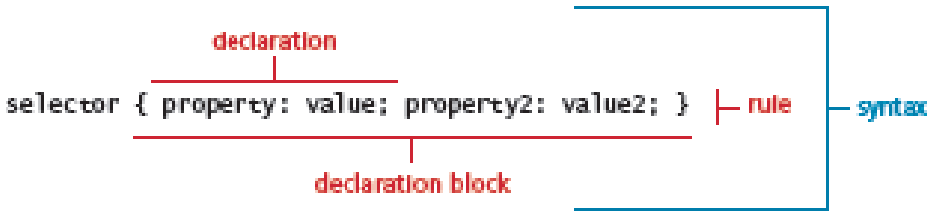
■ **Improved accessibility -** CSS-driven sites are more accessible.

■ **Improved page download speed -** A web site that uses a single CSS files will be quicker to download, because each individual HTML file will contain less style information and markup, and thus be smaller.

■ **Improved output flexibility -** CSS can be used to adopt a page for different output devices with varying sizes. This approach to CSS page design is often referred to as **responsive design**.

CSS Syntax

- A CSS rule consists of a **selector** that identifies the HTML elements that will be affected, followed by a series of **property:value pairs** (each pair is also called a **declaration**), as shown in Figure below.
- The series of declarations is called the **declaration block**. A declaration block can be together on a single line, or spread across multiple lines.
- The browser ignores white space (i.e., spaces, tabs, and returns) between your CSS rules so you can format the CSS however you want. Each declaration is terminated with a semicolon. The semicolon for the last declaration in a block is in fact optional.



Eg:
p{color: red; font-weight:bold;} =>here p is the selector; the declarations are implied to the p tag.

h2,p { font-size:40; color:green;}=> the declarations are implied to both h2 & p tags.

Selectors
Every CSS rule begins with a **selector**. The selector identifies which element or elements in the HTML document will be affected by the declarations in the rule. They are a pattern that is used by the browser to select the HTML elements that will receive the style.

Properties
Each individual CSS declaration must contain a property. These property names are predefined by the CSS standard. The CSS recommendation defines over a hundred different property names. Eg: font-family, font-size, font-style, font-weight, text-align, text-decoration, background-color, background-image, border-color, border-width, border-style, border-top, etc.

Values
Each CSS declaration also contains a value for a property. The unit of any given value is dependent upon the property. Most of the property values are from a predefined list of keywords. Eg: px, em, %, In, cm, mm etc. are some of the units for measuring values.

Location of Styles (Levels of Stylesheets)

CSS style rules can be located in three different locations, in order from lowest level to highest level, are **inline, document level, and external**.

Inline Styles

Inline styles are style rules placed within an HTML element using the style attribute, as shown below. An inline style only affects the element it is defined within and overrides any other style definitions. Selector is not necessary with inline styles and that semicolons are only required for separating multiple rules.

Disadvantages of using inline style-
Style is applied to an element only
Maintaining the inline style is difficult

The advantage of using inline style is that it can be quickly tested for a style change.

Eg: <h2 style = “font-size:24pt;”> Description</h2>
<h2 style = “font-size:24pt; font-weight:bold;”> Reviews </h2>

Embedded Style Sheet (Document Level/Internal)

Embedded style sheets (also called **internal styles or document level styles**) are style rules placed within the <style> element (inside the <head> element of an HTML document) and apply to the whole body of the document.

The disadvantage of using embedded styles is that it is difficult to consistently style multiple documents when using embedded styles. But it is helpful when quickly testing out a style that is used in multiple places within a single HTML document. Spaces are ignored in <style> element.

```
<head>
<title>Student Data</title>
<style>
h1 { font-size: 24pt; }
h2 {
font-size: 18pt;
font-weight: bold;
}
</style>
</head>
<body>
<h1>Student count</h1>
<h2>CSE/ISE Department</h2>
……..
</body>
```

External Style Sheet

External style sheets are style rules placed within an external text file with the .**css** extension. This style provides the best maintainability. When you make a change to an external style sheet, all HTML documents that reference that style sheet will automatically use the updated version.

To reference an external style sheet, you must use a <link> element (within the <head> element). Several style sheets can be linked at a same time. Each linked style sheet will require its own <link> element.

```
<head >
<title>Share Your Travels -- New York - Central Park</title>
<link rel="stylesheet" href="styles.css" />
</head>
```

Selectors

The selector identifies which element or elements in the HTML document will be affected by the declarations in the rule. They are a pattern that is used by the browser to select the HTML elements that will receive the style.

- Element Selectors
- Class Selectors
- Id Selectors
- Attribute Selectors
- Pseudo-Element and Pseudo-Class Selectors
- Contextual Selectors

Element Selectors

Element selectors select an element or group of elements of the HTML document, and the properties are applied on it.

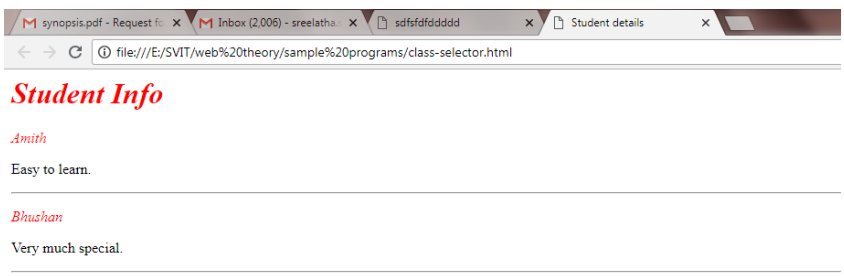
The group of elements are separated using commas is called **grouped selector**.

Universal element selector - All elements of the document can be selected by using the * (asterisk) character.

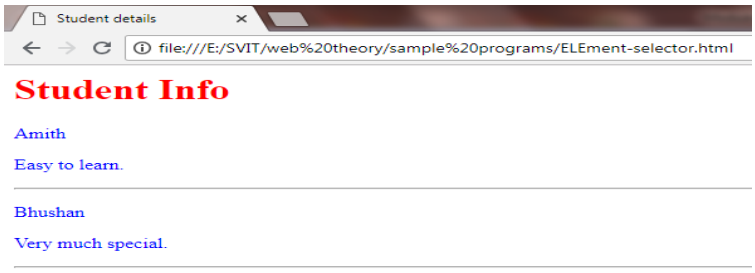
Eg of element selector -

```
<head>
<title>Student details </title>
<style>
*{ color:blue;}
h1 {
```

Output:



```
color: red;
}
</style>
</head>
<body>
<h1 >Student Info</h1>
<div>
<p >Amith</p>
<p>Easy to learn.</p>
</div>
<hr/>
<div>
<p >Bhushan</p>
<p>Very much special.</p>
</div>
<hr/>
</body>
```



Eg of grouped selector

```
p, div, h3 {
margin: 0;
padding: 0;
}
```

Eg of universal element selector

```
*{
    color : red;
  }
```

Class Selectors

A **class selector** allows to simultaneously target different HTML elements. The HTML elements with the same class attribute value, can be styled by using a class selector.
Syntax: period (.)classname{ styles}

Eg:

```
<head>
    <title>Student details </title>
    <style>
    .first {
    font-style: italic;
    color: red;
    }
    </style>
</head>
<body>
     <h1 class="first">Student Info</h1>
     <div>
     <p class="first">Amith</p>
```

Output:

```
    <p>Easy to learn.</p>
    </div>
    <hr/>
    <div>
    <p class="first">Bhushan</p>
    <p>Very much special.</p>
  </div>
  <hr/>
</body>
```

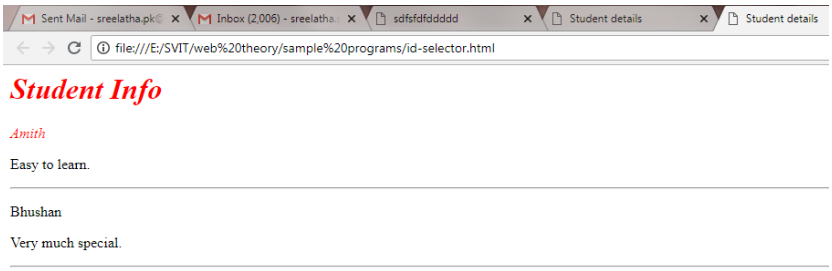
Id Selectors

An **id selector** allows to assign style to a specific element by its id attribute.
Syntax: hash (#)id name

Eg:

```
<head>
    <title>Student details </title>
    <style>
    #first {
    font-style: italic;
    color: red;
    }
    </style>
</head>
<body>
    <h1 id="first">Student Info</h1>
    <div>
    <p id="first">Amith</p>
    <p>Easy to learn.</p>
    </div>
    <hr/>
    <div>
    <p >Bhushan</p>
    <p>Very much special.</p>
    </div>
    <hr/>
</body>
```

Output:



Attribute Selectors

An **attribute selector** provides a way to select HTML elements either by the presence of an element attribute or by the value of an attribute.
Eg: [src], [src$=".jpg"] , a[href*="gala"] etc.

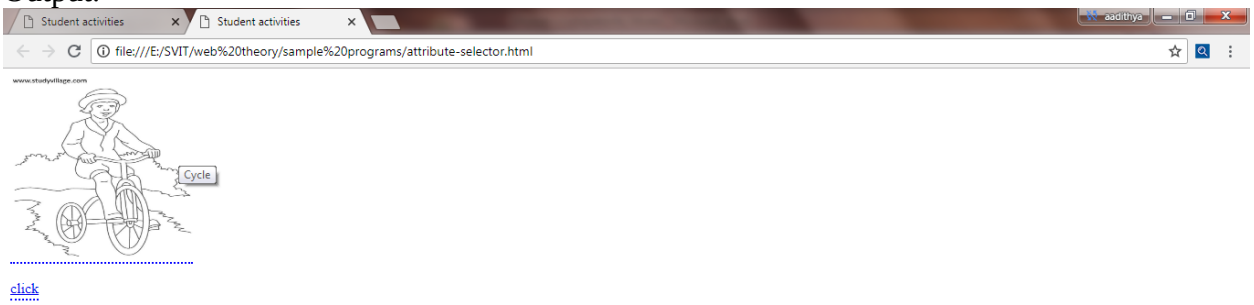
[src] – selects all the elements which have 'src' as an attribute
[src$=".jpg"] – selects all the elements with 'src' value ending with .jpg
a[href*="gala"] – selects <a> tag with 'href' value having text 'gala'.

Attribute selectors is very helpful technique in the styling of hyperlinks and images. Suppose, we want special attention of user when a pop-up tooltip is available for a link or image. This can be done by using the following attribute selector:
[title] { … }

Eg:

```
<head >
    <title>Student activities</title>
    <style>
    [title] {
    cursor: help;
    padding-bottom: 3px;
    border-bottom: 2px dotted blue;
    }
    </style>
</head>
<body>
<div>
<img src="cycling.jpg" title="Cycle" />
<a href = “s1.jpg” title= “link to photo”> click </a>
</div>
</body>
```

Output:



Selector	Matches	Example
[]	A specific attribute.	`[title]` Matches any element with a title attribute
[=]	A specific attribute with a specific value.	`a[title="posts from this country"]` Matches any <a> element whose title attribute is exactly `"posts from this country"`
[~=]	A specific attribute whose value matches at least one of the words in a space-delimited list of words.	`[title~="Countries"]` Matches any `title` attribute that contains the word `"Countries"`
[^=]	A specific attribute whose value begins with a specified value.	`a[href^="mailto"]` Matches any <a> element whose `href` attribute begins with `"mailto"`
[*=]	A specific attribute whose value contains a substring.	`img[src*="flag"]` Matches any <img> element whose `src` attribute contains somewhere within it the text `"flag"`
[$=]	A specific attribute whose value ends with a specified value.	`a[href$=".pdf"]` Matches any <a> element whose `href` attribute ends with the text ".pdf"

Pseudo-Element and Pseudo-Class Selectors

A **pseudo-element selector** is a way to select something that does not exist explicitly as an element in the HTML document but which is still a recognizable selectable object.
For instance, first line or first letter of any HTML element.

Selector	Type	Description
`a:link`	pseudo-class	Selects links that have not been visited
`a:visited`	pseudo-class	Selects links that have been visited
`:focus`	pseudo-class	Selects elements (such as text boxes or list boxes) that have the input focus.
`:hover`	pseudo-class	Selects elements that the mouse pointer is currently above.
`:active`	pseudo-class	Selects an element that is being activated by the user. A typical example is a link that is being clicked.
`:checked`	pseudo-class	Selects a form element that is currently checked. A typical example might be a radio button or a check box.
`:first-child`	pseudo-class	Selects an element that is the first child of its parent. A common use is to provide different styling to the first element in a list.
`:first-letter`	pseudo-element	Selects the first letter of an element. Useful for adding drop-caps to a paragraph.
`:first-line`	pseudo-element	Selects the first line of an element.

A **pseudo-class selector** does apply to an HTML element, but targets a particular state.

The most common use of this type of selectors is for targeting link states. By default, the browser displays link text blue and visited text links purple.

Contextual Selectors

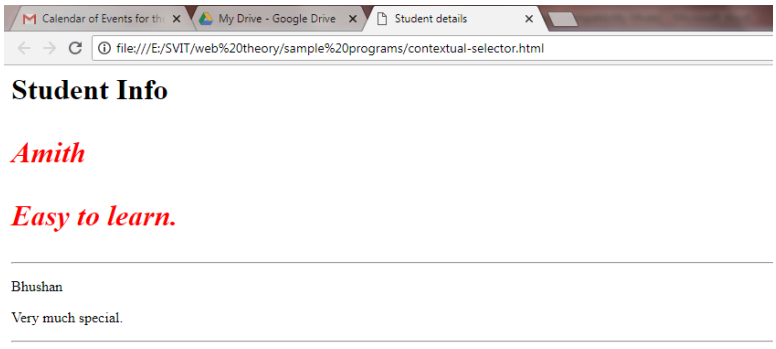
A **contextual selector** (in CSS3 also called **combinators**) allows to select elements based on their *ancestors*, *descendants*, or *siblings*. It selects elements based on their context or relation to other elements in the document tree.

As shown in below table, **descendant selector** matches all elements that are contained within another element. The character used to indicate descendant selection is the space character.

Selector	Matches	Example
Descendant	A specified element that is contained somewhere within another specified element.	`div p` Selects a <p> element that is contained somewhere within a element. That is, the <p> can be any descendant, not just a child.
Child	A specified element that is a direct child of the specified element.	`div>h2` Selects an <h2> element that is a child of a element.
Adjacent sibling	A specified element that is the next sibling (i.e., comes directly after) of the specified element.	`h3+p` Selects the first <p> after any <h3>.
General sibling	A specified element that shares the same parent as the specified element.	`h3~p` Selects all the <p> elements that share the same parent as the <h3>.

```
<head>
    <title>Student details </title>
    <style>
    #first  p{
    font-style: italic;
    color: red;
    }
    </style>
</head>
<body>
    <h1 id="first">Student Info
    <div>
    <p >Amith</p>
    <p>Easy to learn.</p>
    </div>
    </h1>
    <hr/>
    <div>
    <p >Bhushan</p>
```

Output:



```
    <p>Very much special.</p>
    </div>
    <hr/>
</body>
```

The Cascade: How Styles Interact

Multiple CSS rules can be defined for the same HTML element, at different locations – inline, embedded or external. The browser determines the style to be applied on an element, depending on the location and hierarchy of the html element.
The "Cascade" in CSS refers to how conflicting rules are handled. CSS uses the following cascade principles to help it deal with conflicts: inheritance, specificity, and location.

Inheritance
Inheritance is the first of these cascading principles. Many (but not all) CSS properties affect not only themselves but their descendants as well. Font, color, list, and text properties are inheritable; layout, sizing, border, background, and spacing properties are not inheritable.

If suppose, this is a document,

```
<head>
<style>
    body {
    font-family: Arial;
    color: red;
    border: 8pt solid green;
    margin: 100px;
    }

    div {
    font-weight: bold;
    }
</style>
</head>
<body>
<div>Will be displayed in red, with arial font and bold</div>
</body>
```

The font settings are inherited from the parent tag, border and margin are not inheritable.
However it is possible to tell elements to inherit properties that are normally not inheritable, by explicitly specifying as 'inherit'.

```
    div {
    font-weight: bold;
    border: inherit;
    margin: inherit;
    }
```

Specificity

Specificity is how the browser determines which style rule takes precedence when more than one style rule could be applied to the same element. In CSS, the more specific the selector, the more it takes precedence (i.e., overrides the previous definition)

```
<head>
<style>
    body {
    font-family: Arial;
    color: red;
    border: 8pt solid green;
    margin: 100px;
    }

    div {
    font-weight: bold;
    color: blue;
    }
</style>
</head>
<body>
<div>Will be displayed in blue, with arial font and bold</div>
</body>
```

The content of is displayed in blue, as the red color setting of <body> tag is overridden in the specification of tag.

Location

The principle of location is that when rules have the same specificity, then the latest are given more weight. Ie., an inline style will override one defined in an embedded style sheet and embedded style will override the external style sheet.

Styles defined in external style sheet X will override styles in external style sheet Y if X's <link> element is after Y's in the HTML document.

```
<link rel= "stylesheet" href= "Y">
<link rel= "stylesheet" href= "X">
```

When the same style property is defined multiple times within a single declaration block, the last one will take precedence.

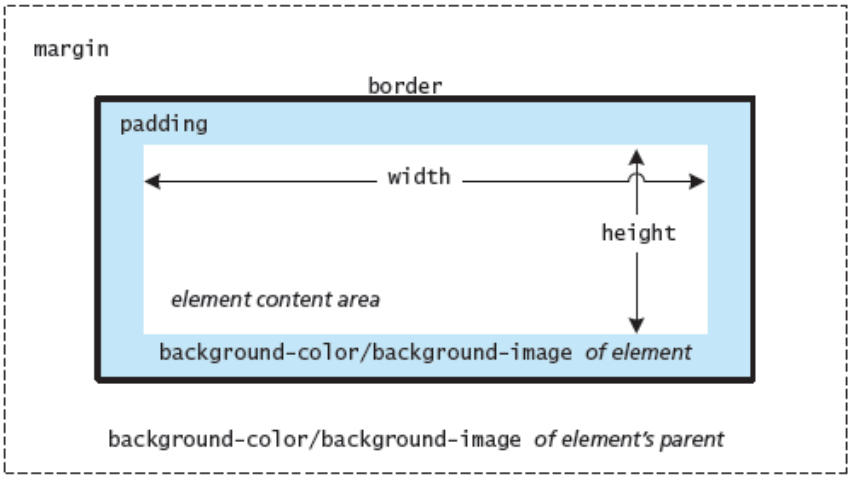
Specificity algorithm:

■ First count 1 if the declaration is from a "style" attribute in the HTML, 0 otherwise (let that value = a).
■ Count the number of ID attributes in the selector (let that value = b).
■ Count the number of class selectors, attribute selectors, and pseudo-classes in the selector (let that value = c).
■ Count the number of element names and pseudo-elements in the selector (let that value = d).
■ Finally, concatenate the four numbers a+b+c+d together to calculate the selector's specificity.

The following sample selectors are given along with their specificity value.
<tag style="color: red"> 1000
body .example 0011
body .example strong 0012
div#first 0101
div#first .error 0111
#footer .twitter a 0111
#footer .twitter a:hover 0121
body aside#left div#cart strong.price 0214

The Box Model

In CSS, all HTML elements exist within a rectangular **element box** shown in Figure.



The background color or image of an element fills an element within its border.
Some of the common background properties are –

Property	Description
background	A combined shorthand property that allows you to set multiple background property.
background-attachment	Specifies whether the background image scrolls with the document (default) or remains fixed. values are: *fixed, scroll*
background-color	Sets the background color of the element.
background-image	Specifies the background image
background-position	Specifies where background image will be placed. Possible values include: *bottom, center, left, and right*. Pixel or percentage numeric position value may also be specified.
background-repeat	Determines whether the background image will be repeated – for a tiled background.

	Possible values are: repeat, repeat-x, repeat-y, and no-repeat
background-size	To modify the size of the background image.

Borders

Borders are used to visually separate elements. Borders are put around all four sides of an element, or just one, two, or three of the sides. Various border properties are –

Property	Description
border	A shorthand property that allows to set the style, width, and color of a border in one property. The order is important and must be: border-style border-width border-color
border-style	Specifies the line type of the border. Possible values are: *solid, dotted, dashed, double, groove, ridge, inset, and outset*.
border-width	The width of the border in a unit(usually in px). A variety of keywords (thin, medium, thick etc.) are also supported.
border-color	The color of the border in a color unit.
border-radius	The radius of a rounded corner.
border-image	The URL of an image to use as a border

Eg: border-style: dotted;
Border-width: 15px;
Border-color:red;

Note: It is possible to set the properties for one or more sides of the element box in a single property, or to set them individually (all sides separate settings) using separate properties.
Eg:
margin-top:30px;
margin-left:10px;
padding-top:10px;
border-top-width:5px;
border-buttom-width:20px;
border-top-style:dashed; etc.

For instance, we can set the side properties individually:

border-top-color: red; / sets just the top side */*
border-right-color: green; / sets just the right side */*
border-bottom-color: yellow; / sets just the bottom side */*

border-left-color: blue; / sets just the left side */*

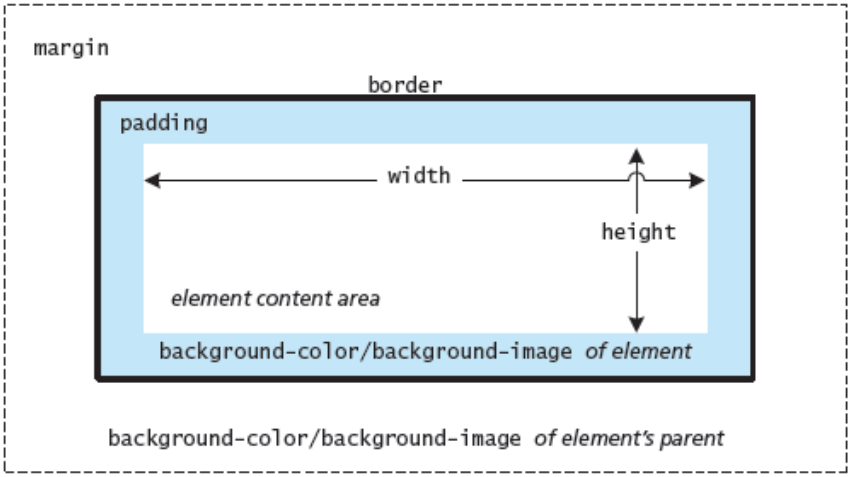
```
border-color: red green orange blue;
```

Alternately, we can set all four sides to a single value via:
border-color: red; / sets all four sides to red */*
Or
border-color: red green orange blue; (mnemonic **TRouBLe)**
When using this multiple values shortcut, they are applied in clockwise order starting at the top. Thus the order is: **top right bottom left .**

Another shortcut is to use just two values; in this case the first value sets top and bottom, while the second sets the right and left.

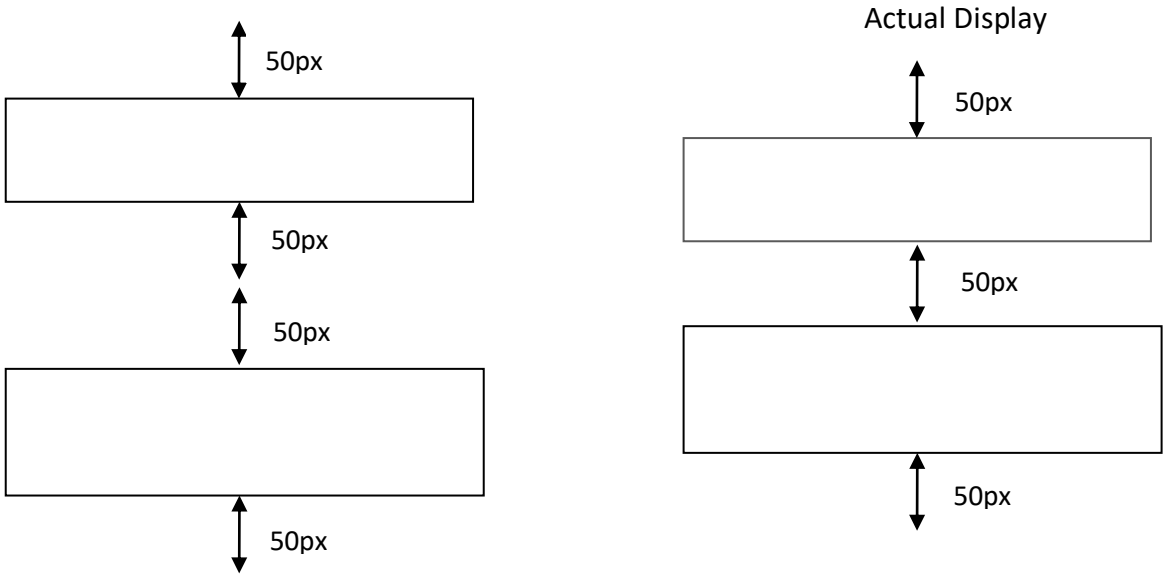
border-color: red yellow; /* *top+bottom=red, right+left=yellow* */

Margins and Padding
Margins and padding are essential properties for adding space in between two elements, which can help differentiate one element from another. Margins add spacing around an element's content, and padding adds spacing within elements. Borders divide the margin area from the padding area.



The adjoining vertical margins collapse with each other, and the margin value of the highest element is chosen between the two elements. Horizontal margins **never** collapse with each other.

If suppose, the margin of a <p> tag is set to 50px. Then the space between 2 paragraphs should be (50px +50px) 100 px. But the vertical margins are collapsed and the space of 50px is displayed.



CSS Text Styling

CSS provides two types of properties that affect text – the font properties and the paragraph properties.

Font Family

- The first of these problems involves specifying the font family. A word processor on a desktop machine can make use of any font that is installed on the computer; browsers are no different. However, just because a given font is available on the web developer's computer, it does not mean that that same font will be available for all users who view the site. For this reason, it is conventional to supply a so-called **web**

- **font stack**, that is, a series of alternate fonts to use in case the original font choice is not on the user's computer.

- One common approach is to make your font stack contain, in this order, the following: *ideal*, *alternative*, *common*, and then *generic*. Take for instance, the following font stack:

font-family { "Hoefler Text", Cambria, "Times New Roman", serif; }

Font Sizes

Another potential problem with web fonts is font sizes. In a print-based program such as a word processor, specifying a font size is unproblematic. Making some text 12 pt will mean that the font's bounding box (which in turn is roughly the size of its characters) will be 1/6 of an inch tall when printed, while making it 72 pt will make it roughly one inch tall when printed.

Property	Description
font	A combined shorthand property that allows you to set the family, style, size, variant, and weight in one property. While you do not have to specify each property, you must include at a minimum the font size and font family. In addtion, the order is important and must be: `style weight variant size font-family`
font-family	Specifies the typeface/font (or generic font family) to use. More than one can be specified.
font-size	The size of the font in one of the measurement units.
font-style	Specifies whether italic, oblique (i.e., skewed by the browser rather than a true italic), or normal.
font-variant	Specifies either small-caps text or none (i.e., regular text).
font-weight	Specifies either normal, bold, bolder, lighter, or a value between 100 and 900 in multiples of 100, where larger number represents weightier (i.e., bolder) text.

Paragraph Properties

Property	Description
letter-spacing	Adjusts the space between letters. Can be the value normal or a length unit.
line-height	Specifies the space between baselines (equivalent to leading in a desktop publishing program). The default value is normal, but can be set to any length unit. Can also be set via the shorthand font property.
list-style-image	Specifies the URL of an image to use as the marker for unordered lists.
list-style-type	Selects the marker type to use for ordered and unordered lists. Often set to none to remove markers when the list is a navigational menu or a input form.
text-align	Aligns the text horizontally in a container element in a similar way as a word processor. Possible values are left, right, center, and justify.
text-decoration	Specifies whether the text will have lines below, through, or over it. Possible values are: none, underline, overline, line-through, and blink. Hyperlinks by default have this property set to underline.
text-direction	Specifies the direction of the text, left-to-right (ltr) or right-to-left (rtl).
text-indent	Indents the first line of a paragraph by a specific amount.
text-shadow	A new CSS3 property that can be used to add a drop shadow to a text. Not yet supported in IE9.
text-transform	Changes the capitalization of text. Possible values are none, capitalize, lowercase, and uppercase.
vertical-align	Aligns the text vertically in a container element. Most common values are: top, bottom, and middle.
word-spacing	Adjusts the space between words. Can be the value normal or a length unit.

Module II
HTML Tables and Forms & Advanced CSS

2.1 Introduction to Tables

- A **table** in HTML is created using the <table> element and can be used to represent information that exists in a two-dimensional grid.
- A table is a matrix of cells. The cells in the top row often contain column labels, those in the leftmost column often contain row labels, and most of the rest of the cells contain the data of the table.
- The content of a cell can be almost any document element, including text, a heading, a horizontal rule, an image, and a nested table.

Basic Table Tags
A table is specified as the content of the block tag <table>.
There are two kinds of lines in tables:
- the line around the whole table is called the border;
- the lines that separate the cells from each other are called rules.

- A table that does not include the border attribute will be a matrix of cells with neither a border nor rules.
- The browser has default widths for table borders and rules, which are used if the border attribute is assigned the value "border." Otherwise, a number can be given as border's value, which specifies the border width in pixels. For example, border = "3" specifies a border 3 pixels wide.
- A border value of "0" specifies no border and no rules. The rules are set at 1 pixel when any nonzero border value is specified. The border attribute is the most common attribute for the <table> tag.

