2018-19 Onwards (MR-18)	MALLA REDDY ENGINEERING COLLEGE (Autonomous)	B.Tech. IV Semester		
Code: 80602	WEB TECHNOLOGIES (Common for CSE and IT)	L	T	P
Credits: 3		3	-	-

Prerequisites: Object Oriented Programming
Course Objectives:
This course enables the students to identify the fundamental concepts for developing web application using PHP language for server side scripting, analyze how data can be transported using XML, developing a web application with Server side programming using JavaServlets & JSP and Client side scripting with JavaScript and AJAX.

MODULE– I **[10 Periods]**
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

MODULE – II **[09 Periods]**
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 Parsers in java.

MODULE – III **[10 Periods]**
A: Introduction to Servlets: Common Gateway Interface (CGI), Lifecycle of a Servlet, deploying a servlet.
B: The Servlet API, Reading Servlet parameters, Reading Initialization parameters, Handling Http Request & Responses, Using Cookies and Sessions, connecting to a database using JDBC.

MODULE – IV **[09 Periods]**
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.

MODULE- V **[10 Periods]**
Client side Scripting: Introduction to Javascript: Javascript language – declaring variables,scope of variables, functions, event handlers (onclick, onsubmit etc.), Document Object Model, Form validation. Simple AJAX application.

TEXT BOOKS:
1. Web Technologies, Uttam K Roy, Oxford University Press

2. The Complete Reference PHP – Steven Holzner, Tata McGraw-Hill

REFERENCES:

1. Web Programming, building internet applications, Chris Bates 2nd edition, WileyDreamtech
2. Java Server Pages –Hans Bergsten, SPD O'Reilly
3. Java Script, D. Flanagan, O'Reilly,SPD.
4. Beginning Web Programming-Jon Duckett WROX.
5. Programming World Wide Web, R. W. Sebesta, Fourth Edition, Pearson.
6. Internet and World Wide Web – How to program, Dietel and Nieto, Pearson

E-RESOURCES:

1. https://www.w3schools.com/html/
2. https://www.javatpoint.com/servlet-tutorial
3. https://ndl.iitkgp.ac.in/result?q={%22t%22:%22search%22,%22k%22:%22web%20technologies%22,%22s%22:[],%22b%22:{%22filters%22:[]}}
4. http://nptel.ac.in/courses/106105084/

Course Outcomes:

At the end of the course, students will be able to

1. **Understand** the concepts of client side scripting, validation of forms and AJAX programming
2. **Identify** the role of server side scripting with PHP language.
3. **Create** web pages using XML and explore how to parse and use XML Data with Java.
4. **Design** dynamic web application using Server side programming with Java Servlets and JSP.
5. **Contrast** on how to connect and retrieve data through a web page from database using JDBC.

CO- PO,PSO Mapping
(3/2/1 indicates strength of correlation) 3-Strong, 2-Medium, 1-Weak

COs	**Programme Outcomes(POs)**												**PSOs**		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1			3		2	2				2		3	2	3	3
CO2			3		2	2				2		3	2	3	3
CO3			3		2	2				2		3	2	3	3
CO4			3		2	2				2		3	2	3	3
CO5			3		2	2				2		3	2	3	3

MODULE– 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 PHP code goes here ?>

<?   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
<?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:

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

2nd way:

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

PHP Indexed Array Example

File: array1.php

```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
<?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:

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

2nd way:

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

Example

File: arrayassociative1.php

```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
<?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
<?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
<?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
<?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

```
array array_chunk ( array $array , int $size [, bool $preserve_keys = false ] )
```

Example

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

print_r(array_chunk($salary,2));
?>
```

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

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

Example

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

Output:

```
4
```

5) PHP sort() function

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

Syntax

```
bool sort ( array &$array [, int $sort_flags = SORT_REGULAR ] )
```

Example

```php
<?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

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

Example

```php
<?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
<?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
<?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;
?>
```

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';
```

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

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

```
<?php
$str="Hello text within double quote";
echo $str;
?>
```

Output:

```
Hello text within double quote
```

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

Example 2

```
<?php
$str1="Using double "quote" directly inside double quoted string";
echo $str1;
?>
```

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

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

Output:

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

Example

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

```php
<?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:

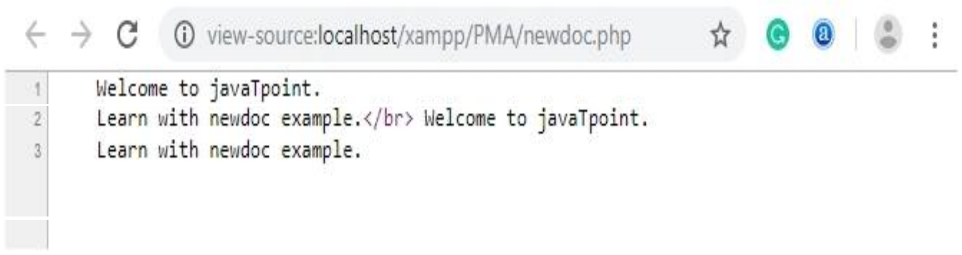
```
My name is "Gunjan". I am printing some DEMO example.
Now, I am printing Example2.
It will print a capital 'A': A
```

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
<?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
<?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

```
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:

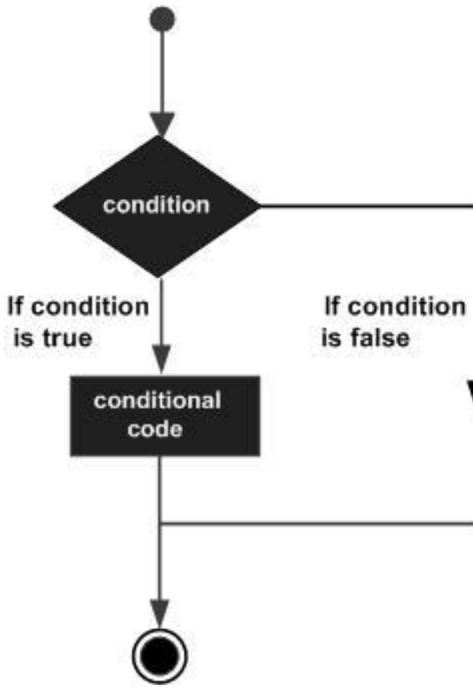
```
a: [1]
b: []
```

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!" −

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

```
    <?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.

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

```
      <?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.

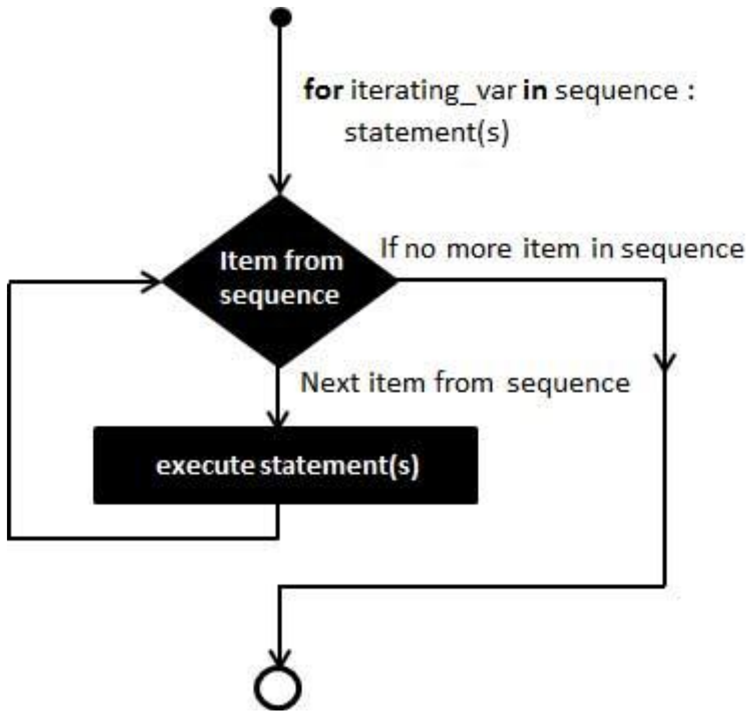
- **for** − loops through a block of code a specified number of times.
- **while** − loops through a block of code if and as long as a specified condition is true.
- **do...while** − loops through a block of code once, and then repeats the loop as long as a special condition is true.

- **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.

The 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>
```

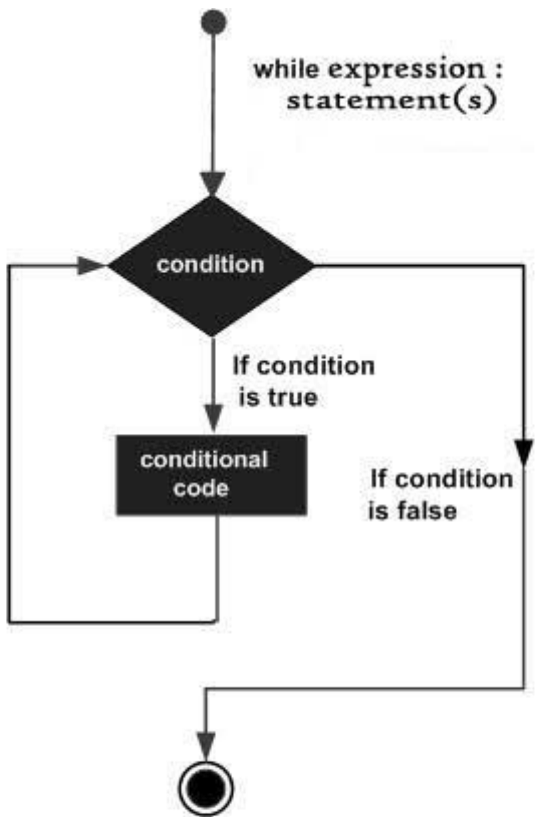
This will produce the following result −

```
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>
```

This will produce the following result −

```
Loop stopped at i = 10
```

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>
```

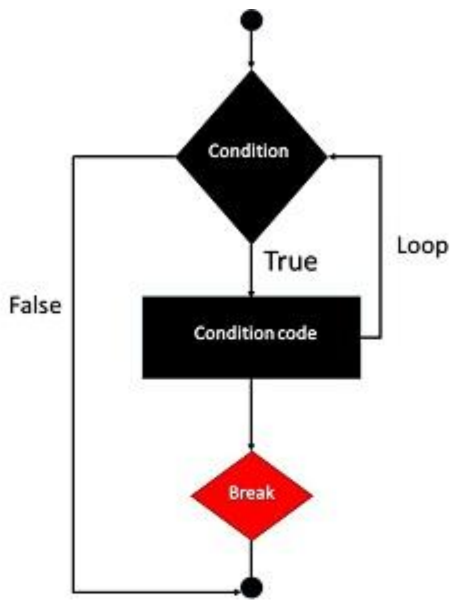
This will produce the following result −

```
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>
```

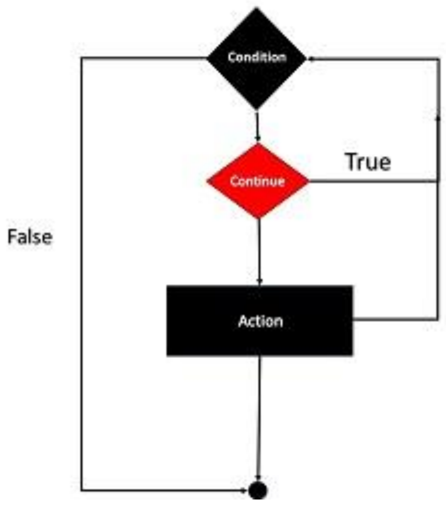
This will produce the following result −

```
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

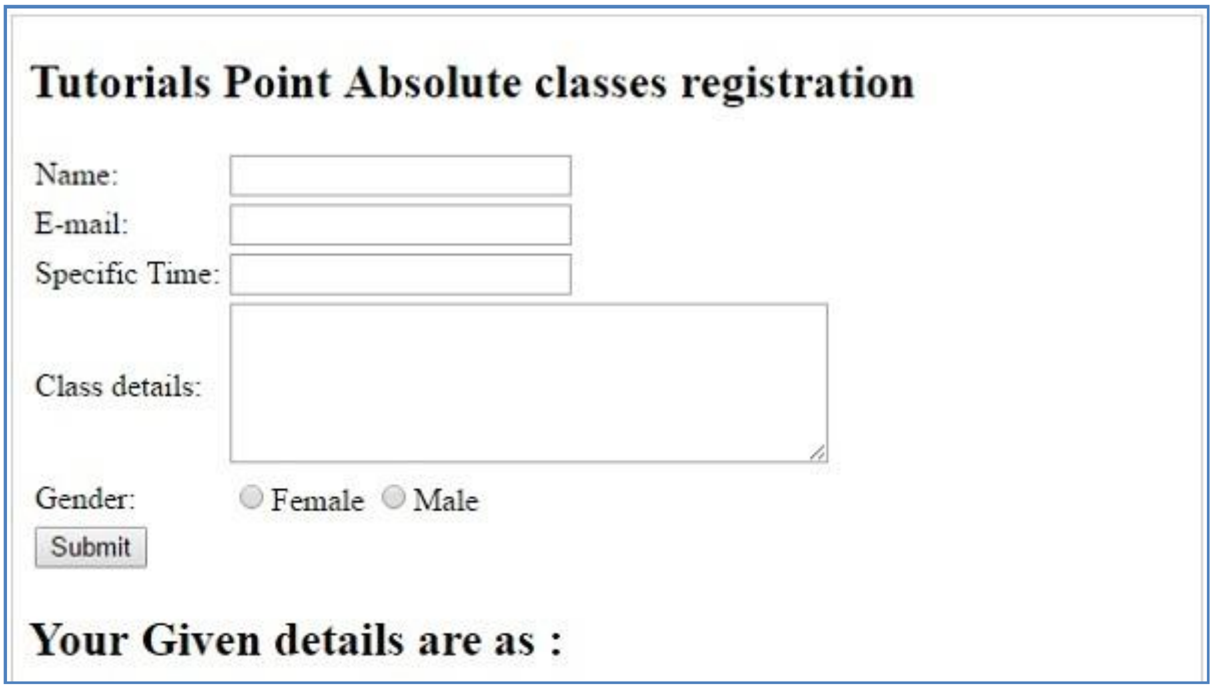
Example

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

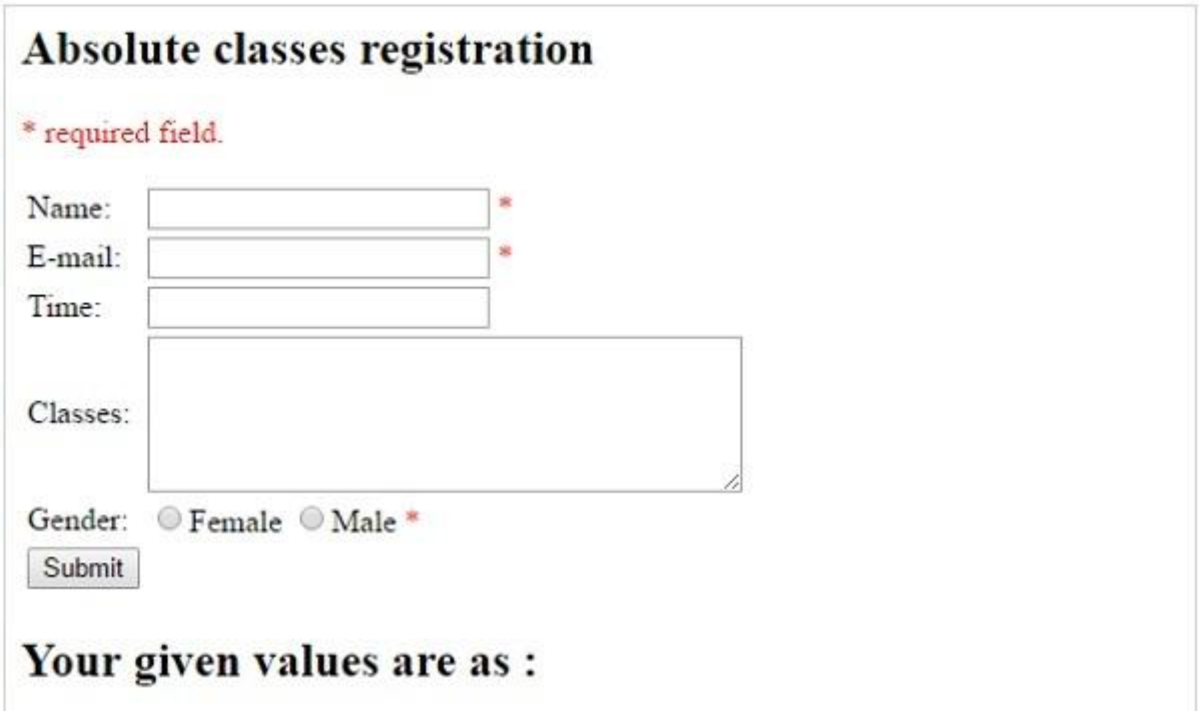
Field	Validation Rules
Name	Should required letters and white-spaces
Email	Should required @ and .
Website	Should required a valid URL
Radio	Must be selectable at least once
Check Box	Must be checkable at least once
Drop Down menu	Must be selectable at least once

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.

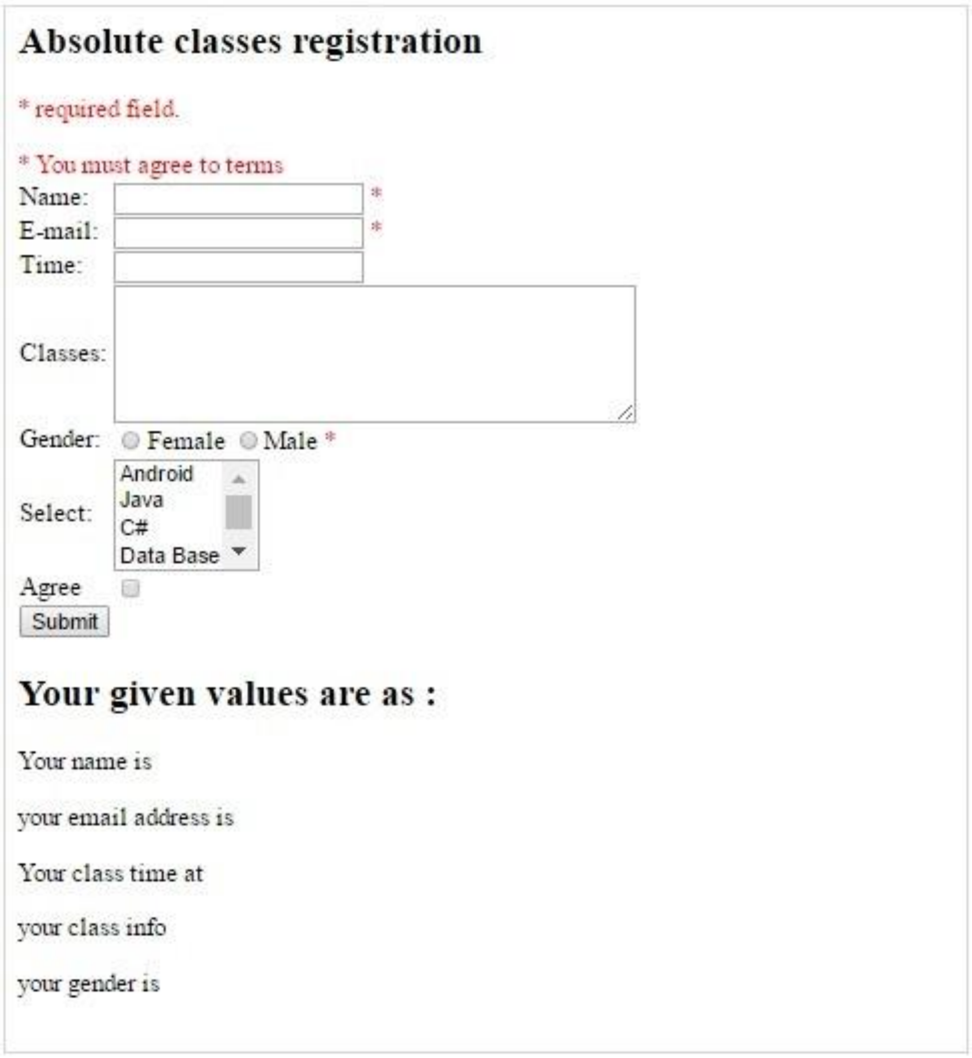
Example

```
<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 –


Absolute classes registration

* required field.

* You must agree to terms

Name: *

E-mail: *

Time:

Classes:

Gender: Female Male *

Select: Android Java C# Data Base

Agree

Submit

Your given values are as :

Your name is

your email address is

Your class time at

your class info

your gender is

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.

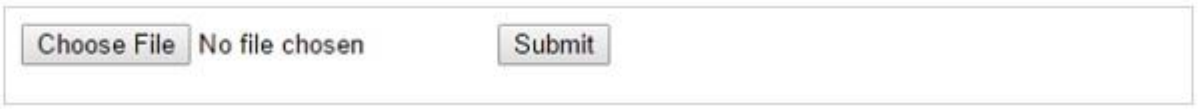
An uploaded file could be a text file or image file or any document.

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 −

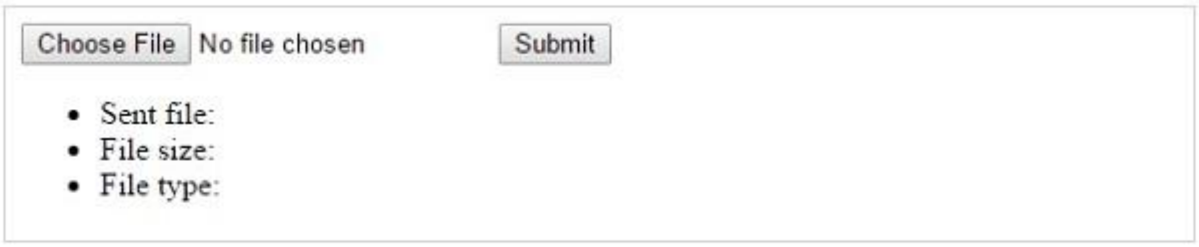
- **$_FILES['file']['tmp_name']** − the uploaded file in the temporary directory on the web server.
- **$_FILES['file']['name']** − the actual name of the uploaded file.
- **$_FILES['file']['size']** − the size in bytes of the uploaded file.
- **$_FILES['file']['type']** − the MIME type of the uploaded 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:

mysqli_fetch_field(*result*)

Example 2: Procedural style

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

```
<?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.

