Unit-4

PHP and XML 8

INTRODUCTION TO PHP

The PHP Hypertext Pre-processor (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.PHP is probably the most popular scripting language on the web. It is used to enhance web pages.PHP is known as a **server-sided language.** That is because the PHP doesn't get executed on the client's computer, but on the computer the user had requested the page from. The results are then handed over to client, and then displayed in the browser.

Features of PHP:

- ❖ PHP is a server side scripting language that is embedded in HTML
- ❖ PHP was originally developed by the **Danish Greenlander Rasmus Lerdorf**, and was subsequently developed as open source.
- ❖ 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 supports a large number of 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 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 the user can send data, return data to the user.
- The user can add, delete, and modify elements within the database through PHP.
- They can access cookies variables and set cookies.
- Using PHP, the user can restrict users to access some pages of the website.
- It can encrypt data.

Working of PHP

When the client requests a PHP page residing on the server, the server first performs the operations mentioned by the PHP code of the page. Then is sends the output of the PHP page in HTML format. So when the user views the source code of the page, it will be full of HTML tags. All the work is done at the server side.

4.1 PROGRAMMING WITH PHP

- **Comments**

A comment is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP:

- **Single-line comments:** They are generally used for short explanations or notes relevant to the local code.
- **Multi-line comments:** They are generally used to provide pseudocode algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C.

Rules of PHP

- PHP is white space insensitive
- PHP is case sensitive
- Statements are expressions terminated by semicolons
- Expressions are combinations of tokens
- Braces make blocks

- **PHP Variable Types**

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 in PHP do not have **intrinsic types (data 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 automatically converts types from one to another when necessary. PHP has a total of eight data types:

- Integers are whole numbers, without a decimal point. They can be in decimal, octal or hexadecimal. Eg: 87.
- Doubles are floating-point numbers. Eg: 3.87
- 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
- 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 arrays and objects are compound types.
- The compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

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 and Static variables.

Variable Naming

Rules for naming a variable are:

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

PHP Constants

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. To define a constant , use define() function and retrieve the value of a constant. The function constant() is used to read a constant's value .

define(name, value, case-insensitive)

The name specifies the name of the constant, value: Specifies the value of the constant and case-insensitive: Specifies whether the constant name should be case-insensitive. Default is false

Differences between constants and variables in PHP

Constants in PHP	Variables in PHP
No $ sign before constants.	$ sign is present before variables.
Constants are defined using define().	Variables are defined using assignment statement.
Constants may be defined and accessed anywhere without any regard.	Variables follow certain scope.
Constants cannot be redefined or undefined.	They can be redefined.

Pre-defined constants

Name	Description
__LINE____	The current line number of the file.
__FILE____	The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP 4.0.2, __FILE___always contains an absolute path whereas in older versions it contained relative path under some circumstances
__FUNCTION____	The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__CLASS____	The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__METHOD____	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

➢ Echo and Print statements

Differences between echo and print statement

echo	print
This does not have return value.	This has a return value of 1.
This cannot be used in expressions.	This can be used in expressions.
This can take multiple parameters.	This can take only one parameter.
This is slightly faster than print.	This is comparatively slower.

➢ Operators

Operators are used to perform operations on variables and values.PHP divides the operators in the following groups: Arithmetic operators, Assignment operators, Comparison operators, Increment/Decrement operators, Logical operators, String operators and Array operators.

PHP CONTROL STATMENTS

Decision Making Statements

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

- if statement - executes some code only if a specified condition is true.
- if...else statement - executes some code if a condition is true and another code if the condition is false.
- if...else if ...else statement - specifies a new condition to test, if the first condition is false
- switch statement - selects one of many blocks of code to be executed

➢ if statement

if statement

```
<?php
$t = date("H");
if ($t < "20") //This program outputs the echo statement if the hour is <20
{
   echo "Have a good day!";
}
?>
```

- **if-else statement**

  Use the if ...else statement to execute some code if a condition is true and another code if the condition is false.

if else statement

```
<?php
$t = date("H");
if ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

- **if-else ladder**

  Use the if....else if...else statement to specify a new condition to test, if the first condition is false.

if-else ladder

```
<?php
$t = date("H");
if ($t < "10") {
    echo "Have a good morning!";
} elseif ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

- **switch statement**

  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 switch statement works in an unusual way. First it evaluates given expression then seeks a label 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 labels match then statement will execute any specified default code.

Switch statement

```
<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>
```

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.

➢ For loop

for 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>
```

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

➢ While loop

The while statement will execute a block of code 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.

While loop

```
<html> <body>
<?php
$i = 0;
$num = 50;
while( $i < 10)
{
$num--;
$i++;
}
echo ("Loop stopped at i = $i and num = $num" );
?> </body></html>
```

```
Loop stopped at i = 1 and num = 40
```

- **Do…while loop**

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

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

- **For Each loop**

The for each 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**.**

```
<html> <body>
<?
php $array = array( 11, 12, 13,14, 15);
```

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

```
Value is 11
Value is 12
Value is 13
Value is 14
Value is 15
```

- **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. After coming out of a loop immediate statement to the loop will be executed**.**

- **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.

FUNCTIONS

A function is a block of statements that can be used repeatedly in a program. A function will not execute immediately when a page loads. A function will be executed by a call to the function. There are two types of functions: Built-in functions and User defined functions

User defined Functions

A user defined function declaration starts with the word function. Information can be passed to functions through arguments. An argument is just like a variable. Arguments are specified after the function name, inside the parentheses.

```
<?php
function sum($x, $y) {
  $z = $x + $y;
  return $z; }
echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";