- In most cases, a displayed table is preceded by a title, given as the content of a <caption> tag, which can immediately follow the opening <table> tag.
- The cells of a table are specified one row at a time. Each row of a table is specified with a row tag, **<tr>.**
- Within each row, the row label is specified by the table heading tag, **<th> .** Each data cell of a row is specified with a table data tag, **<td>**

The first row of a table usually has the table's column labels. For example, if a table has three data columns and their column labels are, Apple, Orange, and Screwdriver respectively, the first row can be specified by the following:

```
<tr>
<th> Apple </th>
<th> Orange </th>
<th> Screwdriver </th>
</tr>
```

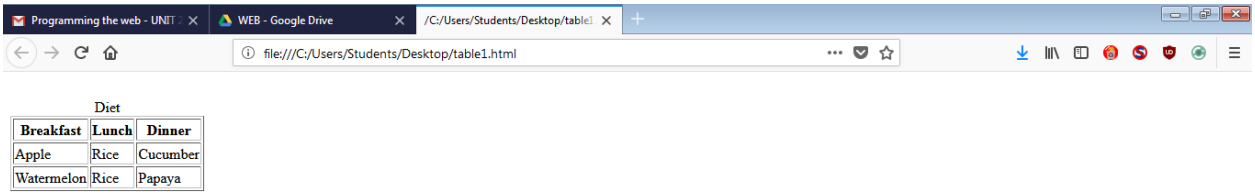
Each data row of a table is specified with a heading tag and one data tag for each data column. For example, the first data row for our work-in-progress table might be as follows:

<tr>
<th> Breakfast </th>
<td> 0 </td>
<td> 1 </td>
<td> 0 </td>
</tr>

The following document describes the whole table:

Eg:

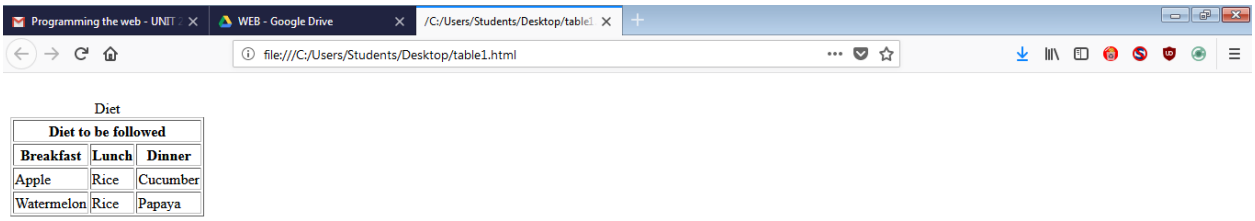
```
<html>
<body>
<br />
<table border="1">
<caption> Diet </caption>
<tr>
        <th>Breakfast</th>
        <th>Lunch</th>
        <th>Dinner</th>
</tr>
<tr>
        <td>Apple</td>
        <td>Rice</td>
        <td>Cucumber</td>
</tr>
<tr>
        <td>Watermelon</td>
        <td>Rice</td>
        <td>Papaya</td>
</tr>
</table>
</body>
</html>
```



The rowspan and colspan Attributes

In many cases, tables have multiple levels of row or column labels in which one label covers two or more secondary labels. For example, consider the display of a partial table shown in Figure below. In this table, the upper-level label 'Diet to be followed' spans the three lower-level label cells. Multiple-level labels can be specified with the rowspan and colspan attributes.

```
<tr>
<th colspan="3">Diet to be followed</th>
</tr>
```



Diet

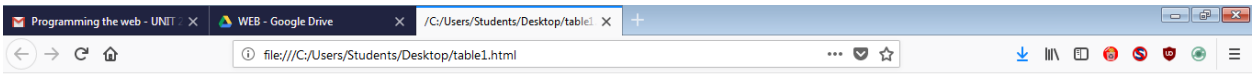
Diet to be followed		
Breakfast	**Lunch**	**Dinner**
Apple	Rice	Cucumber
Watermelon	Rice	Papaya

The colspan attribute specification in a table header or table data tag tells the browser to make the cell as wide as the specified number of rows.

Eg:

```
<html>
<body>
   <table border="1">
  <caption> Diet </caption>
        <tr>
              <th rowspan="2" ></th>
              <th colspan="3">Fruit Juice Drinks</th>
        </tr>
        <tr>
              <th>Apple</th>
              <th>Orange</th>
              <th>Strawberry</th>
        </tr>
        <tr>
              <th>Breakfast</th>
              <td>0</td>
              <td>1</td>
              <td>0</td>
        </tr>
        <tr>
              <th> Lunch</th>
              <td>1</td>
              <td>1</td>
```

```
            <td>1</td>
        </tr>
        <tr>
            <th>Dinner</th>
            <td>0</td>
            <td>1</td>
            <td>0</td>
        </tr>
</table>
</body>
</html>
```



Diet

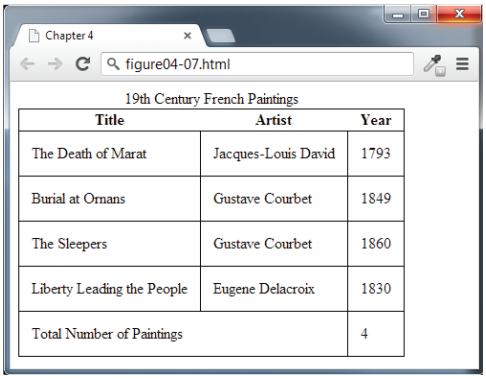
	Fruit Juice Drinks		
	Apple	Orange	Strawberry
Breakfast	0	1	0
Lunch	1	1	1
Dinner	0	1	0

Additional Table Elements

- The <thead>, <tfoot>, and <tbody> elements can also be used in table.
- The headings of the table are put in <thead> element, the content of rows in <tbody> element and if any summaries in <tfoot> element.
- These elements divide the table into different section. CSS can be applied on these sections separately.
- The <col> and <colgroup> elements are also mainly used to aid in the styling of the table. Number of colums can be grouped and similar style is applied to the whole group. The possible properties that can be set are borders, backgrounds, width, and visibility.
- HTML tables were frequently used to create page layouts, which divide the window into many frames. Images, link tag <a> and text can be put in different cells of the table. Some of the problems occurred due to this approach are –
  - Tend to dramatically increase the size of the HTML document
  - Large number of extra tags are required for <table> elements
  - These files take longer to download and are difficult to maintain because of the extra markup.
  - It is not semantic, as tables are meant to indicate tabular data

```
<table>
<tr>
        <td>
        <img src="images/959.jpg" alt="Castle"/>
        </td>
<td>
<h2>Castle</h2>
<p>Lewes, UK</p>
<p>Photo by: Michele Brooks</p>
```

```
<p>Built in 1069, the castle has a tremendous view of the
town of Lewes and the surrounding countryside.
</p>
<h3>Other Images by Michele Brooks</h3>
        <table>
        <tr>
                <td><img src="images/464.jpg" /></td>
                <td><img src="images/537.jpg" /></td>
        </tr>
        <tr>
                <td><img src="images/700.jpg" /></td>
                <td><img src="images/828.jpg" /></td>
        </tr>
        </table>
</td>
</tr>
</table>
```



2.2 Styling Tables

All the CSS properties can be applied on tables, the other styling properties for only table are-

- Table Borders
- Boxes and Zebras

Table Borders

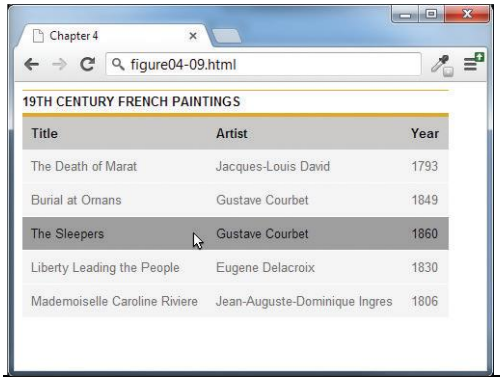
Table borders can be assigned to both the <table> and the <td> element (or <th>). Borders cannot be assigned to the <tr>, <thead>, <tfoot>, and <tbody> elements.

The border-collapse property selects the table's border model. By default, each cell has its own unique borders. The space between the adjacent borders can be changed by using the border-spacing property.

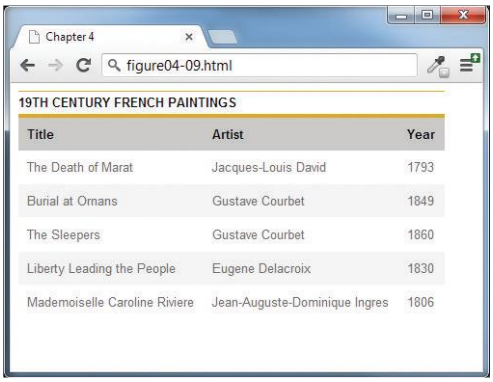
.

Boxes and Zebras

By using the CSS style, it is possible to change the background colors and borders, change the appearance of a row when mouse moves over it and also change the format of nth child




```
tbody tr:hover {
background-color: #9e9e9e;
color: black;
}
```




```
tbody tr:nth-child(odd) {
background-color: white;
}
```

2.3 Introducing Forms

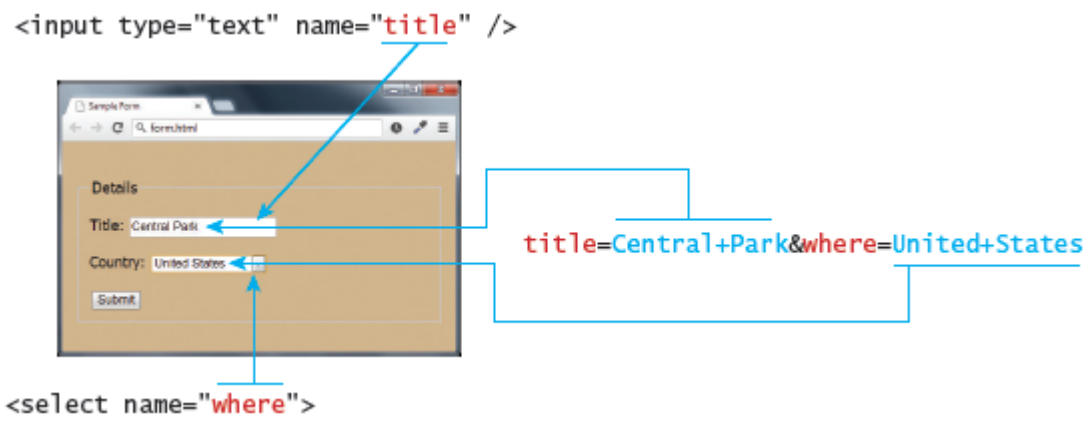
- **Forms** provide the user with an alternative way to interact with a web server. Another way is by using hyperlinks.
- Using a form, the user can enter text, choose items from lists, and click buttons. Programs running on the server will take the input from HTML forms and processes it, or save it.
- A form is defined by a <form> element, which is a container for other elements that represent the various input elements within the form as well as plain text and almost any other HTML element.

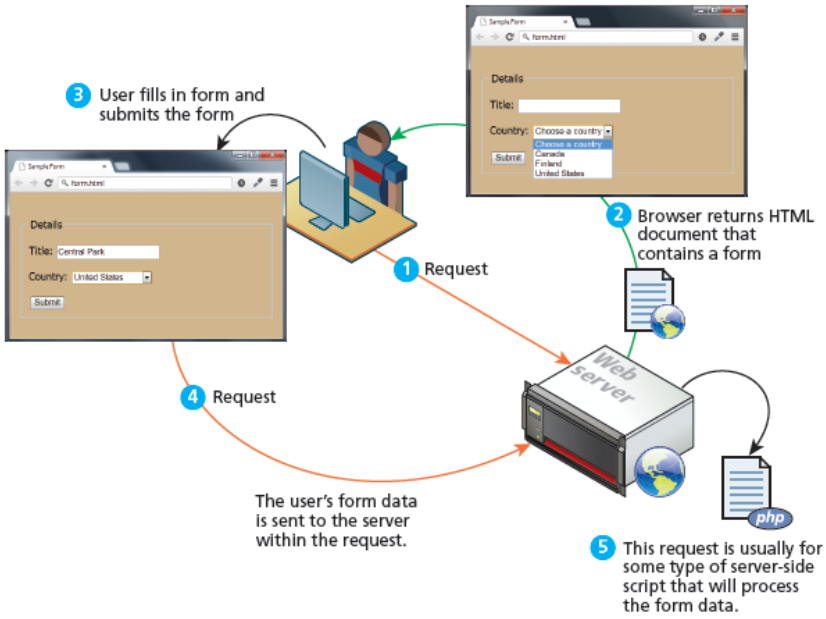
How Forms Work

While forms are constructed with HTML elements, a form also requires some type of server-side resource that processes the user's form input.

Sending of data to the server (query string):

- The browser packages the user's data input into something called a query string.
- A **query string** is a series of name=value pairs separated by ampersands (the **&** character). The names in the query string are defined in the HTML form; each form element contains a name attribute, which is used to define the name for the form data in the query string.
- The values in the query string are the data entered by the user.
- Query strings have certain rules defined by the HTTP protocol.
- Certain characters such as spaces, punctuation symbols, and foreign characters cannot be part of a query string.



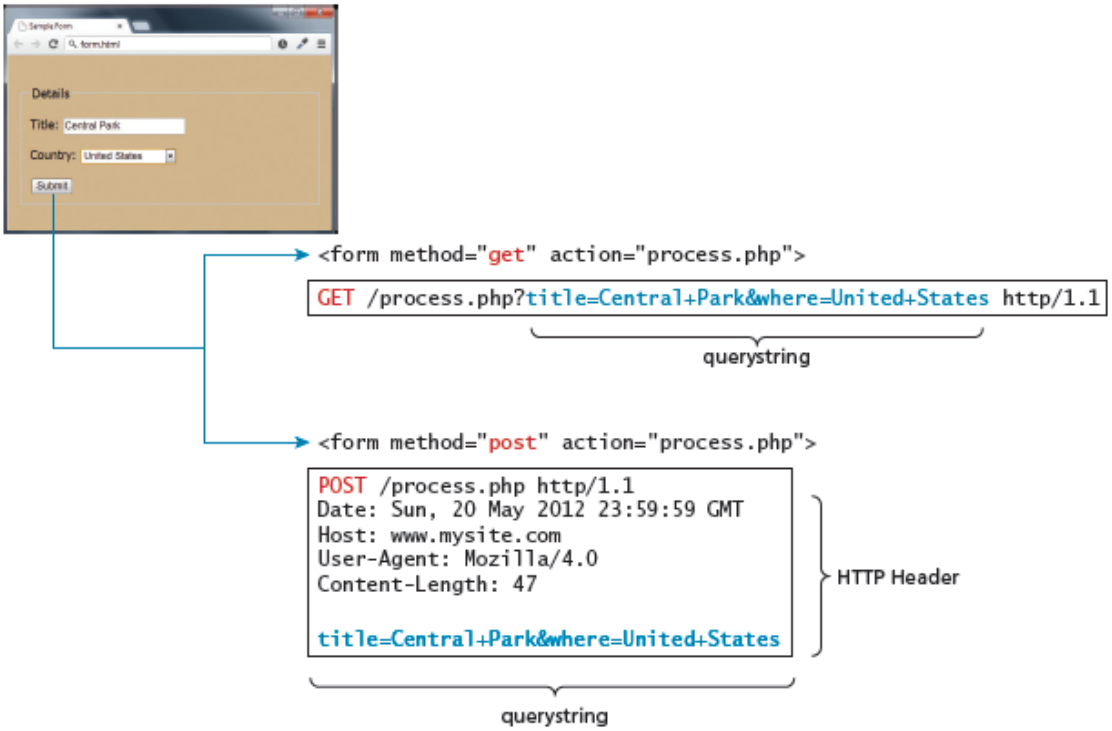


Example: Suppose the user a request for an HTML page from the server (1) that contains some type of form on it. This could be something as complex as a user registration form or as simple as a search box. After the user fills out the form, there needs to be some mechanism for submitting the form data back to the server. This is typically achieved via a **submit** button. As interaction between the browser and the web server is governed by the HTTP protocol, the form data must be sent to the server via a standard HTTP request. This request is typically some type of server-side program that will process the form data in some way; this could include checking it for validity, storing it in a database, or sending it in an email.

The <form> Element

- The HTML form contains two important attributes that are essential features of any form, namely the **action and the method** attributes.
- **Action** attribute specifies the **URL of the server-side resource** that will process the form data. This could be a resource on the same server as the form or a completely different server.
- The **method** attribute specifies how the query string data will be transmitted from the browser to the server. There are two possibilities: GET and POST.

The use of GET or POST method decides where the browser locates the user input in the HTTP request. Using **GET**, the browser locates the data in the URL of the request; with **POST**, the form data is located in the HTTP header after the HTTP variables.



	GET Method	POST Method
1	Clearly seen in the address bar	Data is hidden from the user
2	Ususally used during development, for testing	Used once the product is ready
3	Data remains in browser history and cache.	Submitted data is not stored in browser history or cache.
4	Limits on the number of characters in the form.	There is no limit on number of characters entered in form.

Generally, form data is sent using the POST method. However, the GET method is useful when you are testing or developing a system, since you can examine the query string directly in the browser's address bar.

2.4 Form Control Elements

Some of the form related HTML elements are –

Type	Description
<button>	Defines a clickable button.
<datalist>	An HTML5 element that defines lists of pre-defined values to use with input fields.
<fieldset>	Groups related elements in a form together.
<form>	Defines the form container.
<input>	Defines an input field. HTML5 defines over 20 different types of input.
<label>	Defines a label for a form input element.
<legend>	Defines the label for a fieldset group.
<option>	Defines an option in a multi-item list.
<optgroup>	Defines a group of related options in a multi-item list.
<select>	Defines a multi-item list.
<textarea>	Defines a multiline text entry box.

Text Input Controls

Text Input Controls – used to get text information from the user. It is used in search box to enter search element, in a login form to enter the name, in user registration form to enter name, parents name, caste etc. Different text input controls are –

Type	Description
text	Creates a single-line text entry box. <input type="text" name="title" />
textarea	Creates a multiline text entry box. <textarea rows="3" cols=”50” />
password	Creates a single-line text entry box for a password (which masks the user entry as bullets or some other character) <input type="password" ... />
email	Creates a single-line text entry box suitable for entering an email address. This is an HTML5 element. Some browsers will perform validation when form is submitted. <input type="email" ... />
tel	Creates a single-line text entry box suitable for entering a telephone. This is an HTML5 element. Since telephone numbers have different formats in different parts of the world, current browsers do not perform any special formatting or validation. Some devices may, however, provide a specialized

	keyboard for this element. <input type="tel" ... />
url	Creates a single-line text entry box suitable for entering a URL. This is an HTML5 element. Browsers perform validation on submission. <input type="url" ... />

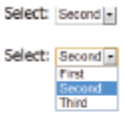
Choice Controls

Forms often need the user to select an option from a group of choices. HTML provides several ways to do this.

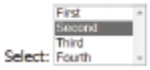
Select Lists

- The <select> element is used to create a multiline box for selecting one or more items.
- The options (defined using the <option> element) can be hidden in a dropdown list or multiple rows of the list can be visible.
- Option items can be grouped together via the <optgroup> element. The selected attribute in the <option> makes it a default value.
- The value attribute of the <option> element is used to specify what value will be sent back to the server in the query string when that option is selected.
- The value attribute is optional; if it is not specified, then the text within the container is sent.

The new **<datalist>** element is a new addition to HTML5. This element allows to define a list of values that can appear in a drop-down autocomplete style list for a text element. This can be helpful for situations in which the user must have the ability to enter anything, but are often entering one of a handful of common elements.

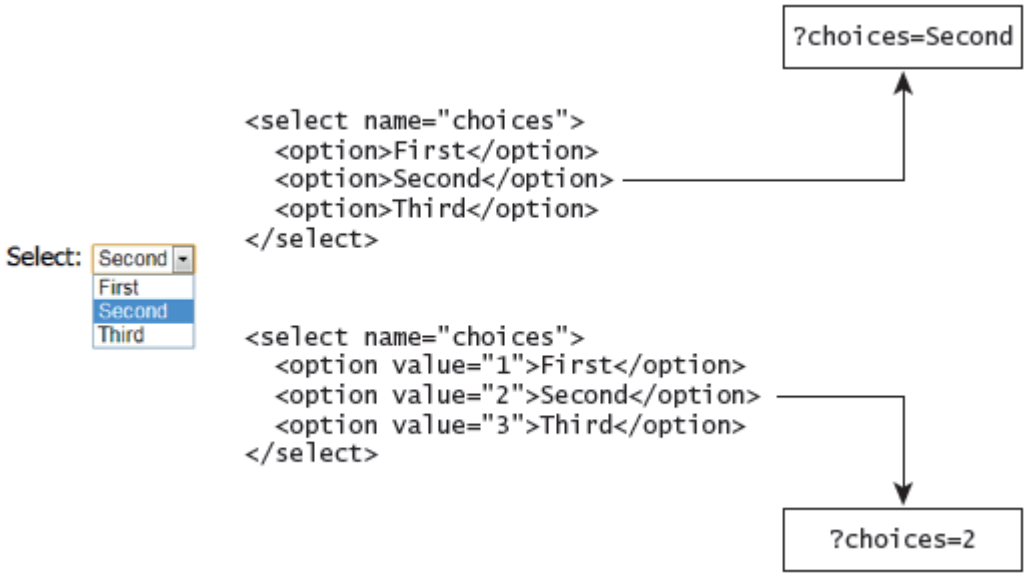



```
<select name="choices">
  <option>First</option>
  <option selected>Second</option>
  <option>Third</option>
</select>
```




```
<select size="3" ... >
```


```
<select … >
 <optgroup label="North America">
   <option>Calgary</option>
   <option>Los Angeles</option>
 </optgroup>
 <optgroup label="Europe">
   <option>London</option>
   <option>Paris</option>
   <option>Prague</option>
 </optgroup>
</select>
```




```
<input type="text" name="city" list="cities" />
<datalist id="cities">
<option>Calcutta</option>
<option>Calgary</option>
<option>London</option>
<option>Los Angeles</option>
<option>Paris</option>
<option>Prague</option>
</datalist>
```

Radio Buttons

Radio buttons are used when the user has to select a single item from a small list of choices and all the choices have to be visible. They are added by using the <input type="radio"> element. The buttons are made mutually exclusive (i.e., only one can be chosen) by sharing the same name attribute. The checked attribute is used to indicate the default choice, while the value entered in value attribute is sent to the server when submit button is clicked.

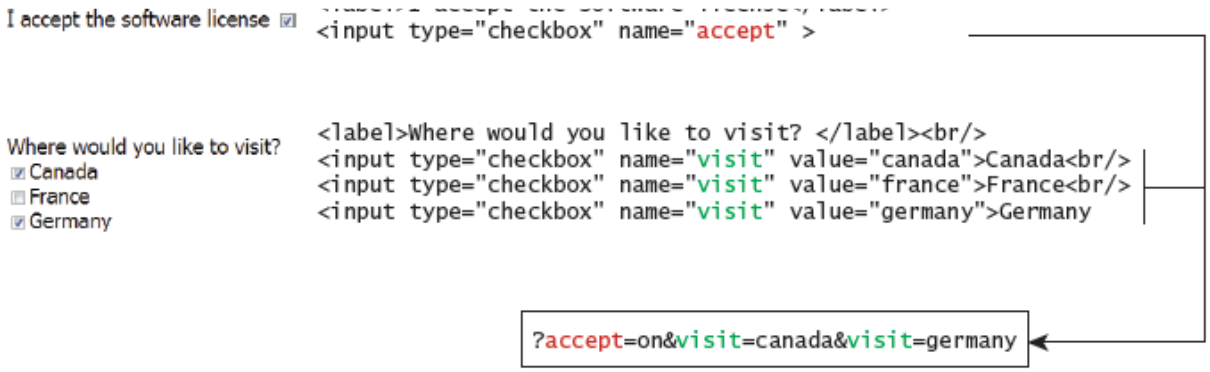
Eg:

```
<input type="radio" name="where" value="1">North America<br/>
<input type="radio" name="where" value="2" checked>South America<br/>
<input type="radio" name="where" value="3">Asia
```

Continent:
- North America
- South America
- Asia

Checkboxes

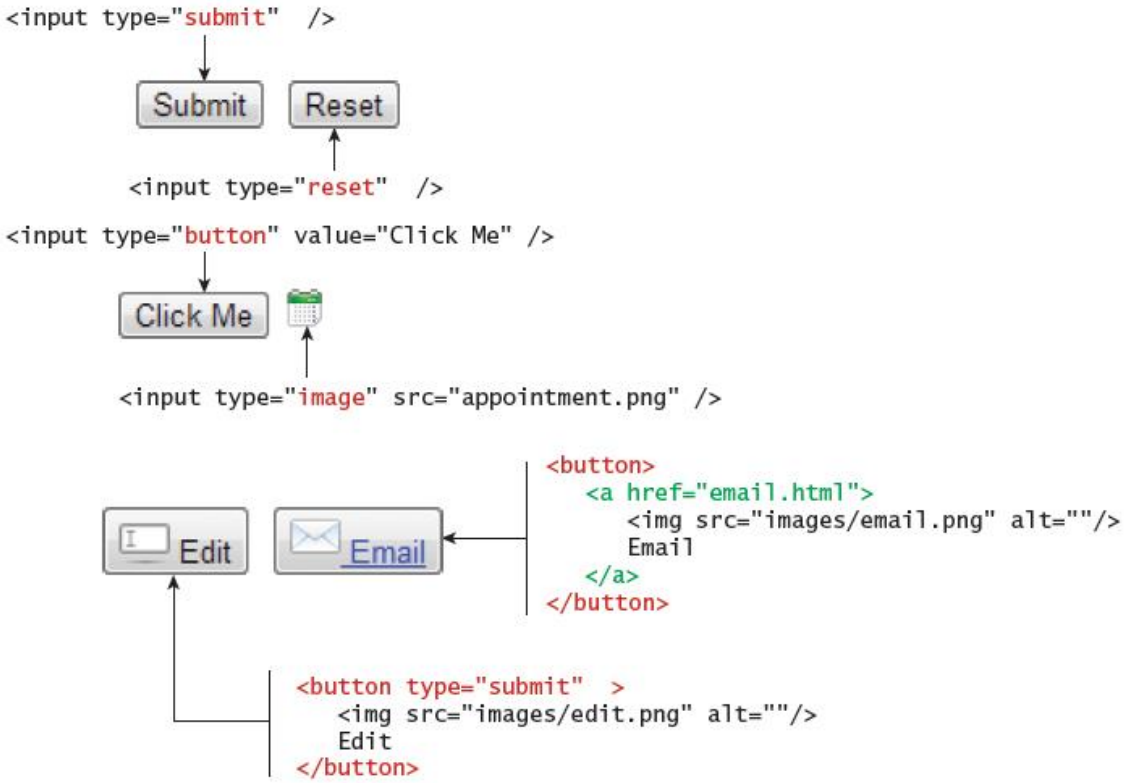
Checkboxes are used for getting yes/no or on/off responses from the user. Checkboxes are added by using the <input type="checkbox"> element. You can also group checkboxes together by sharing the same name attribute. Each checked checkbox will have its value sent to the server.



Button Controls

HTML defines several different types of buttons.

Type	Description
<input type="submit">	Creates a button that submits the form data to the server
<input type="reset">	Creates a button that clears the user's already entered form data.
<input type="button">	Creates a custom button. This button may requires a script for it to actually perform any action
<input type="image">	Creates a custom submit button that uses an image for its display
<button>	Creates a custom button. The <button> element differs from <input type="button"> in that you can completely customize what appears in the button. It can be used to include both images and text.

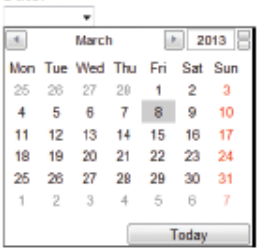


Date and Time Controls

The user entering of date or time may need to be validated, as it may cause error when manually entered. The new HTML date and time controls make it easier for users to input these tricky date and time values. The format output of all date & time controls vary depending on the browser.

Type	Description
date	Creates a general date input control. The format for the date is "mm-dd-yyyy".
time	Creates a time input control. The format for the time is "HH:MM AM/PM," for hours:minutes
datetime	Creates a control in which the user can enter a date and time.
datetime-local	Creates a control in which the user can enter a date and time without specifying a time zone
month	Creates a control in which the user can enter a month in a year. The format is "mm- yyyy."
week	Creates a control in which the user can specify a week in a year. The format is "W##- yyyy."

Date:

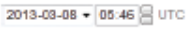



```
<label>Date: <br/>
<input type="date" ... />
```

Time:


```
<input type="time" ... />
```

DateTime:

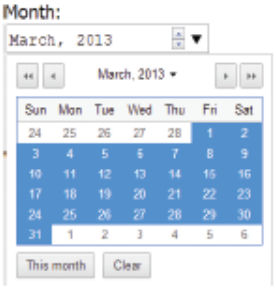



```
<input type="datetime" ... />
```

DateTime Local:

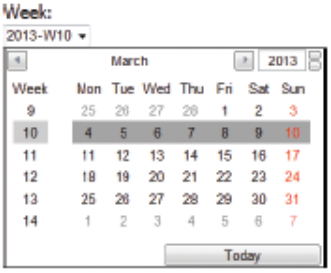

```
<input type="datetime-local" ... />
```

Month:




```
<input type="month" ... />
```

Week:




```
<input type="week" ... />
```

2.5 Table and Form Accessibility

Users with sight disabilities, for instance, experience the web using voice reading software. Color blind users might have trouble differentiating certain colors in proximity; users with muscle control problems may have difficulty using a mouse, while older users may have trouble with small text and image sizes.

The term **web accessibility** refers to the assistive technologies, various features of HTML that work with those technologies, and different coding and design practices that can make a site more usable for people with visual, mobility, auditory, and cognitive disabilities.
In order to improve the accessibility of websites, the W3C created the **Web Accessibility Initiative (WAI)** in 1997. The WAI produces guidelines and recommendations, as well as organizing different working groups on different accessibility issues.

