4.1 PHP: INTRODUCTION TO PHP-USING PHP

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.

```
<html>
    <head>
    <title>Hello World</title>
  </head>
    <body>
    <?php echo "Hello, World!";?>
  </body>
</html>
```

Output

```
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>

A most common tag is the <?php...?>

4.2 VARIABLES

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 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.

4.2.1 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).

4.2.2 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>");

?>
```

Output

```
2.28888 + 2.21112 = 4.5
```

4.2.3 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>");
```

4.2.4 Interpreting other types as Booleans

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 = "";

4.2.5 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.

4.2.6 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);

?>
```

Output

```
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 (\)

4.2.7 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

4.2.8 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.

4.3 PROGRAM CONTROL

4.3.1 Decision Making

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.

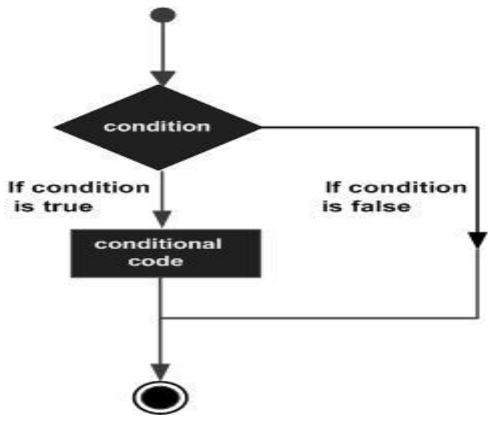


Fig: 4.1 Decision making

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>
```

Output

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>
```

Output:

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>
```

Output:

```
Today is Monday
```

4.3.2 Looping Statements

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

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

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.

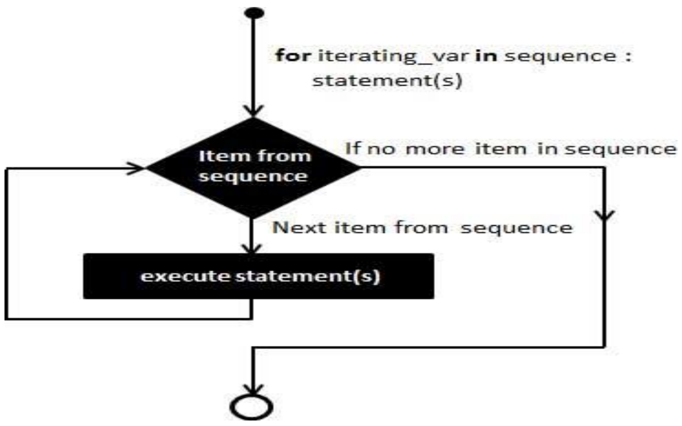


Fig: 4.2 for loop statement

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;
```

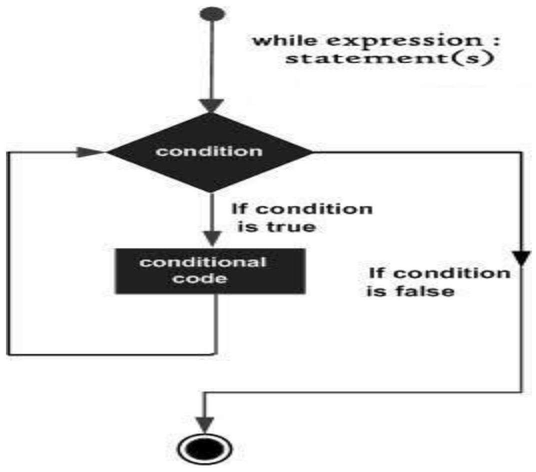
```
        $b += 5;

      }

            echo ("At the end of the loop a = $a and b = $b" );

  ?>

  </body>

</html>
```

Output:

```
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.