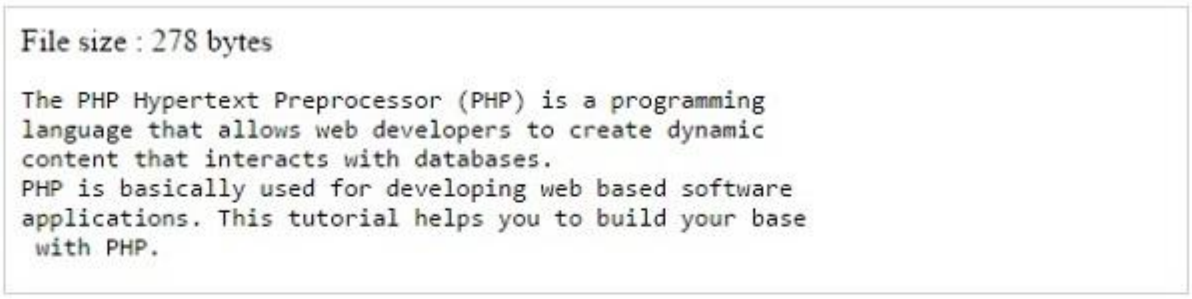
- Open a file using **fopen()** function.
- Get the file's length using **filesize()** function.

- 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.
```

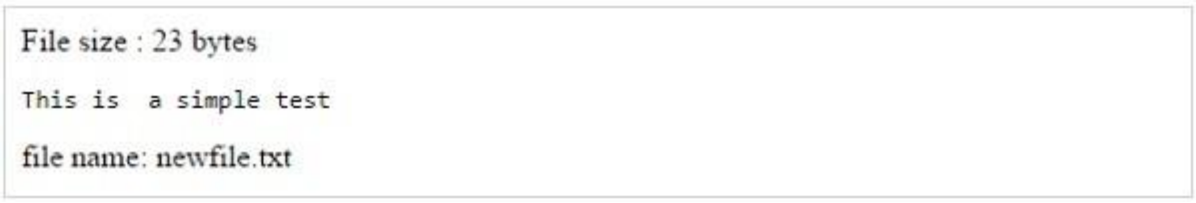
Writing a file

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 [, resour
ce $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
```

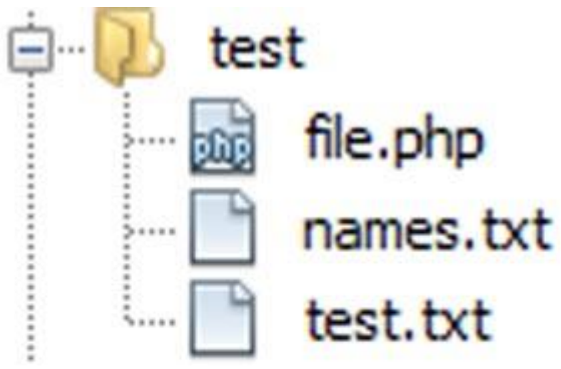
PHP: LIST ALL FILES IN A DIRECTORY

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 want to 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

XML

We have a requirement to save the data with some additional content which can describe the data so that we can further understand and use it and to meet this requirement we used to design our own encoding format and write logic of encoding and decoding the content as a part of our Application Development.

- This makes us to concentrate on low level logic and increases the development time and cost.
- To Solve this problem first IBM has introduced GML(Generalized Markup Language) where GML was used only for the IBM internal purpose i.e,IBM Projects
- A small advancement for GML was given by IBM in the form of SGML(Standard general markup language) but later on SGML is taken by w3C(world wide web consortium) where W3C is an open community
- At this point SGML was be more standardized and was declared standards for developing markup languages

Example:

- HTML is a markup language designed following SGML Standards
- But using SGML for developing a markup language was time taking and complex because of the number of features in it and flexibility
- To solve the above problems and make advantage of markup languages available to even small requirements w3c wants to simplify the standards.
- And as a part of this requirement XML was introduced.

XML:(eXtensible markup language)

- is a meta markup language i.e is a language used to develop some other markup languages
- is a subset of SGML added with some additional services to simplify the language development
- we can say that XML is a restricted form SGML.

Markup language:

are used to describe structured data,it is tag based language which can describe the content which it is enclosing XML-standards for developing markup language

XML Markup language:

A markup language developed according to the standards of XML

i.e following XML standards   Example: XHTML,MathML,CML,VML,WML   etc

XML Dcoument

is a document which is designed following XML and one of the XML markup language standards

To Develop a markup language we require to define the following things.

1. Declare all the elements of the language
    a. i.e tags(elements),attributes,entities.....
2. Define the grammar rules for elements declared
3. An application which can put the document in to action

To perform the first 2 operations we can use DTD or XML Schema which are part of XML Specification.
And to develop an XML Application we can use XML Parsers
which are even standardized under XML specification by W3c   ...i.e parser specifications
where XML Application is an application using XML Document and can be developed using any programing language like JAVA,JAVASCRIPT,C,C++,C#.....

DTD:(Document Type Defination):

is used to declare the elements and give the type definition,where XML document can be designed based on the type defination given by DTD

Using DTD we can declare and define:

I. Elements
II. Attributes
III. Entities
IV. Notations

i)Element

Definition:

Elements are used to describe the content which it encloses

Types of Elements:

i)child only
ii)Text only
iii)Empty
iv)Mixed
v)ANY(is a special type)

i)Child only:

these type of elements consists of one or more elements as a contents

Syntax:

```
<!ELEMENT elemnet_name(list of child element names)>
```

Example:

```
<account>
<name>       </name>
<bal>        </bal>
</account>
<!ELEMENT account(name,bal)>
```

Example2:

```
<bank>
<account>  </account>
<account>  </account>
</bank>
<!ELEMENT bank(account*)>
```

occurence Specifiers

*   indicate 0 or More

+   indiactes 1 or More

?   indicates 0 or 1

No Symbol   ----only for one time

Example:

```
<emps>
<emp>
<name>   </name>
<sal>    </sal>
</emp>

<emp>
<name>                         </name>
<wages>                        </wages>
</emp>
</emps>
<!ELEMENT emps(emp+)>
<!ELEMENT emp(name,(sal|wages))>
```

ii)Text only:

These type of elements can take only text as a content where char,string,int,float,double,boolean...are considered as a text. and are refered with a type PCDATA
PCDATA:Parsed character DATA

Syntax:
```
<!ELEMENT element_name(#PCDATA)>
```
Example:
```
<name>MREC</name>
<!ELEMENT name(#PCDATA)>
<sal>1000</sal>
<!ELEMENT sal(#PCDATA)>
```
PCDATA allows all the characters of our encoding format except markup char like <..

iii)Empty:
These type of elements does not takes any content
Syntax:
```
<!ELEMENT element_name EMPTY>
```
Example:
```
<br> </br>
or
<br/>
<!ELEMENT br EMPTY>
```
iv)Mixed:
These type of elements can contain child elements or text or child elemnets and text or even it can be empty
Syntax:
```
<!ELEMENT element_name(#PCDATA|list of child elements with | as a separator)*>
```
Example:
```
<p>Welcome,<b>to MREC </b> and <i>B.Tech(CSE)</i><br/>Hello
</p>
<!ELEMENT p(#PCDATA|b|i|br)*>
```
v)ANY
These type of elements can take any type of content i.e:text or can be empty or any element declared in the document
Syntax:
```
<!ELEMENT element_name ANY>
```
Example:
```
<!ELEMENT MyElement ANY>
```

The above declaration describes that element MyElement can hold text and even any element declared in the document and it can be empty also

2. Attributes:

Are used to give a extra meaning for the content described by element

- Attribute resides in the opening tag of the element
- One element can be declared with any number of attributes,where element name and each of these attributes are separated with space character.
- Each of the attribute consist of one name and value where these are separated with '=' character and value should be in quotes ' or "(Single quotes or double quotes)
- Attribute name cannot have a space character

Example:

<emp empno="e101">

Syntax to declare an attribute:

<!ATTLIST element_name attribute_name type specifier[defaultvalue]>

Types:

1. CDATA(character data):
   This type allows all the characters including numbers and space character
2. NMTOKEN:
   is same as CDATA but does not accept space character
3. NMTOKENS:
   it accepts one or more tokens(where one token is a sequence of characters without space character) and in this case space is taken as separator between tokens
4. ID:The value of ID type attribute should be unique
   it should not start with number but it contain number
5. IDREF:it allows one of the ID type attribute value
6. IDREFS:it can take one or more ID type attribute values where space is the separator
7. enum:in this case while declaring attribute we will specify the list of values and it allows to use any one of the specified value.
8. ENTITY:it allows one entity name where this entity should be umparsed entity
9. ENTITIES:allows one or more entity names where space is the separator

Example:

<!ATTLIST empno working(yes|no) 'yes'>

Specifiers:

#REQUIRED --------------- Mandatory

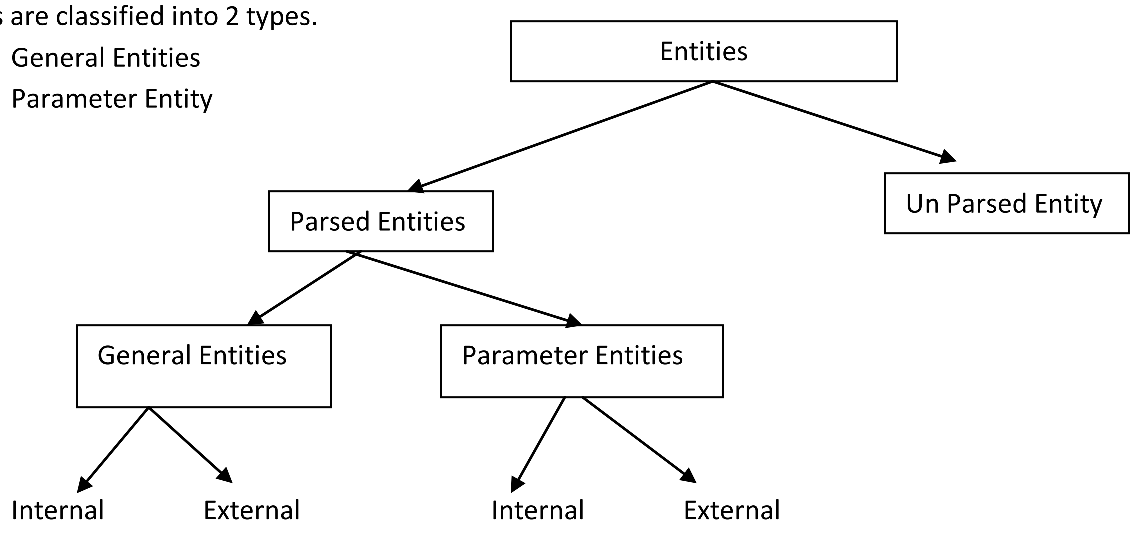
#IMPLIED ---------------Optional

#FIXED ------------- -is Optional and even if it is used it has to be given with the value which is specified while declaring the attribute(i.e its value will be fixed)same as final in java

3)Entity:

is reference to some content.i.e is used to represent some reusable content.we have a requirement where some content is required to be used for more number of times within the XML documents and even in some cases we have content being repeated in DTD document also based on this requirement Entities are classified into 2 types.

1. General Entities
2. Parameter Entity



General Entities:

Are declared in DTD and used in XML documents

Internal Entity:

In this casethe content which has to be replaced where ever the entity is refered,will be placed in the declaration of the entity directly i.e in DTD document itself.

Syntax:

<!ENTITY entity_name "content which had to replaced">
To use the entity
this can done in XML document
&entity_name

Example:

<!ENTITY copyrights "copyrights Myshop 2013-2014">

External Entity:

Here the content which has to be replaced will be placed in separate file and in the declaration of the entity insted of specifying the content we will provide the filename with its path.

Syntax:

<!ENTITY enitity_name SYSTEM "filename with path">

Example:

<!ENTITY mylogo SYSTEM "shoplogo.gif">

Parameter Entity:

These entities are declared and used in DTD itself

Internal entity:
Syntax:

<!ENTITY % entity_name "content">
to use (i.e in DTD it self)
%entity_name

External Entities:

Syntax:
<!ENTITY % entity_name SYSTEM "filename">
to use
%entity_name
Example:

<!ENTITY % text "#PCDATA">
<!ELEMENT name(%text;)>

Unparsed Entities:

To refer some content which is of different encoding format we have to go for unparsed entities

i.e like to refer gif,.bmp,.jpg......files

Syntax:

<!ENTITY entity_name SYSTEM "filenamewith path" NDATA notation_name>

Notations:

These are used to refer some content which provides some additional description like MIME/Contenttype ....

Syntax:

<!NOTATION notation_name "content">

Example:

<!NOTATION gif "image|gif">

<!ENTITY mylogo SYSTEM "shoplogo.gif" NDATA gif>

<!ATTLIST shop logo ENTITY #REQUIED>

Example:

```
<emps logo="mylogo">
```

To associate the definitions to the XML document i.e the definitions which are given following DTD Standards. We use DOCTYPE element

There are 2 Types of DTD

1. Internal DTD
2. External DTD

Internal DTD:

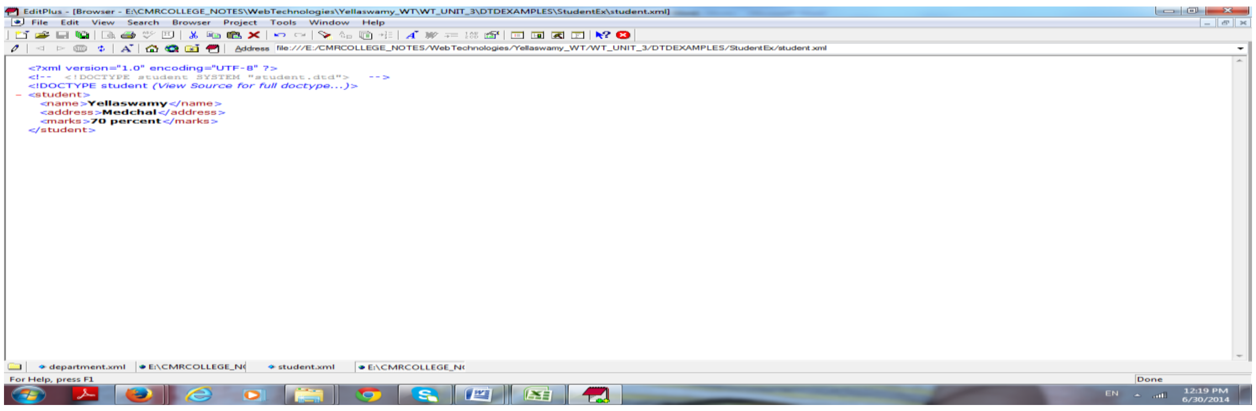
Here the DTD Content is placed inside the XML document.

```
<!DOCTYPE  root_element_name [DTD code]>
```

Example:Student.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- <!DOCTYPE student SYSTEM "student.dtd"> -->
<!DOCTYPE student [<!ELEMENT student (name,address,marks)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT marks (#PCDATA)>]>
<student>
<name>Yellaswamy</name>
<address>Medchal</address>
<marks>70 percent</marks>
</student>
```

Output:



```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- <!DOCTYPE student SYSTEM "student.dtd"> -->
<!DOCTYPE student (View Source for full doctype...)>
- <student>
    <name>Yellaswamy</name>
    <address>Medchal</address>
    <marks>70 percent</marks>
  </student>
```

External DTD:

Here the DTD code is written in to a separate file and referred by the XML document.

<!DOCTYPE root_element_name SYSTEM “dtd file name with path”>

<!DOCTYPE root_element_name PUBLIC “fpi string” “dtd file url”>

FPI (Formal public identifier):

This String gives some information about the vendor and dtd which we are referring

This String is divided into 4 parts and these parts are separated with //

1. Part takes + or –
2. Part takes the company name or the person name who developed the DTD and responsible for the DTD
3. Part the purpose and version of DTD
4. Part the 2 letter language code(i.e the codes given under the ISO standards)
   Example:
   EN for English

Example:

-//MREC //Examples DTD 1.0//EN

Example for External DTD

Department.dtd

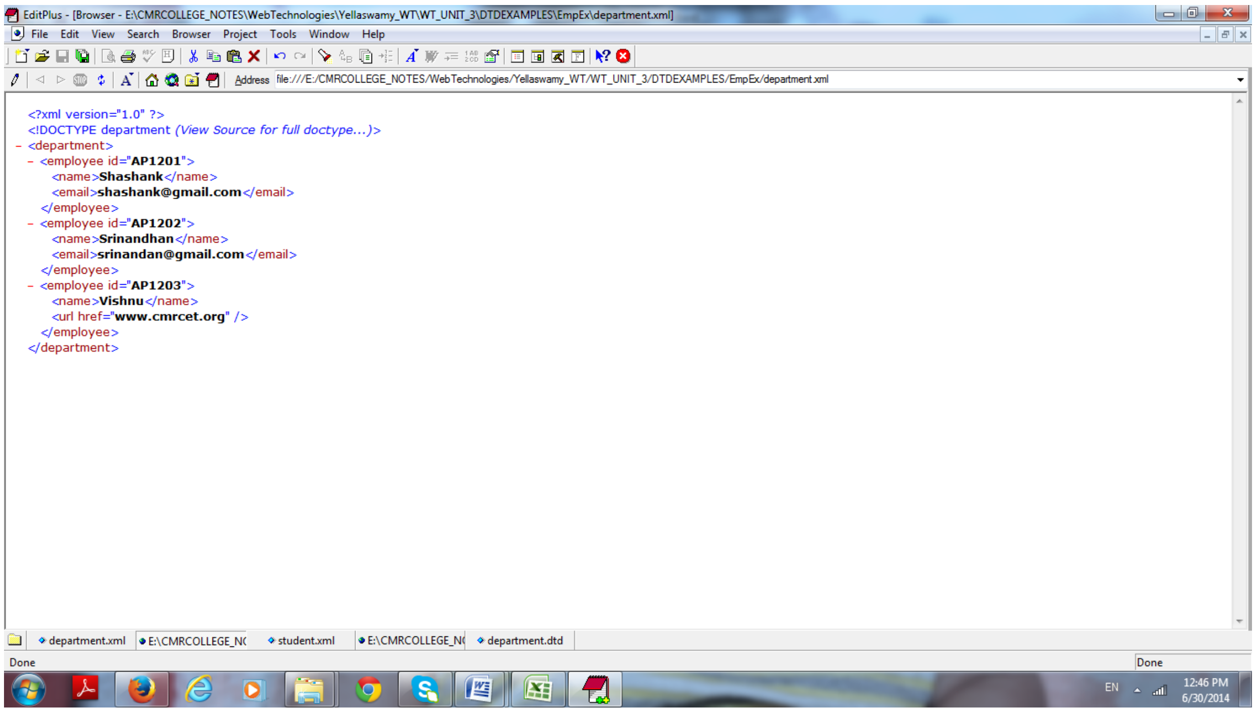
```
<!ELEMENT department (employee)*>
<!ELEMENT employee (name, (email | url))>
<!ATTLIST employee id CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT url EMPTY>
<!ATTLIST url href CDATA #REQUIRED>
```

Department.xml