echo "2 + 4 = " . sum(2, 4); ?>
```

```
5 + 10 = 15

7 + 13 = 20

2 + 4 = 6
```

Built-in functions

Array functions

Function	Description
array()	Creates an array
array_chunk()	Splits an array into chunks of arrays
array_column()	Returns the values from a single column in the input array
array_combine()	Creates an array by using the elements from one "keys" array and one "values" array
array_count_values()	Counts all the values of an array
array_diff()	Compare arrays, and returns the differences (compare values only)
array_diff_key()	Compare arrays, and returns the differences (compare keys only)
array_fill()	Fills an array with values
array_fill_keys()	Fills an array with values, specifying keys
array_filter()	Filters the values of an array using a callback function
array_flip()	Flips/Exchanges all keys with their associated values in an array
array_intersect()	Compare arrays, and returns the matches (compare values only)
array_key_exists()	Checks if the specified key exists in the array
array_keys()	Returns all the keys of an array
array_map()	Sends each value of an array to a user-made function, which returns new values
array_merge()	Merges one or more arrays into one array
array_multisort()	Sorts multiple or multi-dimensional arrays

array_pad()	Inserts a specified number of items, with a specified value, to an array
array_pop()	Deletes the last element of an array
array_product()	Calculates the product of the values in an array
array_push()	Inserts one or more elements to the end of an array
array_rand()	Returns one or more random keys from an array
array_reduce()	Returns an array as a string, using a user-defined function
array_replace()	Replaces the values of the first array with the values from following arrays
array_replace_recursive()	Replaces the values of the first array with the values from following arrays recursively
array_reverse()	Returns an array in the reverse order
array_search()	Searches an array for a given value and returns the key
array_shift()	Removes the first element from an array, and returns the value of the removed element
array_slice()	Returns selected parts of an array
array_splice()	Removes and replaces specified elements of an array
array_sum()	Returns the sum of the values in an array
array_udiff()	Compare arrays, and returns the differences (compare values only, using a user-defined key comparison function)
array_uintersect()	Compare arrays, and returns the matches (compare values only, using a user-defined key comparison function)
array_unique()	Removes duplicate values from an array
array_unshift()	Adds one or more elements to the beginning of an array
array_values()	Returns all the values of an array
array_walk()	Applies a user function to every member of an array
arsort()	Sorts an associative array in descending order, according to the value
asort()	Sorts an associative array in ascending order, according to the value
compact()	Create array containing variables and their values

count()	Returns the number of elements in an array
current()	Returns the current element in an array
each()	Returns the current key and value pair from an array
end()	Sets the internal pointer of an array to its last element
extract()	Imports variables into the current symbol table from an array
in_array()	Checks if a specified value exists in an array
key()	Fetches a key from an array
list()	Assigns variables as if they were an array
natcasesort()	Sorts an array using a case insensitive "natural order" algorithm
natsort()	Sorts an array using a "natural order" algorithm
next()	Advance the internal array pointer of an array
prev()	Rewinds the internal array pointer
range()	Creates an array containing a range of elements
reset()	Sets the internal pointer of an array to its first element
rsort()	Sorts an indexed array in descending order
shuffle()	Shuffles an array
sort()	Sorts an indexed array in ascending order
uasort()	Sorts an array by values using a user-defined comparison function
uksort()	Sorts an array by keys using a user-defined comparison function
usort()	Sorts an array using a user-defined comparison function

- **Calendar Functions**

The calendar extension contains functions that simplify converting between different calendar formats. It is based on the Julian Day Count, which is a count of days starting from January 1st, 4713 B.C. To convert between calendar formats, first convert to Julian Day Count, then to the calendar of the user’s choice.

Function	Description
cal_days_in_month()	Returns the number of days in a month for a specified year and calendar
cal_from_jd()	Converts a Julian Day Count into a date of a specified calendar

cal_info()	Returns information about a specified calendar
cal_to_jd()	Converts a date in a specified calendar to Julian Day Count
easter_date()	Returns the Unix timestamp for midnight on Easter of a specified year
easter_days()	Returns the number of days after March 21, that the Easter Day is in a specified year
gregoriantojd()	Converts a Gregorian date to a Julian Day Count
jddayofweek()	Returns the day of the week
jdmonthname()	Returns a month name
jdtogregorian()	Converts a Julian Day Count to a Gregorian date
jdtounix()	Converts Julian Day Count to Unix timestamp
jewishtojd()	Converts a Jewish date to a Julian Day Count
juliantojd()	Converts a Julian date to a Julian Day Count
unixtojd()	Converts Unix timestamp to Julian Day Count

- **Date Functions**

The date/time functions allow to get the date and time from the server on which PHP script runs. These functions depend on the locale settings of the server. Remember to take daylight saving time and leap years into consideration when working with these functions.

Function	Description
checkdate()	Validates a Gregorian date
date_add()	Adds days, months, years, hours, minutes, and seconds to a date
date_create_from_format()	Returns a new DateTime object formatted according to a specified format
date_create()	Returns a new DateTime object
date_date_set()	Sets a new date
date_default_timezone_get()	Returns the default timezone used by all date/time functions
date_default_timezone_set()	Sets the default timezone used by all date/time functions

date_diff()	Returns the difference between two dates
date_format()	Returns a date formatted according to a specified format
date_interval_format()	Formats the interval
date_isodate_set()	Sets the ISO date
date_modify()	Modifies the timestamp
date_parse()	Returns an associative array with detailed info about a specified date
date_sub()	Subtracts days, months, years, hours, minutes, and seconds from a date
date_sun_info()	Returns an array containing info about sunset/sunrise and twilight begin/end, for a specified day and location
date_sunrise()	Returns the sunrise time for a specified day and location
date_sunset()	Returns the sunset time for a specified day and location
date_time_set()	Sets the time
date()	Formats a local date and time
getdate()	Returns date/time information of a timestamp or the current local date/time
gettimeofday()	Returns the current time
gmdate()	Formats a GMT/UTC date and time
gmmktime()	Returns the Unix timestamp for a GMT date
gmstrftime()	Formats a GMT/UTC date and time according to locale settings
idate()	Formats a local time/date as integer
localtime()	Returns the local time
microtime()	Returns the current Unix timestamp with microseconds
mktime()	Returns the Unix timestamp for a date

strftime()	Formats a local time and/or date according to locale settings
time()	Returns the current time as a Unix timestamp
timezone_name_get()	Returns the name of the timezone
timezone_offset_get()	Returns the timezone offset from GMT
timezone_open()	Creates new DateTimeZone object
timezone_version_get()	Returns the version of the timezone db

- **Directory functions**

The directory function allows retrieving information about directories and their contents.

Function	Description
chdir()	Changes the current directory
chroot()	Changes the root directory
closedir()	Closes a directory handle
dir()	Returns an instance of the Directory class
getcwd()	Returns the current working directory
opendir()	Opens a directory handle
readdir()	Returns an entry from a directory handle
rewinddir()	Resets a directory handle
scandir()	Returns an array of files and directories of a specified directory

- **Error handling functions**

The error functions are used to deal with error handling and logging. The error functions allow us to define own error handling rules, and modify the way the errors can be logged. The logging functions allow us to send messages directly to other machines, emails, or system logs. The error reporting functions allow us to customize what level and kind of error feedback is given.

Function	Description
debug_backtrace()	Generates a backtrace
debug_print_backtrace()	Prints a backtrace
error_get_last()	Returns the last error that occurred
error_log()	Sends an error message to a log, to a file, or to a mail account
error_reporting()	Specifies which errors are reported
restore_error_handler()	Restores the previous error handler
restore_exception_handler()	Restores the previous exception handler
set_error_handler()	Sets a user-defined error handler function
set_exception_handler()	Sets a user-defined exception handler function
trigger_error()	Creates a user-level error message
user_error()	Alias of trigger_error()
debug_backtrace()	Generates a backtrace

- **File system Functions**

The file system functions allow the user to access and manipulate the file system.

Function	Description
basename()	Returns the filename component of a path
chgrp()	Changes the file group
chmod()	Changes the file mode
chown()	Changes the file owner
clearstatcache()	Clears the file status cache
copy()	Copies a file
dirname()	Returns the directory name component of a path
disk_free_space()	Returns the free space of a directory
disk_total_space()	Returns the total size of a directory

fclose()	Closes an open file
feof()	Tests for end-of-file on an open file
fflush()	Flushes buffered output to an open file
fgetc()	Returns a character from an open file
fgets()	Returns a line from an open file
fgetss()	Returns a line, with HTML and PHP tags removed, from an open file
file()	Reads a file into an array
file_exists()	Checks whether or not a file or directory exists
file_get_contents()	Reads a file into a string
file_put_contents()	Writes a string to a file
fileatime()	Returns the last access time of a file
filectime()	Returns the last change time of a file
filegroup()	Returns the group ID of a file
fileinode()	Returns the inode number of a file
filemtime()	Returns the last modification time of a file
fileowner()	Returns the user ID (owner) of a file
fileperms()	Returns the permissions of a file
filesize()	Returns the file size
filetype()	Returns the file type
flock()	Locks or releases a file
fnmatch()	Matches a filename or string against a specified pattern
fopen()	Opens a file or URL
fpassthru()	Reads from an open file, until EOF, and writes the result to the output buffer
fputcsv()	Formats a line as CSV and writes it to an open file
fread()	Reads from an open file
fscanf()	Parses input from an open file according to a specified format

fseek()	Seeks in an open file
fstat()	Returns information about an open file
ftell()	Returns the current position in an open file
ftruncate()	Truncates an open file to a specified length
fwrite()	Writes to an open file
glob()	Returns an array of filenames / directories matching a specified pattern
is_dir()	Checks whether a file is a directory
is_executable()	Checks whether a file is executable
is_file()	Checks whether a file is a regular file
is_link()	Checks whether a file is a link
is_readable()	Checks whether a file is readable
is_uploaded_file()	Checks whether a file was uploaded via HTTP POST
is_writable()	Checks whether a file is writeable
lchgrp()	Changes group ownership of symlink
lchown()	Changes user ownership of symlink
link()	Creates a hard link