Fig:4.3 while loop statement

```
<html>

  <body>

      <?php

      $i = 0;

      $num = 50;

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

            $i++;

         }

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

      ?>

        </body>

</html>
```

Output

```
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>
```

Output:

```
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

```
<html>
  <body>
    <?php
      $array = array( 1, 2, 3, 4, 5);
      foreach( $array as $value ) {
        echo "Value is $value <br />";
      }
    ?>
  </body>
</html>
```

Output:

```
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>
```

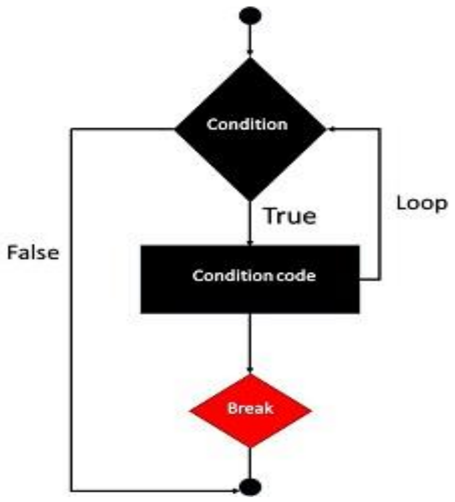


Fig: 4.4 Break statements

Output:

```
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>
```

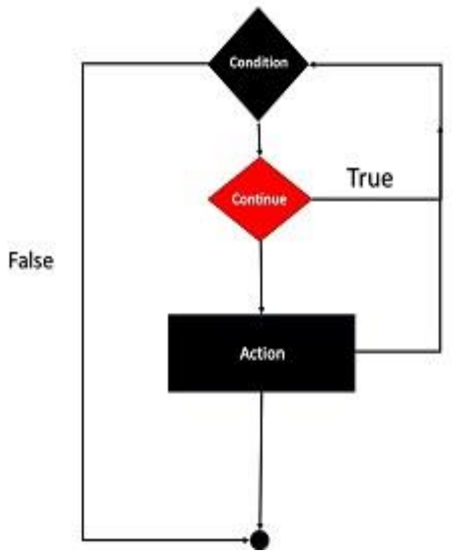


Fig: 4.5 continue statement

Output:

```
Value is 1

Value is 2

Value is 4

Value is 5
```

4.4 BUILT IN FUNCTIONS:

4.4.1 PHP Functions

- PHP function is a piece of code that can be reused many times. It can take input as argument list and return value. There are thousands of built-in functions in PHP.
- In PHP, we can define **Conditional function**, **Function within Function** and **Recursive function** also.

4.4.2 Advantage of PHP Functions

- **Code Reusability**: PHP functions are defined only once and can be invoked many times, like in other programming languages.
- **Less Code**: It saves a lot of code because you don't need to write the logic many times. By the use of function, you can write the logic only once and reuse it.

- **Easy to understand**: PHP functions separate the programming logic. So it is easier to understand the flow of the application because every logic is divided in the form of functions.

4.4.3 PHP User-defined Functions

Syntax

```
function functionname(){

//code to be executed

}
```

Note: Function name must be start with letter and underscore only like other labels in PHP. It can't be start with numbers or special symbols.

PHP Functions Example

File: function1.php

```
<?php

function sayHello(){

echo "Hello PHP Function";

}

sayHello();//calling function

?>
```

Output:

```
Hello PHP Function
```

4.4.4 PHP Function Arguments

- Information pass in PHP function through arguments which is separated by comma.
- PHP supports **Call by Value** (default), **Call by Reference**, **Default argument values** and **Variable-length argument list**.

Example

File: functionarg.php

```
<?php

function sayHello($name){
```

```
echo "Hello $name<br/>";
}
sayHello("Sonoo");
sayHello("Vimal");
sayHello("John");
?>
```

Output:

```
Hello Sonoo
Hello Vimal
Hello John
```

Example

File: functionarg2.php

```
<?php
function sayHello($name,$age){
echo "Hello $name, you are $age years old<br/>";
}
sayHello("Sonoo",27);
sayHello("Vimal",29);
sayHello("John",23);
?>
```

Output:

```
Hello Sonoo, you are 27 years old
Hello Vimal, you are 29 years old
Hello John, you are 23 years old
```

4.4.5 PHP Call By Reference

- Value passed to the function doesn't modify the actual value by default (call by value). But we can do so by passing value as a reference.
- By default, value passed to the function is call by value. To pass value as a reference, you need to use ampersand (&) symbol before the argument name.

Example:

File: functionref.php

```
<?php
function adder(&$str2)
{
   $str2 .= 'Call By Reference';
}
$str = 'Hello ';
adder($str);
echo $str;
?>
```

Output:

```
Hello Call By Reference
```

4.4.6 PHP Function: Default Argument Value

- We can specify a default argument value in function. While calling PHP function if you don't specify any argument, it will take the default argument. Let's see a simple example of using default argument value in PHP function.

File: functiondefaultarg.php

```
<?php
function sayHello($name="Sonoo"){
echo "Hello $name<br/>";
}
sayHello("Rajesh");
sayHello();//passing no value
sayHello("John");
?>
```

Output:

```
Hello Rajesh

Hello Sonoo
```

Hello John

4.4.7 PHP Function: Returning Value

Example:

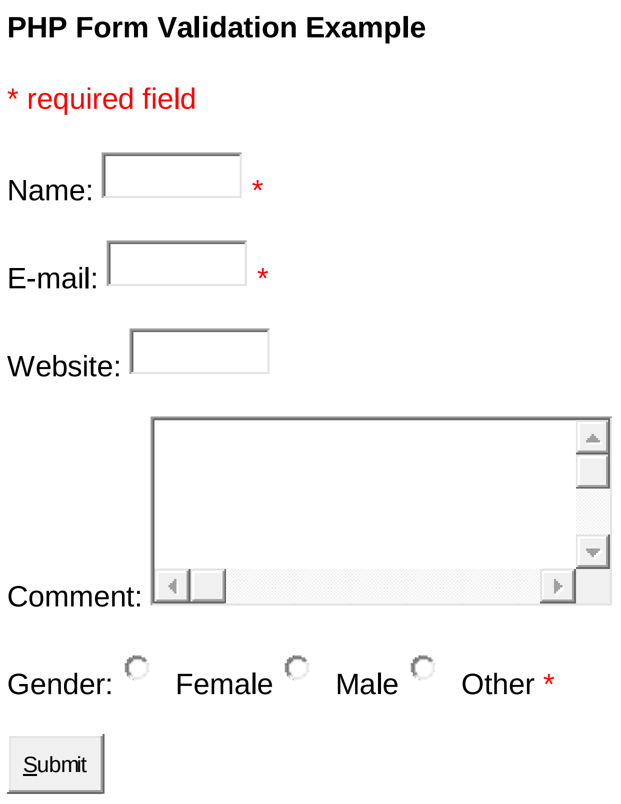
File: functiondefaultarg.php

```
<?php
function cube($n){
return $n*$n*$n;
}
echo "Cube of 3 is: ".cube(3);
?>
```

Output:

```
Cube of 3 is: 27
```

4.5 FORM VALIDATION



PHP Form Validation Example

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male Other *

Input:

The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

4.5.1 Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

```
Name: <input type="text" name="name">

E-mail: <input type="text" name="email">

Website: <input type="text" name="website">

Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

4.5.2 Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

```
Gender:

<input type="radio" name="gender" value="female">Female

<input type="radio" name="gender" value="male">Male

<input type="radio" name="gender" value="other">Other
```

4.5.3 The Form Element

The HTML code of the form looks like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

When the form is submitted, the form data is sent with method="post".

4.5.4 What is the $_SERVER["PHP_SELF"] variable?

- The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.
- So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

4.5.5 What is the htmlspecialchars() function?

- The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

4.5.6 Big Note on PHP Form Security

- The $_SERVER["PHP_SELF"] variable can be used by hackers!
- If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.
- Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.
- Assume we have the following form in a page named "test_form.php":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

- Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

```
<form method="post" action="test_form.php">
```

- However, consider that a user enters the following URL in the address bar:

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```

- In this case, the above code will be translated to:

```
<form method="post" action="test_form.php/"><script>alert('hacked')</script>
```

- This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.
- Be aware of that any JavaScript code can be added inside the <script> tag! A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How To Avoid $_SERVER["PHP_SELF"] Exploits?