```
<?xml version="1.0"?>
<!DOCTYPE department SYSTEM "department.dtd">
<department>
```

```
  <employee id="AP1201">
   <name>Shashank</name>
   <email>shashank@gmail.com</email>
  </employee>

  <employee id="AP1202">
   <name>Srinandhan</name>
   <email>srinandan@gmail.com</email>
  </employee>

  <employee id="AP1203">
   <name>Vishnu</name>
   <url href="www.mrec.ac.in"/>
  </employee>
</department>
```

OUTPUT:



```
  <?xml version="1.0" ?>
  <!DOCTYPE department (View Source for full doctype...)>
- <department>
  - <employee id="AP1201">
      <name>Shashank</name>
      <email>shashank@gmail.com</email>
    </employee>
  - <employee id="AP1202">
      <name>Srinandhan</name>
      <email>srinandan@gmail.com</email>
    </employee>
  - <employee id="AP1203">
      <name>Vishnu</name>
      <url href="www.cmrcet.org" />
    </employee>
  </department>
```

Combination of Internal and External:

<!DOCTYPE root_element_name SYSTEM “external dtd file path” [internal DTD code]>

XML Document Structure

<?	?>	Processing Instruction Tag
<!	>	Instruction Tag
<!--	-->	Comment Instruction Tag
<	>	opening tag
</	>	closing Tag
<	/>	Self ending tag

Structure of XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE …….>
<root_element_name>
</root_element_name>
```

- XML processing instruction tags is optional but recommended to be used
- If used should be the first element (even comment is not allowed before this)
- If used **version** attribute is mandatory this takes the XML version where the current version is 1.0 **encoding** is optional if not given takes the system encoding format **standalone** is also optional takes yes or No if not given by default it takes No.This attribute indicates whether this document depends on any external resources or not(if it is depending it should be given as 'no' if not 'yes'

XML Schema:

Is used to declare the elements of the Markup Language and Grammar rules i.e an alternative to DTD

An XML Schema describes the structure of an XML document.The XML Schema language is also referred to as XML Schema Definition (XSD)

An XML Schema:

- Defines elements that can appear in a document
- Defines attributes that can appear in a document
- Defines which elements are child elements
- Defines the number of child elements
- Defines whether an element is empty or can include text
- Defines data types for elements and attributes
- Defines default and fixed values for elements and attributes

Differences Between DTD and XML Schema

- DTD uses a small language to define the rules where as xml schema is xml document.XML schema documents are more descriptive than compared to DTD
- With DTD &XML Schemas we have a provision to declare complex types but with DTD the type name and the element name should be same which is not required in XML Schema
- With DTD we don't have a support to specify a particular occurrence for a element i.e MIN and MAX occurrence(We were allowed to given MIN as 0 or 1 and MAX 1 or more) where as with XML Schema we can specify the required Max and Min occurrences.
- DTD doesn't supports all the common types(i.e it considers numbers.. all as text #PCDATA) where as with XML Schema we can specific type like String,char,number,double,float,Boolean
- XML schema supports NameSpace.Since XML Schema document is also an XML document it can be generated/written using any tool which supports

XML Schemas are the Successors of DTDs

We think that vey soon XML Schemas will be used in most Web Applications as a replacement for DTDs.

Here are Some reasons:

- XML Schemas are extensible to future additions
- XML Schemas are richer and more useful than DTDs
- XML Schemas are written in XML
- XML Schemas support dat types
- XML Schemas support namespaces

XML Schema has support for Data Types

One of the greatest strengths of XML Schema is the Support for data types

With the support for data types:

- It is easier to describe permissible document content
- It is easier to validate the correctness of data
- It is easier to work with data from a database
- It is easier to define facets(restrictions on data)
- It is easier to define data patterns
- It is easier to convert data between different data types

XML Schemas Secure Data Communication:

When data is sent from sender to a receiver it is essential that both parts have the same "expectations" about the content.

With XML Schemas,the sender can describe the data in way that the receiver will understand.

Well-Formed is not enough

A well-formed XML document is a document that conforms to the XML syntax rules:

- Must begin with the XML declaration
- Must have one unique root element
- All start tags must match end tags
- XML tags are case sensitive
- All elements must be closed
- All elements must be properly nested
- All attribute values must be quoted
- XML entities must be used for special characters

Even if documents are well-Formed they can still contain errors and those errors can have serious consequences. Think of this situation: you order 5 gross of laser printers, instead of 5 laser printers. With XML Schema most of these errors can be caught by your validating software.

A simple XML Document

“note.xml”

```
<?xml version="1.0"?>
<note>
<to>Srinandhan</to>
<from>shashank</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

A simple DTD

This simple DTD file called “**Note.dtd”** that defines the elements of the XML document above(“note.xml”)

```
<!ELEMENT note(to,from,heading,body)>
<!ELEMENT to(#PCDATA)>
<!ELEMENT from(#PCDATA)>
<!ELEMENT heading(#PCDATA)>
<!ELEMENT body(#PCDATA)>
```

The <schema> Element:

The <schema> Element is the root element of every XML Schema

Syntax:

```
<?xml  version=”1.0”?>
```

```
<xs:schema>
----
----
</xs:schema>
```

The <schema> Element may contain some attributes. A schema declaration often looks something like this:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com" xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
--
---

</xs:schema>
```

A simple XML Schema

This simple XML Schema file called “Note.xsd” that defines the elements of the XML document above(“note.xml”)

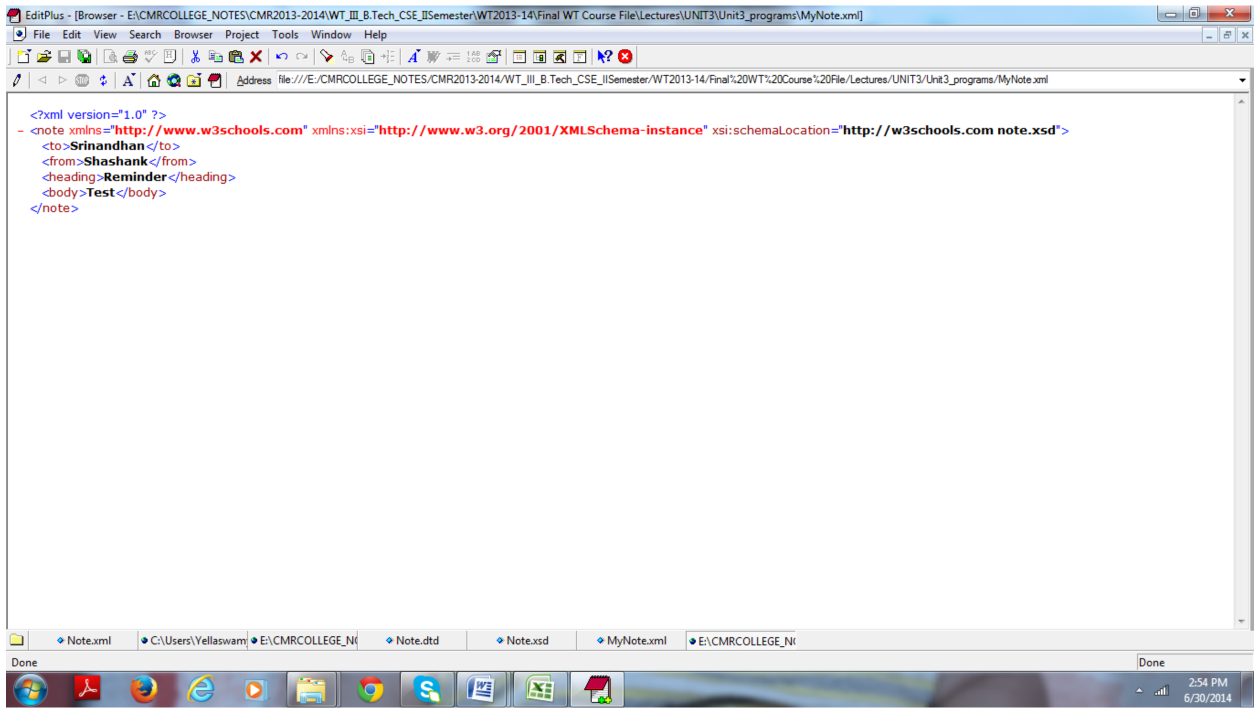
“Note.xsd”

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com" xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xs:element name="note">
<xs:complexType>
<xs:sequence>
<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
<xs:element name="heading" type="xs:string"/>
<xs:element name="body" type="xs:string"/>
</xs:sequence>
</xs:complexType>

</xs:element>
</xs:schema>
```

A reference to an XML Schema:

```
        <?xml version="1.0"?>
        <note xmlns="http://www.w3schools.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
        instance" xsi:schemaLocation="http://w3schools.com note.xsd">
        <to>
        Srinandhan
        </to>
        <from>Shashank</from>
        <heading>Reminder</heading>
        <body>Test</body>
</note>
```

Output:


EditPlus - [Browser - E:\CMRCOLLEGE_NOTES\CMR2013-2014\WT_III_B.Tech_CSE_IISemester\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\MyNote.xml]

Address file:///E:/CMRCOLLEGE_NOTES/CMR2013-2014/WT_III_B.Tech_CSE_IISemester/WT2013-14/Final%20WT%20Course%20File/Lectures/UNIT3/Unit3_programs/MyNote.xml

```
  <?xml version="1.0" ?>
- <note xmlns="http://www.w3schools.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://w3schools.com note.xsd">
    <to>Srinandhan</to>
    <from>Shashank</from>
    <heading>Reminder</heading>
    <body>Test</body>
  </note>
```

Namespace:

Namespace is used to make the element/attribute unique

i.e this is most required when multiple markup language elements are used in one document in such a case if the element names are same from both the markup languages then a small prefix can represent a element uniquely describing that the element is of a particular markup language.

Types of Namespaces

1. General Namespace
2. Default Namespace

Declaring a Namespace:

Namespace is declared as an attribute in the element.

Where the value will be the unique URI given by the markup language provider.

Namespace declared has a scope within that element including that element i.e the namespace declared can be used for that element and its child and its Childs.

But not applicable for its parent or siblings

To declare General namespace:

xmlns:<namespace_name>="<namespace uri>"

Where

<namespace_name> can be any name without special characters and space this is used as a prefix for the elements/attributes.

<Namespace uri>---is given by the markup language provider whose elements we wanted to refer.

To Declare Default Namespace:

xmlns="<namespace uri>"

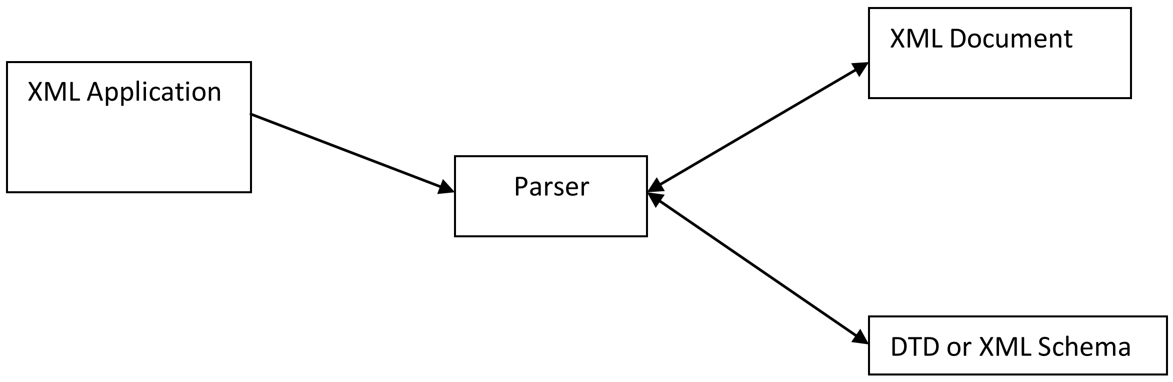
in this case we dont have any prefix and if default namespace is declared then all the unqualified elements (i.e. the elements without any prefix) within the scope will be considered under the default namespace.

XML Parsers:

Parser is a standard abstraction between the xml application and xml document.

The responsibility of parser is:

- should read the given xml document
    - if available read the schema definition(i.e DTD or XML Schema)
- check the XML document (i.e validation)
- Makes the XML Content available to the application



Types of Parsers:

1. Based on the type definition syntax it understands
    a. DTD Validator
    b. XML Schema Validator
2. Based on the Validation
    a. Non Validating Parser
    b. validating parser
3. Based on the approach of making the data available to the application
    a. Tree based approach(object based)
    b. Event based approach

Non Validating Parsers:

These type of parsers checks only wellformness of the Document.

where if xml document follows the following rules then it is said to be well formed document.

Rules

1. only one root element is allowed.
2. every opening tag should have a closing tag.
3. All the elements should be properly arranged in tree structure i.e, first child tag has to be closed and then parent tag.
4. All the attributes values should be in quotes
   i.e single or double quotes
5. All the entities used in the document should be declared.

Validating Parsers:

These parsers checks for wellformness and then if it is well formed then it checks the xml document following the grammar rules given under the DTD or XML Schema.

JAXP (Java API for XML Processing)

JAXP API

The Main JAXP API are defined in the javax.xml.parsers package This package contain vendor neutral factory classes

- SAXParserFactory
- DocumentBuilderFactory
- TransformerFactory

javax.xml.parsers:

The JAXP API,which provides a common interface for different vendors SAX and DOM Parsers

org.w3c.dom:

Defines the Document class as well as classes for all the components of a DOM

org.xml.sax:

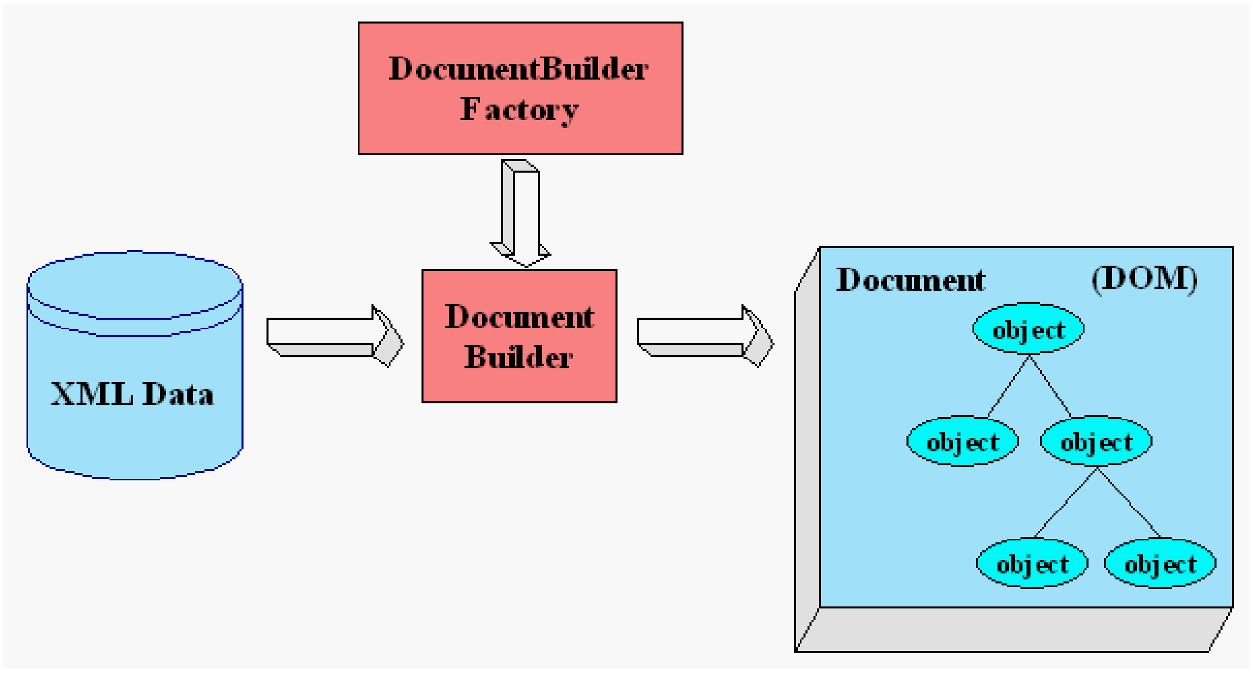
Defines the basic SAX API

javax.xml.transform:

Defines the XSLT API that let you transform XML into other forms

DOM (Document Object Model)

- Is a Specification for w3c
- is a validating parser
- DOM is a DTD Validator and DOM Level 3 parser supports XML Schema also
- is a tree based i.e it follows tree based approach(Makes the complete object tree available to the application)
- for each part of the xml document it prepares an object and construct an object tree representing the xml document and org.w3c.dom.Node is the super most type for all the types in DOM Specification
- These Specifications are implemented by 3rd party vendors



You use the `javax.xml.parsers.`DocumentBuilderFactory class to get a DocumentBuilder instance, and use that to produce a Document (a DOM) that conforms to the DOM specification. The builder you get, in fact, is determined by the System property, `javax.xml.parsers.DocumentBuilderFactory`, which selects the factory implementation that is used to produce the builder. (The platform's default value can be overridden from the command line.)

You can also use the DocumentBuilder `newDocument()` method to create an empty Document that implements the
org.w3c.dom.Document interface. Alternatively, you can use one of the builder's parse methods to create a Document
from existing XML data. The result is a DOM tree like that shown in the diagram.

Example:

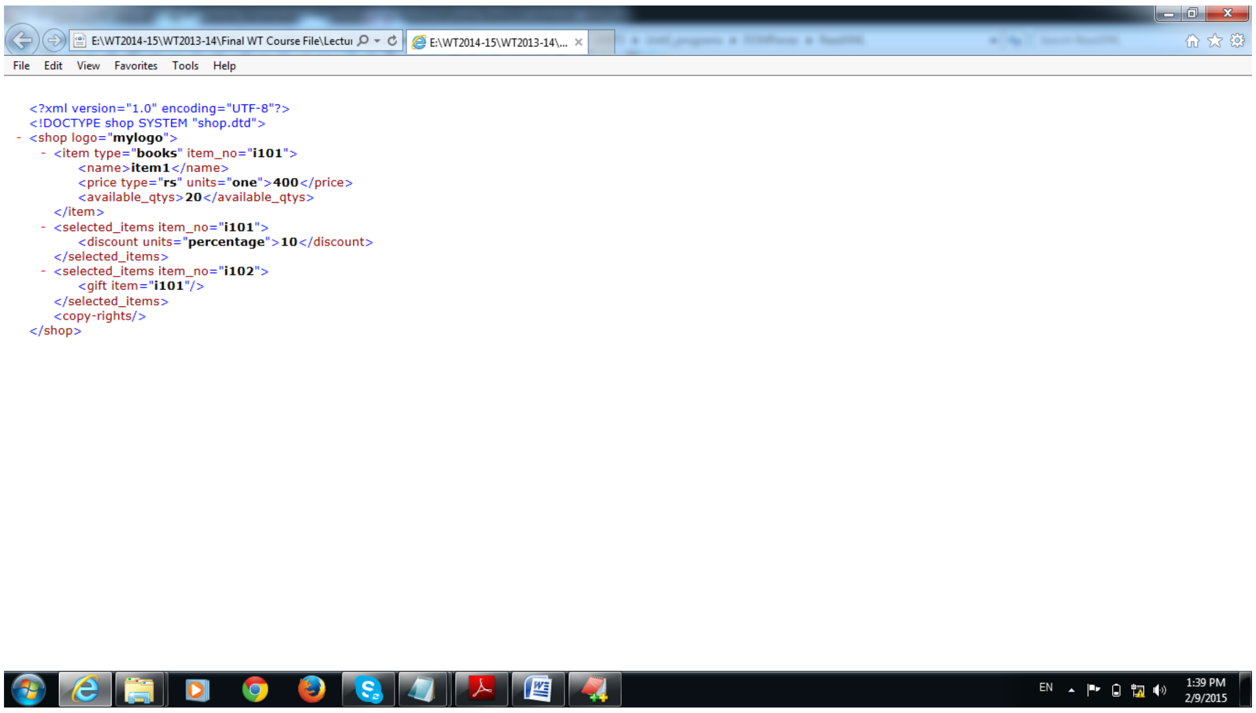
Shop.dtd

