UNIT – I

Introduction to PHP: Declaring variables, data types, arrays, strings, operators, expressions, control structures, functions, Reading data from web form controls like text boxes, radio buttons, lists etc., Handling File Uploads, Connecting to database (MySQL as reference), executing simple queries, handling results, Handling sessions and cookies.
File Handling in PHP: File operations like opening, closing, reading, writing, appending, deleting etc. on text and binary files, listing directories.

What is PHP?

- PHP stands for Hypertext Preprocessor.
- PHP is an interpreted language, i.e., there is no need for compilation.
- PHP is a server-side scripting language.
- PHP is faster than other scripting languages, for example, ASP and JSP.

Web Development

PHP is widely used in web development nowadays. PHP can develop dynamic websites easily.
But you must have the basic the knowledge of following technologies for web development as well.

- HTML
- CSS
- JavaScript
- Ajax
- XML and JSON
- jQuery

PHP - INTRODUCTION

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

Common uses of PHP

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
- You add, delete, modify elements within your database through PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

Characteristics of PHP

Five important characteristics make PHP's practical nature possible −

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

"Hello World" Script in PHP

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this −

```
<html>

  <head>
    <title>Hello World</title>
  </head>

  <body>
    <?php echo "Hello, World!";?>
  </body>

</html>
```

It will produce following result −

```
Hello, World!
```

If you examine the HTML output of the above example, you'll notice that the PHP code is not present in the file sent from the server to your Web browser. All of the PHP present in the Web page is processed and stripped from the page; the only thing returned to the client from the Web server is pure HTML output.

All PHP code must be included inside one of the three special markup tags ATE are recognised by the PHP Parser.

```
<?    PHP code goes here ?>

<script language = "php"> PHP code goes here </script>
```

PHP VARIABLES

A variable in PHP is a name of memory location that holds data. A variable is a temporary storage that
is used to store data temporarily.

In PHP, a variable is declared using $ sign followed by variable name.

Syntax of declaring a variable in PHP is given below:

```
$variablename=value;
```

PHP Variable: Declaring string, integer and float

Let's see the example to store string, integer and float values in PHP variables.

File: variable1.php

```
<?php
$str="hello string";
$x=200;
$y=44.6;
echo "string is: $str <br/>";
echo "integer is: $x <br/>";
echo "float is: $y <br/>";
?>
```

Output:

```
string is: hello string
integer is: 200
float is: 44.6
```

PHP Variable: Sum of two variables

File: variable2.php

```
<?php
$x=5;
$y=6;
$z=$x+$y;
echo $z;
?>
```

Output:

```
11
```

PHP Variable: case sensitive

In PHP, variable names are case sensitive. So variable name "color" is different from Color, COLOR, COLor etc.

File: variable3.php

```php
<?php
$color="red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>
```

Output:

```
My car is  red
Notice: Undefined variable: COLOR in C:\wamp\www\variable.php on line 4
My house is
Notice: Undefined variable: coLOR in C:\wamp\www\variable.php on line 5
My boat is
```

PHP Variable: Rules

PHP variables must start with letter or underscore only.

PHP variable can't be start with numbers and special symbols.

File: variablevalid.php

```php
<?php
$a="hello";//letter (valid)
$_b="hello";//underscore (valid)

echo "$a <br/> $_b";
?>
```

Output:

```
hello
hello
```

File: variableinvalid.php

```php
<?php
$4c="hello";//number (invalid)
$*d="hello";//special symbol (invalid)

echo "$4c <br/> $*d";
?>
```

Output:

```
Parse error: syntax error, unexpected '4' (T_LNUMBER), expecting variable
(T_VARIABLE)
  or '$' in  C:\wamp\www\variableinvalid.php  on  line  2
```

The main way to store information in the middle of a PHP program is by using a variable.

Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign ($).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

PHP DATA TYPES

PHP has a total of eight data types which we use to construct our variables −

- **Integers** − are whole numbers, without a decimal point, like 4195.
- **Doubles** − are floating-point numbers, like 3.14159 or 49.1.
- **Booleans** − have only two possible values either true or false.
- **NULL** − is a special type that only has one value: NULL.
- **Strings** − are sequences of characters, like 'PHP supports string operations.'
- **Arrays** − are named and indexed collections of other values.
- **Objects** − are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources** − are special variables that hold references to resources external to PHP (such as database connections).

The first five are *simple types*, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

We will explain only simple data type in this chapters. Array and Objects will be explained separately.

Integers

They are whole numbers, without a decimal point, like 4195. They are the simplest type .they correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions, like so −

```
$int_var = 12345;
```

$another_int = -12345 + 12345;

Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimals have a leading 0x.

For most common platforms, the largest integer is (2**31 . 1) (or 2,147,483,647), and the smallest (most negative) integer is . (2**31 . 1) (or .2,147,483,647).

Doubles

They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code −

```
<?php
   $many = 2.2888800;
   $many_2 = 2.2111200;
   $few = $many + $many_2;

   print("$many + $many_2 = $few <br>");
?>
```

It produces the following browser output −

```
2.28888 + 2.21112 = 4.5
```

Boolean

They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so −

```
if (TRUE)
   print("This will always print<br>");

else
   print("This will never print<br>");
```

Interpreting other types as Booleans

Here are the rules for determine the "truth" of any value not already of the Boolean type −

- If the value is a number, it is false if exactly equal to zero and true otherwise.
- If the value is a string, it is false if the string is empty (has zero characters) or is the string "0", and is true otherwise.
- Values of type NULL are always false.
- If the value is an array, it is false if it contains no other values, and it is true otherwise. For an object, containing a value means having a member variable that has been assigned a value.
- Valid resources are true (although some functions that return resources when they are successful will return FALSE when unsuccessful).
- Don't use double as Booleans.

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

```
$true_num = 3 + 0.14159;
$true_str = "Tried and true"
$true_array[49] = "An array element";
$false_array = array();
$false_null = NULL;
$false_num = 999 - 999;
$false_str = "";
```

NULL

NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this −

```
$my_var = NULL;
```

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed −

```
$my_var = null;
```

A variable that has been assigned NULL has the following properties −

- It evaluates to FALSE in a Boolean context.
- It returns FALSE when tested with IsSet() function.

Strings

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string

```
$string_1 = "This is a string in double quotes";
$string_2 = 'This is a somewhat longer, singly quoted string';
$string_39 = "This string has thirty-nine characters";
$string_0 = ""; // a string with zero characters
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<?php
  $variable = "name";
  $literally = 'My $variable will not print!';

  print($literally);
  print "<br>";

  $literally = "My $variable will print!";
  print($literally);
?>
```

This will produce following result −

```
My $variable will not print!
My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP −

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with $) are replaced with string representations of their values.

The escape-sequence replacements are −

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \$ is replaced by the dollar sign itself ($)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

Here Document

You can assign multiple lines to a single string variable using here document −

```
<?php
   $channel =<<<_XML_

   <channel>
      <title>What's For Dinner</title>
      <link>http://menu.example.com/ </link>
      <description>Choose what to eat tonight.</description>
   </channel>
   _XML_;

   echo <<<END
   This uses the "here document" syntax to output multiple lines with variable
   interpolation. Note that the here document terminator must appear on a line with
   just a semicolon. no extra whitespace!


   END;

   print $channel;
?>
```

This will produce following result −

```
This uses the "here document" syntax to output
multiple lines with variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!
```

```
<channel>
<title>What's For Dinner<title>
<link>http://menu.example.com/<link>
<description>Choose what to eat tonight.</description>
```

VARIABLE SCOPE

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types –

- Local variables
- Function parameters
- Global variables
- Static variables.

Variable Naming

Rules for naming a variable is −

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like + , - , % , (,) . & , etc

There is no size limit for variables.

PHP - LOCAL VARIABLES

A variable declared in a function is considered local; that is, it can be referenced solely in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function −

```
<?php
  $x = 4;

  function assignx () {
    $x = 0;
    print "\$x inside function is $x. <br />";
  }

  assignx();
  print "\$x outside of function is $x. <br />";
?>
```

This will produce the following result −

```
$x inside function is 0.
$x outside of function is 4.
```

PHP - FUNCTION PARAMETERS

Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be −

```
<?php
   // multiply a value by 10 and return it to the caller
   function multiply ($value) {
      $value = $value * 10;
      return $value;
   }

   $retval = multiply (10);
   Print "Return value is $retval\n";
?>
```

This will produce the following result −

```
Return value is 100
```

PHP - GLOBAL VARIABLES

In contrast to local variables, a global variable can be accessed in any part of the program. However, in order to be modified, a global variable must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword **GLOBAL** in front of the variable that should be recognized as global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name. Consider an example −

```
<?php
   $somevar = 15;

   function addit() {
      GLOBAL $somevar;
      $somevar++;

      print "Somevar is $somevar";
   }

   addit();
?>
```

This will produce the following result −

```
Somevar is 16
```

PHP - STATIC VARIABLES

The final type of variable scoping that I discuss is known as static. In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.

You can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

```
<?php
   function keep_track() {
      STATIC $count = 0;
      $count++;
      print $count;
      print "<br />";
   }

   keep_track();
   keep_track();
   keep_track();
?>
```

This will produce the following result −

```
1
2
3
```

PHP ARRAY TYPES

There are 3 types of array in PHP.

1. Indexed Array
2. Associative Array
3. Multidimensional Array

PHP Indexed Array

PHP index is represented by number which starts from 0. We can store number, string and object in the PHP array. All PHP array elements are assigned to an index number by default.

There are two ways to define indexed array:

1st way:

```
$season=array("summer","winter","spring","autumn");
```

2nd way:

```
$season[0]="summer";
$season[1]="winter";
$season[2]="spring";
$season[3]="autumn";
```

Example

File: array1.php

```
<?php
$season=array("summer","winter","spring","autumn");
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

Output:

```
Season are: summer, winter, spring and autumn
```

File: array2.php

```php
<?php
$season[0]="summer";
$season[1]="winter";
$season[2]="spring";
$season[3]="autumn";
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

Output:

```
Season are: summer, winter, spring and autumn
```

PHP Associative Array

We can associate name with each array elements in PHP using => symbol.

There are two ways to define associative array:

1st way:

```php
$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");
```

2nd way:

```php
$salary["Sonoo"]="350000";
$salary["John"]="450000";
$salary["Kartik"]="200000";
```

Example

File: arrayassociative1.php

```php
<?php
$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
?>
```

Output:

```
Sonoo salary: 350000
John salary: 450000
Kartik salary: 200000
```

File: arrayassociative2.php

```
<?php
$salary["Sonoo"]="350000";
$salary["John"]="450000";
$salary["Kartik"]="200000";
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
?>
```

Output:

```
Sonoo salary: 350000
John salary: 450000
Kartik salary: 200000
```

PHP Indexed Array

PHP indexed array is an array which is represented by an index number by default. All elements of array are represented by an index number which starts from 0.

PHP indexed array can store numbers, strings or any object. PHP indexed array is also known as numeric array.

Definition

There are two ways to define indexed array:

1st way:

```
$size=array("Big","Medium","Short");
```

2nd way:

```
$size[0]="Big";
$size[1]="Medium";
$size[2]="Short";
```

PHP Indexed Array Example

File: array1.php

```
<?php
$size=array("Big","Medium","Short");
echo "Size: $size[0], $size[1] and $size[2]";
?>
```

Output:

```
Size: Big, Medium and Short
```

File: array2.php

```
<?php
```

```
$size[0]="Big";
$size[1]="Medium";
$size[2]="Short";
echo "Size: $size[0], $size[1] and $size[2]";
?>
```

Output:

```
Size: Big, Medium and Short
```

Traversing PHP Indexed Array

We can easily traverse array in PHP using foreach loop. Let's see a simple example to traverse all the elements of PHP array.

File: array3.php

```
<?php
$size=array("Big","Medium","Short");
foreach( $size as $s )
{
  echo "Size is: $s<br />";
}
?>
```

Output:

```
Size is: Big
Size is: Medium
Size is: Short
```

Count Length of PHP Indexed Array

PHP provides count() function which returns length of an array.

```
<?php
$size=array("Big","Medium","Short");
echo count($size);
?>
```

Output:

```
3
```

PHP Associative Array

PHP allows you to associate name/label with each array elements in PHP using => symbol. Such way, you can easily remember the element because each element is represented by label than an incremented number.

Definition

There are two ways to define associative array:

1st way:

```
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
```

2nd way:

```
$salary["Sonoo"]="550000";
$salary["Vimal"]="250000";
$salary["Ratan"]="200000";
```

Example

File: arrayassociative1.php

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "Vimal salary: ".$salary["Vimal"]."<br/>";
echo "Ratan salary: ".$salary["Ratan"]."<br/>";
?>
```

Output:

```
Sonoo salary: 550000
Vimal salary: 250000
Ratan salary: 200000
```

File: arrayassociative2.php

```
<?php
$salary["Sonoo"]="550000";
$salary["Vimal"]="250000";
$salary["Ratan"]="200000";
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "Vimal salary: ".$salary["Vimal"]."<br/>";
echo "Ratan salary: ".$salary["Ratan"]."<br/>";
?>
```

Output:

```
Sonoo salary: 550000
Vimal salary: 250000
Ratan salary: 200000
```

PHP Multidimensional Array

PHP multidimensional array is also known as array of arrays. It allows you to store tabular data in an array. PHP multidimensional array can be represented in the form of matrix which is represented by row * column.

Definition

```
$emp = array
 (
  array(1,"sonoo",400000),
  array(2,"john",500000),
array(3,"rahul",300000)
 );
```

File: multiarray.php

PHP Multidimensional Array Example

Let's see a simple example of PHP multidimensional array to display following tabular data. In this example, we are displaying 3 rows and 3 columns.

```
<?php
$emp = array
  (
  array(1,"sonoo",400000),
  array(2,"john",500000),
  array(3,"rahul",300000)
  );

for ($row = 0; $row < 3; $row++) {
  for ($col = 0; $col < 3; $col++) {
    echo $emp[$row][$col]." ";
  }
  echo "<br/>";
}
?>
```

Output:

```
1 sonoo 400000
2 john 500000
3 rahul 300000
```

PHP Array Functions

PHP provides various array functions to access and manipulate the elements of array. The important PHP array functions are given below.

1) PHP array() function

PHP array() function creates and returns an array. It allows you to create indexed, associative and multidimensional arrays.

Syntax

```
array array ([ mixed $... ] )
```

Example

```
<?php
$season=array("summer","winter","spring","autumn");
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

Output:

```
Season are: summer, winter, spring and autumn
```

2) PHP array_change_key_case() function

PHP array_change_key_case() function changes the case of all key of an array.

Note: It changes case of key only.

Syntax

```
array array_change_key_case ( array $array [, int $case = CASE_LOWER ] )
```

Example

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");

print_r(array_change_key_case($salary,CASE_UPPER));
?>
```

Output:

```
Array ( [SONOO] => 550000 [VIMAL] => 250000 [RATAN] => 200000 )
```

Example

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");

print_r(array_change_key_case($salary,CASE_LOWER));
?>
```

Output:

```
Array ( [sonoo] => 550000 [vimal] => 250000 [ratan] => 200000 )
```

3) PHP array_chunk() function

PHP array_chunk() function splits array into chunks. By using array_chunk() method, you can divide array into many parts.

Syntax

1. **array** array_chunk (**array** $array , int $size [, bool $preserve_keys = false])

Example

1. <?php
2. $salary=**array**("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
3. print_r(array_chunk($salary,2));
4. ?>

Output:

```
Array (
[0] => Array ( [0] => 550000 [1] => 250000 )
[1] => Array ( [0] => 200000 )
)
```

4) PHP count() function

PHP count() function counts all elements in an array.

Syntax

1. int count (mixed $array_or_countable [, int $mode = COUNT_NORMAL])

Example

1. <?php
2. $season=**array**("summer","winter","spring","autumn");
3. echo count($season);
4. ?>

Output:

```
4
```

5) PHP sort() function

PHP sort() function sorts all the elements in an array.

Syntax

1. bool sort (**array** &$array [, int $sort_flags = SORT_REGULAR])

Example

```
<?php
$season=array("summer","winter","spring","autumn");
sort($season);
foreach( $season as $s )
{
  echo "$s<br />";
}
?>
```

Output:

```
autumn
spring
summer
winter
```

6) PHP array_reverse() function

PHP array_reverse() function returns an array containing elements in reversed order.

Syntax

```
array array_reverse ( array $array [, bool $preserve_keys = false ] )
```

Example

```
<?php
$season=array("summer","winter","spring","autumn");
$reverseseason=array_reverse($season);
foreach( $reverseseason as $s )
{
  echo "$s<br />";
}
?>
```

Output:

```
autumn
spring
winter
summer
```

7) PHP array_search() function

PHP array_search() function searches the specified value in an array. It returns key if search is successful.

Syntax

```
mixed array_search ( mixed $needle , array $haystack [, bool $strict = false ] )
```

Example

```
<?php
$season=array("summer","winter","spring","autumn");
$key=array_search("spring",$season);
echo $key;
?>
```

Output:

```
2
```

8) PHP array_intersect() function

PHP array_intersect() function returns the intersection of two array. In other words, it returns the matching elements of two array.

Syntax

```
array array_intersect ( array $array1 , array $array2 [, array $... ] )
```

Example

```
<?php
$name1=array("sonoo","john","vivek","smith");
$name2=array("umesh","sonoo","kartik","smith");
$name3=array_intersect($name1,$name2);
foreach( $name3 as $n )
{
echo "$n<br />";
}
?>
```

Output:

```
sonoo
smith
```

PHP STRING

PHP string is a sequence of characters i.e., used to store and manipulate text. PHP supports only 256-character set and so that it does not offer native Unicode support. There are 4 ways to specify a string literal in PHP.

1. single quoted
2. double quoted
3. heredoc syntax
4. newdoc syntax (since PHP 5.3)

Single Quoted

We can create a string in PHP by enclosing the text in a single-quote. It is the easiest way to specify string in PHP.

For specifying a literal single quote, escape it with a backslash (\) and to specify a literal backslash (\) use double backslash (\\). All the other instances with backslash such as \r or \n, will be output same as they specified instead of having any special meaning.

For Example

Following some examples are given to understand the single quoted PHP String in a better way:

Example 1

```
<?php
      $str='Hello text within single quote';
echo $str;4.
?>
```

Output:

```
Hello text within single quote
```

We can store multiple line text, special characters, and escape sequences in a single-quoted PHP string.

Example 2

```
<?php
$str1='Hello text
multiple line
text within single quoted string';
$str2='Using double "quote" directly inside single quoted string';
$str3='Using escape sequences \n in single quoted string';
echo "$str1 <br/> $str2 <br/> $str3";
?>
```

Output:

```
Hello text multiple line text within single quoted string
Using double "quote" directly inside single quoted string
Using escape sequences \n in single quoted string
```

Example 3

```
<?php
$num1=10;
$str1='trying variable $num1';
```

```
4.  $str2='trying backslash n and backslash t inside single quoted string \n \t';
5.  $str3='Using single quote \'my quote\' and \\backslash';
6.  echo "$str1 <br/> $str2 <br/> $str3";
7.  ?>
```

Output:

```
trying variable $num1
trying backslash n and backslash t inside single quoted string \n \t
Using single quote 'my quote' and \backslash
```

Note: *In single quoted PHP strings, most escape sequences and variables will not be interpreted. But, we can use single quote through \' and backslash through \\ inside single quoted PHP strings.*

Double Quoted

In PHP, we can specify string through enclosing text within double quote also. But escape sequences and variables will be interpreted using double quote PHP strings.

Example 1

```
1.  <?php
2.  $str="Hello text within double quote";
3.  echo $str;
4.  ?>
```

Output:

```
Hello text within double quote
```

Now, you **can't use double quote directly** inside double quoted string.

Example 2

```
1.  <?php
2.  $str1="Using double "quote" directly inside double quoted string";
3.  echo $str1;
4.  ?>
```

Output:

```
Parse error: syntax error, unexpected 'quote' (T_STRING) in
C:\wamp\www\string1.php on line 2
```

We **can store multiple line text, special characters and escape sequences** in a double quoted PHP string.

Example 3

```
1.  <?php
```

```
$str1="Hello text
multiple line
text within double quoted string";
$str2="Using double \"quote\" with backslash inside double quoted string";
$str3="Using escape sequences \n in double quoted string";
echo "$str1 <br/> $str2 <br/> $str3";
?>
```

Output:

```
Hello text multiple line text within double  quoted string
Using double "quote" with backslash inside double quoted string
Using escape sequences in double quoted string
```

In double quoted strings, **variable will be interpreted**.

Example 4

```
<?php
$num1=10;
echo "Number is: $num1";
?>
```

Output:

```
Number is: 10
```

Heredoc

Heredoc syntax (<<<) is the third way to delimit strings. In Heredoc syntax, an identifier is provided after this heredoc <<< operator, and immediately a new line is started to write any text. To close the quotation, the string follows itself and then again that same identifier is provided. That closing identifier must begin from the new line without any whitespace or tab.

Naming Rules

The identifier should follow the naming rule that it must contain only alphanumeric characters and underscores, and must start with an underscore or a non-digit character.

For Example

Valid Example

```
<?php
    $str = <<<Demo
It is a valid example
Demo;    //Valid code as whitespace or tab is not valid before closing identifier
echo $str;
?>
```

Output:

```
It is a valid example
```

Invalid Example

We cannot use any whitespace or tab before and after the identifier and semicolon, which means identifier must not be indented. The identifier must begin from the new line.

```
<?php
    $str = <<<Demo
It is Invalid example
        Demo;    //Invalid code as whitespace or tab is not valid before closing identifier
echo $str;
?>
```

This code will generate an error.

Output:

```
Parse error: syntax error, unexpected end of file in
C:\xampp\htdocs\xampp\PMA\heredoc.php on line 7
```

Heredoc is similar to the double-quoted string, without the double quote, means that quote in a heredoc are not required. It can also print the variable's value.

Example

```
<?php
    $city = 'Delhi';
    $str = <<<DEMO
Hello! My name is Misthi, and I live in $city.
DEMO;
 echo $str;7.
 ?>
```

Output:

```
Hello! My name is Misthi, and I live in Delhi.
```

Example

We can add multiple lines of text here between heredoc syntax.

```
<?php
    $str = <<<DEMO
It is the example
of multiple
lines of text.
DEMO;
    echo $str;

echo '</br>';

echo <<<DEMO    // Here we are not storing string content in variable str.
It is the example
of multiple
lines of text.
DEMO;
?>
```

Output:

```
It is the example of  multiple  lines  of  text.
It is the example of multiple lines of text.
```

Below are the example with class and their variable

Example

```
<?php
class heredocExample{
        var $demo;
        var $example;
        function __construct()
        {
                $this->demo = 'DEMO';
        $this->example = array('Example1', 'Example2', 'Example3');
        }
    }
    $heredocExample = new heredocExample();
    $name = 'Gunjan';

    echo <<<ECO
    My name is "$name". I am printing some $heredocExample->demo example.
    Now, I am printing {$heredocExample->example[1]}.
    It will print a capital 'A': \x41
 ECO;
?>
```

Output:

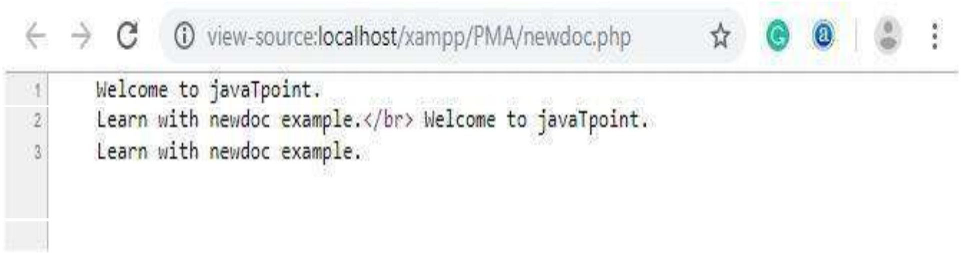
```
My name is "Gunjan". I am printing some DEMO example.
Now, I am printing Example2.
It will print a capital 'A': A
```

Newdoc

Newdoc is similar to the heredoc, but in newdoc parsing is not done. It is also identified with three less than symbols <<< followed by an identifier. But here identifier is enclosed in single-quote, **e.g. <<<'EXP'**. Newdoc follows the same rule as heredocs.

The difference between newdoc and heredoc is that - Newdoc is a **single-quoted string** whereas heredoc is a **double-quoted string**.

Note: *Newdoc works as single quotes.*



```
Welcome to javaTpoint.
Learn with newdoc example.</br> Welcome to javaTpoint.
Learn with newdoc example.
```

Example-1:

```
<?php
   $str = <<<'DEMO'
   Welcome to javaTpoint.
           Learn with newdoc example.
DEMO;
echo $str;
echo '</br>';

echo <<< 'Demo'     // Here we are not storing string content in variable str.
   Welcome to javaTpoint.
           Learn with newdoc example.
Demo;
?>
```

Output:

```
Welcome to javaTpoint. Learn with newdoc example.
Welcome to javaTpoint. Learn with newdoc example.
```

Example

The below example shows that newdoc does not print the variable's value.

```
<?php
class heredocExample{
        var $demo;
        var $example;
        function __construct()
        {
                $this->demo = 'DEMO';
        $this->example = array('Example1', 'Example2', 'Example3');
        }
    }
    $heredocExample = new heredocExample();
    $name = 'Gunjan';

    echo <<<ECO
    My name is "$name". I am printing some $heredocExample->demo example.
    Now, I am printing {$heredocExample->example[1]}.
    It will print a capital 'A': \x41
ECO;
?>
```

Output:

The output of the above program will be like:

```
My name is "$name". I am printing some $heredocExample->demo example.
Now, I am printing {$heredocExample->example[1]}.
It will print  a  capital  'A':  \x41
```

Note: *newdoc supported by PHP 5.3.0+ versions.*

Invalid Example

We cannot use any whitespace or tab before and after the identifier and semicolon, means identifier must not be indented. The identifier must begin from the new line. It is also invalid in newdoc same as heredoc.

```
<?php
    $str = <<<'Demo'
It is Invalid example
Demo; //Invalid code as whitespace or tab is not valid before closing identifier
echo $str;
?>
```

This code will generate an error.

Output:

```
Parse error: syntax error, unexpected end of file in
C:\xampp\htdocs\xampp\PMA\newdoc.php on line 7
```

PHP String Function Examples

1) PHP strtolower() function

The strtolower() function returns string in lowercase letter.

Syntax

```
string strtolower ( string $string )
```

Example

```
<?php
$str="My name is KHAN";
$str=strtolower($str);
echo $str;
?>
```

Output:

```
my name is  khan
```

2) PHP strtoupper() function

The strtoupper() function returns string in uppercase letter.

Syntax

```
string strtoupper ( string $string )
```

Example

```
<?php
$str="My name is KHAN";
$str=strtoupper($str);
echo $str;
?>
```

Output:

```
MY NAME IS  KHAN
```

3) PHP ucfirst() function

The ucfirst() function returns string converting first character into uppercase. It doesn't change the case of other characters.

Syntax

```
string ucfirst ( string $str )
```

Example

```
<?php
$str="my name is KHAN";
$str=ucfirst($str);
echo $str;
```

```
?>
```

Output:

```
My name is  KHAN
```

4) PHP lcfirst() function

The lcfirst() function returns string converting first character into lowercase. It doesn't change the case of other characters.

Syntax

```
string lcfirst ( string $str )
```

Example

```
<?php
$str="MY name IS KHAN";
$str=lcfirst($str);
echo $str;
?>
```

Output:

```
mY name IS  KHAN
```

5) PHP ucwords() function

The ucwords() function returns string converting first character of each word into uppercase.

Syntax

```
string ucwords ( string $str )
```

Example

```
<?php
$str="my name is Sonoo jaiswal";
$str=ucwords($str);
echo $str;
?>
```

Output:

```
My Name Is  Sonoo  Jaiswal
```

6) PHP strrev() function

The strrev() function returns reversed string.

Syntax

```
string strrev ( string $string )
```

Example

```
<?php
$str="my name is Sonoo jaiswal";
$str=strrev($str);
echo $str;
?>
```

Output:

```
lawsiaj oonoS si  eman  ym
```

7) PHP strlen() function

The strlen() function returns length of the string.

Syntax

```
int strlen ( string $string )
```

Example

```
<?php
$str="my name is Sonoo jaiswal";
$str=strlen($str);
echo $str;
?>
```

Output:

```
24
```

PHP Math

PHP provides many predefined math constants and functions that can be used to perform mathematical operations.

PHP Math: abs() function

The abs() function returns absolute value of given number. It returns an integer value but if you pass floating point value, it returns a float value.

Syntax

```
number abs ( mixed $number )
```

Example

```
<?php
echo (abs(-7)."<br/>"); // 7 (integer)
echo (abs(7)."<br/>"); //7 (integer)
echo (abs(-7.2)."<br/>"); //7.2 (float/double)
?>
```

Output:

```
7
7
7.2
```

PHP Math: ceil() function

The ceil() function rounds fractions up.

Syntax

```
float ceil ( float $value )
```

Example

```
<?php
echo (ceil(3.3)."<br/>");// 4
echo (ceil(7.333)."<br/>");// 8
echo (ceil(-4.8)."<br/>");// -4
?>
```

Output:

```
4
8
-4
```

PHP Math: floor() function

The floor() function rounds fractions down.

Syntax

```
float floor ( float $value )
```

Example

```
<?php
echo (floor(3.3)."<br/>");// 3
echo (floor(7.333)."<br/>");// 7
echo (floor(-4.8)."<br/>");// -5
?>
```

Output:

```
3
7
-5
```

PHP Math: sqrt() function

The sqrt() function returns square root of given argument.

Syntax

```
float sqrt ( float $arg )
```

Example

```
<?php
echo (sqrt(16)."<br/>");// 4
echo (sqrt(25)."<br/>");// 5
echo (sqrt(7)."<br/>");// 2.6457513110646
?>
```

Output:

```
4
5
2.6457513110646
```

PHP Math: decbin() function

The decbin() function converts decimal number into binary. It returns binary number as a string.

Syntax

```
string decbin ( int $number )
```

Example

```
<?php
echo (decbin(2)."<br/>");// 10
echo (decbin(10)."<br/>");// 1010
echo (decbin(22)."<br/>");// 10110
?>
```

Output:

```
10
1010
10110
```

PHP Math: dechex() function

The dechex() function converts decimal number into hexadecimal. It returns hexadecimal representation of given number as a string.

Syntax

```
string dechex ( int $number )
```

Example

```
<?php
  echo  (dechex(2)."<br/>");//  2
  echo (dechex(10)."<br/>");// a
echo (dechex(22)."<br/>");// 16
?>
```

Output:

```
2
a
16
```

PHP Math: decoct() function

The decoct() function converts decimal number into octal. It returns octal representation of given number as a string.

Syntax

```
string decoct ( int $number )
```

Example

```
<?php
echo (decoct(2)."<br/>");// 2
echo (decoct(10)."<br/>");// 12
echo (decoct(22)."<br/>");// 26
?>
```

Output:

```
2
12
26
```

PHP Math: base_convert() function

The base_convert() function allows you to convert any base number to any base number. For example, you can convert hexadecimal number to binary, hexadecimal to octal, binary to octal, octal to hexadecimal, binary to decimal etc.

Syntax

```
string base_convert ( string $number , int $frombase , int $tobase )
```

Example

```
<?php
$n1=10;
echo (base_convert($n1,10,2)."<br/>");// 1010
?>
```

Output:

```
1010
```

PHP Math: bindec() function

The bindec() function converts binary number into decimal.

Syntax

1. number bindec (string $binary_string)

Example

```
<?php
echo (bindec(10)."<br/>");// 2
echo (bindec(1010)."<br/>");// 10
echo (bindec(1011)."<br/>");// 11
?>
```

Output:

```
2
10
11
```

PHP - OPERATOR TYPES

What is Operator?

Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. PHP language supports following type of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Lets have a look on all operators one by one.

Arithmetic Operators

There are following arithmetic operators supported by PHP language −

Assume variable A holds 10 and variable B holds 20 then −

Show Examples

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by de-numerator	B / A will give 2

%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Comparison Operators

There are following comparison operators supported by PHP language

Assume variable A holds 10 and variable B holds 20 then −

Show Examples

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

There are following logical operators supported by PHP language

Assume variable A holds 10 and variable B holds 20 then −

Show Examples

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

Assignment Operators

There are following assignment operators supported by PHP language −

Show Examples

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the	C /= A is equivalent

	result to left operand	to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Conditional Operator

There is one more operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditional operator has this syntax −

Show Examples

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Operators Categories

All the operators we have discussed above can be categorised into following categories −

- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.

Precedence of PHP Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator −

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right

Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=	Right to left

PHP - EXPRESSIONS

An expression is a combination of values, variables, operators, and functions that results in a value. For example,

y = 3(abs(2*x*) + 4)

which in PHP would be:

$y = 3 * (abs(2 * $x) + 4);

The value returned (*y*, or $y in this case) can be a number, a string, or a *Boolean value*

TRUE or FALSE?

A basic Boolean value can be either TRUE or FALSE. For example, the expression “20 > 9” (20 is greater than 9) is TRUE, and the expression “5 == 6” (5 is equal to 6) is FALSE. (You can combine Boolean operations using operators such as AND, OR, and XOR)

Following example shows some simple expressions:

Example 4-1. Four simple Boolean expressions

```
<?php
  echo "a: [" . (20 > 9) . "]<br>";
  echo "b: [" . (5 == 6) . "]<br>";
  echo "c: [" . (1 == 0) . "]<br>";
  echo "d: [" . (1 == 1) . "]<br>";
?>
```

The output from this code is as follows:

```
a: [1]
b: []
c: []
d: [1]
```

Notice that both expressions a: and d: evaluate to TRUE, which has a value of 1. But b: and c:, which evaluate to FALSE, do not show any value, because in PHP the constant FALSE is defined as NULL, or nothing.

Example 4-2. Outputting the values of TRUE and FALSE

```
<?php // test2.php
  echo "a: [" . TRUE  . "]<br>";
  echo "b: [" . FALSE . "]<br>";
?>
```

which outputs the following:

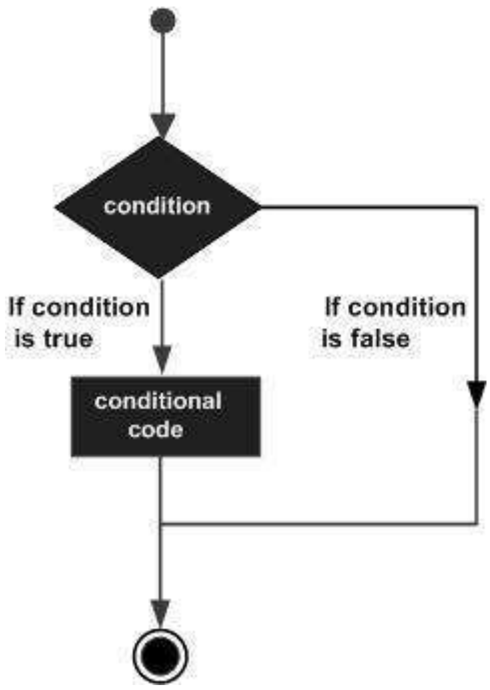
```
a: [1]
b: []
```

By the way, in some languages FALSE may be defined as 0 or even −1, so it's worth checking on its definition in each language.

PHP - DECISION MAKING

The if, elseif ...else and switch statements are used to take decision based on the different condition.

You can use conditional statements in your code to make your decisions. PHP supports following three decision making statements −



- **if...else statement** − use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
- **elseif statement** − is used with the if...else statement to execute a set of code if **one** of the several condition is true
- **switch statement** − is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if... else statement.

Syntax

```
if (condition)
   code to be executed if condition is true;
else
   code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, Otherwise, it will output "Have a nice day!":

```
<html>
   <body>

      <?php
         $d = date("D");

         if ($d == "Fri")
            echo "Have a nice weekend!";

         else
            echo "Have a nice day!";
      ?>

   </body>
</html>
```

It will produce the following result −

```
Have a nice weekend!
```

The ElseIf Statement

If you want to execute some code if one of the several conditions are true use the elseif statement

Syntax

```
if (condition)
   code to be executed if condition is true;
elseif (condition)
   code to be executed if condition is true;
else
   code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise, it will output "Have a nice day!" −

```
    <?php
       $d = date("D");

       if ($d == "Fri")
          echo "Have a nice weekend!";

       elseif ($d == "Sun")
          echo "Have a nice Sunday!";

       else
          echo "Have a nice day!";
    ?>

  </body>
</html>
```

It will produce the following result −

```
Have a nice Weekend!
```

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.

The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax

```
switch (expression){
   case label1:
      code to be executed if expression = label1;
      break;

   case label2:
      code to be executed if expression = label2;
      break;
      default:

   code to be executed
   if expression is different
   from both label1 and label2;
}
```

Example

The *switch* statement works in an unusual way. First it evaluates given expression then seeks a lable to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the lable matches then statement will execute any specified default code.

```
    <?php
      $d = date("D");

      switch ($d){
        case "Mon":
          echo "Today is Monday";
          break;

        case "Tue":
          echo "Today is Tuesday";
          break;

        case "Wed":
          echo "Today is Wednesday";
          break;

        case "Thu":
          echo "Today is Thursday";
          break;

        case "Fri":
          echo "Today is Friday";
          break;

        case "Sat":
          echo "Today is Saturday";
          break;

        case "Sun":
          echo "Today is Sunday";
          break;

        default:
          echo "Wonder which day is this ?";
      }
    ?>

  </body>
</html>
```

It will produce the following result −

```
Today is Monday
```

PHP - LOOP TYPES

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

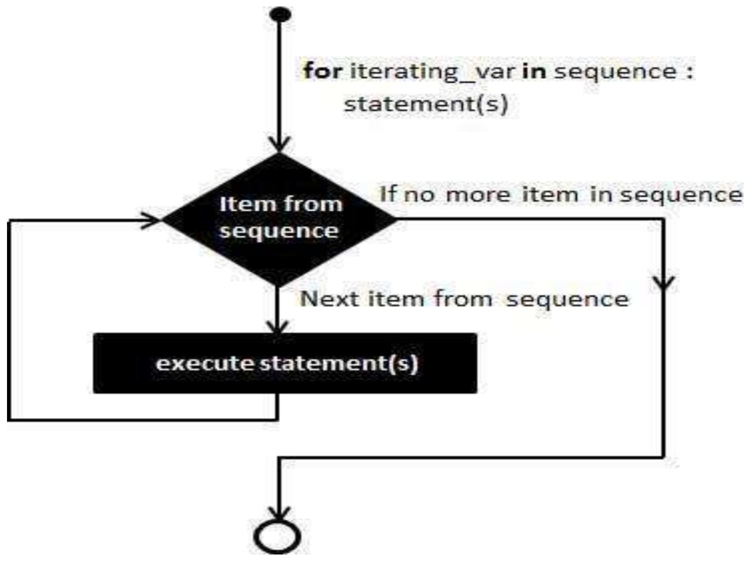
- **for** − loops through a block of code a specified number of times.
- **while** − loops through a block of code if and as long as a specified condition is true.
- **do...while** − loops through a block of code once, and then repeats the loop as long as a special condition is true.

- **foreach** − loops through a block of code for each element in an array.

We will discuss about **continue** and **break** keywords used to control the loops execution.

for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.



Syntax

for (*initialization*; *condition*; *increment*){
  code to be executed;
}

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it $i.

Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop −

```
<html>
  <body>

    <?php
      $a = 0;
      $b = 0;
      for( $i = 0; $i<5; $i++ ) {
        $a += 10;
        $b += 5;
      }
      echo ("At the end of the loop a = $a and b = $b" );
    ?>

  </body>
</html>
```

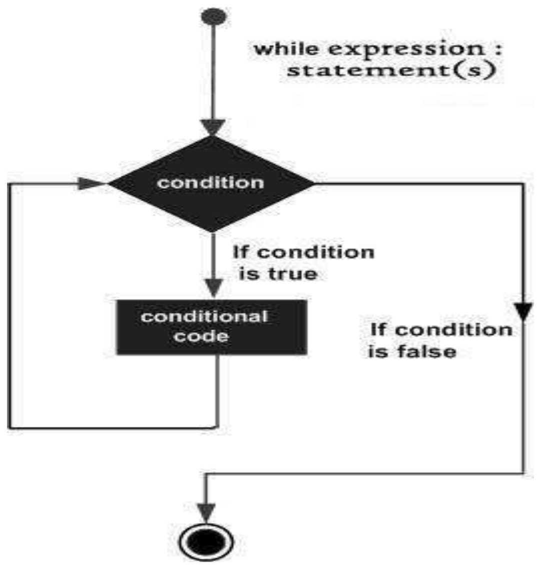
This will produce the following result −

```
At the end of the loop a = 50 and b = 25
```

The while loop statement

The while statement will execute a block of code if and as long as a test expression is true.

If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.



Syntax

```
while (condition) {
  code to be executed;
}
```

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

```
<html>
  <body>

    <?php
      $i = 0;
      $num = 50;

      while( $i < 10) {
        $num--;
        $i++;
      }

      echo ("Loop stopped at i = $i and num = $num" );
    ?>

  </body>
</html>
```

This will produce the following result −

```
Loop stopped at i = 10 and num = 40
```

The do...while loop statement

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

Syntax

```
do {
   code to be executed;
}
while (condition);
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10 −

```
<html>
   <body>
   
      <?php
         $i = 0;
         $num = 0;
         
         do {
            $i++;
         }
         while( $i < 10 );
         echo ("Loop stopped at i = $i" );
      ?>
   
   </body>
</html>
```

The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

foreach (*array* as *value*) {
  code to be executed;
}

Example

Try out following example to list out the values of an array.

```
<html>
   <body>

      <?php
         $array = array( 1, 2, 3, 4, 5);

         foreach( $array as $value ) {
            echo "Value is $value <br />";
         }
      ?>

   </body>
</html>
```

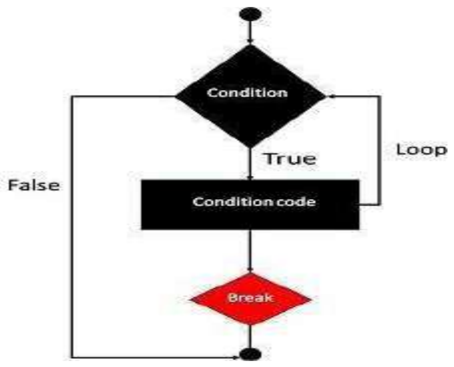
This will produce the following result −

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely.

The **break** statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.



Example

In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

```
<html>
  <body>

    <?php
      $i = 0;

      while( $i < 10) {
        $i++;
        if( $i == 3 )break;
      }
      echo ("Loop stopped at i = $i" );
    ?>

  </body>
</html>
```

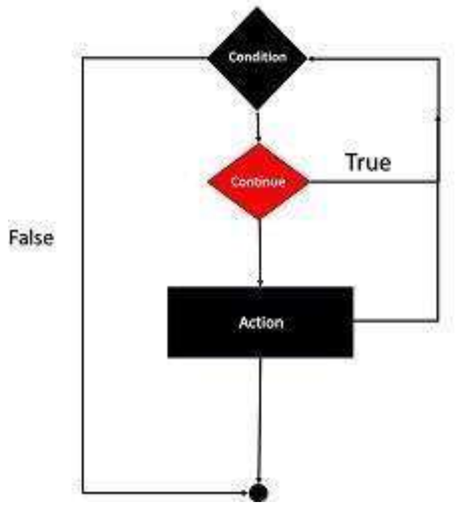
This will produce the following result −

```
Loop stopped at i = 3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.



Example

In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

```
<html>
  <body>

    <?php
      $array = array( 1, 2, 3, 4, 5);

      foreach( $array as $value ) {
        if( $value == 3 )continue;
        echo "Value is $value <br />";
      }
    ?>

  </body>
</html>
```

This will produce the following result −

```
Value is 1
Value is 2
Value is 4
Value is 5
```

PHP - FUNCTIONS

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you −

- Creating a PHP Function
- Calling a PHP Function

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

Please refer to PHP Function Reference for a complete set of useful functions.

Creating PHP Function

Its very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below −

```
<html>

   <head>
      <title>Writing PHP Function</title>
   </head>

   <body>

      <?php
         /* Defining a PHP Function */
         function writeMessage() {
            echo "You are really a nice person, Have a nice time!";
         }

         /* Calling a PHP Function */
         writeMessage();
      ?>

   </body>
</html>
```

This will display following result −

```
You are really a nice person, Have a nice time!
```

PHP Functions with Parameters

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

```
<html>

   <head>
      <title>Writing PHP Function with Parameters</title>
   </head>

   <body>

      <?php
         function addFunction($num1, $num2) {
            $sum = $num1 + $num2;
            echo "Sum of the two numbers is : $sum";
         }

         addFunction(10, 20);
      ?>

   </body>
</html>
```

This will display following result −

```
Sum of the two numbers is : 30
```

Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

Following example depicts both the cases.