$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The form code should look like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

- The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

<formmethod="post" action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">

Validate Form Data With PHP

- The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.
- When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

<script>location.href('http://www.hacked.com')</script>

- this would not be executed, because it would be saved as HTML escaped code, like this: &lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;

- The code is now safe to be displayed on a page or inside an e-mail.
- Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function).
- Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

Example

```
<?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;

}
```

?>

4.6 REGULAR EXPRESSIONS:

- Regular expressions are nothing more than a sequence or pattern of characters itself. They provide the foundation for pattern-matching functionality.
- Using regular expression you can search a particular string inside a another string, you can replace one string by another string and you can split a string into many chunks.
- PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression. You can use any of them based on your comfort.
    - ➢ POSIX Regular Expressions
    - ➢ PERL Style Regular Expressions

4.6.1 POSIX Regular Expressions

- ➢ The structure of a POSIX regular expression is not dissimilar to that of a typical arithmetic expression: various elements (operators) are combined to form more complex expressions.
- ➢ The simplest regular expression is one that matches a single character, such as g, inside strings such as g, haggle, or bag.

Brackets

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Sr.No	Expression & Description
1	[0-9]- It matches any decimal digit from 0 through 9.
2	[a-z]- It matches any character from lower-case a through lowercase z
3	[A-Z]- It matches any character from uppercase A through uppercase Z.
4	[a-Z]- It matches any character from lowercase a through uppercase Z.

- The ranges shown above are general; you could also use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from b through v.

Quantifiers

- The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, *, ?, {int. range}, and $ flags all follow a character sequence.

Sr.No	Expression & Description	
1	p+	It matches any string containing at least one p.
2	p*	It matches any string containing zero or more p's.
3	p?	It matches any string containing zero or one p's.
4	p{N}	It matches any string containing a sequence of N p's
5	p{2,3}	It matches any string containing a sequence of two or three p's.
6	p{2, }	It matches any string containing a sequence of at least two p's.
7	p$	It matches any string with p at the end of it.
8	^p	It matches any string with p at the beginning of it.

Examples

Following examples will clear your concepts about matching characters.

Sr.No	Expression & Description	
1	[^a-zA-Z]	It matches any string not containing any of the characters ranging from a through z and A through Z.
2	p.p	It matches any string containing p, followed by any character, in turn followed by another p.
3	^.{2}$	It matches any string containing exactly two characters.
4	<b>(.*)</b>	It matches any string enclosed within <b> and </b>.
5	p(hp)*	It matches any string containing a p followed by zero or more instances of the sequence php.

Predefined Character Ranges

- For your programming convenience several predefined character ranges, also known as character classes, are available. Character classes specify an entire range of characters, for example, the alphabet or an integer set –

Sr.No	Expression & Description	
1	[[:alpha:]]	It matches any string containing alphabetic characters aA through zZ.
2	[[:digit:]]	It matches any string containing numerical digits 0 through 9.
3	[[:alnum:]]	It matches any string containing alphanumeric characters aA through zZ and 0 through 9.
4	[[:space:]]	It matches any string containing a space.

PHP's Regexp POSIX Functions

PHP currently offers seven functions for searching strings using POSIX-style regular expressions –

Sr.No	**Functions & Description**	
1	ereg()	The ereg() function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise.
2	ereg_replace()	The ereg_replace() function searches for string specified by pattern and replaces pattern with replacement if found.
3	eregi()	The eregi() function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive.
4	eregi_replace()	The eregi_replace() function operates exactly like ereg_replace(), except that the search for pattern in string is not case sensitive.
5	split()	The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
6	spliti()	The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive.
7	sql_regcase()	The sql_regcase() function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters.