```
<!ENTITY copyrights "copyrights Myshop 2012-2013">
<!NOTATION gif SYSTEM "image|gif">
<!ENTITY mylogo SYSTEM "shoplogo.gif" NDATA gif>
<!ELEMENT shop (item+,selected_items*,copy-rights)>
<!ELEMENT item (name,price,available_qtys)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT available_qtys (#PCDATA)>
<!ELEMENT selected_items (discount?,gift*)>
<!ELEMENT discount (#PCDATA)>
<!ELEMENT gift EMPTY>
<!ELEMENT copy-rights (#PCDATA)>
<!ATTLIST shop logo ENTITY #IMPLIED>
<!ATTLIST item item_no ID #REQUIRED>
<!ATTLIST item type CDATA #IMPLIED>
<!ATTLIST discount units CDATA #REQUIRED>
<!ATTLIST price units (one|kg|meter) 'one' type NMTOKEN  #IMPLIED>
<!ATTLIST selected_items item_no IDREFS #REQUIRED>
<!ATTLIST gift item IDREF #REQUIRED>
```

Shop.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE shop SYSTEM "shop.dtd">

<shop logo="mylogo">
<item item_no="i101" type="books">
<name>item1</name>
<price units="one" type="rs">400</price>
<available_qtys>20</available_qtys>
</item>
<selected_items item_no="i101">
<discount units="percentage">10</discount>
</selected_items>
<selected_items item_no="i102">
<gift item="i101"/>
</selected_items>
<copy-rights>&copyrights;</copy-rights>
</shop>
```

Output:


E:\WT2014-15\WT2013-14\Final WT Course File\Lectur
E:\WT2014-15\WT2013-14\...
File Edit View Favorites Tools Help
EN
1:39 PM
2/9/2015

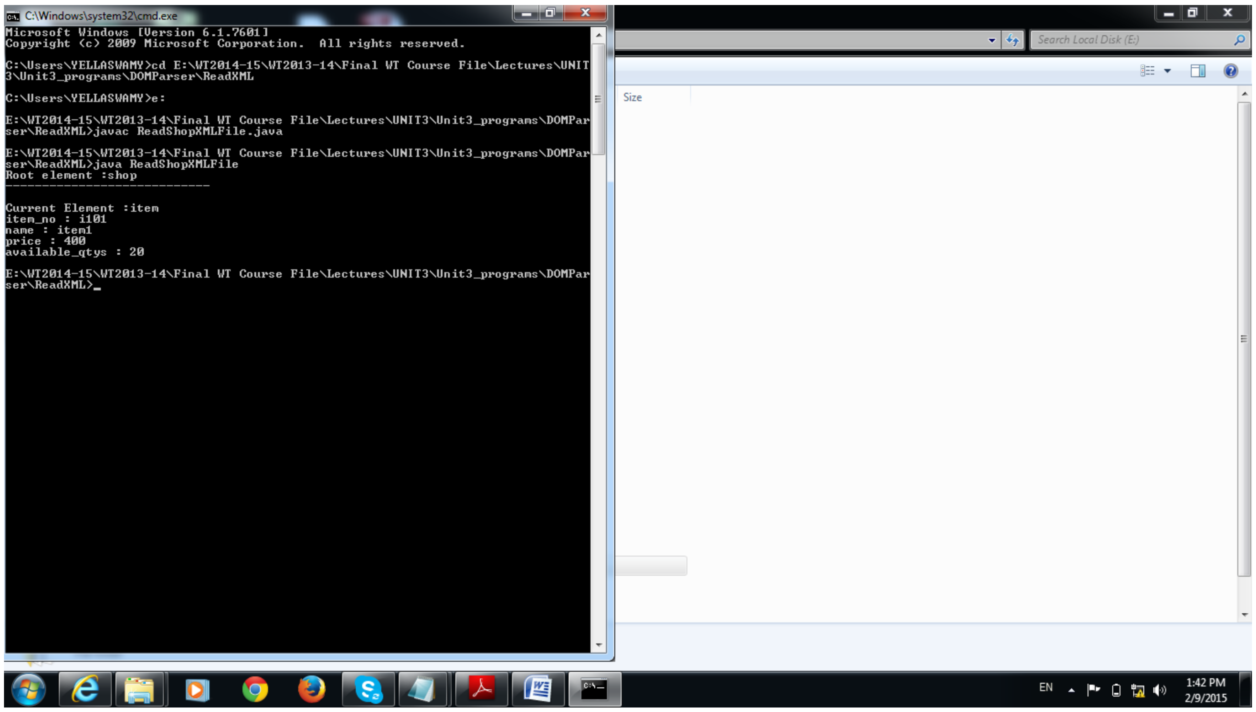
```
  <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE shop SYSTEM "shop.dtd">
- <shop logo="mylogo">
  - <item type="books" item_no="i101">
        <name>item1</name>
        <price type="rs" units="one">400</price>
        <available_qtys>20</available_qtys>
    </item>
  - <selected_items item_no="i101">
        <discount units="percentage">10</discount>
    </selected_items>
  - <selected_items item_no="i102">
        <gift item="i101"/>
    </selected_items>
    <copy-rights/>
  </shop>
```

ReadShopXMLFile.java

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import java.io.File;

public class ReadShopXMLFile
        {

  public static void main(String argv[]) {

    try {

                File fXmlFile = new File("shop.xml");
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(fXmlFile);


        doc.getDocumentElement().normalize();

        System.out.println("Root element :" + doc.getDocumentElement().getNodeName());

        NodeList nList = doc.getElementsByTagName("item");

        System.out.println("----------------------------");

        for (int temp = 0; temp < nList.getLength(); temp++)
                {

                Node nNode = nList.item(temp);

                System.out.println("\nCurrent Element :" + nNode.getNodeName());

                if (nNode.getNodeType() == Node.ELEMENT_NODE) {

                        Element eElement = (Element) nNode;

                        System.out.println("item_no : " + eElement.getAttribute("item_no"));
                        System.out.println("name : " +
eElement.getElementsByTagName("name").item(0).getTextContent());
                        System.out.println("price : " +
eElement.getElementsByTagName("price").item(0).getTextContent());
                        System.out.println("available_qtys : " +
eElement.getElementsByTagName("available_qtys").item(0).getTextContent());
```

```
                                        //System.out.println("Salary : " +
//eElement.getElementsByTagName("salary").item(0).getTextContent());

                              }
                     }
     } catch (Exception e) {
              e.printStackTrace();
     }
   }

}
```

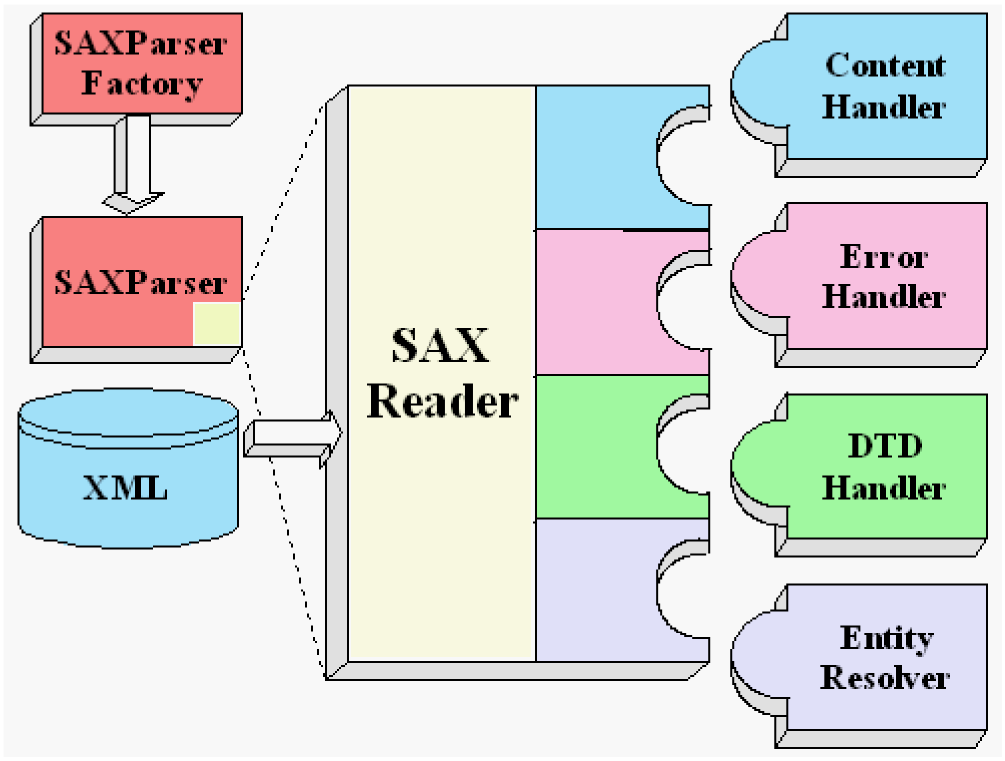
Output:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\YELLASWAMY>cd E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT
3\Unit3_programs\DOMParser\ReadXML

C:\Users\YELLASWAMY>e:

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\DOMPar
ser\ReadXML>javac ReadShopXMLFile.java

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\DOMPar
ser\ReadXML>java ReadShopXMLFile
Root element :shop
----------------------------

Current Element :item
item_no : i101
name : item1
price : 400
available_qtys : 20

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\DOMPar
ser\ReadXML>_
```

SAX (Simple API for XML)

- is used for Simple Search operations
- Follows Event Based Approach
- SAX1.0 is a non-validating parser where as 2.0 can be used for validating also
- These parsers are used for read-only purpose
- in this case at movement when parser reads the data it pushes the data in to the application allowing the application to take decision to store the data or not.
- It reads the content in a forward direction
- once the XML document submitted to the parser it validates the document.
- if its valid then starts processing where it gives the notification of each of the part of the document which it is reading.like startElement,endElement.



Here is a summary of the key
SAX APIs:
`SAXParserFactory`
A SAXParserFactory object creates an instance of the parser determined by the system property, `javax.xml.parsers.SAXParserFactory`.

`SAXParser`

The `SAXParser` interface defines several kinds of `parse()` methods. In general, you pass an XML data source and a DefaultHandler object to the parser, which processes the XML and invokes the appropriate methods in the handler object.

SAXReader

The SAXParser wraps a SAXReader. Typically, you don't care about that, but every once in a while you need to get hold of it using SAXParser's `getXMLReader()`, so you can configure it. It is the SAXReader which carries on the conversation with the SAX event handlers you define.

DefaultHandler
Not shown in the diagram, a DefaultHandler implements the `ContentHandler`, `ErrorHandler`,`DTDHandler`, and `EntityResolver` interfaces (with null methods), so you can override only the ones you're interested in.

ContentHandler

Methods like `startDocument`, `endDocument`, `startElement`, and `endElement` are invoked when an XML tag is recognized. This interface also defines methods `characters` and `processingInstruction`,which are invoked when the parser encounters the text in an XML element or an inline processing instruction,respectively.

ErrorHandler
Methods `error`, `fatalError`, and `warning` are invoked in response to various parsing errors. The default error handler throws an exception for fatal errors and ignores other errors (including validation errors). That's one reason you need to know something about the SAX parser, even if you are using the DOM. Sometimes, the application may be able to recover from a validation error. Other times, it may need to generate an exception. To ensure the correct handling, you'll need to supply your own error handler to the parser.

DTDHandler
Defines methods you will generally never be called upon to use. Used when processing a DTD to recognize and act on declarations for an *unparsed entity*.

EntityResolver
The `resolveEntity` method is invoked when the parser must identify data identified by a URI. In most cases,a URI is simply a URL, which specifies the location of a document, but in some cases the document may be identified by a URN -- a *public identifier*, or name, that is unique in the web space. The public identifier may be specified in addition to the URL. The `EntityResolver` can then use the public identifier instead of the URL to find the document, for example to access a local copy of the document if one exists.
A typical application implements most of the `ContentHandler` methods, at a minimum. Since the default implementations of the interfaces ignore all inputs except for fatal errors, a robust implementation may want to implement the ErrorHandler methods, as well.

The SAX Packages

The SAX parser is defined in the following packages.

Package Description

org.xml.sax Defines the SAX interfaces. The name "`org.xml`" is the package prefix that was settled on by the group that defined the SAX API.

org.xml.sax.ext

Defines SAX extensions that are used when doing more sophisticated SAX processing, for example, to process a document type definitions (DTD) or to see the detailed syntax for a file.

org.xml.sax.helpers

Contains helper classes that make it easier to use SAX -- for example, by defining a default handler that has null-methods for all of the interfaces, so you only need to override the ones you actually want to implement.

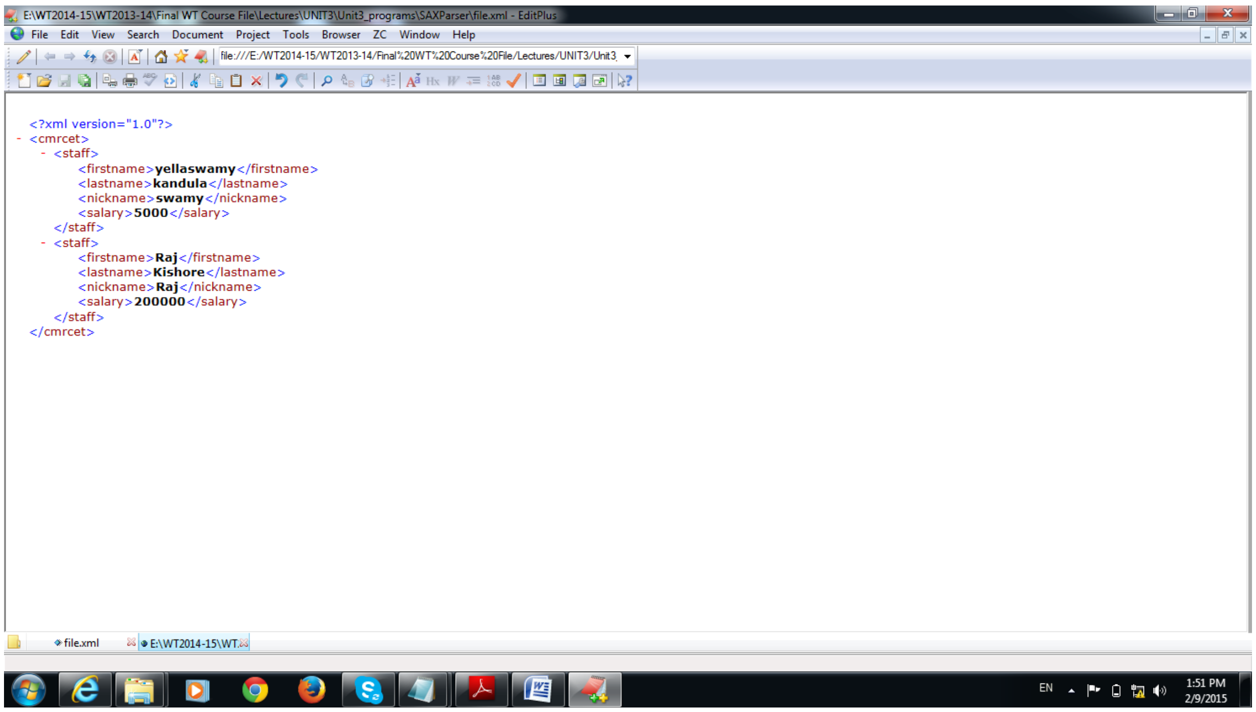
javax.xml.parsers Defines the `SAXParserFactory` class which returns the SAXParser. Also defines exception classes for reporting errors.

Example:

File.xml

```
<?xml version="1.0"?>
<mrec>
	<staff>
		<firstname>yellaswamy</firstname>
		<lastname>kandula</lastname>
		<nickname>swamy</nickname>
		<salary>5000</salary>
	</staff>
	<staff>
		<firstname>Raj</firstname>
		<lastname>Kishore</lastname>
		<nickname>Raj</nickname>
		<salary>200000</salary>
	</staff>
</mrec>
```

Output:


E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\SAXParser\file.xml - EditPlus
File Edit View Search Document Project Tools Browser ZC Window Help
file:///E:/WT2014-15/WT2013-14/Final%20WT%20Course%20File/Lectures/UNIT3/Unit3_
file.xml
E:\WT2014-15\WT...
EN
1:51 PM
2/9/2015

```
  <?xml version="1.0"?>
- <cmrcet>
  - <staff>
        <firstname>yellaswamy</firstname>
        <lastname>kandula</lastname>
        <nickname>swamy</nickname>
        <salary>5000</salary>
    </staff>
  - <staff>
        <firstname>Raj</firstname>
        <lastname>Kishore</lastname>
        <nickname>Raj</nickname>
        <salary>200000</salary>
    </staff>
  </cmrcet>
```

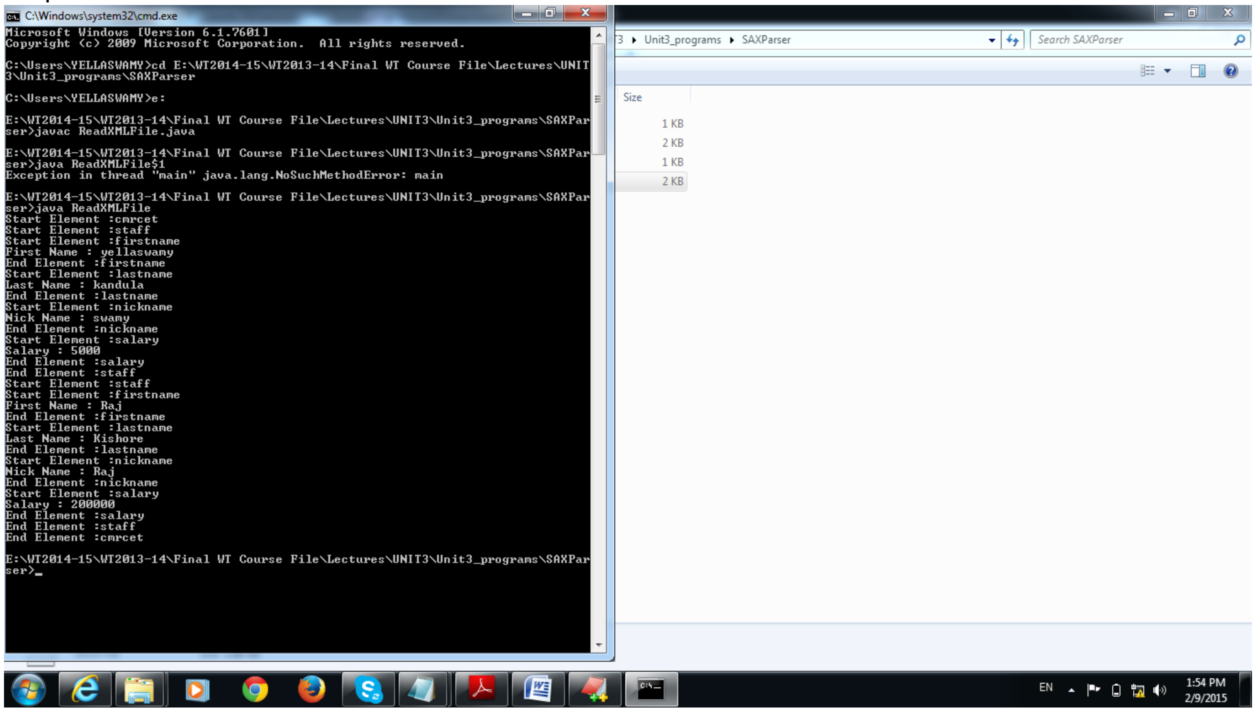
ReadXMLFile.java

```
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 ReadXMLFile {

   public static void main(String argv[]) {

    try {
 //Step1
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
 //Step2  set the dcoument handler
            DefaultHandler handler = new DefaultHandler() {

            boolean bfname = false;
            boolean blname = false;
            boolean bnname = false;
            boolean bsalary = false;

            public void startElement(String uri, String localName,String qName,
                Attributes attributes) throws SAXException {
```

```
		System.out.println("Start Element :" + qName);

		if (qName.equalsIgnoreCase("FIRSTNAME")) {
			bfname = true;
		}

		if (qName.equalsIgnoreCase("LASTNAME")) {
			blname = true;
		}

		if (qName.equalsIgnoreCase("NICKNAME")) {
			bnname = true;
		}

		if (qName.equalsIgnoreCase("SALARY")) {
			bsalary = true;
		}

	}

	public void endElement(String uri, String localName,
		String qName) throws SAXException {

		System.out.println("End Element :" + qName);

	}

	public void characters(char ch[], int start, int length) throws SAXException {

		if (bfname) {
			System.out.println("First Name : " + new String(ch, start, length));
			bfname = false;
		}

		if (blname) {
			System.out.println("Last Name : " + new String(ch, start, length));
			blname = false;
		}

		if (bnname) {
			System.out.println("Nick Name : " + new String(ch, start, length));
			bnname = false;
		}

		if (bsalary) {
```

```
                                        System.out.println("Salary : " + new String(ch, start, length));
                                        bsalary = false;
                                }

                        }

                };

                 saxParser.parse("file.xml", handler);

                } catch (Exception e) {
                 e.printStackTrace();
                }

              }

        }
```