Perhaps the most important guidelines for web accessibility are:

■ *Provide text alternatives for any nontext content so that it can be changed into other forms people need, such as large print, braille, speech, symbols, or simpler language*.
■ *Create content that can be presented in different ways (for example simpler layout) without losing information or structure*.
■ *Provide ways to help users navigate, find content, and determine where they are.*

The guidelines provide detailed recommendations on how to achieve this advice.

Accessible Tables
HTML tables can be quite frustrating, for people with visual disability. One important way to improve the accessibility is to only use tables for tabular data, not for layout. Using the following accessibility features for tables in HTML can improve the using of tables for those users:
1. Describe the table's content using the <caption> element. This provides the user with the ability to discover what the table is about before having to listen to the content of each and every cell in the table.
2. Connect the cells with a textual description in the header. It is quite revealing to listen to reader software recite the contents of a table that has not made these connections. It sounds like this: "row 3, cell 4: 45.56; row 3, cell 5: Canada; row 3, cell 6: 25,000; etc." However, if these connections have been made, it sounds instead like this: "row 3, Average: 45.56; row 3, Country: Canada; row 3, City Count: 25,000; etc.,".

Accessible Forms
HTML forms are also potentially problematic with respect to accessibility. The use of the <fieldset>, <legend>, and <label> elements, provide a connection between the input elements in the form.
The main purpose of <fieldset> and <legend> is to logically group related form input elements together with the <legend> providing a type of caption for those elements.
Each <label> element should be associated with a single input element, by using the 'for' attribute. So that if the user clicks on or taps the <label> text, that control will receive the form's focus (i.e., it becomes the current input element and any keyboard input will affect that control).

Associating label and the input tag -

```
<label for="f-title">Title: </label>
<input type="text" name="title" id="f-title"/>
<label for="f-country">Country: </label>
<select name="where" id="f-country">
<option>Choose a country</option>
<option>Canada</option>
<option>Finland</option>
<option>United States</option>
</select>
```

2.6 Microformats

The web has millions of pages and there are many similar information from site to site. Most sites have Contact Us page, in which addresses and other information are displayed; calendar of upcoming events or information about products or news. These types of common information can be tagged in a similar way, and automated tools can be used to gather and transform the information.

A **microformat** is a small pattern of HTML markup and to represent common blocks of information such as people, events, and news stories so that the information in them can be extracted and indexed by software agents.

Module II-Chapter -2
Advanced CSS

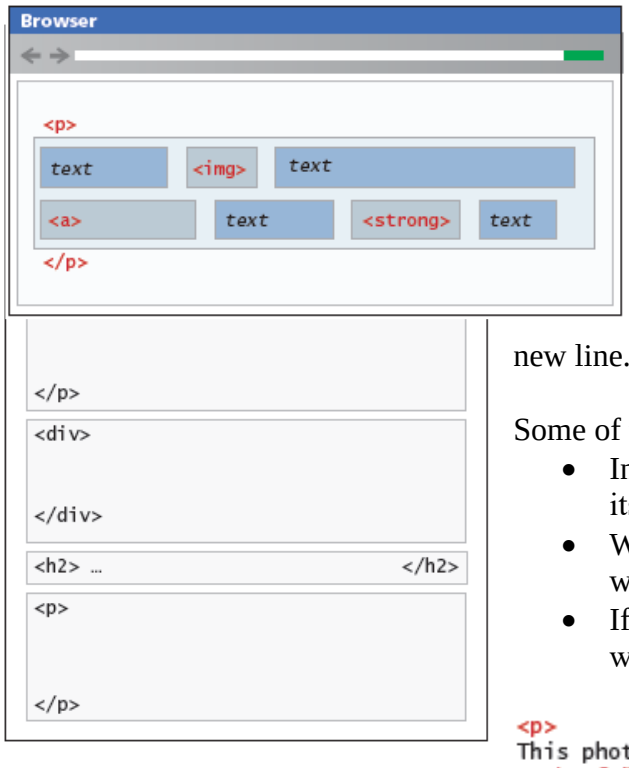
2.7 Normal Flow

The **normal flow** in html refers to how the browser will normally display block-level elements and inline elements from left to right and from top to bottom.

Block-level elements such as <p>, , <h2>, <ul>, and <table> are elements that are contained on their own line, because block-level elements begin with a line break (new line). Two block-level elements can't exist on the same line, without styling.

Some of the properties of block-level elements -

- Each block exists on its own line.
- It is displayed in normal flow from the browser window's top to its bottom.
- By default each block level element fills up the entire width of its parent (browser window).
- CSS box model properties can be used to customize, for instance, the width of the box and the margin space between other block level elements



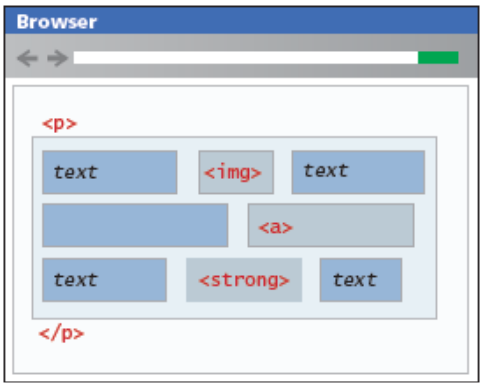
Inline elements do not form their own blocks but instead are displayed within lines. Normal text in an HTML document is inline, and also elements such as , <a>, <img>, and are inline. Inline elements line up next to one another horizontally from left to right on the same line, when there is no enough space left on the line, the content moves to a new line.

Some of the properties of inline elements are –

- Inline element is displayed in normal flow from its container's left to right.
- When a line is filled with content, the next line will receive the remaining content, and so on.
- If the browser window resizes, then inline content will be "re-flowed" based on the new width.

```
<p>
This photo <img src="photo-con.png" alt="" /> of Conservatory Pond in
<a href="http://www.centralpark.com/">Central Park</a> New York City
was taken on October 22, 2015 with a <strong>Canon EOS 30D</strong>
camera.
</p>
```

If the <p> tag contains many tags as shown, the inline tags are placed with-in the <p> tag as shown. If the window is re-sized the elements are re-flowed and now occupies three rows.



There are two types of inline elements: replaced and nonreplaced. **Replaced inline elements** are elements whose content and appearance is defined by some external resource, such as <img> and the various form elements.
Nonreplaced inline elements are those elements whose content is defined within the document, which includes all the other inline elements. Eg: <a>,<b>,<i>,.
Replaced inline elements have a width and height that are defined by the external resource, eg. the size of image is defined externally.

In a document with normal flow, block-level elements and inline elements are placed together. Block-level elements will flow from top to bottom, and inline elements flow from left to right within a block. A block element can contain another block.

It is possible to change whether an element is block-level or inline using the CSS 'display' property. Consider the following two CSS rules:
span { display: block; }
li { display: inline; }
These two rules will make all elements behave like block-level elements and all <li> elements like inline (that is, each list item will be displayed on the same line).

2.8 Positioning Elements

It is possible to
1) move an item from its regular position in the normal flow
2) move an item outside of the browser viewport so that it is not visible
3) move to position so that it is always visible in a fixed position while the rest of the content scrolls.

The position property is used to specify the type of positioning, and the possible values are

Type	Description
absolute	The element is removed from normal flow and positioned in relation to its nearest positioned ancestor.
fixed	The element is fixed in a specific position in the window even when the document is scrolled
relative	The element is moved relative to where it would be in the normal flow.
static	The element is positioned according to the normal flow. **This is the default**.

The left, right, top, and bottom properties are used to indicate the distance the element will move.

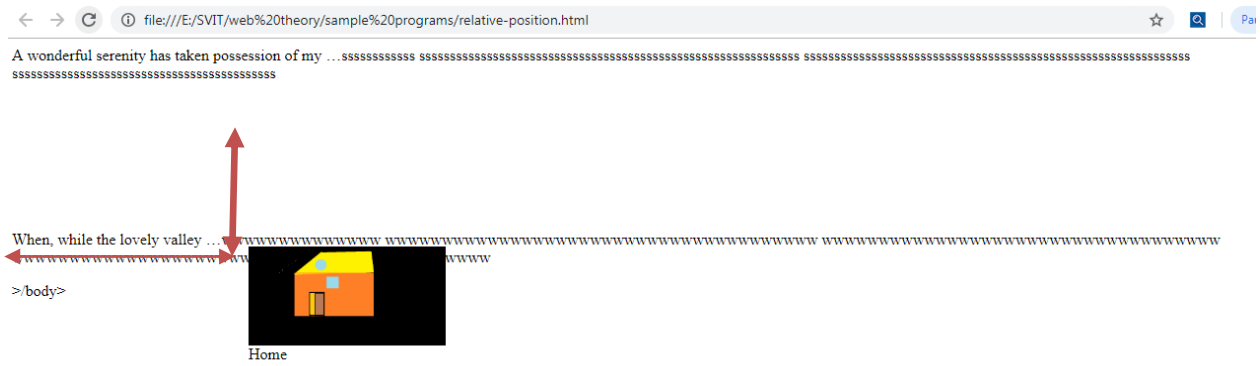
Relative Positioning

In **relative positioning** an element is displaced out of its normal flow position and moved relative to where it would have been placed normally. The other contents around the relatively positioned element remain in its old position in the flow; thus the space the element would have occupied is preserved as shown in the example below.

Eg:

```
<html>
<head>
	<style>
		figure {
		position: relative;
		top: 150px;
		left: 200px;
		}
	</style>
</head>
<body>
<p>A wonderful serenity has taken possession of my …sssssssssss
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
sssssssssssssssssssssssssssssssssssssssssss</p>
<figure>
<img src="HOUSE.png" width="200" height="100" />
<figcaption>Home</figcaption>
</figure>
<p>When, while the lovely valley …wwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww</p>

</body>
</html>
```



The repositioned element overlaps other content: that is, the <p> element following the <figure> element does not change to accommodate the moved <figure>.

Absolute Positioning

When an element is positioned absolutely, it is removed completely from normal flow. Here, space is not left for the moved element, as it is no longer in the normal flow. Its position is moved in relation to its container block. In the below example, <figure> block's container is body block.

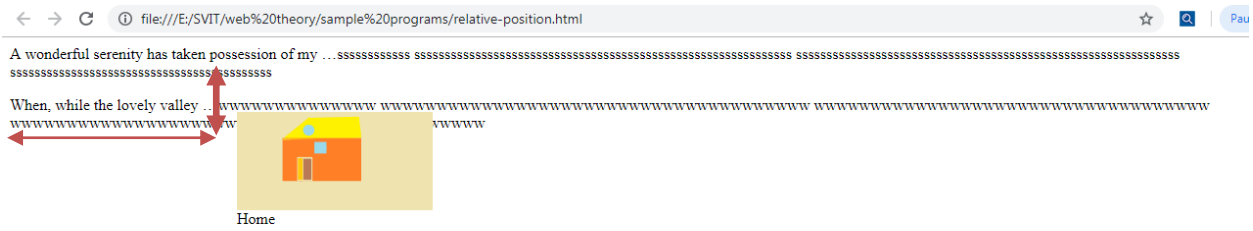
The moved block can overlap the content in the underlying normal flow.

Eg:

```
<html>
<head>
        <style>
        figure {
        position: absolute;
        top: 60px;
        left: 200px;
        }
        </style>
</head>
<body>
<p>A wonderful serenity has taken possession of my …sssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssss</p>
<figure>
<img src="HOUSE.png" width="200" height="100" />
<figcaption>Home</figcaption>
</figure>
<p>When, while the lovely valley …wwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww</p>

</body>
```

</html>



Z-Index

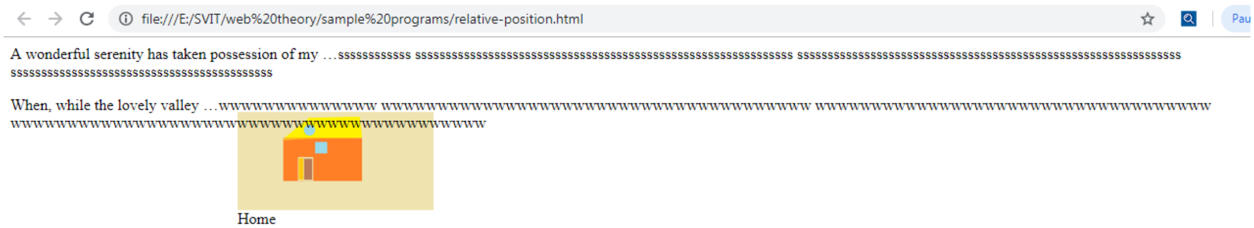
Each positioned element has a stacking order defined by the z-index property (named for the z-axis). Items closest to the viewer (and thus on the top) have a larger **z-index** value, as shown in the example below.

Eg:

```
<html>
<head>
      <style>
          figure {
          position: absolute;
          top: 60px;
          left: 200px;
          z-index:-1;
          }

          body{
          z-index:1:
          }
       </style>
</head>
<body>
<p>A wonderful serenity has taken possession of my …sssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
sssssssssssssssssssssssssssssssssssssssssssssssssssss</p>
<figure>
<img src="HOUSE.bmp" width="200" height="100" />
<figcaption>Home</figcaption>
</figure>
<p>When, while the lovely valley …wwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww</p>

</body>
</html>
```



Observe the <p> tag content has appeared over the image.

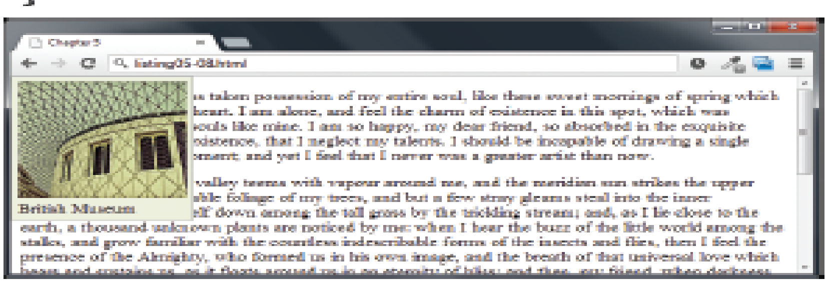
Fixed Position

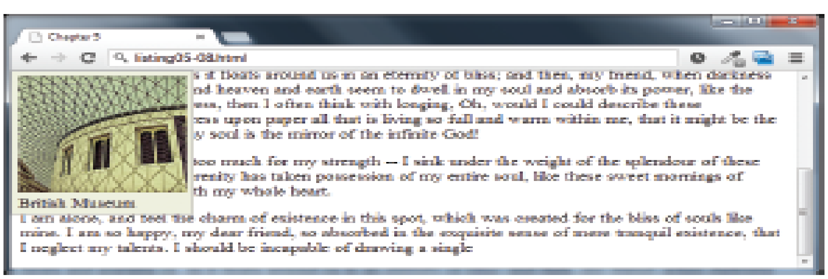
The fixed position value is used relatively infrequently. The element is positioned in relation to the viewport (i.e., to the browser window). Elements with **fixed positioning** do not move when the user scrolls up or down the page.

The fixed position is used to ensure that navigation elements or **advertisements are always visible.**

```
<head>
      <style>
          figure {
          position: fixed;
          top: 60px;
          left: 200px;
          }
       </style>
</head>
```

```
figure {
    ...
    position: fixed;
    top: 0;
    left: 0;
}
```





2.9 Floating Elements

It is possible to displace an element out of its position in the normal flow via the CSS float **property**. An element can be floated to the left or floated to the right.

When an item is floated, it is moved all the way to the far left or far right of its containing block and the rest of the content is "**re-flowed**" around the floated element.

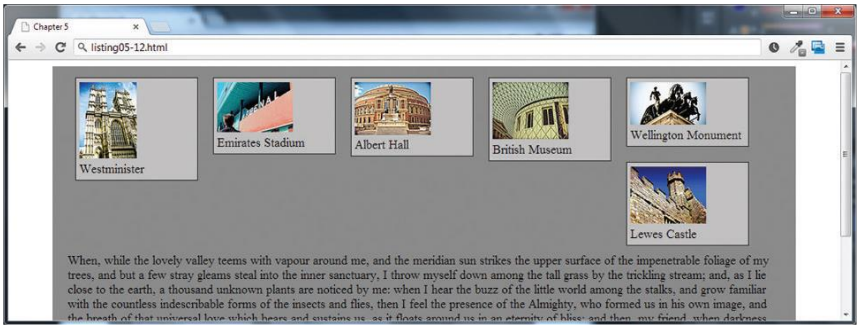
Notice that a floated block-level element must have a width specified; otherwise, the width will be set to auto, which will mean it implicitly fills the entire width of the containing block, and there will be no room available to flow content around the floated item.

Floating within a Container

It should be reiterated that a floated item moves to the left or right of its container.
The floated figure contained within an <article> element that is indented from the browser's edge. The relevant margins and padding areas are color coded to help make it clearer how the float interacts with its container.

Floating Multiple Items Side by Side

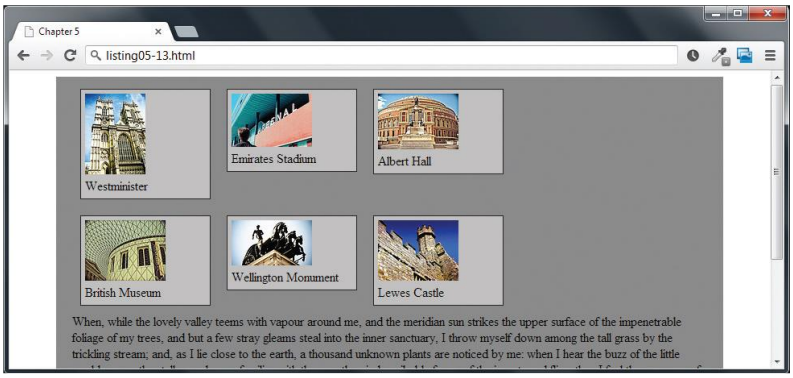
A common use of float property is to place multiple items side by side on the same line. When multiple items are floated, each element will be nestled up beside the previously floated item. All other content in the containing block will flow around all the floated elements.




```
figure {
...
width: 150px;
float: left;
}
```

This arrangement of images floated changes as the browser window size changes. If suppose any element has to be stopped from flowing around a floated element, it can be done by using the **clear** CSS **property**.

By setting the clear property of third image to left, it means that there should be no elements to its left. { clear : left;}



The other values for clear property are described below –

Value	Description
left	The left-hand edge of the element cannot be adjacent to another element.
right	The right-hand edge of the element cannot be adjacent to another element.
both	the left-hand and right-hand edges of the element cannot be adjacent
none	The element can be adjacent to other elements.

Overlaying and Hiding Elements

One of the more common design tasks with CSS is to place two elements on top of each other, or to selectively hide and display elements. Positioning is important to both of these tasks. Positioning is often used for smaller design changes, such as moving items relative to other elements within a container.

An image that is the same size as the underlying one is placed on top of the other image using absolute positioning.

There are in fact two different ways to hide elements in CSS: using the display property and using the visibility property. The display property takes an item **out of the flow**: it is as if the element no longer exists. The visibility property just **hides the** element, but the space for that element remains.

```
figure {
    ...
    display: auto;
}
```

```
figure {
    ...
    display: none;
}
```

```
figure {
    ...
    visibility: hidden;
}
```

2.10 Constructing Multicolumn Layouts

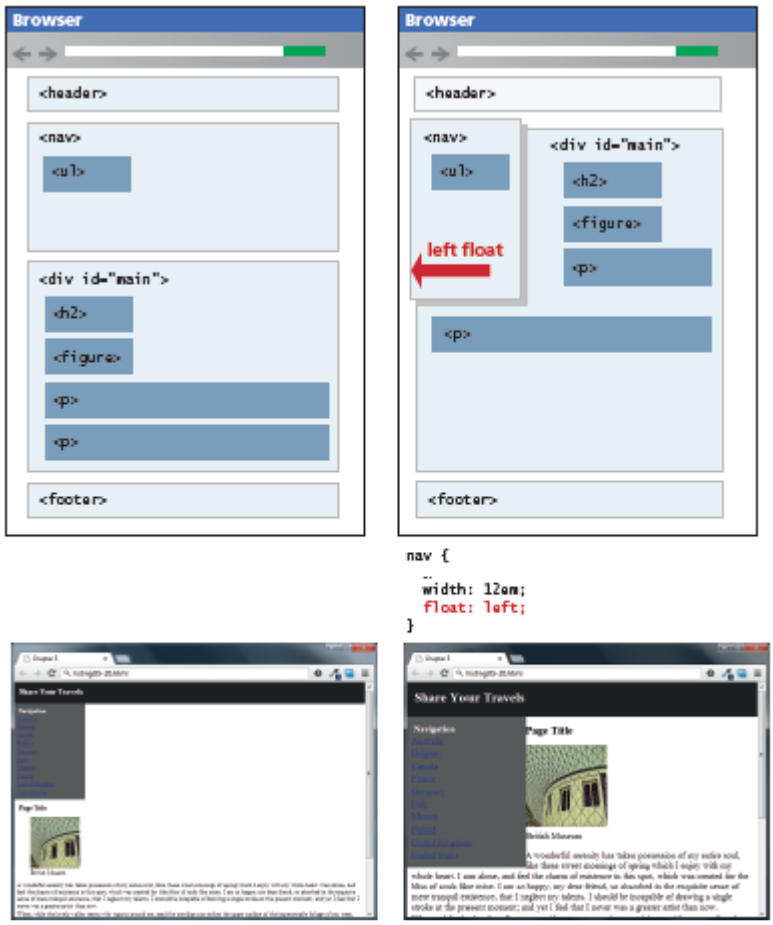
The previous sections showed two different ways to move items out of the normal top-down flow, by using positioning (relative, absolute or fixed) and by using floats. They are the techniques that can be used to create more complex layouts. The below topics are about the creation of layout using float and positioning property of CSS.

Using Floats to Create Columns

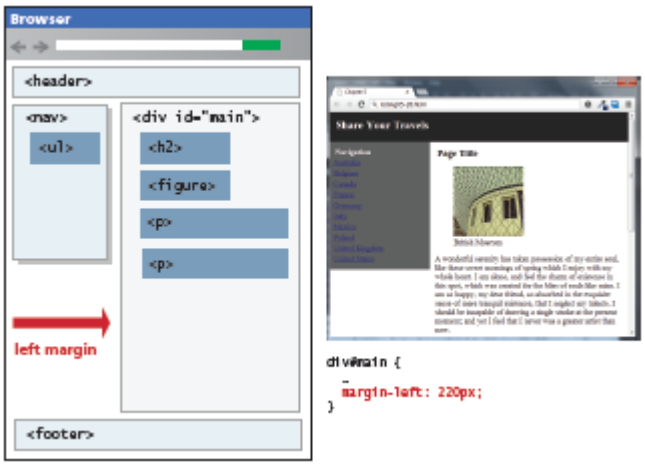
Using floats is the most common way to create columns of content. The steps for this approach are as follows –

1. float the content container that will be on the left-hand side. (the floated container needs to have a width specified).
2. The other content will flow around the floated element.
3. Set the left – hand side margin for the non-floated element.

The layout without float Property	The layout after using the float property for left side element



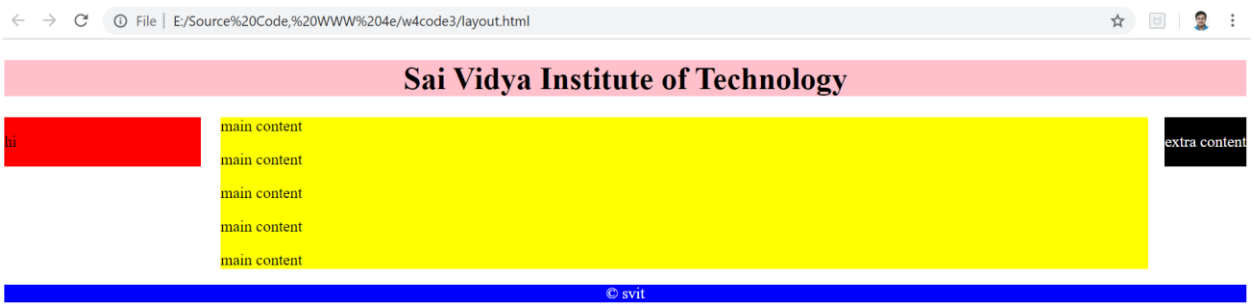
Step 3: setting the left margin -




```
<!DOCTYPE html>
<html>
        <head>
                <title> Simple Form </title>
                <style type = "text/css">
                        nav{
```

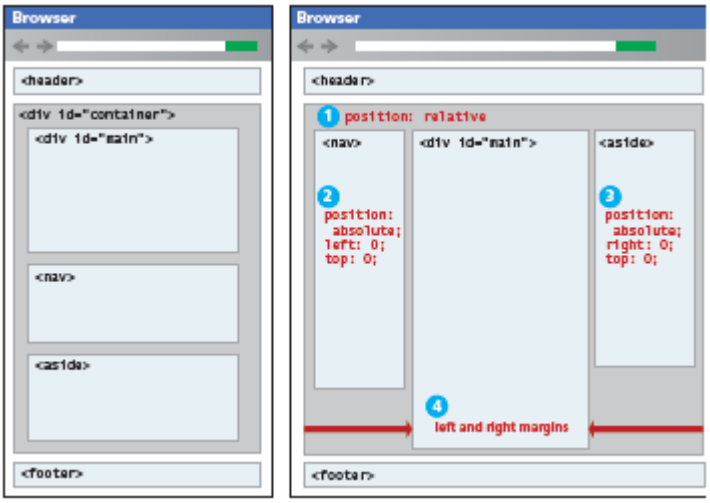
```
                        width:200px;
                        float:left;
                        background-color:red;
                        }
                div{
                        margin-left: 220px;
                        margin-right: 100px;
                        background-color:yellow;
                        }
        </style>
    </head>
    <body>
        <header style="background-color:pink;align:center;">
            <h1 style="text-align:center;">Sai Vidya Institute of Technology</h1>
        </header>
        <aside style="background-color:black;color:white;float:right;">
            <p>extra content</p>
        </aside>
        <nav>
            <p>hi</p>
        </nav>
        <div>
            <p> main content </p>
            <p> main content </p>
            <p> main content </p>
            <p> main content </p>
            <p> main content </p>
        </div>

        <footer style="background-color:blue;color:white;">
            <p style="text-align:center;">&copy svit</p>
        </footer>
    </body>
</html>
```



Using Positioning to Create Columns

Positioning can also be used to create a multicolumn layout. Typically, the approach is to absolute position the elements.
This approach uses some type of container, in which the elements are positioned.



The following steps are followed –

1. Position the container element (into which all other elements are positioned) with respect to the browser window.
2. Position the other elements with respect to the container element, created in step 1.

```
<html>
<head>
<style>

#container{
        position:relative;              //main container
        top:0px;
        left:0px;
}
nav
{
        position:absolute;              // positioned with respect to main container
        top: 0px; left:0px;
        width:150px;
        height:300px;
        background-color: #cc0055
}

#side{
        margin-left: 150px;                     // column in the middle
        margin-right:150px;
        height: 300px;
        background-color:#CCCCCC}
h1{   background-color:red; }
```

```
</style>
</head>
<body>

<h1 align="center" > HHHHH</h1>

<div id="container">

<nav>
<ul>
<li>abc</li>
<li>def</li>
<li>abc</li>
<li>def</li>
</ul>
</nav>

<article id="side">
<figure>
<img src="HOUSE.bmp" width="200" height="100" />
<figcaption>Home</figcaption>
</figure>

<p>A wonderful serenity has taken possession of my …sssssssssss
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
sssssssssssssssssssssssssssssssssssssssss</p>

</article>

<p style = "clear:left" >When, while the lovely valley …wwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww</p>
</div>
</body>
</html>
```

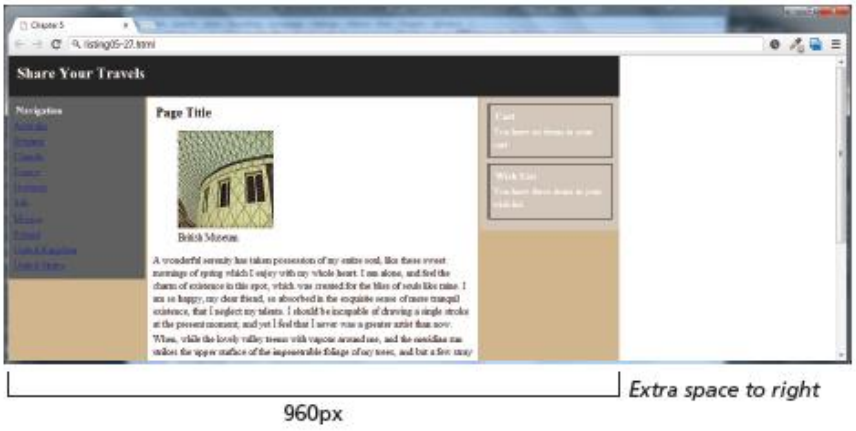
2.11 Approaches to CSS Layout

One of the main problems faced by web designers is that the size of the screen used to view the page can vary. Some users will visit a site on a 21-inch wide screen monitor that can display 1920 × 1080 pixels (px); others will visit it on an older iPhone with a 3.5 screen and a resolution of 320 × 480 px. Users with the large monitor might expect a site to take advantage of the extra size; users with the small monitor will expect the site to scale to the smaller size and still be usable.

Most designers take one of two basic approaches to dealing with the problems of screen size - **Fixed Layout and Liquid Layout.**

Fixed Layout

In a **fixed layout**, the basic width of the design is set by the designer, typically corresponding to an "ideal" width based on a "typical" monitor resolution. A common width used is something in the 960 to 1000 pixel range, which fits nicely in the common desktop monitor resolution (1024 × 768). This content may be positioned on the left or the center of the monitor.