PERL Style Regular Expressions

- Perl-style regular expressions are similar to their POSIX counterparts. The POSIX syntax can be used almost interchangeably with the Perl-style regular expression functions. In fact, you can use any of the quantifiers introduced in the previous POSIX section.

Sr.No	Functions & Description	
1	preg_match()	The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise.
2	preg_match_all()	The preg_match_all() function matches all occurrences of pattern in string.
3	preg_replace()	The preg_replace() function operates just like ereg_replace(), except that regular expressions can be used in the pattern and replacement input parameters.
4	preg_split()	The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.
5	preg_grep()	The preg_grep() function searches all elements of input_array, returning all elements matching the regexp pattern.
6	preg_ quote()	Quote regular expression characters

Meta characters

- A meta character is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.
- For instance, you can search for large money sums using the '\d' meta character: **/([\d]+)000/**, Here **\d** will search for any string of numerical character.
- Following is the list of meta characters which can be used in PERL Style Regular Expressions.

Modifiers

Several modifiers are available that can make your work with regexps much easier, like case sensitivity, searching in multiple lines etc.

Modifier	**Description**
i	Makes the match case insensitive
m	Specifies that if the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of a

	string boundary
o	Evaluates the expression only once
s	Allows use of . to match a newline character
x	Allows you to use white space in the expression for clarity
g	Globally finds all matches
cg	Allows a search to continue even after a global match fails

PHP's Regexp PERL Compatible Functions

PHP offers following functions for searching strings using Perl-compatible regular expressions −

4.7 FILE HANDLING

Functions related to files −

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

4.7.1 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.

Sr.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.

- 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.

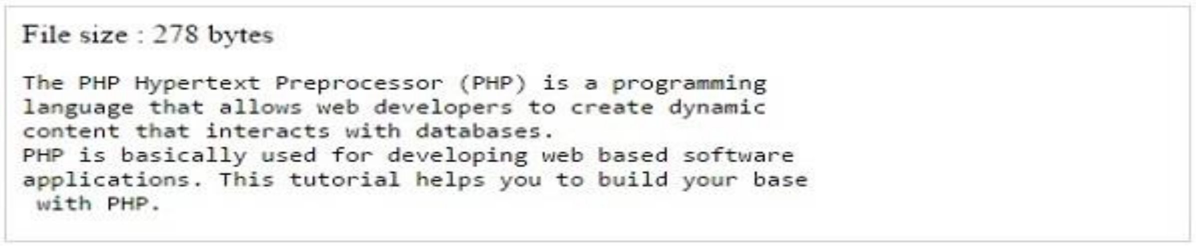
- After making a changes to the opened file it is important to close it with the fclose() function. The fclose() function requires a file pointer as its argument and then returns true when the closure succeeds or false if it fails.

4.7.2 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>
```

Output:



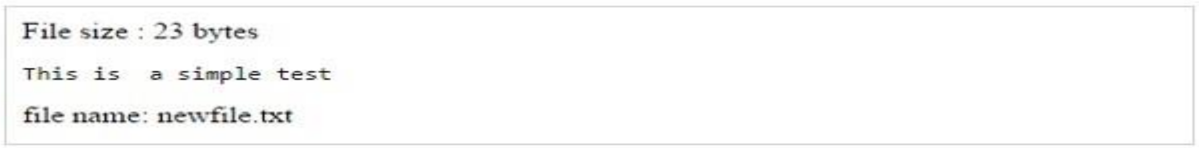
```
File size : 278 bytes

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.
```

4.7.3 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>
```

Output:



```
File size : 23 bytes
This is  a simple test
file name: newfile.txt
```

4.8 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.

4.8.1 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

- 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"].