```
<html>

   <head>
      <title>Passing Argument by Reference</title>
   </head>

   <body>

      <?php
         function addFive($num) {
            $num += 5;
         }

         function addSix(&$num) {
            $num += 6;
         }

         $orignum = 10;
         addFive( $orignum );

         echo "Original Value is $orignum<br />";
         addSix( $orignum );
         echo "Original Value is $orignum<br />";
      ?>

   </body>
</html>
```

This will display following result −

```
Original Value is 10
Original Value is 16
```

PHP Functions returning value

A function can return a value using the **return** statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.

You can return more than one value from a function using **return array(1,2,3,4)**.

Following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

```
<html>

  <head>
    <title>Writing PHP Function which returns value</title>
  </head>

  <body>

    <?php
      function addFunction($num1, $num2) {
        $sum = $num1 + $num2;
        return $sum;
      }
      $return_value = addFunction(10, 20);

      echo "Returned value from the function : $return_value";
    ?>

  </body>
</html>
```

This will display following result −

```
Returned value from the function : 30
```

Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it.

Following function prints NULL in case use does not pass any value to this

function.

```
<html>
   <head><title>Writing PHP Function which returns value
   </title></head>
   <body>
      <?php
         function printMe($param = NULL) {
            print $param;
         }
         printMe("This is test");
         printMe();
      ?>
   </body>
</html>
```

This will produce following result −

```
This is test
```

Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. Following example depicts this behaviour.

```
<html>

   <head>
      <title>Dynamic Function Calls</title>
   </head>

   <body>

      <?php
         function sayHello() {
            echo "Hello<br />";
         }

         $function_holder = "sayHello";
         $function_holder();
      ?>

   </body>
</html>
```

This will display following result −

```
Hello
```

PHP - FORM INTRODUCTION

Dynamic Websites

The Websites provide the functionalities that can use to store, update, retrieve, and delete the data in a database.

What is the Form?

A Document that containing black fields, that the user can fill the data or user can select the data.Casually the data will store in the data base

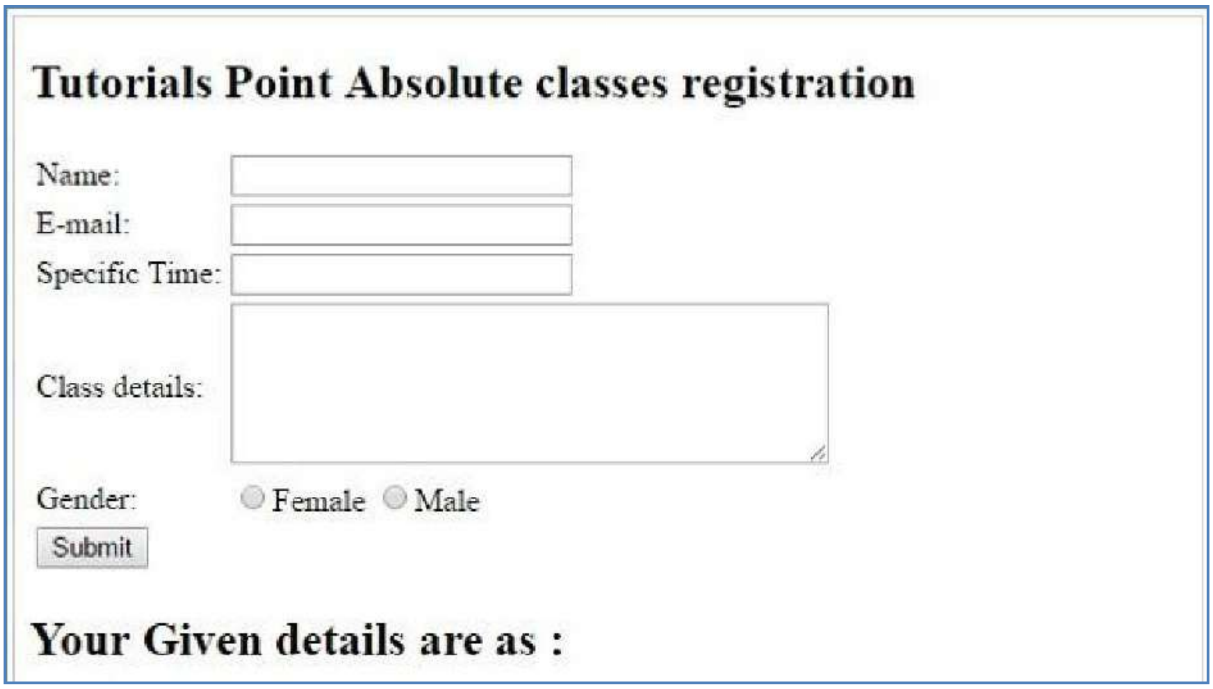
Example

Below example shows the form with some specific actions by using post method.

```
<html>
  <head>
    <title>PHP Form Validation</title>
  </head>

  <body>
    <?php
      // define variables and set to empty values
      $name = $email = $gender = $comment = $website = "";
      if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $name = test_input($_POST["name"]);
        $email = test_input($_POST["email"]);
        $website = test_input($_POST["website"]);
        $comment = test_input($_POST["comment"]);
        $gender = test_input($_POST["gender"]);
      }

      function test_input($data) {
        $data = trim($data);
        $data = stripslashes($data);
        $data = htmlspecialchars($data);
        return $data;
      }
    ?>
    <h2>Tutorials Point Absolute classes registration</h2>

    <form method = "post" action = "/php/php_form_introduction.htm">
      <table>
        <tr>
          <td>Name:</td>
          <td><input type = "text" name = "name"></td>
        </tr>

        <tr>
          <td>E-mail:</td>
          <td><input type = "text" name = "email"></td>
        </tr>
```

```
         <tr>
            <td>Specific Time:</td>
            <td><input type = "text" name = "website"></td>
         </tr>

         <tr>
            <td>Class details:</td>
            <td><textarea name = "comment" rows = "5" cols = "40"></textarea></td>
         </tr>

         <tr>
            <td>Gender:</td>
            <td>
               <input type = "radio" name = "gender" value = "female">Female
               <input type = "radio" name = "gender" value = "male">Male
            </td>
         </tr>

         <tr>
            <td>
               <input type = "submit" name = "submit" value = "Submit">
            </td>
         </tr>
      </table>
   </form>

   <?php
      echo "<h2>Your Given details are as :</h2>";
      echo $name;
      echo "<br>";

      echo $email;
      echo "<br>";

      echo $website;
      echo "<br>";

      echo $comment;
      echo "<br>";

      echo $gender;
   ?>

  </body>
</html>
```

It will produce the following result −



Tutorials Point Absolute classes registration

Name:	
E-mail:	
Specific Time:	
Class details:	
Gender:	Female Male

Your Given details are as :

PHP - VALIDATION EXAMPLE

equired field will check whether the field is filled or not in the proper way. Most of cases we will use the ***** symbol for required field.

What is Validation ?

Validation means check the input submitted by the user. There are two types of validation are available in PHP. They are as follows −

- **Client-Side Validation** − Validation is performed on the client machine web browsers.
- **Server Side Validation** − After submitted by data, The data has sent to a server and perform validation checks in server machine.

Some of Validation rules for field

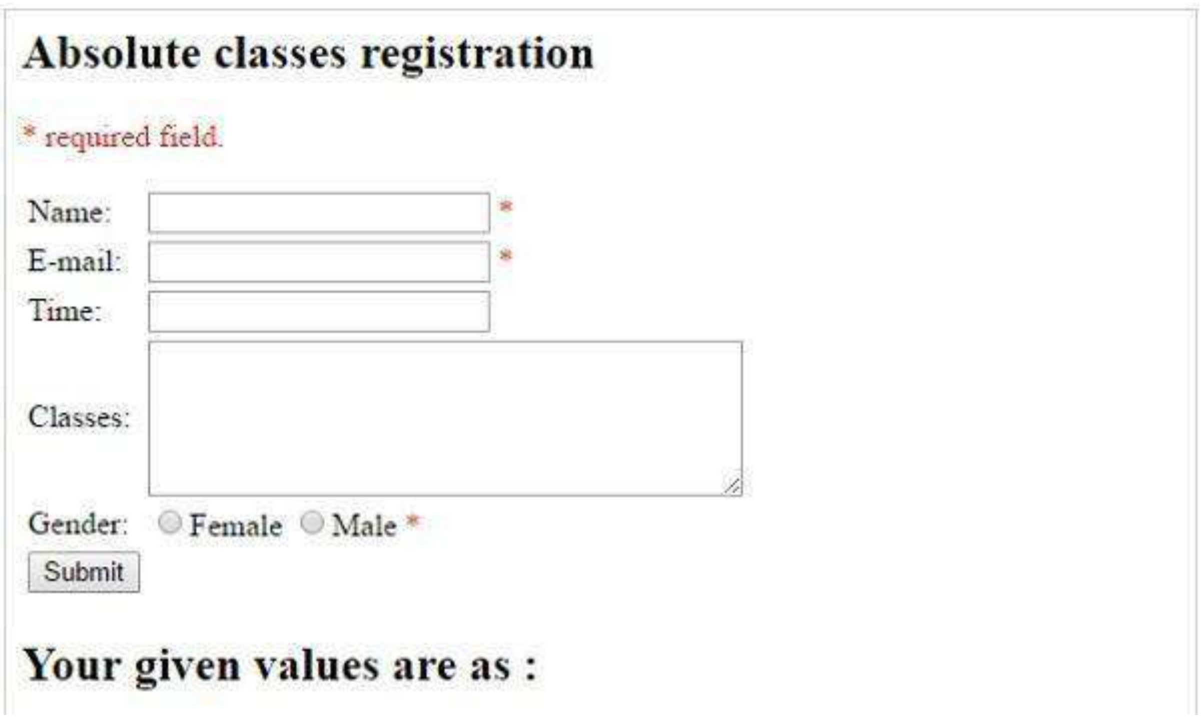
Field	Validation Rules
Name	Should required letters and white-spaces
Email	Should required @ and .
Website	Should required a valid URL
Radio	Must be selectable at least once
Check Box	Must be checkable at least once
Drop Down menu	Must be selectable at least once

Example below shows the form with required field validation

```
<html>

  <head>
    <style>
      .error {color: #FF0000;}
    </style>
  </head>

  <body>
    <?php
      // define variables and set to empty values
      $nameErr = $emailErr = $genderErr = $websiteErr = "";
      $name = $email = $gender = $comment = $website = "";

      if ($_SERVER["REQUEST_METHOD"] == "POST") {
        if (empty($_POST["name"])) {
          $nameErr = "Name is required";
        }else {
          $name = test_input($_POST["name"]);
        }

        if (empty($_POST["email"])) {
          $emailErr = "Email is required";
        }else {
          $email = test_input($_POST["email"]);

          // check if e-mail address is well-formed
          if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $emailErr = "Invalid email format";
          }
        }

        if (empty($_POST["website"])) {
          $website = "";
        }else {
          $website = test_input($_POST["website"]);
        }

        if (empty($_POST["comment"])) {
          $comment = "";
        }else {
          $comment = test_input($_POST["comment"]);
        }

        if (empty($_POST["gender"])) {
          $genderErr = "Gender is required";
        }else {
          $gender = test_input($_POST["gender"]);
        }
      }
```

```
   function test_input($data) {
      $data = trim($data);
      $data = stripslashes($data);
      $data = htmlspecialchars($data);
      return $data;
   }
?>

<h2>Absolute classes registration</h2>

<p><span class = "error">* required field.</span></p>

<form method = "post" action = "<?php
   echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
   <table>
      <tr>
         <td>Name:</td>
         <td><input type = "text" name = "name">
            <span class = "error">* <?php echo $nameErr;?></span>
         </td>
      </tr>

      <tr>
         <td>E-mail: </td>
         <td><input type = "text" name = "email">
            <span class = "error">* <?php echo $emailErr;?></span>
         </td>
      </tr>

      <tr>
         <td>Time:</td>
         <td> <input type = "text" name = "website">
            <span class = "error"><?php echo $websiteErr;?></span>
         </td>
      </tr>

      <tr>
         <td>Classes:</td>
         <td> <textarea name = "comment" rows = "5" cols = "40"></textarea></td>
      </tr>

      <tr>
         <td>Gender:</td>
         <td>
            <input type = "radio" name = "gender" value = "female">Female
            <input type = "radio" name = "gender" value = "male">Male
            <span class = "error">* <?php echo $genderErr;?></span>
         </td>
      </tr>

      <td>
         <input type = "submit" name = "submit" value = "Submit">
         </td>

       </table>

     </form>

     <?php
       echo "<h2>Your given values are as:</h2>";
       echo $name;
       echo "<br>";

       echo $email;
       echo "<br>";

       echo $website;
       echo "<br>";

       echo $comment;
       echo "<br>";

       echo $gender;
     ?>

   </body>
</html>
```

It will produce the following result −



Absolute classes registration

* required field.

Name:	*
E-mail:	*
Time:	
Classes:	
Gender:	Female Male *

Submit

Your given values are as :

PHP - COMPLETE FORM

This page explains about time real-time form with actions. Below example will take input fields as text, radio button, drop down menu, and checked box.

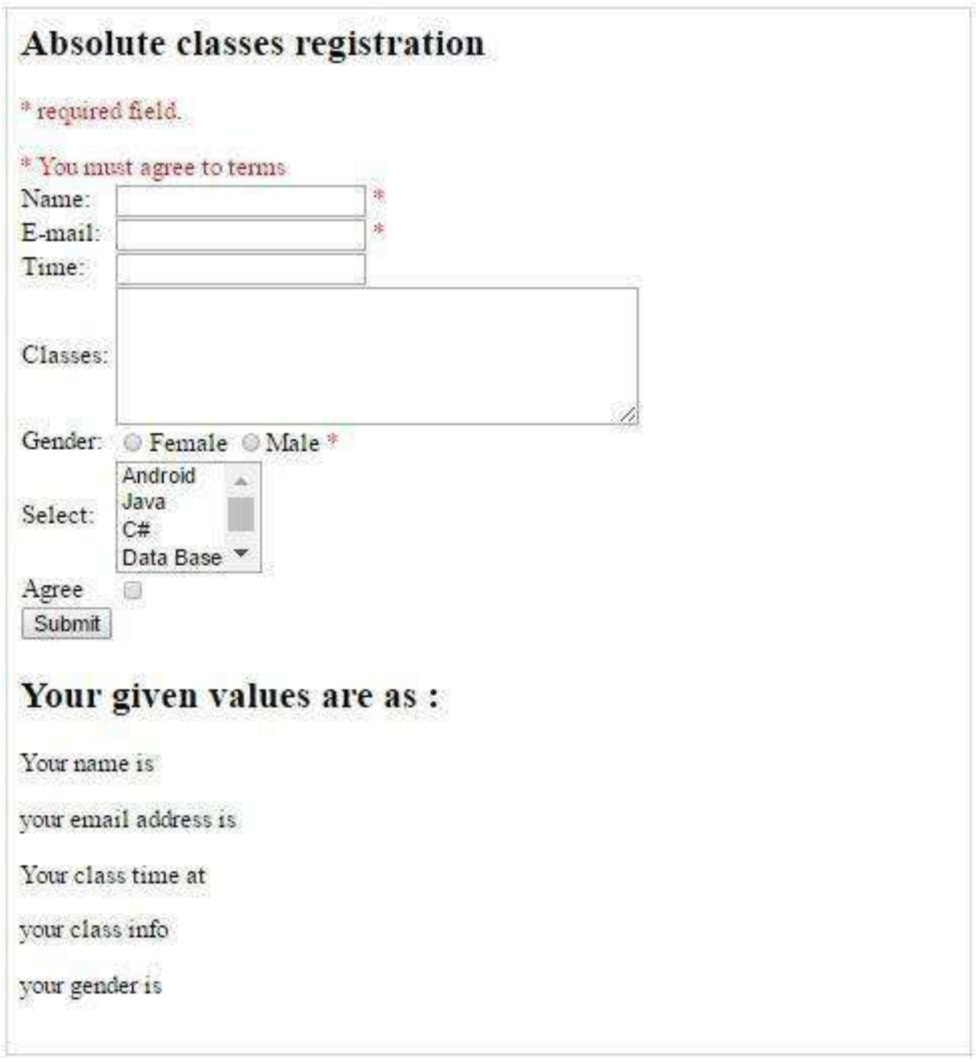
Example

```
<html>

   <head>
      <style>
         .error {color: #FF0000;}
      </style>
   </head>

   <body>
      <?php
         // define variables and set to empty values
         $nameErr = $emailErr = $genderErr = $websiteErr = "";
         $name = $email = $gender = $class = $course = $subject = "";

         if ($_SERVER["REQUEST_METHOD"] == "POST") {
            if (empty($_POST["name"])) {
               $nameErr = "Name is required";
            }else {
               $name = test_input($_POST["name"]);
            }

            if (empty($_POST["email"])) {
               $emailErr = "Email is required";
            }else {
               $email = test_input($_POST["email"]);

               // check if e-mail address is well-formed
               if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
                  $emailErr = "Invalid email format";
               }
            }

            if (empty($_POST["course"])) {
               $course = "";
            }else {
               $course = test_input($_POST["course"]);
            }

            if (empty($_POST["class"])) {
               $class = "";
            }else {
               $class = test_input($_POST["class"]);
            }

            if (empty($_POST["gender"])) {
               $genderErr = "Gender is required";
            }else {
               $gender = test_input($_POST["gender"]);
            }
```

```
          if (empty($_POST["subject"])) {
            $subjectErr = "You must select 1 or more";
          }else {
            $subject = $_POST["subject"];
          }
        }

        function test_input($data) {
          $data = trim($data);
          $data = stripslashes($data);
          $data = htmlspecialchars($data);
          return $data;
        }
      ?>

      <h2>Absolute classes registration</h2>

      <p><span class = "error">* required field.</span></p>

      <form method = "POST" action = "<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
        <table>
          <tr>
            <td>Name:</td>
            <td><input type = "text" name = "name">
              <span class = "error">* <?php echo $nameErr;?></span>
            </td>
          </tr>

          <tr>
            <td>E-mail: </td>
            <td><input type = "text" name = "email">
              <span class = "error">* <?php echo $emailErr;?></span>
            </td>
          </tr>

          <tr>
            <td>Time:</td>
            <td> <input type = "text" name = "course">
              <span class = "error"><?php echo $websiteErr;?></span>
            </td>
          </tr>

          <tr>
            <td>Classes:</td>
            <td> <textarea name = "class" rows = "5" cols = "40"></textarea></td>
          </tr>

          <tr>
            <td>Gender:</td>
            <td>
              <input type = "radio" name = "gender" value = "female">Female
```

```
                <input type = "radio" name = "gender" value = "male">Male
                <span class = "error">* <?php echo $genderErr;?></span>
             </td>
          </tr>

          <tr>
             <td>Select:</td>
             <td>
                <select name = "subject[]" size = "4" multiple>
                   <option value = "Android">Android</option>
                   <option value = "Java">Java</option>
                   <option value = "C#">C#</option>
                   <option value = "Data Base">Data Base</option>
                   <option value = "Hadoop">Hadoop</option>
                   <option value = "VB script">VB script</option>
                </select>
             </td>
          </tr>

          <tr>
             <td>Agree</td>
             <td><input type = "checkbox" name = "checked" value = "1"></td>
             <?php if(!isset($_POST['checked'])){ ?>
             <span class = "error">* <?php echo "You must agree to terms";?></span>
             <?php } ?>
          </tr>

          <tr>
             <td>
                <input type = "submit" name = "submit" value = "Submit">
             </td>
          </tr>

       </table>
    </form>

    <?php
       echo "<h2>Your given values are as :</h2>";
       echo ("<p>Your name is $name</p>");
       echo ("<p> your email address is $email</p>");
       echo ("<p>Your class time at $course</p>");
       echo ("<p>your class info $class </p>");
       echo ("<p>your gender is $gender</p>");

       for($i = 0; $i < count($subject); $i++) {
          echo($subject[$i] . " ");
       }
    ?>

  </body>
</html>
```

It will produce the following result −



PHP – HANDLING FILE UPLOADS

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script.

Information in the **phpinfo.php** page describes the temporary directory that is used for file uploads as **upload_tmp_dir** and the maximum permitted size of files that can be uploaded is stated as **upload_max_filesize**. These parameters are set into PHP configuration file **php.ini**

The process of uploading a file follows these steps −

- The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
- The user clicks the browse button and selects a file to upload from the local PC.
- The full path to the selected file appears in the text filed then the user clicks the submit button.
- The selected file is sent to the temporary directory on the server.
- The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
- The PHP script confirms the success to the user.

As usual when writing files it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only then process will fail.

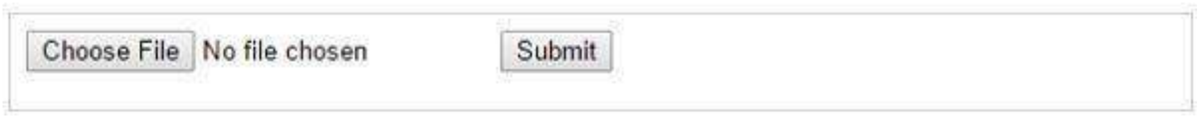
An uploaded file could be a text file or image file or any document.

Creating an upload form

The following HTM code below creates an uploader form. This form is having method attribute set to **post** and enctype attribute is set to **multipart/form-data**

```
<?php
  if(isset($_FILES['image'])){
    $errors= array();
    $file_name = $_FILES['image']['name'];
    $file_size =$_FILES['image']['size'];
    $file_tmp =$_FILES['image']['tmp_name'];
    $file_type=$_FILES['image']['type'];
    $file_ext=strtolower(end(explode('.',$_FILES['image']['name'])));

    $extensions= array("jpeg","jpg","png");

    if(in_array($file_ext,$extensions)=== false){
      $errors[]="extension not allowed, please choose a JPEG or PNG file.";
    }

    if($file_size > 2097152){
      $errors[]='File size must be excately 2 MB';
    }

    if(empty($errors)==true){
      move_uploaded_file($file_tmp,"images/".$file_name);
      echo "Success";
    }else{
      print_r($errors);
    }
  }
?>
<html>
  <body>

    <form action="" method="POST" enctype="multipart/form-data">
      <input type="file" name="image" />
      <input type="submit"/>
    </form>

  </body>
</html>
```

It will produce the following result −



Creating an upload script

There is one global PHP variable called **$_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create following five variables −

- **$_FILES['file']['tmp_name']** − the uploaded file in the temporary directory on the web server.
- **$_FILES['file']['name']** − the actual name of the uploaded file.
- **$_FILES['file']['size']** − the size in bytes of the uploaded file.
- **$_FILES['file']['type']** − the MIME type of the uploaded file.
- **$_FILES['file']['error']** − the error code associated with this file upload.

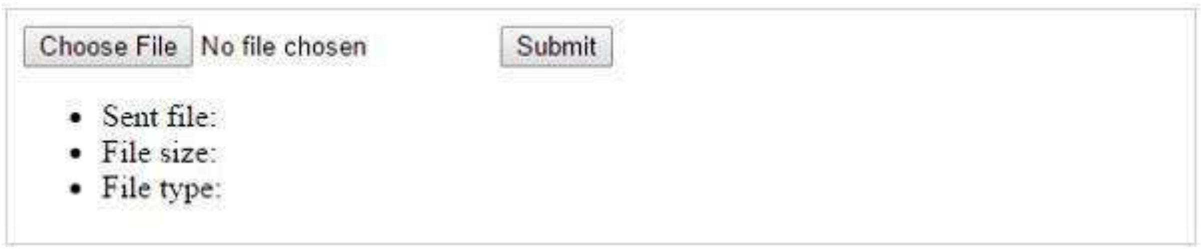
Example

Below example should allow upload images and gives back result as uploaded file information.

```
<?php
  if(isset($_FILES['image'])){
    $errors= array();
    $file_name = $_FILES['image']['name'];
    $file_size = $_FILES['image']['size'];
    $file_tmp = $_FILES['image']['tmp_name'];
    $file_type = $_FILES['image']['type'];
    $file_ext=strtolower(end(explode('.',$_FILES['image']['name'])));

    $extensions= array("jpeg","jpg","png");

    if(in_array($file_ext,$extensions)=== false){
      $errors[]="extension not allowed, please choose a JPEG or PNG file.";
    }

    if($file_size > 2097152) {
      $errors[]='File size must be excately 2 MB';
    }

    if(empty($errors)==true) {
      move_uploaded_file($file_tmp,"images/".$file_name);
      echo "Success";
    }else{
      print_r($errors);
    }
  }
?>
<html>
  <body>

    <form action = "" method = "POST" enctype = "multipart/form-data">
      <input type = "file" name = "image" />
      <input type = "submit"/>
```

```
        <ul>
           <li>Sent file: <?php echo $_FILES['image']['name'];  ?>
           <li>File size: <?php echo $_FILES['image']['size'];  ?>
           <li>File type: <?php echo $_FILES['image']['type'] ?>
        </ul>

      </form>

   </body>
</html>
```

It will produce the following result −



- Sent file:
- File size:
- File type:

PHP & MySQL

PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.

What you should already have?

- Downloaded and installed a latest version of MySQL.
- Created database user **guest** with password **guest123**.
- If you have not created a database then you would need root user and its password to create a database.

We have divided this chapter in the following sections −

MySQL DATABASE CONNECTION

PHP mysqli connect() Function

Example 1: Object Oriented style

Open a new connection to the MySQL server:

```
<?php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");

// Check connection
if ($mysqli -> connect_errno) {
  echo "Failed to connect to MySQL: " . $mysqli -> connect_error;
  exit();
}
?>
```

Definition and Usage

The connect() / mysqli_connect() function opens a new connection to the MySQL server.

Syntax

Object oriented style:

$mysqli -> new mysqli(*host, username, password, dbname, port, socket*)

Procedural style:

mysqli_connect(*host, username, password, dbname, port, socket*)

Parameter Values

Parameter	**Description**
host	Optional. Specifies a host name or an IP address
username	Optional. Specifies the MySQL username
password	Optional. Specifies the MySQL password
dbname	Optional. Specifies the default database to be used
port	Optional. Specifies the port number to attempt to connect to the MySQL server
socket	Optional. Specifies the socket or named pipe to be used

Example 2: Procedural style

Open a new connection to the MySQL server:

```
<?php
$con = mysqli_connect("localhost","my_user","my_password","my_db");

// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  exit();
}
?>
```

MySQL EXECUTING SIMPLE QUERIES, HANDLING RESULTS

PHP mysqli fetch_array() Function

Example 1: Object Oriented style

Fetch a result row as a numeric array and as an associative array:

```
<?php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");

if ($mysqli -> connect_errno) {
  echo "Failed to connect to MySQL: " . $mysqli -> connect_error;
  exit();
}

$sql = "SELECT Lastname, Age FROM Persons ORDER BY Lastname";
$result -> $mysqli -> query($sql);

// Numeric array
$row = $result -> fetch_array(MYSQLI_NUM);
printf ("%s (%s)\n", $row[0], $row[1]);

// Associative array
$row = $result -> fetch_array(MYSQLI_ASSOC);
printf ("%s (%s)\n", $row["Lastname"], $row["Age"]);

// Free result set
$result -> free_result();

$mysqli -> close();
?>
```

Definition and Usage

The fetch_array() / mysqli_fetch_array() function fetches a result row as an associative array, a numeric array, or both.

Note: Fieldnames returned from this function are case-sensitive.

Syntax

Object oriented style:

$mysqli_result -> fetch_array(*resulttype*)

Procedural style:

mysqli_fetch_array(*result*,*resulttype*)

Parameter Values

Parameter	Description
result	Required. Specifies a result set identifier returned by mysqli_query(), mysqli_store_result() or mysqli_use_result()
resulttype	Optional. Specifies what type of array that should be produced. Can be one of the following values: MYSQLI_ASSOC MYSQLI_NUM MYSQLI_BOTH (this is default)

Example 2 : Procedural style

Fetch a result row as a numeric array and as an associative array:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");

if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  exit();
}

$sql = "SELECT Lastname, Age FROM Persons ORDER BY Lastname";
$result = mysqli_query($con,$sql);

// Numeric array
$row = mysqli_fetch_array($result, MYSQLI_NUM);
printf ("%s (%s)\n", $row[0], $row[1]);

// Associative array
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
printf ("%s (%s)\n", $row["Lastname"], $row["Age"]);

// Free result set
mysqli_free_result($result);

mysqli_close($con);
?>
```

PHP mysqli fetch_field() Function

Example 1: Object Oriented style

Return the next field (column) in the result-set, then print each field's name, table, and max length:

```
<?php
```

```
$mysqli = new mysqli("localhost","my_user","my_password","my_db");

if ($mysqli -> connect_errno) {
  echo "Failed to connect to MySQL: " . $mysqli -> connect_error;
  exit();
}

$sql = "SELECT Lastname, Age FROM Persons ORDER BY Lastname";

if ($result = $mysqli -> query($sql)) {
  // Get field information for all fields
  while ($fieldinfo = $result -> fetch_field()) {
    printf("Name: %s\n", $fieldinfo -> name);
    printf("Table: %s\n", $fieldinfo -> table);
    printf("Max. Len: %d\n", $fieldinfo -> max_length);
  }
  $result -> free_result();
}

$mysqli -> close();
?>
```

Definition and Usage

The fetch_field() / mysqli_fetch_field() function returns the next field (column) in the result-set, as an object.

Syntax

Object oriented style:

$mysqli_result -> fetch_field()

Procedural style:

Example 2: Procedural style

Return the next field (column) in the result-set, then print each field's name, table, and max length:

mysqli_fetch_field(*result*)

```
<?php
$con = mysqli_connect("localhost","my_user","my_password","my_db");

if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  exit();
}
```

```
$sql = "SELECT Lastname, Age FROM Persons ORDER BY Lastname";

if ($result = mysqli_query($con, $sql)) {
  // Get field information for all fields
  while ($fieldinfo = mysqli_fetch_field($result)) {
    printf("Name: %s\n", $fieldinfo -> name);
    printf("Table: %s\n", $fieldinfo -> table);
    printf("max. Len: %d\n", $fieldinfo -> max_length);
  }
  mysqli_free_result($result);
}

mysqli_close($con);
?>
```

PHP - SESSIONS

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

The location of the temporary file is determined by a setting in the **php.ini** file called **session.save_path**. Before using any session variable make sure you have setup this path.

When a session is started following things happen −

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
- A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ ie sess_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session

A PHP session is easily started by making a call to the **session_start()** function.This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to **session_start()** at the beginning of the page.

Session variables are stored in associative array called **$_SESSION[]**. These variables can be accessed during lifetime of a session.

The following example starts a session then register a variable called **counter** that is incremented each time the page is visited during the session.

Make use of **isset()** function to check if session variable is already set or not.

Put this code in a test.php file and load this file many times to see the result −

```
<?php
  session_start();

  if( isset( $_SESSION['counter'] ) ) {
    $_SESSION['counter'] += 1;
  }else {
    $_SESSION['counter'] = 1;
  }

  $msg = "You have visited this page ".  $_SESSION['counter'];
  $msg .= "in this session.";
?>

<html>

  <head>
    <title>Setting up a PHP session</title>
  </head>

  <body>
    <?php echo ( $msg ); ?>
  </body>

</html>
```

It will produce the following result −

```
You have visited this page 1in this session.
```

Destroying a PHP Session

A PHP session can be destroyed by **session_destroy()** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable then you can use **unset()** function to unset a session variable.

Here is the example to unset a single variable −

```
<?php
  unset($_SESSION['counter']);
?>
```

Here is the call which will destroy all the session variables −

```
<?php
  session_destroy();
?>
```

Turning on Auto Session

You don't need to call start_session() function to start a session when a user visits your site if you can set **session.auto_start** variable to 1 in **php.ini** file.

Sessions without cookies

There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.

Alternatively, you can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session_name=session_id. Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

```
<?php
   session_start();

   if (isset($_SESSION['counter'])) {
      $_SESSION['counter'] = 1;
   }else {
      $_SESSION['counter']++;
   }

   $msg = "You have visited this page ".  $_SESSION['counter'];
   $msg .= "in this session.";

   echo ( $msg );
?>

<p>
   To continue  click following link <br />

   <a  href = "nextpage.php?<?php echo htmlspecialchars(SID); ?>">
</p>
```

It will produce the following result −

```
You have visited this page 1in this session.
To continue click following link
```

PHP – COOKIES

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users −

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

The Anatomy of a Cookie

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A PHP script that sets a cookie might send headers that look something like this −

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
            path=/; domain=tutorialspoint.com
Connection:      close
Content-Type: text/html
```

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server.The browser's headers might look something like this −

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

A PHP script will then have access to the cookie in the environmental variables $_COOKIE or $HTTP_COOKIE_VARS[] which holds all cookie names and values. Above cookie can be accessed using $HTTP_COOKIE_VARS["name"].

Setting Cookies with PHP

PHP provided **setcookie()** function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

```
setcookie(name, value, expire, path, domain, security);
```

Here is the detail of all the arguments −

- **Name** − This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value** − This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** − This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path** − This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** − This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.

- **Security** − This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies **name** and **age** these cookies will be expired after one hour.

```
<?php
   setcookie("name", "John Watkin", time()+3600, "/","", 0);
   setcookie("age", "36", time()+3600, "/", "",  0);
?>
<html>

   <head>
      <title>Setting Cookies with PHP</title>
   </head>

   <body>
      <?php echo "Set Cookies"?>
   </body>

</html>
```

Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either $_COOKIE or $HTTP_COOKIE_VARS variables. Following example will access all the cookies set in above example.

```
<html>

   <head>
      <title>Accessing Cookies with PHP</title>
   </head>

   <body>

      <?php
         echo $_COOKIE["name"]. "<br />";

         /* is equivalent to */
         echo $HTTP_COOKIE_VARS["name"]. "<br />";

         echo $_COOKIE["age"] . "<br />";

         /* is equivalent to */
         echo $HTTP_COOKIE_VARS["age"] . "<br />";
      ?>

   </body>
</html>
```

You can use **isset()** function to check if a cookie is set or not.

```
<html>

   <head>
```

```
      <title>Accessing Cookies with PHP</title>
   </head>

   <body>

      <?php
         if( isset($_COOKIE["name"]))
            echo "Welcome " . $_COOKIE["name"] . "<br />";

         else
            echo "Sorry... Not recognized" . "<br />";
      ?>

   </body>
</html>
```

Deleting Cookie with PHP

Officially, to delete a cookie you should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired −

```
<?php
   setcookie( "name", "", time()- 60, "/","", 0);
   setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>

   <head>
      <title>Deleting Cookies with PHP</title>
   </head>

   <body>
      <?php echo "Deleted Cookies" ?>
   </body>

</html>
```

PHP - FILES & I/O

This section will explain following functions related to files −

- Opening a file
- Reading a file
- Writing a file
- Closing a file

Opening and Closing Files

The PHP **fopen()** function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Files modes can be specified as one of the six options in this table.

Sl.No	Mode & Purpose
1	r Opens the file for reading only. Places the file pointer at the beginning of the file.
2	r+ Opens the file for reading and writing. Places the file pointer at the beginning of the file.
3	w Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
4	w+ Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
5	a Opens the file for writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.
6	a+ Opens the file for reading and writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.

If an attempt to open a file fails then **fopen** returns a value of **false** otherwise it returns a **file pointer** which is used for further reading or writing to that file.

After making a changes to the opened file it is important to close it with the **fclose()** function. The **fclose()** function requires a file pointer as its argument and then returns **true** when the closure succeeds or **false** if it fails.

Reading a file

Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The files length can be found using the **filesize()** function which takes the file name as its argument and returns the size of the file expressed in bytes.

So here are the steps required to read a file with PHP.

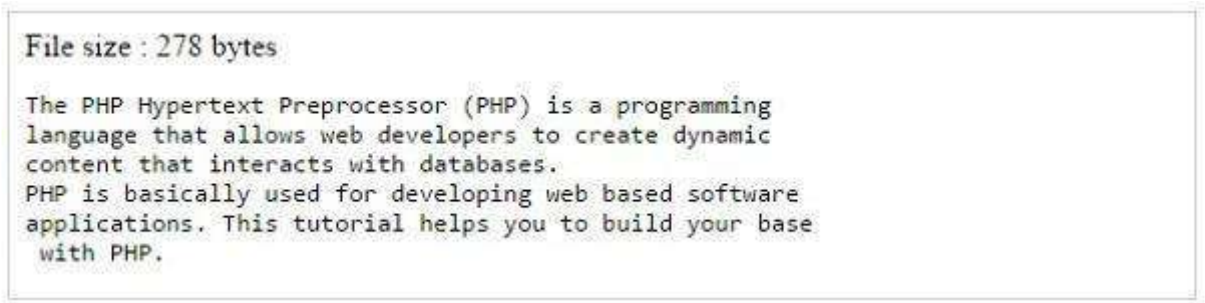
- Open a file using **fopen()** function.
- Get the file's length using **filesize()** function.

- Read the file's content using **fread()** function.
- Close the file with **fclose()** function.

The following example assigns the content of a text file to a variable then displays those contents on the web page.

```
<html>

  <head>
    <title>Reading a file using PHP</title>
  </head>

  <body>

    <?php
      $filename = "tmp.txt";
      $file = fopen( $filename, "r" );

      if( $file == false ) {
        echo ( "Error in opening file" );
        exit();
      }

      $filesize = filesize( $filename );
      $filetext = fread( $file, $filesize );
      fclose( $file );

      echo ( "File size : $filesize bytes" );
      echo ( "<pre>$filetext</pre>" );
    ?>

  </body>
</html>
```

It will produce the following result −



```
File size : 278 bytes

The PHP Hypertext Preprocessor (PHP) is a programming
language that allows web developers to create dynamic
content that interacts with databases.
PHP is basically used for developing web based software
applications. This tutorial helps you to build your base
 with PHP.
```

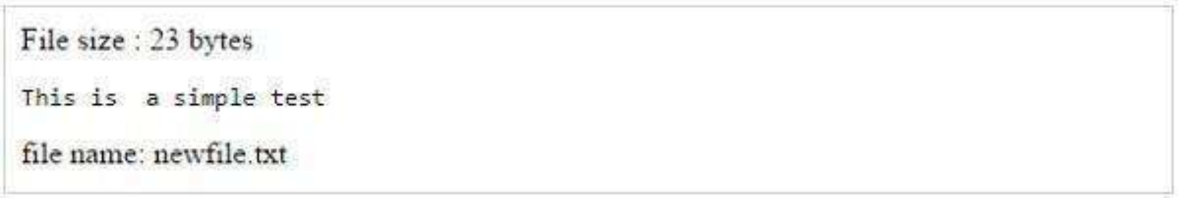
Writing a file

A new file can be written or text can be appended to an existing file using the PHP **fwrite()** function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included, writing would will stop after the specified length has been reached.

The following example creates a new text file then writes a short text heading inside it. After closing this file its existence is confirmed using **file_exist()** function which takes file name as an argument

```
<?php
   $filename = "/home/user/guest/newfile.txt";
   $file = fopen( $filename, "w" );

   if( $file == false ) {
      echo ( "Error in opening new file" );
      exit();
   }
   fwrite( $file, "This is a simple test\n" );
   fclose( $file );
?>
<html>

   <head>
      <title>Writing a file using PHP</title>
   </head>

   <body>

      <?php
         $filename = "newfile.txt";
         $file = fopen( $filename, "r" );

         if( $file == false ) {
            echo ( "Error in opening file" );
            exit();
         }

         $filesize = filesize( $filename );
         $filetext = fread( $file, $filesize );

         fclose( $file );

         echo ( "File size : $filesize bytes" );
         echo ( "$filetext" );
         echo("file name: $filename");
      ?>

   </body>
</html>
```

It will produce the following result −



```
File size : 23 bytes

This is  a simple test

file name: newfile.txt
```

PHP FILE HANDLING

PHP File System allows us to create file, read file line by line, read file character by character, write file, append file, delete file and close file.

PHP Open File - fopen()

The PHP fopen() function is used to open a file.

Syntax

```
resource fopen ( string $filename , string $mode [, bool $use_include_path = false [, resource $context ]] )
```

Example

```
<?php
$handle = fopen("c:\\folder\\file.txt", "r");
?>
```

PHP Close File - fclose()

The PHP fclose() function is used to close an open file pointer.

Syntax

```
bool fclose ( resource $handle )
```

Example

```
<?php
fclose($handle);
?>
```

PHP Read File - fread()

The PHP fread() function is used to read the content of the file. It accepts two arguments: resource and file size.

Syntax

```
string fread ( resource $handle , int $length )
```

Example

```
<?php
$filename = "c:\\myfile.txt";
$handle = fopen($filename, "r");//open file in read mode

```

```
$contents = fread($handle, filesize($filename));//read file

echo $contents;//printing data of file
fclose($handle);//close file
?>
```

Output

```
hello php file
```

PHP Write File - fwrite()

The PHP fwrite() function is used to write content of the string into file.

Syntax

```
int fwrite ( resource $handle , string $string [, int $length ] )
```

Example

```
<?php
$fp = fopen('data.txt', 'w');//open file in write mode
fwrite($fp, 'hello ');
fwrite($fp, 'php file');
fclose($fp);

echo "File written successfully";
?>
```

Output

```
File written successfully
```

PHP Delete File - unlink()

The PHP unlink() function is used to delete file.

Syntax

```
bool unlink ( string $filename [, resource $context ] )
```

Example

```
<?php
unlink('data.txt');

echo "File deleted successfully";
?>
```

PHP Open File

PHP fopen() function is used to open file or URL and returns resource. The fopen() function accepts two arguments: $filename and $mode. The $filename represents the file to be opended and $mode represents the file mode for example read-only, read-write, write-only etc.

Syntax

```
resource fopen ( string $filename , string $mode [, bool $use_include_path = false [, resour
ce $context ]] )
```

PHP Open File Example

```
<?php
$handle = fopen("c:\\folder\\file.txt", "r");
?>
```

PHP Read File

PHP provides various functions to read data from file. There are different functions that allow you to read all file data, read data line by line and read data character by character.

The available PHP file read functions are given below.

- fread()
- fgets()
- fgetc()

PHP Read File - fread()

The PHP fread() function is used to read data of the file. It requires two arguments: file resource and file size.