Fixed layouts are created using pixel units, typically with the entire content within a container whose width property has been set to some width.

```
<style>
div#wrapper {
width: 960px;
background_color: tan;
}
</style>

<body>
<div id="wrapper">
……
</div>
……..
</body>
```

The advantage of a fixed layout –

- easy to produce
- predictable visual result
- optimized for typical desktop monitors

The disadvantage of a fixed layout –

- For larger screens, there may be an excessive amount of blank space to the left and/or right of the content.
- When the browser window is less than the fixed width; the user will have to horizontally scroll to see all the content.
- If smaller mobile devices are used, more horizontal scrolling has to be done

Liquid Layout

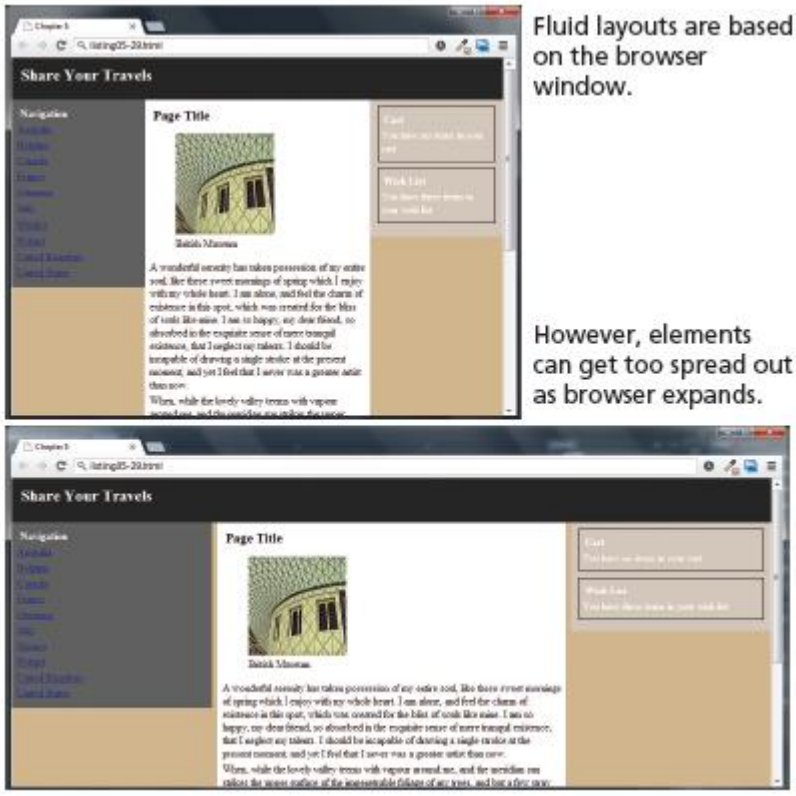
In Liquid Layout, the widths are not specified using pixels, but percentage values. Percentage values in CSS are a percentage of the current browser width, so a layout in which all widths expressed as percentages should adapt to any browser size.

The advantage of a liquid layout –

- Adapts to different browser sizes, so there is neither wasted white space nor any need for horizontal scrolling

The disadvantage of a liquid layout –

- more difficult to create because some elements, such as images, have fixed pixel sizes.
- The screen may grow or shrink dramatically.



Other Layout Approaches

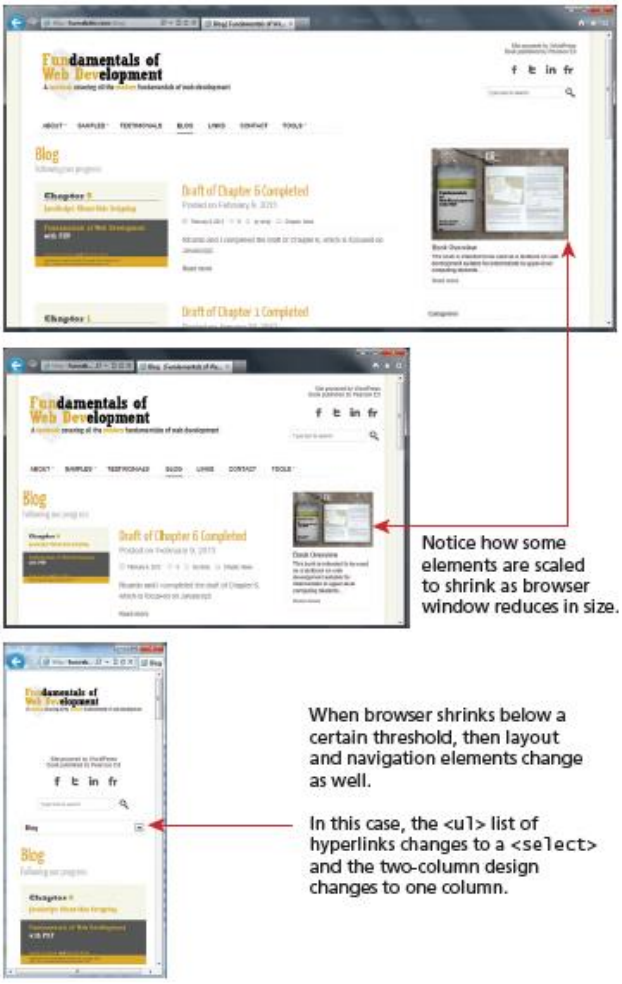
While the fixed and liquid layouts are the two basic paradigms for page layout, there are some other approaches that combine the two layout styles. Most of the other approaches are **hybrid layout**, were there is a fixed layout and liquid layout.

Fixed layout is commonly used for a sidebar column containing graphic advertising images that must always be displayed and which always are the same width. But liquid layout are used for the main content or navigation areas, with perhaps min and max size limits in pixels set for the navigation areas.

2.12 Responsive Design

In a **responsive design**, the page "responds" to changes in the browser size that go beyond the width scaling of a liquid layout.

One of the problems of a liquid layout is that images and horizontal navigation elements tend to take up a fixed size, and when the browser window shrinks to the size of a mobile browser, liquid layouts can become unusable. In a responsive layout, images will be scaled down and navigation elements will be replaced as the browser shrinks, as shown in the figure below.



There are four key components that make responsive design work. They are:

1. Liquid layouts
2. Scaling images to the viewport size
3. Setting viewports via the <meta> tag
4. Customizing the CSS for different viewports using media queries

Responsive designs begin with a liquid layout, in which most elements have their widths specified as percentages. Making images scale in size is done as follows:

```
img {
max-width: 100%;
}
```

But this does not change the downloaded size of the image; it only shrinks or expands its visual display to fit the size of the browser window, never expanding beyond its actual dimensions.

Setting Viewports

A key technique in creating responsive layouts is the ability of current mobile browsers to shrink or grow the web page to fit the width of the screen. The mobile browser renders the page on a canvas called the **viewport**. On iPhones, for instance, the viewport width is 980 px, and then that viewport is scaled to fit the current width of the device. The mobile Safari browser introduced the viewport <meta> tag as a way for developers to control the size of that initial viewport.

```
<html>
<head>
<meta name="viewport" content="width=device-width" />
```

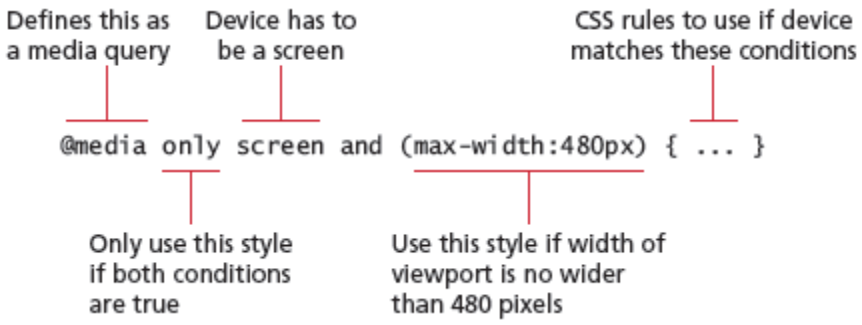
By setting the viewport as above, the page is telling the browser that no scaling is needed, and to make the viewport as many pixels wide as the device screen width. This means that if the device has a screen that is 320 px wide, the viewport width will be 320 px; if the screen is 480 px, then the viewport width will be 480 px.

Media Queries

The other key component of responsive designs is **CSS media queries**. A media query is a way to apply style rules based on the medium that is displaying the file. Use these queries to look at the capabilities of the device, and then define CSS rules to target that device.

Example of media query

@media only screen and (max-width: 480px) {……}



These queries are Boolean expressions and can be added to your CSS files or to the <link> element to conditionally use a different external CSS file based on the capabilities of the device.

Few elements of the browser features that can be examined with media queries are –

Feature	Description
width	Width of the viewport
height	Height of the viewport
device-width	Width of the device
device-height	Height of the device
orientation	Whether the device is portrait or landscape
color	The number of bits per color

Contemporary responsive sites will typically provide CSS rules for phone displays first, then tablets, then desktop monitors, an approach called **progressive enhancement**. The media queries can be within your CSS file or within the <link> element; the later requires more HTTP requests but results in more manageable CSS files.

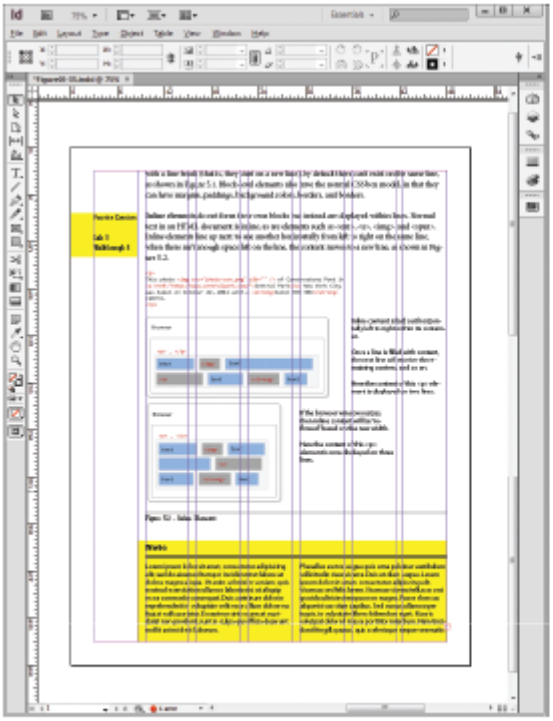
2.13 CSS Frameworks

A **CSS framework** is a precreated set of CSS classes or other software tools that make it easier to use and work with CSS. They are two main types of CSS framework: grid systems and CSS preprocessors.

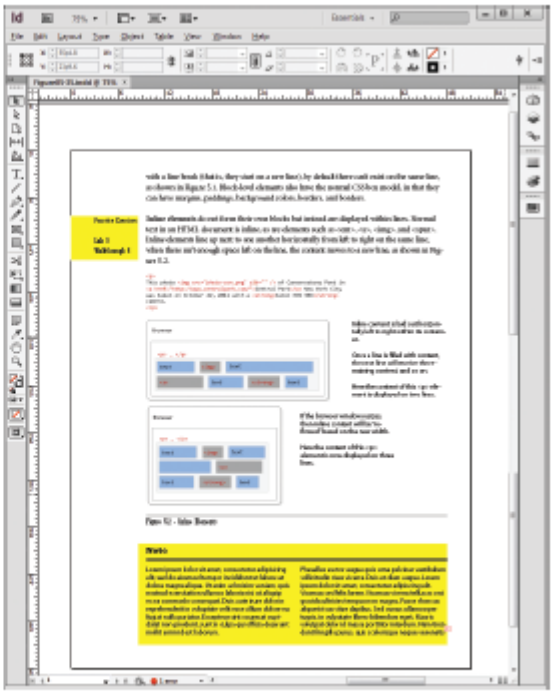
Grid Systems

Grid systems make it easier to create multicolumn layouts. There are many CSS grid systems; some of the most popular are Bootstrap (**twitter.github.com/bootstrap**), Blueprint (**www.blueprintcss.org**), and 960 (**960.gs**).

The most important of these capabilities is a grid system. Print designers typically use grids as a way to achieve visual uniformity in a design. In print design, the very first thing a designer may do is to construct, for instance, a 5- or 7- or 12-column grid in a page layout program. The rest of the document, whether it be text or graphics, will be aligned and sized according to the grid, as shown below.



Most page design begins with a grid. In this case, a seven-column grid is being used to layout page elements in Adobe InDesign.



Without the gridlines visible, the elements on the page do not look random, but planned and harmonious.

CSS frameworks provide similar grid features. The 960 framework uses either a 12- or 16-column grid. Bootstrap uses a 12-column grid. Blueprint uses a 24-column grid. The grid is constructed using elements with classes defined by the framework. The HTML elements for the rest of the site are then placed within these elements.

Eg –

```
<head>
<link rel="stylesheet" href="reset.css" />
<link rel="stylesheet" href="text.css" />
<link rel="stylesheet" href="960.css" />
</head>
<body>
<div class="container_12">
<div class="grid_2">
left column
</div>
<div class="grid_7">
main content
</div>
<div class="grid_3">
right column
</div>
<div class="clear"></div>
</div>
</body>
```

The above code creates a three column layout similar as shown in the above figure. The frameworks allow columns to be nested, making it quite easy to construct the most complex of layouts.

CSS Preprocessors

CSS preprocessors are tools that allow the developer to write CSS that takes advantage of programming ideas such as variables, inheritance, calculations, and functions. A CSS preprocessor is a tool that takes code written in some type of preprocessed language and then converts that code into normal CSS.

The advantage of a CSS preprocessor is that it can provide additional functionalities that are not available in CSS. One of the best ways to see the power of a CSS preprocessor is with colors. Most sites make use of some type of color scheme, perhaps four or five colors. Many items will have the same color.

As shown in the below figure, the background color of the .box class, the text color in the <footer> element, the border color of the <fieldset>, and the text color for placeholder text within the <textarea> element, might all be set to #796d6d. The trouble with regular CSS is that when a change needs to be made, then some type of copy and replace is necessary, which always leaves the possibility that a change might be made to the wrong elements. Similarly, it is common for different site elements to have similar CSS formatting, for instance, different boxes to have the same padding.

In a programming language, a developer can use variables, nesting, functions, or inheritance to handle duplication and avoid copy-and-pasting and search-and-replacing. CSS preprocessors such as LESS, SASS, and Stylus provide this type of functionality.

```
$colorSchemeA: #796d6d;
$colorSchemeB: #9c9c9c;
$paddingCommon: 0.25em;


footer {
  background-color: $colorSchemeA;
  padding: $paddingCommon * 2;
}


@mixin rectangle($colorBack, $colorBorder) {
  border: solid 1pt $colorBorder;
  margin: 3px;
  background-color: $colorBack;
}

fieldset {
  @include rectangle($colorSchemeB, $colorSchemeA);
}

.box {
  @include rectangle($colorSchemeA, $colorSchemeB);
  padding: $paddingCommon;
}
```

SASS source file, e.g. `source.scss`

This example uses SASS (Syntactically Awesome Stylesheets). Here three variables are defined.

You can reference variables elsewhere. SASS also supports math operators on its variables.

A mixin is like a function and can take parameters. You can use mixins to encapsulate common styling.

A mixin can be referenced/called and passed parameters.

The processor is some type of tool that the developer would run.

```
footer {
  padding: 0.50em;
  background-color: #796d6d;
}

fieldset {
  border: solid 1pt #796d6d;
  margin: 3px;
  background-color: #9c9c9c;
}

.box {
  border: solid 1pt #9c9c9c;
  margin: 3px;
  background-color: #796d6d;
  padding: 0.25em;
}
```

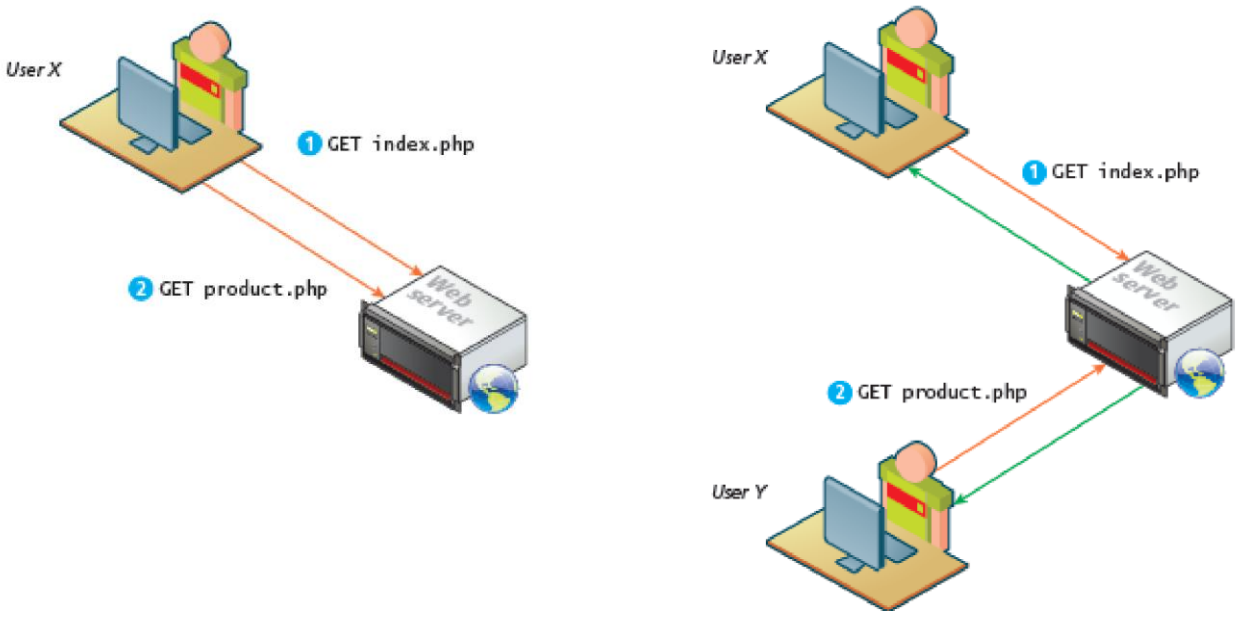
Generated CSS file, e.g., `styles.css`

The output from the processor is a normal CSS file that would then be referenced in the HTML source file.

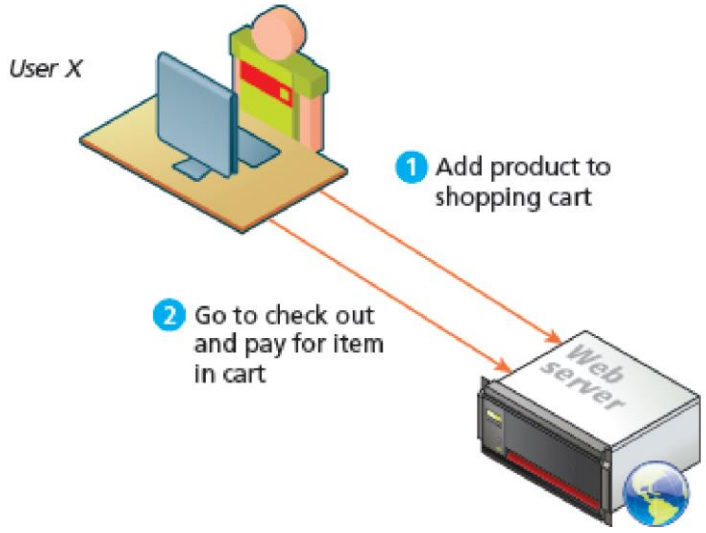
Module V
Managing State, jQuery, XML & WebServices

5.1 The Problem of State in Web Applications

- A problem that is unique in web development is the sharing of information among the request. The information obtained from the first request must be saved for the next request.
- In Single-user desktop applications there is no such problem, as all information is stored in memory and can be easily accessed throughout the application. But a web application consists of a series of disconnected HTTP requests to a web server where each request for a server page is a request to run a separate program on server.



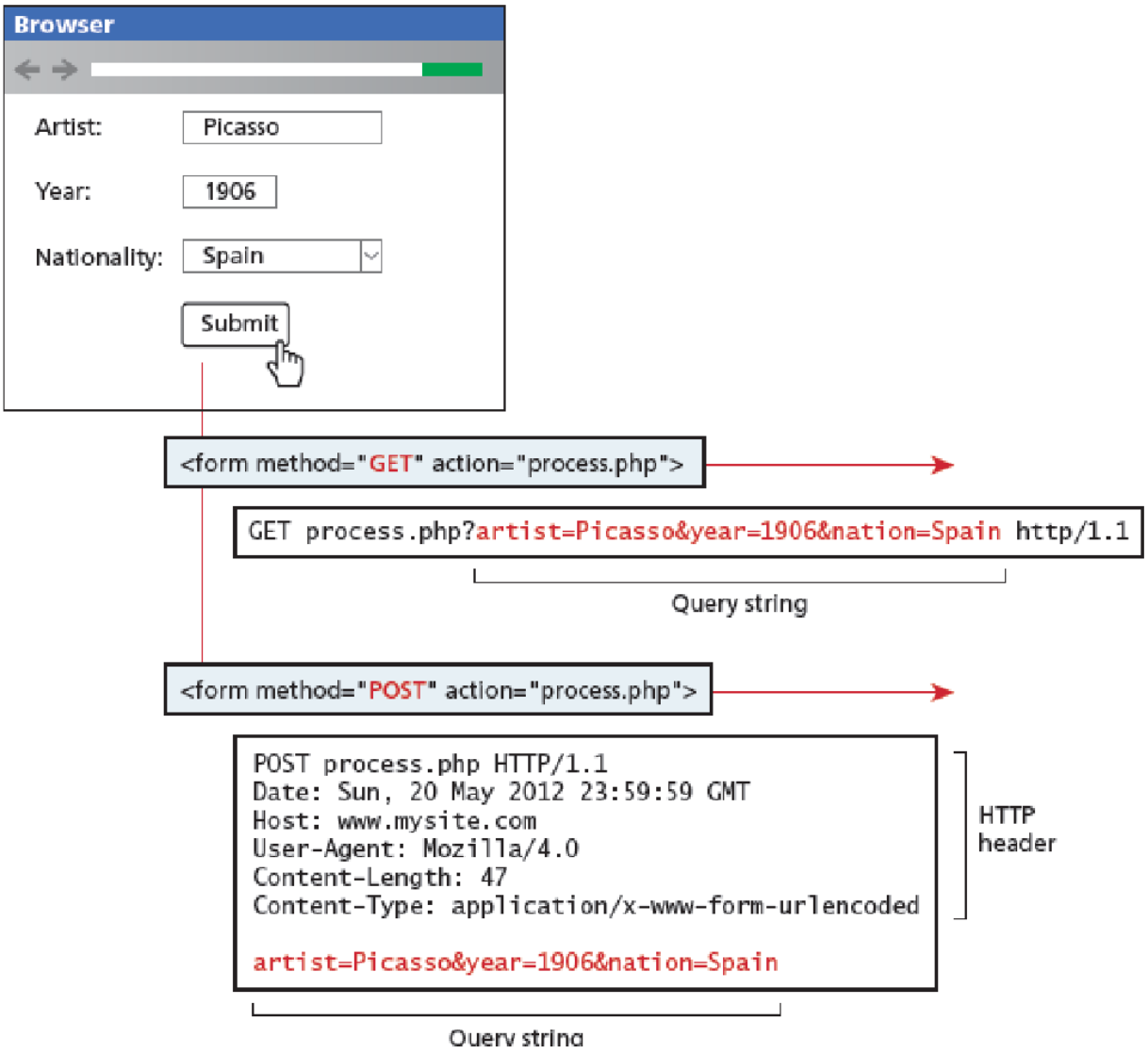
- The web server sees only requests. The HTTP protocol does not, without programming intervention, distinguish two requests by one source from two requests from two different sources.
- While the HTTP protocol disconnects the user's identity from the requests, there are many occasions when we want the web server to connect requests together.
- Consider the scenario of a web shopping cart, the user (and the website owner) most certainly wants the server to recognize that the request to add an item to the cart and the subsequent request to check out and pay for the item in the cart are connected to the same individual.



In HTTP, we can pass information using:
■ Query strings
■ Cookies

5.2 Passing Information via Query Strings

A web page can pass query string information from the browser to the server using one of the two methods: a query string within the URL (GET) and a query string within the HTTP header (POST).



5.3Passing Information via the URL Path

- While query strings are a vital way to pass information from one page to another, they become long and complicated. There is some dispute about whether dynamic URLs (i.e., ones with query string parameters) or static URLs are better.
- Users prefer static URL, but dynamic URLs are essential part of web application development to sent informations. The solution is to rewrite the dynamic URL into a static one. This process is commonly called **URL rewriting**.
- The URL is rewritten differently in different servers. Like, instead of using the querystring in URL, the relevant information is sent in the folder path or filename format.

Eg –
www.somedomain.com/DisplayArtist.php?artist=16
may be to rewritten as:
www.somedomain.com/artists/16.php

http://www.1st-art-gallery.com/Raphael?La-Donna-Velata = 1516
is rewritten as,
http://www.1st-art-gallery.com/Raphael/La-Donna-Velata-1516.html

The file name extension is being rewritten to .html, to make the URL friendlier. These are not static HTML file.

URL Rewriting in Apache and Linux
Depending on web development platform, there are different ways to implement URL rewriting. On web servers running Apache, the solution is to use the mod_rewrite module in Apache along with the .htaccess file.

The mod_rewrite module uses a rule-based rewriting engine that uses regular expressions to change the URLs, so that the requested URL can be mapped or redirected to another URL.

5.4 Cookies

- **Cookies** are a client-side approach for preserving state information. They are name=value pairs that are saved within one or more text files that are managed by the browser. These pairs accompany both server requests and responses within the HTTP header.
- Cookies cannot contain viruses. The user-related information is preserved on the user's computer and is managed by the user's browser.
- Cookies are not associated with a specific page. It is associated with the page's domain.
- The browser and server will exchange cookie information, when using the same domain.
- It is mainly used to maintain the continuity of information, after some time in a web application. One typical use of cookies in a website is to "remember" the visitor, so that the server can customize the site for the user.
- Some sites will use cookies as part of their shopping cart implementation so that items added to the cart will remain there even if the user leaves the site and then comes back later.
- Cookies are also frequently used to keep track of whether a user has logged into a site.

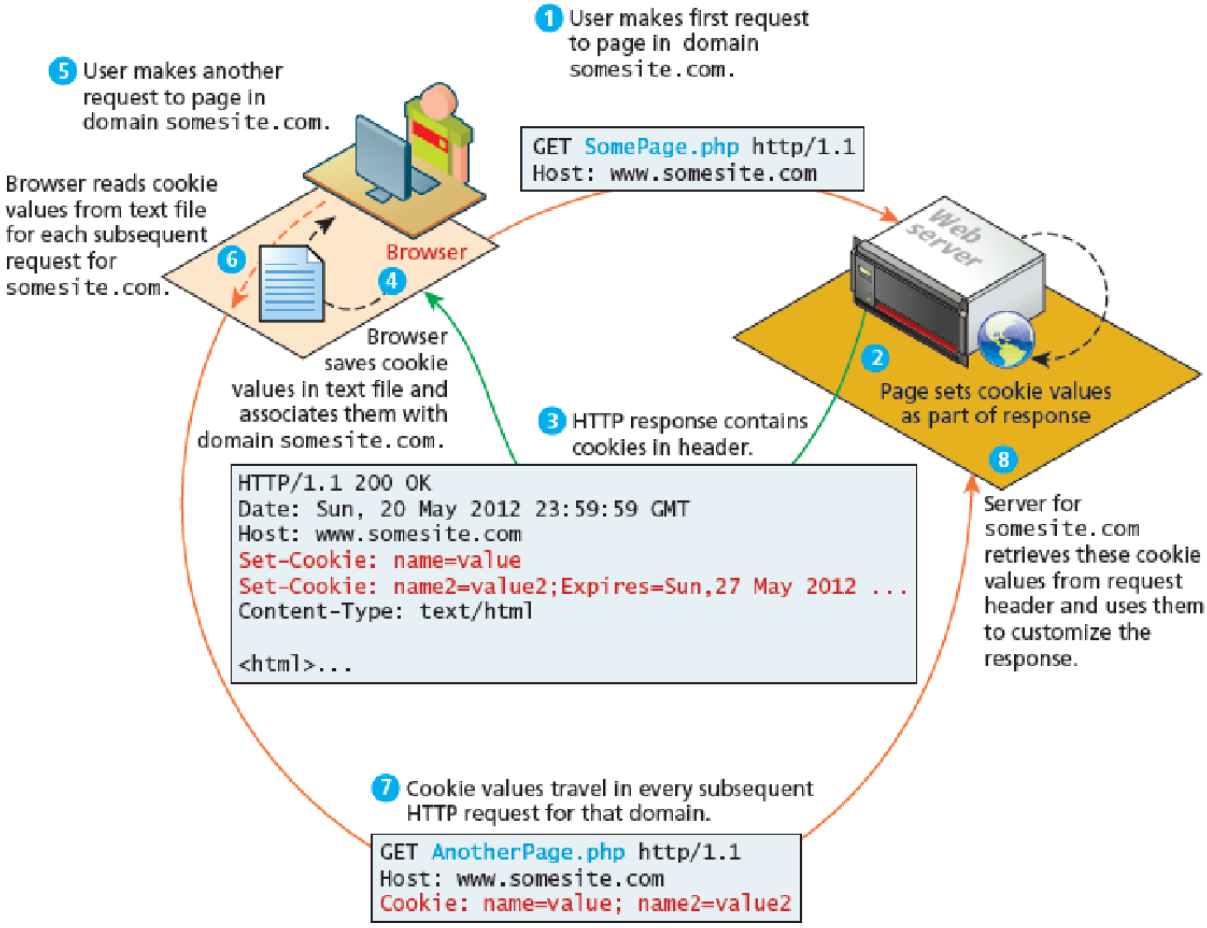
How Do Cookies Work?