4.8.2 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>
```

4.8.3 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>
```

4.8.4 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>
```

4.9 CONNECTION TO DATABASE

Since PHP 5.5, **mysql_connect()** extension is *deprecated*. Now it is recommended to use one of the 2 alternatives.

- **mysqli_connect()**
- **PDO::__construct()**

4.9.1 PHP mysqli_connect()

PHP mysqli_connect() function is used to connect with MySQL database. It returns resource if connection is established or null.

Syntax

resource mysqli_connect (server, username, password)

4.9.2 PHP mysqli_close()

PHP mysqli_close() function is used to disconnect with MySQL database. It returns true if connection is closed or false.

Syntax

bool mysqli_close(resource $resource_link)

4.9.3 PHP MySQL Connect Example

Example

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$conn = mysqli_connect($host, $user, $pass);
if(! $conn )
{
  die('Could not connect: ' . mysqli_error());
}
echo 'Connected successfully';
mysqli_close($conn);
?>
```

Output:

```
Connected successfully
```

4.10 XML: BASIC XML

XML stands for **Ex**tensible **M**arkup **L**anguage and is a text-based markup language derived from Standard Generalized Markup Language (SGML).

4.11 DOCUMENT TYPE DEFINITIONS

- The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.
- An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Basic syntax of a DTD is as follows −

```
<!DOCTYPE element DTD identifier
[
  declaration1
  declaration2
  ........
]>
```

In the above syntax,

- The DTD starts with <!DOCTYPE delimiter.
- An element tells the parser to parse the document from the specified root element.
- DTD identifier is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called External Subset.
- The square brackets [ ] enclose an optional list of entity declarations called Internal Subset.

4.11.1 Internal DTD

- A DTD is referred to as an internal DTD if elements are declared within the XML files.
- To refer it as internal DTD, standalone attribute in XML declaration must be set to yes. This means, the declaration works independent of an external source.

Syntax

Following is the syntax of internal DTD −

```
<!DOCTYPE root-element [element-declarations]>
```

- where root-element is the name of root element and element-declarations is where you declare the elements.

Example

Following is a simple example of internal DTD −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
```

```
<!DOCTYPE address [
  <!ELEMENT address (name,company,phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT company (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

Let us go through the above code −

- **Start Declaration** − Begin the XML declaration with the following statement.

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
```

- **DTD** − Immediately after the XML header, the document type declaration follows, commonly referred to as the DOCTYPE −

```
<!DOCTYPE address [
```

- The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.
- DTD Body − The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations.

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

- Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element name to be of type "#PCDATA". Here #PCDATA means parse-able text data.
- **End Declaration** − Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document follows immediately.

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) − it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

4.11.2 External DTD

- In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL. To refer it as external DTD, standalone attribute in the XML declaration must be set as no. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD −

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where file-name is the file with .dtd extension.

Example

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

- The content of the DTD file address.dtd is as shown −

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

4.11.3 Types

System Identifiers

- A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows −

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

Public Identifiers

- Public identifiers provide a mechanism to locate DTD resources and is written as follows

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

- Keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called Formal Public Identifiers, or FPIs.

4.12 XML SCHEMA DOM AND PRESENTING XML

4.12.1 XML Schema

- XML Schema is commonly known as XML Schema Definition (XSD).
- It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types.
- Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

Example

The following example shows how to use schema −

```
<?xml version = "1.0" encoding = "UTF-8"?>

<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

  <xs:element name = "contact">

    <xs:complexType>

      <xs:sequence>

        <xs:element name = "name" type = "xs:string" />

        <xs:element name = "company" type = "xs:string" />

        <xs:element name = "phone" type = "xs:int" />

      </xs:sequence>

    </xs:complexType>

  </xs:element>

</xs:schema>
```

- The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

Elements

Elements are the building blocks of XML document. An element can be defined within an XSD as follows −

```
<xs:element name = "x" type = "y"/>
```

Definition Types

XML schema can be define elements in the following ways −

Simple Type

Simple type element is used only in the context of the text. Some of the predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example −

```
<xs:element name = "phone_number" type = "xs:int" />
```

Complex Type

A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example −

```
<xs:element name = "Address">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "name" type = "xs:string" />
      <xs:element name = "company" type = "xs:string" />
      <xs:element name = "phone" type = "xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In the above example, *Address* element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types