```
string fread (resource $handle , int $length )
```

$handle represents file pointer that is created by fopen() function.

$length represents length of byte to be read.

Example

```
<?php
$filename = "c:\\file1.txt";
$fp = fopen($filename, "r");//open file in read mode

$contents = fread($fp, filesize($filename));//read file

echo "<pre>$contents</pre>";//printing data of file
```

```
fclose($fp);//close file
?>
```

Output

```
this is first  line
this is another line
this is third line
```

PHP Read File - fgets()

The PHP fgets() function is used to read single line from the file.

Syntax

```
string fgets ( resource $handle [, int $length ] )
```

Example

```
<?php
$fp = fopen("c:\\file1.txt", "r");//open file in read mode
echo fgets($fp);
fclose($fp);
?>
```

Output

```
this is first  line
```

PHP Read File - fgetc()

The PHP fgetc() function is used to read single character from the file. To get all data using fgetc() function, use !feof() function inside the while loop.

Syntax

```
string fgetc ( resource $handle )
```

Example

```
<?php
$fp = fopen("c:\\file1.txt", "r");//open file in read mode
while(!feof($fp)) {
echo fgetc($fp);
}
fclose($fp);
?>
```

Output

```
this is first  line  this  is  another  line  this  is  third  line
```

PHP Write File

PHP fwrite() and fputs() functions are used to write data into file. To write data into file, you need to use w, r+, w+, x, x+, c or c+ mode.

PHP Write File - fwrite()

The PHP fwrite() function is used to write content of the string into file.

Syntax

```
int fwrite ( resource $handle , string $string [, int $length ] )
```

Example

```
<?php
$fp = fopen('data.txt', 'w');//opens file in write-only mode
fwrite($fp, 'welcome ');
fwrite($fp, 'to php file write');
fclose($fp);

echo "File written successfully";
?>
```

Output: data.txt

```
welcome to php  file  write
```

PHP Overwriting File

If you run the above code again, it will erase the previous data of the file and writes the new data. Let's see the code that writes only new data into data.txt file.

```
<?php
$fp = fopen('data.txt', 'w');//opens file in write-only mode
fwrite($fp, 'hello');
fclose($fp);

echo "File written successfully";
?>
```

Output: data.txt

```
hello
```

PHP Append to File

You can append data into file by using a or a+ mode in fopen() function. Let's see a simple example that appends data into data.txt file.

Let's see the data of file first.

data.txt

welcome to php file write

PHP Append to File - fwrite()

The PHP fwrite() function is used to write and append data into file.

Example

```
<?php
$fp = fopen('data.txt', 'a');//opens file in append mode
fwrite($fp, ' this is additional text ');
fwrite($fp, 'appending data');
fclose($fp);

echo "File appended successfully";
?>
```

Output: data.txt

welcome to php file write this is additional text appending data

PHP Delete File

In PHP, we can delete any file using unlink() function. The unlink() function accepts one argument only: file name. It is similar to UNIX C unlink() function.

PHP unlink() generates E_WARNING level error if file is not deleted. It returns TRUE if file is deleted successfully otherwise FALSE.

Syntax

```
bool unlink ( string $filename [, resource $context ] )
```

$filename represents the name of the file to be deleted.

PHP Delete File Example

```
<?php
$status=unlink('data.txt');
if($status){
echo "File deleted successfully";
}else{
echo "Sorry!";
}
?>
```

Output

```
File deleted successfully
```

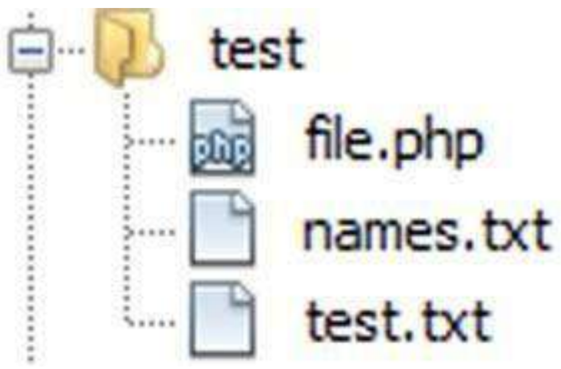
PHP: LIST ALL FILES IN A DIRECTORY

This section deals with the process of how to list all files in a directory using PHP. We will do this using PHP's glob function, which allows us to retrieve a list of file pathnames that match a certain pattern.

For this example, I have created a folder called "test". Inside the folder, I have created three files:

- test.txt
- names.txt
- file.php

Here is a screenshot of the directory:



In our first PHP code snippet, we will simply list everything that is in the test folder:

```php
<?php

//Get a list of file paths using the glob function.
$fileList = glob('test/*');

//Loop through the array that glob returned.
foreach($fileList as $filename){
   //Simply print them out onto the screen.
   echo $filename, '<br>';
}
```

The result will look something like this:

```
test/file.php
test/names.txt
test/test.txt
```

However, what if we wanted to list all files with a particular file extension? i.e. What if we only wantto list the .txt files and not the **.php** file that is currently present?

Well, the solution is pretty simple:

```
//Get a list of all files ending in .txt
$fileList = glob('test/*.txt');
```

In the code snippet above, we told the glob function to return a list of file pathnames that ended .txt

Warning: In some cases, the folder may have subdirectories. In cases where you are listing everything that is inside a specified folder, these subdirectories will be returned by the glob function. To avoid printing out or interacting with subdirectories, you can simply use the is_file function to confirm that the file pathname in question leads to an actual file:

```
<?php
$fileList = glob('test/*');
foreach($fileList as $filename){
    //Use the is_file function to make sure that it is not a directory.
    if(is_file($filename)){
    echo $filename, '<br>';
    }
}
```

Course Contents are Compiled from the following Resources:

1. https://www.tutorialspoint.com/php/index.htm
2. https://www.geeksforgeeks.org/php-scandir-function/
3. https://www.phptpoint.com/php-tutorial/
4. https://www.w3schools.com/php/php_forms.asp
5. https://www.w3schools.com/php/default.asp
6. https://www.tutorialspoint.com/php/php_and_mysql.htm
7. https://www.w3schools.com/php/php_form_complete.asp
8. https://www.w3schools.com/php/php_ref_mysqli.asp

Introduction to HTML

HTML is the standard markup language for creating Web pages.

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of Web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

Example: Simple HTML Document

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

Example Explained

- The <!DOCTYPE html> declaration defines this document to be HTML5
- The <html> element is the root element of an HTML page
- The <head> element contains meta information about the document
- The <title> element specifies a title for the document
- The <body> element contains the visible page content
- The <h1> element defines a large heading
- The <p> element defines a paragraph

HTML Tags

HTML tags are element names surrounded by angle brackets:
<tagname>content goes here...</tagname>

- HTML tags normally come **in pairs** like <p> and </p>
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

Tip: The start tag is also called the **opening tag**, and the end tag the **closing tag**.

HTML Page Structure

Below is a visualization of an HTML page structure:

```
<html>
	<head>
		<title>Page title</title>
	</head>
	<body>
		<h1>This is a heading</h1>
		<p>This is a paragraph.</p>
		<p>This is another paragraph.</p>
	</body>
</html>
```

HTML Headings

HTML headings are defined with the **<h1>** to **<h6>** tags.

<h1> defines the most important heading. <h6> defines the least important heading:

Example

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```

HTML Paragraphs

HTML paragraphs are defined with the **<p>** tag:

Example

```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

HTML Links

HTML links are defined with the **<a>** tag:

Example

```
<a href="https://www.w3schools.com">This is a link</a>
```

HTML Images

HTML images are defined with the **<img>** tag.

The source file (src), alternative text (alt), width, and height are provided as attributes:

Example

```
<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">
```

HTML Elements

An HTML element usually consists of a **start** tag and **end** tag, with the content inserted in between:

```
<tagname>Content goes here...</tagname>
```

The HTML **element** is everything from the start tag to the end tag:

HTML Attributes

- All HTML elements can have **attributes**
- Attributes provide **additional information** about an element
- Attributes are always specified in **the start tag**
- Attributes usually come in name/value pairs like: **name="value"**

The **title** Attribute

Here, a **title** attribute is added to the **<p>** element. The value of the title attribute will be displayed as a tooltip when you mouse over the paragraph:

Example

```
<p title="I'm a tooltip">
This is a paragraph.
</p>
```

The **href** Attribute

HTML links are defined with the **<a>** tag. The link address is specified in the **href** attribute:

Example

```
<a href="https://www.w3schools.com">This is a link</a>
```

HTML Horizontal Rules

The <hr> tag defines a thematic break in an HTML page, and is most often displayed as a horizontal rule.

The <hr> element is used to separate content (or define a change) in an HTML page:

Example

```
<h1>This is heading 1</h1>
<p>This is some text.</p>
<hr>
<h2>This is heading 2</h2>
<p>This is some other text.</p>
<hr>
```

The HTML <pre> Element

The HTML <pre> element defines preformatted text.

The text inside a <pre> element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks:

Example

```
<pre>
  My Bonnie lies over the ocean.

  My Bonnie lies over the sea.

  My Bonnie lies over the ocean.

  Oh, bring back my Bonnie to me.
</pre>
```

The HTML Style Attribute

Setting the style of an HTML element, can be done with the **style attribute**.
The HTML style attribute has the following **syntax**:
<tagname style="*property*:*value;*">
The ***property*** is a CSS property. The ***value*** is a CSS value.

```
<!DOCTYPE html>
<html>
<body>
<p>I am normal</p>
<p style="color:red;">I am red</p>
<p style="color:blue;">I am blue</p>
<p style="font-size:36px;">I am big</p>
</body>
</html>
```

HTML Formatting Elements

HTML also defines special **elements** for defining text with a special **meaning**.

HTML uses elements like <b> and <i> for formatting output, like **bold** or *italic* text.

Formatting elements were designed to display special types of text:

- <b> - Bold text
- <strong> - Important text
- <i> - Italic text
- <em> - Emphasized text
- <mark> - Marked text
- <small> - Small text
- <del> - Deleted text
- <ins> - Inserted text
- <sub> - Subscript text
- <sup> - Superscript text

HTML Comment Tags

- You can add comments to your HTML source by using the following syntax:
- <!-- Write your comments here -->

- Notice that there is an exclamation point (!) in the opening tag, but not in the closing tag.
- **Note:** Comments are not displayed by the browser, but they can help document your HTML source code.

HTML Links - Hyperlinks

HTML links are hyperlinks.

You can click on a link and jump to another document.

When you move the mouse over a link, the mouse arrow will turn into a little hand.

Note: A link does not have to be text. It can be an image or any other HTML element.

HTML Links - Syntax

In HTML, links are defined with the **<a>** tag:

<a href="*url*">*link text*</a>

Example

```
<a href="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

Local Links

The example above used an absolute URL (A full web address).

A local link (link to the same web site) is specified with a relative URL (without http://www ...).

Example

```
<a href="html_images.asp">HTML Images</a>
```

HTML Images Syntax

In HTML, images are defined with the **<img>** tag.

The <img> tag is empty, it contains attributes only, and does not have a closing tag.

The src attribute specifies the URL (web address) of the image:

<img src="*url*" alt="*some_text*" style="width:*width*;height:*height*;">

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Spectacular Mountain</h2>
<img src="pic_mountain.jpg" alt="Mountain View" style="width:304px;height:228px;">
```

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

Example

```
<img src="programming.gif" alt="Computer Man" style="width:48px;height:48px;">
```

Using an Image as a Link

To use an image as a link, simply nest the <img> tag inside the <a> tag:

Example

```
<a href="default.asp">
  <img src="smiley.gif" alt="HTML tutorial" style="width:42px;height:42px;border:0;">
</a>
```

Defining an HTML Table

An HTML table is defined with the **<table>** tag.

Each table row is defined with the **<tr>** tag. A table header is defined with the **<th>** tag. By default, table headings are bold and centered. A table data/cell is defined with the **<td>** tag.

Example

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Note: The <td> elements are the data containers of the table.
They can contain all sorts of HTML elements; text, images, lists, other tables, etc.

Unordered HTML List

An unordered list starts with the **<ul>** tag. Each list item starts with the **<li>** tag.

The list items will be marked with bullets (small black circles) by default:

Example

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - Square

```
<ul style="list-style-type:square">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Ordered HTML List

An ordered list starts with the **<ol>** tag. Each list item starts with the **<li>** tag.

The list items will be marked with numbers by default:

Example

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Ordered HTML List - The Type Attribute

The **type** attribute of the <ol> tag, defines the type of the list item marker:

Type	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers
type="i"	The list items will be numbered with lowercase roman numbers

HTML Forms

The <form> Element

The HTML **<form>** element defines a form that is used to collect user input:

```
<form>
.
form elements
.
</form>
```

An HTML form contains **form elements**.

Form elements are different types of input elements, like text fields, checkboxes, radio buttons, submit buttons, and more.

The <input> Element

The **<input>** element is the most important form element.

The <input> element can be displayed in several ways, depending on the **type** attribute.

Here are some examples:

Type	Description
<input type="text">	Defines a one-line text input field
<input type="radio">	Defines a radio button (for selecting one of many choices)
<input type="submit">	Defines a submit button (for submitting the form)

Text Input

<input type="text"> defines a one-line input field for **text input**:

Example

```
<form>
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

Radio Button Input

<input type="radio"> defines a **radio button**.

Radio buttons let a user select ONE of a limited number of choices:

Example

```
<form>
  <input type="radio" name="gender" value="male" checked> Male<br>
  <input type="radio" name="gender" value="female"> Female<br>
  <input type="radio" name="gender" value="other"> Other
</form>
```

The Submit Button

<input type="submit"> defines a button for **submitting** the form data to a **form-handler**.

The form-handler is typically a server page with a script for processing input data.

The form-handler is specified in the form's **action** attribute:

Example

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
</form>
```

The Action Attribute

The **action** attribute defines the action to be performed when the form is submitted.

Normally, the form data is sent to a web page on the server when the user clicks on the submit button.

In the example above, the form data is sent to a page on the server called "/action_page.php". This page contains a server-side script that handles the form data:

<form **action="/action_page.php**">

If the action attribute is omitted, the action is set to the current page.

The Method Attribute

The **method** attribute specifies the HTTP method (**GET** or **POST**) to be used when submitting the form data:

<form action="/action_page.php" **method="get"**>

or:

<form action="/action_page.php" **method="post"**>

When to Use GET?

The default method when submitting form data is GET.

However, when GET is used, the submitted form data will be **visible in the page address field**:

/action_page.php?firstname=Mickey&lastname=Mouse

Note: GET must NOT be used when sending sensitive information! GET is best suited for short, non-sensitive, amounts of data, because it has size limitations too.

When to Use POST?

Always use POST if the form data contains sensitive or personal information. The POST method does not display the submitted form data in the page address field.

POST has no size limitations, and can be used to send large amounts of data.

The Name Attribute

Each input field must have a **name** attribute to be submitted.

If the name attribute is omitted, the data of that input field will not be sent at all.

This example will only submit the "Last name" input field:

Example

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
</form>
```

HTML Form Elements

The <input> Element

The most important form element is the **<input>** element.

The <input> element can be displayed in several ways, depending on the **type** attribute.

Input Type Text

<input type="text"> defines a **one-line text input field**:

Example

```
<form>
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

Input Type Password

<input type="password"> defines a **password field**:

Example

```
<form>
  User name:<br>
  <input type="text" name="username"><br>
  User password:<br>
  <input type="password" name="psw">
</form>
```

Input Type Reset

<input type="reset"> defines a **reset button** that will reset all form values to their default values:

Example

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
  <input type="reset">
</form>
```

The <select> Element

The **<select>** element defines a **drop-down list**:

Example

```
<select name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

he **<option>** elements defines an option that can be selected.

By default, the first item in the drop-down list is selected.

To define a pre-selected option, add the **selected** attribute to the option:

Example

```
<option value="fiat" selected>Fiat</option>
```

The <textarea> Element

The **<textarea>** element defines a multi-line input field (**a text area**):

Example

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

The **rows** attribute specifies the visible number of lines in a text area.

The **cols** attribute specifies the visible width of a text area.

The <button> Element

The **<button>** element defines a clickable **button**:

Example

```
<button type="button" onclick="alert('Hello World!')">Click Me!</button>
```

Input Type Radio

<input type="radio"> defines a **radio button**.

Radio buttons let a user select ONLY ONE of a limited number of choices:

Example

```
<form>
  <input type="radio" name="gender" value="male" checked> Male<br>
  <input type="radio" name="gender" value="female"> Female<br>
  <input type="radio" name="gender" value="other"> Other
</form>
```

Input Type Checkbox

<input type="checkbox"> defines a **checkbox**.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

Example

```
<form>
  <input type="checkbox" name="vehicle1" value="Bike"> I have a bike<br>
  <input type="checkbox" name="vehicle2" value="Car"> I have a car
</form>
```

Input Type Date

The **<input type="date">** is used for input fields that should contain a date.

Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  Birthday:
  <input type="date" name="bday">
</form>
```

Input Type Email

The **<input type="email">** is used for input fields that should contain an e-mail address.

Depending on browser support, the e-mail address can be automatically validated when submitted.

Some smartphones recognize the email type, and adds ".com" to the keyboard to match email input.

Example

```
<form>
  E-mail:
  <input type="email" name="email">
</form>
```

HTML Input Attributes

The value Attribute

The **value** attribute specifies the initial value for an input field:

Example

```
<form action="">
First name:<br>
<input type="text" name="firstname" value="John">
</form>
```

The readonly Attribute

The **readonly** attribute specifies that the input field is read only (cannot be changed):

Example

```
<form action="">
First name:<br>
<input type="text" name="firstname" value="John" readonly>
</form>
```

The disabled Attribute

The **disabled** attribute specifies that the input field is disabled.

A disabled input field is unusable and un-clickable, and its value will not be sent when submitting the form:

Example

```
<form action="">
First name:<br>
<input type="text" name="firstname" value="John" disabled>
</form>
```

The size Attribute

The **size** attribute specifies the size (in characters) for the input field:

Example

```
<form action="">
First name:<br>
<input type="text" name="firstname" value="John" size="40">
</form>
```

The maxlength Attribute

The **maxlength** attribute specifies the maximum allowed length for the input field:

Example

```
<form action="">
First name:<br>
<input type="text" name="firstname" maxlength="10">
</form>
```

HTML <frame> Tag: Example

A simple three-framed page:

```
<frameset cols="25%,50%,25%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
  <frame src="frame_c.htm">
</frameset>
```

Definition and Usage

The <frame> tag is not supported in HTML5.

The <frame> tag defines one particular window (frame) within a <frameset>.

Each <frame> in a <frameset> can have different attributes, such as border, scrolling, the ability to resize, etc.

Example

A simple three-framed page:

```
<frameset cols="25%,50%,25%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
  <frame src="frame_c.htm">
</frameset>
```

Optional Attributes

Attribute	Value	Description
frameborder	0 1	Not supported in HTML5. Specifies whether or not to display a border around a frame
longdesc	URL	Not supported in HTML5. Specifies a page that contains a long description of the content of a frame
marginheight	pixels	Not supported in HTML5. Specifies the top and bottom margins of a frame
marginwidth	pixels	Not supported in HTML5. Specifies the left and right margins of a frame
name	text	Not supported in HTML5. Specifies the name of a frame
noresize	noresize	Not supported in HTML5. Specifies that a frame is not resizable
scrolling	yes no auto	Not supported in HTML5. Specifies whether or not to display scrollbars in a frame
src	URL	Not supported in HTML5. Specifies the URL of the document to show in a frame

Example: 1 (Mixed frameset)

```
<frameset rows="50%,50%">
  <frame src="frame_a.htm">
  <frameset cols="25%,75%">
    <frame src="frame_b.htm">
    <frame src="frame_c.htm">
  </frameset>
</frameset>
```

Example: 2 (Horizontal frameset)

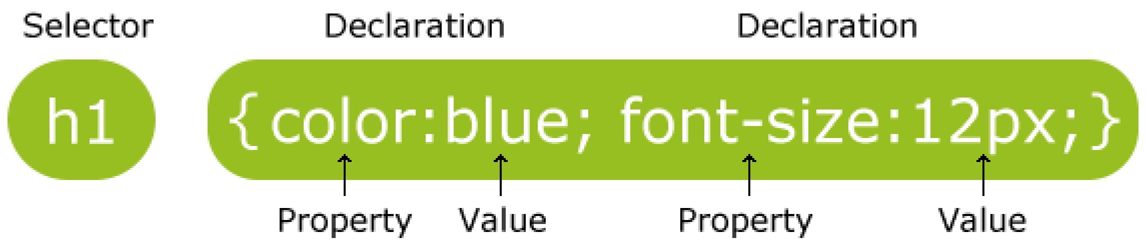
```
<frameset rows="25%,*,25%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
  <frame src="frame_c.htm">
</frameset>
```

INTRODUCTION TO CASCADING STYLE SHEETS

- **CSS** stands for **C**ascading **S**tyle **S**heets
- CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**
- CSS **saves a lot of work**. It can control the layout of multiple web pages all at once
- External stylesheets are stored in **CSS files**

CSS Syntax

A CSS rule-set consists of a selector and a declaration block:



The selector points to the HTML element you want to style.

The declaration block contains one or more declarations separated by semicolons.

Each declaration includes a CSS property name and a value, separated by a colon.

A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces.

In the following example all <p> elements will be center-aligned, with a red text color:

Example

```
p {
   color: red;
   text-align: center;
}
```

The element Selector

The element selector selects elements based on the element name.

You can select all <p> elements on a page like this (in this case, all <p> elements will be center-aligned, with a red text color):

Example

```
p {
   text-align: center;
   color: red;
}
```

The **id** Selector

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element should be unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

The style rule below will be applied to the HTML element with id="para1":

Example

```
#para1 {
    text-align: center;
    color: red;
}
```

The class Selector

The class selector selects elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the name of the class.

In the example below, all HTML elements with class="center" will be red and center-aligned:

Example

```
.center {
    text-align: center;
    color: red;
}
```

Grouping Selectors

If you have elements with the same style definitions, like this:

```
h1 {
    text-align: center;
    color: red;
}

h2 {
    text-align: center;
    color: red;
}

p {
    text-align: center;
    color: red;
}
```

CSS Comments

Comments are used to explain the code, and may help when you edit the source code at a later date.

Comments are ignored by browsers.

A CSS comment starts with /* and ends with */. Comments can also span multiple lines:

Example

```
p {
  color: red;
  /* This is a single-line comment */
  text-align: center;
}

/* This is
a multi-line
comment */
```

CSS How To?

When a browser reads a style sheet, it will format the HTML document according to the information in the style sheet.

Three Ways to Insert CSS

There are three ways of inserting a style sheet:

- External style sheet
- Internal style sheet
- Inline style

External Style Sheet

With an external style sheet, you can change the look of an entire website by changing just one file!

Each page must include a reference to the external style sheet file inside the <link> element. The <link> element goes inside the <head> section:

Example

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

An external style sheet can be written in any text editor. The file should not contain any html tags. The style sheet file must be saved with a .css extension.

Here is how the "mystyle.css" looks:

```
body {
    background-color: lightblue;
}

h1 {
    color: navy;
    margin-left: 20px;
}
```

Internal Style Sheet

An internal style sheet may be used if one single page has a unique style.

Internal styles are defined within the <style> element, inside the <head> section of an HTML page:

Example

```
<head>
<style>
body {
    background-color: linen;
}

h1 {
    color: maroon;
    margin-left: 40px;
}
</style>
</head>
```

Inline Styles

An inline style may be used to apply a unique style for a single element.

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

The example below shows how to change the color and the left margin of a <h1> element:

Example

```
<h1 style="color:blue;margin-left:30px;">This is a heading</h1>
```

CSS Height and Width

The height and width properties are used to set the height and width of an element.

The height and width can be set to auto (this is default. Means that the browser calculates the height and width), or be specified in *length values*, like px, cm, etc., or in percent (%) of the containing block.

This element has a height of 200 pixels and a width of 50%

Example

```
div {
    height: 200px;
    width: 50%;
    background-color: powderblue;
}
```

Text Color

The color property is used to set the color of the text.

With CSS, a color is most often specified by:

- a color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

The default text color for a page is defined in the body selector.

Example

```
body {
    color: blue;
}

h1 {
    color: green;
}
```

Text Alignment

The text-align property is used to set the horizontal alignment of a text.

A text can be left or right aligned, centered, or justified.

CSS Text Properties

Property	Description
color	Sets the color of text
direction	Specifies the text direction/writing direction
letter-spacing	Increases or decreases the space between characters in a text
line-height	Sets the line height
text-align	Specifies the horizontal alignment of text
text-decoration	Specifies the decoration added to text
text-indent	Specifies the indentation of the first line in a text-block

text-shadow	Specifies the shadow effect added to text
text-transform	Controls the capitalization of text
text-overflow	Specifies how overflowed content that is not displayed should be signaled to the user
unicode-bidi	Used together with the direction property to set or return whether the text should be overridden to support multiple languages in the same document
vertical-align	Sets the vertical alignment of an element
white-space	Specifies how white-space inside an element is handled
word-spacing	Increases or decreases the space between words in a text

Font Family

The font family of a text is set with the font-family property.

The font-family property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font, and so on.

Start with the font you want, and end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.

Note: If the name of a font family is more than one word, it must be in quotation marks, like: "Times New Roman".

More than one font family is specified in a comma-separated list:

Example

```
p {
   font-family: "Times New Roman", Times, serif;
}

h1 {
   font-size: 40px;
}
```

Styling Links

Links can be styled with any CSS property (e.g. color, font-family, background, etc.).

Example

```
a {
   color: hotpink;
}
```

In addition, links can be styled differently depending on what **state** they are in.

The four links states are:

- a:link - a normal, unvisited link
- a:visited - a link the user has visited
- a:hover - a link when the user mouses over it
- a:active - a link the moment it is clicked

HTML Lists and CSS List Properties

In HTML, there are two main types of lists:

- unordered lists (<ul>) - the list items are marked with bullets
- ordered lists (<ol>) - the list items are marked with numbers or lettersThe CSS

list properties allow you to:

- Set different list item markers for ordered lists
- Set different list item markers for unordered lists
- Set an image as the list item marker
- Add background colors to lists and list items

Different List Item Markers

The list-style-type property specifies the type of list item marker.

The following example shows some of the available list item markers:

Example

```
ul.a {
   list-style-type: circle;
}
ul.b {
   list-style-type: square;
}
ol.c {
   list-style-type: upper-roman;
}
ol.d {
   list-style-type: lower-alpha;
}
```

An Image as The List Item Marker

The list-style-image property specifies an image as the list item marker:

Example

```
ul {
   list-style-image: url('sqpurple.gif');
}
```

HTML

Introduction to Internet:- A global computer network providing a variety of information and communication facilities, consisting of interconnected networks using standardized communication protocols. "the guide is also available on the Internet"

The Internet is the global system of interconnected computer networks that use the Internetprotocol suite (TCP/IP) to link devices worldwide. It is a network of networks that consists of private, public, academic, business, and government networks of local to global scope, linked by a broad array of electronic, wireless, and optical networking technologies. The Internet carries a vast range of information resources and services.

History of Internet

This marvelous tool has quite a history that holds its roots in the cold war scenario. A need was realized to connect the top universities of the United States so that they can share all the research data without having too much of a time lag. This attempt was a result of Advanced Research Projects Agency (ARPA) which was formed at the end of 1950s just after the Russians had climbed the space era with the launch of Sputnik. After the ARPA got success in 1969, it didn‘t take the experts long to understand that how much potential can this interconnection tool have. In 1971 Ray Tomlinson made a system to send electronic mail. This was a big step in the making as this opened gateways for remote computer accessing i.e.telnet.

During all this time, rigorous paper work was being done in all the elite research institutions. From giving every computer an address to setting out the rules, everything was getting penned down. 1973 saw the preparations for the vital TCP/IP and Ethernet services. At the end of 1970s, Usenet groups had surfaced up. By the time the 80s had started, IBM came up with its PC based on Intel 8088 processor which was widely used by students and universities for it solved the purpose of easy computing. By 1982, the Defense Agencies made the TCP/IP compulsory and the term ―internet‖ was coined. The domain name services arrived in the year 1984 which is also the time around which various internet based marked their debut. A worm, or a rust the computers, attacked in 1988 and disabled over 10% of the computer systems all over the world. While most of the researchers regarded it as an opportunity to enhance computing as it was still in its juvenile phase, quite a number of computer companies became interested in dissecting the cores of the malware which resulted to the formation Computer Emergency Rescue Team (CERT). Soon after the world got over with the computer worm, World Wide Web came into existence. Discovered by Tim Berners-Lee, World Wide Web was seen as a service to connect documents in websites usinghyperlinks.

World Wide Web

The World Wide Web (abbreviated WWW or the Web) is an information space where documents and other web resources are identified by Uniform Resource Locators (URLs), interlinked by hypertext links, and can be accessed via the Internet. English scientist TimBerners-Lee invented the World Wide Web in 1989. He wrote the first web browser computerprogram in 1990 while employed at CERN in Switzerland. The Web browser was released outside CERN in 1991, first to other research institutions starting in January 1991 and to the general public on the Internet in August 1991.

The World Wide Web has been central to the development of the Information Age and is the primary tool billions of people use to interact on the Internet. Web pages are primarily text documents formatted and annotated with Hypertext Markup Language (HTML). In addition to formatted text, web pages may contain images, video, audio, and software components that are rendered in the user's web browser as coherent pages of multimedia content.

Embedded hyperlinks permit users to navigate between web pages. Multiple web pages with a common theme, a common domain name, or both, make up a website. Website content can largely be provided by the publisher, or interactively where users contribute content or the content depends upon the users or their actions. Websites may be mostly informative, primarily for entertainment, or largely for commercial, governmental, or non-governmental organizational purposes

WWW is another example of client/server computing. Each time a link is followed, the client is requesting a document (or graphic or sound file) from a server (also called a Web server) that's part of the World Wide Web that "serves" up the document. The server uses a protocol called HTTP or Hyper Text Transport Protocol. The standard for creating hypertext documents for the WWW is Hyper Text Markup Language or HTML. HTML essentially codes plain text documents so they can be viewed on the Web.

Browsers:

WWW Clients, or "Browser": The program you use to access the WWW is known as a browser because it "browses" the WWW and requests these hypertext documents. Browsers can be graphical, allows to see and hear the graphics and audio;

text-only browsers (i.e., those with no sound or graphics capability) are also available. All of these programs understand http and other Internet protocols such as FTP, gopher, mail, and news, making the WWW a kind of "one stop shopping" for Internetusers.

Year	List of Web browsers
1991	World Wide Web (Nexus)
1992	Viola WWW, Erwise, MidasWWW, MacWWW (Samba)
1993	Mosaic,Cello,[2] Lynx 2.0, Arena, AMosaic 1.0
1994	IBM WebExplorer, Netscape Navigator, SlipKnot 1.0, MacWeb, IBrowse, Agora (Argo), Minuet
1995	Internet Explorer 1, Internet Explorer 2, Netscape Navigator 2.0, OmniWeb, UdiWWW, Grail
1996	Arachne 1.0, Internet Explorer 3.0, Netscape Navigator 3.0,Opera 2.0, PowerBrowser 1.5,[4] Cyberdog,Amaya 0.9,[5] AWeb,Voyager
1997	Internet Explorer 4.0, Netscape Navigator 4.0, Netscape Communicator 4.0, Opera3.0,[6] Amaya 1.0[5]
1998	iCab, Mozilla
1999	Amaya 2.0,[5] Mozilla M3, Internet Explorer 5.0
2000	Konqueror,Netscape 6, Opera 4,[7] Opera 5,[8] K-Meleon 0.2, Amaya 3.0,[5] Amaya 4.0[5]
2001	Internet Explorer 6, Galeon 1.0, Opera 6,[9] Amaya 5.0[5]
2002	Netscape 7, Mozilla 1.0, Phoenix 0.1, Links 2.0, Amaya 6.0,[5] Amaya 7.0[5]
2003	Opera 7,[10] Apple Safari 1.0, Epiphany 1.0, Amaya 8.0[5]
2004	Firefox 1.0, Netscape Browser, OmniWeb 5.0
2005	Opera8,[11]Apple Safari2.0, Netscape Browser 8.0, Epiphany 1.8, Amaya 9.0,[5] AOL Explorer 1.0, Maxthon 1.0,Shiira 1.0
2006	Mozilla Firefox 2.0, Internet Explorer 7,Opera 9,[12], SeaMonkey 1.0, K-Meleon 1.0, Galeon 2.0, Camino 1.0, Avant11, iCab 3
2007	Apple Safari 3.0, Maxthon 2.0, Netscape Navigator9,NetSurf 1.0, Flock 1.0, Conkeror
2008	Google Chrome 1, Mozilla Firefox 3, Opera 9.5,[13], Apple Safari 3.1, Konqueror 4, Amaya 10.0,[5] Flock 2, Amaya 11.0[5]
2009	Google Chrome 2–3, Mozilla Firefox 3.5, Internet Explorer 8,Opera 10,[14], Apple Safari 4, SeaMonkey 2, Camino 2,surf, Pale Moon 3.0[15]
2010	Google Chrome 4–8, Mozilla Firefox 3.6, Opera 10.50,[16], Opera 11, Apple Safari 5, K-Meleon 1.5.4,
2011	Google Chrome 9–16, Mozilla Firefox 4-9, Internet Explorer 9,Opera 11.50, Apple Safari 5.1, Maxthon 3.0, SeaMonkey 2.1–2.6
2012	Google Chrome 17–23, Mozilla Firefox 10–17, Internet Explorer 10, Opera 12, Apple Safari 6, Maxthon 4.0, SeaMonkey 2.7-2.14
2013	Google Chrome24–31,Mozilla Firefox 18–26,Internet Explorer 11, Opera 15–18, Apple Safari 7, SeaMonkey 2.15-2.23

2014	Google Chrome 32–39, Mozilla Firefox 27–34, Opera 19–26, Apple Safari 8
2015	Google Chrome 40–47, Microsoft Edge,Mozilla Firefox 35–43, Opera 27–34, Vivaldi
2016	Google Chrome 48–55,Mozilla Firefox 44–50,Microsoft Edge 14, Opera35–42, Apple Safari 10, SeaMonkey 2.24–2.30, Pale Moon 26.0.0[17], Pale Moon 27.0.0[18]
2017	Google Chrome56–60,Microsoft Edge 15,Mozilla Firefox 51–55.0.2, Opera43–45, Opera Neon

Uniform Resource Locators, or URLs: A Uniform Resource Locator, or URL is the address of a document found on the WWW. Browser interprets the information in the URL in order to connect to the proper Internet server and to retrieve your desired document. Each time a click on a hyperlink in a WWW document instructs browser to find the URL that's embedded within the hyperlink.

The elements in a URL: **Protocol://server's address/filename**

Hypertext protocol: http://www.aucegypt.eduFile Transfer Protocol: ftp://ftp.dartmouth.eduTelnet Protocol: telnet://pac.carl.org

News Protocol: news:alt.rock-n-roll.stones

What are Domains? Domains divide World Wide Web sites into categories based on the nature of their owner, and they form part of a site's address, or uniform resource locator (URL). Common top-level domainsare:

.com—commercial enterprises	.mil—military site
org—organization site (non-profits, etc.)	int—organizations established by international treaty
.net—network	.biz—commercial and personal
.edu—educational site (universities, schools, etc.)	.info—commercial and personal
.gov—government organizations	.name—personal sites

Additional three-letter, four-letter, and longer top-level domains are frequently added. Each country linked to the Web has a two-letter top-level domain, for example .fr is France, .ie is Ireland.

MIME (Multi-Purpose Internet Mail Extensions):- MIME is an extension of the original Internet e-mail protocol that lets people use the protocol to exchange different kinds of data files on the Internet: audio, video, images, application programs, and other kinds, as well as the ASCII text handled in the original protocol, the Simple Mail Transport Protocol (SMTP). In 1991, Nathan Borenstein of Bellcore proposed to the IETF that SMTP be extended so that Internet (but

New MIME data types are registered with the Internet Assigned Numbers Authority (IANA).

MIME is specified in detail in Internet Request for Comments 1521 and 1522, which amend the original mail protocol specification, RFC 821 (the Simple Mail Transport Protocol) and the ASCII messaging header, RFC 822.

Hypertext Transport Protocol:

HTTP means HyperText Transfer Protocol. HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.
For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page. The other main standard that controls how the World Wide Web works is HTML, which covers how Web pages are formatted anddisplayed.

HTTP is called a stateless protocol because each command is executed independently, without any knowledge of the commands that came before it. This is the main reason that it is difficult to implement Web sites that react intelligently to user input.

HTTPS: A similar abbreviation, HTTPS means Hyper Text Transfer Protocol Secure. Basically, it is the secure version of HTTP. Communications between the browser and website are encrypted by Transport Layer Security (TLS), or its predecessor, Secure Sockets Layer (SSL).

The Web Programmer‟sToolbox:

- **HTML** - a *markup*language
  - To describe the general form and layout ofdocuments
    - HTML is **not** a programming language - it cannot beused describe **computations**.
  - An HTML document is a mix of **content** and **controls**
    - Controls are **tags** and their**attributes**
      - Tags often delimit content and specify something about howthe content should be arranged in thedocument
        For example, <p>Write a paragraph here </p> is an *element*.
      - Attributes provide additional information about the content of a tag
        For example, <img src = "redhead.jpg"/><font color ="Red"/>
- Plugins
  - Integrated into tools like word processors, effectively converting themto WYSIWYG HTMLeditors
- Filters
  - Convert documents in other formats toHTML
- Advantages of both filters and plug-ins:
  - Existing documents produced with other tools can be converted to HTML documents
  - Use a tool you already know to produceHTML
- Disadvantages of both filters andplug-ins:

  - HTML output of both is not perfect - must be finetuned
  - HTML may benon-standard
  - You have two versions of the document, which are difficult tosynchronize
- XML
  - A meta-markup language (a language for defining markuplanguage)
  - Used to create a new markup language for a particular purpose orarea
  - Because the tags are designed for a specific area, they can bemeaningful
- JavaScript
  - A client-side HTML-embedded scriptinglanguage
  - Provides a way to access elements of HTML documents and dynamicallychange them
- Flash
  - A system for building and displaying text, graphics, sound, interactivity,and animation(movies)
  - Twoparts:
    1. Authoringenvironment
    2. Player

Supports both motion and shape animation

PHP

A server-side scripting language

Great for form processing and database access through the Web

Ajax

Asynchronous JavaScript + XML

- No new technologies orlanguages

Much faster for Web applications that have extensive user/server interactions

Uses asynchronous requests to the server

Requests and receives small parts of documents, resulting in much faster responses

Java Web Software

Servlets – server-side Java classes

JavaServer Pages (JSP) – a Java-based approach to server-side scripting

JavaServer Faces – adds an event-driven interface model on JSP

ASP.NET

Does what JSP and JSF do, but in the .NET environment

Allows.NET languages to be used as server-side scripting language

Ruby

A pure object-oriented interpreted scripting language

Every data value is an object, and all operations are via method calls

Both classes and objects are dynamic

Rails

A development framework for Web-based applications
Particularly useful for Web applications that access databases
Written in Ruby and uses Ruby as its primary user language

HTML Common tags:-

HTML is the building block for web pages. HTML is a format that tells a computer how to display a web page. The documents themselves are plain text files with special "tags" or codes that a web browser uses to interpret and display information on your computer screen.

- HTML stands for Hyper Text MarkupLanguage
- An HTML file is a text file containing small markuptags
- The markup tags tell the Web browser how to display thepage
- An HTML file must have an htm or html fileextension.

HTML Tags:- HTML tags are used to mark-up HTML elements .HTML tags are surrounded by the two characters < and >. The surrounding characters are called angle brackets. HTML tags normally come in pairs like **and** The first tag in a pair is the start tag, the second tag is the end tag . The text between the start and end tags is the element content . HTML tags are not case sensitive, <B>**means the same as<b>.**

The most important tags in HTML are tags that define headings, paragraphs and line breaks.

Tag	**Description**
<!DOCTYPE...>	This tag defines the document type and HTML version.
<html>	This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.
<head>	This tag represents the document's header which can keep other HTML tags like <title>, <link> etc.
<title>	The <title> tag is used inside the <head> tag to mention the document title.
<body>	This tag represents the document's body which keeps other HTML tags like <h1>, <div>, <p> etc.
<p>	This tag represents a paragraph.
<h1> to <h6>	Defines header 1 to header 6
<br>	Inserts a single line break
<hr>	Defines a horizontal rule
<!-->	Defines a comment

Headings:-

Headings are defined with the <h1> to <h6> tags. <h1> defines the largest heading while <h6> defines the smallest.

```
<h1>This is a heading</h1>
<h2>This is aheading</h2>
<h3>This is aheading</h3>
<h4>This is aheading</h4>
<h5>This is aheading</h5>
<h6>This is aheading</h6>
```

Paragraphs:-

Paragraphs are defined with the <p> tag. Think of a paragraph as a block of text. You can use the align attribute with a paragraph tag as well.

```
<p align="left">This is a paragraph</p>
<p align="center">this is another paragraph</p>
```

Note: You must indicate paragraphs with <p> elements. A browser ignores any indentations or blank lines in the source text. Without <p> elements, the documentbecomes one large paragraph. HTML automatically adds an extra blank line before and after a paragraph.

Line Breaks:-

The <br> tag is used when you want to start a new line, but don't want to start a new paragraph. The <br> tag forces a line break wherever you place it. It is similar to single spacing in a document.

This Code	output
<p>This <br> is a para<br> graph with line breaks</p>	This<br>is a para<br>graph with line breaks

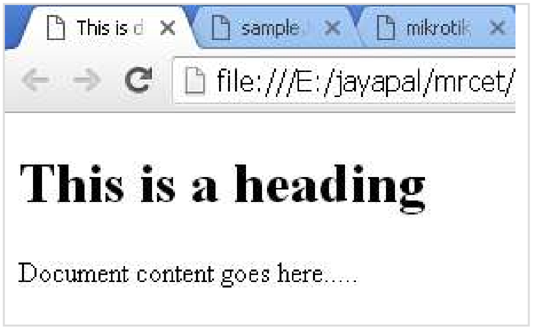
Horizontal Rule The element is used for horizontal rules that act as dividers between sections like this:

The horizontal rule does not have a closing tag. It takes attributes such as align and width

Code	Output
<hr width="50%" align="center">	———

Sample html program

```
<!DOCTYPE html>
<html>
        <head>
                <title>This is document title
                </title>
```



This is a heading

Document content goes here.....

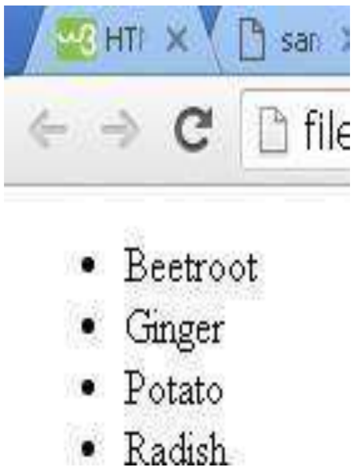
</html>

- Type the above program in notepad and save with some file nameeg:sample.html
- Open the file with browser and the webpage looks likethis

Lists:-HTML offers web authors three ways for specifying lists of information.
All lists must contain one or more list elements. Lists are of three types
1) Un ordered list
2)Ordered List
3)Definitionlist

HTML Unordered Lists: An unordered list is a collection of related items that have no special order or sequence. This list is created by using HTML <ul> tag. Each item in the list is marked with a bullet.

```
<!DOCTYPE html>
<html>
        <head>
                <title>HTML Unordered List</title>
        </head>
        <body>
                 <ul>    <li>Beetroot</li>
                         <li>Ginger</li><li>Potato</li>
                         <li>Radish</li>
                </ul>
        </body>
```

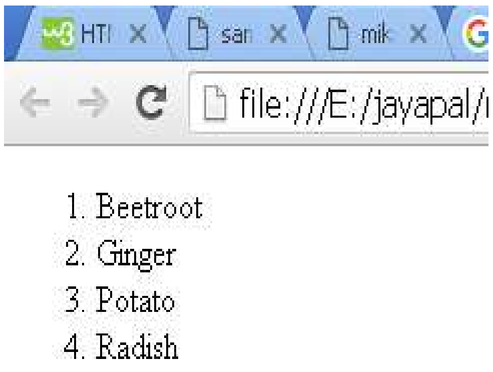


- Beetroot
- Ginger
- Potato
- Radish

Example

HTML Ordered Lists:- items are numbered list instead of bulleted, This list is created by using **<ol>**tag.

```
<!DOCTYPE html>
<html>
        <head>
                <title>HTML Ordered List</title>
        </head>
        <body>
                <ol>
                        <li>Beetroot</li>
                        <li>Ginger</li>
                        <li>Potato</li>
                        <li>Radish</li>
                </ol>
        </body>
</html>
```

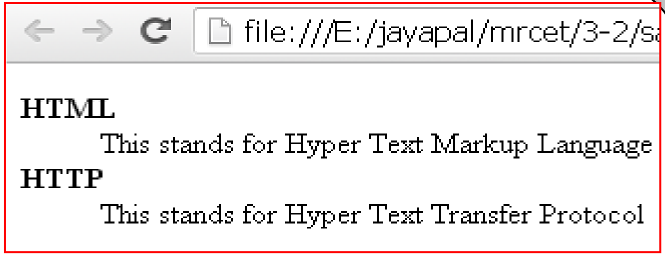


1. Beetroot
2. Ginger
3. Potato
4. Radish

HTML Definition Lists:- HTML and XHTML supports a list style which is called definition lists where entries are listed like in a dictionary or encyclopedia. The definition list is the ideal way to present a glossary, list of terms, or other name/value list. Definition List makes use of following three tags.