linkinfo()	Returns information about a hard link
lstat()	Returns information about a file or symbolic link
mkdir()	Creates a directory
move_uploaded_file()	Moves an uploaded file to a new location
pathinfo()	Returns information about a file path
pclose()	Closes a pipe opened by popen()
popen()	Opens a pipe
readfile()	Reads a file and writes it to the output buffer
readlink()	Returns the target of a symbolic link
realpath()	Returns the absolute pathname
realpath_cache_get()	Returns realpath cache entries

realpath_cache_size()	Returns realpath cache size
rename()	Renames a file or directory
rewind()	Rewinds a file pointer
rmdir()	Removes an empty directory
set_file_buffer()	Sets the buffer size of an open file
stat()	Returns information about a file
symlink()	Creates a symbolic link
touch()	Sets access and modification time of a file
umask()	Changes file permissions for files
unlink()	Deletes a file

➢ **Math functions**

Function	Description
abs()	Returns the absolute (positive) value of a number
acos()	Returns the arc cosine of a number
acosh()	Returns the inverse hyperbolic cosine of a number
asin()	Returns the arc sine of a number
asinh()	Returns the inverse hyperbolic sine of a number
atan()	Returns the arc tangent of a number in radians
atan2()	Returns the arc tangent of two variables x and y
atanh()	Returns the inverse hyperbolic tangent of a number
bindec()	Converts a binary number to a decimal number
ceil()	Rounds a number up to the nearest integer
cos()	Returns the cosine of a number
cosh()	Returns the hyperbolic cosine of a number
decbin()	Converts a decimal number to a binary number
dechex()	Converts a decimal number to a hexadecimal number
decoct()	Converts a decimal number to an octal number
deg2rad()	Converts a degree value to a radian value

exp()	Calculates the exponent of e
expm1()	Returns exp(x) – 1
floor()	Rounds a number down to the nearest integer
fmod()	Returns the remainder of x/y
getrandmax()	Returns the largest possible value returned by rand()
hexdec()	Converts a hexadecimal number to a decimal number
hypot()	Calculates the hypotenuse of a right-angle triangle
max()	Returns the highest value in an array, or the highest value of several specified values
min()	Returns the lowest value in an array, or the lowest value of several specified values
octdec()	Converts an octal number to a decimal number
pi()	Returns the value of PI
pow()	Returns x raised to the power of y
rad2deg()	Converts a radian value to a degree value
rand()	Generates a random integer
round()	Rounds a floating-point number
sin()	Returns the sine of a number
sinh()	Returns the hyperbolic sine of a number
sqrt()	Returns the square root of a number
srand()	Seeds the random number generator
tan()	Returns the tangent of a number
tanh()	Returns the hyperbolic tangent of a number

- **String functions**

Function	Description
bin2hex()	Converts a string of ASCII characters to hexadecimal values
chop()	Removes whitespace or other characters from the right end of a string
chr()	Returns a character from a specified ASCII value
chunk_split()	Splits a string into a series of smaller parts

convert_cyr_string()	Converts a string from one Cyrillic character-set to another
convert_uudecode()	Decodes a uuencoded string
convert_uuencode()	Encodes a string using the uuencode algorithm
count_chars()	Returns information about characters used in a string
crc32()	Calculates a 32-bit CRC for a string
crypt()	One-way string encryption (hashing)
echo()	Outputs one or more strings
explode()	Breaks a string into an array
fprintf()	Writes a formatted string to a specified output stream
hex2bin()	Converts a string of hexadecimal values to ASCII characters
html_entity_decode()	Converts HTML entities to characters
htmlentities()	Converts characters to HTML entities
implode()	Returns a string from the elements of an array
join()	Alias of implode()
lcfirst()	Converts the first character of a string to lowercase
levenshtein()	Returns the Levenshtein distance between two strings
localeconv()	Returns locale numeric and monetary formatting information
ltrim()	Removes whitespace or other characters from the left side of a string
number_format()	Formats a number with grouped thousands
ord()	Returns the ASCII value of the first character of a string
parse_str()	Parses a query string into variables
print()	Outputs one or more strings
printf()	Outputs a formatted string
rtrim()	Removes whitespace or other characters from the right side of a string
setlocale()	Sets locale information
sha1()	Calculates the SHA-1 hash of a string

sha1_file()	Calculates the SHA-1 hash of a file
similar_text()	Calculates the similarity between two strings
sprintf()	Writes a formatted string to a variable
sscanf()	Parses input from a string according to a format
str_ireplace()	Replaces some characters in a string (case-insensitive)
str_pad()	Pads a string to a new length
str_repeat()	Repeats a string a specified number of times
str_replace()	Replaces some characters in a string (case-sensitive)
str_shuffle()	Randomly shuffles all characters in a string
str_split()	Splits a string into an array
str_word_count()	Count the number of words in a string
strcasecmp()	Compares two strings (case-insensitive)
strchr()	Finds the first occurrence of a string inside another string (alias of strstr())
strcmp()	Compares two strings (case-sensitive)
strcoll()	Compares two strings (locale based string comparison)
strcspn()	Returns the number of characters found in a string before any part of some specified characters are found
stripos()	Returns the position of the first occurrence of a string inside another string (case-insensitive)
stristr()	Finds the first occurrence of a string inside another string (case-insensitive)
strlen()	Returns the length of a string
strncmp()	String comparison of the first n characters (case-sensitive)
strpbrk()	Searches a string for any of a set of characters
strpos()	Returns the position of the first occurrence of a string inside another string (case-sensitive)
strrchr()	Finds the last occurrence of a string inside another string
strrev()	Reverses a string

strrpos()	Finds the position of the last occurrence of a string inside another string (case-sensitive)
strspn()	Returns the number of characters found in a string that contains only characters from a specified charlist
strstr()	Finds the first occurrence of a string inside another string (case-sensitive)
strtok()	Splits a string into smaller strings
strtolower()	Converts a string to lowercase letters
strtoupper()	Converts a string to uppercase letters
strtr()	Translates certain characters in a string
substr()	Returns a part of a string
substr_compare()	Compares two strings from a specified start position (binary safe and optionally case-sensitive)
substr_count()	Counts the number of times a substring occurs in a string
substr_replace()	Replaces a part of a string with another string
trim()	Removes whitespace or other characters from both sides of a string
vfprintf()	Writes a formatted string to a specified output stream
vprintf()	Outputs a formatted string
vsprintf()	Writes a formatted string to a variable
wordwrap()	Wraps a string to a given number of characters

- **Miscellaneous Functions**

Function	Description
connection_aborted()	Checks whether the client has disconnected
connection_status()	Returns the current connection status
connection_timeout()	Deprecated in PHP 4.0.5. Checks whether the script has timed out
constant()	Returns the value of a constant

define()	Defines a constant
defined()	Checks whether a constant exists
die()	Prints a message and exits the current script
eval()	Evaluates a string as PHP code
exit()	Prints a message and exits the current script
get_browser()	Returns the capabilities of the user's browser
halt_compiler()	Halts the compiler execution
pack()	Packs data into a binary string
php_check_syntax()	Deprecated in PHP 5.0.5
php_strip_whitespace()	Returns the source code of a file with PHP comments and whitespace removed
sleep()	Delays code execution for a number of seconds
sys_getloadavg()	Gets system load average
time_nanosleep()	Delays code execution for a number of seconds and nanoseconds
time_sleep_until()	Delays code execution until a specified time
uniqid()	Generates a unique ID
unpack()	Unpacks data from a binary string
usleep()	Delays code execution for a number of microseconds

4.3 DATABASE CONNECTIVITY

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

Opening Database Connection:

PHP provides mysql_connect function to open a database connection. This function takes five parameters and returns a MySQL link identifier on success, or FALSE on failure.

connection mysql_connect(server,user,password,new_link,client_flag);

- **Server:** The host name running database server. If not specified then default value is localhost:3306. This is optional.
- **User:** The username accessing the database. If not specified then default is the name of the user that owns the server process. This is optional.
- **Password:** The password of the user accessing the database. If not specified then default is an empty password.
- **new_link:** If a second call is made to mysql_connect() with the same arguments, no new connection will be established; instead, the identifier of the already opened connection will be returned. This is optional.
- **client_flags:** This is optional. A combination of the following constants:
  1. MYSQL_CLIENT_SSL - Use SSL encryption
  2. MYSQL_CLIENT_COMPRESS - Use compression protocol
  3. MYSQL_CLIENT_IGNORE_SPACE - Allow space after function names
  4. MYSQL_CLIENT_INTERACTIVE - Allow interactive timeout seconds of inactivity before closing the connection

➢ **Closing Database Connection:**

PHP uses mysql_close to close a database connection. This function takes connection resource returned by mysql_connect function. It returns TRUE on success or FALSE on failure. If a resource is not specified then last opened database is closed.

bool mysql_close (resource $link_identifier);

➢ **Creating a Database:**

To create and delete a database the users should have admin privilege. PHP uses mysql_query function to create a MySQL database. This function takes two parameters and returns TRUE on success or FALSE on failure.

bool mysql_query(sql, connection);

- **Sql:** SQL query to create a database
- **Connection:** if not specified then last opened connection by mysql_connect will be used.

➢ **Selecting a Database:**

Once the user establishes a connection with a database server then it is required to select a particular database where all the tables are associated. This is required because there may be multiple databases residing on a single server. PHP provides function mysql_select_db to select a database. It returns TRUE on success or FALSE on failure.

bool mysql_select_db(db_name, connection);

- **db_name:** Database name to be selected
- **connection**: if not specified then last opened connection by mysql_connect will be used.

➢ **Creating Database Tables:**

To create tables in the new database the user need to do the same thing as creating the database. First create the SQL query to create the tables then execute the query using mysql_query() function.

Creating and selecting a database table

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass); //Creating a connection
if(! $conn )
{ die('Could not connect: ' . mysql_error()); }
echo 'Connected successfully';
$sql = 'CREATE TABLE employee( '. 'emp_id INT NOT NULL AUTO_INCREMENT, '.
      'emp_name VARCHAR(20) NOT NULL, '. 'emp_address VARCHAR(20) NOT NULL,
'.'emp_salary   INT NOT NULL,  '. 'join_date     timestamp(14) NOT NULL,  '.
      'primary key ( emp_id ))'; //Creating a table
mysql_select_db('test_db'); //Seeting a table
$retval = mysql_query( $sql, $conn );
if(! $retval )
```

```
{ die('Could not create table: ' . mysql_error()); }
echo "Table employee created successfully\n";
mysql_close($conn); //Closing a connection ?>
```