With the global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as follows −

```
<xs:element name = "AddressType">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "name" type = "xs:string" />
      <xs:element name = "company" type = "xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Now use this type in our example as follows −

```
<xs:element name = "Address1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "address" type = "AddressType" />
      <xs:element name = "phone1" type = "xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name = "Address2">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "address" type = "AddressType" />
      <xs:element name = "phone2" type = "xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition.
- This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes

Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below –

<xs:attribute name = “x” type = “y”/>

4.12.2 XML DOM

- The Document Object Model (DOM) is the foundation of XML. XML documents have a hierarchy of informational units called *nodes*; DOM is a way of describing those nodes and the relationships between them.
- A DOM document is a collection of nodes or pieces of information organized in a hierarchy. This hierarchy allows a developer to navigate through the tree looking for specific information. Because it is based on a hierarchy of information, the DOM is said to be *tree based*.
- The XML DOM, on the other hand, also provides an API that allows a developer to add, edit, move, or remove nodes in the tree at any point in order to create an application.

Example

- The following example (sample.htm) parses an XML document ("address.xml") into an XML DOM object and then extracts some information from it with JavaScript −

```
<!DOCTYPE html>

<html>

  <body>

    <h1>TutorialsPoint DOM example </h1>

    <div>

      <b>Name:</b> <span id = "name"></span><br>

      <b>Company:</b> <span id = "company"></span><br>

      <b>Phone:</b> <span id = "phone"></span>

    </div>

    <script>

      if (window.XMLHttpRequest)
```

```
        {// code for IE7+, Firefox, Chrome, Opera, Safari
          xmlhttp = new XMLHttpRequest();
        }
        else
        {// code for IE6, IE5
          xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        xmlhttp.open("GET","/xml/address.xml",false);
        xmlhttp.send();
        xmlDoc = xmlhttp.responseXML;
        document.getElementById("name").innerHTML=
          xmlDoc.getElementsByTagName("name")[0].childNodes[0].nodeValue;
        document.getElementById("company").innerHTML=
          xmlDoc.getElementsByTagName("company")[0].childNodes[0].nodeValue;
        document.getElementById("phone").innerHTML=
          xmlDoc.getElementsByTagName("phone")[0].childNodes[0].nodeValue;
    </script>
  </body>
</html>
```

Contents of address.xml are as follows −

```
<?xml version = "1.0"?>
<contact-info>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</contact-info>
```

4.13 XML PARSER AND VALIDATION

4.13.1 XML Parser

- **XML parser** is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers.
- Following diagram shows how XML parser interacts with XML document −

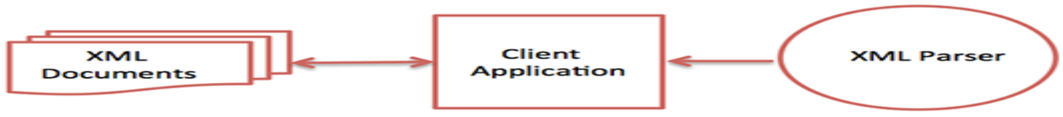


Fig:4.6 XML Parser

- The goal of a parser is to transform XML into a readable code.
- To ease the process of parsing, some commercial products are available that facilitate the breakdown of XML document and yield more reliable results.
- Some commonly used parsers are listed below −
    - **MSXML (Microsoft Core XML Services)** − This is a standard set of XML tools from Microsoft that includes a parser.
    - **System.Xml.XmlDocument** − This class is part of .NET library, which contains a number of different classes related to working with XML.
    - **Java built-in parser** − The Java library has its own parser. The library is designed such that you can replace the built-in parser with an external implementation such as Xerces from Apache or Saxon.
    - **Saxon** − Saxon offers tools for parsing, transforming, and querying XML.
    - **Xerces** − Xerces is implemented in Java and is developed by the famous open source Apache Software Foundation.