1). <dl> - Defines the start of the list
2). <dt> - A term
3). <dd> - Termdefinition
4). </dl> - Defines the end of thelist

```
<!DOCTYPE html>
<html>
    <head>
          <title>HTML Definition List</title>
   </head>
   <body>
    <dl>
              <dt><b>HTML</b></dt><dd>This stands for Hyper Text Markup Language</dd>
              <dt><b>HTTP</b></dt><dd>This stands for Hyper Text Transfer Protocol</dd>
   </dl>
</body>
</html>
```



HTML
This stands for Hyper Text Markup Language
HTTP
This stands for Hyper Text Transfer Protocol

HTML tables:

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells. The HTML tables are created using the **<table>**tag inwhich the **<tr>**tag is used to create table rows and **<td>**tag is used to create data cells.

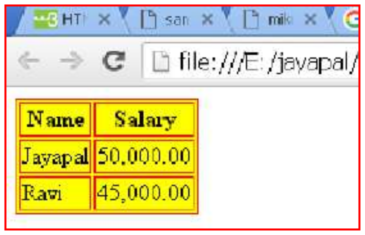
Example:

```
<!DOCTYPE html>
 <html>
 <head>
 <title>HTML Tables</title>
 </head>
 <body>
          <table border="1">
                    <tr>
                              <td>Row 1, Column 1</td><td>Row 1, Column 2</td>
                    </tr>
                    <tr><td>Row 2, Column 1</td><td>Row 2, Column 2</td>
                    </tr>
          </table>
 </body>
</html>
```

Table Heading: Table heading can be defined using **<th>**tag. This tag will be put to replace <td> tag, which is used to represent actual data cell. Normally you will put your top row as table heading as shown below, otherwise you can use <th> element in any row.

Tables Backgrounds: set table background using one of the following two ways:
1)bgcolor attribute - You can set background color for whole table or just for one cell.
2) background attribute - You can set background image for whole table or just for one cell. You can also set border color also using bordercolorattribute.

```
<!DOCTYPE html>
 <html>
 <head>
 <title>HTML Tables</title></head>
 <body>
            <table border="1"bordercolor="red" bgcolor="yellow">
                        <tr><th>Name</th>
                         <th>Salary</th></tr>
                                    <td>Jayapal     </td><td>50,000.00</td>
                        </tr>
                        <tr><td>Ravi</td><td>45,000.00</td>
                        </tr>
            </table>
</body>
</html>
```



Name	Salary
Jayapal	50,000.00
Ravi	45,000.00

Images are very important to beautify as well as to depict many complex concepts in simple way on your web page.

Insert Image:

insert any image in the web page by using **<img>**tag.
<img align="left|right|middle|top|bottom">

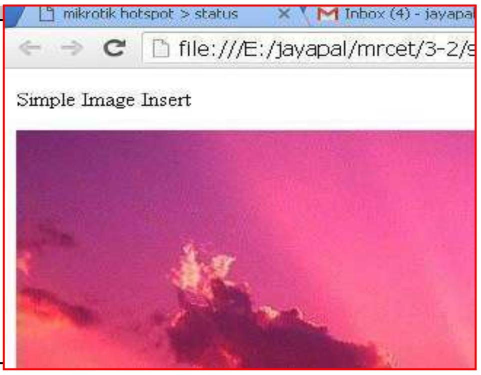
Attribute Values

Value	Description
left	Align the image to the left
right	Align the image to the right
middle	Align the image in the middle

```
<img src="Image URL" ... attributes-
```

top	Align the image at the top
bottom	Align the image at the bottom

```
    <!DOCTYPE html>
<html><head>
                <title>Using Image in Webpage</title>
        </head>
        <body><p>Simple Image Insert</p>
                <img src="test.png" alt="Test Image" />
        </body>
</html>
```



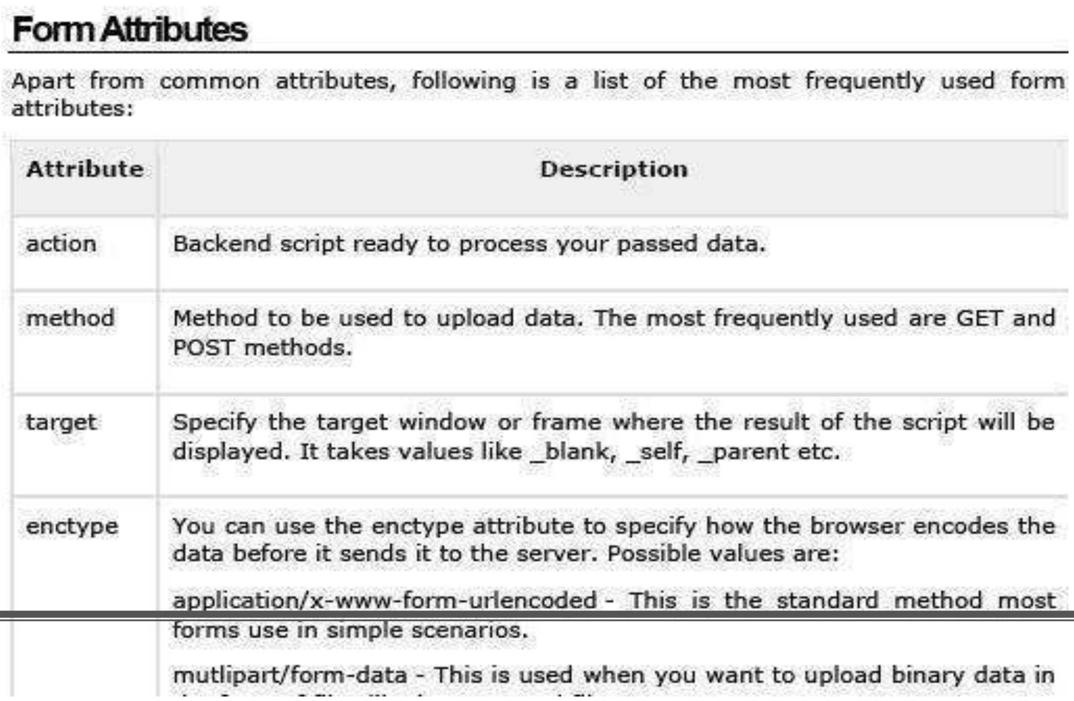
Simple Image Insert

Example

HTML FORMS:

HTML Forms are required to collect some data from the site visitor. For example, during user registration you would like to collect information such as name, email address, credit card, etc. A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application. There are various form elements available like text fields, text area fields, drop-down menus, radio buttons,

```
<form action="Script URL" method="GET|POST"> form elements like input, text area etc. </form>
```



Form Attributes

Apart from common attributes, following is a list of the most frequently used form attributes:

Attribute	Description
action	Backend script ready to process your passed data.
method	Method to be used to upload data. The most frequently used are GET and POST methods.
target	Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.
enctype	You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are: application/x-www-form-urlencoded - This is the standard method most forms use in simple scenarios. mutlipart/form-data - This is used when you want to upload binary data in

There are different types of form controls that you can use to collect data using HTML form:

- Text InputControls
- Checkboxes Controls
- Radio BoxControls
- Select BoxControls
- File Selectboxes
- Hidden Controls
- ClickableButtons
- Submit and ResetButton

Text Input Controls:-

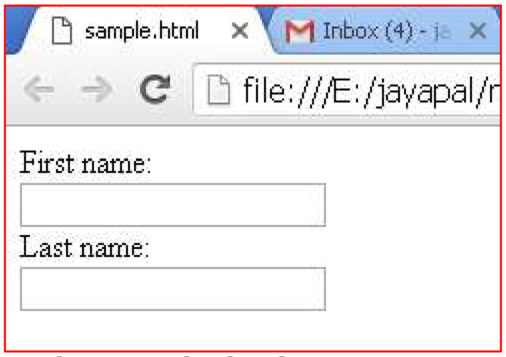
There are three types of text input used on forms:

1) **Single-line text input controls -** This control is used for items that require only one line of user input, such as search boxes or names. They are created usingHTML **<input>**tag.

<input type="text">defines a one-line input field for **text input**:

Example:

```
<form>
 Firstname:<br>
 <input type="text"name="firstname"><br>
 Lastname:<br>
 <input type="text" name="lastname">
</form>
```



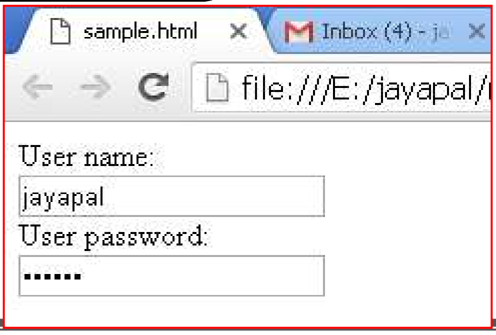
First name:

Last name:

2) **Password input controls -** This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML <input>tag.

<input type="password">defines a **password field**:

```
<form>
  User name:<br>
  <input type="text" name="username"><br>
  User password:<br>
  <input type="password" name="psw">
</form>
```



User name:

jayapal

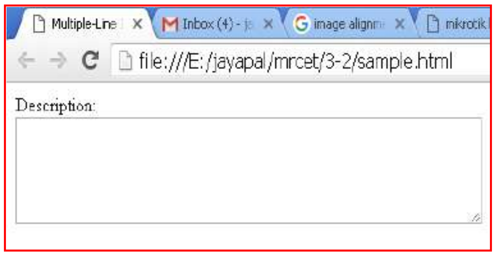
User password:

••••••

Input Type Password

3) Multi-line text input controls - This is used when the user is required to give details that may belongerthanasinglesentence.Multi-lineinputcontrolsarecreatedusing HTML <textarea> tag.

```
<!DOCTYPE html>
<html>
   <head>
                    <title>Multiple-Line Input Control</title>
   </head>
   <body>
     <form> Description: <br />
           <textarea rows="5" cols="50" name="description"> Enter description here... </textarea>
     </form>
   </body>
</html>
```



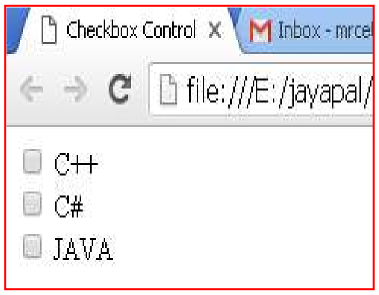
Description:

Checkboxes Controls:-

Checkboxes are used when more than one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to checkbox.

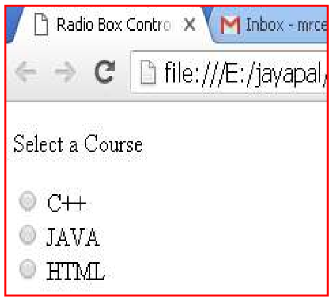
Here is an example HTML code for a form with two checkboxes:

```
<!DOCTYPE html>
<html><head><title>Checkbox Control</title></head>
 <body>
        <form>
                <input type="checkbox" name="C++" value="on"> C++
                <br>
                <input type="checkbox" name="C#" value="on"> C#
                <br>
                <input type="checkbox" name="JAVA" value="on"> JAVA
        </form>
 </body></html>
```



Radio Button Control:-

```
<!DOCTYPE html>
<html><head><title>Radio Box Control</title></head>
        <body><p>Select a Course</p>
                <form>
                        <input type="radio" name="subject" value="C++"> C++
                        <br>
                        <input type="radio" name="subject" value="JAVA"> JAVA
                        <br>
                         <input type="radio" name="subject" value="HTML"> HTML
                </form>     </body></html>
```

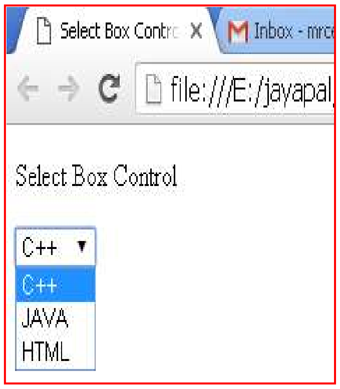


Radio buttons are used when

out of many options, just one option is required to be selected. They are also createdusing HTML <input> tag but type attribute is set toradio.

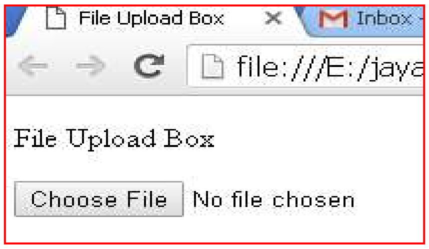
Select Box Controls :- A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or moreoptions.

```
<!DOCTYPE html>
<html>
 <head>
        <title>Select Box Control</title>
  </head>
  <body>
        <form>
                <select name="dropdown">
                        <option value="C++" selected>C++</option>
                        <option value="JAVA">JAVA</option>
                        <option value="HTML">HTML</option>
                </select>
        </form>
  </body>
 </html>
```



File Select boxes:- If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. This is also created using the <input> element but type attribute is set to **file**.

```
<!DOCTYPE html>
<html>
        <head>
                <title>File Upload Box</title>
        </head>
        <body>
                <p>File Upload Box</p>
                <form>
                        <input type="file" name="fileupload" accept="image/*" />
                </form>
        </body>
 </html>
```



Hidden Controls:- Hidden form controls are used to hide data inside the page which later on can be pushed to the server. This control hides inside the code and does not appear on the actual page. For example, following hidden form is being used to keep current page number. When a user will click next page then the value of hidden control will be sent to the web server and there it will decide which page will be displayed next based on the passed currentpage.

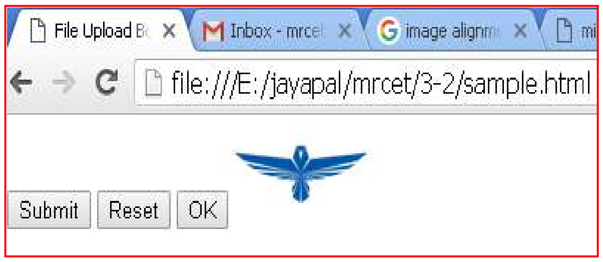
```
   <html><head>             <title>File Upload Box</title>        </head>
        <body>
                <form>
                        <p>This is page 10</p>
                        <input type="hidden" name="pagename" value="10" />
                        <input type="submit" name="submit" value="Submit" />
                        <input type="reset" name="reset" value="Reset" />
                </form>             </body>      </html>
```

Button Controls:-

There are various ways in HTML to create clickable buttons. You can also create a clickable button using <input> tag by setting its type attribute to **button**. The type attribute can take the following values:

Type	Description
submit	This creates a button that automatically submits a form.
reset	This creates a button that automatically resets form controls to their initial values.
button	This creates a button that is used to trigger a client-side script when the user clicks that button.
image	This creates a clickable button but we can use an image as background of the button.

```
<!DOCTYPE html>
 <html>
 <head>
            <title>File Upload Box</title>
 </head>
             <body>
             <form>
                        <input type="submit" name="submit" value="Submit" />
                        <input type="reset" name="reset" value="Reset" />
                        <input type="button" name="ok" value="OK" />
                        <input type="image" name="imagebutton" src="test1.png" />
             </form>
             </body></html>
```



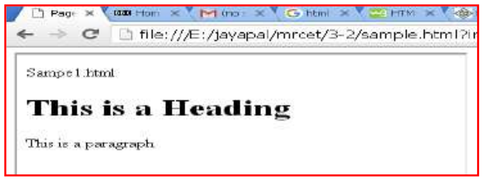
HTML frames: These are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables areorganized: into rows andcolumns.

To use frames on a page we use <frameset> tag instead of <body> tag. The <frameset> tag defines, how to divide the window into frames. The **rows** attribute of <frameset> tag defines horizontal frames and **cols** attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

Note: HTML <**frame**>Tag. Not Supported in HTML5.

```
<frameset cols="25%,50%,25%">
          <framesrc="frame_a.htm">
          <framesrc="frame_b.htm">
          <framesrc="frame_c.htm">
</frameset>
```

```
<!DOCTYPE html>
<html>
        <head>
                <title>Page Title</title>
        </head>
        <body>
        <iframe src="sample1.html" height="400" width="400"frameborder="1">
        <h1>This is aHeading</h1>
        <p>This is aparagraph.</p>
        </iframe>
        </body>
</html>
```



file:///E:/jayapal/mrcet/3-2/sample.html?i

Sampe1.html

This is a Heading

This is a paragraph.

CSS stands for Cascading Style Sheets

CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**.
CSS **saves a lot of work**. It can control the layout of multiple web pages all at once.
CSS can be added to HTML elements in 3 ways:

- **Inline** - by using the style attribute in HTMLelements
- **Internal** - by using a <style> element in the <head>section
- **External** - by using an external CSSfile

Inline CSS

An inline CSS is used to apply a unique style to a single HTML element.
An inline CSS uses the style attribute of an HTML element.
This example sets the text color of the <h1> element to blue:

```
<h1 style="color:blue;">This is a Blue Heading</h1>
```

```
<html>      <head>      <title>Page Title</title></head>
        <body>
                <h1 style="color:blue;">This is a Blue Heading</h1>
        </body>
</html>
```

Internal CSS: An internal CSS is used to define a style for a single HTML page. An internal CSS is defined in the <head> section of an HTML page, within a <style> element:

```
<html>
            <head>
            <style>
            body {background-color: powderblue;}
            h1 {color: blue;}
            p    {color:red;}
            </style>
            </head>
<body>
<h1>This is aheading</h1>
<p>This is aparagraph.</p>
</body>
</html>
```

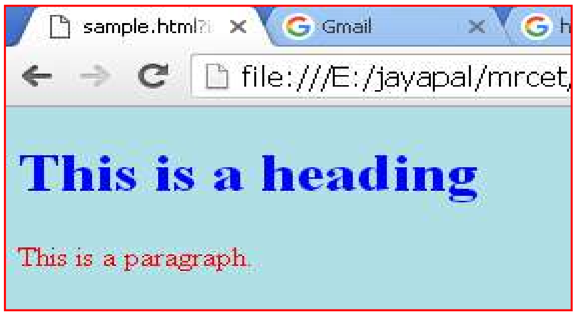
External CSS:-

An external style sheet is used to define the style for many HTML pages. **With an external style sheet, you can change the look of an entire web site, by changing one file!** To use an external style sheet, add a link to it in the <head> section of the HTML page:

```
<html>
            <head>
              <link rel="stylesheet" href="styles.css">
            </head>
            <body>
                        <h1>This is aheading</h1>
                        <p>This is aparagraph.</p>
            </body>
</html>
```

An external style sheet can be written in any text editor. The file must not contain any HTML code, and must be saved with **a .css extension**.
Here is how the "styles.css" looks:

```
body    {     background-color: powderblue;      }

h1        {    color:blue;      }

p          {    color:red;      }
```

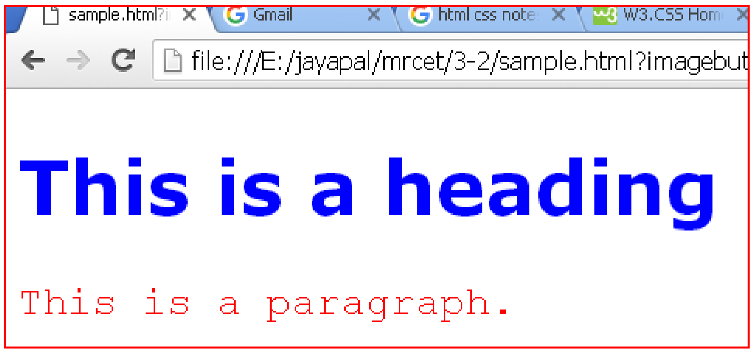


This is a heading

This is a paragraph.

CSS Fonts: The CSS **color** property defines the text color to be used.
The CSS **font-family** property defines the font to
be used.The CSS **font-size** property defines the text

```
<html>
<head>
<style>
h1 {
    color: blue;
    font-family: verdana;
    font-size: 300%;
}
p{
    color: red;
    font-family: courier;
    font-size: 160%;
}
</style>
</head>
<body>
<h1>This is aheading</h1>
<p>This is aparagraph.</p>
</body>
</html>
```



file:///E:/jayapal/mrcet/3-2/sample.html?imagebut

This is a heading

This is a paragraph.

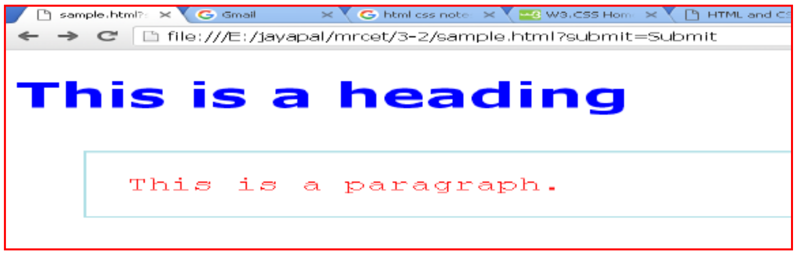
size to be used.

CSS Border: The CSS border property defines a border around an HTML element.

CSS Padding: The CSS padding property defines a padding (space) between the text andtheborder.

CSS Margin: The CSS margin property defines a margin

```
<html><head>
<style>h
1 {
    color: blue;
    font-family: verdana;
    font-size: 300%; }
p {
    color: red; font-size: 160%; border: 2px solid powderblue; padding: 30px; margin: 50px; }
</style>
</head>
<body>
<h1>This is aheading</h1>
<p>This is aparagraph.</p>
</body>
</html>
```



file:///E:/jayapal/mrcet/3-2/sample.html?submit=Submit

This is a heading

This is a paragraph.

UNIT – II

XML: Introduction to XML, Defining XML tags, their attributes and values, Document Type Definition, XML Schemas, Document Object Model, XHTML Parsing XML Data – DOM and SAX in java.

What is XML?

Extensible Markup Language (XML) lets you define and store data in a shareable manner. XML supports information exchange between computer systems such as websites, databases, and third-party applications.

Predefined rules make it easy to transmit data as XML files over any network because the recipient can use those rules to read the data accurately and efficiently.

- **Xml** (eXtensible Markup Language) is a mark up language.
- XML is designed to store and transport data.
- Xml was released in late 90's. it was created to provide an easy to use and store self describing data.
- XML became a W3C Recommendation on February 10, 1998.
- XML is not a replacement for HTML.
- XML is designed to be self-descriptive.
- XML is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is platform independent and language independent.

What is mark-up language?

- A **mark up language** is a modern system for highlight or underline a document.
- Students often underline or highlight a passage to revise easily, same in the sense of modern mark up language highlighting or underlining is replaced by tags.

Why xml?

Platform Independent and Language Independent: The main benefit of xml is that you can use it to take data from a program like Microsoft SQL, convert it into XML then share that XML with other programs and platforms. You can communicate between two platforms which are generally very difficult.

The main thing which makes XML truly powerful is its international acceptance. Many corporation use XML interfaces for databases, programming, office application mobile phones and more. It is due to its platform independent feature.

Features and Advantages of XML

XML is widely used in the era of web development. It is also used to simplify data storage and data sharing.

The main features or advantages of XML are given below.

1) XML separates data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can focus on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.

With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

2) XML simplifies data sharing

In the real world, computer systems and databases contain data in incompatible formats.

XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.

This makes it much easier to create data that can be shared by different applications.

3) XML simplifies data transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

4) XML simplifies Platform change

Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.

XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

5) XML increases data availability

Different applications can access your data, not only in HTML pages, but also from XML data sources.

With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

6) XML can be used to create new internet languages

A lot of new Internet languages are created with XML.

Here are some examples:

- **XHTML**
- **WSDL** for describing available web services
- **WAP** and **WML** as markup languages for handheld devices
- **RSS** languages for news feeds
- **RDF** and **OWL** for describing resources and ontology
- **SMIL** for describing multimedia for the web

Why is XML important?

Extensible Markup Language (XML) is a markup language that provides rules to define any data. Unlike other programming languages, XML cannot perform computing operations by itself. Instead, any programming language or software can be implemented for structured data management.

For example, consider a text document with comments on it. The comments might give suggestions like these:

- Make the title bold
- This sentence is a header
- This word is the author

Such comments improve the document's usability without affecting its content. Similarly, XML uses markup symbols to provide more information about any data. Other software, like browsers and data processing applications, use this information to process structured data more efficiently.

Defining XML tags

You use markup symbols, called tags in XML, to define data. For example, to represent data for a bookstore, you can create tags such as <book>, <title>, and <author>. Your XML document for a single book would have content like this:

```
<book>

<title> Learning Amazon Web Services </title>

<author> Mark Wilkins </author>

</book>
```

Tags bring sophisticated data coding to integrate information flows across different systems.

What are the applications of XML?

Extensible Markup Language (XML) is the underlying technology in thousands of applications, ranging from common productivity tools like word processing to book publishing software and even complex application configuration systems.

Data transfer

You can use XML to transfer data between two systems that store the same data in different formats. For example, your website stores dates in MM/DD/YYYY format, but your accounting system stores dates in DD/MM/YYYY format. You can transfer the data from the website to the accounting system by using XML. Your developers can write code that automatically converts the following:

- Website data to XML format
- XML data to accounting system data
- Accounting system data back to XML format
- XML data back to website data

Web applications

XML gives structure to the data that you see on webpages. Other website technologies, like HTML, work with XML to present consistent and relevant data to website visitors. For example, consider an e-commerce website that sells clothes. Instead of showing all clothes to all visitors, the website uses XML to create customized webpages based on user preferences. It shows products from specific brands by filtering the <brand> tag.

Documentation

You can use XML to specify the structural information of any technical document. Other programs then process the document structure to present it flexibly. For example, there are XML tags for a paragraph, an item in a numbered list, and a heading. Using these tags, other types of software automatically prepare the document for uses such as printing and webpage publication.

Data type

Many programming languages support XML as a data type. With this support, you can easily write programs in other languages that work directly with XML files.

What are the components of an XML file?

An Extensible Markup Language (XML) file is a text-based document that you can save with the .xml extension. You can write XML similar to other text files. To create or edit an XML file, you can use any of the following:

- Text editors like Notepad or Notepad++
- Online XML editors
- Web browsers

Any XML file includes the following components.

XML document

The <xml></xml> tags are used to mark the beginning and end of an XML file. The content within these tags is also called an XML document. It is the first tag that any software will look for to process XML code.

XML declaration

An XML document begins with some information about XML itself. For example, it might mention the XML version that it follows. This opening is called an XML declaration. Here's an example.

```
<?xml version="1.0" encoding="UTF-8"?>
```

XML elements

All the other tags you create within an XML document are called XML elements. XML elements can contain these features:

- Text
- Attributes
- Other elements

All XML documents begin with a primary tag, which is called the root element.

For example, consider the XML file below.

```
<InvitationList>

<family>

    <aunt>

    <name>Christine</name>

     <name>Stephanie</name>

    </aunt>

</family>

</InvitationList>
```

<InvitationList> is the root element; *family* and *aunt* are other element names.

XML attributes

XML elements can have other descriptors called attributes. You can define your own attribute names and write the attribute values within quotation marks as shown below.

```
<person age=“22”>
```

XML content

The data in XML files is also called XML content. For example, in the XML file, you might see data like this.

```
<friend>

    <name>Charlie</name>

    <name>Steve</name>

</friend>
```

The data values *Charlie* and *Steve* are the content.

XML Example

XML documents create a hierarchical structure looks like a tree so it is known as XML Tree that starts at "the root" and branches to "the leaves".

Example of XML Document

XML documents uses a self-describing and simple syntax:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).

The next line describes the root element of the document (like saying: "this document is a note"):

```
<note>
```

The next 4 lines describe 4 child elements of the root (to, from, heading, and body).

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

And finally the last line defines the end of the root element.

```
</note>
```

XML documents must contain a **root element.** This element is "the parent" of all other elements.

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

All elements can have sub elements (child elements).

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

Another Example of XML: Books

File: books.xml

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

The root element in the example is <bookstore>. All elements in the document are contained within <bookstore>.

The <book> element has 4 children: <title>,< author>, <year> and <price>.

Another Example of XML: Emails

File: emails.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<emails>
<email>
  <to>Vimal</to>
  <from>Sonoo</from>
  <heading>Hello</heading>
  <body>Hello brother, how are you!</body>
</email>
<email>
  <to>Peter</to>
  <from>Jack</from>
  <heading>Birth day wish</heading>
  <body>Happy birth day Tom!</body>
</email>
<email>
  <to>James</to>
  <from>Jaclin</from>
  <heading>Morning walk</heading>
  <body>Please start morning walk to stay fit!</body>
</email>
<email>
  <to>Kartik</to>
  <from>Kumar</from>
  <heading>Health Tips</heading>
  <body>Smoking is injurious to health!</body>
</email>
</emails>
```

XML Related Technologies

Here we have pointed out XML related technologies. There are following XML related technologies:

No.	Technology	Meaning	Description
1)	XHTML	Extensible html	It is a clearer and stricter version of XML. It belongs to the family of XML markup languages. It was developed to make html more extensible and increase inter-operability with other data.
2)	XML DOM	XML document object model	It is a standard document model that is used to access and manipulate XML. It defines the XML file in tree structure.
3)	XSL it contain three parts: i) XSLT (xsl transform) ii) XSL iii)XPath	Extensible style sheet language	i) It transforms XML into other formats, like html. ii) It is used for formatting XML to screen, paper etc. iii) It is a language to navigate XML documents.
4)	XQuery	XML query language	It is a XML based language which is used to query XML based data.
5)	DTD	Document type definition	It is an standard which is used to define the legal elements in an XML document.
6)	XSD	XML schema definition	It is an XML based alternative to dtd. It is used to describe the structure of an XML document.
7)	XLink	XML linking language	xlink stands for XML linking language. This is a language for creating hyperlinks (external and internal links) in XML documents.
8)	XPointer	XML pointer language	It is a system for addressing components of XML based internet media. It allows the xlink hyperlinks to point to more specific parts in the XML document.
9)	SOAP	Simple object access protocol	It is an acronym stands simple object access protocol. It is XML based protocol to let applications exchange information over http. in simple words you can say that it is protocol used for accessing web services.
10)	WSDL	web services description	It is an XML based language to describe web services. It also describes the functionality

		languages	offered by a web service.
11)	RDF	Resource description framework	RDF is an XML based language to describe web resources. It is a standard model for data interchange on the web. It is used to describe the title, author, content and copyright information of a web page.
12)	SVG	Scalable vector graphics	It is an XML based vector image format for two-dimensional images. It defines graphics in XML format. It also supports animation.
13)	RSS	Really simple syndication	RSS is a XML-based format to handle web content syndication. It is used for fast browsing for news and updates. It is generally used for news like sites.

XML Attributes

XML elements can have attributes. By the use of attributes we can add the information about the element.

XML attributes enhance the properties of the elements.

Note: XML attributes must always be quoted. We can use single or double quote.

Let us take an example of a book publisher. Here, book is the element and publisher is the attribute.

1. <**book** publisher="Tata McGraw Hill"></**book**>

Or

1. <**book** publisher='Tata McGraw Hill'></**book**>

Metadata should be stored as attribute and data should be stored as element.

1. <**book**>
2. <**book** category="computer">
3. <**author**> A & B </**author**>
4. </**book**>

Data can be stored in attributes or in child elements. But there are some limitations in using attributes, over child elements.

Why should we avoid XML attributes?

- Attributes cannot contain multiple values but child elements can have multiple values.
- Attributes cannot contain tree structure but child element can.
- Attributes are not easily expandable. If you want to change in attribute's vales in future, it may be complicated.
- Attributes cannot describe structure but child elements can.
- Attributes are more difficult to be manipulated by program code.
- Attributes values are not easy to test against a DTD, which is used to define the legal elements of an XML document.

Difference between attribute and sub-element

In the context of documents, attributes are part of markup, while sub elements are part of the basic document contents.

In the context of data representation, the difference is unclear and may be confusing.

Same information can be represented in two ways:

1st way:

1. **<book** publisher="Tata McGraw Hill"> **</book>**

2nd way:

<book>

1. **<publisher>** Tata McGraw Hill **</publisher>**
2. **</book>**

In the first example publisher is used as an attribute and in the second example publisher is an element.

Both examples provide the same information but it is good practice to avoid attribute in XML and use elements instead of attributes.

XML Comments

XML comments are just like HTML comments. We know that the comments are used to make codes more understandable other developers.

XML Comments add notes or lines for understanding the purpose of an XML code. Although XML is known as self-describing data but sometimes XML comments are necessary.

Syntax An XML comment should be written as:

<!-- Write your comment-->

You cannot nest one XML comment inside the another.

XML Comments Example

Let's take an example to show the use of comment in an XML example:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!--Students marks are uploaded by months-->
<students>
  <student>
    <name>Ratan</name>
    <marks>70</marks>
  </student>
  <student>
    <name>Aryan</name>
    <marks>60</marks>
  </student>
 </students>
```

Rules for adding XML comments

- Don't use a comment before an XML declaration.
- You can use a comment anywhere in XML document except within attribute value.
- Don't nest a comment inside the other comment.

XML Tree Structure

An XML document has a self descriptive structure. It forms a tree structure which is referred as an XML tree. The tree structure makes easy to describe an XML document.

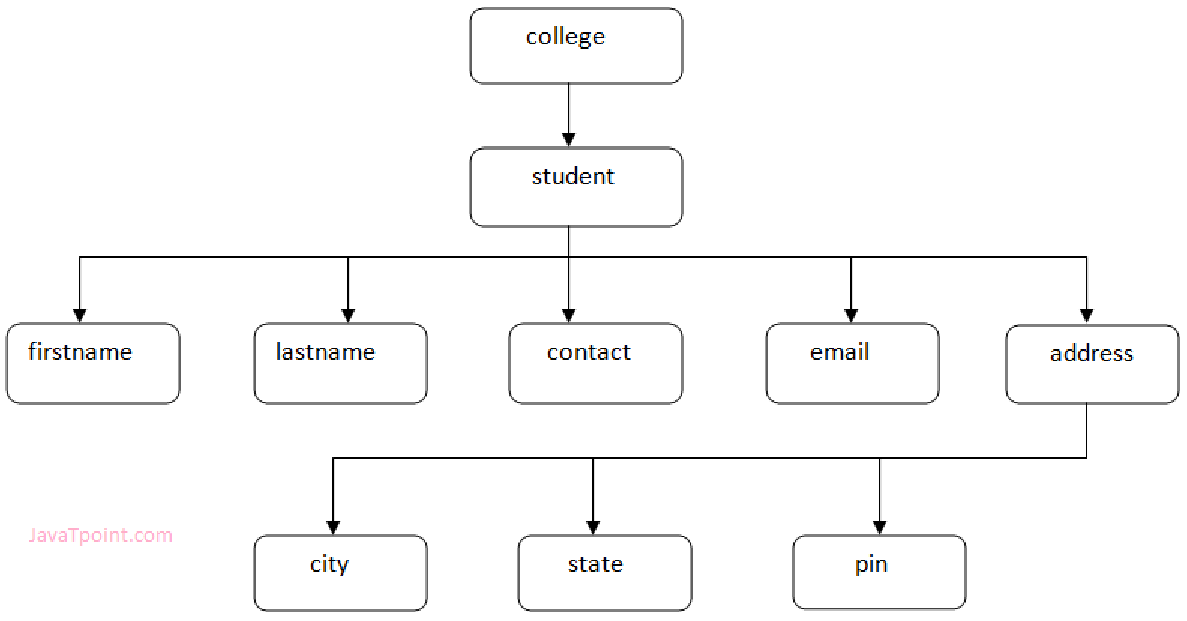
A tree structure contains root element (as parent), child element and so on. It is very easy to traverse all succeeding branches and sub-branches and leaf nodes starting from the root.

Example of an XML document

```xml
<?xml version="1.0"?>
<college>
  <student>
    <firstname>Tamanna</firstname>
    <lastname>Bhatia</lastname>
    <contact>09990449935</contact>
    <email>tammanabhatia@abc.com</email>
    <address>
        <city>Ghaziabad</city>
```

```
            <state>Uttar Pradesh</state>
            <pin>201007</pin>
        </address>
    </student>
</college>
```

Let's see the tree-structure representation of the above example.



In the above example, first line is the XML declaration. It defines the XML version 1.0. Next line shows the root element (college) of the document. Inside that there is one more element (student). Student element contains five branches named <firstname>, <lastname>, <contact>, <Email> and <address>.

<address> branch contains 3 sub-branches named <city>, <state> and <pin>.

Note: DOM parser represents the XML document in Tree structure.

XML Tree Rules

These rules are used to figure out the relationship of the elements. It shows if an element is a child or a parent of the other element.

Descendants: If element A is contained by element B, then A is known as descendant of B. In the above example "College" is the root element and all the other elements are the descendants of "College".

Ancestors: The containing element which contains other elements is called "Ancestor" of other element. In the above example Root element (College) is ancestor of all other elements.

XML Validation

A well formed XML document can be validated against DTD or Schema.

A well-formed XML document is an XML document with correct syntax. It is very necessary to know about valid XML document before knowing XML validation.

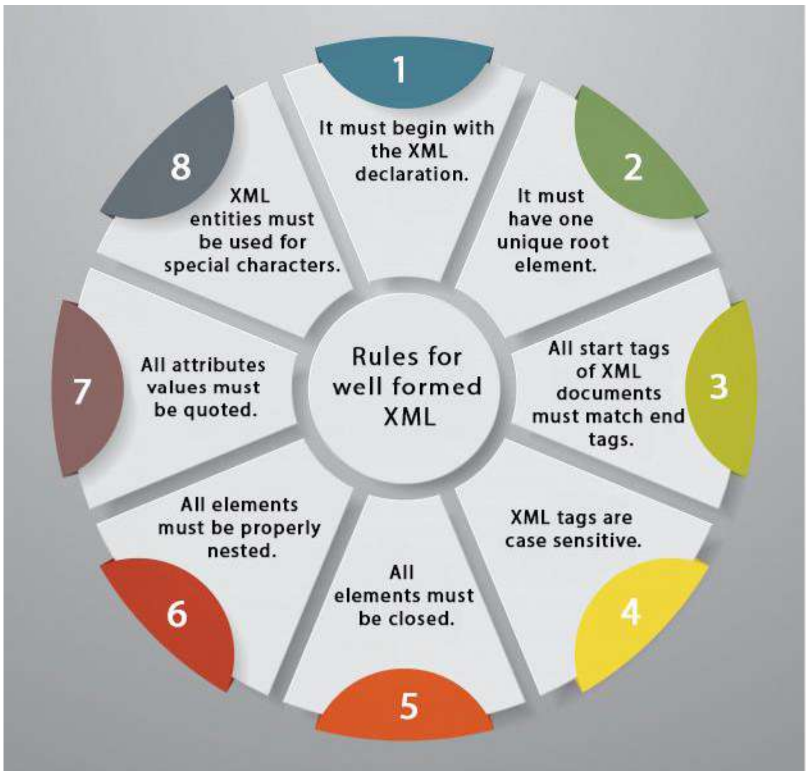
Valid XML document

It must be well formed (satisfy all the basic syntax condition)

It should be behave according to predefined DTD or XML schema

Rules for well formed XML

- It must begin with the XML declaration.
- It must have one unique root element.
- All start tags of XML documents must match end tags.
- XML tags are case sensitive.
- All elements must be closed.
- All elements must be properly nested.
- All attributes values must be quoted.
- XML entities must be used for special characters.


Rules for well formed XML
1
It must begin with the XML declaration.
2
It must have one unique root element.
3
All start tags of XML documents must match end tags.
4
XML tags are case sensitive.
5
All elements must be closed.
6
All elements must be properly nested.
7
All attributes values must be quoted.
8
XML entities must be used for special characters.

XML DTD

A DTD defines the legal elements of an XML document

In simple words we can say that a DTD defines the document structure with a list of legal elements and attributes.

XML schema is a XML based alternative to DTD.

Actually DTD and XML schema both are used to form a well formed XML document.

We should avoid errors in XML documents because they will stop the XML programs.

XML schema

It is defined as an XML language

Uses namespaces to allow for reuses of existing definitions

It supports a large number of built in data types and definition of derived data types

What is DTD?

DTD stands for **Document Type Definition**. It defines the legal building blocks of an XML document. It is used to define document structure with a list of legal elements and attributes.

Purpose of DTD

Its main purpose is to define the structure of an XML document. It contains a list of legal elements and define the structure with the help of them.

Checking Validation

Before proceeding with XML DTD, you must check the validation. An XML document is called "well-formed" if it contains the correct syntax.

A well-formed and valid XML document is one which have been validated against DTD.

Valid and well-formed XML document with DTD

Let's take an example of well-formed and valid XML document. It follows all the rules of DTD.

employee.xml

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

In the above example, the DOCTYPE declaration refers to an external DTD file. The content of the file is shown in below paragraph.

employee.dtd

```
<!ELEMENT employee (firstname,lastname,email)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Description of DTD

<!DOCTYPE employee : It defines that the root element of the document is employee.

<!ELEMENT employee: It defines that the employee element contains 3 elements "firstname, lastname and email".

<!ELEMENT firstname: It defines that the firstname element is #PCDATA typed. (parse-able data type).

<!ELEMENT lastname: It defines that the lastname element is #PCDATA typed. (parse-able data type).

<!ELEMENT email: It defines that the email element is #PCDATA typed. (parse-able data type).

XML DTD with entity declaration

A doctype declaration can also define special strings that can be used in the XML file.

An entity has three parts:

1. An ampersand (&)
2. An entity name
3. A semicolon (;)

Syntax to declare entity:

```
<!ENTITY entity-name "entity-value">
```

Let's see a code to define the ENTITY in doctype declaration.

author.xml

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE author [
  <!ELEMENT author (#PCDATA)>
  <!ENTITY sj "Sonoo Jaiswal">
]>
<author>&sj;</author>
```

In the above example, sj is an entity that is used inside the author element. In such case, it will print the value of sj entity that is "Sonoo Jaiswal".

Note: A single DTD can be used in many XML files.

XML Schema

What is XML schema?

XML schema is a language which is used for expressing constraint about XML documents. There are so many schema languages which are used now a days for example Relax- NG and XSD (XML schema definition).

An XML schema is used to define the structure of an XML document. It is like DTD but provides more control on XML structure.

Checking Validation

An XML document is called "well-formed" if it contains the correct syntax. A well-formed and valid XML document is one which have been validated against Schema.