In case the user need to create many tables then it is better to create a text file first and put all the SQL commands in that text file and then load that file into $sql variable and execute those commands.

- **Deleting a Database:**

  If a database is no longer required then it can be deleted forever. The users can use pass an SQL command to mysql_query to delete a database.

- **Deleting a Table:**

  It is again a matter of issuing one SQL command through mysql_query function to delete any database table. But be very careful while using this command because by doing so the users can delete some important information the user has in the table.

Deleting a table

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{ die('Could not connect: ' . mysql_error()); }
$sql = 'DROP TABLE employee'; //Deleting a table
$retval = mysql_query( $sql, $conn );
if(! $retval )
{ die('Could not delete table employee: ' . mysql_error()); }
echo "Table deleted successfully\n";
mysql_close($conn); ?>
```

- **Insert Data into MySQL Database**

  Data can be entered into MySQL tables by executing SQL INSERT statement through PHP function mysql_query. In real application, all the values will be taken using HTML form and then those values will be captured using PHP script and finally they will be inserted into MySQL tables.

Inserting values

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{ die('Could not connect: ' . mysql_error()); }
$sql = 'INSERT INTO employee '. '(emp_name,emp_address, emp_salary, join_date) '.
    'VALUES ( "guest", "XYZ", 2000, NOW() )';
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{ die('Could not enter data: ' . mysql_error()); }
echo "Entered data successfully\n";
mysql_close($conn); ?>
```

- **Getting Data From MySQL Database**

  Data can be fetched from MySQL tables by executing SQL SELECT statement through PHP function mysql_query. The user have several options to fetch data from MySQL. The most frequently used option is to use function mysql_fetch_array(). This function returns row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows.

Fetching data from tables

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
```

```
if(! $conn )
{ die('Could not connect: ' . mysql_error()); }
$sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{ die('Could not get data: ' . mysql_error()); }
while($row = mysql_fetch_array($retval,  MYSQL_ASSOC))
{     echo "EMP ID :{$row['emp_id']} <br> ".
        "EMP NAME : {$row['emp_name']} <br> ".
        "EMP SALARY : {$row['emp_salary']} <br> "; }
echo "Fetched data successfully\n";
mysql_close($conn); ?>
```

- **Deleting Data from MySQL Database**

  Data can be deleted from MySQL tables by executing SQL DELETE statement through PHP function mysql_query. To delete a record in any table it is required to locate that record by using a conditional clause.

Deleting data from tables

```
<html> <head> <title>Delete a Record from MySQL Database</title> </head>
<body> <?php
if(isset($_POST['delete']))
{ $dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{ die('Could not connect: ' . mysql_error()); }
$emp_id = $_POST['emp_id'];
$sql = "DELETE employee WHERE emp_id = $emp_id" ; //Query to delete a record
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{ die('Could not delete data: ' . mysql_error()); }
echo "Deleted data successfully\n";
mysql_close($conn); }
else
{ ?>
<form method="post" action="<?php $_PHP_SELF ?>">
<table width="400" border="0" cellspacing="1" cellpadding="2"> <tr>
<td width="100">Employee ID</td>
<td><input name="emp_id" type="text" id="emp_id"></td> </tr>
<tr> <td width="100"></td>
<td></td> </tr>
<tr> <td width="100"></td>
<td> <input name="delete" type="submit" id="delete" value="Delete"> </td> </tr>
</table> </form>
<?php } ?> </body></html>
```

- **Updating Data into MySQL Database**

  Data can be updated into MySQL tables by executing SQL UPDATE statement through PHP function mysql_query. To update a record in any table it is required to locate that record by using a conditional clause.

Updating data