Output:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\YELLASWAMY>cd E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT
3\Unit3_programs\SAXParser

C:\Users\YELLASWAMY>e:

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\SAXPar
ser>javac ReadXMLFile.java

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\SAXPar
ser>java ReadXMLFile$1
Exception in thread "main" java.lang.NoSuchMethodError: main

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\SAXPar
ser>java ReadXMLFile
Start Element :cmrcet
Start Element :staff
Start Element :firstname
First Name : yellaswamy
End Element :firstname
Start Element :lastname
Last Name : kandula
End Element :lastname
Start Element :nickname
Nick Name : swamy
End Element :nickname
Start Element :salary
Salary : 5000
End Element :salary
End Element :staff
Start Element :staff
Start Element :firstname
First Name : Raj
End Element :firstname
Start Element :lastname
Last Name : Kishore
End Element :lastname
Start Element :nickname
Nick Name : Raj
End Element :nickname
Start Element :salary
Salary : 200000
End Element :salary
End Element :staff
End Element :cmrcet

E:\WT2014-15\WT2013-14\Final WT Course File\Lectures\UNIT3\Unit3_programs\SAXPar
ser>_
```

WEB SERVERS & SERVLETS

Overview

- **Tomcat Webserver**
- **Introduction To Servlets**
- **Life Cycle of Servlet**
- **Servlet API**
- **Reading Servlet Parameters**
- **Steps to Run a Servlet**
- **Example to handle Http Request and Response**
- **Using Cookies and Session Tracking**
- **Security Issues**

Tomcat WebServer

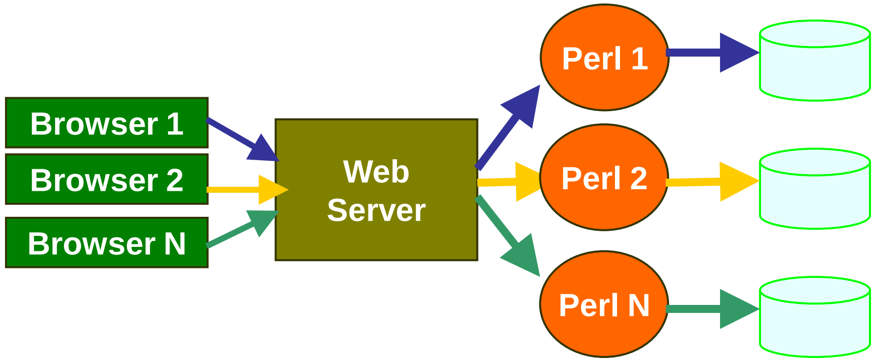
- A “servlet container” is like a mini server, but only for serving html, jsp and servlets.
- Many servlet containers could be used for this course. Some may even be easier to configure than tomcat, but tomcat provides an easy-to-use development/deployment tool and also complies with the servlet specification best of all containers.
- Tomcat is from Apache and is open-source.
- Tomcat can be used as a stand-alone servlet container.
- You can install the Apache server with Tomcat, and then proceed to configure each for their individual purposes. (The server would relay servlet requests to Tomcat.)

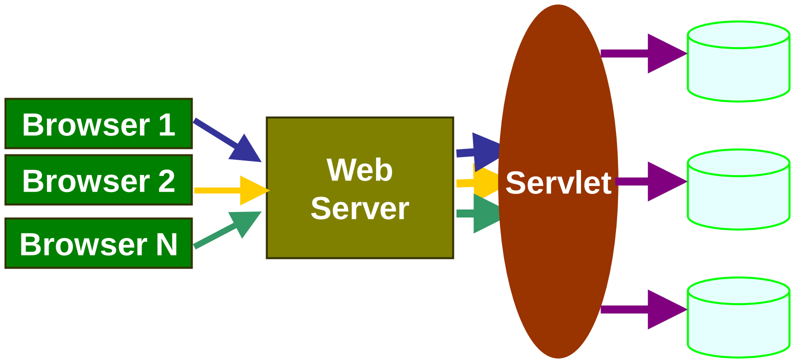
Introduction to Servlets

What can you build with Servlets?

- Search Engines
- E-Commerce Applications
- Shopping Carts
- Product Catalogs
- Intranet Applications
- Groupware Applications:
  - bulletin boards
  - file sharing

Servlets vs. CGI





- A Servlet does not run in a separate process.
- A Servlet stays in memory between requests.
- A CGI program needs to be loaded and started for each CGI request.
- There is only a single instance of a servlet which answers all requests concurrently.

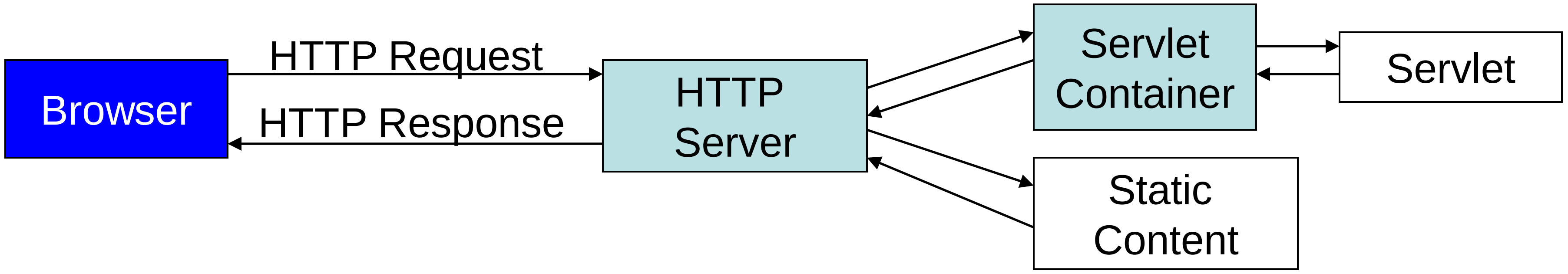
Benefits of Java Servlets

- **Performance**
  - The performance of servlets is superior to CGI because there is no process creation for each client request.
  - Each request is handled by the servlet container process.
  - After a servlet has completed processing a request, it stays resident in memory, waiting for another request.
- **Portability**
  - Like other Java technologies, servlet applications are portable.
- **Rapid development cycle**
  - As a Java technology, servlets have access to the rich Java library that will help speed up the development process.
- **Robustness**
  - Servlets are managed by the Java Virtual Machine.
  - Don't need to worry about memory leak or garbage collection, which helps you write robust applications.
- **Widespread acceptance**
  - Java is a widely accepted technology.

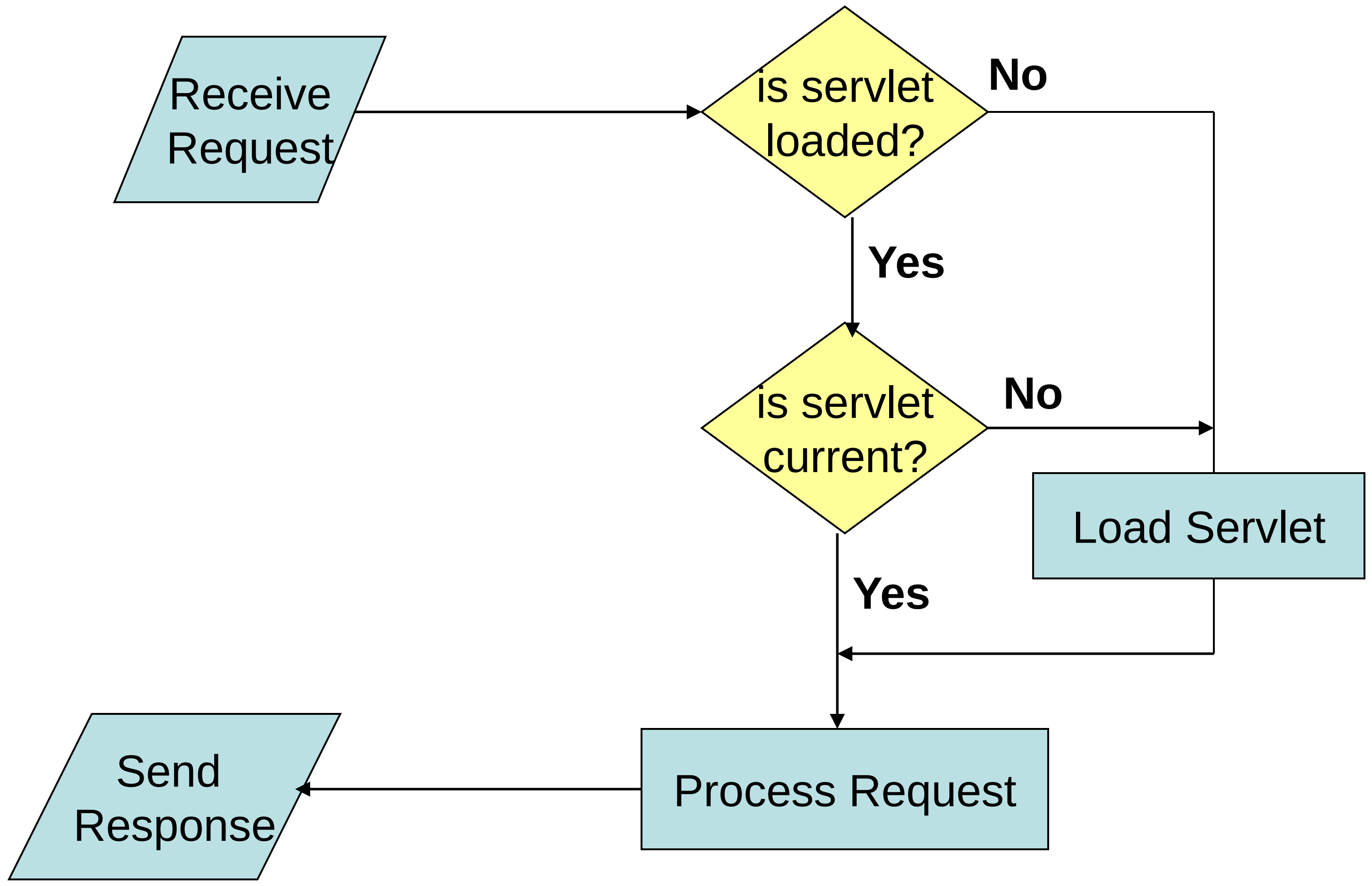
Definitions

- A ***servlet*** is a Java class that can be loaded dynamically into and run by a special web server.
- This servlet-aware web server, is known as ***servlet container***.
- Servlets interact with clients via a request-response model based on HTTP.
- Therefore, a servlet container must support HTTP as the protocol for client requests and server responses.
- A servlet container also can support similar protocols such as HTTPS (HTTP over SSL) for secure transactions.

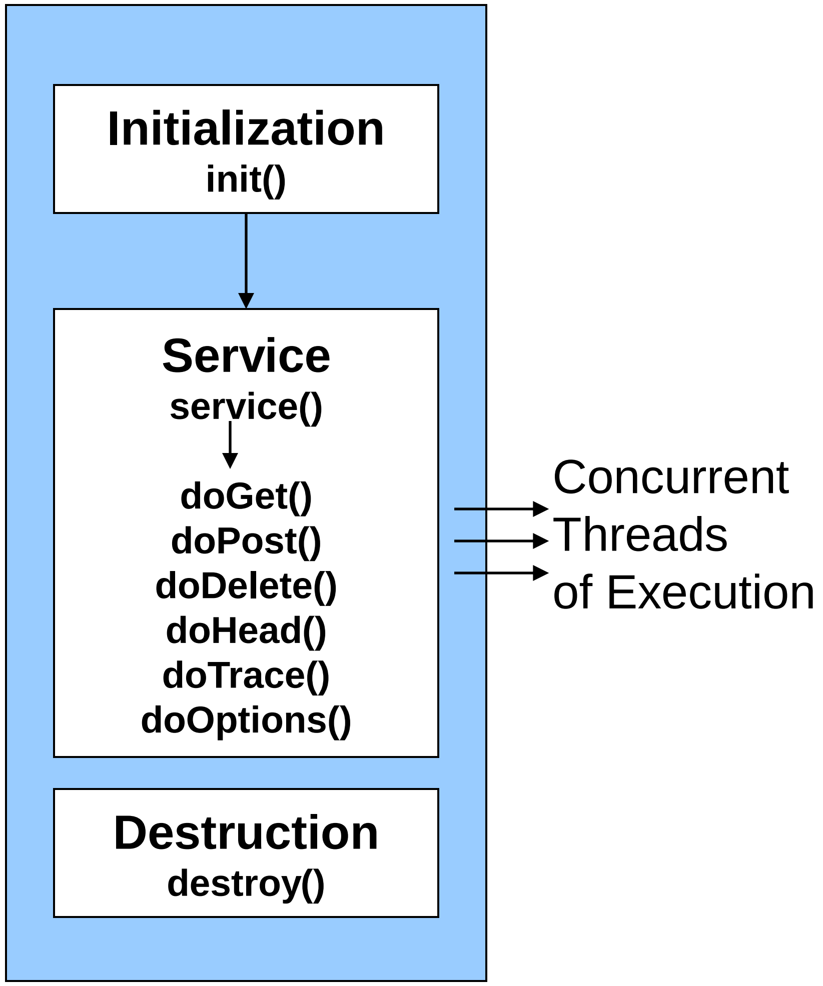
Servlet Container Architecture



How Servlets Work



Servlet Life Cycle



Servlet Life Cycle

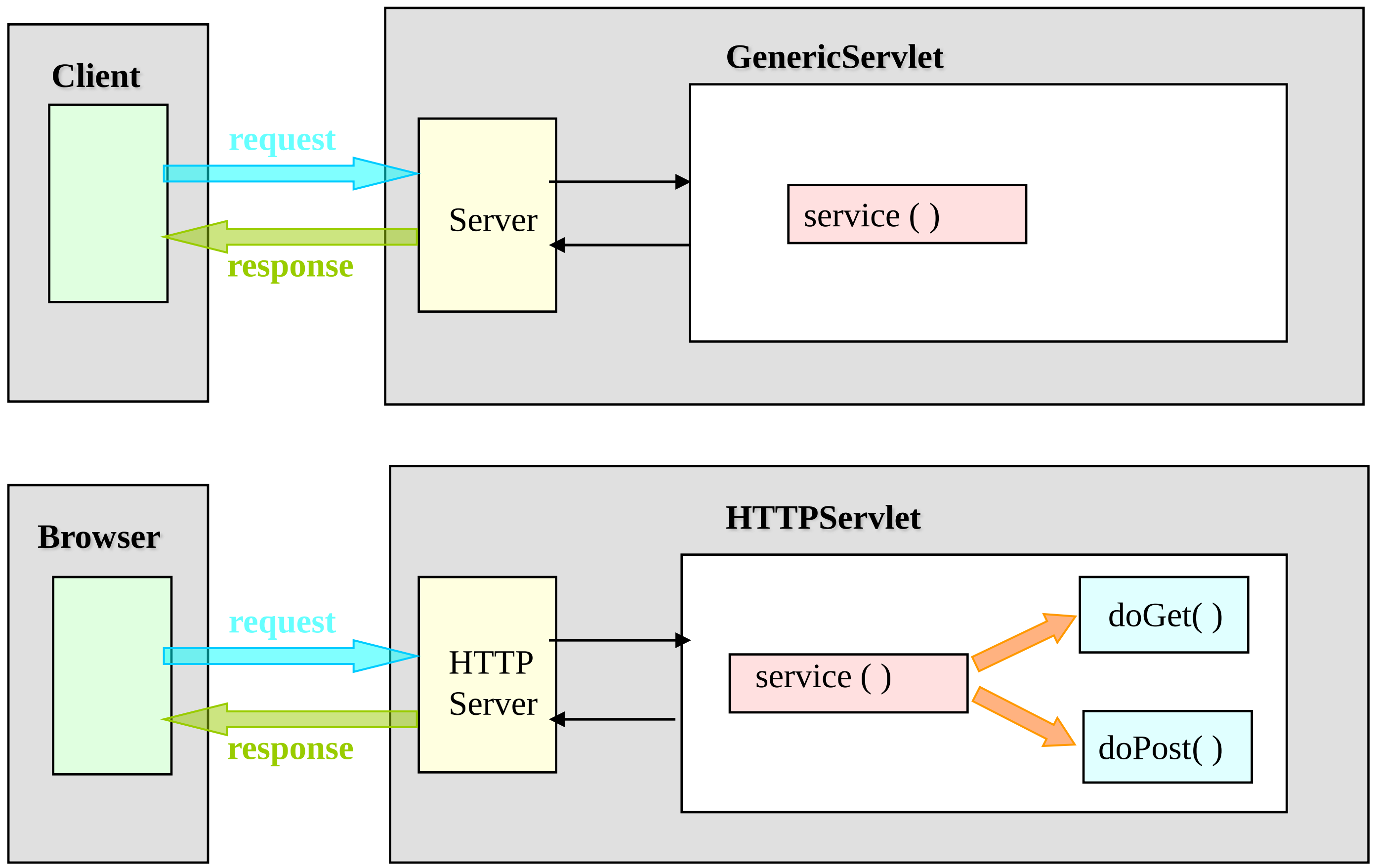
- When a servlet is FIRST requested, it is loaded into the servlet engine. The init() method of the servlet is invoked so that the servlet may initialize itself.
- Once initialization is complete, the request is then forwarded to the appropriate method (ie. doGet or doPost)
- The servlet is then held in memory. Subsequent requests are simply forwarded to the servlet object.
- When the engine wishes to remove the servlet, its destroy() method is invoked.

NOTE: Servlets can receive multiple requests for multiple clients at any given time. Therefore, servlets must be thread safe

Servlet APIs

- Every servlet must implement javax.servlet.Servlet interface
- Most servlets implement the interface by extending one of these classes
  - javax.servlet.GenericServlet
  - javax.servlet.http.HttpServlet

Generic Servlet & HTTP Servlet



Interface javax.servlet.Servlet

- The Servlet interface defines methods
  - to initialize a servlet
  - to receive and respond to client requests
  - to destroy a servlet and its resources
  - to get any startup information
  - to return basic information about itself, such as its author, version and copyright.
- Developers need to directly implement this interface only if their servlets cannot (or choose not to) inherit from GenericServlet or HttpServlet.

Life Cycle Methods

GenericServlet - Methods

- **void** **init**(ServletConfig config)
  - Initializes the servlet.
- void **service**(ServletRequest req, ServletResponse res)
  - Carries out a single request from the client.
- **void** **destroy**()
  - Cleans up whatever resources are being held (e.g., memory, file handles, threads) and makes sure that any persistent state is synchronized with the servlet's current in-memory state.
- ServletConfig **getServletConfig**()
  - Returns a servlet config object, which contains any initialization parameters and startup configuration for this servlet.
- String **getServletInfo**()
  - Returns a string containing information about the servlet, such as its author, version, and copyright.

HttpServlet - Methods

- void doGet (HttpServletRequest request,
  HttpServletResponse response)
  - handles GET requests
- void doPost (HttpServletRequest request,
  HttpServletResponse response)
  - handles POST requests
- void doPut (HttpServletRequest request,
  HttpServletResponse response)
  - handles PUT requests
- void doDelete (HttpServletRequest request,
  HttpServletResponse response)
  - handles DELETE requests

Servlet Request Objects

- provides client request information to a servlet.
- the servlet container creates a servlet request object and passes it as an argument to the servlet's service method.
- the ServletRequest interface define methods to retrieve data sent as client request:
  - –parameter name and values
  - – attributes
  - – input stream
- HTTPServletRequest extends the ServletRequest interface to provide request information for HTTP servlets

HttpServletRequest - Methods

Enumeration	**getParameterNames()** an Enumeration of String objects, each String containing the name of a request parameter; or an empty Enumeration if the request has no parameters
java.lang.String[]	**getParameterValues** (java.lang.String name) Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.
java.lang.String	**getParameter** (java.lang.String name) Returns the value of a request parameter as a String, or null if the parameter does not exist.

HttpServletRequest - Methods

Cookie[]	**getCookies**() Returns an array containing all of the Cookie objects the client sent with this request.
java.lang.String	**getMethod**() Returns the name of the HTTP method with which\thi request was made, for example, GET, POST, or PUT.
java.lang.String	**getQueryString**() Returns the query string that is contained in the request URL after the path.
HttpSession	**getSession**() Returns the current session associated with this request, or if the request does not have a session, creates one.

Servlet Response Objects

- Defines an object to assist a servlet in sending a response to the client.
- The servlet container creates a ServletResponse object and passes it as an argument to the servlet's service method.

HttpServletResponse - Methods

java.io.PrintWriter	**getWriter**() Returns a PrintWriter object that can send character text to the client
void	**setContentType** (java.lang.String type) Sets the content type of the response being sent to the client. The content type may include the type of character encoding used, for example, text/html; charset=ISO-8859-4
int	**getBufferSize**() Returns the actual buffer size used for the response

Reading Servlet Parameters

- The request object (which implements HttpServletRequest) provides information from the HTTP request to the servlet
- One type of information is parameter data, which is information from the query string portion of the HTTP request