XML Schema Example

Let's create a schema file.

employee.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.javatpoint.com"
xmlns="http://www.javatpoint.com"
elementFormDefault="qualified">

<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Let's see the xml file using XML schema or XSD file.

employee.xml

```xml
<?xml version="1.0"?>
<employee
xmlns="http://www.javatpoint.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.javatpoint.com employee.xsd">

  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

Description of XML Schema

<xs:element name="employee"> : It defines the element name employee.

<xs:complexType> : It defines that the element 'employee' is complex type.

<xs:sequence> : It defines that the complex type is a sequence of elements.

<xs:element name="firstname" type="xs:string"/> : It defines that the element 'firstname' is of string/text type.

<xs:element name="lastname" type="xs:string"/> : It defines that the element 'lastname' is of string/text type.

<xs:element name="email" type="xs:string"/> : It defines that the element 'email' is of string/text type.

XML Schema Data types

There are two types of data types in XML schema.

1. simpleType
2. complexType

simpleType

The simpleType allows you to have text-based elements. It contains less attributes, child elements, and cannot be left empty.

complexType

The complexType allows you to hold multiple attributes and elements. It can contain additional sub elements and can be left empty.

DTD vs XSD

There are many differences between DTD (Document Type Definition) and XSD (XML Schema Definition). In short, DTD provides less control on XML structure whereas XSD (XML schema) provides more control.

The important differences are given below:

No.	DTD	XSD
1)	DTD stands for **Document Type Definition**.	XSD stands for XML Schema Definition.
2)	DTDs are derived from **SGML** syntax.	XSDs are written in XML.
3)	DTD **doesn't support datatypes**.	XSD **supports datatypes** for elements and attributes.
4)	DTD **doesn't support namespace**.	XSD **supports namespace**.
5)	DTD **doesn't define order** for child elements.	XSD **defines order** for child elements.
6)	DTD is **not extensible**.	XSD is **extensible**.
7)	DTD is **not simple to learn**.	XSD is **simple to learn** because you don't need to learn new language.
8)	DTD provides **less control** on XML structure.	XSD provides **more control** on XML structure.

CDATA vs PCDATA

CDATA

CDATA: (Unparsed Character data): CDATA contains the text which is not parsed further in an XML document. Tags inside the CDATA text are not treated as markup and entities will not be expanded.

Let's take an example for CDATA:

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
<![CDATA[
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
]]>
</employee>
```

In the above CDATA example, CDATA is used just after the element employee to make the data/text unparsed, so it will give the value of employee:

```
<firstname>vimal</firstname><lastname>jaiswal</lastname><email>vimal@javatpoint.com
</email>
```

PCDATA

PCDATA: (Parsed Character Data): XML parsers are used to parse all the text in an XML document. PCDATA stands for Parsed Character data. PCDATA is the text that will be parsed by a parser. Tags inside the PCDATA will be treated as markup and entities will be expanded.

In other words you can say that a parsed character data means the XML parser examine the data and ensure that it doesn't content entity if it contains that will be replaced.

Let's take an example:

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

In the above example, the employee element contains 3 more elements 'firstname', 'lastname', and 'email', so it parses further to get the data/text of firstname, lastname and email to give the value of employee as:

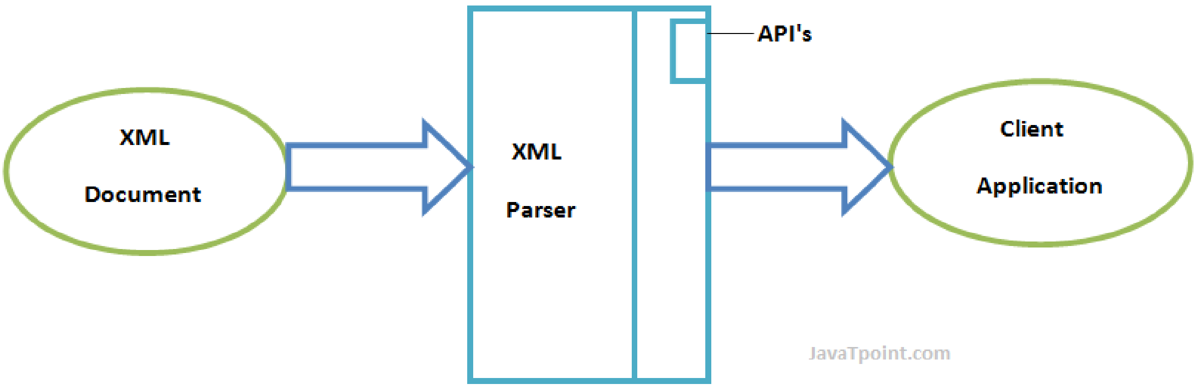
```
vimal jaiswal vimal@javatpoint.com
```

XML Parsers

An XML parser is a software library or package that provides interfaces for client applications to work with an XML document. The XML Parser is designed to read the XML and create a way for programs to use XML.

XML parser validates the document and check that the document is well formatted.

Let's understand the working of XML parser by the figure given below:



Types of XML Parsers

These are the two main types of XML Parsers:

1. DOM
2. SAX

DOM (Document Object Model)

A DOM document is an object which contains all the information of an XML document. It is composed like a tree structure. The DOM Parser implements a DOM API. This API is very simple to use.

Features of DOM Parser

A DOM Parser creates an internal structure in memory which is a DOM document object and the client applications get information of the original XML document by invoking methods on this document object.

DOM Parser has a tree based structure.

Advantages

1) It supports both read and write operations and the API is very simple to use.

2) It is preferred when random access to widely separated parts of a document is required.

Disadvantages

1) It is memory inefficient. (consumes more memory because the whole XML document needs to loaded into memory).

2) It is comparatively slower than other parsers.

SAX (Simple API for XML)

A SAX Parser implements SAX API. This API is an event based API and less intuitive.

Features of SAX Parser

It does not create any internal structure.

Clients does not know what methods to call, they just overrides the methods of the API and place his own code inside method.

It is an event based parser, it works like an event handler in Java.

Advantages

1) It is simple and memory efficient.

2) It is very fast and works for huge documents.

Disadvantages

1) It is event-based so its API is less intuitive.

2) Clients never know the full information because the data is broken into pieces.

XML DOM

What is XML DOM?

DOM is an acronym stands for Document Object Model. It defines a standard way to access and manipulate documents. The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

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

XML DOM defines a standard way to access and manipulate XML documents.

What does XML DOM?

The XML DOM makes a tree-structure view for an XML document.

We can access all elements through the DOM tree.

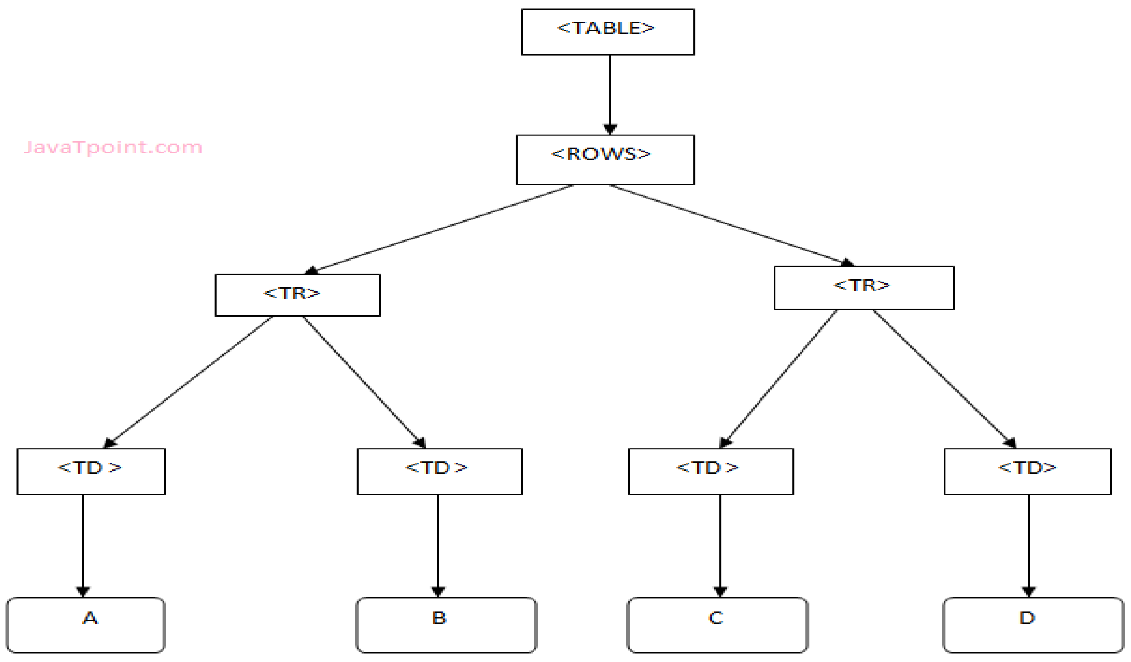
We can modify or delete their content and also create new elements. The elements, their content (text and attributes) are all known as nodes.

For example, consider this table, taken from an HTML document:

1. **<TABLE>**
2. **<ROWS>**

3. **<TR>**
4. **<TD>**A**</TD>**
5. **<TD>**B**</TD>**
6. **</TR>**
7. **<TR>**
8. **<TD>**C**</TD>**
9. **<TD>**D**</TD>**
10. **</TR>**
11. **</ROWS>**
12. **</TABLE>**

The Document Object Model represents this table like this:



XML DOM Example : Load XML File

Let's take an example to show how an XML document ("note.xml") is parsed into an XML DOM object.

This example parses an XML document (note.xml) into an XML DOM object and extracts information from it with JavaScript.

Let's see the XML file that contains message.

note.xml

1. **<?xml** version="1.0" encoding="ISO-8859-1"**?>**
2. **<note>**
3. **<to>**sonoojaiswal@javatpoint.com**</to>**

```
  <from>vimal@javatpoint.com</from>
  <body>Hello XML DOM</body>
</note>
```

Let's see the HTML file that extracts the data of XML document using DOM.

xmldom.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Important Note</h1>
<div>
<b>To:</b> <span id="to"></span><br>
<b>From:</b> <span id="from"></span><br>
<b>Message:</b> <span id="message"></span>
</div>
<script>
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.open("GET","note.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;
document.getElementById("to").innerHTML=
xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue;
document.getElementById("from").innerHTML=
xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;
document.getElementById("message").innerHTML=
xmlDoc.getElementsByTagName("body")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

Output:

Important Note
To: sonoojaiswal@javatpoint.com
From: vimal@javatpoint.com
Message: Hello XML DOM

XML DOM Example : Load XML String

This example parses an XML string into an XM DOM object and then extracts some information from it with a JavaScript.

Let's see the HTML file that extracts the data of XML string using DOM.

xmldom.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Important Note2</h1>
<div>
<b>To:</b> <span id="to"></span><br>
<b>From:</b> <span id="from"></span><br>
<b>Message:</b> <span id="message"></span>
</div>
<script>
txt1="<note>";
txt2="<to>Sania Mirza</to>";
txt3="<from>Serena William</from>";
txt4="<body>Don't forget me this weekend!</body>";
txt5="</note>";
txt=txt1+txt2+txt3+txt4+txt5;

if (window.DOMParser)
  {
  parser=new DOMParser();
  xmlDoc=parser.parseFromString(txt,"text/xml");
  }
else // Internet Explorer
  {
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async=false;
  xmlDoc.loadXML(txt);
  }
```

```
document.getElementById("to").innerHTML=
xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue;
document.getElementById("from").innerHTML=
xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;
document.getElementById("message").innerHTML=
xmlDoc.getElementsByTagName("body")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

Hence, conclusive differences between SAX Parser and DOM Parser in Java is as follows

SAX Parser	**DOM Parser**
It is called a Simple API for XML Parsing.	It is called as Document Object Model.
It's an event-based parser.	It stays in a tree structure.
SAX Parser is slower than DOM Parser.	DOM Parser is faster than SAX Parser.
Best for the larger sizes of files.	Best for the smaller size of files.
It is suitable for making XML files in Java.	It is not good at making XML files in low memory.
The internal structure can not be created by SAX Parser.	The internal structure can be created by DOM Parser.
It is read-only.	It can insert or delete nodes.
In the SAX parser backward navigation is not possible.	In DOM parser backward and forward search is possible
Suitable for efficient memory.	Suitable for large XML document.
A small part of the XML file is only loaded in memory.	It loads whole XML documents in memory.

Parsing XML Data – DOM and SAX in java.

The sample XML considered in the examples is:

```
<employees>
  <employee id="111">
    <firstName>Rakesh</firstName>
    <lastName>Mishra</lastName>
    <location>Bangalore</location>
  </employee>
  <employee id="112">
    <firstName>John</firstName>
    <lastName>Davis</lastName>
    <location>Chennai</location>
  </employee>
  <employee id="113">
    <firstName>Rajesh</firstName>
    <lastName>Sharma</lastName>
    <location>Pune</location>
  </employee>
</employees>
```

And the obejct into which the XML content is to be extracted is defined as below:

```
class Employee{
  String id;
  String firstName;
  String lastName;
  String location;

  @Override
  public String toString() {
    return firstName+" "+lastName+"("+id+")"+location;
  }
}
```

There are 3 main parsers for which I have given sample code:

- DOM Parser
- SAX Parser
- StAX Parser

Using DOM Parser

I am making use of the DOM parser implementation that comes with the JDK and in my example I am using JDK 7. The DOM Parser loads the complete XML content into a Tree structure. And we iterate through the Node and NodeList to get the content of the XML. The code for XML parsing using DOM parser is given below.

```
public class DOMParserDemo {

  public static void main(String[] args) throws Exception {
    //Get the DOM Builder Factory
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();

    //Get the DOM Builder
    DocumentBuilder builder = factory.newDocumentBuilder();

    //Load and Parse the XML document
    //document contains the complete XML as a Tree.
    Document document =
      builder.parse(
        ClassLoader.getSystemResourceAsStream("xml/employee.xml"));

    List<Employee> empList = new ArrayList<>();

    //Iterating through the nodes and extracting the data.
    NodeList nodeList = document.getDocumentElement().getChildNodes();

    for (int i = 0; i < nodeList.getLength(); i++) {

      //We have encountered an <employee> tag.
      Node node = nodeList.item(i);
      if (node instanceof Element) {
        Employee emp = new Employee();
        emp.id = node.getAttributes().
            getNamedItem("id").getNodeValue();

        NodeList childNodes = node.getChildNodes();
        for (int j = 0; j < childNodes.getLength(); j++) {
          Node cNode = childNodes.item(j);

          //Identifying the child tag of employee encountered.
          if (cNode instanceof Element) {
            String content = cNode.getLastChild().
                getTextContent().trim();
            switch (cNode.getNodeName()) {
              case "firstName":
                emp.firstName = content;
                break;
              case "lastName":
                emp.lastName = content;
                break;
              case "location":
                emp.location = content;
                break;
            }
          }
        }
        empList.add(emp);
      }

    }
```

```

        //Printing the Employee list populated.
        for (Employee emp : empList) {
          System.out.println(emp);
        }

      }
    }

    class Employee{
      String id;
      String firstName;
      String lastName;
      String location;

      @Override
      public String toString() {
        return firstName+" "+lastName+"("+id+")"+location;
      }
    }
```

The output for the above will be:

```
Rakesh Mishra(111)Bangalore
John Davis(112)Chennai
Rajesh Sharma(113)Pune
```

Using SAX Parser

SAX Parser is different from the DOM Parser where SAX parser doesn't load the complete XML into the memory, instead it parses the XML line by line triggering different events as and when it encounters different elements like: opening tag, closing tag, character data, comments and so on. This is the reason why SAX Parser is called an event based parser.

Along with the XML source file, we also register a handler which extends the DefaultHandler class. The DefaultHandler class provides different callbacks out of which we would be interested in:

- **startElement()** – triggers this event when the start of the tag is encountered.
- **endElement()** – triggers this event when the end of the tag is encountered.
- **characters()** – triggers this event when it encounters some text data.

The code for parsing the XML using SAX Parser is given below:

```
import java.util.ArrayList;
import java.util.List;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
```

```

public class SAXParserDemo {

  public static void main(String[] args) throws Exception {
    SAXParserFactory parserFactor = SAXParserFactory.newInstance();
    SAXParser parser = parserFactor.newSAXParser();
    SAXHandler handler = new SAXHandler();
    parser.parse(ClassLoader.getSystemResourceAsStream("xml/employee.xml"),
             handler);

    //Printing the list of employees obtained from XML
    for ( Employee emp : handler.empList){
      System.out.println(emp);
    }
  }
}
/**
 * The Handler for SAX Events.
 */
class SAXHandler extends DefaultHandler {

  List<Employee> empList = new ArrayList<>();
  Employee emp = null;
  String content = null;
  @Override
  //Triggered when the start of tag is found.
  public void startElement(String uri, String localName,
                           String qName, Attributes attributes)
                           throws SAXException {

    switch(qName){
      //Create a new Employee object when the start tag is found
      case "employee":
        emp = new Employee();
        emp.id = attributes.getValue("id");
        break;
    }
  }

  @Override
  public void endElement(String uri, String localName,
                         String qName) throws SAXException {
   switch(qName){
     //Add the employee to list once end tag is found
     case "employee":
       empList.add(emp);
       break;
     //For all other end tags the employee has to be updated.
     case "firstName":
       emp.firstName = content;
       break;
     case "lastName":
       emp.lastName = content;
```

```
          break;
        case "location":
          emp.location = content;
          break;
       }
     }

     @Override
     public void characters(char[] ch, int start, int length)
             throws SAXException {
       content = String.copyValueOf(ch, start, length).trim();
     }

   }

   class Employee {

     String id;
     String firstName;
     String lastName;
     String location;

     @Override
     public String toString() {
       return firstName + " " + lastName + "(" + id + ")" + location;
     }
   }
```

The output for the above would be:

```
Rakesh Mishra(111)Bangalore
John Davis(112)Chennai
Rajesh Sharma(113)Pune
```

What is XHTML?

XHTML stands for extensible hypertext markup language which is a connection between HTML (hypertext mark-up language) and XML (extensible markup language) also at most of the places XHTML is considered superior than HTML.

XHTML is easy to use with other data formats, and it creates more neat code as it is stricter than HTML. Therefore, it is more compatible with most browsers, and it maintains a standard of code that can be used for various devices.

Example

Below is an example of XHTML,

```
<!DOCTYPE html PUBLIC "-// W //DT XHTML 1.2 //EN"
" http : // www . myblogpost .org/T /xhtml12 / DT / xhtml12.dtd">
<html xmlns=http://www. myblogpost . org / 199 / xhtml >
```

```
<head>
<title> XHTML document </title>
</head>
<body>

Wrong XHTML rule<br>
Correct XHTML rule<br />

Wrong XHTML rule <hr>
Correct XHTML rule <hr />

Wrong XHTML rule <img src=" pic/ document / mypic.gif" alt=" picture of a boy playing i
n a field  ">
Correct XHTML rule <img src="pic/ document / mypic.gif" alt="picture of a boy playing in
a field " />

</body>
</html>
```

Output

The above code gives the following output.

```
Correct XHTML rule

Correct XHTML rule

────────────────────────────────────

Correct XHTML rule
```

Advantages of XHTML

Here are the following advantages of XHTML, such as:

- While using XHTML, the code of web applications becomes more stylish and easy to reuse.
- It can help the developer create more advanced web projects due to the compatibility with various devices, and it also supports self-created markups like SVG (scalable vector graphics).
- XHTML code can easily be converted to PDFs, RSS, and RFT, which allows the developer to work with a vast range of files.

- XHTML reduce the loading time required by the browser to load an event which can result in overall speedy development, thus reducing time and energy
- It contains closing tags which is an advantage for beginners, and this also makes the code look clean and easy to reuse.

Disadvantages of XHTML

XHTML also has some disadvantages, such as:

- Very few browsers use XHTML.
- Case sensitive as every part of code should be in lowercase.
- It is mandatory to write < DOCTYPE > declaration.
- And all the tags must be closed in the necessary order.

HTML vs XML

There are many differences between HTML (Hyper Text Markup Language) and XML (eXtensible Markup Language). The important differences are given below:

No.	HTML	XML
1)	HTML is used **to display data** and focuses on how data looks.	XML is a software and hardware independent tool used **to transport and store data**. It focuses on what data is.
2)	HTML is a **markup language** itself.	XML provides a **framework to define markup languages**.
3)	HTML is **not case sensitive**.	XML is **case sensitive**.
4)	HTML is a presentation language.	XML is neither a presentation language nor a programming language.
5)	HTML **has its own predefined tags**.	You **can define tags according to your need**.
6)	In HTML, it is **not necessary to use a closing tag**.	XML **makes it mandatory to use a closing tag**.
7)	HTML is **static** because it is used to display data.	XML is **dynamic** because it is used to transport data.
8)	HTML **does not preserve whitespaces**.	XML **preserve whitespaces**.

Difference between HTML & XHTML

S.No.	HTML	XHTML
1.	Hypertext mark-up language - - > HTML	Extensible Hypertext Mark-up Language - - > XHTML.
2.	Tim Berners created in 1991	World wide web consortium or W3C created in 2000
4.	It is an extension of standard generalized markup language or SGML	It is a combination of extensible markup language XML and hypertext markup language HTML
5.	It stored in a document file format	It stored as a markup language format
6.	It is not case sensitive as there is no mandatory rule to write the entire mark up in uppercase or lower case. It can also be a combination of both.	It is case-sensitive, and every tag and attribute used inside must be in lowercase.
7.	It is not mandatory to add document label < DOCTYPE >at the top of every page. We can even skip it.	It is mandatory to add a document label < DOCTYPE > at the beginning of the page.
8.	We can close any tag anytime and anywhere as per our needs	It is mandatory to close all the tags in strict residing order as they were declared.
9.	We can add attributes without any quotes.	It is mandatory to add quotes on every attribute we declare
10.	,html and .htm are the extensions used by HTML	.xhtml, .xml and .xht are the file extensions used by XHTML
11	Lewd structure is used	It contains a very strict structure, and the developer cannot go out of the bounds of these structures.

UNIT- III - Servlets

1. What is Servlet? Explain about Servlet Container?

Servlet : A servlet is a Java technology based web component, managed by a container, which generates dynamic content. Like other Java-based components, servlets are platform independent Java classes that are compiled to platform neutral bytecode that can be loaded dynamically into and run by a Java enabled web server. Containers, sometimes called servlet engines, are web server extensions that provide servlet functionality. Servlets interact with web clients via a request/response paradigm implemented by the servlet container.

Servlet Container : The servlet container is a part of a web server or application server that provides the network services over which requests and responses are sent, decodes MIME based requests, and formats MIME based responses. A servlet container also contains and manages servlets through their lifecycle.

A servlet container can be built into a host web server, or installed as an add on component to a Web Server via that server's native extension API. Servlet containers can also be built into or possibly installed into web-enabled application servers.

All servlet containers must support HTTP as a protocol for requests and responses, but additional request/response based protocols such as HTTPS (HTTP over SSL) may be supported. The minimum required version of the HTTP specification that a container must implement is HTTP/1.0. It is strongly suggested that containers implement the HTTP/1.1 specification as well.

An Example

The following is a typical sequence of events:

1. A client (e.g., a web browser) accesses a web server and makes an HTTP request.
2. The request is received by the web server and handed off to the servlet container. The servlet container can be running in the same process as the host web server, in a different process on the same host, or on a different host from the web server for which it processes requests.
3. The servlet container determines which servlet to invoke based on the configuration of its servlets, and calls it with objects representing the request and response.
4. The servlet uses the request object to find out that the remote user is, what HTTP POST parameters may have been sent as part of this request, and other relevant data. The servlet performs whatever logic it was programmed with, and generates data to send back to the client. It sends this data back to the client via the response object.
5. Once the servlet has finished processing the request, the servlet container ensures that the response is properly flushed, and returns control back to the host web server.

Comparing Servlets with Other Technologies

In functionality, servlets lie somewhere between Common Gateway Interface (CGI) programs and proprietary server extensions such as the Netscape Server API (NSAPI) or Apache Modules.

Servlets have the following advantages over other server extension mechanisms:

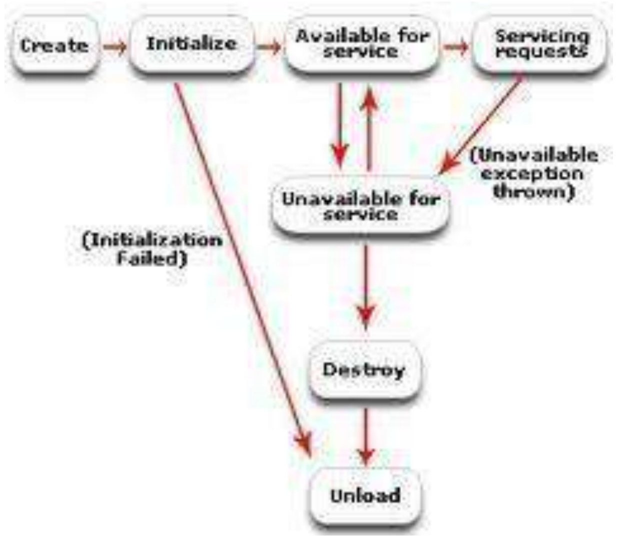
1. They are generally much faster than CGI scripts because a different process model is used.
2. They use a standard API that is supported by many web servers.
3. They have all the advantages of the Java programming language, including ease of development and platform independence.
4. They can access the large set of APIs available for the Java platform.

2. Explain Servlet Life Cycle Methods.

Servlets run on the web server platform as part of the same process as the web server itself. The web server is responsible for initializing, invoking, and destroying each servlet instance.

A web server communicates with a servlet through a simple interface, javax.servlet.Servlet. This interface consists of three main methods:

- init()
- service()
- destroy()



Servlet Life Cycle Architecture

The init() Method

When a servlet is first loaded, its init () method is invoked. This allows the servlet to per form any setup processing such as opening files or establishing connections to their servers. If a servlet has been permanently installed in a server, it loads when the server starts to run. Otherwise, the server activates a servlet when it receives the first client request for the services provided by the servlet.

The init() method is guaranteed to finish before any other calls are made to the servlet--such as a call to the service() method. Note that init() will only be called once; it will not be called again unless the servlet has been unloaded and then reloaded by the server.

The init() method takes one argument, a reference to a ServletConfig object which provides initialization arguments for the servlet. This object has a method getServletContext() that returns a ServletContext object containing information about the servlet's environment (see the discussion on Servlet Initialization Context below).

The service() Method

The service() method is the heart of the servlet. Each request message from a client results in a single call to the servlet's service () method. The service() method reads the request and produces the response message from its two parameters:

- A ServletRequest object with data from the client. The data consists of name/value pairs of parameters and an InputStream. Several methods are provided that return the client's parameter information. The InputStream from the client can be obtained via the getInputStream() method. This method returns a ServletInputStream, which can be used to get additional data from the client. If you are interested in processing character-level data instead of byte-level data, you can get a BufferedReader instead with getReader().
- A ServletResponse represents the servlet's reply back to the client. When preparing a response, the method setContentType() is called first to set the MIME type of the reply. Next, the method getOutputStream() or getWriter() can be used to obtain a ServletOutputStream or PrintWriter, respectively, to send data back to the client.

As you can see, there are two ways for a client to send information to a servlet. The first is to send parameter values and the second is to send information via the InputStream (or Reader). Parameter values can be embedded into a URL. How this is done is discussed below. How the parameter values are read by the servlet is discussed later.

The service() method's job is conceptually simple--it creates a response for each client request sent to it from the host server. However, it is important to realize that there can be multiple service requests being processed at once. If your service method requires any outside resources, such as files, databases, or some external data, you must ensure that resource access is thread-safe. Making your servlets thread-safe is discussed in a later section of this course.

The destroy() Method

The destroy() method is called to allow your servlet to clean up any resources (such as open files or database connections) before the servlet is unloaded. If you do not require any clean-up operations, this can be an empty method.

The server waits to call the destroy() method until either all service calls are complete, or a certain amount of time has passed. This means that the destroy() method *can* be called while some longer-running service() methods are still running. It is important that you write your destroy() method to avoid closing any necessary resources until all service() calls have completed.

Sample Servlet

The code below implements a simple servlet that returns a static HTML page to a browser. This example fully implements the Servletinterface.

```
import      java.io.*;      import
javax.servlet.*;
public class SampleServlet implements Servlet {private ServletConfig config;

   public    void    init    (ServletConfig    config)   throws
      ServletException   { this.config = config;
   }

   public void destroy() {} // do nothing public void service

   (ServletRequest req,
      ServletResponse res
   )     throws      ServletException,      IOException       {
      res.setContentType(   "text/html"   );   PrintWriter   out   =
      res.getWriter(); out.println( "<html>" );
      out.println( "<head>" );
      out.println( "<title>A Sample Servlet</title>" );out.println( "</head>" );
      out.println( "<body>" );
      out.println( "<h1>A Sample Servlet</h1>" );out.println( "</body>" );
      out.println( "</html>" );out.close();
   }
}
```

3. <u>Explain Servlet Protocol Support?</u>

The Servlet API provides a tight link between a server and servlets. This allows servlets to add new protocol support to a server. (You will see how HTTP support is provided for you in the API packages.) Essentially, any protocol that follows a request/response computing model can be implemented by a servlet. This could include:

- SMTP
- POP
- FTP

Servlet support is currently available in several web servers, and will probably start appearing in other types of application servers in the near future. You will use a web server to host the servlets in this class and only deal with the HTTP protocol.

Because HTTP is one of the most common protocols, and because HTML can provide such a rich presentation of information, servlets probably contribute the most to building HTTP based systems.

HTML Support

HTML can provide a rich presentation of information because of its flexibility and the range of content that it can support. Servlets can play a role in creating HTML content. In fact, servlet support for HTML is so common, the javax.servlet.http package is dedicated to supporting HTTP protocol and HTML generation.

Complex web sites often need to provide HTML pages that are tailored for each visitor, or even for each hit. Servlets can be written to process HTML pages and customize them as they are sent to a client. This can be as simple as on the fly substitutions or it can be as complex as compiling a grammar-based description of a page and generating custom HTML.

HTTP Support

Servlets that use the HTTP protocol are very common. It should not be a surprise that there is specific help for servlet developers who write them. Support for handling the HTTP protocol is provided in the package javax.servlet.http. Before looking at this package, take a look at the HTTP protocol itself.

HTTP stands for the HyperText Transfer Protocol. It defines a protocol used by web browsers and servers to communicate with each other. The protocol defines a set of text-based request messages called *HTTP methods*. (Note: The HTTP specification calls these *HTTP methods*; do not confuse this term with Java methods. Think of *HTTP methods* as messages requesting a certain type of response). The HTTP methods include:

- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE
- CONNECT
- OPTIONS

For this course, you will only need to look at only three of these methods: GET, HEAD, and POST.

The HTTP GETMethod

The HTTP GET method requests information from a web server. This information could be a file, output from a device on the server, or output from a program (such as a servlet or CGI script).

The POSTMethod

An HTTP POST request allows a client to send data to the server. This can be used for several purposes, such as

- Posting information to a newsgroup
- Adding entries to a web site's guest book
- Passing more information than a GETrequest allows

Pay special attention to the third bullet above. The HTTP GET request passes all its arguments as part of the URL. Many web servers have a limit to how much data they can accept as part of the URL. The POST method passes all of its parameter data in an input stream, removing this limit.

4. Explain Servlet Interface?

The Servlet interface is the central abstraction of the servlet API. All servlets implement this interface either directly, or more commonly, by extending a class that implements the interface. The two classes in the servlet API that implement the Servlet interface are GenericServlet and HttpServlet. For most purposes, developers will extend HttpServlet to implement their servlets.

Request Handling Methods

The basic Servlet interface defines a service method for handling client requests. This method is called for each request that the servlet container routes to an instance of a servlet.

The handling of concurrent requests to a web application generally requires the web developer design servlets that can deal with multiple threads executing within the service method at a particular time.

Generally the web container handles concurrent requests to the same servlet by concurrent execution of the service method on different threads.

HTTP Specific Request Handling Methods

The HttpServlet abstract subclass adds additional methods beyond the basic Servlet interface which are automatically called by the service method in the HttpServlet class to aid in processing HTTP based requests. These methods are:

- doGet for handling HTTP GET requests
- doPost for handling HTTP POST requests
- doPut for handling HTTP PUT requests
- doDelete for handling HTTP DELETE requests
- doHead for handling HTTP HEAD requests

- doOptions for handling HTTP OPTIONS requests
- doTrace for handling HTTP TRACE requests

Typically when developing HTTP based servlets, a Servlet Developer will only concern himself with the doGet and doPost methods. The other methods are considered to be methods for use by programmers very familiar with HTTP programming.

Example:

```
import       java.io.*;       import
javax.servlet.*;
import javax.servlet.http.*;
/** Simple servlet for testing the use of packages. */public class HelloServlet2 extends
HttpServlet {
        public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException,
        IOException {
                response.setContentType("text/html");   PrintWriter   out   =
                response.getWriter();
                out.println("<HTML><HEAD><TITLE>");
                out.println("Hello    (2)</TITLE></HEAD>");    out.println("<BODY
                BGCOLOR=\"#FDF5E6\">");
                out.println("<H1>Hello                              (2)</H1>");
                out.println("</BODY></HTML>");
        }
}
```

Servlet Context

A servlet lives and dies within the bounds of the server process. To understand its operating environment, a servlet can get information about its environment at different times. Servlet initialization information is available during servlet start-up; information about the hosting server is available at any time; and each service request can contain specific contextual information.

Server Context Information

Server context information is available at any time through the ServletContext object. A servlet can obtain this object by calling the getServletContext() method on the ServletConfig object. Remember that this was passed to the servlet during the initialization phase. A well written init() method saves the reference in a private variable.

The ServletContextinterface defines several methods. These are outlined below.

`getAttribute()`	An extensible way to get information about a server via attribute name/value pairs. This is server specific.
`getMimeType()`	Returns the MIME type of a given file.
`getRealPath()`	This method translates a relative or virtual path to a new path relative to the server's HTML documentation root location.

getServerInfo()	Returns the name and version of the network service under which the servlet is running.
getServlet()	Returns a Servlet object of a given name. Useful when you want to access the services of other servlets.
getServletNames()log()	Returns an enumeration of servlet names available in the current namespace.
	Writes information to a servlet log file. The log file name and format are server specific.

The Request :

The request object encapsulates all information from the client request. In the HTTP protocol, this information is transmitted from the client to the server in the HTTP headers and the message body of the request.

HTTP Protocol Parameters

Request parameters for the servlet are the strings sent by the client to a servlet container as part of its request. When the request is a HttpServletRequest object, and conditions set out below are met, the container populates the parameters from the URI query string and POST-ed data.

The parameters are stored as a set of name-value pairs. Multiple parameter values can exist for any given parameter name. The following methods of the ServletRequest interface are available to access parameters:

- getParameter
- getParameterNames
- getParameterValues

The getParameterValues method returns an array of String objects containing all the parameter values associated with a parameter name. The value returned from the getParameter method must be the first value in the array of String objects returned by getParameterValues.

Attributes

Attributes are objects associated with a request. Attributes may be set by the container to express information that otherwise could not be expressed via the API, or may be set by a servlet to communicate information to another servlet (via the RequestDispatcher). Attributes are accessed with the following methods of the ServletRequest interface:

- getAttribute
- getAttributeNames
- setAttribute

Only one attribute value may be associated with an attribute name.

Attribute names beginning with the prefixes of “java.” and “javax.” Are reserved for definition by this specification. Similarly attribute names beginning with the prefixes of "sun.", and "com.sun." are reserved for definition by Sun Microsystems. It is suggested that all attributes placed into the attribute set be named in accordance with the reverse domain name convention suggested by the Java Programming Language Specification1 for package naming.

Example program for getting user requests

HTML File

```
<html>
    <head>
            <title> Sample Program </title>
    </head>
    <body>
            <form name="loginForm" action="CheckValidUser" > Enter Name : <input
            type=text name="user" > <p> Enter Password : <input type=password
            name="pwd" ><p>
            <input type=submit value=" Send" >
            <input type=reset value="Clear" >
            </form>
    </body>
</html>
```

Servlet File

```
import      javax.servlet.http.*;      import
javax.servlet.*; import java.io.*;

public class CheckValidUser extends GenericServlet
{
    public void service(ServletRequest req, ServletResponse res) throws ServletException
            ,IOException
    {
            PrintWriter out = res.getWriter(); String user =
            req.getParameter("user");     String     pwd     =
            req.getParameter("pwd");
            if (user.equals("bhagavan") && pwd.equals("ram")) out.println("<br><h1> Welcome to Trendz</h1>
                    ");
else
                    out.println(" Invalid User ");
  }
}
```

Example program to get output stream from HttpServletResponse

```
import          java.io.ByteArrayOutputStream;        import
    java.io.DataInputStream;                          import
    java.io.DataOutputStream;                         import
    java.io.IOException;

    import        javax.servlet.ServletException;        import
    javax.servlet.ServletOutputStream;                   import
    javax.servlet.http.HttpServlet;
    import            javax.servlet.http.HttpServletRequest;          import
    javax.servlet.http.HttpServletResponse;                           import
    javax.servlet.http.HttpSession;

    public class CounterServer extends HttpServlet { static final String
      COUNTER_KEY = "Counter.txt";

      public    void    doPost(HttpServletRequest    req,    HttpServletResponse    resp)    th    rows
    ServletException,IOException {
        HttpSession session = req.getSession(true);int count = 1;
        Integer i = (Integer) session.getAttribute(COUNTER_KEY);if (i != null) {
          count = i.intValue() + 5;
        }
        session.setAttribute(COUNTER_KEY, new Integer(count)); DataInputStream in = new
        DataInputStream(req.getInputStream());         resp.setContentType("application/octet-
        stream");  ByteArrayOutputStream  byteOut  =  new  ByteArrayOutputStream();
        DataOutputStream out = new DataOutputStream(byteOut); out.writeInt(count);
        out.flush();
        byte[]      buf      =      byteOut.toByteArray();
        resp.setContentLength(buf.length);
        ServletOutputStream servletOut = resp.getOutputStream();servletOut.write(buf);
        servletOut.close();
      }
    }
```

5. <u>Explain Servlet RequestDispatcher Interface?</u>

Dispatching Requests

When building a web application, it is often useful to forward processing of a request to another servlet, or to include the output of another servlet in the response.

An object implementing the RequestDispatcher interface may be obtained from the ServletContext via the following methods:

- getRequestDispatcher
- getNamedDispatcher

The getRequestDispatcher method takes a String argument describing a path within the scope of the ServletContext. This path must be relative to the root of the

ServletContext and begin with a '/'. The method uses the path to look up a servlet, wraps it with a RequestDispatcher object, and returns the resulting object. If no servlet can be resolved based on the given path, a RequestDispatcher is provided that returns the content for that path.

The getNamedDispatcher method takes a String argument indicating the name of a servlet known to the ServletContext. If a servlet is found, it is wrapped with a RequestDispatcher object and the object returned. If no servlet is associated with the given name, the method must return null.

To allow RequestDispatcher objects to be obtained using relative paths that are relative to the path of the current request (not relative to the root of the ServletContext), the following method is provided in the ServletRequest interface:

- getRequestDispatcher

The behavior of this method is similar to the method of the same name in the ServletContext. The servlet container uses information in the request object to transform the given relative path against the current servlet to a complete path. For example, in a context rooted at '/' and a request to /garden/tools.html, a request dispatcher obtained via ServletRequest.getRequestDispatcher("header.html") will behave exactly like a call to ServletContext.getRequestDispatcher("/ garden/header.html").

Using a Request Dispatcher

To use a request dispatcher, a servlet calls either the include method or forward method of the RequestDispatcher interface. The parameters to these methods can be either the request and response arguments that were passed in via the service method of the Servlet interface, or instances of subclasses of the request and response wrapper classes that have been introduced for version 2.3 of the specification. In the latter case, the wrapper instances must wrap the request or response objects that the container passed into the service method.

The Container Provider must ensure that the dispatch of the request to a target servlet occurs in the same thread of the same VM as the original request.

The Include Method

The include method of the RequestDispatcher interface may be called at any time. The target servlet of the include method has access to all aspects of the request object, but its use of the response object is more limited:

It can only write information to the ServletOutputStream or Writer of the response object and commit a response by writing content past the end of the response buffer, or by explicitly calling the flushBuffer method of the ServletResponse interface. It cannot set headers or call any method that affects the headers of the response. Any attempt to do so must be ignored.

Example for Include Method:

```
import  javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import  javax.servlet.http.HttpServletResponse;

public class MultipleInclude extends HttpServlet

{

protected void doGet(HttpServletRequest request,
          HttpServletResponse response) throws ServletException,java.io.IOException {

        response.setContentType("text/html");    java.io.PrintWriter    out    =
        response.getWriter();

        out.println("<html>");out.println("<head>");
        out.println("<title>Multiple                              Includes</title>");
        out.println("</head>");
        out.println("<body>");  out.println("<h1>Hello  from  Level
        1</h1>");
        out.println("This text is displayed at Level 1.");
        RequestDispatcher    dispatcher    =    request.getRequestDispatcher("/Level4");    dispatcher.include(request,
        response);
        out.println("</body>");
        out.println("</html>");
        out.close();

    }
}

// here is another servlet
import  javax.servlet.ServletException;
import  javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import  javax.servlet.http.HttpServletResponse;

public class Level4 extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response
)
        throws ServletException, java.io.IOException {

                java.io.PrintWriter    out    =    response.getWriter();   out.println("<h4>Hello    from    another
              doGet</h4>");
        out.println("Hello from another doGet.");
    }
}
```

<u>web.xml</u>

```
      <servlet>
              <servlet-name>MultipleInclude</servlet-name>
              <servlet-class>MultipleInclude</servlet-class>
        </servlet>
```

```
<servlet-mapping>
        <servlet-name>MultipleInclude</servlet-name>
        <url-pattern>/MultipleInclude</url-pattern>
</servlet-mapping>
<servlet>
        <servlet-name>Level4</servlet-name>
        <servlet-class>Level4</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>Level4</servlet-name>
        <url-pattern>/Level4</url-pattern>
</servlet-mapping>
```

The Forward Method

The forward method of the RequestDispatcher interface may be called by the calling servlet only when no output has been committed to the client. If output data exists in the response buffer that has not been committed, the content must be cleared before the target servlet's service method is called. If the response has been committed, an IllegalStateException must be thrown.

The path elements of the request object exposed to the target servlet must reflect the path used to obtain the RequestDispatcher.

The only exception to this is if the RequestDispatcher was obtained via the getNamedDispatcher method. In this case, the path elements of the request object must reflect those of the original request.

Before the forward method of the RequestDispatcher interface returns, the response content must be sent and committed, and closed by the servlet container.