- While cookie information is stored and retrieved by the browser, the information in a cookie travels within the HTTP header. There are limitations to the amount of information that can be stored in a cookie (around 4K) and to the number of cookies for a domain (for instance, Internet Explorer 6 limited a domain to 20 cookies).
- Cookies can expire. Expiry time of the cookie is configured during the time of creation. The browser will delete cookies that are beyond their expiry date. If a cookie does not have an expiry date specified, the browser will delete it when the browser closes.

There are two types of cookies: session cookies and persistent cookies.

- A **session cookie** has no expiry stated and thus will be deleted at the end of the user browsing session.
- **Persistent cookies** have an expiry date specified; they will persist in the browser's cookie file until the expiry date occurs, after which they are deleted.

The most important limitation of cookies is that the browser may be configured to refuse the usage of cookies. As a consequence, sites that use cookies should not depend on their availability for critical features. Similarly, the user can also delete cookies or even tamper with the cookies, which may lead to some serious problems if not handled.



Using Cookies

Like any other web development technology, PHP provides mechanisms for writing and reading cookies. Cookies in PHP are *created* using the setcookie() function and are *retrieved* using the $_COOKIES superglobal associative array.

The setcookie() function sets the value to a particular key and also sets the expiry time.

Eg –

Setcookie(key,value, expiry time)

Writing a cookie

```
<?php
// add 1 day to the current time for expiry time
$expiryTime = time()+60*60*24;
// create a persistent cookie
$name = "Username";
$value = "Ricardo";
setcookie($name, $value, $expiryTime);
?>
```

Reading a cookie

```
<?php
if( !isset($_COOKIE['Username']) ) {
        //no valid cookie found
}
else {
        echo "The username retrieved from the cookie is:";
        echo $_COOKIE['Username'];
}
?>
```

Before reading a cookie, we must check to ensure that the cookie exists. In PHP, if the cookie has expired, then the client's browser would not send anything along with the request, and so the $_COOKIE array would be blank

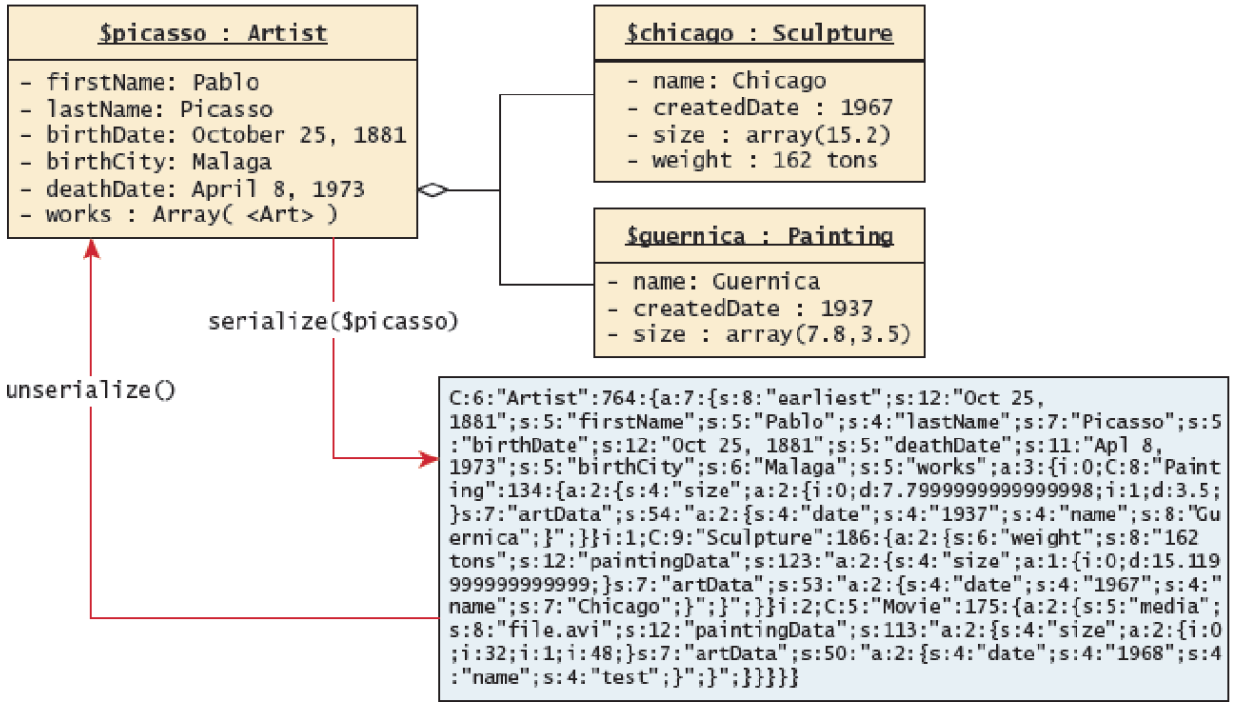
Persistent Cookie Best Practices

- Persistent cookie store the information, which will be retained in the cookie for a long period of time.
- Many sites provide a "Remember Me" checkbox on login forms, which relies on the use of a persistent cookie. This login cookie would contain the user's username but not the password. But cookies should not be used for stored important information required for site's operation.
- Another common use of cookies would be to use them to store user preferences. For instance, some sites allow the user to choose their preferred site color scheme or their country of origin. In these cases, saving the user's preferences in a cookie will make for a more contented user, but if the user's browser does not accept cookies, even then the site will work. But the user will have to reselect his or her preferences again.
- Another use of cookies is to track a user's browsing behavior on a site. This information can be used by the site administrator as an analytic tool to help understand how users navigate through the site.

5.5 Serialization

Serialization is the process of taking a complicated object and converting it to zeros and ones for either storage or transmission. Later that sequence of zeros and ones can be reconstituted into the original object.

In PHP objects can easily be reduced down to a binary string using the serialize() function. The resulting string is a binary representation of the object and therefore may contain unprintable characters. The string can be reconstituted back into an object using the unserialize() method.



Objects must implement the Serializable interface which requires adding implementations for serialize() and unserialize() to any class that implements this interface.

The Serializable interface

```
interface Serializable {
        /* Methods */
        public function serialize();
        public function unserialize($serialized);
}
```

The Student class must be modified to implement the Serializable interface by adding the implements keyword to the class definition and adding implementations for the two methods.

```
class Student implements Serializable {
//...
```

```
// Implement the Serializable interface methods
public function serialize() {
// use the built-in PHP serialize function
return serialize(
array("name" => $this->name,
"usn" => $this->usn,
"address" => $this->address,
"average" => $this->avg
);
);
}

public function unserialize($data) {
// use the built-in PHP unserialize function
$data = unserialize($data);
$this->name = $data['name'];
$this->usn = $data['usn'];
$this->address = $data['address'];
$this->avg= $data['average'];
}
//...
}
```

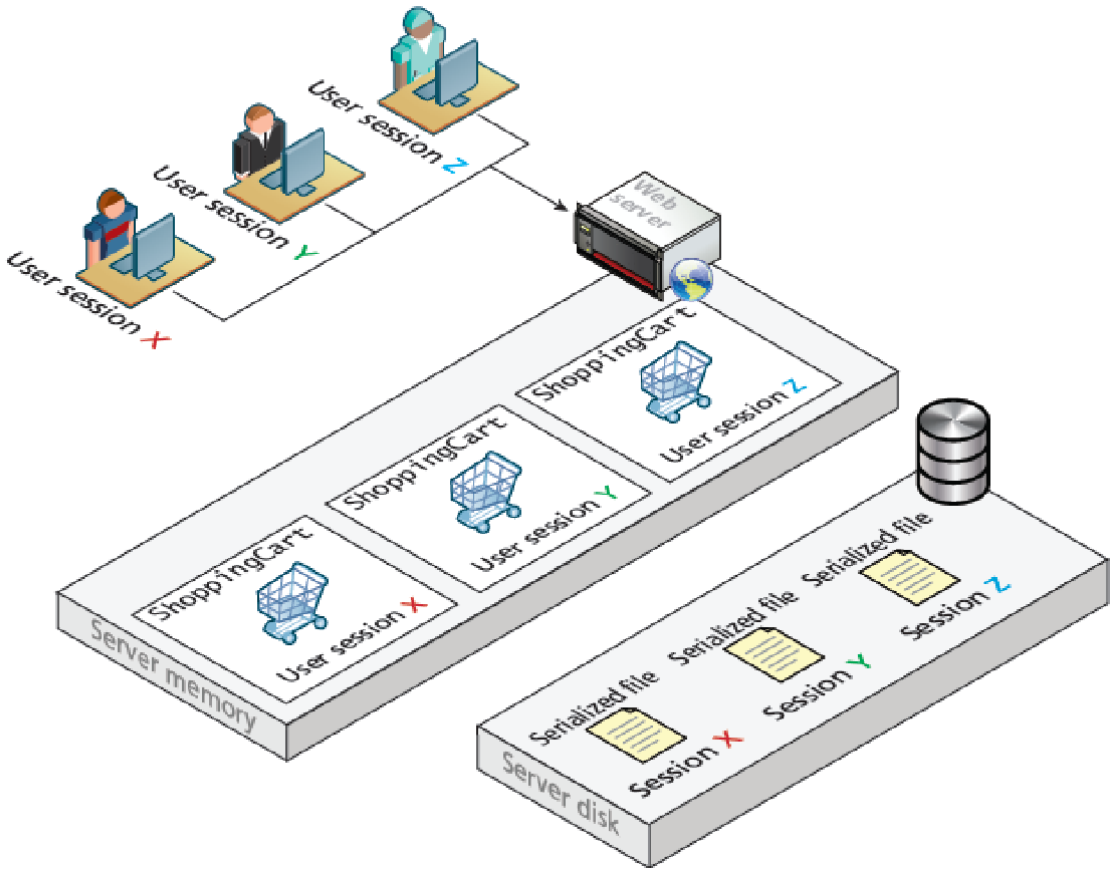
The serialized data can easily be used to reconstruct the object by passing it to unserialize(). In the above example $data is the serialized data. It can be used to recreate the object.
$s1Clone = unserialize($data);

Application of Serialization
Since each request from the user requires objects to be reconstructed, using serialization to store and retrieve objects, it is a rapid way to maintain information between requests. At the end of a request the information is stored in a serialized form, and then the next request would begin by deserializing it to reestablish the previous state.

5.6 Session State

Session state is a server-based state mechanism that allows web applications to store and retrieve objects of any type for each unique user session. That is, each browser session has its own session state stored as a serialized file on the server, which is deserialized and loaded into memory as needed for each request.

As server storage is a finite resource, objects loaded into memory are released when the request completes, making room for other requests and their session objects. This means there can be more active sessions on disk than in memory at any one time. Session state is ideal for storing more complex objects or data structures that are associated with a user session.



In PHP, session state is available to the developer as a superglobal associative array, like the $_GET, $_POST, and $_COOKIE arrays. It can be accessed via the $_SESSION variable.

```
<?php
session_start();
if ( isset($_SESSION['user']) ) {
// User is logged in
}
else {
// No one is logged in (guest)
}
?>
```

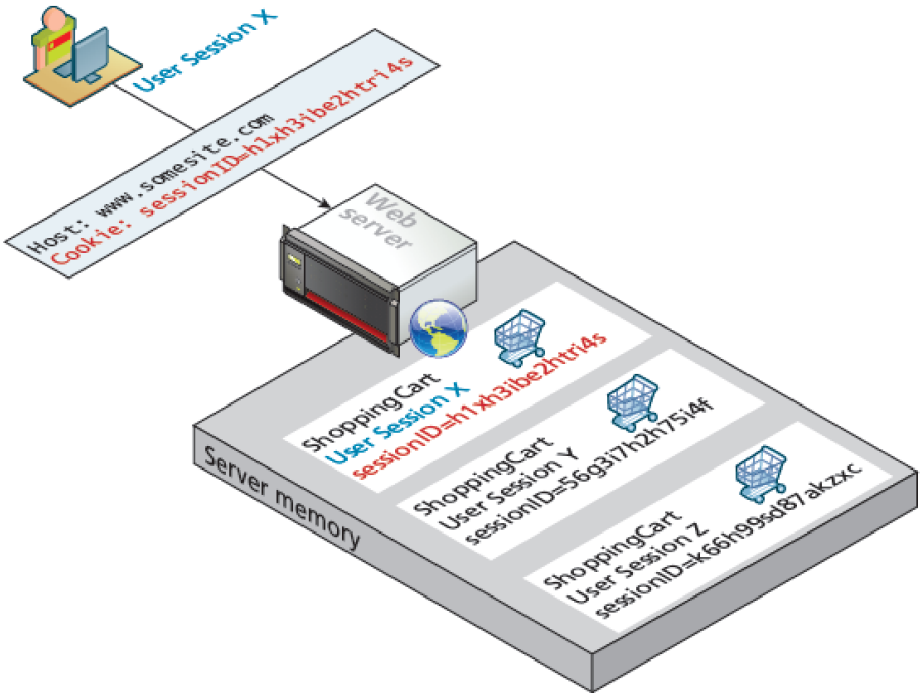
Session information gets deleted after the session. As a result, it should be checked if an item retrieved from session state still exists before using the retrieved object. If the session object does not yet exist (either because it is the first time the user has requested it or because the session has timed out), one might generate an error or redirect to another page.

```
<?php
session_start();
// always check for existence of session object before accessing it
```

```
if ( !isset($_SESSION["Cart"]) ) {
//session variables can be strings, arrays, or objects, but
// smaller is better
$_SESSION["Cart"] = new ShoppingCart();
}
$cart = $_SESSION["Cart"];
?>
```

How Does Session State Work?

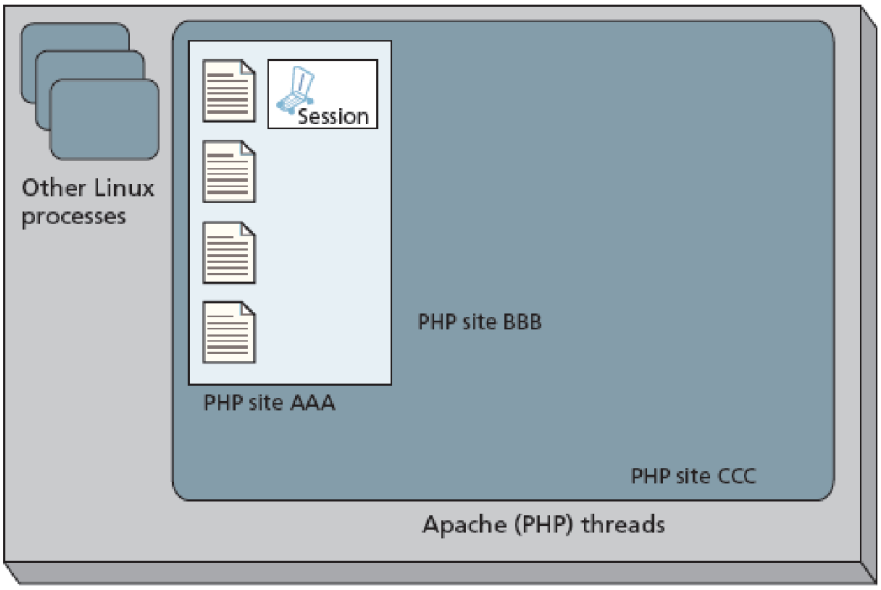
The session state works within the same HTTP context as any web request. Sessions in PHP are identified with a unique session ID. In PHP, this is a unique 32-byte string that is by default transmitted back and forth between the user and the server via a session cookie.



After generating or obtaining of a session ID for a new session, PHP assigns an initially empty dictionary-style collection that can be used to hold any state values for this session. When the request processing is finished, the session state is saved to some type of state storage mechanism, called a session state provider. Finally, when a new request is received for an already existing session, the session's dictionary collection is filled with the previously saved session data from the session state provider.

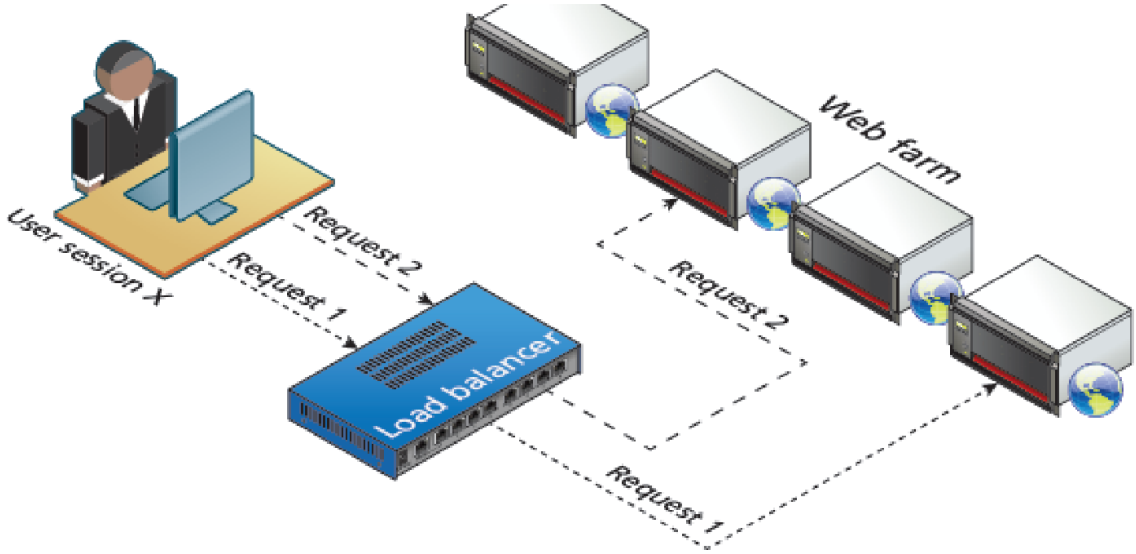
Session Storage and Configuration

The session states can be saved in files. The decision to save sessions to files than in memory resolves the issue of memory usage that can occur on shared hosts. For each application, server memory stores not only session information, but pages being executed, and caching information.



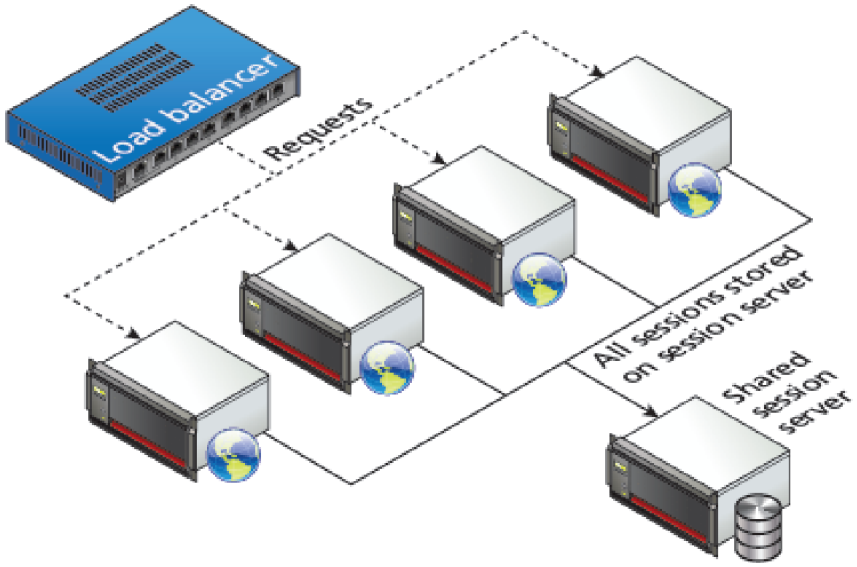
On a busy server hosting multiple sites, it is common for the Apache application process to be restarted on occasion. If the sessions were stored in memory, the sessions would all expire. But as session states are stored into files, they can be instantly recovered as though nothing happened. One disadvantage of storing the sessions in files is degradation in performance compared to memory storage.

Higher-volume web applications often run in an environment in which **multiple web servers** are servicing requests. Each incoming request is forwarded by a load balancer to any one of the available servers in the farm.



In such a situation the in-process session state (session state stored in a server) will not work, since one server may service one request for a particular session, and then a completely different server may service the next request for that session. There are effectively two categories of solution to this problem.

1. Configure the load balancer to be “session aware” and relate all requests using a session to the same server.
2. Use a shared location to store sessions, either in a database, or some other shared session state mechanism.



5.7 HTML5 Web Storage

Web storage is a JavaScript API introduced in HTML5. It is a replacement (or perhaps supplement) to cookies.
Web storage is managed by the browser; and it is not transported to and from the server with every request and response. Web storage is not limited to the 4K size barrier of cookies; browsers are allowed to store more than a limit of 5MB per domain.

There were two types of global web storage objects: localStorage and sessionStorage. The localStorage object is for saving information that will persist between browser sessions. The sessionStorage object is for information that will be lost once the browser session is finished.

Using Web Storage

The below snippet in JavaScript is for writing information to web storage. This mechanism uses JavaScript. There are two ways to store values in web storage: using the setItem() function, or using the property shortcut (e.g., sessionStorage.FavoriteArtist).

```
<form ... >
<h1>Web Storage Writer</h1>
<script language="javascript" type="text/javascript">
if (typeof (localStorage) === "undefined" ||typeof (sessionStorage) === "undefined") {
alert("Web Storage is not supported on this browser...");
}
else {
sessionStorage.setItem("TodaysDate", new Date());
```

```
sessionStorage.highestaverage = 92;

localStorage.name = "Ram";
document.write("web storage modified");
}
</script>
</form>
```

The below snippet shows the process of reading from web storage. The difference between sessionStorage and localStorage that if you close the browser after writing and then run the code, only the localStorage item will still contain a value.

```
<form id="form1" runat="server">
<h1>Web Storage Reader</h1>
<script language="javascript" type="text/javascript">
if (typeof (localStorage) === "undefined" || typeof (sessionStorage) === "undefined") {
alert("Web Storage is not supported on this browser...");
}
else {
var today = sessionStorage.getItem("TodaysDate");
var a = sessionStorage. highestaverage;
var user = localStorage.name;
document.write("date saved=" + today);
document.write("<br/>Highest average=" + a);
document.write("<br/>user name = " + user);
}
</script>
</form>
```

Why Would We Use Web Storage?

Cookies have the disadvantage of being limited in size, potentially disabled by the user, vulnerable to security attacks, and for every single request and response the cookie value is sent.

But the advantage of sending cookies with every request and response is also their main advantage, it is easy to implement data sharing between the client browser and the server.

Transporting the information within web storage between the server and browser is a complicated affair involving the construction of a web service on the server and then using asynchronous communication via JavaScript to push the information to the server.

Web storage mechanism is not a cookie replacement but acts as a local cache for relatively static items available to JavaScript. One practical use of web storage is to store static content downloaded asynchronously such as XML or JSON from a web service in web storage, thus reducing server load for subsequent requests by the session.

5.8 Caching

Caching is storing the information. Browser uses caching to speed up the user experience by using locally stored versions of images and other files rather than re-requesting the files from the server.

Caching on server side is necessary, because every time a PHP page is requested, it must be fetched, parsed, and executed by the PHP engine, and the end result is HTML that is sent back to the requestor. For the typical PHP page, this might also involve numerous database queries and processing to build. If this page is being served thousands of times per second, the dynamic generation of that page may become unsustainable.

One way to address this problem is to **cache** the generated markup in server memory so that subsequent requests can be served from memory rather than from the execution of the page. There are two basic strategies to caching web applications –

1. **page output caching**, which saves the rendered output of a page or user control and reuses the output instead of reprocessing the page when a user requests the page again.
2. **application data caching**, which allows the developer to programmatically cache data.

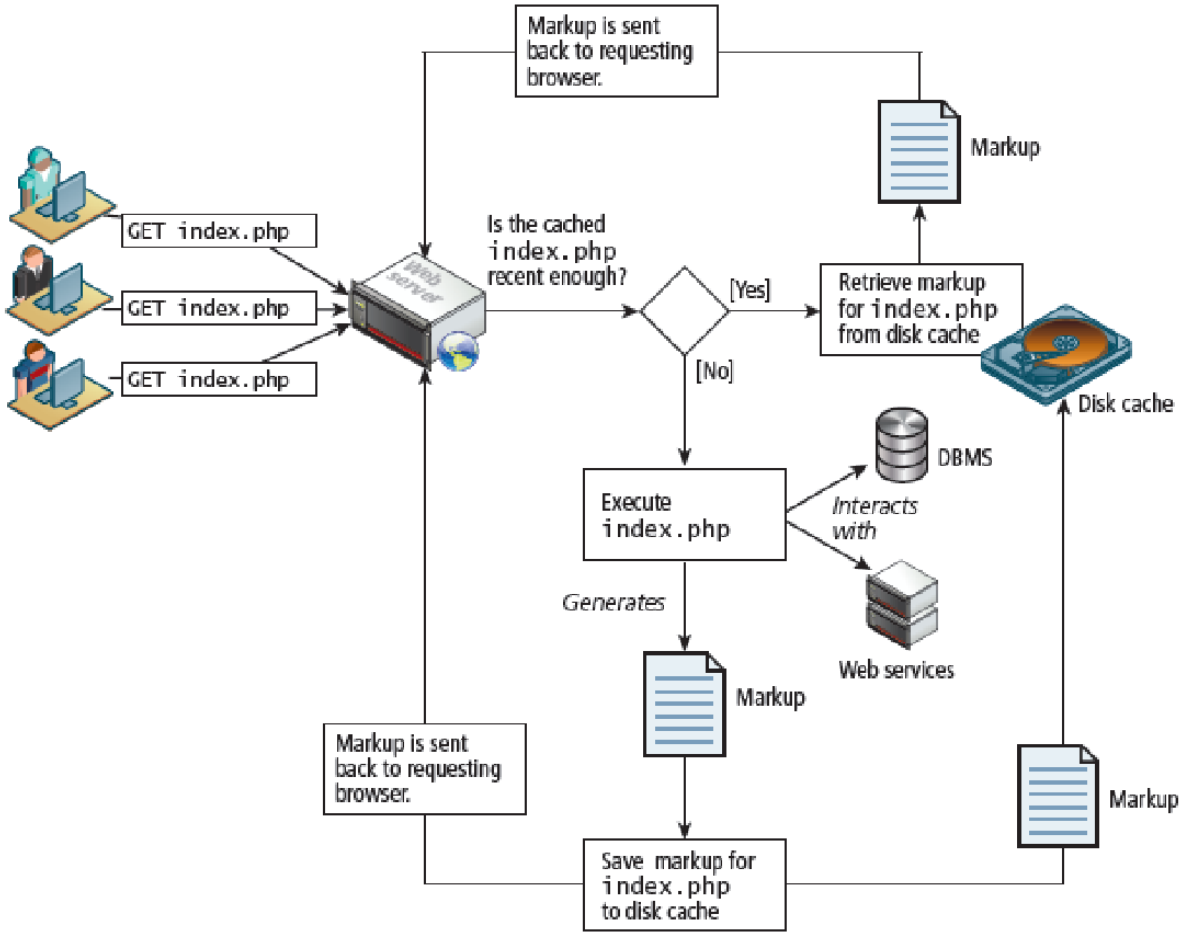
Page Output Caching

In this type of caching, the contents of the rendered PHP page(page ready to display on screen) are written to disk for fast retrieval. This can be particularly helpful because it allows PHP to send a page response to a client without going through the entire page processing life cycle again.

Page output caching is especially useful for pages whose content does not change frequently but which require significant processing to create. There are two models for page caching: full page caching and partial page caching.

In full page caching, the entire contents of a page are cached. In partial page caching, only specific parts of a page are cached while the other parts are dynamically generated in the normal manner.

Page caching is not included in PHP by default, it has to be attached by using add-ons such as Alternative PHP Cache (open source) and Zend (commercial).



Application Data Caching

One of the biggest drawbacks with page output caching is that performance gains will only be had if the entire cached page is the same for numerous requests.

In application data caching a page will programmatically place commonly used collections of data into cache memory, and then other pages that also need that same data can use the cache version rather than re-retrieve it from its original location. The data that require time-intensive queries from the database or web server are stored in cache memory.

While the default installation of PHP does not come with an application caching ability, a widely available free PECL extension called memcache is widely used to provide this ability. memcache is not used to store large collections. The size of the memory cache is limited. If too many things are placed in it, its performance advantages will be lost as items get paged in and out. Hence it is used to store small collections of data that are frequently accessed on multiple pages.

```
<?php
// create connection to memory cache
$memcache = new Memcache;
```

```
$memcache->connect('localhost', 11211) or die ("Could not connect to memcache server");
$cacheKey = 'topCountries';
/* If cached data exists retrieve it, otherwise generate and cache
it for next time */
if ( ! isset($countries = $memcache->get($cacheKey)) ) {
// since every page displays list of top countries as links
// we will cache the collection
// first get collection from database
$cgate = new CountryTableGateway($dbAdapter);
$countries = cgate->getMostPopular();
// now store data in the cache (data will expire in 240 seconds)
$memcache->set($cacheKey, $countries, false, 240)
or die ("Failed to save cache data at the server");
}
// now use the country collection
displayCountryList($countries);
?>
```

XML Processing & Web Services

5.9 XML Overview

XML (eXtensible Markup Language) is a markup language. It can be used to mark up any type of data. XML is used not only for web development but is also used as a file format in many nonweb applications. One of the key benefits of XML data is that as plain text, it can be read and transferred between applications and different operating systems as well as being human-readable and understandable as well. XML is also used in the web context as a format for moving information between different systems.

XML is not only used on the web server and to communicate asynchronously with the browser, but is also used as a data interchange format for moving information between systems.

Well-Formed XML

Syntax rules for XML -

- Element names are composed of any of the valid characters (most punctuation symbols and spaces are not allowed) in XML.
- Element names can't start with a number.
- There must be a single-root element. A **root element** is one that contains all the other elements; for instance, in an HTML document, the root element is <html>.
- All elements must have a closing element (or be self-closing).
- Elements must be properly nested.
- Elements can contain attributes.
- Attribute values must always be within quotes.
- Element and attribute names are case sensitive.