```
<html><head> <title>Update a Record in MySQL Database</title> </head><body>
<?php
if(isset($_POST['update']))
{ $dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{ die('Could not connect: ' . mysql_error()); }
$emp_id = $_POST['emp_id'];
$emp_salary = $_POST['emp_salary'];
$sql = "UPDATE employee SET emp_salary = $emp_salary WHERE emp_id = $emp_id"
;
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{ die('Could not update data: ' . mysql_error()); }
echo "Updated data successfully\n";
mysql_close($conn); }
else {?>
<form method="post" action="<?php $_PHP_SELF ?>">
<table width="400" border="0" cellspacing="1" cellpadding="2">
<tr> <td width="100">Employee ID</td>
<td><input name="emp_id" type="text" id="emp_id"></td> </tr>
<tr> <td width="100">Employee Salary</td>
<td><input name="emp_salary" type="text" id="emp_salary"></td> </tr>
<tr> <td width="100"></td> <td></td> </tr>
<tr> <td width="100"></td> <td>
<input name="update" type="submit" id="update" value="Update"> </td> </tr>
</table> </form> <?php }
?> </body></html>
```

4.6 NEWSFEEDS and ATOM

NEWS FEED

News feeds are an example of automated syndication. News feed technologies allow information to be automatically provided and updated on Web sites, emailed to users, etc. As the name implies news feeds are normally used to provide news; however the technology can be used to syndicate a wide range of information**.**

The **BBC ticker** is an example of a news feed application. A major limitation with this approach is that the ticker can only be used with information provided by the BBC. The RSS (Really Simple Syndication)standard was developed as an open standard for news syndication, allowing applications to display news supplied by any RSS provider.

RSS (Really Simple Syndication)

RSS is an XML dialect used to publish frequently updated content, such as blog posts or news headlines.

It is a way to easily distribute a list of headlines, update notices, and sometimes content to a wide number of people. It is used by computer programs that organize those headlines and notices for easy reading.

The content feed is identified by a unique URI, and this URI is used by an RSS Reader application to retrieve and display the content feed from a web site. An RSS feed can contain one or more images or items. An item can be a synopsis of an article with a link to the full article or the entire article itself. An RSS has a well-defined structure. Some Web APIs use RSS as the data format returned when a request is made.

Working of RSS

RSS works by having the website author maintain a list of notifications on their website in a standard way. This list of notifications is called an **RSS Feed**. People who are interested in finding out the latest headlines or changes can check this list. Special computer programs called **RSS aggregators** have been developed that automatically access the RSS feeds of websites of user’s interest and organize the results.

Producing an RSS feed is very simple and hundreds of thousands of websites now provide this feature, including major news organizations like the New York Times, the BBC, and Reuters, as well as many weblogs.

RSS provides very basic information to do its notification. It is made up of a list of items presented in order from newest to oldest. Each item usually consists of a simple title describing the item along with a more complete description and a link to a web page with the actual information being described. Sometimes this description is the full information the user want to read (such as the content of a weblog post) and sometimes it is just a summary.

Creating RSS feed

The special XML-format file that makes up an RSS feed is usually created in one of a variety of ways. Most large news websites and most weblogs are maintained using special **content management** programs. Authors add their stories and postings to the website by interacting with those programs and then use the program's publish facility to create the HTML files that make up the website. Those programs often also can update the RSS feed XML file at the same time, adding an item referring to the new story or post, and removing less recent items.

Blog creation tools like Blogger, LiveJournal, Movable Type, and Radio automatically create feeds. Websites that are produced in a more custom manner, such as with Macromedia Dreamweaver or a simple text editor, usually do not automatically create RSS feeds. Authors of such websites either maintain the XML files by hand, just as they do the website itself, or use a tool such as Software Garden, Inc.'s ListGarden program to maintain it.

There are also services that periodically read requested websites themselves and try to automatically determine changes (this is most reliable for websites with a somewhat regular news-like format), or that let the users create RSS feed XML files that are hosted by that service provider.

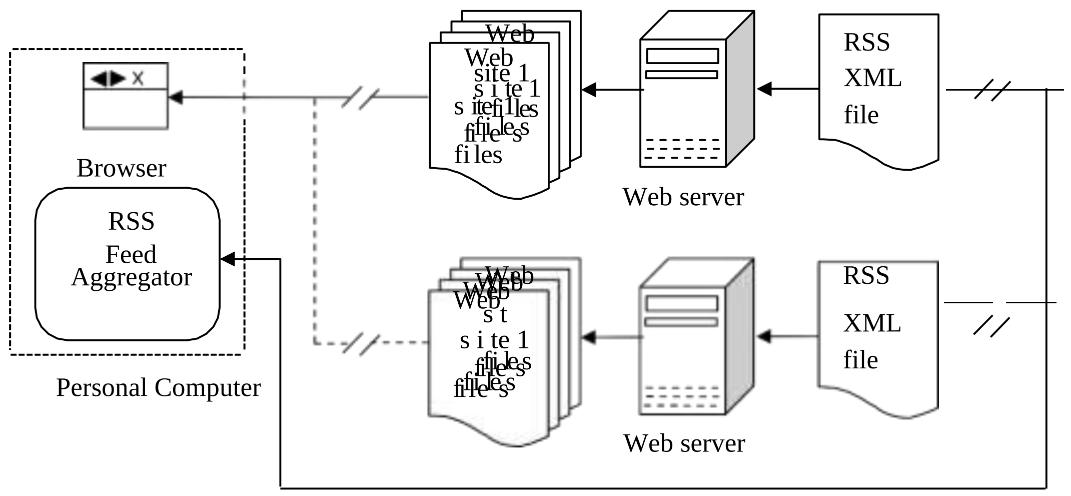

Fig 4.3: Communication between websites, RSS feed and PC

In the above diagram, a web browser being used to read first Web Site 1 over the Internet and then Web Site 2. It also shows the RSS feed XML files for both websites being monitored simultaneously by an RSS Feed Aggregator.

- **Sections of an RSS file**

Apart from the root element there are four main sections of the RSS file. These are the channel, image, item, and text input sections. In practical use, the channel and item elements are requirements for any useful RSS file, while the image and text input are optional.

The channel section

The channel element contains metadata that describe the channel itself, telling what the channel is and who created it. The channel is a required element that includes the name of the channel, its description, its language, and a URL. The URL is normally used to point to the channel's source of information.

Channel element

```
<channel><title>MozillaZine</title>

<link>http://www.mozillazine.org</link>

<description>The user source for Mozilla news, advocacy, interviews, builds, and more!
</description>

<language>en-us</language> </channel>
```

The title can be treated as a headline link with the description following. The **Channel Language definition** allow aggregators to filter news feeds and gives the rendering software the information necessary to display the language properly. The </channnel> tag is used after all the channel elements to close the channel. As RSS conforms to XML specs, the element must be well formed; it requires the closing tag.

The image section

The image element is an optional element that is usually used to include the logo of the channel provider. The default size for the image is 88 pixels wide by 31 pixels high, but it can be enlarged to 144 pixels wide by 400 pixels wide.

Image element

```
<image><title>MozillaZine</title>

<url>http://www.mozillazine.org/image/mynetscape88.gif</url>

<link>http://www.mozillazine.org</link>

<width>88</width>

<height>31</height> </image>
```

The image's title, URL, link, width, and height tags allow renderers to translate the file into HTML. The title tag is normally used for the image's ALT text.

The items

Items form the dynamic part of an RSS file. While channel, image, and text input elements create the channel's identity and typically stay the same over long periods of time, channel items are rendered as news headlines, and the channel's value depends on their changing fairly frequently.

Item element

```
<item><title>Java2 in Navigator 5?</title>
<link>http://www.mozillazine.org/talkback.html?article=607</link>
<description>Will Java2 be an integrated part of Navigator 5?
    Read more about it in this discussion...</description> </item>
```

Fifteen items are allowed in a channel. Titles should be less than 100 characters, while descriptions should be under 500 characters. The item title is normally rendered as a headline that links to the full article whose URL is provided by the item link. The item description is commonly used for either a summary of the article's content or for commentary on the article.

News feed channels use the description to highlight the content of news articles, usually on the channel owner's site, and Web log channels use the description to provide commentary on a variety of content, often on third-party sites.

The text input

The text input area is an optional element, with only one allowed per channel. This lets the user respond to the channel.