```
http://www.example.com/servlet/PrintThis?arg=aString
```

Query string with one parameter

Steps to Running a Servlet

- Create a directory structure under Tomcat for your application.
- Write the servlet source code.
- Compile your source code.
- deploy the servlet
- Run Tomcat
- Call your servlet from a web browser

Create a Directory Structure

- The webapps directory is the Tomcat installation dir (CATALINA_HOME) is where you store your web applications.
- A *web application* is a collection of servlets and other contents installed under a specific subset of the server's URL namespace.
- A separate directory is dedicated for each servlet application.
- Create a directory called myApp under the webapps directory.
- Create the src and WEB-INF directories under myApp, and create a directory named classes under WEB-INF.
  - The src directory is for your source files, and the classes directory under WEB-INF is for your Java classes.
  - If you have html files, you put them directly in the myApp directory.
- The admin, ROOT, and examples directories are for applications created automatically when you install Tomcat

Write the Servlet Code

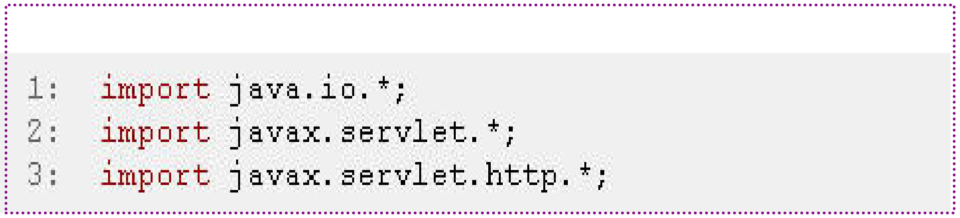
- Servlets implement the javax.servlet.Servlet interface.
- Because most servlets extend web servers that use the HTTP protocol to interact with clients, the most common way to develop servlets is by specializing the javax.servlet.http.HttpServlet class.
- The HttpServlet class implements the Servlet interface by extending the GenericServlet base class, and provides a framework for handling the HTTP protocol.
- Its service() method supports standard HTTP requests by dispatching each request to a method designed to handle it.
- In myApp/src, create a file called TestingServlet.java

Servlet Example

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet
{
  protected void doGet(HttpServletRequest req,
                       HttpServletResponse res)
  {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println( "<HTML><HEAD><TITLE> Hello You!” +
                 “</Title></HEAD>” +
                 “<Body> HelloYou!!!</BODY></HTML>“ );
    out.close();
  }
 }
```

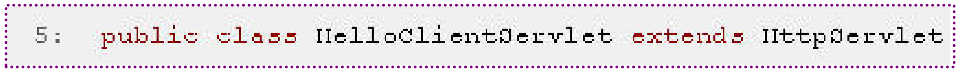
An Example of Servlet (I)

Lines 1 to 3 import some packages which contain many classes which are used by the Servlet (almost every Servlet needs classes from these packages).



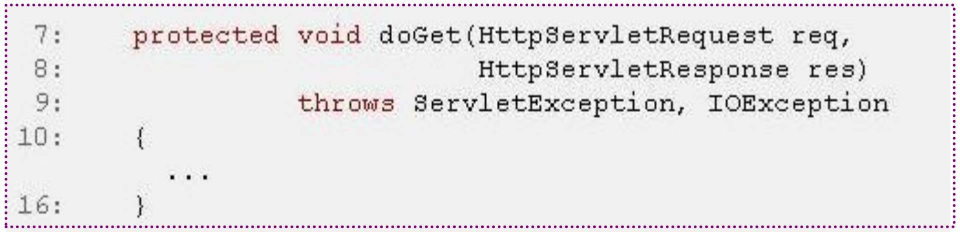
```
1:  import java.io.*;
2:  import javax.servlet.*;
3:  import javax.servlet.http.*;
```

The Servlet class is declared in line 5. Our Servlet extends *javax.servlet.http.HttpServlet*, the standard base class for HTTP Servlets.



```
5:  public class HelloClientServlet extends HttpServlet
```

In lines 7 through 16 *HttpServlet'*s *doGet* method is getting overridden



```
7:    protected void doGet(HttpServletRequest req,
8:                         HttpServletResponse res)
9:                 throws ServletException, IOException
10:   {
        ...
16:   }
```

An Example of Servlet (II)

In line 11 we use a method of the *HttpServletResponse* object to set the content type of the response that we are going to send. All response headers must be set *before* a *PrintWriter* or *ServletOutputStream* is requested to write body data to the response.

```
11:       res.setContentType("text/html");
```

In line 12 we request a *PrintWriter* object to write text to the response message.

```
12:       PrintWriter out = res.getWriter();
```

In lines 13 and 14 we use the *PrintWriter* to write the text of type *text/html* (as specified through the content type).

```
13:       out.println("<HTML><HEAD><TITLE>Hello Client!</TITLE>"+
14:                   "</HEAD><BODY>Hello Client!</BODY></HTML>");
```

An Example of Servlet (III)

The *PrintWriter* gets closed in line 15 when we are finished writing to it.

```
15:        out.close();
```

In lines 18 through 21 we override the *getServletInfo()* method which is supposed to return information about the Servlet, e.g. the Servlet name, version, author and copyright notice. This is not required for the function of the *HelloClientServlet* but can provide valuable information to the user of a Servlet who sees the returned text in the administration tool of the Web Server.

```
18:     public String getServletInfo()
19:     {
20:       return "HelloClientServlet 1.0 by Stefan Zeiger";
21:     }
```

Compile the Servlet

- Compile the Servlet class
- The resulting TestingServlet.class file should go under myApp/WEB-INF/classes

Deploy the Servlet

- In the Servlet container each application is represented by a servlet context
- each servlet context is identified by a unique path prefix called context path
  - For example our application is identified by /myApp which is a directory under webapps.
- The remaining path is used in the selected context to find the specific Servlet to run, following the rules specified in the deployment descriptor.

Deployment Descriptor

- The deployment descriptor is a XML file called web.xml that resides in the WEB-INF directory whitin an application.

```
<web-app xmlns=http://java.sun.com/xml/ns/j2ee……>

  <display-name>test</display-name>
  <description>test example</description>

   <servlet>
       <servlet-name>Testing</servlet-name>
       <servlet-class>TestingServlet</servlet-class>
   </servlet>

  <servlet-mapping>
      <servlet-name>Testing</servlet-name>
      <url-pattern>/servlet/TestingServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Run the Servlet

- To execute your Servlet, type the following URL in the Browser’s address field:
- http://localhost/myApp/servlet/myServlet

Web Client Session Tracking

- HTTP is a stateless protocol that takes requests from web clients and responds with a file. It does not memorize what has happened in the past.
- FTP and Telnet protocols know the client states, such as users, connections, and disconnections but HTTP does not.
- A client session consists of a series of conversations between the client and web applications on the web server.

Web Client Session Tracking contd..

- Using the HttpSession API in session management is quite straightforward, and it may be the best option for session tracking in most cases.
- The HttpSession object can hold a session id that can be used to identify whether the requests are within the same session so that they can share the same data.
- Each HttpSession object represents a single user HTTP session.

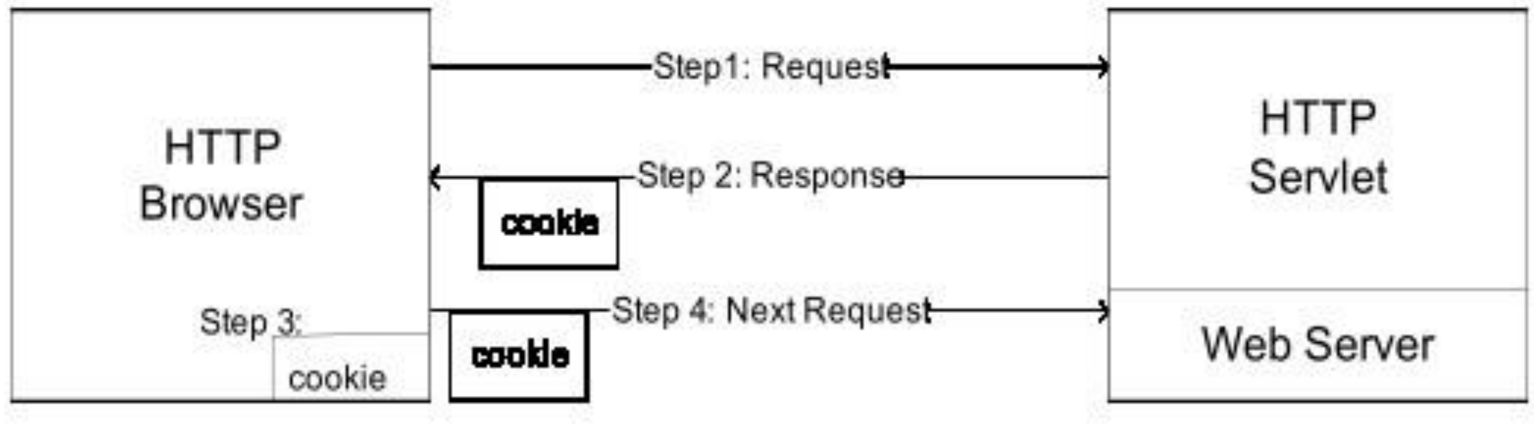
Web Client Session Tracking contd..

- Mechanisms for session maintenance:
  - Cookies
  - URL Rewriting
  - Hidden Form fields

Servlet Cookies

- Cookies are text files that store sets of param/value pairs. The Servlet at the server side generates the cookie based on the client’s HTTP request.
- The cookie is created on the server side and sent back to the client along with the HttpServletResponse object.
- The cookie is stored in the client’s browser.

Servlet Cookies contd..



URL Rewriting

- Rewrite the URLs of the links of a web page to contain extra information in the form of query string or extra path information.
- Example : a user named John Doe log in with session ID=1234 and enter page1.cgi , page1.cgi contains a link to page2.cgi
- When user click link to page2.cgi, the URL is:

http://sample.com/page2.cgi?fname=John&lname=Doe&sessionid=1234

Hidden Form Fields

- <input type=“HIDDEN” name=“id” value=“1234”>
- Typically contained in forms that are placed in a common frame of a frameset
- Accessed using client-side javascript
- When javascript executes in one page of an application, it stored values(session ID) in hidden form fields.

Security Issues

Server-side Security Issues

- Interception of Session State Information
- Forgery of Session State Information
- Session Timeout
- Buffer Overflow
- Data Validation

Security Issues contd..

- Page Sequencing
- Information Reporting
- Browser Residue
- User Authentication
- Logging of Sensitive Information

Important Questions

1. Briefly explain about Tomcat web server.

2 a) What are the limitations of Servlets?

  b) Explain Servlet Vs CGI

3. Explain the life cycle of a servlet.

4. Write a session tracker that tracks the number of accesses and last access data of a particular web page.

5.a) Discuss about javax.servelet package.

  b) What are the security issues related to servlets.

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:

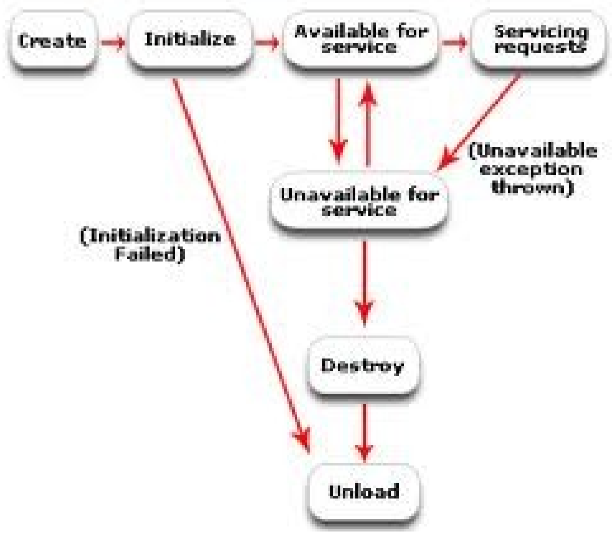
1. They are generally much faster than CGI scripts because a different process model is used.
2. They use a standard API that is supported by many web servers.
3. They have all the advantages of the Java programming language, including ease of development and platform independence.
4. They can access 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 `Servlet` interface.

```
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. **Explain Servlet Protocol Support?**

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 `GET` Method

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 `POST` Method

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 `GET` request 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. <u>Explain Servlet Interface?</u>

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 `ServletContext` interface 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()	Returns an enumeration of servlet names available in the current namespace.
log()	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>
```

<u>Servlet File</u>

```
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. **Explain Servlet RequestDispatcher Interface?**

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);
      }
}
```

Welcome.java 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);
      }
}
```

Web.xml:

```
<!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. <u>**Explain Servlet SessionMangement Mechanisms?**</u>

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 will never 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) throws
ServletException,
      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 unique
    return 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) throws
ServletException,
      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 necessary
    String sessionid = req.getPathInfo();
    if (sessionid == null) {
      sessionid = generateSessionId();
    }

    // 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 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.

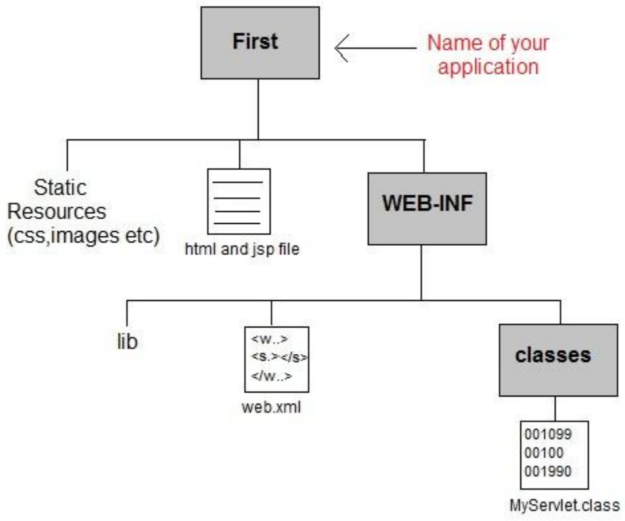
After installing Tomcat Server on your machine follow the below mentioned steps :

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.

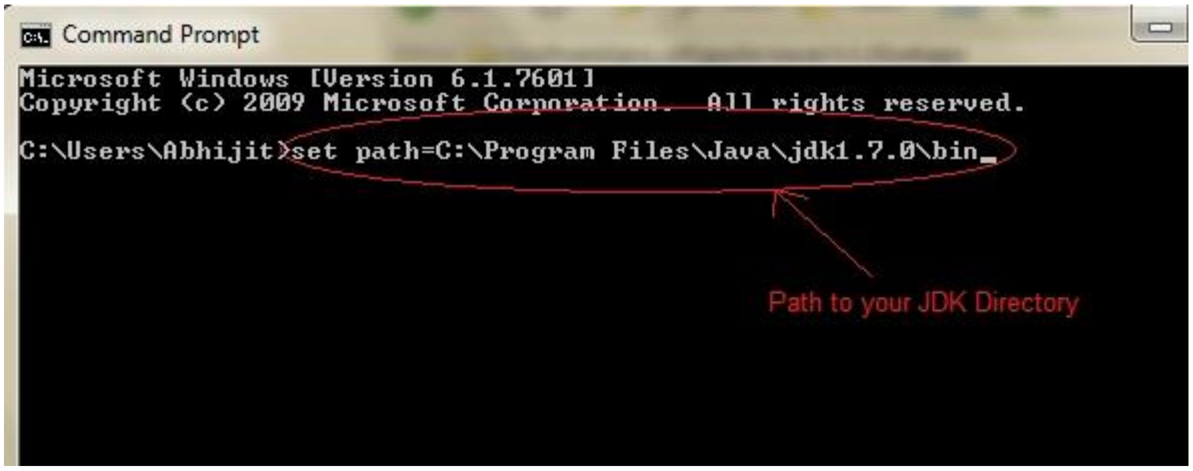
Compile it(explained in next step) from there and paste the class file into WEB-INF/classes/ directory that you have to create inside **Tomcat/webapps** directory.

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.

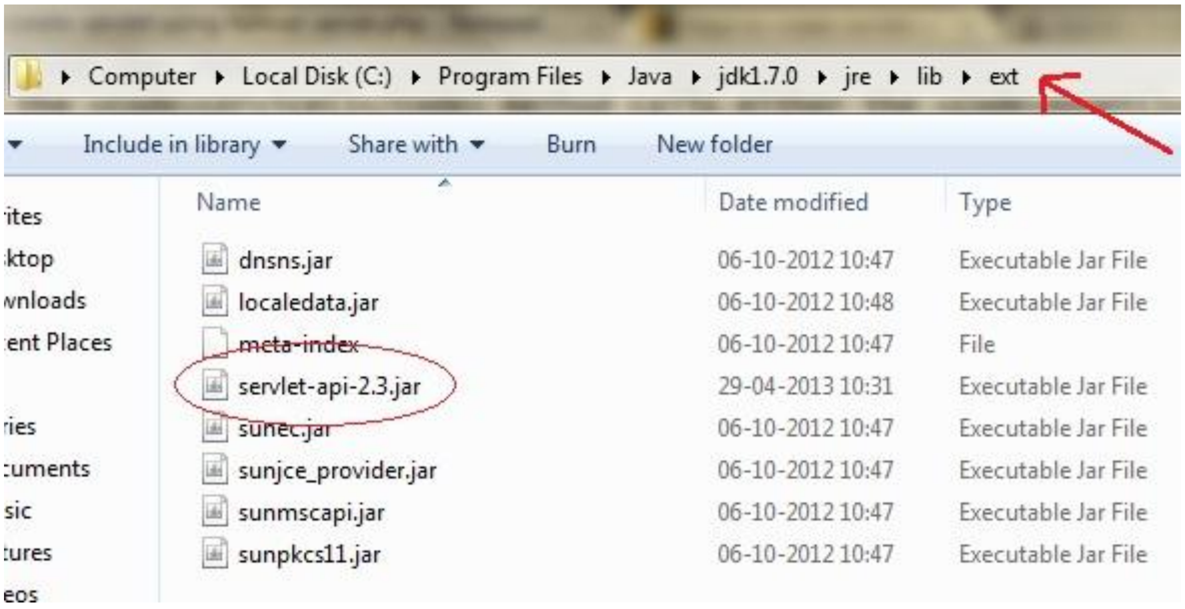
Steps to compile a Servlet:

- Set the Class Path.



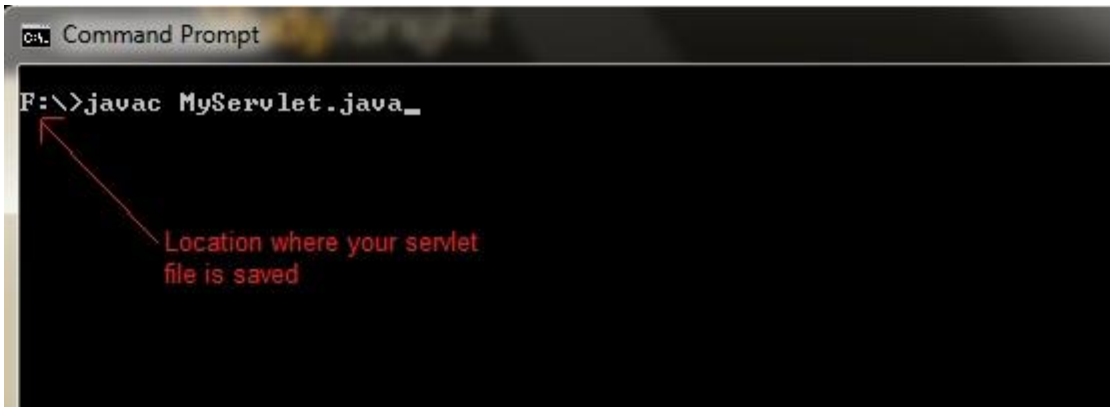
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

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_
```