Sample XML document –

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
        <painting id="290">
        <title>Balcony</title>
        <artist>
        <name>Manet</name>
        <nationality>France</nationality>
        </artist>
        <year>1868</year>
        <medium>Oil on canvas</medium>
        </painting>

        <painting id="192">
        <title>The Kiss</title>
        <artist>
        <name>Klimt</name>
        <nationality>Austria</nationality>
        </artist>
        <year>1907</year>
        <medium>Oil and gold on canvas</medium>
        </painting>

        <painting id="139">
        <title>The Oath of the Horatii</title>
        <artist>
        <name>David</name>
        <nationality>France</nationality>
        </artist>
        <year>1784</year>
        <medium>Oil on canvas</medium>
        </painting>
</art>
```

In this example, the root element is called <art>. Some type of XML parser is required to verify that an XML document is well formed.

Valid XML

A **valid XML** document is one that is well formed and whose element and content conform to the rules of either its document type definition (DTD) or its schema.

DTDs were the original way for an XML parser to check an XML document for validity. They tell the XML parser which elements and attributes to expect in the document as well as the order and nesting of those elements. A DTD can be defined within an XML document or within an external file.

The DTD for the above XML file is –

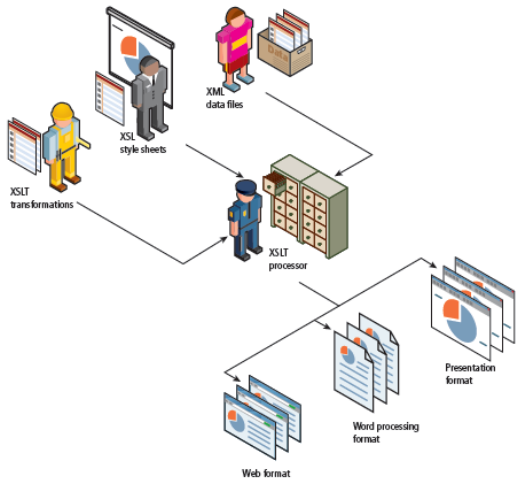
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE art [
<!ELEMENT art (painting*)>
<!ELEMENT painting (title,artist,year,medium)>
<!ATTLIST painting id CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT artist (name,nationality)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT nationality (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT medium (#PCDATA)>
]>
```

The main drawback with DTDs is that they can only validate the existence and ordering of elements (and the existence of attributes). They provide no way to validate the values of attributes or the textual content of elements. For this type of validation, one must instead use XML schemas, which have the added advantage of using XML syntax.

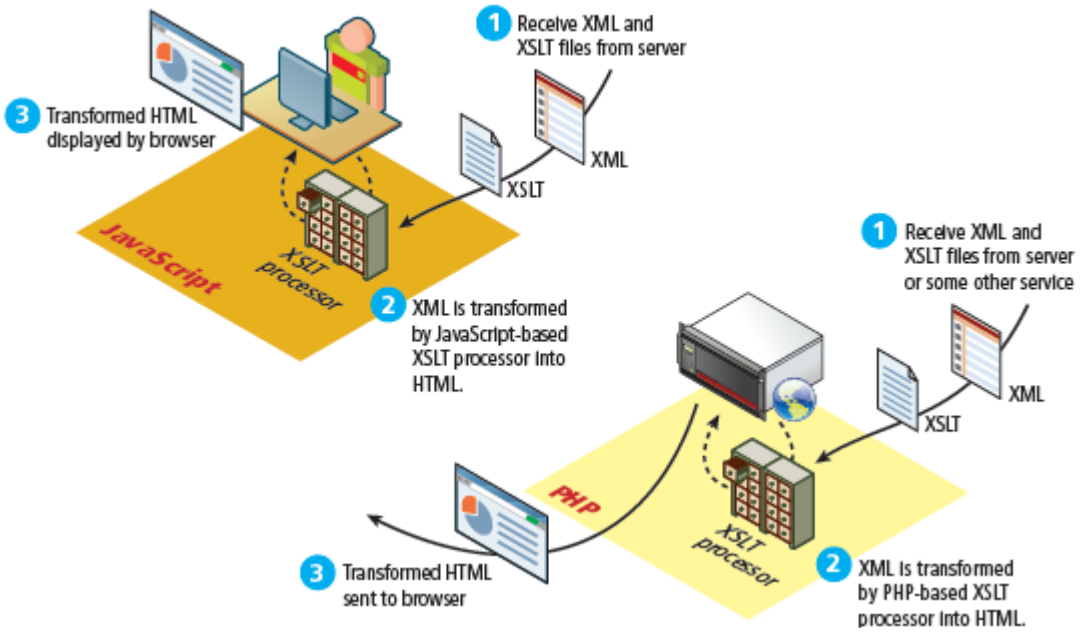
The disadvantage of schema is long-winded and harder for humans to read and comprehend; hence they are usually created with tools.

XSLT

XSLT stands for XML Stylesheet Transformations. XSLT is an XML-based programming language that is used for transforming XML into other document formats. The most common translation is the conversion of XML to HTML.



Usage of XSLT



The below example shows an XSLT document that convert the XML into HTML list -

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xsl:version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/1999/xhtml">
<body>
<h1>Catalog</h1>
<ul>
<xsl:for-each select="/art/painting">
<li>
<h2><xsl:value-of select="title"/></h2>
<p>By: <xsl:value-of select="artist/name"/><br/>
Year: <xsl:value-of select="year"/>
[<xsl:value-of select="medium"/>]</p>
</li>
</xsl:for-each>
</ul>
</body>
</html>
```

The strings within the **select attribute**: these are XPath expressions, which are used for selecting specific elements within the XML source document. The <xsl:for-each> element is one of the iteration constructs within XSLT. In this example, it iterates through each of the <painting> elements.

XPath

XPath is a standardized syntax for **searching an XML document** and for navigating to elements within the XML document. XPath is used for programmatically manipulating XML document in PHP and other languages.

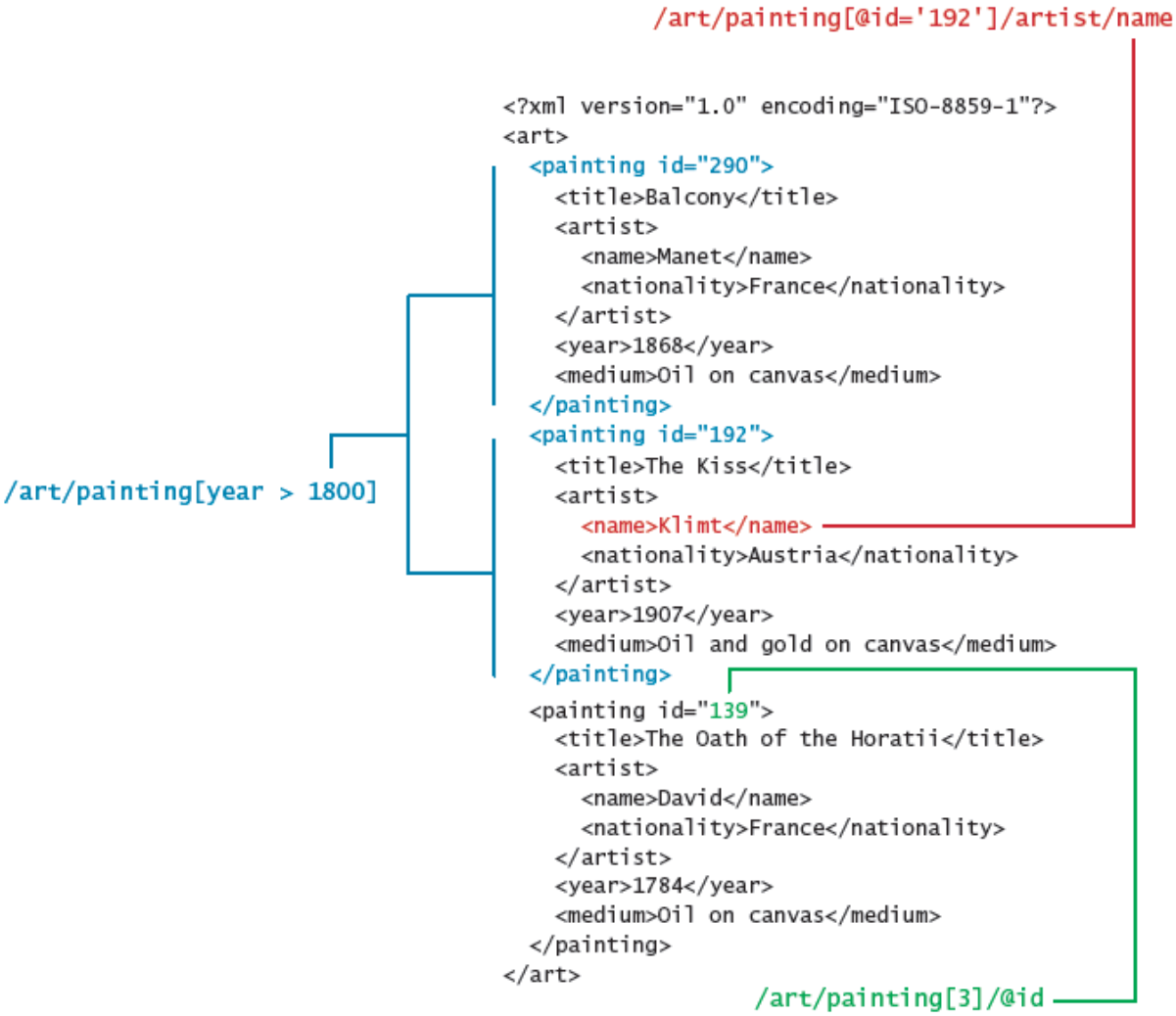
XPath uses a syntax that is similar to accessing directories. For instance, to select all the painting elements in the below XML file, XPath expression is - /art/painting. As in file representation, the forward slash is used to separate elements contained within other elements. An XPath expression beginning with a forward slash is an absolute path beginning with the start of the document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
        <painting id="290">
        <title>Balcony</title>
        <artist>
        <name>Manet</name>
        <nationality>France</nationality>
        </artist>
        <year>1868</year>
        <medium>Oil on canvas</medium>
        </painting>

        <painting id="192">
        <title>The Kiss</title>
        <artist>
        <name>Klimt</name>
        <nationality>Austria</nationality>
        </artist>
        <year>1907</year>
        <medium>Oil and gold on canvas</medium>
        </painting>

        <painting id="139">
        <title>The Oath of the Horatii</title>
        <artist>
        <name>David</name>
        <nationality>France</nationality>
        </artist>
        <year>1784</year>
        <medium>Oil on canvas</medium>
        </painting>
</art>
```

In XPath terminology, an XPath expression returns zero, one, or many XML nodes. In XPath, a **node** generally refers to an XML element. From a node, its attributes, textual content, and child nodes can be specifically extracted.



The XPath expression: '/art/painting[@id='192']/artist/name' - selects the <name> element within the <artist> element for the <painting> element with an id attribute of 192.
Here square brackets are used to specify a criteria expression at the current path node, in the above example is /art/painting (i.e., each painting node is examined to see if its id attribute is equal to the value 192).

The XPath expression: '/art/painting[3]/@id' - selects the id attribute of the element <painting> whose index is 3. XPath expressions begin with one and not zero. Attributes are identified in XPath expressions by being prefaced by the @ character.

The XPath expression: '/art/painting[year >1800]' - selects the whole <painting> element whos <year> element value is > 1800.

5.10 XML Processing

XML processing in PHP, JavaScript, and other modern development environments is divided into two basic styles:

1. The **in-memory approach**, which involves reading the entire XML file into memory into some type of data structure with functions for accessing and manipulating the data.
2. The **event or pull approach**, which lets you pull in just a few elements or lines at a time, thereby avoiding the memory load of large XML files.

XML Processing in JavaScript

All modern browsers have a built-in XML parser and their JavaScript implementations support an **in-memory** XML DOM API, which loads the entire document into memory where it is transformed into a hierarchical tree data structure.

The DOM functions such as getElementById(), getElementsByTagName(), and createElement() are used to access and manipulate the data.

For instance, the below code shows the loading of an XML document into an XML DOM object, and it displays the id attributes of the <painting>elements as well as the content of each painting's <title> element

```
<script>
// load the external XML file
xmlhttp.open("GET","art.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

// now extract a node list of all <painting> elements
paintings = xmlDoc.getElementsByTagName("painting");

if (paintings) {
        // loop through each painting element
        for (var i = 0; i < paintings.length; i++)
        {
                // display its id attribute
                alert("id="+paintings[i].getAttribute("id"));
                // find its <title> element
                title = paintings[i].getElementsByTagName("title");
                if (title) {
                // display the text content of the <title> element
                alert("title="+title[0].textContent);
                }
        }
}
</script>
```

JavaScript supports a variety of node traversal functions as well as properties for accessing information within an XML node.

alert() function – displays the string in a alert dialog box.
GetAttribute(id) - returns the value of the attribute.
getElementsByTagName(tag) – returns an array of tags with the specified title.
textContent() – returns the html content in the specified tag.

XML Processing in PHP

PHP provides several extensions or APIs for working with XML -

- The **DOM extension** loads the entire document into memory where it is transformed into a hierarchical tree data structure. This DOM approach is a standardized approach. Diffeent languages implementing this approach use relatively similarly named functions/methods for accessing and manipulating the data.

- The **SimpleXML** extension loads the data into an object that allows the developer to access the data via array properties and modifying the data via methods.

- The **XML parser** is an event-based XML extension.

- The **XMLReader** is a read-only pull-type extension that uses a cursor-like approach similar to that used with database processing. The XMLWriter provides an analogous approach for creating XML files.

In general, the SimpleXML and the XMLReader extensions are used to read and process XML content. The SimpleXML approach reads the entire XML file into memory and transforms into a complex object. It is not usually used for processing very large XML files because it reads the entire file into server memory; however, since the file is in memory, it offers fast performance.
Eg –

```
<?php
$filename = 'art.xml';
if (file_exists($filename)) {
        $art = simplexml_load_file($filename);

        // access a single element
        $painting = $art->painting[0];
        echo '<h2>' . $painting->title . '</h2>';
        echo '<p>By ' . $painting->artist->name . '</p>';

        // display id attribute
        echo '<p>id=' . $painting["id"] . '</p>';
} else {
exit('Failed to open ' . $filename);
}
?>
```

simplexml_load_file() function - XML file is transformed into an object.
The elements in the XML document is manipulated using regular PHP object techniques using the -> operator.

The SimpleXML extension is used to only small XML files as the read contents are stored directly to memory. For reading the large XML files, the XMLReader is a used. The XMLReader is sometimes referred to as a pull processor, here it reads a single node at a time, the value in that node is processed, and then the next node is read.

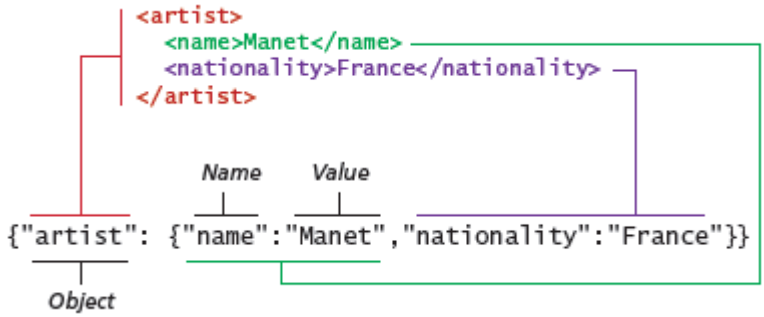
```
$filename = 'art.xml';
if (file_exists($filename)) {
        // create and open the reader
        $reader = new XMLReader();
        $reader->open($filename);

        // loop through the XML file
        while ( $reader->read() ) {
                $nodeName = $reader->name;
                // check  node type
                if ($reader->nodeType == XMLREADER::ELEMENT
                && $nodeName == 'painting') {
                        $id = $reader->getAttribute('id');
                        echo '<p>id=' . $id . '</p>';
                }
                if ($reader->nodeType == XMLREADER::ELEMENT
                && $nodeName =='title') {
                // read the next node to get at the text node
                $reader->read();
                echo '<p>' . $reader->value . '</p>';
        }
        }
} else {
exit('Failed to open ' . $filename);
}
```

5.11 JSON

JSON is a data serialization format, like XML. That is, it is used to represent object data in a text format so that it can be transmitted from one computer to another. Many REST web services encode their returned data in the JSON data format instead of XML. While **JSON** stands for **JavaScript Object Notation**, its use is not limited to JavaScript. It was originally designed to provide a lightweight serialization format to represent objects in JavaScript. While it doesn't have the validation and readability of XML, it has the advantage of generally requiring significantly fewer bytes to represent data than XML.

An example of how an XML data element would be represented in JSON is –
{"artist": {"name":"Manet","nationality":"France"}}



Just like XML, JSON data can be nested to represent objects within objects.

{
"paintings": [
{
"id":290,
"title":"Balcony",
"artist":{
"name":"Manet",
"nationality":"France"
},
"year":1868,
"medium":"Oil on canvas"
},
{
"id":192,
"title":"The Kiss",
"artist":{
"name":"Klimt",
"nationality":"Austria"
},
"year":1907,
"medium":"Oil and gold on canvas"
},
{
"id":139,
"title":"The Oath of the Horatii",
"artist":{
"name":"David",
"nationality":"France"
},
"year":1784,
"medium":"Oil on canvas"
}

```
]
}
```

In general JSON data will have all white space removed to reduce the number of bytes traveling across the network. Square brackets are used to contain the three painting object definitions: this is the JSON syntax for defining an array.

Using JSON in JavaScript

Since the syntax of JSON is the same used for creating objects in JavaScript, it is easy to make use of the JSON format in JavaScript:

```
<script>
var a = {"artist": {"name":"Manet","nationality":"France"}};
alert(a.artist.name + " " + a.artist.nationality);
</script>
```

The JSON information will be contained within a string, and the JSON.parse() function can be used to transform the string containing the JSON data into a JavaScript object:

```
var text = '{"artist": {"name":"Manet","nationality":"France"}}';
var a = JSON.parse(text);
alert(a.artist.nationality);
```

JavaScript also provides a mechanism to translate a JavaScript object into a JSON string:

```
var text = JSON.stringify(artist);
```

Using JSON in PHP

PHP comes with a JSON extension . Converting a JSON string into a PHP object is quite straightforward:

```
<?php
// convert JSON string into PHP object
$text = '{"artist": {"name":"Manet","nationality":"France"}}';
$anObject = json_decode($text);
echo $anObject->artist->nationality;
// convert JSON string into PHP associative array
$anArray = json_decode($text, true);
echo $anArray['artist']['nationality'];
?>
```

The json_decode() function can return either a PHP object or an associative array. Since JSON data is often coming from an external source, we should check for parse errors before using it, which can be done via the json_last_error() function:

```
<?php
```

```
// convert JSON string into PHP object
$text = '{"artist": {"name":"Manet","nationality":"France"}}';
$anObject = json_decode($text);
// check for parse errors
if (json_last_error() == JSON_ERROR_NONE) {
echo $anObject->artist->nationality;
}
?>
```

To convert a PHP object into a JSON string, use the json_encode() function.

```
// convert PHP object into a JSON string
$text = json_encode($anObject);
```

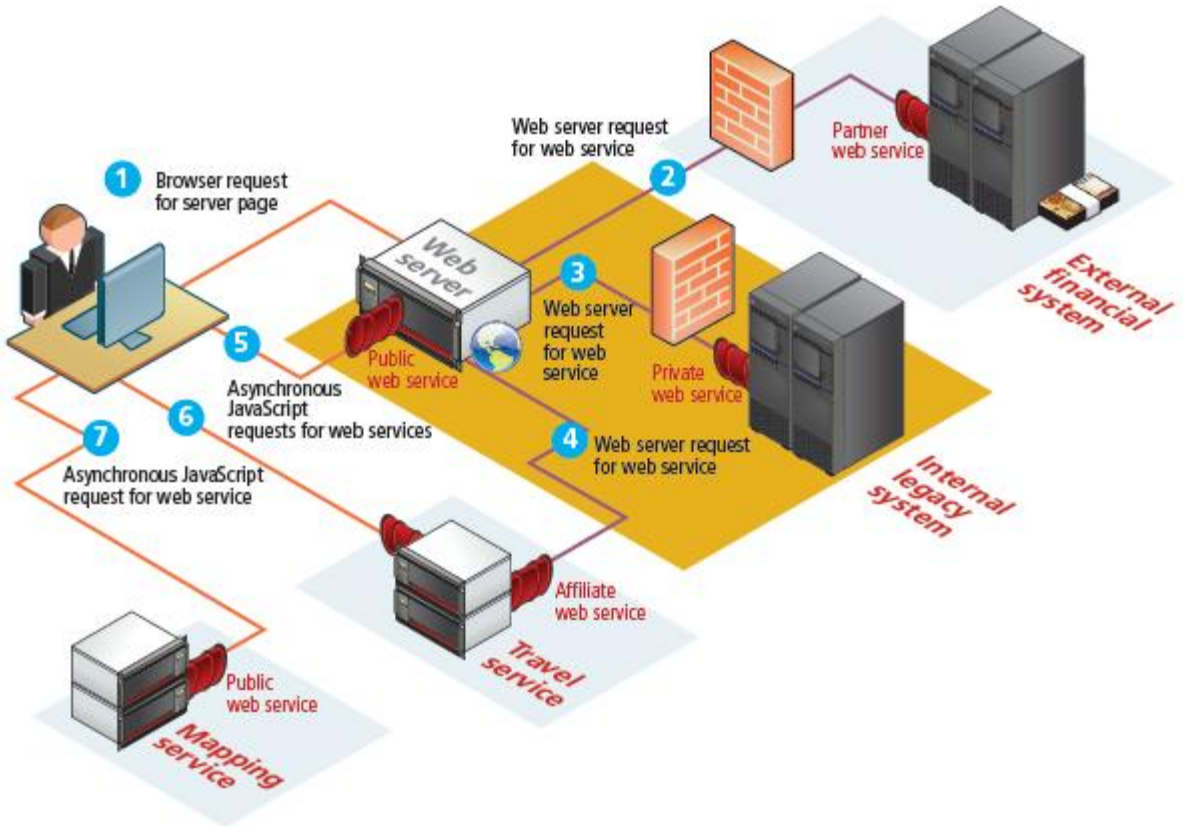
5.12 Overview of Web Services

Web services are the most common example of a computing paradigm commonly referred to as **service-oriented computing** (SOC), which utilizes something called "services" as a key element in the development and operation of software applications.

A **service** is a piece of software with a platform-independent interface that can be dynamically located and invoked. **Web services** are a relatively standardized mechanism by which one software application can connect to and communicate with another software application using web protocols. Web services make use of the **HTTP protocol** so that they can be used by any computer with Internet connectivity. Web services use XML or JSON to encode data within HTTP transmissions so that almost any platform should be able to encode or retrieve the data contained within a web service.

The benefit of web services is that they potentially provide interoperability between different software applications running on different platforms. Because web services use common and universally supported standards (HTTP and XML/ JSON), they are supported on a wide variety of platforms. Another key benefit of web services is that they can be used to implement **service-oriented architecture** (SOA). This type of software architecture aims to achieve very loose coupling among interacting software services.

Overview of web services

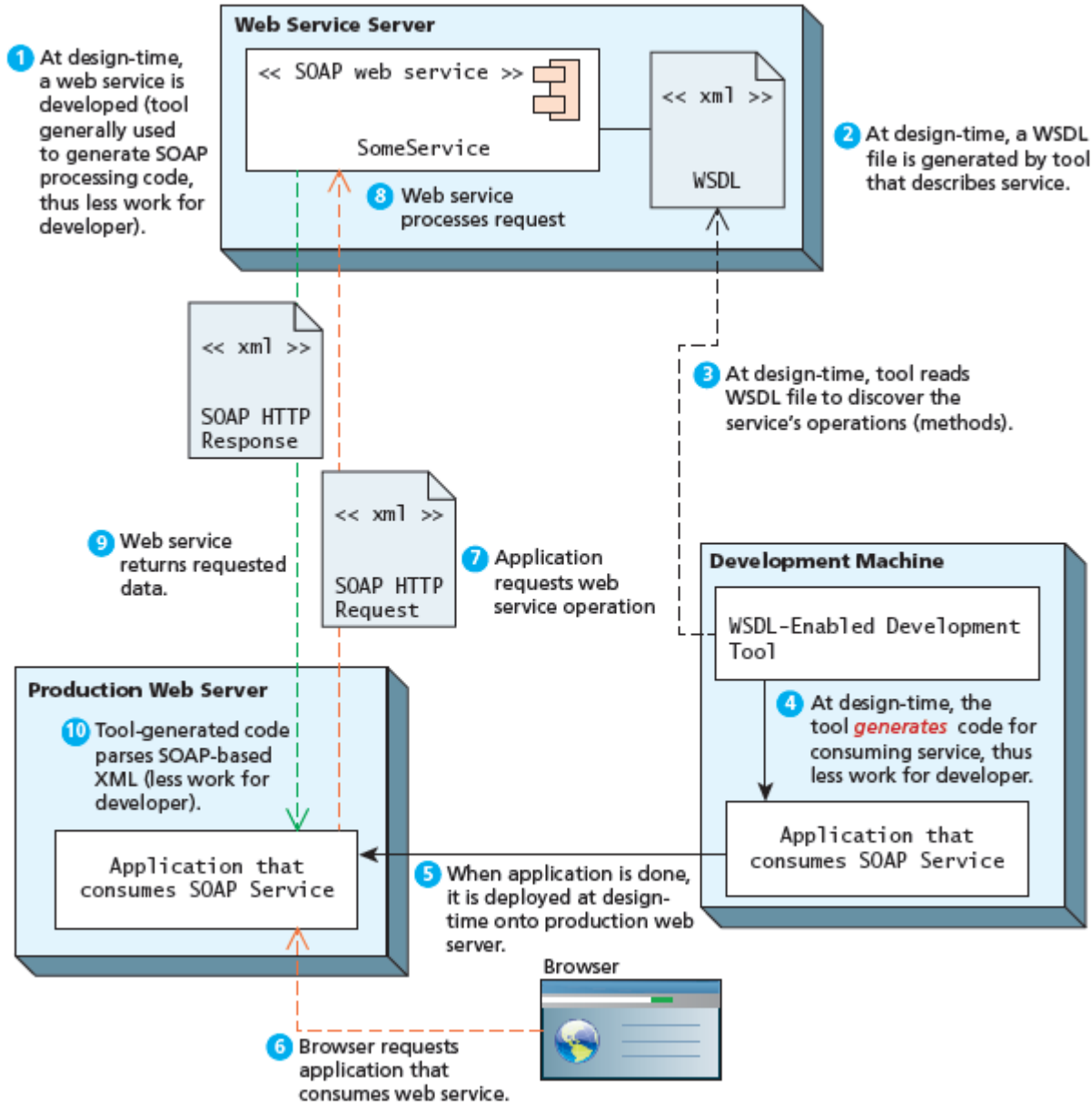


SOAP Services

A series of related technologies are used in web services, namely –
SOAP(Simple Object Access Protocol)
WSDL(Web Services Description Language)
REST (Representational State Transfer)

SOAP is the message protocol used to encode the service invocations and their return values via XML within the HTTP header, using POST method.
WSDL is used to describe the operations and data types provided by the service.
REST is the message protocol used to encode the service invocations and their return values via XML/JSON/text format within the HTTP header, GET method.

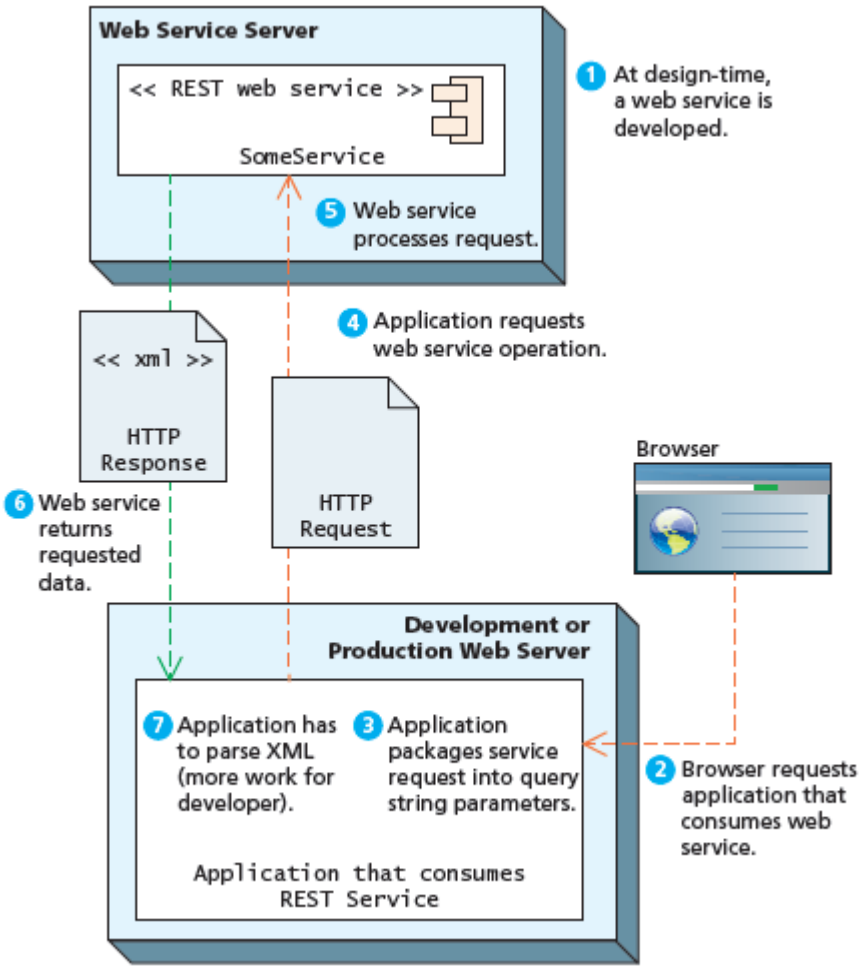
SOAP and WSDL are complex XML schemas, and is well supported in the .NET and Java environments. Detailed knowledge of the SOAP and WSDL specifications is not required to create and consume SOAP-based services. Using SOAP-based services is like compiler: its output may be complicated to understand, but it certainly makes life easier for most programmers. There are standard tools to generate SOAP and WSDL files like compiled files.