```
<textinput><title>Send</title>
<description>Comments about MozillaZine?</description>
<name>responseText</name>
<link>http://www.mozillazine.org/cgi-bin/sampleonly.cgi</link>    </textinput>
```

The title is normally rendered as the label of the form's ssubmit button, and the description as the text displayed before or above the input field. The text input name is supplied along with the contents of the text field when the submit button is clicked.

ATOM

Atom is a syndication data format like RSS, as well as a publishing protocol. The Atom data format uses XML syntax with one or more entry elements containing the full and/or summary content.

The **Atom Publishing Protocol** (APP) defines a hierarchy for organizing published content (services, workspaces, collections, and resources) and uses the HTTP Get, Post, Put, and Delete methods for retrieving, creating, deleting, and editing published content. Atom's use of HTTP's built-in methods and XML as a data format is in the spirit of a RESTful web services implementation.

Atom was designed to be a universal publishing standard for blogs and other Web sites where content is updated frequently. Users visiting a Web site with an Atom feed can discover a file described as "atom.xml" in the URL that can be copied and pasted into an aggregator to subscribe to the feed.

Atom was originally developed as an alternative to RSS 2.0, the standard developed by Dave Winer and copyrighted by Harvard University, as a means of improving perceived shortcomings of the RSS format by the blogging community.

Features of ATOM

- Atom was developed to be vendor neutral and freely extensible by any user; it is an open standard.
- Atom lies within an XML-namespace.
- Atom includes auto discovery, allowing feed aggregators to automatically detect the presence of a feed.
- Atom differentiates between relative and non-relative URIs.
- Atom has separate summary and content elements.
- Atom explicitly labels a payload as plaintext or HTML.
- Each entry has a globally unique ID.

4.2 PHP COOKIES and REGULAR EXPRESSIONS IN PHP

***Cookies are small data stored on the client computer and they are kept of use tracking purpose*.**

PHP COOKIES

PHP transparently supports HTTP cookies. There are three steps involved in identifying returning users:

- Server script sends a set of cookies to the browser
- 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.

➢ **Setting Cookies with PHP:**

PHP provided **setcookie()** function to set a cookie. This function requires 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);
```

- **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 the user wants 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 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.

Setting a cookie

```
<?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.isset() function is used to check if a cookie is set or not.

Accessing cookies

```
<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["name"] . "<br />";
?> </body></html>
```

- **Deleting Cookie**

  To delete a cookie users 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>
```

REGULAR EXPRESSIONS IN PHP

Regular expressions are nothing more than a sequence or pattern of characters. They provide the foundation for pattern-matching functionality.

PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression: POSIX Regular Expressions and PERL Style Regular Expressions.

POSIX Regular Expressions

- The structure of a POSIX regular expression is similar to 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.
    - **Brackets:** Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.
    - **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.
    - **Predefined Character Ranges**: For 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.

Expression	Description
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing zero or more p's. This is just an alternative way to use p*.
p{N}	It matches any string containing a sequence of N p's
p{2,3}	It matches any string containing a sequence of two or three p's.

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

Function	Description
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.
ereg_replace()	The ereg_replace() function searches for string specified by pattern and replaces pattern with replacement if found.
eregi()	The eregi() function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive.
eregi_replace()	The eregi_replace() function operates exactly like ereg_replace(), except that the search for pattern in string is not case sensitive.
split()	The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
spliti()	The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive.
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

- **Metacharacters:** A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

Metacharacters	Description
.	a single character
\s	a whitespace character (space, tab, newline)
\S	non-whitespace character
\d	a digit (0-9)
\D	a non-digit

\w	a word character (a-z, A-Z, 0-9, _)
\W	a non-word character
[aeiou]	matches a single character in the given set
[^aeiou]	matches a single character outside the given set
(foo|bar|baz)	matches any of the alternatives specified

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

Modifiers	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 string boundary
O	Evaluates the expression only once
S	Allows use of . to match a newline character
X	Allows the user 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

Functions	Description
preg_match()	The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise.
preg_match_all()	The preg_match_all() function matches all occurrences of pattern in string.
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.
preg_split()	The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.
preg_grep()	The preg_grep() function searches all elements of input_array, returning all elements matching the regexp pattern.
preg_ quote()	Quote regular expression characters

4.5 DOCUMENT TYPE DECLARATION (DTD) and XML SCHEMAS

The document type declaration attaches a DTD to a document. It is a way to describe XML language precisely.

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

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 **InternalSubset.**

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.

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

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

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>Sharanya</name>
<company>abc</company>
<phone>12347890</phone> </address>
```

Start Declaration- Begin the XML declaration with 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 elements, attributes, entities, and notations are declared.

End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>).

Rules

- The document type declaration must appear at the start of the document.
- 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.

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.

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

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<!DOCTYPE address SYSTEM "address.dtd">

<address> <name>Sharanya</name>

<company>abc</company>

<phone>1234689</phone> </address>
```

address.dtd

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

<!ELEMENT company (#PCDATA)> <!ELEMENT phone (#PCDATA)>
```

Types of external DTD

- **System Identifiers:** A system identifier enables the user to specify the location of an external file containing DTD declarations.

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

- **Public Identifiers:** Public identifiers provide a mechanism to locate DTD resources.

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

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**.

Advantages of DTD

- The XML processor enforces the structure, as defined in the DTD.
- The application easily accesses the document structure.
- The DTD gives hints to the XML processor—that is, it helps separate indenting from content.
- The DTD can declare default or fixed values for attributes. This might result in a smaller document.

XML SCHEMAS

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.

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

```
<?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>
```

- **Elements:** An element can be defined within an XSD as follows:

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

- **Definition Types**

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

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

  - **Complex Type** - A complex type is a container for other element definitions. This allows the user to specify which child elements an element can contain and to provide some structure within the user's XML documents.

```
          <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>
```

  - **Global Types** - With global type, the user can define a single type in the user's document, which can be used by all other references.

```
 <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 let us use this type in our example as below:

```
<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.

- **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"/>
```

4.4 XML (Extensible Markup Language)

XML is a text-based markup language derived from Standard Generalized Markup Language (SGML).It is a new markup language, developed by the W3C (World Wide Web Consortium), mainly to overcome the limitations in HTML. **Markup** is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable.

they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

XML is not a programming language. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML. XML don't have pre - defined tags and the tags are stricter than HTML.

XML is extensible: XML allows the user to create user defined self-descriptive tags, or language that suits the application.

XML carries the data, does not present it: XML allows storing the data irrespective of how it will be presented.

XML is a public standard: XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

Features of XML

- XML simplifies the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize the user data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Any type of data can be expressed as an XML document
- It has syndicated content, where content is being made available to different web sites.
- It suits well for electronic commerce applications where different organizations collaborate to serve a customer.
- It supports scientific applications with new markup languages for mathematical and chemical formulas.
- It also supports electronic books with new markup languages to express rights and ownership.
- XML could also be used in handheld devices and smart phones with new markup languages optimized for these devices.

Syntax Rules

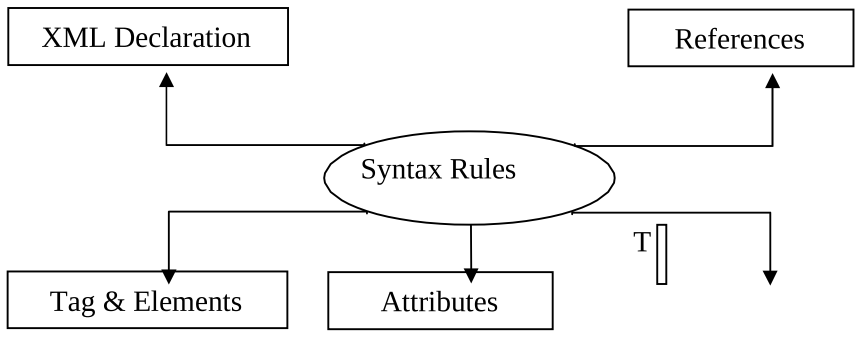

Fig 4.1 Syntax Rules of XML

The XML document can optionally have an XML declaration.

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

Version is the XML version and encoding specifies the character encoding used in the document.

Syntax Rules for XML declaration

- The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
- If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of encoding that the user put in the XML declaration.

➢ **Tags and Elements**

An XML file is structured by several XML-elements also called **XML-nodes** or XML-tags. XML-elements' names are enclosed by angular brackets <>.