4.13.2 XML Validation

- Validation is a process by which an XML document is validated. An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration (DTD), and if the document complies with the constraints expressed in it. Validation is dealt in two ways by the XML parser. They are −
    - Well-formed XML document
    - Valid XML document
    - Well-formed XML Document
- An XML document is said to be well-formed if it adheres to the following rules −
    - Non DTD XML files must use the predefined character entities for amp(&), apos(single quote), gt(>), lt(<), quot(double quote).
    - It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
    - Each of its opening tags must have a closing tag or it must be a self ending tag.(<title>....</title> or <title/>).It must have only one attribute in a start tag, which needs to be quoted.
    - amp(&), apos(single quote), gt(>), lt(<), quot(double quote) entities other than these must be declared.

Example

Following is an example of a well-formed XML document −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address
[
  <!ELEMENT address (name,company,phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT company (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

The above example is said to be well-formed as −

- It defines the type of document. Here, the document type is element type.
- It includes a root element named as address.
- Each of the child elements among name, company and phone is enclosed in its self explanatory tag.
- Order of the tags is maintained.

Valid XML Document

- ➢ If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document.

4.14 XSL AND XSLT TRANSFORMATION

- XSL which stands for E**X**tensible **S**tylesheet **L**anguage. It is similar to XML as CSS is to HTML.

Need for XSL

- In case of HTML document, tags are predefined such as table, div, and span; and the browser knows how to add style to them and display those using CSS styles.
- But in case of XML documents, tags are not predefined. In order to understand and style an XML document, World Wide Web Consortium (W3C) developed XSL which can act as XML based Stylesheet Language. An XSL document specifies how a browser should render an XML document.
- Following are the main parts of XSL –
  - ✓ **XSLT** − used to transform XML document into various other types of document.
  - ✓ **XPath** − used to navigate XML document.
  - ✓ **XSL-FO** − used to format XML document.

What is XSLT?

✓ XSLT, Extensible Stylesheet Language Transformations, provides the ability to transform XML data from one format to another automatically.

How XSLT Works?

- An XSLT stylesheet is used to define the transformation rules to be applied on the target XML document. XSLT stylesheet is written in XML format.
- XSLT Processor takes the XSLT stylesheet and applies the transformation rules on the target XML document and then it generates a formatted document in the form of XML, HTML, or text format.
- This formatted document is then utilized by XSLT formatter to generate the actual output which is to be displayed to the end-user.

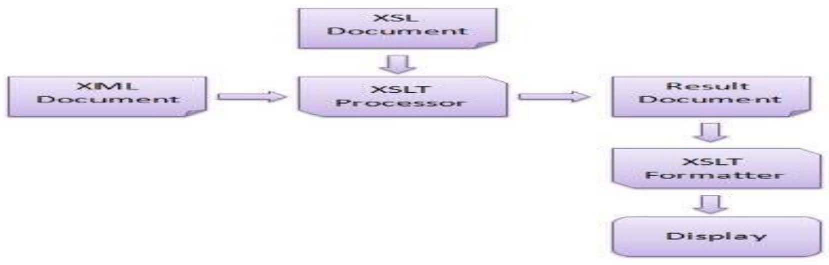


Fig: 4.7 XSLT Transformations

Advantages:

Here are the advantages of using XSLT −

- Independent of programming. Transformations are written in a separate xsl file which is again an XML document.

- Output can be altered by simply modifying the transformations in xsl file. No need to change any code. So Web designers can edit the stylesheet and can see the change in the output quickly.

4.15 NEWS FEED (RSS AND ATOM)

4.15.1 RSS News feed:

- RSS is the short name of Really Simple Syndication.
- RSS is like a broadcast system which is used to deliver updated information to multiple receivers.
- RSS is used to deliver regular changing web contents to the user.

RSS Feed Basic Structure:

```
<?xml version='1.0' encoding='UTF-8'?>

<rss version='2.0'>

<channel>

<title>Title of Webpage</title>

<link>Webpage URL</link>

<description>About Webpage</description>

<language>en-us</language>

<item>

  <title>Article Title</title>

  <link>Article URL</link>

  <description>Article Content</description>

</item>

</channel>

</rss>
```