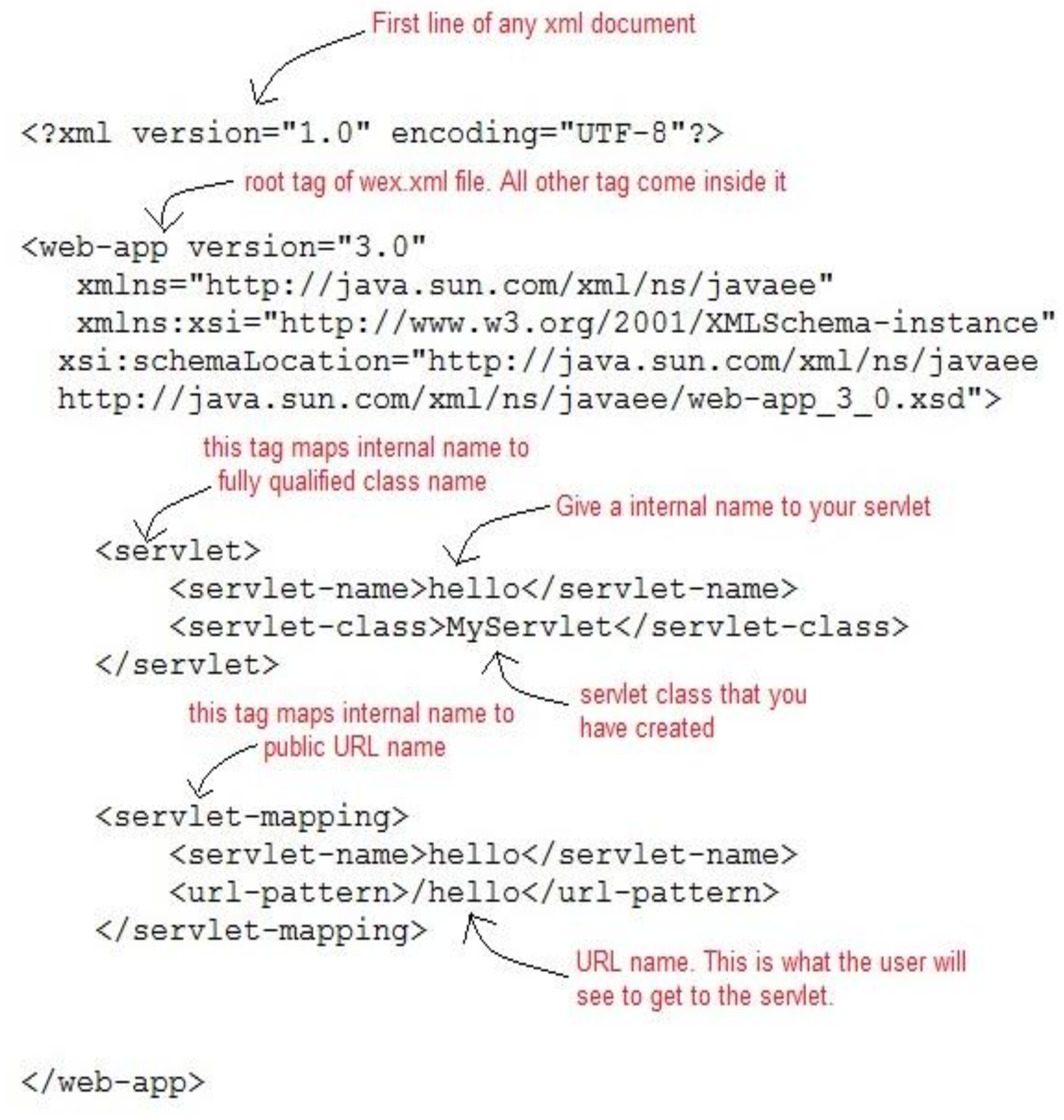
OTE: After compiling your Servlet class you will have to paste the class file into WEB-INF/classes/ directory.

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.

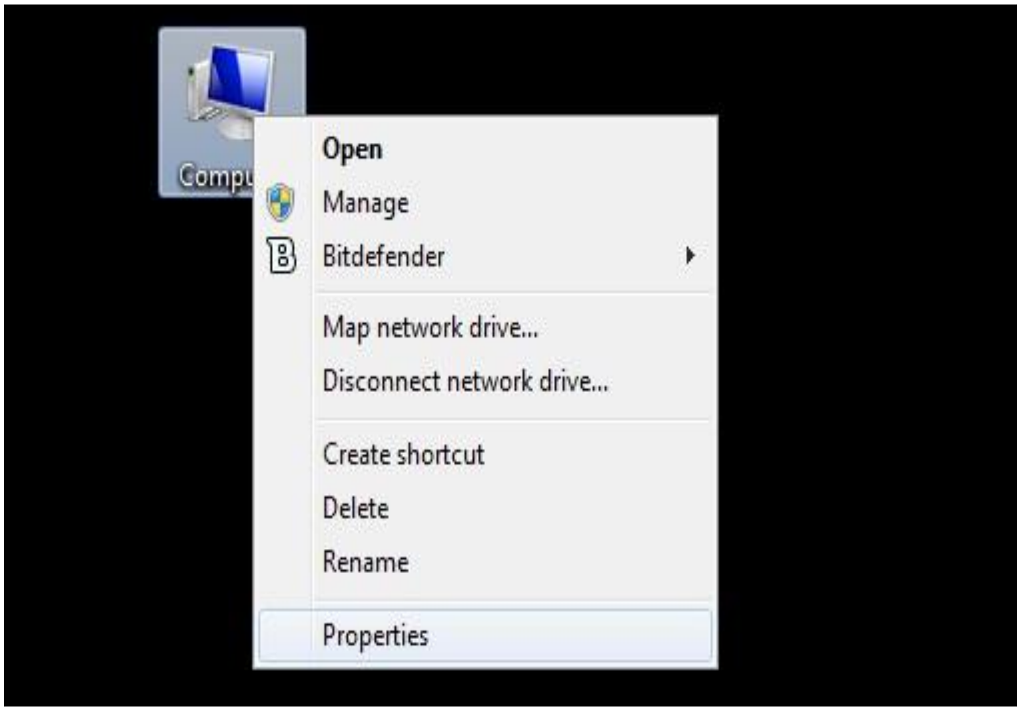
Or, execute the following command on your windows machine using RUN prompt.

```
C:\apache-tomcat-7.0.14\bin\startup.bat
```

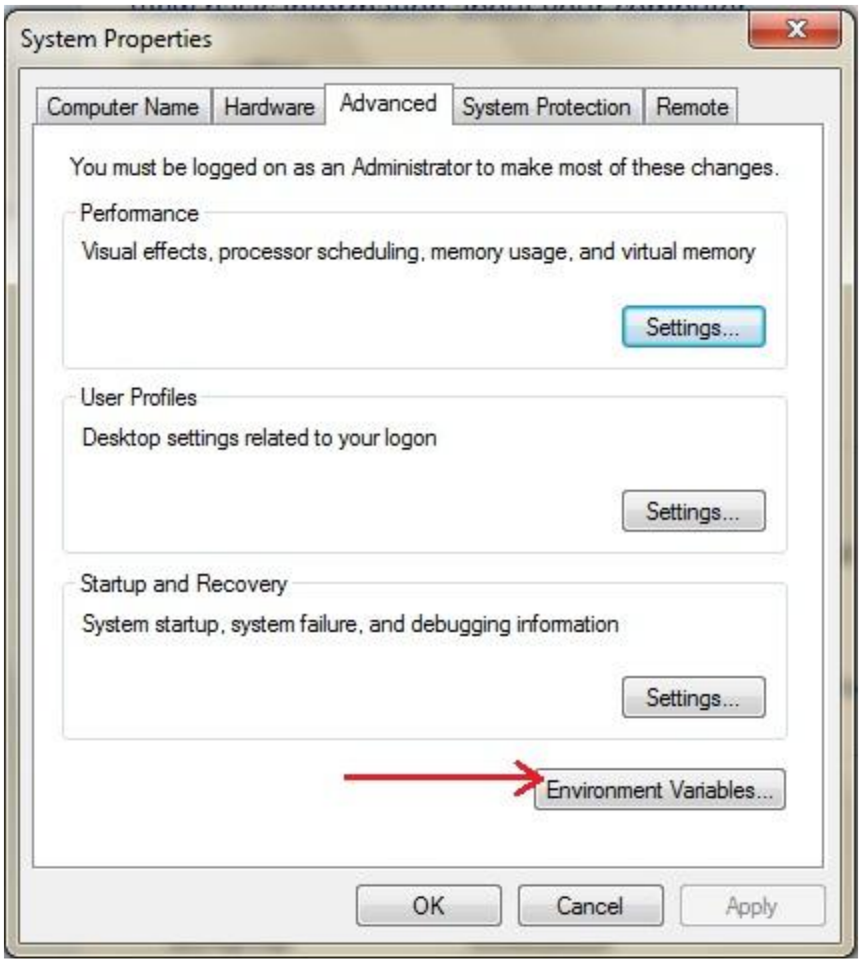
6. Starting Tomcat Server for the first time

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.

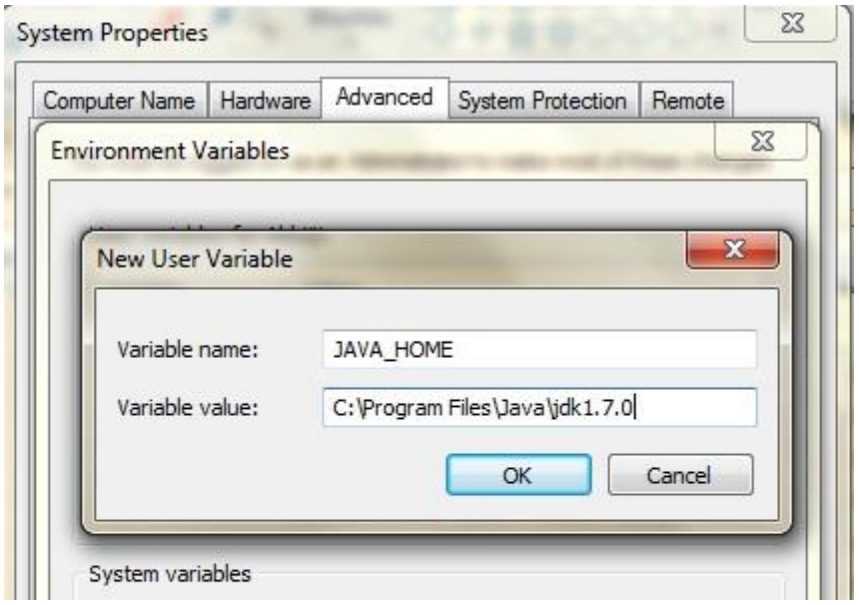
- Right Click on **My Computer**, go to **Properperites**.



- Go to **Advanced** Tab and Click on **Enviroment Variables...** button.


System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your logon
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply

- 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.

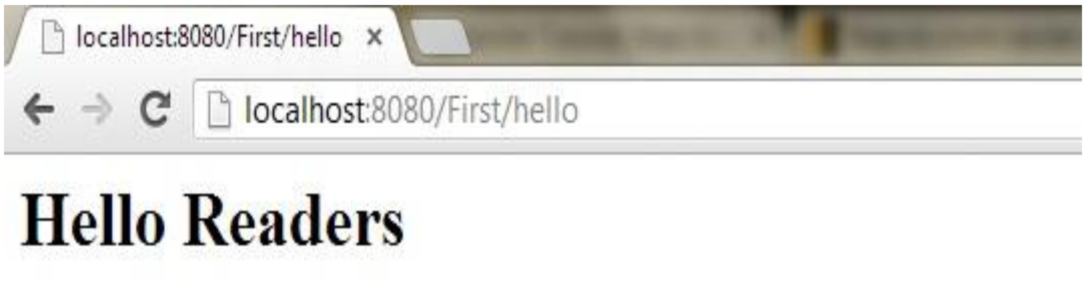


Variable name: JAVA_HOME

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();
```

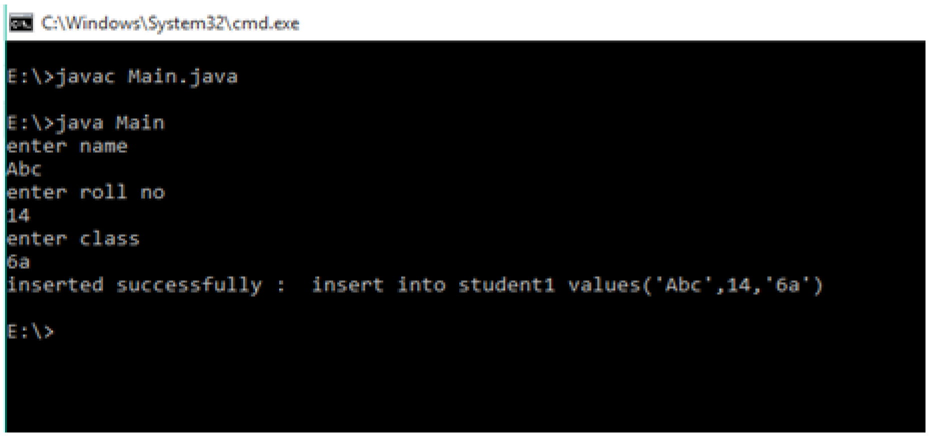
Implementation

```
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

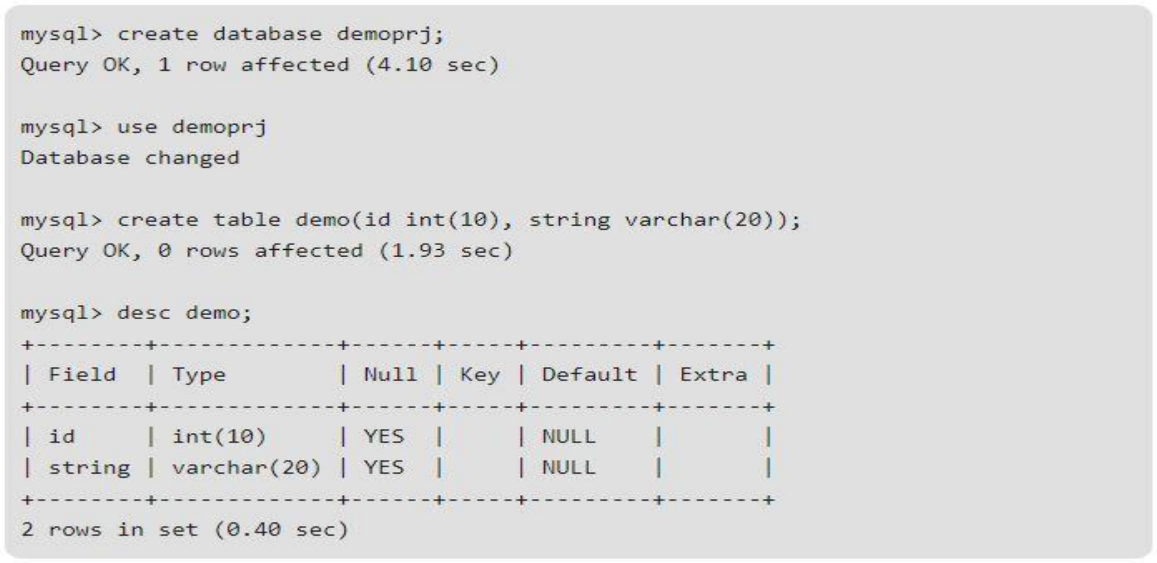
To start with interfacing Java Servlet Program with JDBC Connection:

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,



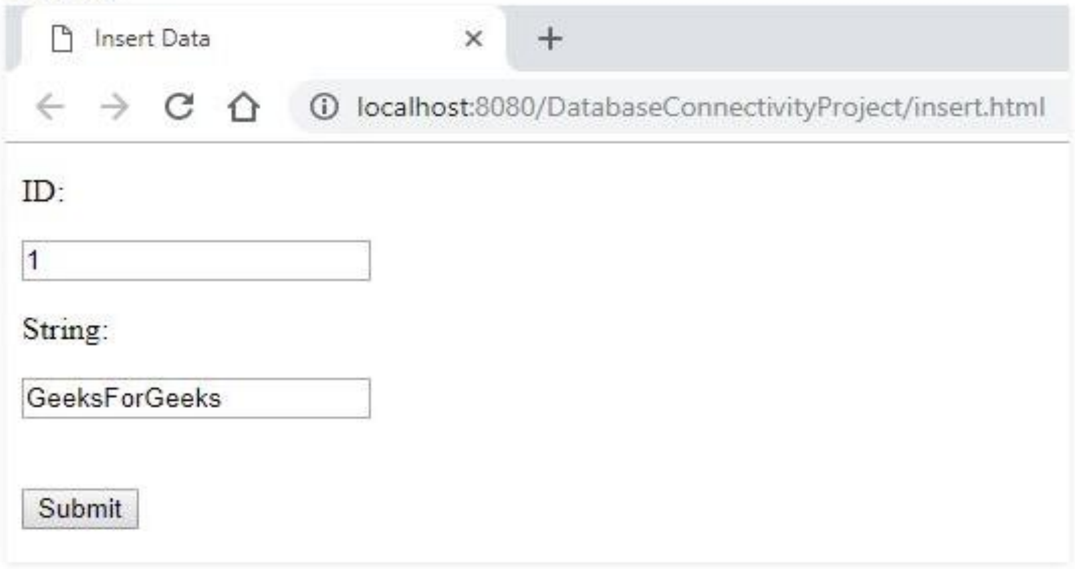
```
mysql> create database demoprj;
Query OK, 1 row affected (4.10 sec)

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:



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,

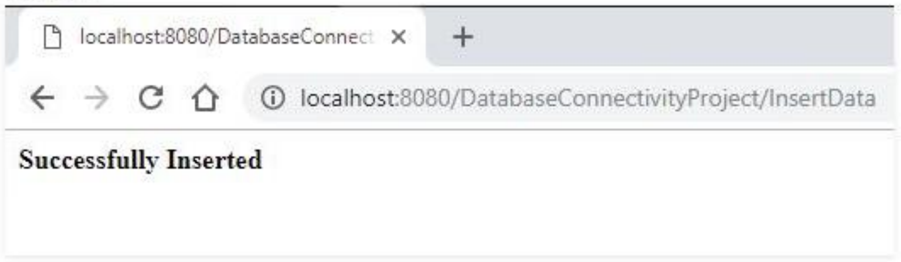
```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;

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/

Web Technologies

MODULE – 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.

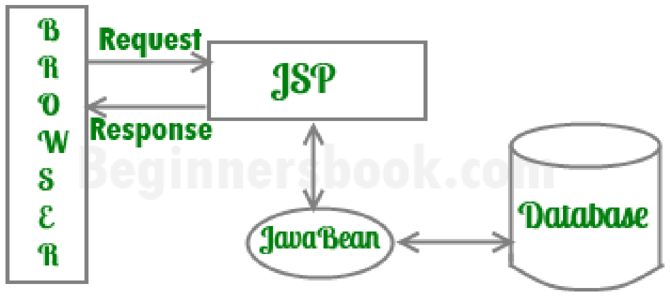
Less code than Servlet

- 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.

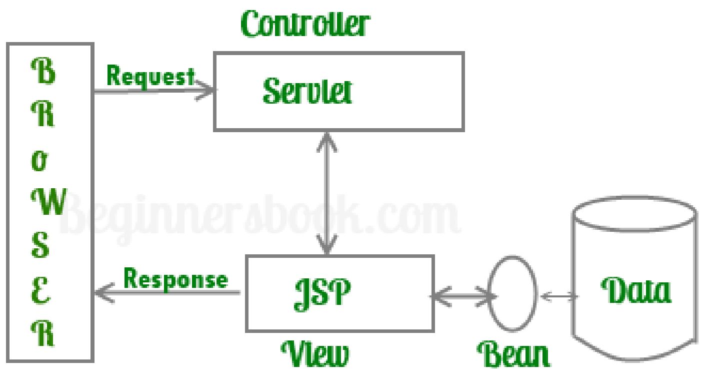
Architecture of a JSP Application

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.

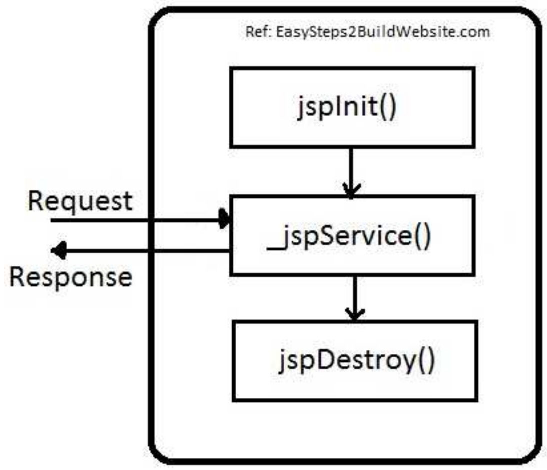


Java Server Pages (JSP) Life Cycle

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.