Example program for forward method and sendRedirect method:

loginSS.html file :

```
<html>
        <head>
                <title> Sample Forward Program </title>
        </head>
        <body>
                <form name="loginForm" action="CheckUser" > Enter Name : <input
                type=text name="user" > <p>
                Enter Password : <input type=password name="pwd" ><p>
                <input type=submit value=" Send" >
                <input type=reset value="Clear" >
                </form>
        </body>
</html>
```

CheckUser.java servlet program:

```
import javax.servlet.http.*;

import javax.servlet.*; import java.io.*;
```

```
public class CheckUser extends HttpServlet
{
        public void doGet(HttpServletRequest req, HttpServletResponse res)throws   ServletException
,IOException
        {
                RequestDispatcher disp; PrintWriter out =
                res.getWriter();
                String user = req.getParameter("user");String pwd =
                req.getParameter("pwd");
                if (user.equals("bhagavan") && pwd.equals("ram"))
                {
                        ServletContext context = getServletContext(); disp =
                        context.getRequestDispatcher("/Welcome");disp.forward(req,res);

                }
                else
                {       res.sendRedirect("loginSS.html");

                }
}
        public void doPost(HttpServletRequest req, HttpServletResponse res)throws   ServletException
,IOException
        {
                doGet(req,res);
        }
}
```

<u>Welcome.java</u> ServletPorgram:

```
import javax.servlet.http.*;import
javax.servlet.*; import java.io.*;

public class Welcome extends HttpServlet
{
        public void doGet(HttpServletRequest req, HttpServletResponse res)throws   ServletException
,IOException
        {
                PrintWriter out = res.getWriter(); String user =
                req.getParameter("user");out.println(" Welcome
                "+user);
        }
        public void doPost(HttpServletRequest req, HttpServletResponse res)throws   ServletException
,IOException
        {
                doGet(req,res);
        }
}
```

<u>Web.xml</u>:

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
   <servlet>
      <servlet-name>CheckUser</servlet-name>
      <servlet-class>CheckUser</servlet-class>
   </servlet>
```

```
    <servlet>
        <servlet-name>Welcome</servlet-name>
        <servlet-class>Welcome</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CheckUser</servlet-name>
        <url-pattern>/CheckUser/*</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Welcome</servlet-name>
        <url-pattern>/Welcome/*</url-pattern>
    </servlet-mapping>
</web-app>
```

6. Explain Servlet SessionMangement Mechanisms?

The Hypertext Transfer Protocol (HTTP) is by design a stateless protocol. To build effective web applications, it is imperative that requests from a particular client be associated with each other. Many strategies for session tracking have evolved over time, but all are difficult or troublesome for the programmer to use directly.

This specification defines a simple HttpSession interface that allows a servlet container to use any of several approaches to track a user's session without involving the Application Developer in the nuances of any one approach.

Session Tracking Mechanisms

The following sections describe approaches to tracking a user's sessions

Cookies

Session tracking through HTTP cookies is the most used session tracking mechanism and is required to be supported by all servlet containers.

The container sends a cookie to the client. The client will then return the cookie on each subsequent request to the server, unambiguously associating the request with a session. The name of the session tracking cookie must be JSESSIONID.

URL Rewriting

URL rewriting is the lowest common denominator of session tracking. When a client will not accept a cookie, URL rewriting may be used by the server as the basis for session tracking. URL rewriting involves adding data, a session id, to the URL path that is interpreted by the container to associate the request with a session.

The session id must be encoded as a path parameter in the URL string. The name of the parameter must be jsessionid. Here is an example of a URL containing encoded path information:

http://www.myserver.com/catalog/index.html;jsessionid=1234

Creating a Session

A session is considered “new” when it is only a prospective session and has not been established. Because HTTP is a request-response based protocol, an HTTP session is considered to be new until a client “joins” it. A client joins a session when session tracking information has been returned to the server indicating that a session has been established. Until the client joins a session, it cannot be assumed that the next request from the client will be recognized as part of a session.

The session is considered “new” if either of the following is true:

- The client does not yet know about the session
- The client chooses not to join a session.

These conditions define the situation where the servlet container has no mechanism by which to associate a request with a previous request.

A Servlet Developer must design his application to handle a situation where a client has not, cannot, or will not join a session.

Session Timeouts

In the HTTP protocol, there is no explicit termination signal when a client is no longer active. This means that the only mechanism that can be used to indicate when a client is no longer active is a timeout period.

The default timeout period for sessions is defined by the servlet container and can be obtained via the getMaxInactiveInterval method of the HttpSession interface. This timeout can be changed by the Developer using the setMaxInactiveInterval method of the HttpSession interface. The timeout periods used by these methods are defined in seconds. By definition, if the timeout period for a session is set to -1, the session willnever expire.

Distributed Environments

Within an application marked as distributable, all requests that are part of a session must handled by one virtual machine at a time. The container must be able to handle all objects placed into instances of the HttpSession class using the setAttribute or putValue methods appropriately. The following restrictions are imposed to meet these conditions:

- The container must accept objects that implement the Serializable interface
- The container may choose to support storage of other designated objects in the HttpSession, such as references to Enterprise JavaBean components and transactions.
- Migration of sessions will be handled by container-specific facilities.

The servlet container may throw an IllegalArgumentException if an object is placed into the session that is not Serializable or for which specific support has not been made available. The IllegalArgumentException must be thrown for objects where the container cannot support the mechanism necessary for migration of a session storing them.

These restrictions mean that the Developer is ensured that there are no additional concurrency issues beyond those encountered in a non-distributed container.

Example program for Session Class Object.

```
import java.io.*;

import java.text.*;

import java.util.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class SessionExample extends HttpServlet { ResourceBundle rb =
   ResourceBundle.getBundle("LocalStrings");
        public          void        doGet(HttpServletRequest                  request,            HttpServletResponse
response)
         throws IOException, ServletException {
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      out.println("<html>"); out.println("<body
      bgcolor=\"white\">");out.println("<head>");

      String title = rb.getString("sessions.title");out.println("<title>" + title
      + "</title>"); out.println("</head>"); out.println("<body>");

      out.println("<a href=\"../sessions.html\">"); out.println("<img src=\"../images/code.gif\"
      height=24 "
            + "width=24 align=right border=0 alt=\"view code\"></a>");out.println("<a href=\"../index.html\">");
      out.println("<img src=\"../images/return.gif\" height=24 "
            + "width=24 align=right border=0 alt=\"return\"></a>");out.println("<h3>" + title + "</h3>");

      HttpSession session = request.getSession(true); out.println(rb.getString("sessions.id") + " " +
      session.getId());out.println("<br>"); out.println(rb.getString("sessions.created") + " ");
      out.println(new Date(session.getCreationTime()) + "<br>");
      out.println(rb.getString("sessions.lastaccessed") + " "); out.println(new
      Date(session.getLastAccessedTime()));

      String dataName = request.getParameter("dataname"); String dataValue =
      request.getParameter("datavalue");if (dataName != null && dataValue != null)
      {
         session.setAttribute(dataName, dataValue);
      }
      out.println("<P>"); out.println(rb.getString("sessions.data") + "<br>");
      Enumeration names = session.getAttributeNames(); while
      (names.hasMoreElements()) {

         String name = (String) names.nextElement();
         String value = session.getAttribute(name).toString();out.println(HTMLFilter.filter(name) + " = "
               + HTMLFilter.filter(value) + "<br>");
      }
      out.println("<P>"); out.print("<form    action=\"");
      out.print(response.encodeURL("SessionExample"));out.print("\" ");
      out.println("method=POST>"); out.println(rb.getString("sessions.dataname"));
      out.println("<input type=text size=20 name=dataname>");out.println("<br>");
      out.println(rb.getString("sessions.datavalue")); out.println("<input type=text size=20
      name=datavalue>");out.println("<br>");
      out.println("<input type=submit>");
      out.println("</form>"); out.println("<P>GET based
      form:<br>");out.print("<form action=\"");
      out.print(response.encodeURL("SessionExample"));out.print("\" ");
      out.println("method=GET>"); out.println(rb.getString("sessions.dataname"));
      out.println("<input type=text size=20 name=dataname>");out.println("<br>");
      out.println(rb.getString("sessions.datavalue")); out.println("<input type=text size=20
      name=datavalue>");out.println("<br>");
      out.println("<input type=submit>");out.println("</form>"); out.print("<p><a href=\"");
      out.print(response.encodeURL("SessionExample?dataname=foo&datavalue=bar"));out.println("\" >URL
      encoded </a>");
      out.println("</body>");
      out.println("</html>");
      out.println("</body>");
      out.println("</html>");
   }

        public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException,
ServletException {
      doGet(request, response);
   }
}

final class HTMLFilter {
   public static String filter(String message) {if (message == null)
         return (null);
      char content[] = new char[message.length()]; message.getChars(0, message.length(),
      content, 0); StringBuffer result = new StringBuffer(content.length + 50);for (int i = 0; i <
      content.length; i++) {
         switch (content[i]) {
         case '<': result.append("&lt;");break;
         case '>': result.append("&gt;");break;
         case '&': result.append("&amp;");
            break;
         case '"': result.append("&quot;");break;
         default: result.append(content[i]);
         }
      }
      return (result.toString());
   }
}
```

Use cookie to save session data

```
import java.io.IOException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;

import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ShoppingCartViewerCookie extends HttpServlet {

     public void doGet(HttpServletRequest req, HttpServletResponse res) throwsServletException,
            IOException                                    {
        res.setContentType("text/html"); PrintWriter out =
        res.getWriter();

        String sessionid = null;
        Cookie[] cookies = req.getCookies(); if (cookies !=
        null) {
            for (int i = 0; i < cookies.length; i++) {
                if      (cookies[i].getName().equals("sessionid"))      {      sessionid      =
                    cookies[i].getValue();
                    break;
                }
            }
        }

        // If the session ID wasn't sent, generate one.
        // Then be sure to send it to the client with the response.


        if    (sessionid    ==    null)    {    sessionid    =
            generateSessionId();
            Cookie c = new Cookie("sessionid", sessionid);res.addCookie(c);
        }

        out.println("<HEAD><TITLE>Current Shopping Cart Items</TITLE></HEAD>");out.println("<BODY>");

        // Cart items are associated with the session ID String[] items =
        getItemsFromCart(sessionid);

        // Print the current cart items.
        out.println("You currently have the following items in your cart:<BR>");if (items == null) {
            out.println("<B>None</B>");
        } else { out.println("<UL>");
            for (int i = 0; i < items.length; i++) {out.println("<LI>" +
                items[i]);
            }
            out.println("</UL>");
        }

        // Ask if they want to add more items or check out. out.println("<FORM
        ACTION=\"/servlet/ShoppingCart\" METHOD=POST>");out.println("Would you like to<BR>");
        out.println("<INPUT TYPE=SUBMIT VALUE=\" Add More Items \">");out.println("<INPUT TYPE=SUBMIT
        VALUE=\" Check Out \">"); out.println("</FORM>");

        // Offer a help page.
        out.println("For help, click <A HREF=\"/servlet/Help"
                + "?topic=ShoppingCartViewerCookie\">here</A>");

        out.println("</BODY></HTML>");
    }

            private         static          String          generateSessionId()           throws
UnsupportedEncodingException {
        String uid = new java.rmi.server.UID().toString(); // guaranteed uniquereturn
        URLEncoder.encode(uid,"UTF-8"); // encode any special chars
    }

    private static String[] getItemsFromCart(String sessionid) {return new String[]{"a","b"};
    }
}
```

Use URL rewrite to save session data

```
import java.io.IOException;

import   java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import javax.servlet.ServletException;
import   javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import  javax.servlet.http.HttpServletResponse;

public class ShoppingCartViewerRewrite extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res) throwsServletException,
          IOException {
      res.setContentType("text/html"); PrintWriter out =
      res.getWriter();

      out.println("<HEAD><TITLE>Current Shopping Cart Items</TITLE></HEAD>");out.println("<BODY>");

      // Get the current session ID, or generate one if necessaryString sessionid =
      req.getPathInfo();
      if (sessionid == null) { sessionid =
          generateSessionId();
      }

      // Cart items are associated with the session IDString[] items =
      getItemsFromCart(sessionid);

      // Print the current cart items.
      out.println("You currently have the following items in your cart:<BR>");if (items == null) {
          out.println("<B>None</B>");
      } else { out.println("<UL>");
          for (int i = 0; i < items.length; i++) {out.println("<LI>" +
              items[i]);
          }
          out.println("</UL>");
      }

      // Ask if the user wants to add more items or check out.
      // Include the session ID in the action URL.
         out.println("<FORM  ACTION=\"/servlet/ShoppingCart/"  +  sessionid  +  "\"METHOD=POST>");
      out.println("Would you like to<BR>");
      out.println("<INPUT TYPE=SUBMIT VALUE=\" Add More Items \">");out.println("<INPUT TYPE=SUBMIT
      VALUE=\" Check Out \">"); out.println("</FORM>");

      // Offer a help page. Include the session ID in the URL. out.println("For help, click <A
      HREF=\"/servlet/Help/" + sessionid
              + "?topic=ShoppingCartViewerRewrite\">here</A>");

      out.println("</BODY></HTML>");
    }

              private              static              String              generateSessionId()                    throws
UnsupportedEncodingException {
      String uid = new java.rmi.server.UID().toString(); // guaranteed unique


      return URLEncoder.encode(uid, "UTF-8"); // encode any special chars
    }

    private static String[] getItemsFromCart(String sessionid) {return new String[] { "a", "b" };
    }
}
```

Use hidden fields to save session data

```
import java.io.IOException;
import   java.io.PrintWriter;
import  javax.servlet.ServletException;
import  javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import  javax.servlet.http.HttpServletResponse;

public class ShoppingCartViewerHidden extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
                                                            throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<HEAD><TITLE>Current Shopping Cart Items</TITLE></HEAD>"
);
        out.println("<BODY>");

        // Cart items are passed in as the item parameter.String[] items =
        req.getParameterValues("item");

        // Print the current cart items.
        out.println("You currently have the following items in your cart:<BR>
");
        if (items == null) { out.println("<B>None</B>");
        }
        else {
            out.println("<UL>");
            for (int i = 0; i < items.length; i++) {out.println("<LI>"
                + items[i]);
            }
            out.println("</UL>");
        }

        // Ask if the user wants to add more items or check out.
        // Include the current items as hidden fields so they'll be passed on
.
        out.println("<FORM  ACTION=\"/servlet/ShoppingCart\"    METHOD=POST>");if (items !=
        null) {
            for (int i = 0; i < items.length; i++) { out.println("<INPUT TYPE=HIDDEN
                NAME=\"item\" VALUE=\"" +
                    items[i] + "\">");
            }
        }
        out.println("Would you like to<BR>");
        out.println("<INPUT TYPE=SUBMIT VALUE=\" Add More Items \">");
        out.println("<INPUT TYPE=SUBMIT VALUE=\" Check Out \">");
        out.println("</FORM>");

        out.println("</BODY></HTML>");
    }
}
```

PROCESS OF EXECUTING THE SERVLET AND CONNECTING TO A DATABASE USING JDBC

Steps to Create Servlet Application using tomcat server

To create a Servlet application you need to follow the below mentioned steps. These steps are common for all the Web server. In our example we are using Apache Tomcat server. Apache Tomcat is an open source web server for testing servlets and JSP technology. Download latest version of Tomcat Server and install it on your machine.

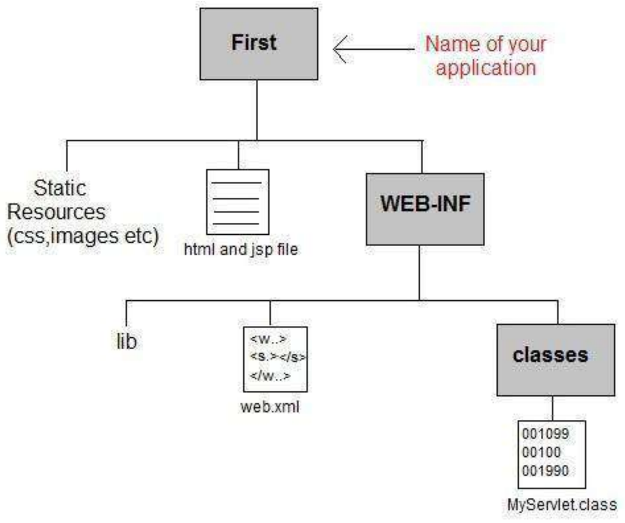
After installing Tomcat Server on your machine follow the below mentioned steps :

1. Create directory structure for your application.
2. Create a Servlet
3. Compile the Servlet
4. Create Deployement Descriptor for your application
5. Start the server and deploy the application

All these 5 steps are explained in details below, lets create our first Servlet Application.

1. Creating the Directory Structure

Sun Microsystem defines a unique directory structure that must be followed to create a servlet application.



Create the above directory structure inside **Apache-Tomcat\webapps** directory. All HTML, static files(images, css etc) are kept directly under **Web application** folder. While all the Servlet classes are kept inside classes folder.

The web.xml (deployement descriptor) file is kept under WEB-INF folder.

2. **Creating a Servlet**

There are three different ways to create a servlet.

- By implementing **Servlet** interface
- By extending **GenericServlet** class
- By extending **HttpServlet** class

But mostly a servlet is created by extending **HttpServlet** abstract class. As discussed earlier **HttpServlet** gives the definition of service() method of the **Servlet** interface. The servlet class that we will create should not override service() method. Our servlet class will override only doGet() or doPost() method.

When a request comes in for the servlet, the Web Container calls the servlet's service() method and depending on the type of request the service() method calls either the doGet() or doPost() method.

NOTE: By default a request is **Get** request.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public MyServlet extends HttpServlet
{
        public void doGet(HttpServletRequest request,HttpServletResposne response)
                   throws ServletException {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                out.println("<html><body>");
                out.println("<h1>Hello Readers</h1>");
                out.println("</body></html>");
        }
}
```

Write above code in a notepad file and save it as **MyServlet.java** anywhere on your PC.

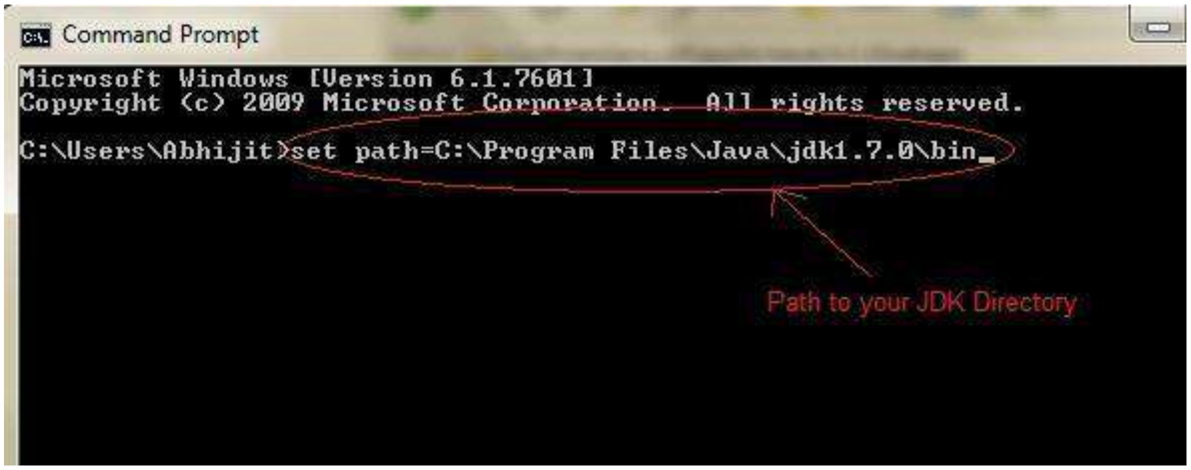
Compile it(explained in next step) from there and paste the class file into WEB-INF/classes/ directory that you have to create inside **Tomcat/webapps** directory.

3. Compiling a Servlet

To compile a Servlet a JAR file is required. Different servers require different JAR files. In Apache Tomcat server servlet-api.jar file is required to compile a servlet class.

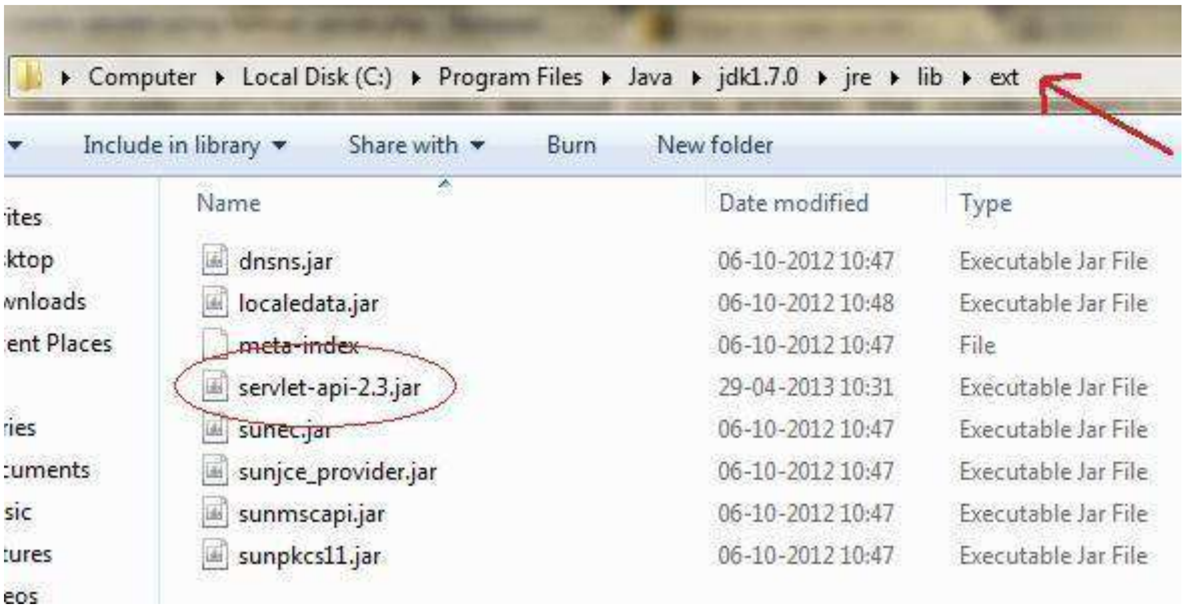
Steps to compile a Servlet:

- Set the Class Path.



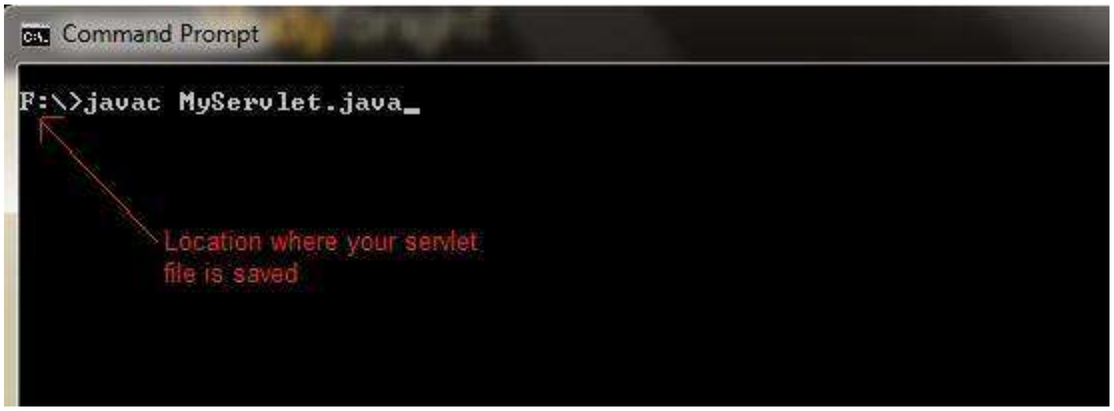
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Abhijit>set path=C:\Program Files\Java\jdk1.7.0\bin_
```

- Download **servlet-api.jar** file.
- Paste the servlet-api.jar file inside Java\jdk\jre\lib\ext directory.



Name	Date modified	Type
dnsns.jar	06-10-2012 10:47	Executable Jar File
localedata.jar	06-10-2012 10:48	Executable Jar File
meta-index	06-10-2012 10:47	File
servlet-api-2.3.jar	29-04-2013 10:31	Executable Jar File
sunec.jar	06-10-2012 10:47	Executable Jar File
sunjce_provider.jar	06-10-2012 10:47	Executable Jar File
sunmscapi.jar	06-10-2012 10:47	Executable Jar File
sunpkcs11.jar	06-10-2012 10:47	Executable Jar File

- Compile the Servlet class.



```
F:\>javac MyServlet.java_
```

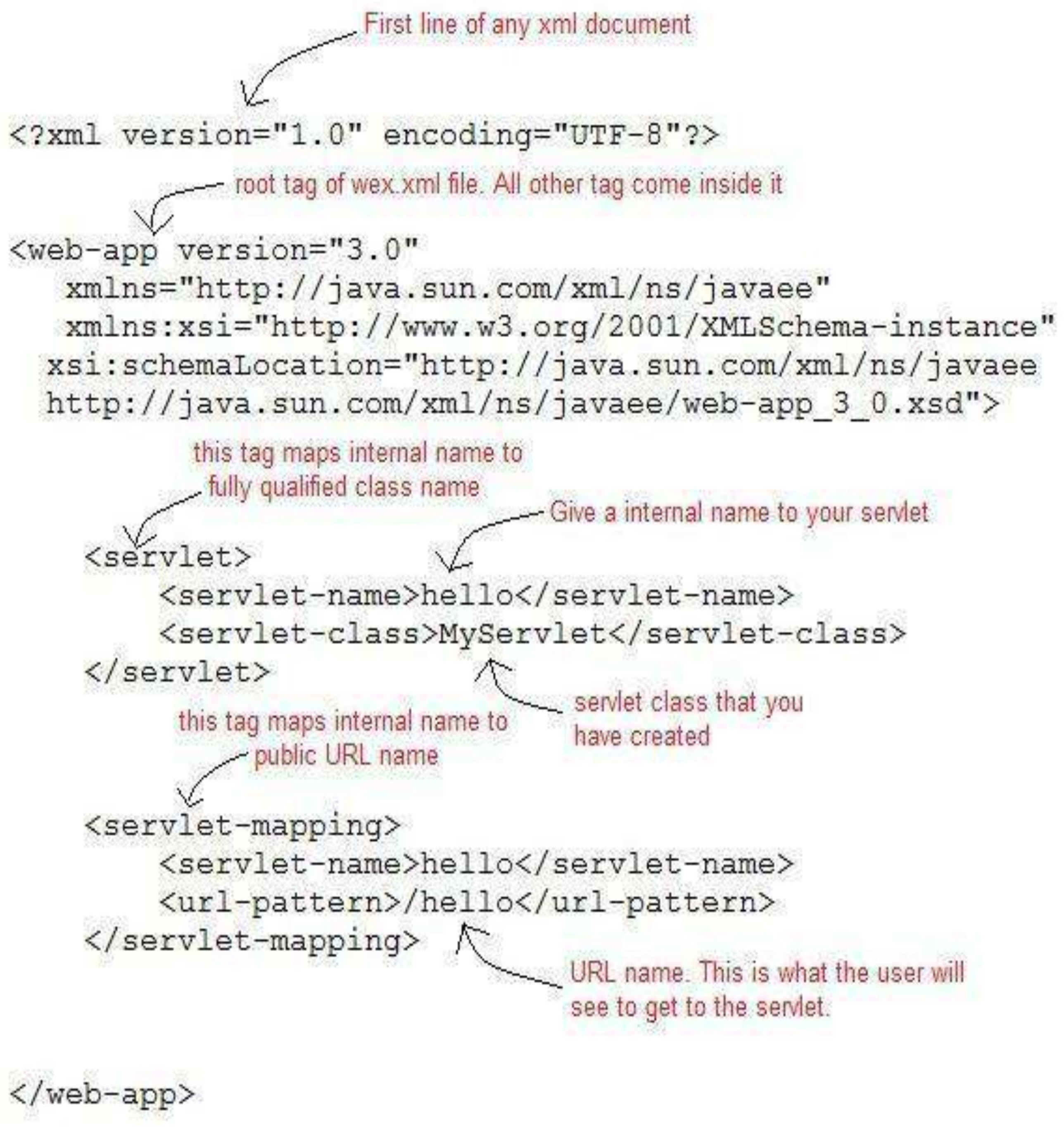
OTE: After compiling your Servlet class you will have to paste the class file into WEB-INF/classes/ directory.

4. Create Deployment Descriptor

Deployment Descriptor(DD) is an XML document that is used by Web Container to run Servlets and JSP pages. DD is used for several important purposes such as:

- Mapping URL to Servlet class.
- Initializing parameters.
- Defining Error page.
- Security roles.
- Declaring tag libraries.

We will discuss about all these in details later. Now we will see how to create a simple **web.xml** file for our web application.



```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="3.0"
   xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>MyServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>

</web-app>
```

First line of any xml document
root tag of wex.xml file. All other tag come inside it
this tag maps internal name to fully qualified class name
Give a internal name to your servlet
servlet class that you have created
this tag maps internal name to public URL name
URL name. This is what the user will see to get to the servlet.

5. Start the Server

Double click on the **startup.bat** file to start your Apache Tomcat Server.

Or, execute the following command on your windows machine using RUN prompt.

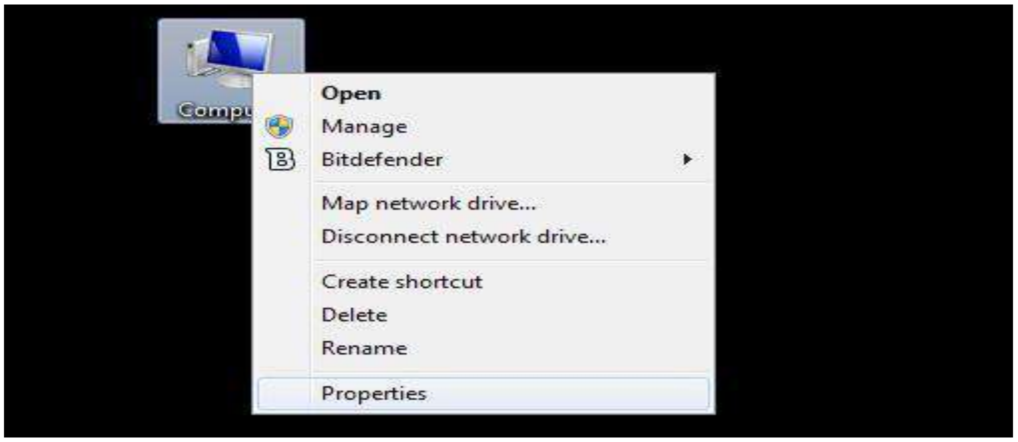
```
C:\apache-tomcat-7.0.14\bin\startup.bat
```

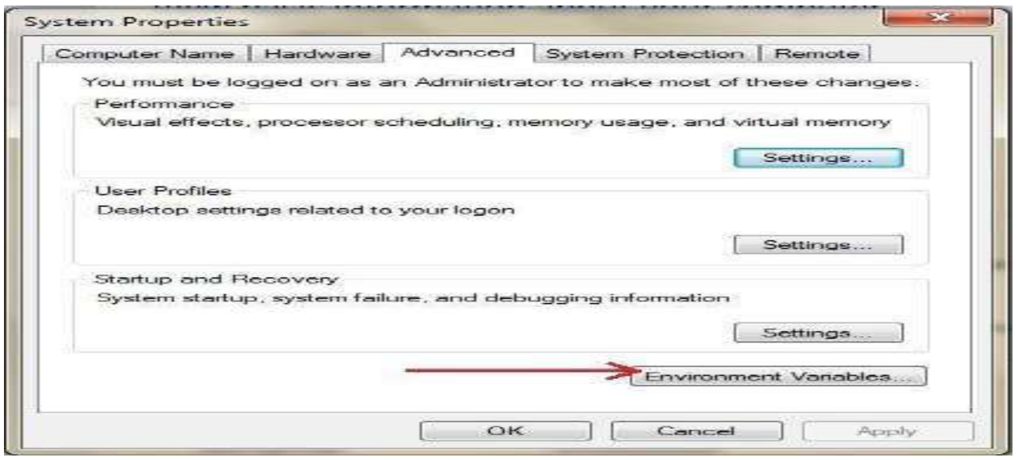
6. Starting Tomcat Server for the first time

If you are starting Tomcat Server for the first time you need to set JAVA_HOME in the Enviroment variable. The following steps will show you how to set it.

- Right Click on **My Computer**, go to **Properites**.

- Go to **Advanced** Tab and Click on **Enviroment Variables...** button.
  \





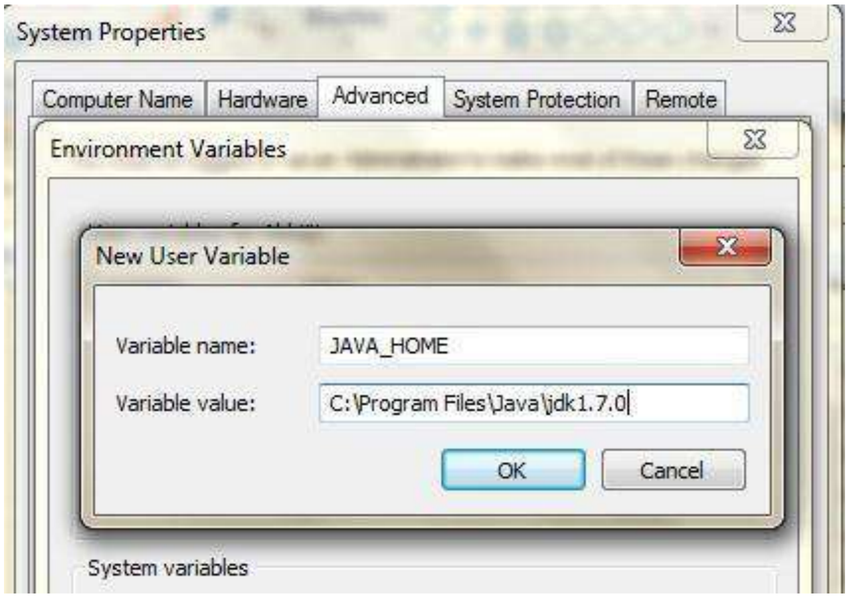
You must be logged on as an Administrator to make most of these changes.

Visual effects, processor scheduling, memory usage, and virtual memory

Desktop settings related to your logon

System startup, system failure, and debugging information

- Click on **New** button, and enter **JAVA_HOME** inside Variable name text field and path of JDK inside Variable value text field. Click OK to save.

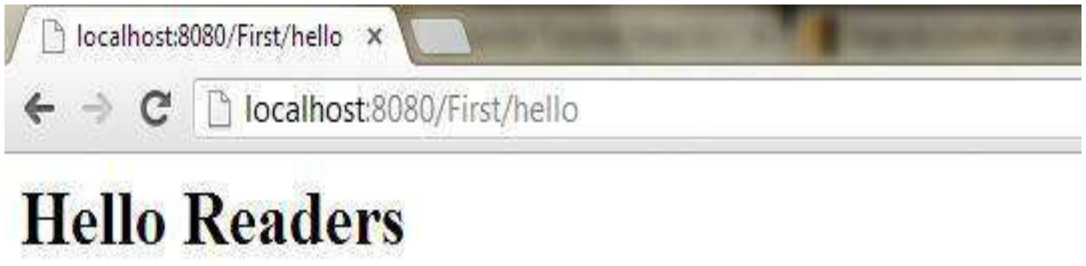


Variable name: JAVA_HOME

Variable value: C:\Program Files\Java\jdk1.7.0

7. **Run Servlet Application**

Open Browser and type **http:localhost:8080/First/hello**



Hello Readers

Hurray! Our first Servlet Application ran successfully.

Compiled from:
https://www.studytonight.com/servlet/steps-to-create-servlet-using-tomcat-server.php

ESTABLISHING JDBC CONNECTION

Before establishing a connection between front end i.e your Java Program and back end i.e the database we should learn what precisely a JDBC is and why it came to existence.

What is JDBC?
JDBC is an acronym for Java Database Connectivity. It’s an advancement for ODBC (Open Database Connectivity). JDBC is a standard API specification developed in order to move data

from frontend to backend. This API consists of classes and interfaces written in Java. It basically acts as an interface (not the one we use in Java) or channel between your Java program and databases i.e it establishes a link between the two so that a programmer could send data from Java code and store it in the database for future use.

Why JDBC came into existence?
As previously told JDBC is an advancement for ODBC, ODBC being platform dependent had a lot of drawbacks. ODBC API was written in C,C++, Python, Core Java and as we know above languages (except Java and some part of Python)are platform dependent . Therefore to remove dependence, JDBC was developed by database vendor which consisted of classes and interfaces written in Java.

STEPS FOR DATABASE CONNECTIVITY IN JAVA

1. Loading the Driver

To begin with, you first need load the driver or register it before using it in the program. Registration is to be done once in your program. You can register a driver in one of two ways mentioned below:

- **Class.forName() :** Here we load the driver's class file into memory at the runtime. No need of using new or creation of object .The following example uses Class.forName() to load the Oracle driver –

  ```
  Class.forName("oracle.jdbc.driver.OracleDriver");
  ```

- **DriverManager.registerDriver():** DriverManager is a Java inbuilt class with a static member register. Here we call the constructor of the driver class at compile time . The following example uses DriverManager.registerDriver()to register the Oracle driver –

  ```
  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())
  ```

2. Create the connections

After loading the driver, establish connections using :

```
Connection con = DriverManager.getConnection(url,user,password)
```

user – username from which your sql command prompt can be accessed.
password – password from which your sql command prompt can be accessed.
con: is a reference to Connection interface.
url : Uniform Resource Locator. It can be created as follows:

```
String url = " jdbc:oracle:thin:@localhost:1521:xe"
```

Where oracle is the database used, thin is the driver used, @localhost is the IP Address where database is stored, 1521 is the "**port number**" and "**xe"** is the service provider. All 3 parameters above are of String type and are to be declared by programmer before calling the function. Use of this can be referred from final code.

3. Create a statement

Once a connection is established you can interact with the database. The JDBCStatement, CallableStatement, and PreparedStatement interfaces define the methods that enable you to send SQL commands and receive data from your database.

Use of JDBC Statement is as follows:

```
Statement st = con.createStatement();
```

Here, con is a reference to Connection interface used in previous step .

4. Execute the query

Now comes the most important part i.e., executing the query. Query here is an SQL Query. Now Some of methods are as follows:

- Query for updating / inserting table in a database.
- Query for retrieving data

The executeQuery() method of Statement interface is used to execute queries of retrieving values from the database. This method returns the object of ResultSet that can be used to get all the records of a table.

The executeUpdate(sql query) method ofStatement interface is used to execute queries of updating/inserting .

Example:

```
int m = st.executeUpdate(sql);

if (m==1)

   System.out.println("inserted successfully : "+sql);

else

   System.out.println("insertion failed");
```

Here sql is sql query of the type String

5. Close the connections

So finally we have sent the data to the specified location and now we are at the verge of completion of our task.

By closing connection, objects of Statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

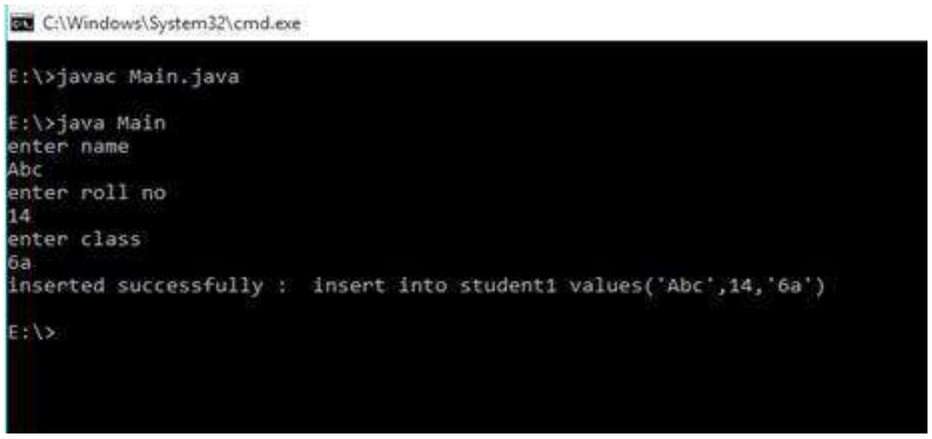
Example :

```
 con.close();
```

Implementation

```
Import          java.sql.*;
import java.util.*;

class Main
{
        public static void main(String a[])
        {
                //Creating the connection
                String url = "jdbc:oracle:thin:@localhost:1521:xe";
                String user = "system";
                String pass = "12345";

                //Entering the data
                Scanner k = new Scanner(System.in);

                System.out.println("enter name");
                String name = k.next();

                System.out.println("enter roll no");
                int roll = k.nextInt();

                System.out.println("enter class");
                String cls = k.next();

                //Inserting data using SQL query
                String sql = "insert into student1 values('"+name+"',"+roll+",'"+cls+"')";
                Connection con=null;

                try
                {
                        DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

                        //Reference to connection interface
                        con = DriverManager.getConnection(url,user,pass);

                        Statement st = con.createStatement();

                        int m = st.executeUpdate(sql);

                        if (m == 1)
                                System.out.println("inserted successfully : " + sql);
                        else
                                System.out.println("insertion failed");

                        con.close();
                }
                catch(Exception ex)
                {
                        System.err.println(ex);
                }
        }
}
```

Output:



```
C:\Windows\System32\cmd.exe

E:\>javac Main.java

E:\>java Main
enter name
Abc
enter roll no
14
enter class
6a
inserted successfully :  insert into student1 values('Abc',14,'6a')

E:\>
```

JAVA SERVLET AND JDBC EXAMPLE | INSERT DATA IN MYSQL

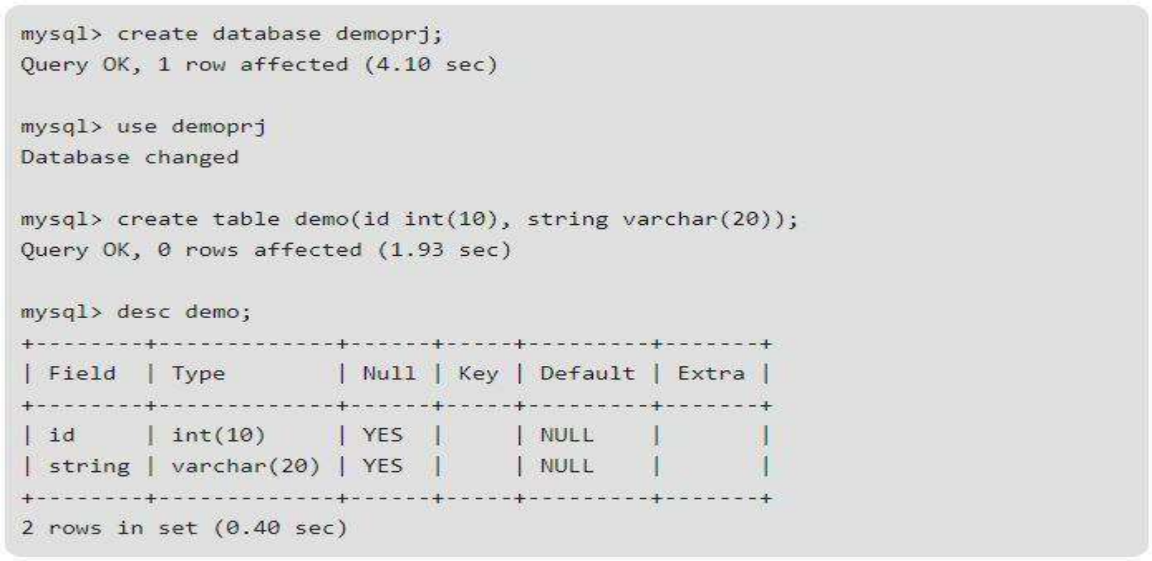
To start with interfacing Java Servlet Program with JDBC Connection:

1. Proper JDBC Environment should set-up along with database creation.
2. To do so, download the mysql-connector.jar file from the internet,
3. As it is downloaded, move the jar file to the apache-tomcat server folder,
4. Place the file in **lib** folder present in the apache-tomcat directory.

5. **To start with the basic concept of interfacing:**

Step 1: Creation of Database and Table in MySQL

As soon as jar file is placed in the folder, create a database and table in MySQL,

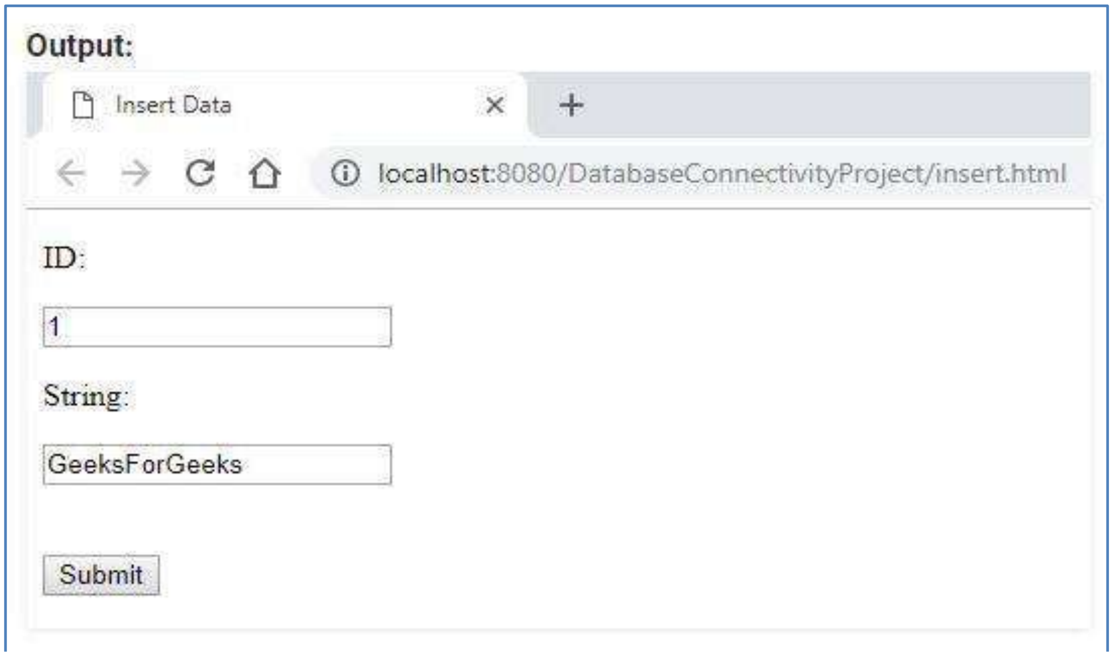


```
mysql> create database demoprj;
Query OK, 1 row affected (4.10 sec)

mysql> use demoprj
Database changed

mysql> create table demo(id int(10), string varchar(20));
Query OK, 0 rows affected (1.93 sec)

mysql> desc demo;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int(10)     | YES  |     | NULL    |       |
| string | varchar(20) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
2 rows in set (0.40 sec)
```

Step 2: Implementation of required Web-pages

Create a form in HTML file, where take all the inputs required to insert data into the database. Specify the servlet name in it, with the POST method as security is important aspects in database connectivity.