<element>... </element> or

<element/>

Nesting of elements

An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

```
<?xml version="1.0"?>
<contact-info>
<company>abc</company>
<contact-info>
```

- **Root element:** An XML document can have only one root element.
- **Case sensitivity:** The names of XML-elements are case-sensitive.

➢ **Attributes**

An attribute specifies a single property for the element, using a name/value pair. An XML-element can have one or more attributes. It is possible to attach additional information to elements in the form of attributes. The names follow the same rules as element names.

<tel preferred="true">513-555-8889</tel>

➢ XML References

References allow the user to add or include additional text or markup in an XML document. References always begin with the symbol "&", which is a reserved character and end with the symbol ";". XML has two types of references:

- **Entity References:** An entity reference contains a name between the start and the end delimiters. The entity reference is replaced by the content of the entity.
- **Character References:** A letter is replaced by its Unicode character code. Character references that start with &#x provides a hexadecimal representation of the character code.

XML Text

The names of XML-elements and XML-attributes are case-sensitive, which means the name of start and end elements need to be written in the same case. To avoid character encoding problems, all XML files should be saved as Unicode UTF-8 or UTF-16 files. Whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attributes will be ignored. Some characters are reserved by the XML syntax itself. Hence, they cannot be used directly.

XML Documents

An XML document is a basic unit of XML information composed of elements and other markup in an orderly package. An XML document can contains wide variety of data.

```
<?xml version="1.0"?> //document prolog
<contact-info> //all the following lines are document element
<name>Sharanya</name>
<company>abc</company>
<phone>(044)257887296</phone>
</contact-info>
```

The XML documents are divided into: Document prolog section and document element sections.

Document prolog: The document prolog comes at the top of the document, before the root element. This section contains: XML declaration and Document type declaration. The first line in the above example belongs to prolog section.

Document element: Document Elements are the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose. The other lines except the first line come under document element section.

XML Declaration

The XML declaration is the first line of the document. The declaration identifies the document as an XML document. The declaration also lists the version of XML used in the document. The declaration can contain other attributes to support other features such as character set encoding.

Syntax:

```
<?xml
  version="version_number"
  encoding="encoding_declaration"
  standalone="standalone_status"
?>
```

- ❖ **Version-** Specifies the version of the XML standard used.
- ❖ **Encoding declaration-** It defines the character encoding used in the document. UTF-8 is the default encoding used.
- ❖ **Standalone-** It informs the parser whether the document relies on the information from an external source, such as external document type definition (DTD), for its content. The default value is set to no. Setting it to yes tells the processor there are no external declarations required for parsing the document.

Rules for XML declaration

- If the XML declaration is present in the XML, it must be placed as the first line in the XML document.
- If the XML declaration is included, it must contain version number attribute.
- The Parameter names and values are case-sensitive.
- The names are always in lower case.
- The order of placing the parameters is important. The correct order is: version, encoding and standalone.
- Either single or double quotes may be used.
- The XML declaration has no closing tag i.e. </?xml>

Example	Description
<?xml >	XML declaration with no parameters.
<?xml version="1.0">	XML declaration with version definition.

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>	XML declaration with all parameters defined.
<?xml version='1.0' encoding='iso-8859-1' standalone='no' ?>	XML declaration with all parameters defined in single quotes.

XML tags

XML tags form the foundation of XML. They define the scope of an element in the XML. They can also be used to insert comments, declare settings required for parsing the environment and to insert special instructions.

- **Start Tag:** The beginning of every non-empty XML element is marked by a start-tag.

  Example: <address>.

- **End Tag:**Every element that has a start tag should end with an end-tag.

  Example: </address>

- **Empty Tag:** The text that appears between start-tag and end-tag is called content. An element which has no content is termed as empty. An empty element can be represented in two ways:

  (1) A start-tag immediately followed by an end-tag:<hr></hr>

  (2) A complete empty-element tag : <hr />

XML Tags Rules

- XML tags are case-sensitive.
- XML tags must be closed in an appropriate order, i.e., an XML tag opened inside another element must be closed before the outer element is closed.

XML Elements

XML elements can be defined as building blocks of an XML. Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

```
<element-name attribute1 attribute2>
....content
</element-name>
```

- **element-name** is the name of the element.

- attribute1, attribute2 are **attributes** of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as: name = "value".

Empty Element: An empty element is an element with no content.

Example: <name attribute1 attribute2.../>

XML Elements Rules

- An element name can contain any alphanumeric characters. The only punctuation mark allowed in names are the hyphen (-), under-score (_) and period (.).
- Names are case sensitive.
- Start and end tags of an element must be identical.
- An element, which is a container, can contain text or elements.

XML Attributes

Attributes are part of the XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements. To be more precise, they define properties of elements. An XML attribute is always a name-value pair.

```
<element-name attribute1 attribute2 >
        ....content….
< /element-name>
```

where attribute1 and attribute2 has the following form: name = "value"

The following are the types of attributes:

➢ **String:** It takes any literal string as a value. CDATA is a String type. CDATA is character data. This means, any string of non-markup characters is a legal part of the attribute.

➢ **Tokenized:** This is more constrained type. The validity constraints noted in the grammar are applied after the attribute value is normalized.

➢ **Enumerated**: This has a list of predefined values in its declaration, out of which, it must assign one value. There are two types of enumerated attribute:

❖ **NotationType:** It declares that an element will be referenced to a NOTATION declared somewhere else in the XML document.

❖ **Enumeration:** Enumeration allows the user to define a specific list of values that the attribute value must match.

Element Attribute Rules

An attribute name must not appear more than once in the same start-tag or empty-element tag. An attribute must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration. Attribute values must not contain direct or indirect entity references to external entities. The replacement text of any entity referred to directly or indirectly in an attribute value must not contain either less than sign <.

XML other features

- **Comments:**

  A comment starts with <!-- and ends with -->. Comments cannot appear before XML declaration. Comments can appear anywhere in a document. Comments must not appear within attribute values. Nested comments are not allowed.

- **Whitespaces:**

  Whitespace is a collection of spaces, tabs, and newlines. XML document contain two types of white spaces Significant Whitespace and Insignificant Whitespace. A **significant Whitespace** occurs within the element which contain text and markup present together.

```
<name>Adhithya Ramanan</name>
```

Insignificant whitespace means the space where only element content is allowed.

```
<address.category="residence">
```

Differences between XML and HTML

XML	HTML
XML was designed to be a software and hardware independent tool used to transport and store data, with focus on what data is.	HTML was designed to display data with focus on how data looks.
XML provides a framework for defining markup languages.	HTML is a markup language itself.
XML is neither a programming language nor a presentation language.	HTML is a presentation language.
XML is case sensitive.	HTML is case insensitive.
XML is used basically to transport data between the application and the database.	HTML is used for designing a web-page to be rendered on the client side.
In XML custom tags can be defined and the tags are invented by the author of the XML document.	HTML has its own predefined tags

XML makes it mandatory for the user the close each tag that has been used.	HTML is not strict if the user does not use the closing tags.
XML preserve white space.	HTML does not preserve white space.
XML is about carrying information, hence it is dynamic.	HTML is about displaying data, hence it is static.

4.5 .XML DOM and XML PARSERS

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.

DOM

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.

```
<!DOCTYPE html> <html><body>
<h1> 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>
```

address.xml

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

XML PARSERS

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 XMLdocuments.

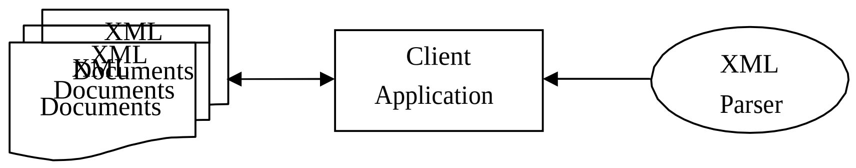

Fig 4.2 XML Parsers

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.