REST Services

REST stands for Representational State Transfer. It is implemented as an alternative to SOAP. A RESTful web service does not need a service description layer and also does not need a separate protocol for encoding message requests and responses. Instead it simply uses HTTP URLs for requesting a resource/object. The serialized representation of this object (XML or JSON stream), is then returned to the requestor as a normal HTTP response.

No special steps are needed to deploy a REST-based service, no special tools are generally needed to use a RESTful service, and it is easier to scale for a large number of clients using well-established practices and experience with caching, clustering, and loadbalancing traditional dynamic HTTP websites.



The lightweight nature of REST made it significantly easier to use than SOAP, hence REST appears to have almost completely displaced SOAP services. If an object is serialized via JSON, it can be turned into a complex JavaScript object in one simple line of JavaScript.

The easy availability of a wide range of RESTful services has given rise to a new style of web development, often referred to as a **mashup**, which generally refers to a website that combines and integrates data from a variety of different sources.

An Example Web Service

The example web service studied here is the Google Geocoding API. The term **geocoding** typically refers to the process of turning a real-world address (such as **British Museum**, **Great Russell Street**, **London**, **WC1B 3DG**) into geographic coordinates, which are usually latitude and longitude values (such as **51.5179231**, **-0.1271022**).

Reverse geocoding is the process of converting geographic coordinates into a human-readable address.

The Google Geocoding API provides a way to perform geocoding operations via an HTTP GET request, and thus is an especially useful example of a RESTful web service.

Like all of the REST web services we will examine a web service beginning with an HTTP request. In this case the request will take the following form:
http://maps.googleapis.com/maps/api/geocode/xml?*parameters*
The parameters in this case are address (for the real-world address to geocode) and sensor (for whether the request comes from a device with a location sensor).

So an example geocode request would look like the following:
http://maps.googleapis.com/maps/api/geocode/xml?address=British%20Museum,+Great+Russell+Street,+London,+WC1B+3DG&sensor=false
Notice that a REST request, like all HTTP requests, must URL encode special characters such as spaces. If the request is well formed and the service is working, it will return an HTTP response similar to that shown below.

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8
Date: Fri, 19 Jul 2013 19:15:54 GMT
Expires: Sat, 20 Jul 2013 19:15:54 GMT
Cache-Control: public, max-age=86400
Vary: Accept-Language
Content-Encoding: gzip
Server: mafe
Content-Length: 512
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
        <status>OK</status>
        <result>
                <type>route</type>
                <formatted_address>
                        Great Russell Street, London Borough of Camden, London, UK
                </formatted_address>
                <address_component>
                        <long_name>Great Russell Street</long_name>
                        <short_name>Great Russell St</short_name>
                        <type>route</type>
                </address_component>
                <address_component>
                        <long_name>London</long_name>
                        <short_name>London</short_name>
                        <type>locality</type>
                        <type>political</type>
                </address_component>
                ...
                <geometry>
                        <location>
```

```
                        <lat>51.5179231</lat>
                        <lng>-0.1271022</lng>
                    </location
                    <location_type>GEOMETRIC_CENTER</location_type>
                    ...
                </geometry>
        </result>
</GeocodeResponse>
```

After receiving this response, XML processing is done in order to extract the latitude and longitude values.

Identifying and Authenticating Service Requests

The Geocoding service illustrated a service that was openly available to any request. Most web services are not open in the same way. Instead, they typically employ one of the following techniques:

■ **Identity**. Each web service request must identify who is making the request.

■ **Authentication**. Each web service request must provide additional evidence that they are who they say they are.

Many web services are not providing information that is especially private or proprietary. For instance, the Flickr web service, which provides URLs to publicly available photos on their site in response to search criteria, is in some ways simply an XML version of the main site's already existing search facility. Since no private user data is being requested, it only expects each web service request to include one or more API keys to identify who is making the request.

This typically is done not only for internal record-keeping, but more importantly to keep service request volume at a manageable level. Most external web service APIs limit the number of web service requests that can be made, generally either per second, per hour, or per day. For instance, Panoramio limits requests to 100,000 per day while Google Maps and Microsoft Bing Maps allow 50,000 geocoding requests per day; Instagram allows 5000 requests per hour but Twitter allows just 100 to 400 requests per hour (it can vary); Amazon and last.fm limit requests to just one per second. Other services such as Flickr, NileGuide, and YouTube have no documented request limits.

While some web services are simply providing information already available on their website, other web services are providing private/proprietary information or are involving financial transactions. In this case, these services not only may require an API key, but they also require some type of user name and password in order to perform an authorization.

Advanced JavaScript & jQuery

5.13 JavaScript Pseudo-Classes

Although JavaScript has no formal class mechanism, it does support objects (such as the DOM). While most object-oriented languages that support objects also support classes formally, JavaScript does not. Instead, you define **pseudo-classes** through a variety of interesting and nonintuitive syntax constructs. Many common features of object-oriented programming, such as inheritance and even simple methods, must be arrived at through these nonintuitive means. Benefits of using object-oriented design in your JavaScript include increased code reuse, better memory management, and easier maintenance.

Almost all modern frameworks (such as jQuery and the Google Maps API) use prototypes to simulate classes, so understanding the mechanism is essential to apply those APIs in applications.
An object can be instantiated using an object represented by the list of key-value pairs with colons between the key and value with commas separating key-value pairs.
A dice object, with a string to hold the color and an array containing the values representing each side (face), could be defined all at once using object literals as follows:

```
var oneDie = { color : "FF0000", faces : [1,2,3,4,5,6] };
```

Once defined, these elements can be accessed using dot notation. For instance, one could change the color to blue by writing:
oneDie.color="0000FF";

Emulate Classes through Functions
Although a formal *class* mechanism is not available to us in JavaScript, it is possible to get close by using functions to encapsulate variables and methods together, as shown below.

```
function Die(col) {
this.color=col;
this.faces=[1,2,3,4,5,6];
}
```

The ‘this’ keyword inside of a function refers to the instance, so that every reference to internal properties or methods manages its own variables, as is the case with PHP. One can create an instance of the object as follows, very similar to PHP.
var oneDie = new Die("0000FF");

Adding Methods to the Object
One of the most common features one expects from a class is the ability to define behaviors with methods. In JavaScript this is relatively easy to do syntactically. To define a method in an object’s function one can either define it internally, or use a reference to a function defined outside the class. External definitions can quickly cause namespace conflict issues, since all method names must remain conflict free with all other methods for other classes. For this reason, one technique for adding a method inside of a class definition is by assigning an anonymous function to a variable, as shown below.

```
function Die(col) {
this.color=col;
this.faces=[1,2,3,4,5,6];
// define method randomRoll as an anonymous function
this.randomRoll = function() {
var randNum = Math.floor((Math.random() * this.faces.length)+ 1);
return faces[randNum-1];
};
}
```

With this method so defined, all dice objects can call the randomRoll function, which will return one of the six faces defined in the Die constructor.

```
var oneDie = new Die("0000FF");
console.log(oneDie.randomRoll() + " was rolled");
```

Although this mechanism for methods is effective, it is not a memory-efficient approach because each inline method is redefined for each new object.

Using Prototypes

Prototypes are an essential syntax mechanism in JavaScript, and are used to make JavaScript behave more like an object-oriented language. The prototype properties and methods are defined *once* for all instances of an *object.* So modification of the definition of the randomRoll() method, is done as shown below by moving the randomRoll() method into the prototype.

```
// Start Die Class
function Die(col) {
this.color=col;
this.faces=[1,2,3,4,5,6];
}
Die.prototype.randomRoll = function() {
var randNum = Math.floor((Math.random() * this.faces.length) + 1);
return faces[randNum-1];
};
// End Die Class
```

This definition is better because it defines the method only once, no matter how many instances of Die are created. The below figure shows how the prototype object (not class) is updated to contain the method so that subsequent instantiations (x and y) refers that one-method definition. Since all instances of a Die share the same prototype object, the function declaration only happens one time and is shared with all Die instances.

A prototype is an object from which other objects inherit.

The above definition sounds almost like a class in an object-oriented language, except that a prototype is itself an *object*, whereas in other oriented-oriented languages a class is an

abstraction, not an object. Despite this distinction, you can make use of a function's prototype object, and assign properties or methods to it that are then available to any new objects that are created.

In addition to the obvious application of prototypes to our own pseudo-classes, prototypes enable you to *extend* existing classes by adding to their prototypes.

Imagine a method added to the String object, which allows you to count instances of a character. The below snippet defines a method, named countChars, that takes a character as a parameter.

```
String.prototype.countChars = function (c) {
var count=0;
for (var i=0;i<this.length;i++) {
if (this.charAt(i) == c)
count++;
}
return count;
}
```

Now any new instances of String will have this method available to them. For instance the following example will output Hello World has 3 letter l's.

```
var hel = "Hello World";
console.log(hel + "has" + hel.countChars("l") + " letter l's");
```

This technique is also useful to assign properties to a pseudo-class that you want available to all instances. Imagine an array of all the *valid* characters attached to some custom string class.

5.14 jQuery Foundations

A **library** or **framework** is software that can be utilized in your own software, which provides some common implementations of standard ideas. A web framework can be expected to have features related to the web including HTTP headers, AJAX, authentication, DOM manipulation, cross-browser implementations, and more.

jQuery's beginnings date back to August 2005, when jQuery founder John Resig was looking into how to better combine CSS selectors with succinct JavaScript notation. Within a year, AJAX and animations were added, and the project has been improving ever since. Many developers find that once they start using a framework like jQuery, there's no going back to "pure" JavaScript because the framework offers so many useful shortcuts and succinct ways of doing things. **jQuery** is now the most popular JavaScript library currently in use.

Since the entire library exists as a source JavaScript file, importing jQuery for use in application is as easy as including a link to a file in the <head> section of HTML page. Either link to a locally hosted version of the library or use an approved third-party host, such as Google, Microsoft, or jQuery itself. Using a third-party **content delivery network (CDN)** is advantageous for several reasons,

1. the bandwidth of the file is offloaded to reduce the demand on your servers

2. the user may already have cached the third-party file and thus not have to download it again, thereby reducing the total loading time. This probability is increased when using a CDN like Google rather than a developer-focused CDN like jQuery.

A disadvantage to the third-party CDN is that your jQuery will fail if the thirdparty host fails, although that is unlikely given the mission-critical demands of large companies like Google and Microsoft.

jQuery Selectors

Selectors offer the developer a way of accessing and modifying a DOM object from an HTML page in a simple way. Although the advanced querySelector() methods allow selection of DOM elements based on CSS selectors, it is only implemented in newest browsers. To address this issue jQuery introduces its own way to select an element. jQuery builds on the CSS selectors and adds its own to let you access elements as you would in CSS or using new shortcut methods.

The relationship between DOM objects and selectors is so important in JavaScript programming that the pseudo-class bearing the name of the framework, jQuery(), lets programmers easily access DOM objects using selectors passed as parameters. Because it is used so frequently, it has a shortcut notation and can be written as $(). This $() syntax can be confusing to PHP developers at first, since in PHP the $ symbol indicates a variable. Nonetheless jQuery uses this shorthand frequently.

CSS selectors are combined with the $() notation to select DOM objects that match CSS attributes. Pass in the string of a CSS selector to $() and the result will be the set of DOM objects matching the selector. Basic selector syntax is used from CSS, as well as some additional ones defined within jQuery.

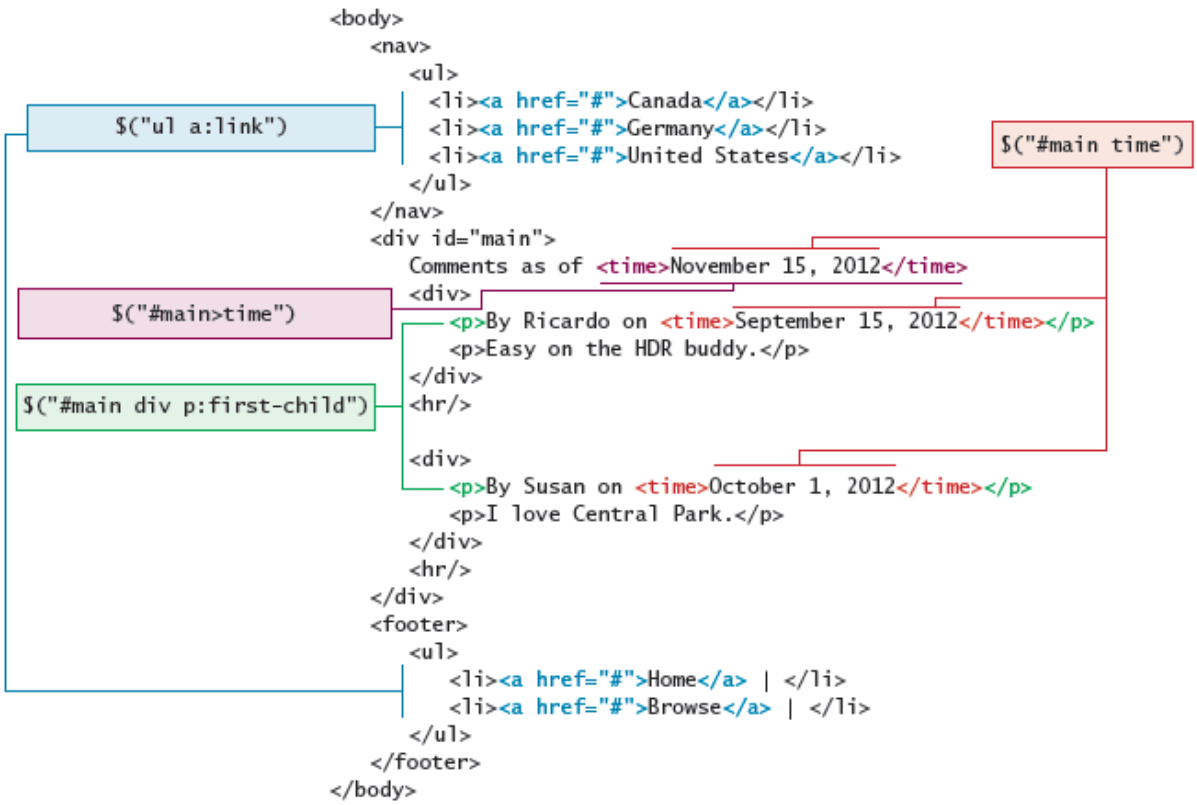
Basic Selectors

The four basic selectors include the universal selector, class selectors, id selectors, and elements selectors:

■ $("*") **Universal selector** matches all elements (and is slow).
■ $("tag") **Element selector** matches all elements with the given element name.
■ $(".class") **Class selector** matches all elements with the given CSS class.
■ $("#id") **Id selector** matches all elements with a given HTML id attribute.

To select the single element with id="grab", write as
var singleElement = $("#grab");
To get a set of all the <a> elements the selector would be:
var allAs = $("a");

These selectors are powerful enough that they can replace the use of getElementById() entirely. Some of the selections can be written as,



Attribute Selector

An **attribute selector** provides a way to select elements by either the presence of an element attribute or by the value of an attribute.

A selector to grab all <img> elements with an src attribute beginning with /artist/ as:
var artistImages = $("img[src^='/artist/']");

You can select by attribute with square brackets ([attribute]), specify a value with an equals sign ([attribute=value]) and search for a particular value in the beginning, end, or anywhere inside a string with ^, $, and * symbols respectively ([attribute^=value], [attribute$=value], [attribute*=value]).

Pseudo-Element Selector

Pseudo-elements are special elements, which are special cases of regular ones. The yallow you to append to any selector using the colon and one of :link, :visited, :focus, :hover, :active, :checked, :first-child, :first-line, and :first-letter.
These selectors can be used in combination with any selectors or alone. Selecting all links that have been visited, for example, would be specified with:
var visitedLinks = $("a:visited");

Contextual Selector

These selectors allowed you to specify elements with certain relationships to one another in your CSS. These relationships included descendant (space), child (>), adjacent sibling (+), and general sibling (~).

To select all <p> elements inside of elements you would write
var para = $("div p");

Content Filters
The **content filter** is the only jQuery selector that allows to append filters to all of the selectors and match a particular pattern. You can select elements that have a particular child using :has(), have no children using :empty, or match a particular piece of text with :contains().

Consider the following example:
var allWarningText = $("body *:contains('warning')");
It will return a list of all the DOM elements with the word *warning* inside of them.

Form Selectors
There are jQuery selectors written especially for form HTML elements. These selectors are well known and frequently used to collect and transmit data. Examples are listed in below table

Selector	CSS Equivalent	Description
$(:button)	$("button, input[type='button']")	Selects all *buttons*.
$(:checkbox)	$('[type=checkbox]')	Selects all *checkboxes*.
$(:checked)	No equivalent	Selects elements that are checked. This includes radio buttons and checkboxes.
$(:disabled)	No equivalent	Selects form elements that are disabled. These could include <button>, <input>, <optgroup>, <option>, <select>, and <textarea>
$(:enabled)	No equivalent	Opposite of :disabled. It returns all elements where the disabled attribute=false as well as form elements with no disabled attribute.
$(:file)	$('[type=file]')	Selects all elements of type file.
$(:focus)	$(document.activeElement)	The element with focus.
$(:image)	$('[type=image]')	Selects all elements of type image.
$(:input)	No equivalent	Selects all <input>, <textarea>, <select>, and <button> elements.
$(:password)	$('[type=password]')	Selects all password fields.
$(:radio)	$('[type=radio]')	Selects all radio elements.
$(:reset)	$('[type=reset]')	Selects all the reset buttons.
$(:selected)	No equivalent	Selects all the elements that are currently selected of type <option>. It does not include checkboxes or radio buttons.
$(:submit)	$('[type=submit]')	Selects all submit input elements.
$(:text)	No equivalent	Selects all input elements of type text. $('[type=text]') is almost the same, except that $(:text) includes <input>. fields with no type specified.

jQuery Attributes

Any set of elements from a web page can be selected. In order to fully manipulate the elements, one must understand an element's *attributes* and *properties*.

HTML Attributes

The core set of attributes related to DOM elements are the ones specified in the HTML tags.
In jQuery we can both set and get an attribute value by using the attr() method on any element from a selector. This function takes a parameter to specify which attribute, and the optional

second parameter is the value to set it to. If no second parameter is passed, then the return value of the call is the current value of the attribute. Some example usages are:

```
// var link is assigned the href attribute of the first <a> tag
var link = $("a").attr("href");
// change all links in the page to http://funwebdev.com
$("a").attr("href","http://funwebdev.com");
// change the class for all images on the page to fancy
$("img").attr("class","fancy");
```

HTML Properties

Many HTML tags include properties as well as attributes, the most common being the *checked* property of a radio button or checkbox. In early versions of jQuery, HTML properties could be set using the attr() method. However, since properties are not technically attributes, this resulted in odd behavior. The prop() method is now the preferred way to retrieve and set the value of a property although, attr() may return some (less useful) values.

To illustrate this subtle difference, consider a DOM element defined by

```
<input class ="meh" type="checkbox" checked="checked">
```

The value of the attr() and prop() functions on that element differ as shown below.

```
var theBox = $(".meh");
theBox.prop("checked") // evaluates to TRUE
theBox.attr("checked") // evaluates to "checked"
```

Changing CSS

Changing a CSS style is syntactically very similar to changing attributes. jQuery provides the extremely intuitive css() methods. There are two versions of this method (with two different method signatures), one to get the value and another to set it. The first version takes a single parameter containing the CSS attribute whose value you want and returns the current value.

```
$color = $("#colourBox").css("background-color"); // get the color
```

To modify a CSS attribute you use the second version of css(), which takes two parameters: the first being the CSS attribute, and the second the value.

```
// set color to red
$("#colourBox").css("background-color", "#FF0000");
```

If you want to use classes instead of overriding particular CSS attributes individually, have a look at the additional shortcut methods described in the jQuery documentation.

Shortcut Methods

jQuery allows the programmer to rely on foundational HTML attributes and properties exclusively as described above. However, as with selectors, there are additional functions that provide easier access to common operations such as changing an object's class or the text within an HTML tag.

The html() method is used to get the HTML contents of an element (the part between the <> and </> tags associated with the innerHTML property in JavaScript).

If passed with a parameter, it updates the HTML of that element. The html() method should be used with caution since the inner HTML of a DOM element can itself contain nested HTML elements! When replacing DOM with text, you may inadvertently introduce DOM errors since no validation is done on the new content (the browser wouldn't want to presume).
You can enforce the DOM by manipulating textNode objects and adding them as children to an element in the DOM tree rather than use html(). While this enforces the DOM structure, it does complicate code. To illustrate, consider that you could replace the content of every <p> element with "jQuery is fun," with the one line of code:
$("p").html("jQuery is fun");

The shortcut methods addClass(className) / removeClass(className) add or remove a CSS class to the element being worked on. The className used for these functions can contain a space-separated list of classnames to be added or removed.

The hasClass(classname) method returns true if the element has the className currently assigned. False, otherwise. The toggleClass(className) method will add or remove the class className, depending on whether it is currently present in the list of classes. The val() method returns the value of the element. This is typically used to retrieve values from input and select fields.

jQuery Listeners
jQuery supports creation and management of listeners/handlers for JavaScript events. The usage of these events is conceptually the same as with JavaScript with some minor syntactic differences.

Set Up after Page Load
In JavaScript, you learned why having your **listeners** set up inside of the window.onload() event was a good practice. Namely, it ensured the entire page and all DOM elements are loaded before trying to attach listeners to them. With jQuery we do the same thing but use the $(document).ready() event as shown below.

```
$(document).ready(function(){
//set up listeners on the change event for the file items.
$("input[type=file]").change(function(){
console.log("The file to upload is "+ this.value);
});
});
```

Listener Management
Setting up listeners for particular events is done in much the same way as JavaScript. While pure JavaScript uses the addEventListener() method, jQuery has on() and off() methods as well as shortcut methods to attach events. Modifying the code to use listeners rather than one handler yields the more modular code as shown below.

```
$(document).ready(function(){
$(":file").on("change",alertFileName); // add listener
```

```
});
// handler function using this
function alertFileName() {
console.log("The file selected is: "+this.value);
}
```

Modifying the DOM

jQuery comes with several useful methods to manipulate the DOM elements themselves.

Creating DOM and textNodes

If you decide to think about your page as a DOM object, then you will want to manipulate the tree structure rather than merely manipulate strings. jQuery is able to convert strings containing valid DOM syntax into DOM objects automatically.

The jQuery methods to manipulate the DOM take an HTML string, jQuery objects, or DOM objects as parameters, you might prefer to define your element as

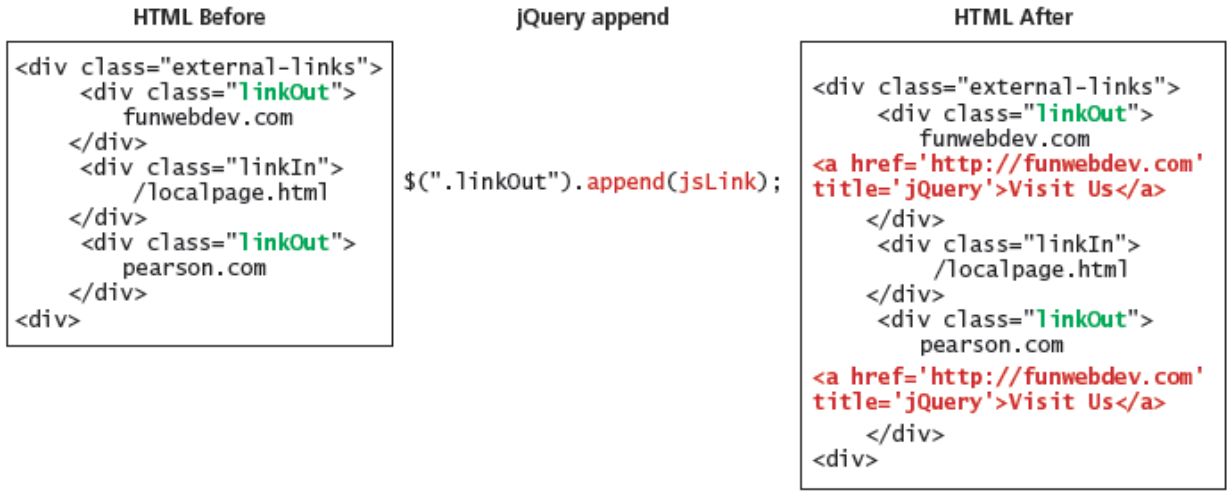
var element = $(" "); *//create new DOM node based on html*

This way you can apply all the jQuery functions to the object, rather than rely on pure JavaScript, which has fewer shortcuts. If we consider creation of a simple <a> element with multiple attributes, you can see the comparison of the JavaScript and jQuery techniques.

Prepending and Appending DOM Elements

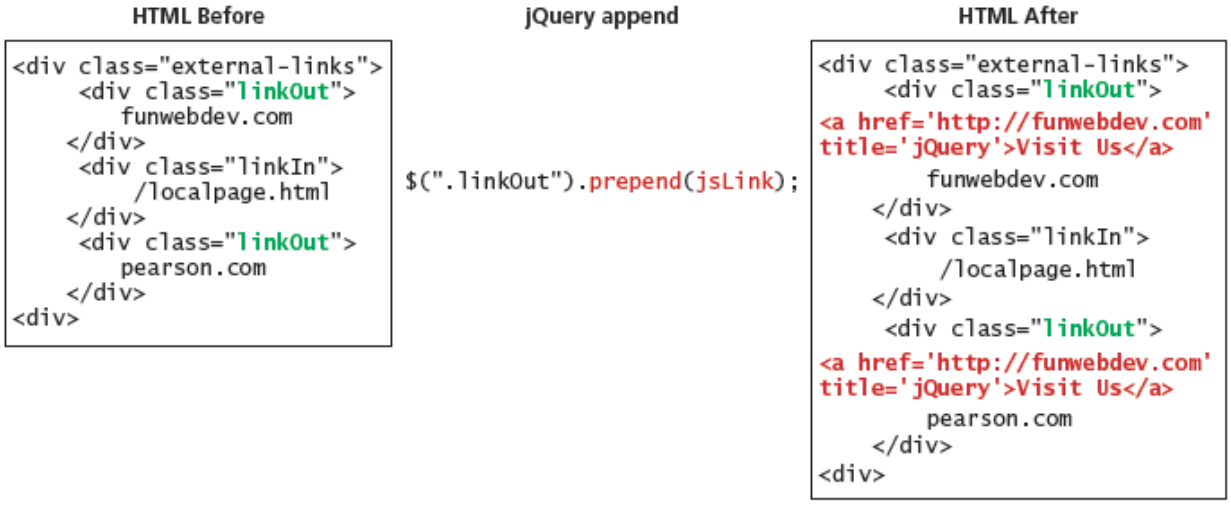
When an element is defined, it must be inserted into the existing DOM tree. You can also insert the element into several places at once if you desire, since selectors can return an array of DOM elements.

The append() method takes as a parameter an HTML string, a DOM object, or a jQuery object. That object is then added as the last child to the element(s) being selected. In below figure we can see the effect of an append() method call. Each element with a class of linkOut has the jsLink element appended to it.



The appendTo() method is similar to append() but is used in the syntactically converse way. If we were to use appendTo(), we would have to switch the object making the call and the parameter to have the same effect as the previous code:
jsLink.appendTo($(".linkOut"));

The prepend() and prependTo() methods operate in a similar manner except that they add the new element as the first child rather than the last.



Wrapping Existing DOM in New Tags

One of the most common ways to enhance a website that supports JavaScript is to add new HTML tags as needed to support some jQuery functions. Imagine for illustration purposes our art galleries being listed alongside some external links as described by the HTML below.

```
<div class="external-links">
<div class="galleryLink">
<div class="gallery">Uffuzi Museum</div>
</div>
<div class="galleryLink">
<div class="gallery">National Gallery</div>
</div>
<div class="link-out">funwebdev.com</div>
</div>
```

If we wanted to wrap all the gallery items in the whole page inside, another with class galleryLink we could write:
$(".gallery").wrap(' ');
which modifies the HTML to that shown below. Note how each and every link is wrapped in the correct opening and closing and uses the galleryLink class.

```
<div class="external-links">
<div class="galleryLink">
```

```
<div class="gallery">Uffuzi Museum</div>
</div>
<div class="galleryLink">
<div class="gallery">National Gallery</div>
</div>
<div class="link-out">funwebdev.com</div>
</div>
```

consider the situation where you wanted to add a title element to each element that reflected the unique contents inside. To achieve this more sophisticated manipulation, you must pass a function as a parameter rather than a tag to the wrap() method, and that function will return a dynamically created element as shown below.