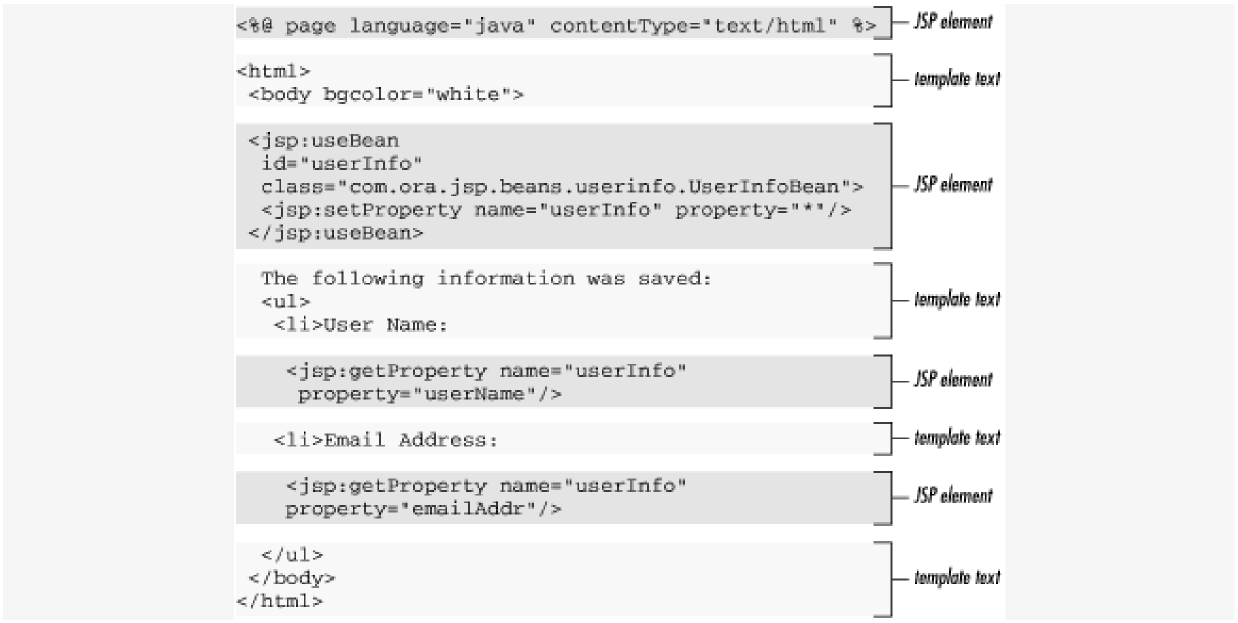
[code language="java"]
void **_jspService**(HttpServletRequest req, HttpServletResponse res)
{
//code goes here
}[/code]

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]

4.1 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.

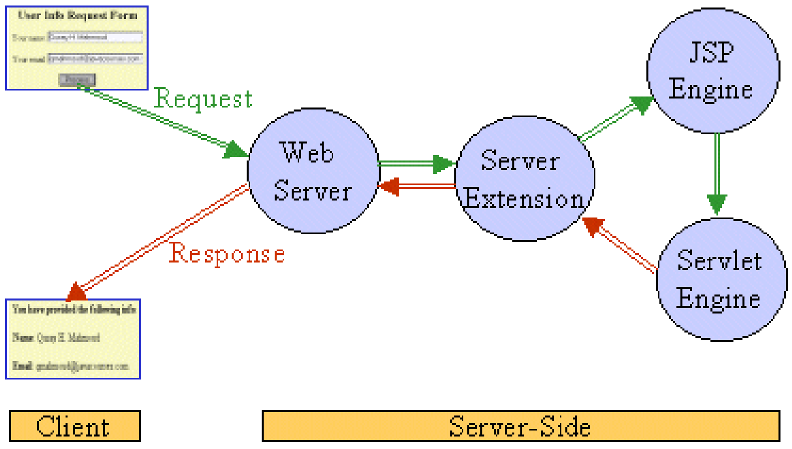
Table: Directive elements

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

4.2 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.

4.3 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:

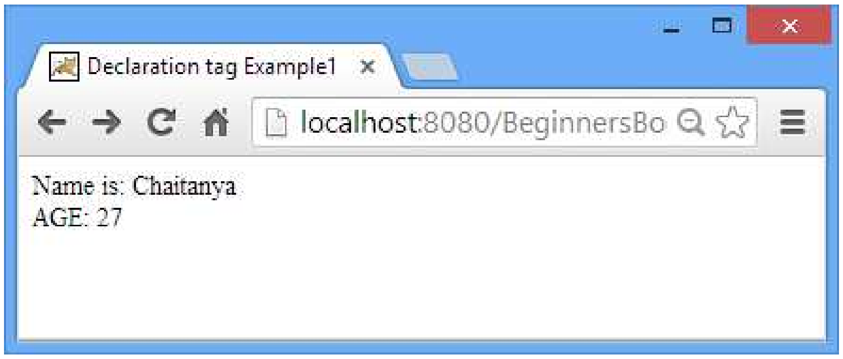
```
<%!  Declaration %>
```

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
```

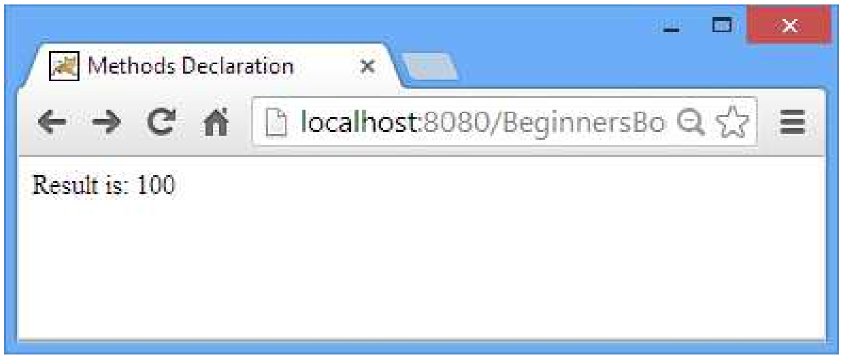
Example 2: Methods declaration

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
```

4.4 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 %>
```

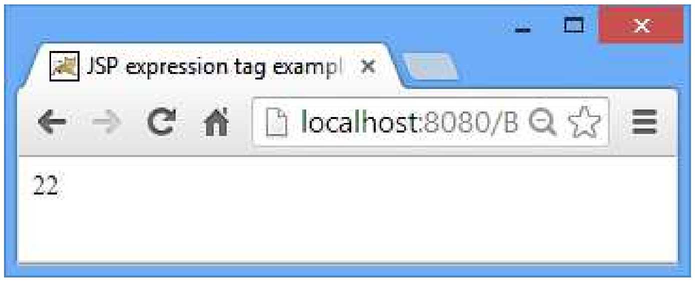
JSP expression tag Examples

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
```

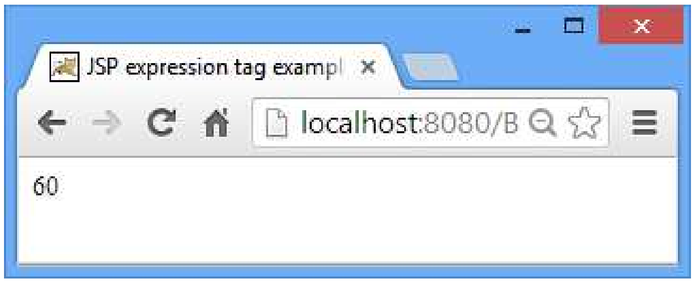
Example 2: Expression of variables

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.

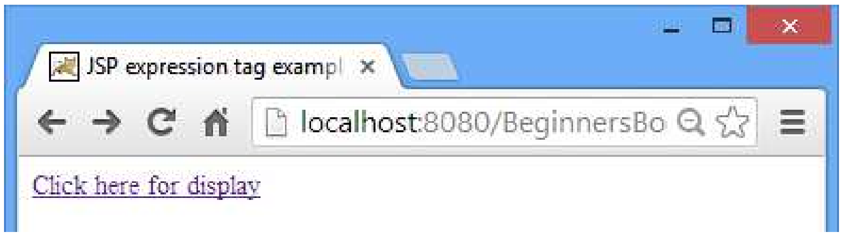
index.jsp

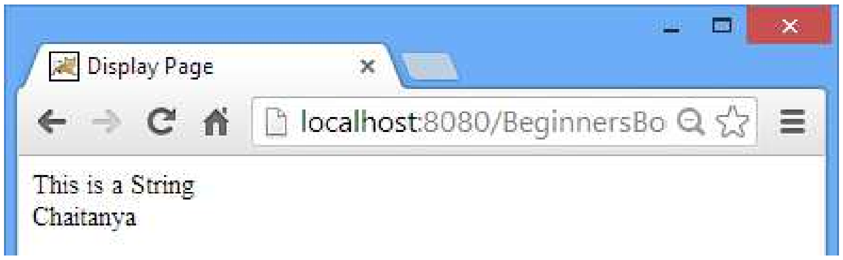
```
<html>
        <head>
                <title> JSP expression tag example3 </title>
        </head>
        <body>
                <% application.setAttribute("MyName", "Chaitanya"); %>
                <a href="display.jsp">Click here for display</a>
        </body>
</html>
```

display.jsp

```
<html>
        <head>
                <title>Display Page</title>
        </head>
        <body>
                <%="This is a String" %><br>
                <%= application.getAttribute("MyName") %>
        </body>
</html>
```

Output:




Display Page

localhost:8080/BeginnersBo

```
This is a String
Chaitanya
```

4.5 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.

4.6 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>
```

4.7 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.

4.8 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;
	}

}
```

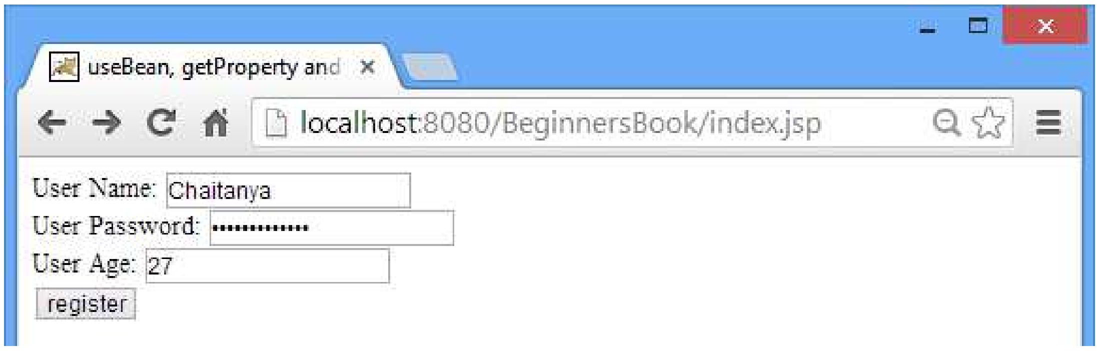
index.jsp

```
<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>
```

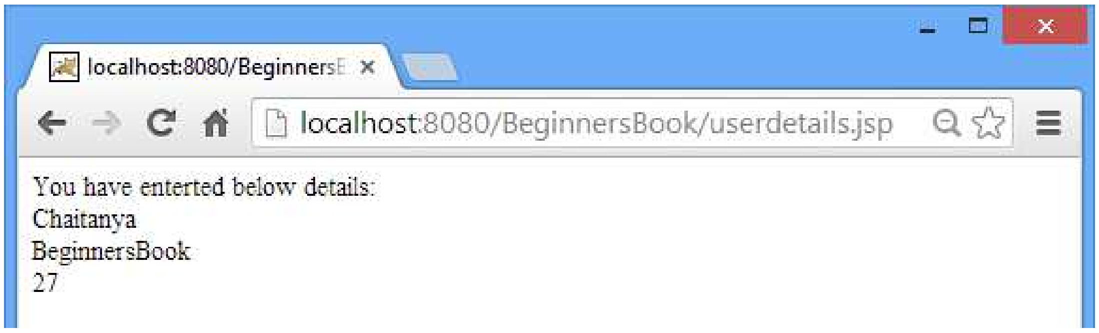
userdetails.jsp

```
<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
register
```


localhost:8080/BeginnersE
localhost:8080/BeginnersBook/userdetails.jsp

```
You have enterted below details:
Chaitanya
BeginnersBook
27
```

4.9 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

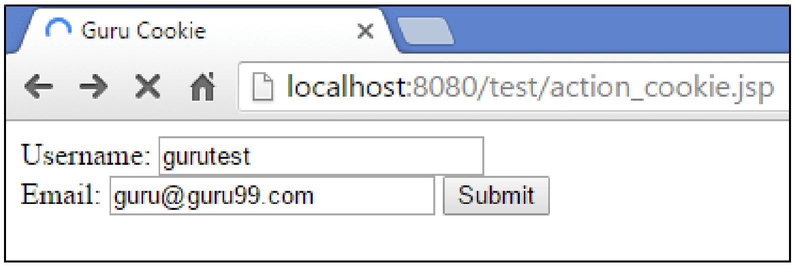
Code Line 6-9: Creating two cookie objects of "username" and "email" using request.getParameter.

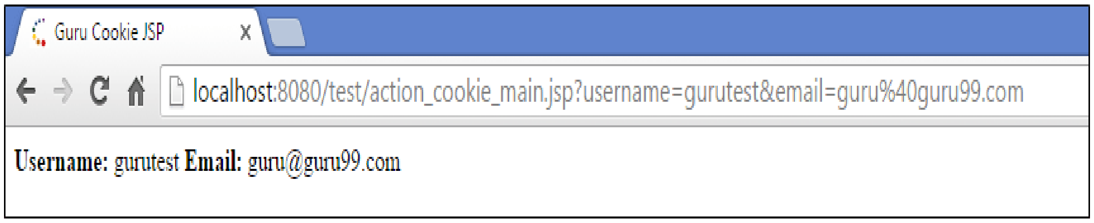
Code Line 12-13: Here we are adding age to both the cookies, which have been created of 10 hours i.e. cookies will expire in that age.

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.

4.10 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 −

```java
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 the following result will be displayed as :

```
Student First Name: Zara

Student Last Name: Ali

Student Age: 10
```

Module – V: Introduction to HTML and Java Script

Lists, Tables, Images, Forms, Frames, Cascading Style Sheets, Introduction to Java Script, Objects in Java Script, Dynamic HTML with Java Script.

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>
```

<u>Ordered HTML List</u>

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

<u>HTML Forms</u>

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)

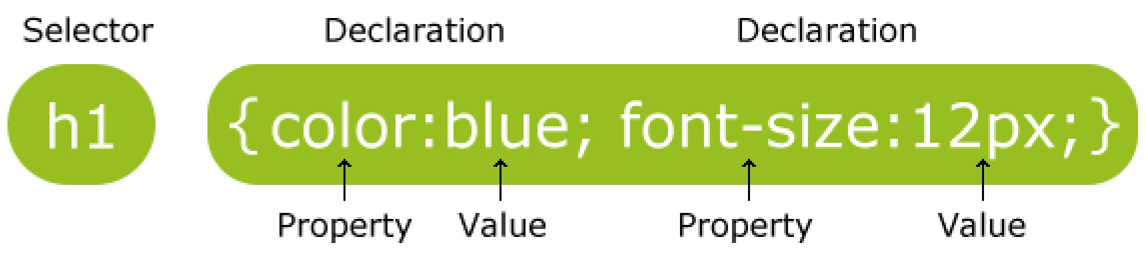
```
<frameset rows="25%,*,25%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
  <frame src="frame_c.htm">
</frameset>
```

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 letters

The 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');
}
```

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

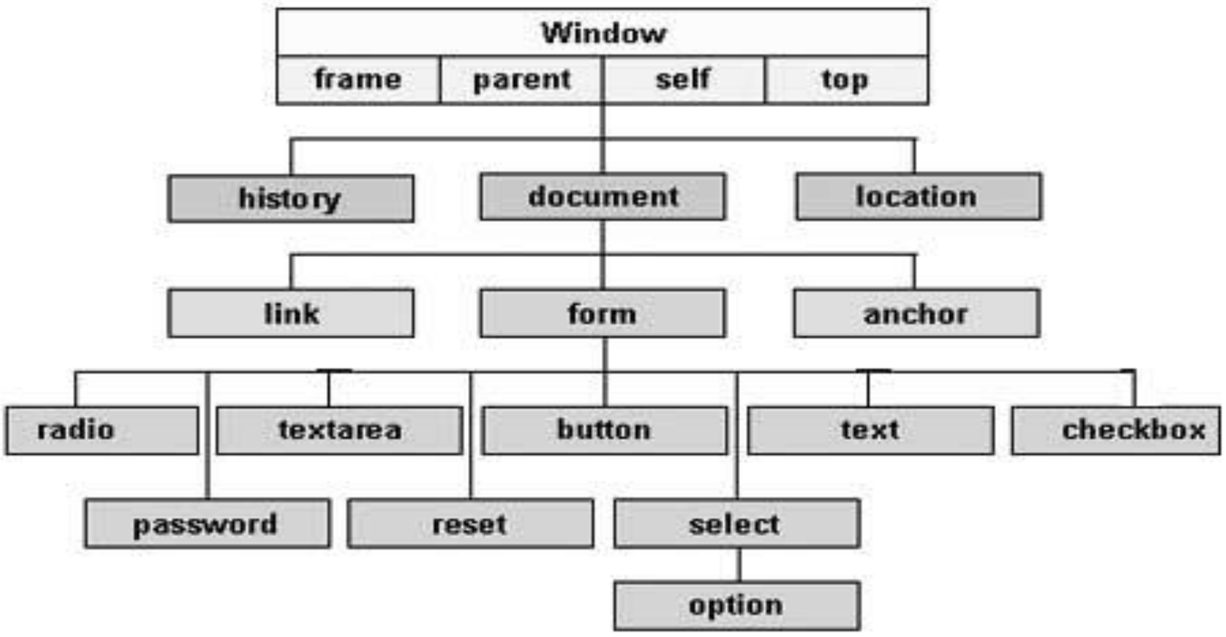
Every web page resides inside a browser window which can be considered as an object.

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.

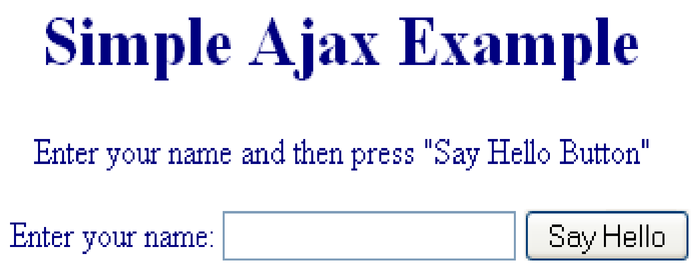
This application is using the XMLHttpRequest object for making a call to the server side script.

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.

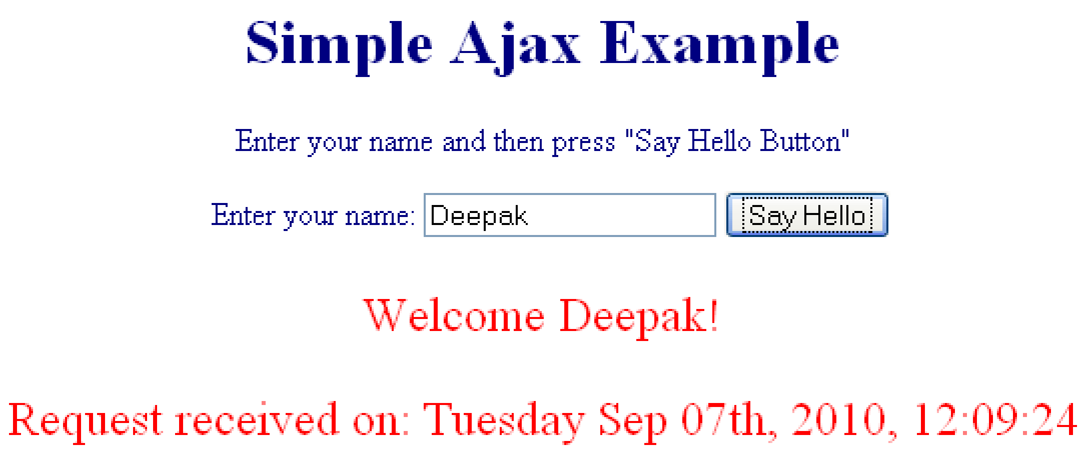
The application will display the form. User can then enter then name and press the "Say Hello" button as shown below in the screen shot.



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
Say Hello
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.responseText);
}
```

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").innerHTML =
 "<font color='red' size='5'>" + str + "</font>";;
}
```

In the php file we are retrieving the data from user and then sending the appropriate message to 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.net/ajax/Simple-Ajax-Example.shtml