VALIDATION

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 and Valid XML document

➢ **Well-formed XML document**

- 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.

```
<?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>Sharanya</name>
<company>abc</company>
<phone>(011) 123-4567</phone> </address>
```

➢ **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.

XSL (XML Style Sheet)

XML concentrates on the structure of the information and not its appearance. The W3C has published two recommendations for style sheets: CSS (Cascading Style Sheet) and XSL(XML Style sheet Language).XSL supports transforming the document before display. XSL would typically be used for advanced styling. XSL originally consisted of three parts:

- XSLT (XSL Transformatio n) - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO (XSL Formatting Objects) - a language for formatting XML documents

XSL

```
<P><B>Table of Contents</B></P> <UL>
<xsl:for-each select=”article/section/title”>
<LI><A><xsl:value-of select=”.”/></A></LI>
</xsl:for-each> </UL>
```

XSLT (XSL Transformation)

XSLT is a language to specify transformation of XML documents. It takes an XML document and transforms it into another XML document. XSLT is an XML-related technology that is used to manipulate and transform XML documents.

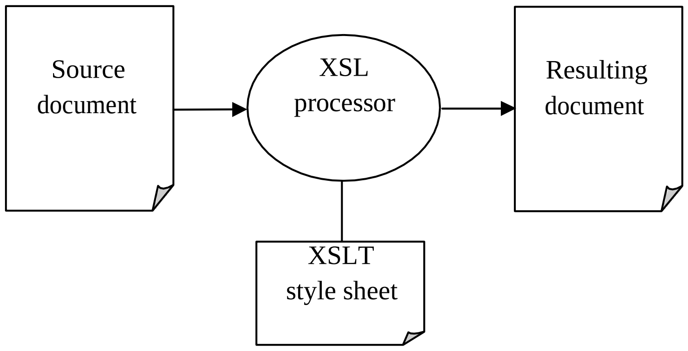

Fig 4.2 XSLT Transformation

With XSLT, the user can take an XML document and choose the elements and values, then generate a new file with new choices. Because of XSLT's ability to change the content of an XML document, XSLT is referred to as the **stylesheet for XML**. XSLT is not limited to styling activities. Many applications require transforming documents. XSLT can be used to:

- Add elements specifically for viewing, such as add the logo or the address of the sender to an XML invoice.

- Create new content from an existing one, such as create the table of contents
- Present information with the right level of details for the reader, such as using a style sheet to present high-level information to a managerial person while using another style sheet to present more detailed technical information to the rest of the staff.
- Convert between different DTDs or different versions of a DTD, such as convert a company specific DTD to an industry standard
- Transform XML documents into HTML for backward compatibility with existing browsers.

XSLT

XML code	XSLT code

`<?xml version="1.0" encoding="UTF-8"?>`
`<?xml-stylesheet type="text/xsl" href="class.xsl"?>`
`<class>`
`<student>Arthi</student>`
`<student>Ambarish</student>`
`<student>Anitha</student>`
`<teacher>Sharanya</teacher>`
`</class>`
	`<?xml version="1.0" ?>`
	`<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >`
	`<xsl:template match="teacher">`
	`<p><u><xsl:value-of select="."/></u></p>`
	`</xsl:template>`
	`<xsl:template match="student">`
	`<p><b><xsl:value-of select="."/></b></p>`
	`</xsl:template>`
	`<xsl:template match="/">`
	`<html><body>`
	`<xsl:apply-templates/>`
	`</body></html>`
	`</xsl:template></xsl:stylesheet>`

The XML file class.xml is linked to the XSLT code by adding the xml-stylesheet reference. The XSLT code then applies its rules to transform the XML document.

- Before XSLT: classoriginal.xml
- After XSLT rules are applied: class.xml

XSLT Syntax

➢ **XSLT - XML Declaration**

The user includes an XML declaration at the top of the XSLT documents. The attribute version defines what version of XML is used.

Example: <?xml version="1.0" ?>

- **XSLT - Stylesheet Root Element**

  Every XSLT file must have the root element xsl:stylesheet. This root element has two attributes that must be included:

  - version - the version of XSLT
  - xmlns:xsl - the XSLT namespace, which is a URI to w3.org

- **XSLT - XSL: Namespace Prefix**

  The root element specifies the XSL namespace. The standard form of an XSL element is: xsl:element

XSLT - Stylesheet Reference

Linking XML document to XSLT stylesheet is stylesheet reference. This is the magic step that connects XML to a XSLT file

XSLT - xml-stylesheet

xml-stylesheet is a special declaration in XML for linking XML with stylesheets. Place this after XML declaration to link the XML file to the XSLT code. xml-stylesheet has two attributes:

- type: the type of file being linked to. We will be using the value text/xsl to specify XSLT.
- href: the location of the file. If the user saved the user XSLT and XML file in the same directory, the user can simply use the XSLT filename.

Make sure that both XSLT and XML file are in the same directory.

Reference

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="class.xsl"?>
<class>         <student>Arun</student>
          <student>Divya</student>
          <teacher>Sharanya</teacher> </class>
```

XSLT: XSL Template

The purpose of XSLT is to help transform an XML document into something new. To transform an XML document, XSLT must be able to do two things well:

- Find information in the XML document.
- Add additional text and/or data.

Both of these items are taken care of with the very important XSL element xsl:template.

XSLT - xsl:template Match Attribute

To find information in an XML document use xsl:template's match attribute. It is in this attribute the knowledge of XPath is used to find information in the XML document. In previous example, to find student elements, we would set the match attribute to a simple XPath expression: student.

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="student">
                Found a learner!
        </xsl:template>
<stylesheet>
```

XSLT - xsl:apply-templates

The xsl:apply-templates element to be more selective of the XML data.

➢ **XSLT - Remove Unwanted Text**

The following attributes are used to remove unwanted text:

- select attribute: lets the user choose specific child elements
- xsl:apply-templates: to decide when and where the xsl:template elements are used

XSLT - Remove Unwanted Children

We could use the select attribute to select specific child elements. To do this, we need a new xsl:template that matches our XML document's root element, class. We can then pick the child student using the select attribute. Here's the XSLT code to get the job done.

apply templates

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="class">
<xsl:apply-templates select="student"/>
</xsl:template>
<xsl:template match="student">
```

```
   Found a learner!
</xsl:template></xsl:stylesheet>
```

```
Found a learner! Found a learner! Found a  learner!
```

The XSLT processor begins at the root element when looking for template matches. Because we have a match for the root element, class, the code we just added is used first.

xsl:apply-templates

In our template that matched class, we use xsl:apply-templates which will check for template matches on all the children of class. The children of class in our XML document are student and teacher.

xsl:apply-templates select="student"

To have the teacher element, "Sharanya," ignored, we use the select attribute of xsl:apply-templates to specify only student children.

The XSLT processor then goes searching templates that only match student elements.

xsl:template match="student"

The processor finds the only other template in our XSLT, which prints out, "Found a learner!" for each student element in the XML document. XSLT finds three students, so "Found a learner!" is displayed three times.

XSLT - Well-Formed Output

To obtain well formed output remove the root element in an XSLT template and inserting a new root element for the output. To do this, we are going to need to add an <html> (root element) tag, a <body> tag, and maybe some <p> tags.

XSLT - Replacing the Old Root Element

In the template that matches the original root element, we will insert the <html> tag to be the output's root element. We can also put the <body> tag there. In the template that matches the student elements, we can insert a <p> tag to make a separate paragraph for each student.

Replacing root

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="class"> <html><body>
<xsl:apply-templates select="student"/> </body></html> </xsl:template>
```

```
<xsl:template match="student">
<p>    Found a learner!</p>  </xsl:template></xsl:stylesheet>
```

```
<html><body>
<p>Found a  learner!</p>
<p>Found  a learner!</p>
<p>Found  a learner!</p>
</body></html>
```