```
<!DOCTYPE html>
<html>
<head>
<title>Insert Data</title>
</head>
<body>
        <!-- Give Servlet reference to the form as an instances
        GET and POST services can be according to the problem statement-->
        <form action="./InsertData" method="post">
                <p>ID:</p>
                <!-- Create an element with mandatory name attribute,
                so that data can be transfer to the servlet using getParameter() -->
                <input type="text" name="id"/>
                <br/>
                <p>String:</p>
                <input type="text" name="string"/>
                <br/><br/><br/>
                <input type="submit"/>
        </form>
</body>
</html>
```


Output:

ID:

1

String:

GeeksForGeeks

Submit the data (with validation) as all the required data are inserted.

Step 3: Creation of Java Servlet program with JDBC Connection

To create a JDBC Connection steps are

1. Import all the packages
2. Register the JDBC Driver
3. Open a connection
4. Execute the query, and retrieve the result
5. Clean up the JDBC Environment

Create a separate class to create a connection of database, as it is a lame process to writing the same code snippet in all the program. Create a .java file which returns a Connection object.

```
import          java.sql.Connection;
import     java.sql.DriverManager;
import java.sql.SQLException;

// This class can be used to initialize the database connection
public class DatabaseConnection {
        protected static Connection initializeDatabase()
                throws SQLException, ClassNotFoundException
        {
                // Initialize all the information regarding
                // Database Connection
                String   dbDriver   =   "com.mysql.jdbc.Driver";
                String dbURL = "jdbc:mysql:// localhost:3306/";
                // Database name to access
```

```
		String dbName = "demoprj";
		String dbUsername = "root";
		String dbPassword = "root";

		Class.forName(dbDriver);
		Connection con = DriverManager.getConnection(dbURL + dbName,

	dbUsername,

	dbPassword);
		return con;
	}
}
```

Step 4: To use this class method, create an object in Java Servlet program

Below program shows Servlet Class which create a connection and insert the data in the **demo** table,

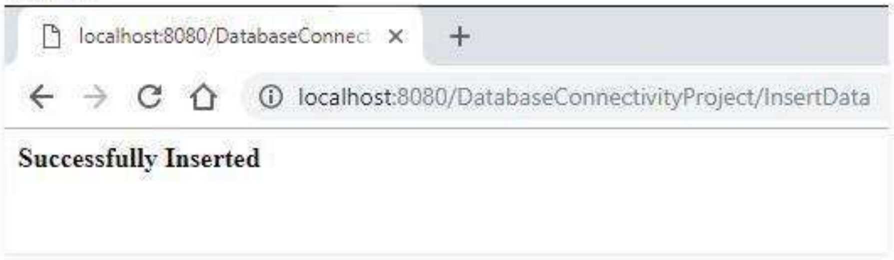
```
import java.io.IOException;
import   java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;

import          javax.servlet.ServletException;
import   javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import    javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

// Import Database Connection Class file
import code.DatabaseConnection;

//           Servlet            Name
@WebServlet("/InsertData")
public class InsertData extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
		throws ServletException, IOException
	{
		try {

			// Initialize the database
			Connection con = DatabaseConnection.initializeDatabase();
```

```
                        // Create a SQL query to insert data into demo table
                        // demo table consists of two columns, so two '?' is used
                        PreparedStatement st = con
                                .prepareStatement("insert into demo values(?, ?)");

                        // For the first parameter,
                        // get the data using request object
                        // sets the data to st pointer
                        st.setInt(1, Integer.valueOf(request.getParameter("id")));

                        // Same for second parameter
                        st.setString(2, request.getParameter("string"));

                        // Execute the insert command using executeUpdate()
                        // to make changes in database
                        st.executeUpdate();

                        // Close all the connections
                        st.close();
                        con.close();

                        // Get a writer pointer
                        // to display the successful result
                        PrintWriter out = response.getWriter();
                        out.println("<html><body><b>Successfully Inserted"
                                    + "</b></body></html>");
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Step 5: Get the data from the HTML file

To get the data from the HTML file, the request object is used which calls getParameter() Method to fetch the data from the channel. After successful insertion, the writer object is created to display a success message.
After insertion operation from Servlet, data will be reflected in MySQL Database

Output:


localhost:8080/DatabaseConnect
localhost:8080/DatabaseConnectivityProject/InsertData

Successfully Inserted

Result in MySQL Interface

```
mysql> select * from demo;
+------+---------------+
| id   | string        |
+------+---------------+
|    1 | GeeksForGeeks |
+------+---------------+
1 row in set (0.06 sec)
```

Compiled from:
https://www.geeksforgeeks.org/establishing-jdbc-connection-in-java/
https://www.geeksforgeeks.org/java-servlet-and-jdbc-example-insert-data-in-mysql/

UNIT – IV

Introduction to JSP: The Anatomy of a JSP Page, JSP Processing, Declarations, Directives, Expressions, Code Snippets, implicit objects, Using Beans in JSP Pages, Using Cookies and session for session tracking, connecting to database in JSP.

Introduction to Java Server Pages

JSP is a server side technology that does all the processing at server. It is used for creating dynamic web applications, using java as programming language.

Basically, any html file can be converted to JSP file by just changing the file extension from ".html" to ".jsp", it would run just fine. What differentiates JSP from HTML is the ability to use java code inside HTML. In JSP, you can embed Java code in HTML using JSP tags. for e.g. run the code below, every time you run this, it would display the current time. That is what makes this code dynamic.

```
<HTML>
<BODY>
        Hello BeginnersBook Readers!
        Current time is: <%= new java.util.Date() %>
</BODY>
</HTML>
```

Your First JSP

Let's start learning JSP with a **simple JSP**.

```
<%-- JSP comment --%>
<HTML>
        <HEAD>
                    <TITLE>MESSAGE</TITLE>
        </HEAD>
        <BODY>
                    <%out.print("Hello, Sample JSP code");%>
        </BODY>
</HTML>
```

The above JSP generates the following output:

Hello, Sample JSP code.

Explanation of above code

1) The line **<%–JSP Comment–%>** represents the JSP element called JSP Comment, While adding comments to a JSP page you can use this tag, we will discuss this in detail in coming posts. **Note:** JSP Comments must starts with a tag **<%–** and ends with **–%>**

2) Head, Title and Body tags are HTML tags – They are HTML tags, frequently used for static web pages. Whatever content they have is delivered to client(Web browser) as such.

3) **<%out.print(“ Hello, Sample JSP code ”);%>** is a JSP element, which is known as Scriptlet. Scriptlets can contain Java codes. **syntax of scriptlet is:** <%Executable java code%>. As the code in Scriptlets is java statement, they must end with a semicolon(;). out.print(“ Hello, Sample JSP code ”) is a java statement, which prints“ Hello, Sample JSP code”.

As discussed, JSP is used for creating dynamic webpages. Dynamic webpages are usually a mix of static & dynamic content.

The **static content** can have text-based formats such as HTML, XML etc and the **dynamic content** is generated by JSP tags using java code inside HTML .

Servlet Vs JSP
Like JSP, Servlets are also used for generating dynamic webpages. Here is the comparison between them.
The major difference between them is that servlet adds HTML code inside java while JSP adds java code inside HTML. There are few other noticeable points that are as follows:

Servlets
1. Servlet is a Java program which supports HTML tags too.
2. Generally used for developing business layer(the complex computational code) of an enterprise application.
3. Servlets are created and maintained by Java developers.

JSP
1. JSP program is a HTML code which supports java statements too. To be more precise, JSP embed java in html using JSP tags.
2. Used for developing presentation layer of an enterprise application
3. Frequently used for designing websites and used by web developers.

Advantages of JSP
1. JSP has all the advantages of servlet, like: Better performance than CGI Built in session features, it also inherits the features of java technology like – multithreading, exception handling, Database connectivity, etc.
2. JSP enables the separation of content generation from content presentation that makes it more flexible.
3. With the JSP, it is now easy for web designers to show case the information what is needed.
4. Web Application Programmers can concentrate on how to process/build the information.

Extension to Servlet
- JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP that makes JSP development easy.

Easy to maintain
- JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

Fast Development: No need to recompile and redeploy

- If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

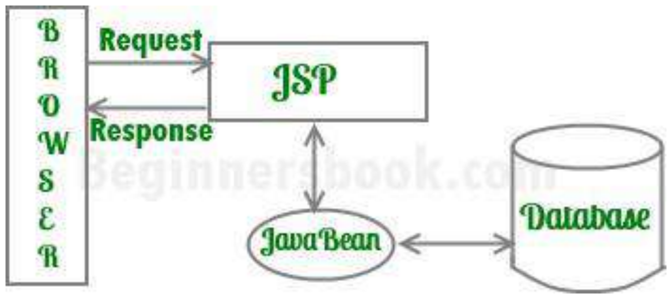
Less code than Servlet

- In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

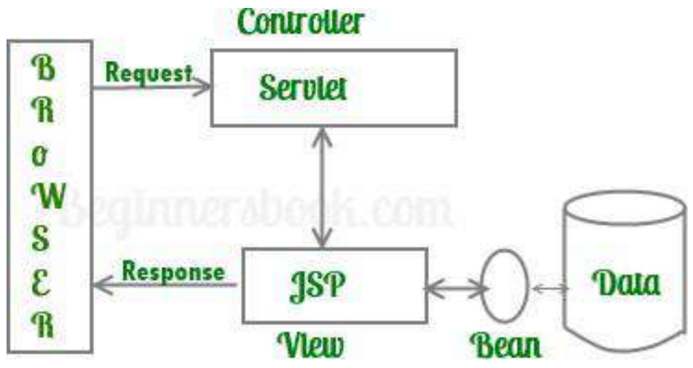
Architecture of a JSP Application

Before we start developing web application, we should have a basic idea of architectures. Based on the location where request processing happens (Servlet OR JSP (java server pages)) there are two architectures for JSP. They are – Model1 Architecture & Model2 Architecture.

1) Model1 Architecture: In this Model, JSP plays a key role and it is responsible for of processing the request made by client. Client (Web browser) makes a request; JSP then creates a bean object which then fulfils the request and passes the response to JSP. JSP then sends the response back to client. Unlike Model2 architecture, in this Model most of the processing is done by JSP itself.



2) Model2 Architecture: In this Model, Servlet plays a major role and it is responsible for processing the client's (web browser) request. Presentation part (GUI part) will be handled by JSP and it is done with the help of bean as shown in image below. The servlet acts as controller and in charge of request processing. It creates the bean objects if required by the JSP page and calls the respective JSP page. The JSP handles the presentation part by using the bean object. In this Model, JSP doesn't do any processing; Servlet creates the bean Object and calls the JSP program as per the request made by client.

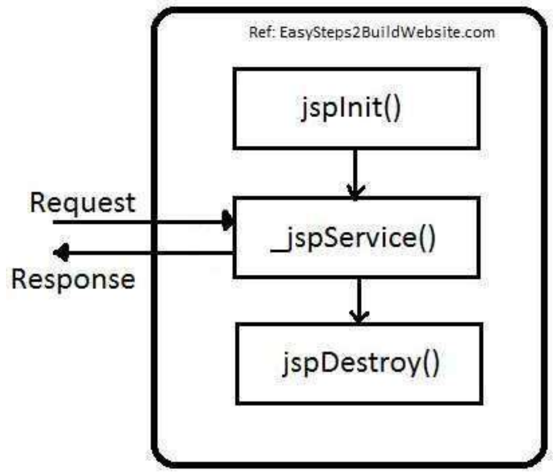


Java Server Pages (JSP) Life Cycle

JSP pages are saved with "**.jsp**" extension which lets the server know that this is a JSP page and needs to go through JSP life cycle stages. When client makes a request to Server, it first goes to container. Then container checks whether the servlet class is older than jsp page(To ensure that the JSP file got modified). If this is the case then container does the translation again (converts JSP to Servlet) otherwise it skips the translation phase (i.e. if JSP webpage is not modified then it doesn't do the translation to improve the performance as this phase takes time and to repeat this step every time is not time feasible)

The steps in the life cycle of jsp page are:

1. Translation
2. Compilation
3. Loading
4. Instantiation
5. Initialization
6. RequestProcessing
7. Destruction



Let see the Life cycle of JSP in more detail –
1) As stated above whenever container receives request from client, it does translation only when servlet class is older than JSP page otherwise it skips this phase (reason I explained above).
2) Then the container –

- compiles the corresponding servlet program
- Loads the corresponding servlet class
- Instantiates the servlet class
- Calls the **jspInit() method** to initialize the servlet instance(Jsp container will do this job only when the instance of servlet file is not running or if it is older than the jsp file.)

[code language="java"]
public void **jspInit()**
{
//code to intialize Servlet instances
}[/code]

3) A new thread is then gets created, which invokes the **_jspService() method**, with a request (HttpServletRequest) and response (HttpServletRespnse) objects as parameters -shown below.

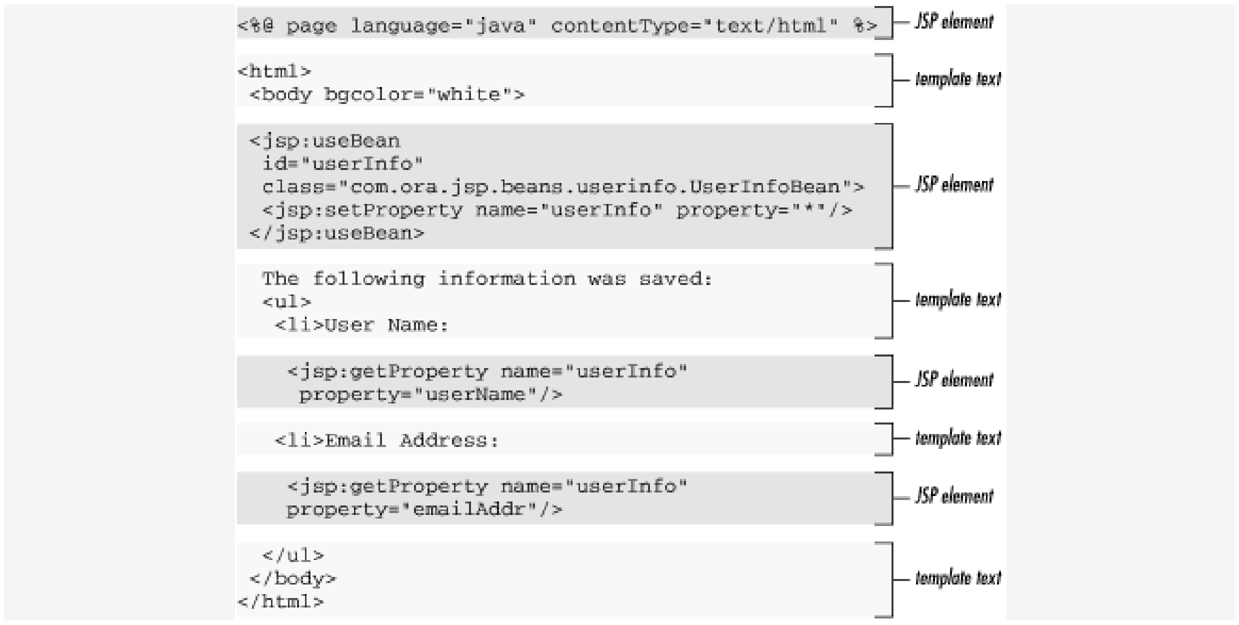
```
[code language="java"]
void _jspService( HttpServletRequest req, HttpServletResponse res)
{
//code goes here
}[/code]
```

4) Invokes the **jspDestroy() method** to destroy the instance of the servlet class. code will look like below –

```
[code language="java"]
public void jspDestory()
{
//code to remove the instances of servlet class
}[/code]
```

THE ANATOMY OF A JSP PAGE

A JSP page is simply a regular web page with JSP elements for generating the parts of the page that differ for each request, as shown in the following figure.



```
<%@ page language="java" contentType="text/html" %>

<html>
 <body bgcolor="white">

 <jsp:useBean
  id="userInfo"
  class="com.ora.jsp.beans.userinfo.UserInfoBean">
  <jsp:setProperty name="userInfo" property="*"/>
 </jsp:useBean>

  The following information was saved:
  <ul>
   <li>User Name:

    <jsp:getProperty name="userInfo"
     property="userName"/>

   <li>Email Address:

    <jsp:getProperty name="userInfo"
    property="emailAddr"/>

  </ul>
 </body>
</html>
```

Figure: Template text and JSP elements

Everything in the page that is not a JSP element is called template text . Template text can really be any text: HTML, WML, XML, or even plain text. Since HTML is by far the most common web page language in use today, most of the descriptions and examples in this book are HTML-based, but keep in mind that JSP has no dependency on HTML; it can be used with any markup language. Template text is always passed straight through to the browser.

When a JSP page request is processed, the template text and the dynamic content generated by the JSP elements are merged, and the result is sent as the response to the browser.

JSP Elements

There are three types of JSP elements you can use: *directive*, *action*, and *scripting*. A new construct added in JSP 2.0 is an Expression Language (EL) expression; let's call this a forth element type, even though it's a bit different than the other three.

Directive elements

The directive elements, shown in the following Table, specify information about the page itself that remains the same between requests—for example, if session tracking is required or not, buffering requirements, and the name of a page that should be used to report errors, if any.

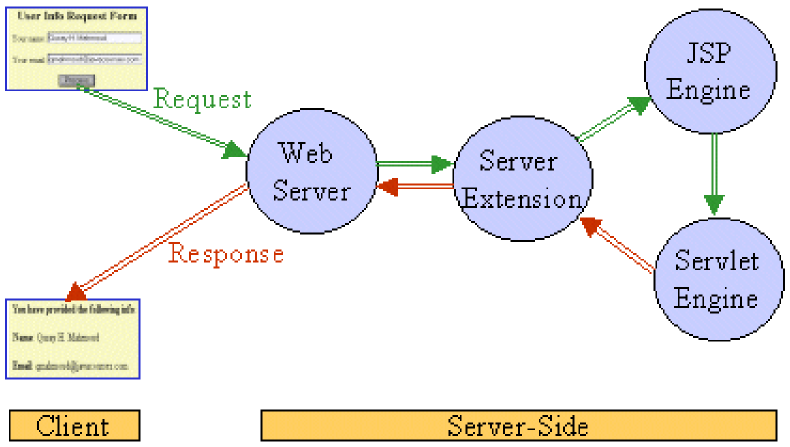
Table: Directive elements

Element	**Description**
<%@ page ... %>	Defines page-dependent attributes, such as session tracking, error page, and buffering requirements
<%@ include ... %>	Includes a file during the translation phase
<%@ taglib ... %>	Declares a tag library, containing custom actions, that is used in the page

JSP PROCESSING

The following shows the processing of JSP in detail.

1. The web server have JSP engine which acts as a container to process JSP pages.
2. All the requests for JSP Pages are intercepted by JSP Container.
3. JSP container along with web server provide the runtime environment to JSP



Please find the below steps that are required to process JSP Page:

1. Web browser sends an HTTP request to the web server requesting JSP page.

2. Web server recognizes that the HTTP request by web browser is for JSP page by checking the extension of the file (i.e .jsp)
3. Web server forwards HTTP Request to JSP engine.
4. JSP engine loads the JSP page from disk and converts it into a servlet
5. JSP engine then compiles the servlet into an executable class and forward original request to a servlet engine.
6. Servlet engine loads and executes the Servlet class.
7. Servlet produces an output in HTML format
8. Output produced by servlet engine is then passes to the web server inside an HTTP response.
9. Web server sends the HTTP response to Web browser in the form of static HTML content.
10. Web browser loads the static page into the browser and thus user can view the dynamically generated page.

JSP DECLARATION TAG

Declaration tag is a block of java code for declaring class wide variables, methods and classes. Whatever placed inside these tags gets initialized during JSP initialization phase and has class scope. JSP container keeps this code outside of the service method (_jspService()) to make them class level variables and methods.

As we know that variables can be initialized using scriptlet too but those declaration being placed inside _jspService() method which doesn't make them class wide declarations. On the other side, **declaration tag** can be used for defining class level variables, methods and classes.

Syntax of declaration tag:

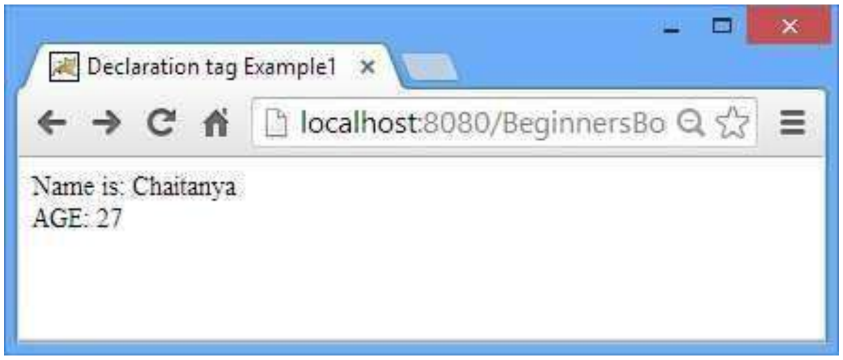
```
<%!  Declaration %>
```

Example 1: Variables declaration

In this example we have declared two variables inside declaration tag and displayed them on client using expression tag.

```
<html>
<head>
 <title>Declaration tag Example1</title>
</head>
<body>
<%! String name="Chaitanya"; %>
<%! int age=27; %>
<%= "Name is: "+ name %><br>
<%= "AGE: "+ age %>
</body>
</html>
```

Output:



Name is: Chaitanya
AGE: 27

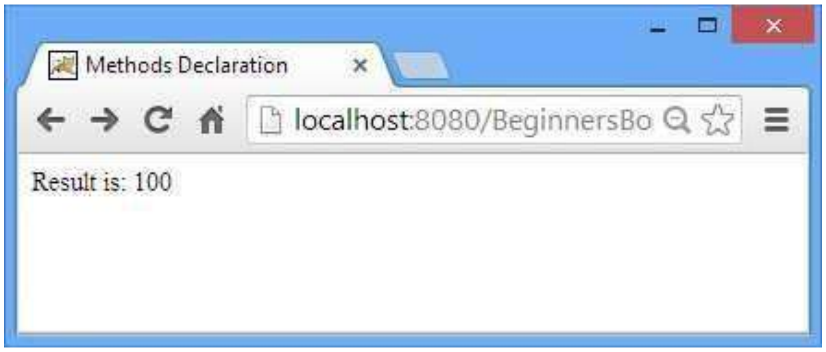
Example 2: Methods declaration

In this example we have declared a method **sum** using **JSP declaration tag**.

```
<html>
        <head>
          <title>Methods Declaration</title>
        </head>

        <body>
            <%!
            int sum(int num1, int num2, int num3)
           {
                     return num1+num2+num3;
            }
            %>

            <%= "Result is: " + sum(10,40,50) %>
        </body>
</html>
```

Output:

Sum of all three integers gets displayed on the browser.



Result is: 100

JSP EXPRESSION TAG

Expression tag evaluates the expression placed in it, converts the result into String and send the result back to the client through response object. Basically it writes the result to the client(browser).

Syntax of expression tag in JSP:

```
<%= expression %>
```

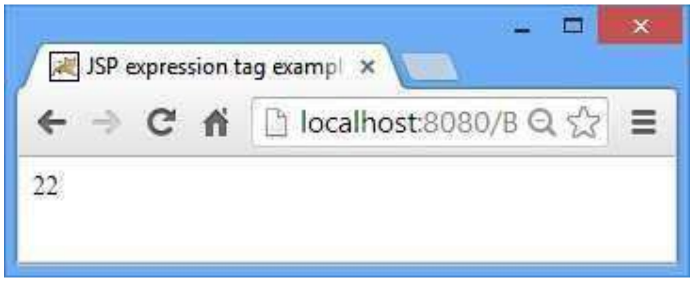
JSP expression tag Examples

Example 1: Expression of values

Here we are simply passing the expression of values inside expression tag.

```
<html>
        <head>
                <title>JSP expression tag example1</title>
        </head>
        <body>
                <%= 2+4*5 %>
        </body>
 </html>
```

Output:



```
22
```

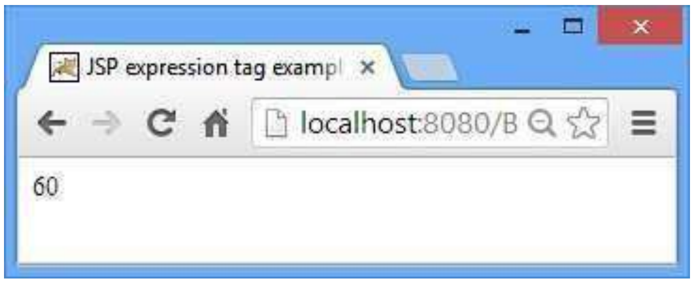
Example 2: Expression of variables

In this example we have initialized few variables and passed the expression of variables in the expression tag for result evaluation.

```
<html>
<head>
  <title>JSP expression tag example2</title>
</head>
<body>
 <%
 int a=10;
 int b=20;
 int c=30;
 %>
 <%= a+b+c %>
```

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

Output:



```
60
```

Example 3: String and implicit object output

In this example we are setting up an attribute using application implicit object and then displaying that attribute and a simple string on another JSP page using expression tag.

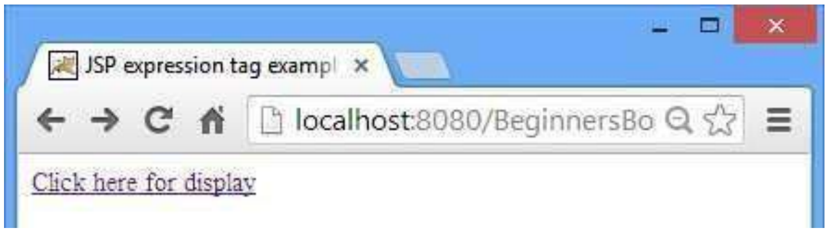
index.jsp

```
<html>
        <head>
                    <title> JSP expression tag example3 </title>
        </head>
        <body>
                     <% application.setAttribute("MyName", "Chaitanya"); %>
                     <a href="display.jsp">Click here for display</a>
        </body>
</html>
```

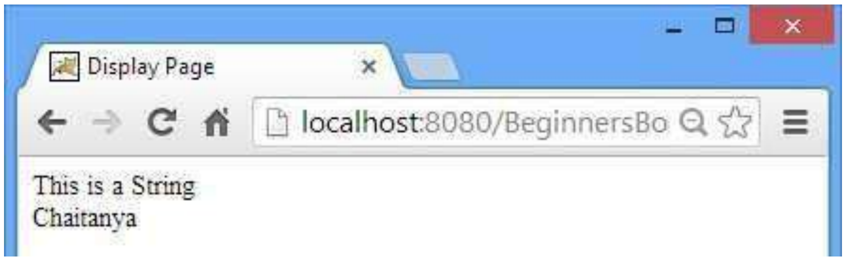
display.jsp

```
<html>
        <head>
                    <title>Display Page</title>
        </head>
        <body>
                     <%="This is a String" %><br>
                     <%= application.getAttribute("MyName") %>
        </body>
</html>
```

Output:



Click here for display



```
This is a String
Chaitanya
```

JSP DIRECTIVES – PAGE, INCLUDE AND TAG LIB

Directives control the processing of an entire JSP page. It gives directions to the server regarding processing of a page.

Syntax of Directives:

```
<%@ directive name [attribute name="value" attribute name="value"...........]%>
```

There are three types of Directives in JSP:

1) Page Directive
2) Include Directive
3) TagLib Directive

1) Page Directive

There are several attributes, which are used along with Page Directives and these are –

1. import
2. session
3. isErrorPage
4. errorPage
5. ContentType
6. isThreadSafe
7. extends
8. info
9. language
10. autoflush
11. buffer

1. import:

This attribute is used to import packages. While doing coding you may need to include more than one packages, In such scenarios this page directive's attribute is very useful as it allows you to mention more than one packages at the same place separated by commas (,). Alternatively you can have multiple instances of page element each one with different package.

Syntax of import attribute –

```
<%@page import="value"%>
```

Here value is package name.

Example of import- The following is an example of how to import more than one package using import attribute of page directive.

```
<%@page import="java.io.*%>
<%@page import="java.lang.*%>
<%--Comment: OR Below Statement: Both are Same--%>
<%@page import="java.io.*, java.lang.*"%>
```

2. session:

Generally while building a user interactive JSP application, we make sure to give access to the user to get hold of his/her personal data till the session is active. Consider an example of logging in into your bank account, we can access all of your data till we signout (or session expires). In order to maintain session for a page the session attribute should be true.

This attribute is to handle HTTP sessions for JSP pages. It can have two values: true or false. Default value for session attribute is true, which means if you do not mention this attribute, server may assume that HTTP session is required for this page.

Default value for this attribute: true

Syntax of session attributes:

```
<%@ page session="value"%>
```

here value is **either true OR false**

Examples of session:

```
<%@ page session="true"%>
```

The above code would allow a page to have session implicit objects.

```
<%@ page session="false"%>
```

If this code is specified in a JSP page, it means session objects will not be available for that page. Hence session cannot be maintained for that page.

3. isErrorPage:

This attribute is used to specify whether the current JSP page can be used as an error page for another JSP page. If value of isErrorPage is true it means that the page can be used for exception handling for another page. Generally these pages has error/warning messages OR exception handling codes and being called by another JSP page when there is an exception occurred there.

There is another use of isErrorPage attribute – The exception implicit object can only be available to those pages which has isErrorPage set to true. If the value is false, the page cannot use exception implicit object.

Default value: false

Syntax of isErrorPage attribute:

```
<%@ page isErrorPage="value"%>
```

Here value is either true OR false.

Example of isErrorPage:

```
<%@ page isErrorPage="true"%>
```

This makes a JSP page, a exception handling page.

4. errorPage:

When isErrorPage attribute is true for a particular page then it means that the page can be called by another page in case of an exception. errorPage attribute is used to specify the URL of a JSP page which has isErrorPage attrbute set to true. It handles the un-handled exceptions in the page.

Syntax of errorPage attribute:

```
<%@ page errorPage="value"%>
```

Here value is a JSP page name which has exception handling code (and isErrorPage set to true).

Example of errorPage:

```
<%@ page errorPage="ExceptionHandling.jsp"%>
```

This means if any exception occurs on the JSP page where this code has been placed, the ExceptionHandling.jsp (this page should have isErrorPage true) page needs to be called.

5. contentType:

This attribute is used to set the content type of a JSP page.

Default value: text/html

Syntax of contentType attribute:

```
<%@ page contentType="value"%>
```

here value of content type can be anything such as: text/html, text/xml etc.

Example of contentType:

Below code can be used for text/html pages.

```
<%@ page contentType="text/html"%>
```

for text/xml based pages:

```
<%@ page contentType="text/xml"%>
```

6. isThreadSafe:

Lets understand this with an example. Suppose you have created a JSP page and mentioned isThreadSafe as true, it means that the JSP page supports multithreading (more than one thread can execute the JSP page simultaneously). On the other hand if it is set to false then JSP engine won’t allow multithreading which means only single thread will execute the page code.

Default value for isThreadSafe attribute: true.

Syntax of isThreadSafe attribute:

```
<%@ page isThreadSafe="value"%>
```

here value can be true OR false.

Example of isThreadSafe:

```
<%@ page isThreadSafe="false"%>
```

Only one thread will be responsible for JSP page execution.

7. buffer:

This attribute is used to specify the buffer size. If you specify this to none during coding then the output would directly written to Response object by JSPWriter. And, if you specify a buffer size then the output first written to buffer then it will be available for response object.

Syntax of buffer attribute:

```
<%@ page buffer="value"%>
```

value is **size in kb** or **none**.

Example of buffer:

No buffer for this page:

```
<%@ page buffer="none"%>
```

5 kb buffer size for the page, which has below code:

```
<%@ page buffer="5kb"%>
```

8. extends:

Like java, here also this attribute is used to extend(inherit) the class.

Syntax of extends attribute:

```
<%@ page extends="value"%>
```

Value is package_name.class_name.

Example of extends:

The below code will inherit the SampleClass from package: mypackage

```
<%@ page extends="mypackage.SampleClass"%>
```

9. info:

It provides a description to a JSP page. The string specified in info will return when we will call getServletInfo() method.

Syntax of info:

```
<%@ page info="value"%>
```
here value is Message or Description

Example of info attribute:

```
<%@ page info="This code is given by Chaitanya Singh"%>
```

10. language:

It specifies the scripting language(underlying language) being used in the page.

Syntax of language:

```
<%@ page language="value"%>
```
value is scripting language here.

Example of language attribute:

```
<%@ page language="java"%>
```

11. autoFlush:

If it is true it means the buffer should be flushed whenever it is full. false will throw an exception when buffer overflows.

Default value: True

Syntax of autoFlush:

```
<%@ page autoFlush="value"%>
```
value can be true or false.

Example of autoFlush attribute:

Buffer will be flushed out when it is full –

```
<%@ page autoFlush="true"%>
```
It will throw an exception when buffer is full due to overflow condition

```
<%@ page autoFlush="true"%>
```

12. isScriptingEnabled:

It has been dropped and not in use.

13. isELIgnored:

This attribute specify whether expressions will be evaluated or not.

Default value: true

Syntax of isELIgnored:

```
<%@ page isELIgnored="value"%>
```
value can be true or false.

Example of isELIgnored attribute:

Any expression present inside JSP page will not be evaluated –

```
<%@ page isELIgnored="false"%>
```
Expression will be evaluated (true is a default value so no need to specify)-

```
<%@ page isELIgnored="true"%>
```

2) Include Directive

Include directive is used to copy the content of one JSP page to another. It’s like including the code of one file into another.

Syntax of Include Directive:

```
<%@include file ="value"%>
```
here value is the JSP file name which needs to be included. If the file is in the same directory then just specify the file name otherwise complete URL(or path) needs to be mentioned in the value field.

Note: It can be used anywhere in the page.

Example:

```
<%@include file="myJSP.jsp"%>
```
You can use the above code in your JSP page to copy the content of myJSP.jsp file. However in this case both the JSP files must be in the same directory. If the myJSP.jsp is in the different directory then instead of just file name you would need to specify the complete path in above code.

3) Taglib Directive

This directive basically allows user to use Custom tags in JSP. we shall discuss about Custom tags in detail in coming JSP tutorials. Taglib directive helps you to declare custom tags in JSP page.

Syntax of Taglib Directive:

```
<%@taglib uri ="taglibURI" prefix="tag prefix"%>
```

Where URI is uniform resource locator, which is used to identify the location of custom tag and tag prefix is a string which can identify the custom tag in the location identified by uri.

Example of Targlib:

```
<%@ taglib uri="http://www.sample.com/mycustomlib" prefix="demotag" %>
<html>
<body>
<demotag:welcome/>
</body>
</html>
```

As you can see that uri is having the location of custom tag library and prefix is identifying the prefix of custom tag.

Note: In above example – <demotag: welcome> has a prefix demotag.

JSP – STANDARD TAG LIBRARY (JSTL)

JSP Standard Tag Library (JSTL) is a set of useful tags to simplify the JSP development. It provides tags to control the JSP page behavior, iteration and control statements, internationalization tags, and SQL tags. JSTL is part of the Java EE API and included in most servlet containers. There are five groups of JSTL: core tags, sql tags, xml tags, internationalization tags and functions tags.

In the code snippet below, there is a simple loop coded with JSTL. Without any tag library or tags, we can write the counterpart code with scriptlets that contain Java code in it. But external tag libraries provide more simple and useful capabilities to us. We can do more with writing less.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:forEach var="number" begin="5" end="10">

  <c:out value="${number}"></c:out>

</c:forEach>
```

JSP IMPLICIT OBJECTS

These objects are created by JSP Engine during translation phase (while translating JSP to Servlet). They are being created inside service method so we can directly use them within Scriptlet without initializing and declaring them. There are total 9 implicit objects available in JSP.

Implicit Objects and their corresponding classes:

out	javax.servlet.jsp.JspWriter
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession

application	javax.servlet.ServletContext
exception	javax.servlet.jsp.JspException
page	java.lang.Object
pageContext	javax.servlet.jsp.PageContext
config	javax.servlet.ServletConfig

1. **Out**: This is used for writing content to the client (browser). It has several methods which can be used for properly formatting output message to the browser and for dealing with the buffer.
2. **Request**: The main purpose of request implicit object is to get the data on a JSP page which has been entered by user on the previous JSP page. While dealing with login and signup forms in JSP we often prompts user to fill in those details, this object is then used to get those entered details on an another JSP page (action page) for validation and other purposes.
3. **Response**: It is basically used for modfying or delaing with the response which is being sent to the client(browser) after processing the request.
4. **Session:** It is most frequently used implicit object, which is used for storing the user's data to make it available on other JSP pages till the user session is active.
5. **Application:** This is used for getting application-wide initialization parameters and to maintain useful data across whole JSP application.
6. **Exception:** Exception implicit object is used in exception handling for displaying the error messages. This object is only available to the JSP pages, which has isErrorPage set to true.
7. **Page:** Page implicit object is a reference to the current Servlet instance (Converted Servlet, generated during translation phase from a JSP page). We can simply use **this** in place of it. I'm not covering it in detail as it is rarely used and not a useful implicit object while building a JSP application.
8. **pageContext**: It is used for accessing page, request, application and session attributes.
9. **Config:** This is a Servlet configuration object and mainly used for accessing getting configuration information such as servlet context, servlet name, configuration parameters etc.

USING BEANS IN JSP PAGES

jsp:useBean, jsp:setProperty and jsp:getProperty Action Tags

Syntax of jsp:useBean:

```
<jsp: useBean id="unique_name_to_identify_bean"
class="package_name.class_name" />
```

Syntax of jsp:setProperty:

```
<jsp:setProperty name="unique_name_to_identify_bean"
property="property_name" />
```

Syntax of jsp:getProperty:

```
<jsp:getProperty name="unique_name_to_identify_bean"
property="property_name" />
```

A complete example of useBean, setProperty and getProperty

1) We have a bean class Details where we are having three variables username, age and password. In order to use the bean class and it's properties in JSP we have initialized the class like this in the userdetails.jsp page –

```
<jsp:useBean id="userinfo" class="beginnersbook.com.Details"></jsp:useBean>
```

We have used useBean action to initialize the class. Our class is in beginnersbook.com package so we have given a fully qualified name **beginnersbook.com.Details**.

2) We have mapped the properties of bean class and JSP using setProperty action tag. We have given '*' in the property field to map the values based on their names because we have used the same property name in bean class and index.jsp JSP page. In the name field we have given the unique identifier which we have defined in useBean tag.

```
<jsp:setProperty property="*" name="userinfo"/>
```

3) To get the property values we have used getProperty action tag.

```
<jsp:getProperty property="propertyname" name="userinfo"/>
```

Details.java

```
package beginnersbook.com;
public class Details {
            public Details() {
   }
   private String username;
   private int age;
   private String password;
            public String getUsername() {
                        return username;
            }
            public void setUsername(String username) {
                        this.username = username;
            }
            public int getAge() {
                        return age;
            }
            public void setAge(int age) {
                        this.age = age;
            }
            public String getPassword() {
                        return password;
            }
            public void setPassword(String password) {
                        this.password = password;
            }

}
```

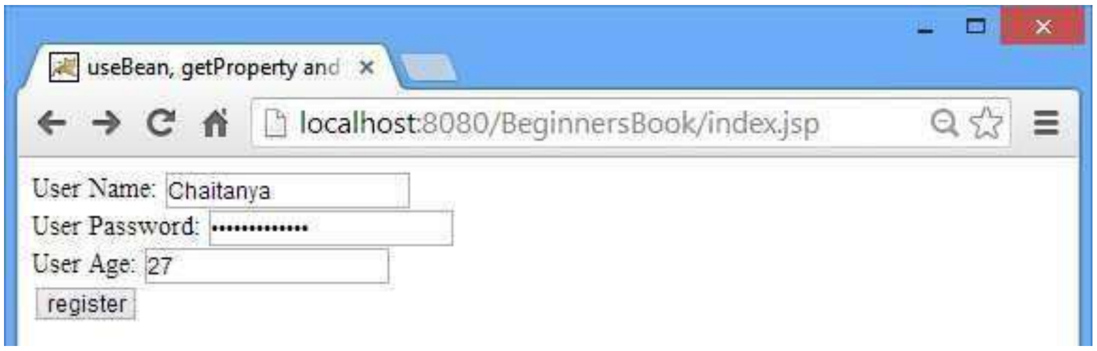
index.jsp

```
<html>
        <head>
          <title>
                    useBean, getProperty and setProperty example
          </title>
        </head>
        <form action="userdetails.jsp" method="post">
          User Name: <input type="text" name="username"><br>
          User Password: <input type="password" name="password"><br>
          User Age: <input type="text" name="age"><br>
          <input type="submit" value="register">
        </form>
</html>
```

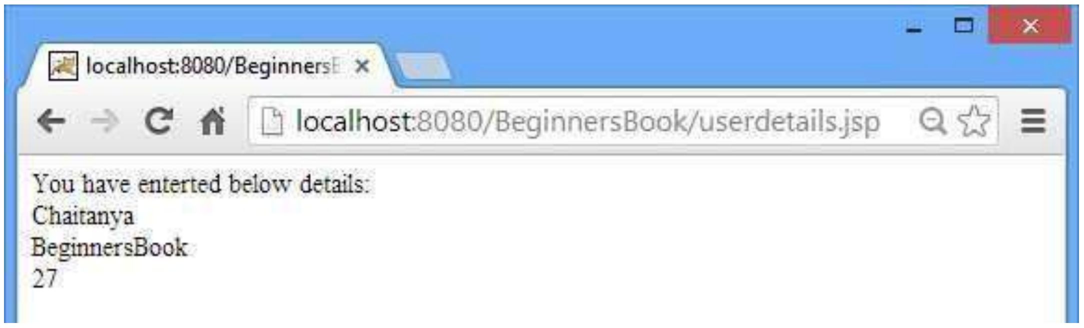
userdetails.jsp

```
<jsp:useBean id="userinfo" class="beginnersbook.com.Details"></jsp:useBean>
<jsp:setProperty property="*" name="userinfo"/>
You have enterted below details:<br>
<jsp:getProperty property="username" name="userinfo"/><br>
<jsp:getProperty property="password" name="userinfo"/><br>
<jsp:getProperty property="age" name="userinfo" /><br>
```

Output:



User Name: Chaitanya
User Password: ••••••••••••••
User Age: 27


localhost:8080/BeginnersE
localhost:8080/BeginnersBook/userdetails.jsp

```
You have enterted below details:
Chaitanya
BeginnersBook
27
```

COOKIES IN JSP

- Cookies are the text files which are stored on the client machine.
- They are used to track the information for various purposes.
- It supports HTTP cookies using servlet technology
- The cookies are set in the HTTP Header.
- If the browser is configured to store cookies, it will keep information until expiry date.

Following are the cookies methods:

- Public void setDomain(String domain)

  It is used to set the domain to which the cookie applies

- Public String getDomain()

  It is used to get the domain to which cookie applies

- Public void setMaxAge(int expiry)

  It sets the maximum time which should apply till the cookie expires

- Public intgetMaxAge()

  It returns the maximum age of cookie

- Public String getName()

  It returns the name of the cookie

- Public void setValue(String value)

  Sets the value associated with the cookie

- Public String getValue()

  Get the value associated with the cookie

- Public void setPath(String path)

  It sets the path to which cookie applies

- Public String getPath()

  It gets the path to which the cookie applies

- Public void setSecure(Boolean flag)

  It should be sent over encrypted connections or not.

- Public void setComment(String cmt)

It describes the cookie purpose

- Public String getComment()

  It the returns the cookie comments which has been described.

How to Handle Cookies in JSP

1. Creating the cookie object
2. Setting the maximum age
3. Sending the cookie in HTTP response headers

Example:

In this example, we are creating cookies of username and email and add age to the cookie for 10 hours and trying to get the variable names in the action_cookie.jsp

Action_cookie.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Guru Cookie</title>
</head>
<body>
<form action="action_cookie_main.jsp" method="GET">
Username: <input type="text" name="username">
<br />
Email: <input type="text" name="email" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Action_cookie_main.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%

Cookie username = new Cookie("username",
request.getParameter("username"));
Cookie email = new Cookie("email",
request.getParameter("email"));


username.setMaxAge(60*60*10);
email.setMaxAge(60*60*10);
```

```
// Add both the cookies in the response header.
response.addCookie( username );
response.addCookie( email );
%>


<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Guru Cookie JSP</title>
</head>
<body>

<b>Username:</b>
<%= request.getParameter("username")%>
<b>Email:</b>
<%= request.getParameter("email")%>

</body>
</html>
```

Explanation of the Code

Action_cookie.jsp

Code Line 10-15: Here we are taking a form which has to be processed in action_cookie_main.jsp. Also, we are taking two fields "username" and "email" which has to be taken input from the user with a submit button.