```
$(".contact").wrap(function(){
return "<div class='galleryLink' title='Visit " + $(this).html() + "'></div>";
});
```

The wrap() method is a callback function, which is called for each element in a set. Each element then becomes this for the duration of one of the wrap() function's executions, allowing the unique title attributes as shown.

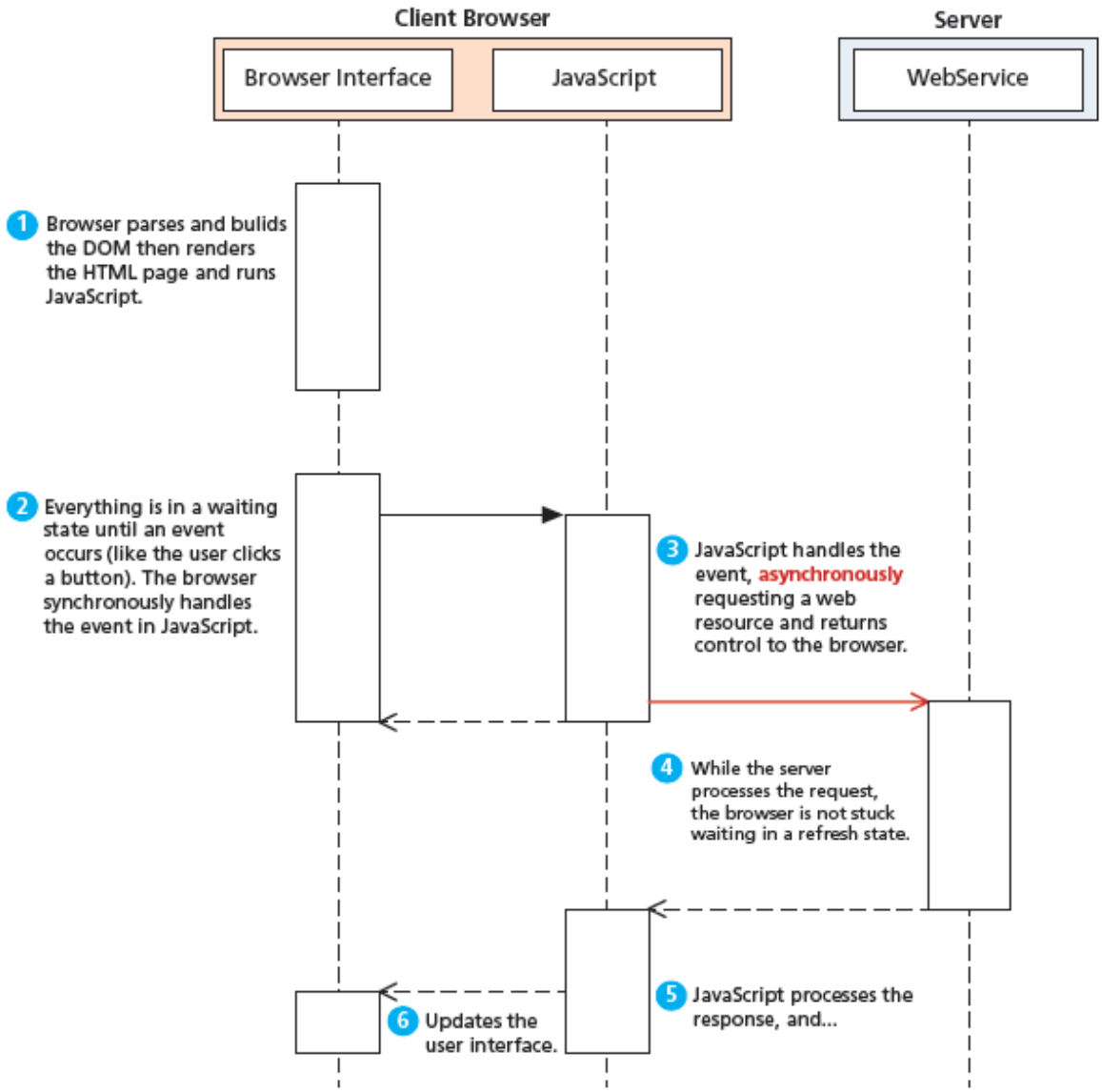
```
<div class="external-links">
<div class="galleryLink" title="Visit Uffuzi Museum">
<div class="gallery">Uffuzi Museum</div>
</div>
<div class="galleryLink" title="Visit National Gallery">
<div class="gallery">National Gallery</div>
</div>
<div class="link-out">funwebdev.com</div>
</div>
```

unwrap() is a method that does not take any parameters and whereas wrap() *added* a parent to the selected element(s), unwrap() *removes* the selected item's parent.

5.15 AJA X

Asynchronous JavaScript with XML (AJAX) is a term used to describe a paradigm that allows a web browser to send messages back to the server without interrupting the flow of what's being shown in the browser. This makes use of a browser's multi-threaded design and lets one thread handle the browser and interactions while other threads wait for responses to asynchronous requests.

The below figure annotates a UML sequence diagram where the white activity bars illustrate where computation is taking place. Between the request being sent and the response being received, the system can continue to process other requests from the client, so it does not appear to be waiting in a loading state.



Responses to asynchronous requests are caught in JavaScript as events. The events can subsequently trigger changes in the user interface or make additional requests. This differs from the typical synchronous requests we have seen thus far, which require the entire web page to refresh in response to a request.
Another way to contrast AJAX and synchronous JavaScript is to consider a web page that displays the current server time. If implemented synchronously, the entire page has to be refreshed from the server just to update the displayed time. During that refresh, the browser enters a waiting state, so the user experience is interrupted.

In contrast, consider the very simple asynchronous implementation of the server time, where an AJAX request updates the server time in the background as illustrated in figure b. In pure JavaScript it is possible to make asynchronous requests, but it's tricky and it differs greatly between browsers.

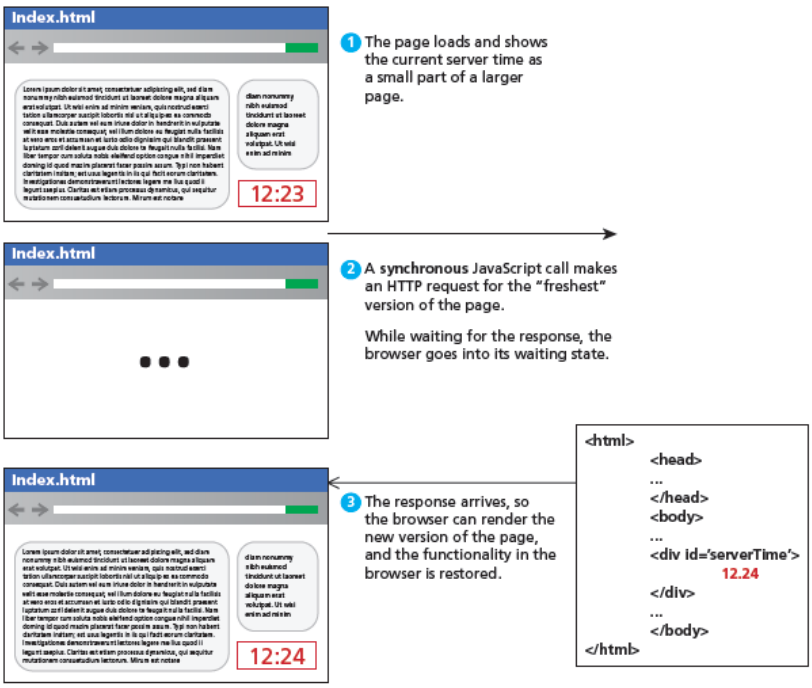


Figure (a)

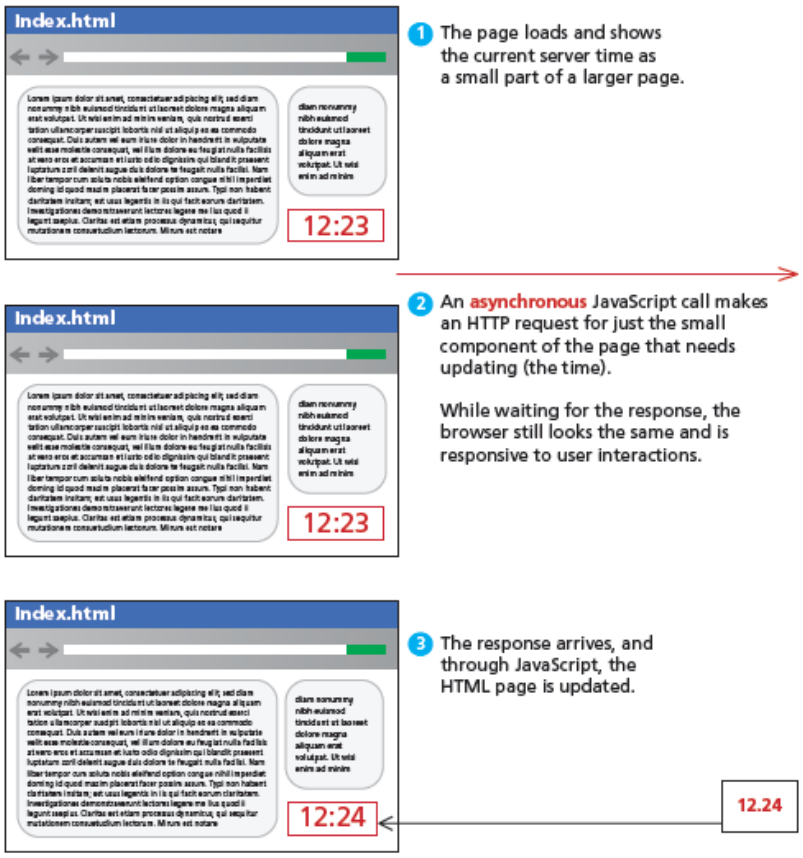


Figure (b)

Get Request via jQuery AJAX

AJAX get request can be done as followes:
jQuery.get (**url [, data] [, success(data, textStatus, jqXHR)] [, dataType]**)

- **url** is a string that holds the location to send the request.
- **data** is an optional parameter that is a query string or a *Plain Object.*
- **success(data,textStatus,jqXHR)** is an optional *callback* function that executes when the response is received.

data holding the body of the response as a string.
textStatus holding the status of the request (i.e., "success").
jqXHR holding a jqXHR object, described shortly.

- **dataType** is an optional parameter to hold the type of data expected from the server.

Example:

```
$.get("/vote.php?option=C", function(data,textStatus,jsXHR) {
  if (textStatus=="success") {
     console.log("success! response is:" + data);
  }
  else {
     console.log("There was an error code"+jsXHR.status);
  }
  console.log("all done");
});
```

LISTING 15.13 jQuery to asynchronously get a URL and outputs when the response arrives

All of the $.get() requests made by jQuery return a **jqXHR** object to encapsulate the response from the server.

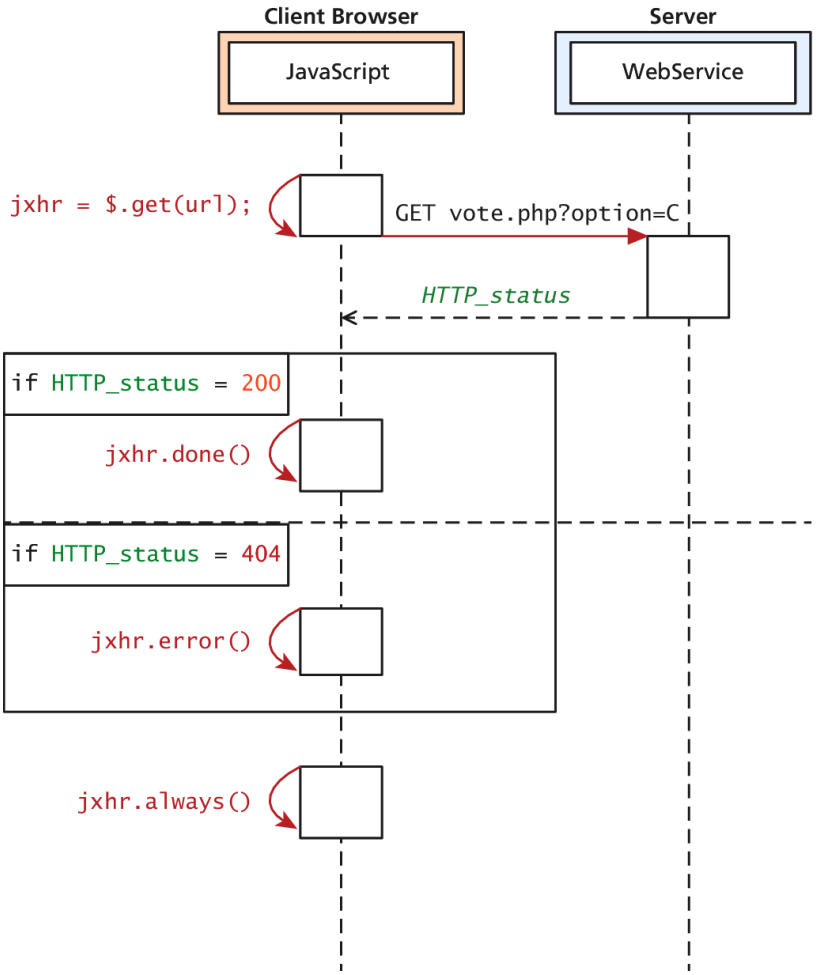
jqXHR - XMLHttpRequest compatibility:

- **abort()** stops execution and prevents any callback or handlers from receiving the trigger to execute.
- **getResponseHeader()** takes a parameter and gets the current value of that header.
- **readyState** is an integer from 1 to 4 representing the state of the request. The values include 1: sending, 3: response being processed, and 4: completed.
- **responseXML** and/or **responseText** the main response to the request.
- **setRequestHeader(name, value)** when used before actually instantiating the request allows headers to be changed for the request.
- **status** is the HTTP request status codes (200 = ok)
- **statusText** is the associated description of the status code.

jqXHR objects have methods

- done()
- fail()
- always()

which allow us to structure our code in a more modular way than the inline callback



Example:

```
var jqxhr = $.get("/vote.php?option=C");

jqxhr.done(function(data) { console.log(data);});
jqxhr.fail(function(jqXHR) { console.log("Error: "+jqXHR.status); })
jqxhr.always(function() { console.log("all done"); });
```

LISTING 15.14 Modular jQuery code using the jqXHR object

Post Request via jQuery AJAX

POST requests are often preferred to GET requests because one can post an unlimited amount of data, and because they do not generate viewable URLs for each action.
GET requests are typically not used when we have forms because of the messy URLs and that limitation on how much data we can transmit.
With POST it is possible to transmit files, something which is not possible with GET.

The HTTP 1.1 definition describes GET as a **safe method** meaning that they should not change anything, and should only read data.
POSTs on the other hand are not safe, and should be used whenever we are changing the state of our system (like casting a vote). get() method.
POST syntax is almost identical to GET.
jQuery.**post** (url [, data] [, success(data, textStatus, jqXHR)] [, dataType])

If we were to convert our vote casting code it would simply change the first line from
var jqxhr = $.get("/vote.php?option=C");
to
var jqxhr = $.post("/vote.php", "option=C");

serialize() can be called on any form object to return its current key-value pairing as an & separated string, suitable for use with post().

```
var postData = $("#voteForm").serialize();
$.post("vote.php", postData);
```

It turns out both the $.get() and $.post() methods are actually shorthand forms for the jQuery().ajax() method
The ajax() method has two versions. In the first it takes two parameters: a URL and a Plain Object, containing any of over 30 fields.
A second version with only one parameter is more commonly used, where the URL is but one of the key-value pairs in the Plain Object.

To pass HTTP headers to the ajax() method, you enclose as many as you would like in a Plain Object. To illustrate how you could override User-Agent and Referer headers in the POST

```
$.ajax({ url: "vote.php",
         data: $("#voteForm").serialize(),
         async: true,
         type: post
});
```

LISTING 15.15 A raw AJAX method code to make a post

To pass HTTP headers to the ajax() method, you enclose as many as you would like in a Plain Object. To illustrate how you could override User-Agent and Referer headers in the POST

```
$.ajax({ url: "vote.php",
  data: $("#voteForm").serialize(),
  async: true,
  type: post,
  headers: {"User-Agent" : "Homebrew JavaScript Vote Engine agent",
            "Referer": "http://funwebdev.com"
           }
});
```

LISTING 15.16 Adding headers to an AJAX post in jQuery

Cross-origin resource sharing (CORS)

- Since modern browsers prevent cross-origin requests by default (which is good for security), sharing content legitimately between two domains becomes harder.
- **Cross-origin resource sharing (CORS)** uses new headers in the HTML5 standard implemented in most new browsers.
- If a site wants to allow any domain to access its content through JavaScript, it would add the following header to all of its responses.
- **Access-Control-Allow-Origin: ***
- A better usage is to specify specific domains that are allowed, rather than cast the gates open to each and every domain. To allow our domain to make cross site requests we would add the header:
- Access-Control-Allow-Origin: www.funwebdev.com
- Rather than the wildcard *.

5.16 Asynchronous File Transmission

Asynchronous file transmission is one of the most powerful tools for modern web applications. In the days of old, transmitting a large file could require your user to wait idly by while the file uploaded, unable to do anything within the web interface. Since file upload speeds are almost always slower than download speeds, these transmissions can take minutes or even hours,

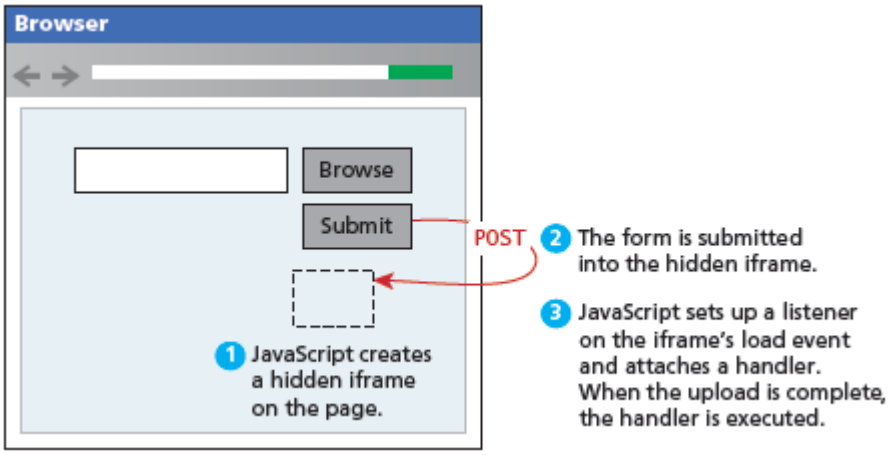
destroying the feeling of a "real" application. Unfortunately jQuery alone does not permit asynchronous file uploads! However, using clever tricks and HTML5 additions, you too can use asynchronous file uploads.

For the following examples consider a simple file-uploading HTML form defined below.

```
<form name="fileUpload" id="fileUpload" enctype="multipart/form-data" method="post" action="upload.php">
<input name="images" id="images" type="file" multiple />
<input type="submit" name="submit" value="Upload files!"/>
</form>
```

Old iframe Workarounds

The original workaround to allow the asynchronous posting of files was to use a hidden <iframe> element to receive the posted files. Given that jQuery still does not natively support the asynchronous uploading of files, this technique persists to this day and may be found in older code you have to maintain. As illustrated in the figure and the snippet below a hidden <iframe> allows one to post synchronously to another URL in another window. If JavaScript is enabled, you can also hide the upload button and use the change event instead to trigger a file upload. You then use the <iframe> element's onload event to trigger an action when it is done loading. When the window is done loading, the file has been received and we use the return message to update our interface much like we do with AJAX normally.




```
$(document).ready(function() {
// set up listener when the file changes
$(":file").on("change",uploadFile);
// hide the submit buttons
$("input[type=submit]").css("display","none");
});

// function called when the file being chosen changes
function uploadFile () {
// create a hidden iframe
var hidName = "hiddenIFrame";
```

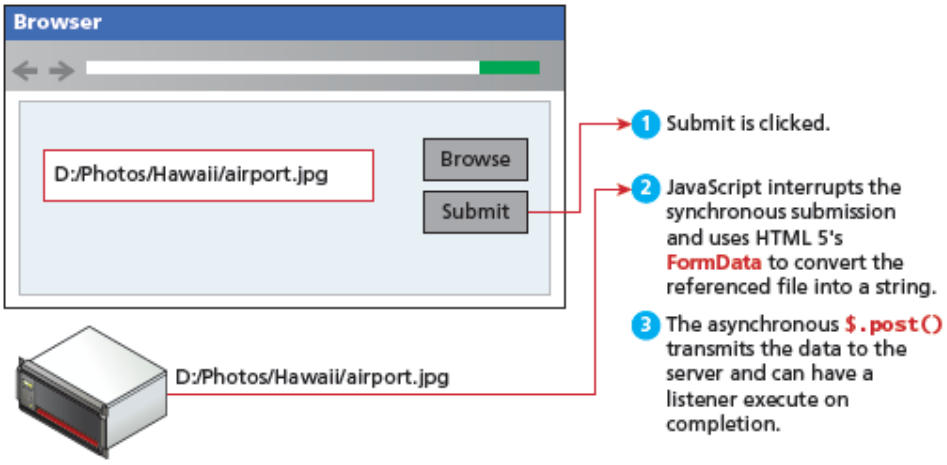
```
$("#fileUpload").append("<iframe id='"+hidName+"' name='"+hidName+"'
style='display:none' src='#' ></iframe>");
// set form's target to iframe
$("#fileUpload").prop("target",hidName);
// submit the form, now that an image is in it.
$("#fileUpload").submit();
// Now register the load event of the iframe to give feedback
$('#'+hidName).load(function() {
var link = $(this).contents().find('body')[0].innerHTML;
// add an image dynamically to the page from the file just uploaded
$("#fileUpload").append("<img src='"+link+"' />");
});
}
```

This technique exploits the fact that browsers treat each <iframe> element as a separate window with its own thread. By forcing the post to be handled in another window, we don't lose control of our user interface while the file is uploading.

Although it works, it's a workaround using the fact that every browser can post a file synchronously. A more modular and "pure" technique would be to somehow serialize the data in the file being uploaded with JavaScript and then post it in the body of a post request asynchronously.

The FormData Interface

The FormData interface provides a mechanism for JavaScript to read a file from the user's computer and encode it for upload. You can use this mechanism to upload a file asynchronously. Intuitively the browser is already able to do this, since it can access file data for transmission in synchronous posts. The FormData interface simply exposes this functionality to the developer, so you can turn a file into a string when you need to.



The <iframe> method uploadFile() can be replaced with the more elegant and straightforward code. In this pure AJAX technique the form object is passed to a FormData constructor, which is

then used in the call to send() the XHR2 object. This code attaches listeners for various events that may occur.

```
function uploadFile () {
// get the file as a string
var formData = new FormData($("#fileUpload")[0]);
var xhr = new XMLHttpRequest();
xhr.addEventListener("load", transferComplete, false);
xhr.addEventListener("error", transferFailed, false);
xhr.addEventListener("abort", transferCanceled, false);
xhr.open('POST', 'upload.php', true);
xhr.send(formData); // actually send the form data
function transferComplete(evt) { // stylized upload complete
$("#progress").css("width","100%");
$("#progress").html("100%");
}
function transferFailed(evt) {
alert("An error occurred while transferring the file.");
}
function transferCanceled(evt) {
alert("The transfer has been canceled by the user.");
}
}
```

Appending Files to a POST

When we consider uploading multiple files, you may want to upload a single file, rather than the entire form every time. To support that pattern, you can access a single file and post it by appending the raw file to a FormData object.. The advantage of this technique is that you submit each file to the server asynchronously as the user changes it; and it allows multiple files to be transmitted at once.

```
var xhr = new XMLHttpRequest();
// reference to the 1st file input field
var theFile = $(":file")[0].files[0];
var formData = new FormData();
formData.append('images', theFile);
```

To support uploading multiple files in our JavaScript code, we must loop through all the files rather than only hard-code the first one. The below snippet handles multiple files being selected and uploaded at once.

```
var allFiles = $(":file")[0].files;
for (var i=0;i<allFiles.length;i++) {
formData.append('images[]', allFiles[i]);
}
```

The main challenge of asynchronous file upload is that your implementation must consider the range of browsers being used by your users.

5.17 Animation

When animation features are used appropriately, they make web applications appear more professional and engaging.

Animation Shortcuts

Animation is done using a raw animate() method and many more easy-to-use shortcuts like fadeIn()/fadeOut(), slideUp()/slideDown().

One of the common things done in a dynamic web page is to show and hide an element. Modifying the visibility of an element can be done using css(), but that causes an element to change instantaneously, which can be visually jarring. To provide a more natural transition from hiding to showing, the hide() and show() methods allow developers to easily hide elements gradually, rather than through an immediate change.
The hide() and show() methods can be called with no arguments to perform a default animation. Another version allows two parameters: the duration of the animation (in milliseconds) and a callback method to execute on completion. Using the callback is a great way to chain animations together, or just ensure elements are fully visible before changing their contents.

A visualization of the show() method is illustrated in Figure below. Note that both the size and opacity are changing during the animation. Although using the very straightforward hide() and show() methods works, you should be aware of some more advanced shortcuts that give you more control.

fadeIn()/fadeOut()

The fadeIn() and fadeOut() shortcut methods control the opacity of an element. The parameters passed are the duration and the callback, just like hide() and show(). Unlike hide() and show(), there is no scaling of the element, just strictly control over the transparency. The below figure shows a span during its animation using fadeIn(). It should be noted that there is another method, fadeTo(), that takes two parameters: a duration in milliseconds and the opacity to fade to (between 0 and 1).

slideDown()/slideUp()

The final shortcut methods we will talk about are slideUp() and slideDown(). These methods do not touch the opacity of an element, but rather gradually change its height. The below figure shows a slideDown() animation using an email icon.

Raw Animation

Just like $.get() and $.post() methods are shortcuts for the complete $.ajax() method, the animations shown this far are all specific versions of the generic animate() method. When you want to do animation that differs from the prepackaged animations, you will need to make use of animate.

The animate() method has several versions, but the one we will look at has the following form:
.animate(properties, options);

The properties parameter contains a Plain Object with all the CSS styles of the final state of the animation. The options parameter contains another Plain Object with any of the options below set.

■ always is the function to be called when the animation completes or stops with a fail condition.
■ done is a function to be called when the animation completes.
■ duration is a number controlling the duration of the animation.
■ fail is the function called if the animation does not complete.
■ progress is a function to be called after each step of the animation.
■ queue is a Boolean value telling the animation whether to wait in the queue of animations or not. If false, the animation begins immediately.
■ step is a function you can define that will be called periodically while the animation is still going. It takes two parameters: a now element, with the current numerical value of a CSS property, and an fx object, which is a temporary object with useful properties like the CSS attribute it represents (called *tween* in jQuery).
■ Advanced options called easing and specialEasing allow for advanced control over the speed of animation.

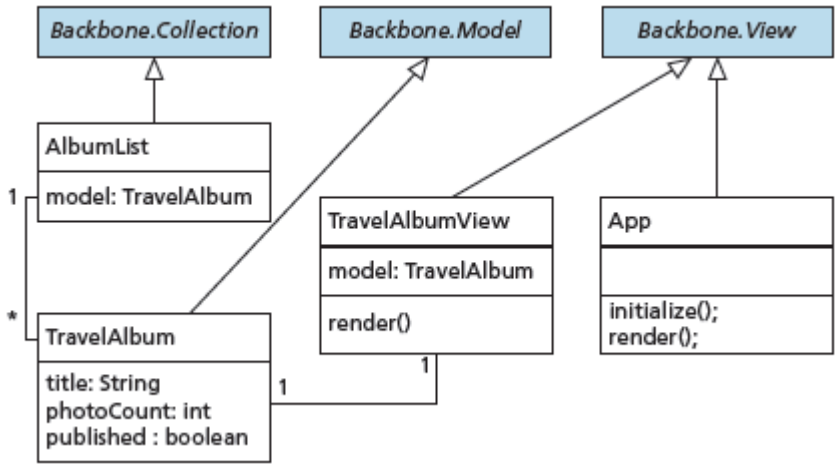
5.18 Backbone MVC Frameworks

MVC frameworks are overkill for small applications, where a small amount of jQuery is used. Backbone is an MVC framework that further abstracts JavaScript with libraries intended to adhere more closely to the MVC model. In Backbone, you build your client scripts around the concept of **models**. These models are often related to rows in the site's database and can be loaded, updated, and eventually saved back to the database using a REST interface.

Rather than writing the code to connect listeners and event handlers, Backbone allows user interface components to be notified of changes so they can update themselves just by setting

everything up correctly. You must download the source for these libraries to your server, and reference them.

<script src="underscore-min.js"></script>
<script src="backbone-min.js"></script>



Backbone Models

The term models can be a challenging one to apply, since authors of several frameworks and software engineering patterns already use the term.

Backbone.js defines **models** as

the heart of any JavaScript application, containing the interactive data as well as a large part of the logic surrounding it: conversions, validations, computed properties, and access control.

Collections

Backbone introduces the concept of **Collections**, which are normally used to contain lists of Model objects. These collections have advanced features and like a database can have indexes to improve search performance. In the snippet below is a collection of Albums, AlbumList, is defined by extending from Backbone's Collection object. In addition an initial list of TravelAlbums, named albums, is instantiated to illustrate the creation of some model objects inside a Collection.

```
// Create a collection of albums
var AlbumList = Backbone.Collection.extend({
// Set the model type for objects in this Collection
model: TravelAlbum,
// Return an array only with the published albums
GetChecked: function(){
return this.where({checked:true});
}
});

// Prefill the collection with some albums.
var albums = new AlbumList([
```

```
new TravelAlbum({ title: 'Banff, Canada', photoCount: 42}),
new TravelAlbum({ title: 'Santorini, Greece', photoCount: 102}),
]);
```

Views

Views allow you to translate your models into the HTML that is seen by the users. They attach themselves to methods and properties of the Collection and define methods that will be called whenever Backbone determines the view needs refreshing.

For our example we extend a View as shown below. In that code we attach our view to a particular tagName (in our case the <li> element) and then associate the click event with a new method named toggleAlbum().

```
var TravelAlbumView = Backbone.View.extend({
TagName: 'li',
events:{
'click': 'toggleAlbum'
},
initialize: function(){
// Set up event listeners attached to change
this.listenTo(this.model, 'change', this.render);
},
render: function(){
// Create the HTML
this.$el.html('<input type="checkbox" value="1" name="' +
this.model.get('title') + '" /> ' +
this.model.get('title') + '<span> ' +
this.model.get('photoCount') + ' images</span>');
this.$('input').prop('checked', this.model.get('checked'));
// Returning the object is a good practice
return this;
},
toggleAlbum: function() {
this.model.toggle();
}
});
```

Always override the render() method since it defines the HTML that is output.