Action_cookie_main.jsp

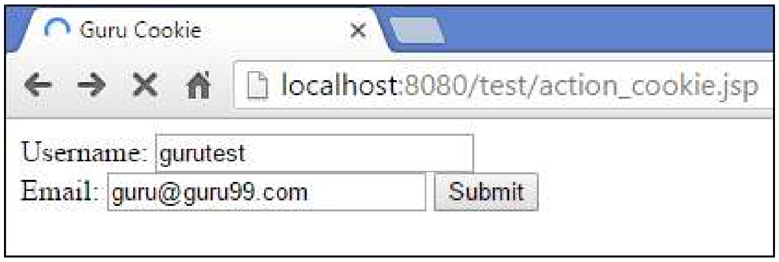
Code Line 6-9: Creating two cookie objects of "username" and "email" using request.getParameter.

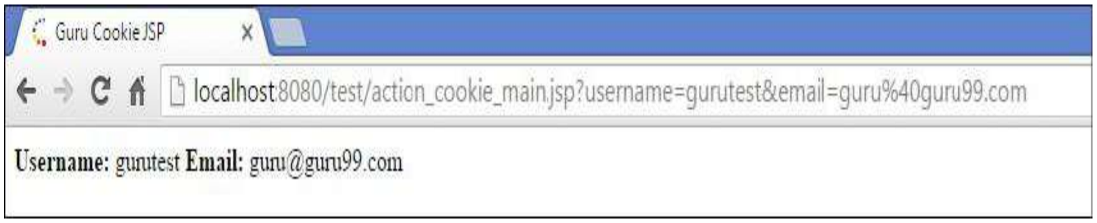
Code Line 12-13: Here we are adding age to both the cookies, which have been created of 10 hours i.e. cookies will expire in that age.

Code Line 16-17: Adding cookies to the session of username and email and these two cookies can fetched when requested by getParameter().

Output:

When you execute the above code you get the following output:




Guru Cookie JSP
localhost:8080/test/action_cookie_main.jsp?username=gurutest&email=guru%40guru99.com

Username: gurutest **Email:** guru@guru99.com

When we execute the action_cookie.jsp we get two fields username and email, and it takes user input and then we click on the submit button.

We get the output from action_cookie_main.jsp where variables are stored in the cookies on the client side.

HOW TO VALIDATE AND INVALIDATE SESSION IN JSP

We have already seen invalidate() method in session implicit object tutorial. In this post we are going to discuss it in detail. Here we will see how to validate/invalidate a session.

Example

Let's understand this with the help of an example: In the below example we have three "jsp" pages.

- index.jsp: It is having four variables which are being stored in session object.
- display.jsp: It is fetching the attributes (variables) from session and displaying them.
- errorpage.jsp: It is first calling session.invalidate() in order to invalidate (make the session inactive) the session and then it has a logic to validate the session (checking whether the session is active or not).

index.jsp

```
<%
  String firstname="Chaitanya";
  String middlename="Pratap";
  String lastname="Singh";
  int age= 26;
  session.setAttribute( "fname", firstname );
  session.setAttribute( "mname", middlename );
  session.setAttribute( "lname", lastname );
  session.setAttribute( "UAge", age );
%>
<a href="display.jsp">See Details</a>
<a href="errorpage.jsp">Invalidate Session</a>
```

display.jsp

```
<%= session.getAttribute( "fname" ) %>
<%= session.getAttribute( "mname" ) %>
<%= session.getAttribute( "lname" ) %>
<%= session.getAttribute( "UAge" ) %>
```

errorpage.jsp

```
<%session.invalidate();%>
```

```
<% HttpSession nsession = request.getSession(false);
if(nsession!=null)
{
   String data=(String)session.getAttribute( "fname" );
   out.println(data);
}
else
  out.println("Session is not active");
%>
```

Output
while opening display page, all the attributes are getting displayed on client (browser). Since we have already called invalidate in the first line of errorpage.jsp, it is displaying the message "Session is not active" on the screen

Points to Note:
1) This will deactivate the session

```
<%session.invalidate();%>
```

2) This logic will exceute the if body when the session is active else it would run the else part.

```
<% HttpSession nsession = request.getSession(false);
if(nsession!=null)
   ...
 else
   ...
 %>
```

Contents compiled from the following References:

1. https://beginnersbook.com/2013/05/jsp-tutorial-introduction/
2. https://examples.javacodegeeks.com/enterprise-java/jsp/jsp-tutorial-beginners/
3. https://www.javatpoint.com/jsp-tutorial
4. https://www.oreilly.com/library/view/javaserver-pages-3rd/0596005636/ch03s03s01.html#jserverpages3-CHP-3-TABLE-1
5. https://www.oreilly.com/library/view/javaserver-pages-3rd/0596005636/ch03s03s01.html
6. http://www.c4learn.com/java/jsp/jsp-processing-architecture/
7. https://www.guru99.com/jsp-implicit-objects.html
8. https://www.guru99.com/jsp-cookies-handling.html

JSP - Database Access

To start with basic concept, let us create a table and create a few records in that table as follows:

Create Table

To create the Employees table in the EMP database, use the following steps:

Step 1

Open a Command Prompt and change to the installation directory as follows:

```
C:\>

C:\>cd Program Files\MySQL\bin

C:\Program Files\MySQL\bin>
```

Step 2

Login to the database as follows:

```
C:\Program Files\MySQL\bin>mysql -u root -p

Enter password: ********

mysql>
```

Step 3

Create the Employee table in the TEST database as follows:

```
mysql> use TEST;

mysql> create table Employees

   (

      id int not null,

      age int not null,

      first varchar (255),

      last varchar (255)

   );

Query OK, 0 rows affected (0.08 sec)

mysql>
```

Create Data Records

Let us now create a few records in the Employee table as follows:

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');

Query OK, 1 row affected (0.05 sec)



mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');

Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');

Query OK, 1 row affected (0.00 sec)


mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');

Query OK, 1 row affected (0.00 sec)


mysql>
```

SELECT Operation

Following example shows how we can execute the SQL SELECT statement using JTSL in JSP programming:

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>

<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>


<html>

   <head>

      <title>SELECT Operation</title>

   </head>


   <body>

      <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"

         url = "jdbc:mysql://localhost/TEST"

         user = "root"  password = "pass123"/>


      <sql:query dataSource = "${snapshot}" var = "result">

         SELECT * from Employees;

      </sql:query>


      <table border = "1" width = "100%">

         <tr>

            <th>Emp ID</th>

            <th>First Name</th>

            <th>Last Name</th>

            <th>Age</th>
         </tr>


         <c:forEach var = "row" items = "${result.rows}">
            <tr>
               <td><c:out value = "${row.id}"/></td>
               <td><c:out value = "${row.first}"/></td>
               <td><c:out value = "${row.last}"/></td>
               <td><c:out value = "${row.age}"/></td>
            </tr>
         </c:forEach>
      </table>


   </body>
</html>
```

Access the above JSP, the following result will be displayed.

Emp ID	First Name	Last Name	Age
100	Zara	Ali	18
101	Mahnaz	Fatma	25
102	Zaid	Khan	30
103	Sumit	Mittal	28

INSERT Operation

Following example shows how we can execute the SQL INSERT statement using JTSL in JSP programming.

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>


<html>
   <head>
      <title>JINSERT Operation</title>
   </head>
   <body>
      <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
         url = "jdbc:mysql://localhost/TEST"
         user = "root"  password = "pass123"/>
         <sql:update dataSource = "${snapshot}" var = "result">
         INSERT INTO Employees VALUES (104, 2, 'Nuha', 'Ali');
      </sql:update>

      <sql:query dataSource = "${snapshot}" var = "result">
         SELECT * from Employees;
      </sql:query>

      <table border = "1" width = "100%">
         <tr>
            <th>Emp ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
         </tr>

         <c:forEach var = "row" items = "${result.rows}">
            <tr>
               <td><c:out value = "${row.id}"/></td>
               <td><c:out value = "${row.first}"/></td>
               <td><c:out value = "${row.last}"/></td>
               <td><c:out value = "${row.age}"/></td>
            </tr>
         </c:forEach>
      </table>

   </body>
</html>
```

Access the above JSP, the following result will be displayed as:

Emp ID	**First Name**	**Last Name**	**Age**

100	Zara	Ali	18
101	Mahnaz	Fatma	25
102	Zaid	Khan	30
103	Sumit	Mittal	28
104	Nuha	Ali	2

DELETE Operation

Following example shows how we can execute the SQL DELETE statement using JTSL in JSP programming.

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>

<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>


<html>

   <head>

      <title>DELETE Operation</title>

   </head>


   <body>

      <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"

         url = "jdbc:mysql://localhost/TEST"

         user = "root" password = "pass123"/>


      <c:set var = "empId" value = "103"/>


      <sql:update dataSource = "${snapshot}" var = "count">

         DELETE FROM Employees WHERE Id = ?

         <sql:param value = "${empId}" />

      </sql:update>


      <sql:query dataSource = "${snapshot}" var = "result">

         SELECT * from Employees;

      </sql:query>
      <table border = "1" width = "100%">
         <tr>
            <th>Emp ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
         </tr>

         <c:forEach var = "row" items = "${result.rows}">
            <tr>
               <td><c:out value = "${row.id}"/></td>
               <td><c:out value = "${row.first}"/></td>
               <td><c:out value = "${row.last}"/></td>
               <td><c:out value = "${row.age}"/></td>
            </tr>
         </c:forEach>
      </table>

   </body>
</html>
```

Access the above JSP, the following result will be displayed.

Emp ID	First Name	Last Name	Age
100	Zara	Ali	18
101	Mahnaz	Fatma	25
102	Zaid	Khan	30

UPDATE Operation

Following example shows how we can execute the SQL UPDATE statement using JTSL in JSP programming.

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>
```

```
<html>
   <head>
      <title>DELETE Operation</title>
   </head>

   <body>
      <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
         url = "jdbc:mysql://localhost/TEST"
         user = "root" password = "pass123"/>

      <c:set var = "empId" value = "102"/>

      <sql:update dataSource = "${snapshot}" var = "count">
         UPDATE Employees SET last = 'Ali'
         <sql:param value = "${empId}" />
      </sql:update>

      <sql:query dataSource = "${snapshot}" var = "result">
         SELECT * from Employees;
      </sql:query>

      <table border = "1" width = "100%">
         <tr>
            <th>Emp ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
         </tr>

         <c:forEach var = "row" items = "${result.rows}">
            <tr>
               <td><c:out value = "${row.id}"/></td>
               <td><c:out value = "${row.first}"/></td>
               <td><c:out value = "${row.last}"/></td>
               <td><c:out value = "${row.age}"/></td>
            </tr>

         </c:forEach>

      </table>


   </body>

</html>
```

Access the above JSP, the following result will be displayed.

Emp ID	First Name	Last Name	Age
100	Zara	Ali	18
101	Mahnaz	Fatma	25
102	Zaid	Ali	30

JSP – JavaBeans (Deploying JavaBeans in a JSP Page)

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

Following are the unique characteristics that distinguish a JavaBean from other Java classes:

- It provides a default, no-argument constructor.
- It should be serializable and that which can implement the **Serializable** interface.
- It may have a number of properties which can be read or written.
- It may have a number of "**getter**" and "**setter**" methods for the properties.

JavaBeans Properties

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including the classes that you define.

A JavaBean property may be read, write, read only, or write only. JavaBean properties are accessed through two methods in the JavaBean's implementation class.

S.No.	Method & Description
1	getPropertyName() For example, if property name is firstName, your method name would be getFirstName() to read that property. This method is called accessor.

2	setPropertyName() For example, if property name is firstName, your method name would be setFirstName() to write that property. This method is called mutator.

A read-only attribute will have only a **getPropertyName()** method, and a write-only attribute will have only a **setPropertyName()** method.

JavaBeans Example

Consider a student class with few properties −

```
package com.tutorialspoint;


public class StudentsBean implements java.io.Serializable {
   private String firstName = null;
   private String lastName = null;
   private int age = 0;

   public StudentsBean() {
   }
   public String getFirstName(){
      return firstName;
   }
   public String getLastName(){
      return lastName;
   }
   public int getAge(){
      return age;
   }
   public void setFirstName(String firstName){
      this.firstName = firstName;
   }
   public void setLastName(String lastName){
      this.lastName = lastName;
   }
   public void setAge(Integer age){
      this.age = age;
   }
}
```

Accessing JavaBeans

The useBean action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full syntax for the useBean tag is as follows:

```
<jsp:useBean id = "bean's name" scope = "bean's scope" typeSpec/>
```

Here values for the scope attribute can be a page, request, session or application based on your requirement. The value of the id attribute may be any value as a long as it is a unique name among other useBean declarations in the same JSP.

Following example shows how to use the useBean action.

```
<html>
   <head>
      <title>useBean Example</title>
   </head>

   <body>
      <jsp:useBean id = "date" class = "java.util.Date" />
      <p>The date/time is <%= date %>
   </body>
</html>
```

You will receive the following result:

```
The date/time is Thu Sep 30 11:18:11 GST 2010
```

Accessing JavaBeans Properties

Along with <jsp:useBean...> action, you can use the <jsp:getProperty/>action to access the get methods and the <jsp:setProperty/> action to access the set methods.

Here is the syntax:

```
<jsp:useBean id = "id" class = "bean's class" scope = "bean's scope">

   <jsp:setProperty name = "bean's id" property = "property name"

      value = "value"/>

   <jsp:getProperty name = "bean's id" property = "property name"/>

   ...........

</jsp:useBean>
```

The name attribute references the id of a JavaBean previously introduced to the JSP by the useBean action. The property attribute is the name of the getor the set methods that should be invoked.

Following example shows how to access the data using the above syntax:

```
<html>

   <head>

      <title>get and set properties Example</title>

   </head>


   <body>

      <jsp:useBean id = "students" class = "com.tutorialspoint.StudentsBean">

         <jsp:setProperty name = "students" property = "firstName" value = "Zara"/>

         <jsp:setProperty name = "students" property = "lastName" value = "Ali"/>

         <jsp:setProperty name = "students" property = "age" value = "10"/>

      </jsp:useBean>


      <p>Student First Name:

         <jsp:getProperty name = "students" property = "firstName"/>

      </p>


      <p>Student Last Name:

         <jsp:getProperty name = "students" property = "lastName"/>

      </p>


      <p>Student Age:
         <jsp:getProperty name = "students" property = "age"/>

      </p>



   </body>

</html>
```

Let us make the StudentsBean.class available in CLASSPATH. Access the above JSP thefollowing result will be displayed as :

```
Student First Name: Zara

Student Last Name: Ali

Student Age: 10
```

UNIT – V

INTRODUCTION TO JAVASCRIPT

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is **getElementById()**.

This example uses the method to "find" an HTML element (with id="demo") and changes the element content (**innerHTML**) to "Hello JavaScript":

Example

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

JavaScript accepts both double and single quotes:

The <script> Tag

In HTML, JavaScript code must be inserted between <script> and </script> tags.

Example

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

JavaScript Functions and Events

A JavaScript **function** is a block of JavaScript code, that can be executed when "called" for.

For example, a function can be called when an **event** occurs, like when the user clicks a button.

JavaScript in <head>

In this example, a JavaScript function is placed in the <head> section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction()

{
   document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>

<body>
```

```
<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

JavaScript in <body>

In this example, a JavaScript function is placed in the <body> section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

External JavaScript

Scripts can also be placed in external files:

External file: myScript.js

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension **.js**.

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

Example

```
<!DOCTYPE html>
<html>
<body>

<script src="myScript.js"></script>

</body>
</html>
```

You can place an external script reference in <head> or <body> as you like.

The script will behave as if it was located exactly where the <script> tag is located.

Note: External scripts cannot contain <script> tags.

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

Example

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

External References

External scripts can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a script:

Example

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

This example uses a script located in a specified folder on the current web site:

Example

```
<script src="/js/myScript1.js"></script>
```

This example links to a script located in the same folder as the current page:

Example

```
<script src="myScript1.js"></script>
```

JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by the computer.

In a programming language, these program instructions are called **statements**.

JavaScript is a **programming language**.

JavaScript statements are separated by **semicolons**:

Example

```
var x, y, z;
x = 5;
y = 6;
z = x + y;
```

JavaScript Statements

This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

Example

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```

Single Line Comments

Single line comments start with //.

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

This example uses a single-line comment before each code line:

Example

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";
// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

avaScript Variables

JavaScript variables are containers for storing data values.

In this example, x, y, and z, are variables:

Example

```
var x =  5;
var  y = 6;
var z = x + y;
```

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

Example

```
function myFunction(p1, p2) {
    return p1 * p2;              // The function returns the product of p1 and p2
}
```

Function Return

When JavaScript reaches a **return statement**, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example
Calculate the product of two numbers, and return the result:

```
var x = myFunction(4, 3);        // Function is called, return value will end up in x

function myFunction(a, b) {
    return a * b;                // Function returns the product of a and b
}
```

JAVASCRIPT OBJECTS

Real Life Objects, Properties, and Methods

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties	Methods

	car.name = Fiat
	car.model = 500
	car.weight = 850kg
	car.color = white
		car.start()
		car.drive()
		car.brake()
		car.stop()

All cars have the same **properties**, but the property values differ from car to car.

All cars have the same **methods**, but the methods are performed at different times.

JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

var car = "Fiat";

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

var car = {type:"Fiat", model:"500", color:"white"};

Object Properties

The name:values pairs (in JavaScript objects) are called **properties**.

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

Property	**Property Value**
firstName	John

lastName	Doe
age	50
eyeColor	Blue

JavaScript Events with HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

<*element event*=**'*some JavaScript*'**>

With double quotes:

<*element event*=**"*some JavaScript*"**>

In the following example, an onclick attribute (with code), is added to a button element:

Example

```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>
```

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using **this**.innerHTML):

Example

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JavaScript Strings

A JavaScript string simply stores a series of characters like "John Doe".

A string can be any text inside quotes. You can use single or double quotes:

Example

```
var carname = "Volvo XC60";
var carname = 'Volvo XC60';
```

String Length

The length of a string is found in the built in property **length**:

Example

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Finding a String in a String

The **indexOf()** method returns the index of (the position of) the **first** occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

The **lastIndexOf()** method returns the index of the **last** occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

Searching for a String in a String

The **search()** method searches a string for a specified value and returns the position of the match:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

JavaScript Numbers

JavaScript has only one type of number. Numbers can be written with, or without, decimals.

Example

```
var x = 34.00;    // A number with decimals
var y = 34;       // A number without decimals
```

JavaScript Number Methods and Properties

Primitive values (like 3.14 or 2014), cannot have properties and methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

The toString() Method

toString() returns a number as a string.

All number methods can be used on any type of numbers (literals, variables, or expressions):

Example

```
var x = 123;
x.toString();            // returns 123 from variable x
(123).toString();        // returns 123 from literal 123
(100 + 23).toString();  // returns 123 from expression 100 + 23
```

The toFixed() Method

toFixed() returns a string, with the number written with a specified number of decimals:

Example

```
var x = 9.656;
x.toFixed(0);           // returns 10
x.toFixed(2);           // returns 9.66
x.toFixed(4);           // returns 9.6560
x.toFixed(6);           // returns 9.656000
```

The toPrecision() Method

toPrecision() returns a string, with a number written with a specified length:

Example

```
var x = 9.656;
x. toPrecision();        // returns 9.656
x.toPrecision(2);       // returns 9.7
x.toPrecision(4);       // returns 9.656
x.toPrecision(6);       // returns 9.65600
```

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Example

```
Math.PI;            // returns 3.141592653589793
```

Math.round()

Math.round(x) returns the value of x rounded to its nearest integer:

Example

```
Math.round(4.7);    // returns 5
Math.round(4.4);    // returns 4
```

Math.pow()

Math.pow(x, y) returns the value of x to the power of y:

Example

```
Math.pow(8, 2);      // returns 64
```

Math.sqrt()

Math.sqrt(x) returns the square root of x:

Example

```
Math.sqrt(64);      // returns 8
```

Math.abs()

Math.abs(x) returns the absolute (positive) value of x:

Example

```
Math.abs(-4.7);     // returns 4.7
```

Math.ceil()

Math.ceil(x) returns the value of x rounded **up** to its nearest integer:

Example

```
Math.ceil(4.4);     // returns 5
```

Math.floor()

Math.floor(x) returns the value of x rounded **down** to its nearest integer:

Example

```
Math.floor(4.7);     // returns 4
```

Math.min() and Math.max()

Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments:

Example

```
Math.min(0, 150, 30, 20, -8, -200);  // returns -200
```

Math.random()

Math.random() returns a random number between 0 (inclusive), and 1 (exclusive):

Example

```
Math.random();     // returns a random number
```

Math Constructor

Unlike other global objects, the Math object has no constructor. Methods and properties are static.

All methods and properties (constants) can be used without creating a Math object first.

Math Object Methods

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
atan2(y, x)	Returns the arctangent of the quotient of its arguments
ceil(x)	Returns the value of x rounded up to its nearest integer

cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of Ex
floor(x)	Returns the value of x rounded down to its nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x, y, z, ..., n)	Returns the number with the highest value
min(x, y, z, ..., n)	Returns the number with the lowest value
pow(x, y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Returns the value of x rounded to its nearest integer
sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

JavaScript Dates

The Date object lets you work with dates (years, months, days, hours, minutes, seconds, and milliseconds)

Displaying Dates

A script to display dates inside a <p> element with id="demo" is shown below:

Example

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Date();
</script>
```

Date Get Methods

Get methods are used for getting a part of a date. Here are the most common (alphabetically):

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

JavaScript Arrays

avaScript arrays are used to store multiple values in a single variable.

Example

```
var cars = ["Saab", "Volvo", "BMW"];
```

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Access the Elements of an Array

You refer to an array element by referring to the **index number**.

This statement accesses the value of the first element in cars:

```
var name = cars[0];
```

This statement modifies the first element in cars:

```
cars[0] = "Opel";
```

Example

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

Example

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

JavaScript Boolean() Function

A JavaScript Boolean represents one of two values: **true** or **false**.

You can use the Boolean() function to find out if an expression (or a variable) is true:

Example

```
Boolean(10 > 9)        // returns true
```

Comparisons and Conditions

Operator	Description	Example
==	equal to	if (day == "Monday")
>	greater than	if (salary > 9000)
<	less than	if (age < 18)

JavaScript If...Else Statements

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false
- Use **switch** to specify many alternative blocks of code to be executed

The if Statement

Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if (condition) {
    block of code to be executed if the condition is true
}
```

Note that **if** is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

Example
Make a "Good day" greeting if the hour is less than 18:00:

```
if (hour < 18) {
    greeting = "Good day";
}
```

The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(expression) {
    case n:
        code block
        break;
    case n:
        code block
        break;
    default:
        code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

Example

The getDay() method returns the weekday as a number between 0 and 6.

(Sunday=0, Monday=1, Tuesday=2 ..)

This example uses the weekday number to calculate the weekday name:

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
```

The break Keyword

When JavaScript reaches a **break** keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

The default Keyword

The **default** keyword specifies the code to run if there is no case match:

Example

The getDay() method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message:

```
switch (new Date().getDay()) {
  case 6:
        text = "Today is Saturday";
        break;
    case 0:
        text = "Today is Sunday";
        break;
    default:
        text = "Looking forward to the Weekend";
}
```

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
for (i = 0; i < cars.length; i++) {
    text += cars[i] + "<br>";
}
```

Different Kinds of Loops

JavaScript supports different kinds of loops:

- **for -** loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **while** - loops through a block of code while a specified condition is true
- **do/while -** also loops through a block of code while a specified condition is true

The For Loop

The for loop is often the tool you will use when you want to create a loop.

The for loop has the following syntax:

for (*statement 1*; *statement 2*; *statement 3*) {
    code block to be executed
}

Statement 1 is executed before the loop (the code block) starts.

Statement 2 defines the condition for running the loop (the code block).

Statement 3 is executed each time after the loop (the code block) has been executed.

Example

```
for (i = 0; i < 5; i++) {
   text += "The number is " + i + "<br>";
}
```

From the example above, you can read:

Statement 1 sets a variable before the loop starts (var i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

JavaScript While Loop

Loops can execute a block of code as long as a specified condition is true.

The While Loop

The while loop loops through a block of code as long as a specified condition is true.

Syntax

```
while (condition) {
   code block to be executed
}
```

Example

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

Example

```
while (i < 10) {
   text += "The number is " + i;
   i++;
}
```

The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {
    code block to be executed
}
while (condition);
```

Example

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

```
do {
    text += "The number is " + i;
    i++;
}
while (i < 10);
```

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Example

```
function validateForm() {
    var x = document.forms["myForm"]["fname"].value;
    if (x == "") {
        alert("Name must be filled out");
        return false;
    }
}
```

The function can be called when the form is submitted:

HTML Form Example

```
<form name="myForm" action="/action_page_post.php" onsubmit="return
validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

Example: JavaScript Can Validate Numeric Input

JavaScript is often used to validate numeric input:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Can Validate Input</h2>

<p>Please input a number between 1 and 10:</p>

<input id="numb">

<button type="button" onclick="myFunction()">Submit</button>

<p id="demo"></p>
<script>
function myFunction() {
    var x, text;

    // Get the value of the input field with id="numb"
    x = document.getElementById("numb").value;

    // If x is Not a Number or less than one or greater than 10
    if (isNaN(x) || x < 1 || x > 10) {
        text = "Input not valid";
    } else {
        text = "Input OK";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

Objects are Variables Containing Variables

JavaScript variables can contain single values:

Example

```
var person = "John Doe";
```

Objects are variables too. But objects can contain many values.

The values are written as **name : value** pairs (name and value separated by a colon).

Example

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

A JavaScript object is a collection of **named values**

Object Properties

The named values, in JavaScript objects, are called **properties**.

Object Methods

Methods are **actions** that can be performed on objects.

Object properties can be both primitive values, other objects, and functions.

An **object method** is an object property containing a **function definition**.

Creating a JavaScript Object

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

- Define and create a single object, using an object literal.
- Define and create a single object, with the keyword new.
- Define an object constructor, and then create objects of the constructed type.

Using an Object Literal

This is the easiest way to create a JavaScript Object.

Using an object literal, you both define and create an object in one statement.

An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

The following example creates a new JavaScript object with four properties:

Example

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Using the JavaScript Keyword new

The following example also creates a new JavaScript object with four properties:

Example

```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Using an Object Constructor

The examples above are limited in many situations. They only create a single object.

Sometimes we like to have an "object type" that can be used to create many objects of one type.

The standard way to create an "object type" is to use an object constructor function:

Example

```
function person(first, last, age, eye) {
   this.firstName = first;
   this.lastName = last;
   this.age = age;
   this.eyeColor = eye;
}
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
```

Dynamic HTML

DHTML allows scripting languages to change variables in a web page's definition language, which in turn affects the look and function of otherwise "static" HTML page content, after the page has been fully loaded and during the viewing process. Thus the dynamic characteristic of DHTML is the way it functions while a page is viewed, not in its ability to generate a unique page with each page load.

- Dynamic HTML or DHTML, is a collection of technologies used together to create interactive and animated web sites by using a combination of a HTML and CSS or JavaScript.
- DHTML allows authors to add effects to their pages that are otherwise difficult to achieve. For example, DHTML allows the page author to:
- Animate text and images in their document, independently moving each element from any starting point to any ending point, following a predetermined path or one chosen by the user.
- Embed a ticker that automatically refreshes its content with the latest news, stock quotes, or other data.
- Use a form to capture user input, and then process and respond to that data without having to send data back to the server.
- Include rollover buttons or drop-down menus.

JAVASCRIPT - Document Object Model or DOM

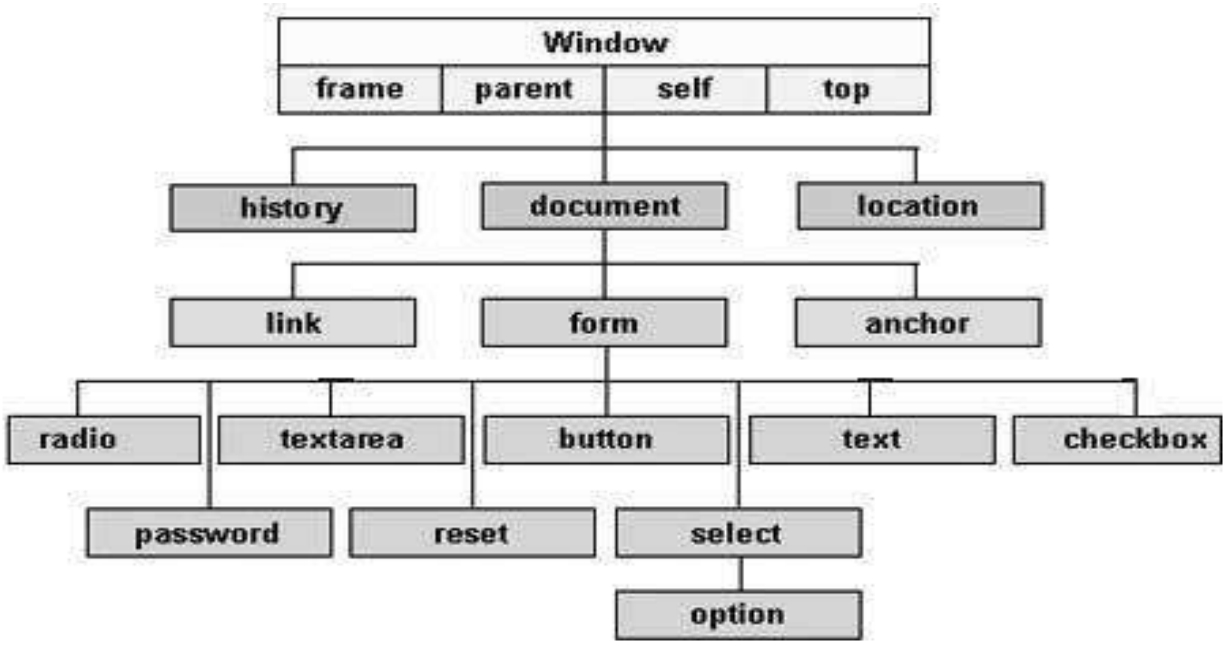
Every web page resides inside a browser window which can be considered as an object.

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object** − Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object** − Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object** − Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements** − The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects −



There are several DOMs in existence. The following sections explain each of these DOMs in detail and describe how you can use them to access and modify document content.

- The Legacy DOM − This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.
- The W3C DOM − This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.

- The IE4 DOM − This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

DOM compatibility

If you want to write a script with the flexibility to use either W3C DOM or IE 4 DOM depending on their availability, then you can use a capability-testing approach that first checks for the existence of a method or property to determine whether the browser has the capability you desire. For example −

```
if (document.getElementById) {
  // If the W3C method exists, use it
} else if (document.all) {
  // If the all[] array exists, use it
} else {
  // Otherwise use the legacy DOM
}
```

JAVASCRIPT - The W3C DOM

This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.

The W3C DOM standardizes most of the features of the legacy DOM and adds new ones as well. In addition to supporting forms[ ], images[ ], and other array properties of the Document object, it defines methods that allow scripts to access and manipulate any document element and not just special-purpose elements like forms and images.

Document Properties in W3C DOM

This model supports all the properties available in Legacy DOM. Additionally, here is a list of document properties which can be accessed using W3C DOM.

Sl.No.	Property & Description
1	body A reference to the Element object that represents the <body> tag of this document. Ex − document.body
2	defaultView Its Read-only property and represents the window in which the document is displayed. Ex − document.defaultView
3	documentElement A read-only reference to the <html> tag of the document. Ex − document.documentElement8/31/2008
4	implementation It is a read-only property and represents the DOMImplementation object that represents the implementation that created this document. Ex − document.implementation

Document Methods in W3C DOM

This model supports all the methods available in Legacy DOM. Additionally, here is a list of methods supported by W3C DOM.

Sl.No.	Property & Description
1	createAttribute(name) Returns a newly-created Attr node with the specified name. Ex − document.createAttribute(name)
2	createComment(text) Creates and returns a new Comment node containing the specified text. Ex − document.createComment(text)
3	createDocumentFragment( ) Creates and returns an empty DocumentFragment node. Ex − document.createDocumentFragment( )
4	createElement(tagName) Creates and returns a new Element node with the specified tag name. Ex − document.createElement(tagName)
5	createTextNode(text) Creates and returns a new Text node that contains the specified text. Ex − document.createTextNode(text)
6	getElementById(id) Returns the Element of this document that has the specified value for its id attribute, or null if no such Element exists in the document. Ex − document.getElementById(id)
7	getElementsByName(name) Returns an array of nodes of all elements in the document that have a specified value for their name attribute. If no such elements are found, returns a zero-length array. Ex − document.getElementsByName(name)
8	getElementsByTagName(tagname) Returns an array of all Element nodes in this document that have the specified tag name. The Element nodes appear in the returned array in the same order they appear in the document source. Ex − document.getElementsByTagName(tagname)
9	importNode(importedNode, deep) Creates and returns a copy of a node from some other document that is suitable for insertion into this document. If the deep argument is true, it recursively copies the children of the node too. Supported in DOM Version 2 Ex − document.importNode(importedNode, deep)

Example

This is very easy to manipulate (Accessing and Setting) document element using W3C DOM. You can use any of the methods like **getElementById**, **getElementsByName**, or **getElementsByTagName**.

Here is an example to access document properties using W3C DOM method.

```
<html>
  <head>
    <title> Document Title </title>
    <script type = "text/javascript">
      <!--
        function myFunc() {
          var ret = document.getElementsByTagName("title");
          alert("Document Title : " + ret[0].text );

          var ret = document.getElementById("heading");
          alert(ret.innerHTML );
        }
      //-->
    </script>
  </head>
  <body>
    <h1 id = "heading">This is main title</h1>
    <p>Click the following to see the result:</p>

    <form id = "form1" name = "FirstForm">
      <input type = "button" value = "Click Me" onclick = "myFunc();" />
      <input type = "button" value = "Cancel">
    </form>

    <form d = "form2" name = "SecondForm">
      <input type = "button" value = "Don't ClickMe"/>
    </form>
  </body>
</html>
```

NOTE

This example returns objects for forms and elements and we would have to access their values by using those object properties which are not discussed in this tutorial.

EXAMPLES FOR JAVASCRIPT - DOM METHODS

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Example

The following example changes the content (the innerHTML) of the <p> element with id="demo":

Example

```
<html>
<body>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

In the example above, getElementById is a **method**, while innerHTML is a **property**.

The getElementById Method

The most common way to access an HTML element is to use the id of the element.

In the example above the getElementById method used id="demo" to find the element.

The innerHTML Property

The easiest way to get the content of an element is by using the innerHTML property.

The innerHTML property is useful for getting or replacing the content of HTML elements.

The innerHTML property can be used to get or change any HTML element,
including <html> and <body>.

Contents Compiled from:
1. https://www.tutorialspoint.com/javascript/javascript_w3c_dom.htm
2. https://www.w3schools.com/js/js_htmldom_methods.asp

SIMPLE AJAX EXAMPLE, DEVELOPING SIMPLE AJAX APPLICATION

Ajax is the method of using JavaScript to send the client data on the server and then retrieve it without refreshing the complete page. We can us the XMLHttpRequest object to perform a GET or POST and then retrieve the server response without page refresh.

Simple Ajax Example

In this tutorial we are going to develop a very simple Ajax Example. This simple Ajax example code will help you in understanding the core concept of Ajax.

In this tutorial we are developing a simple Ajax example application that sends the user name on the server without refreshing the page.

It also process the server response and then display the response text on the same page.

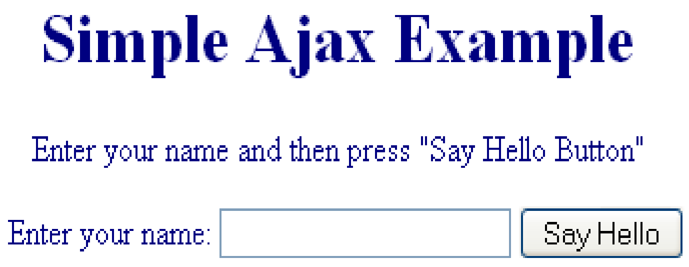
This application is using the XMLHttpRequest object for making a call to the server side script.

We are using PHP script to process the request.

About Simple Ajax Example

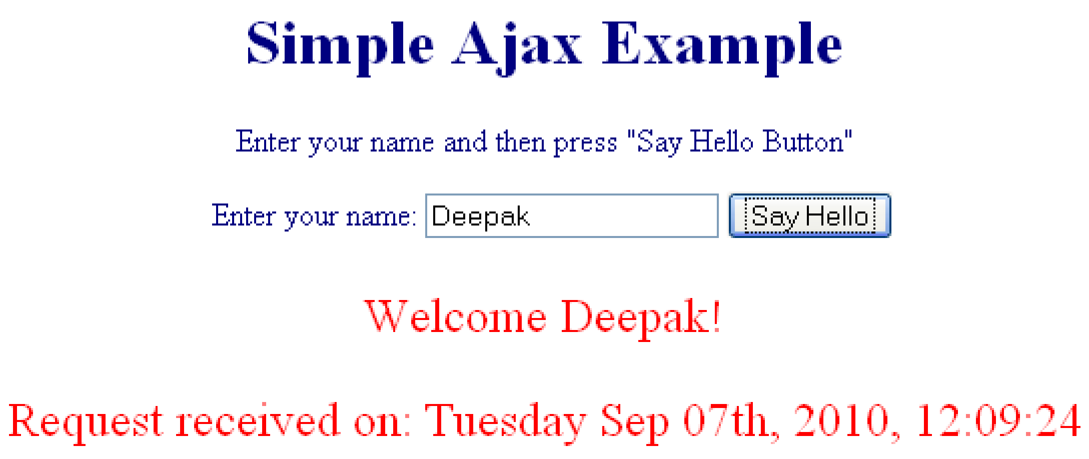
This example will present a form to the user and ask the user to enter his/her name. After entering the name user can press the "Say Hello" button. Then an Ajax call will send the user name on the server to a php file. The php file will return greeting message with the current server date and time. This example will show you how to make Ajax call to a server-side script and then get the data from the server.

The application will display the form. User can then enter then name and press the "Say Hello" button as shown below in the screen shot.


Simple Ajax Example

Enter your name and then press "Say Hello Button"

Enter the name and press the Say Button. Application should send the user name on server and then display the response as shown below:



Simple Ajax Example

Enter your name and then press "Say Hello Button"

Enter your name: Deepak

Welcome Deepak!

Request received on: Tuesday Sep 07th, 2010, 12:09:24

Writing Simple Ajax Example

We have to write the required JavaScript code to make the server call (to call sayhello.php) by passing the user name. Here is the complete code of the simpleajaxexampledemo.html file:

```
<html>
   <head>
   <title>Simple Ajax Example</title>
   <script language="Javascript">
   function postRequest(strURL) {
            var xmlHttp;
       if (window.XMLHttpRequest) { // Mozilla, Safari, ...
                         var xmlHttp = new XMLHttpRequest();
               }else if (window.ActiveXObject) { // IE
                        var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
                 }
               xmlHttp.open('POST', strURL, true);
               xmlHttp.setRequestHeader
          ('Content-Type', 'application/x-www-form-urlencoded');
                        xmlHttp.onreadystatechange = function() {
            if (xmlHttp.readyState == 4) {
                 updatepage(xmlHttp.responseText);
                 }
               }
```

```
                xmlHttp.send(strURL);
        }
     function updatepage(str){
              document.getElementById("result").innerHTML =
           "<font color='red' size='5'>" + str + "</font>";;
        }
   function SayHello(){
              var usr=window.document.f1.username.value;
              var rnd = Math.random();
              var url="sayhello.php?id="+rnd +"&usr="+usr;
              postRequest(url);
        }
  </script>
  </head>
  <body>
   <h1 align="center"><font color="#000080">Simple Ajax Example</font></h1>
   <p align="center"><font color="#000080">Enter your name and then press
   "Say Hello Button"</font></p>
 <form name="f1">
      <p align="center"><font color="#000080">&nbsp;
                          Enter your name: <input type="text" name="username" id="username">
                          <input value="Say Hello" type="button"
      onclick='JavaScript:SayHello()' name="showdate"></font></p>
      <div id="result" align="center"></div>
    </form>
 <div id=result></div>
</body>
</html>
```

We are using the xmlHttp object to send the query to the server using the function xmlHttp.send(strURL).

When the response from the server is ok, calling the updatepage() function.

```
if (xmlHttp.readyState ==
4) {
updatepage(xmlHttp.resp
onseText);
}
```

Finally the in the updatepage function we are adding the response text into the <div> </div>for displaying it to the user.

```
function updatepage(str){
document.getElementById("result").inn
erHTML ="<font color='red' size='5'>"
+ str + "</font>";;
}
```

In the php file we are retrieving the data from user and then sending the appropriate messageto the user. Here is the code of sayhello.php.

```
<?
$usr=$_GET["usr"];
?>
<p>Welcome <?=$usr?>!</p>
<p>Request received on:
<?

print date("l M dS, Y, H:i:s");
?>
</p>
```

In this tutorial we have developed a simple Ajax example.

Contents Compiled from:

1. https://www.roseindia.n