UNIT IV

OPEN SOURCE PROGRAMMING LANGUAGES

PHP: Introduction – Programming in web environment – variables – constants – Data types – operators – Statements – Functions – Arrays – OOP – String Manipulation and regular expression.

PHP: Introduction, Programming in web environment:

PHP is a server-side scripting language. PHP stands for Hypertext Preprocessor. This means that the script is run on your web server, not on the user's browser, so you do not need to worry about compatibility issues. PHP is relatively new (compared to languages such as Perl (CGI) and Java) but is quickly becoming one of the most popular scripting languages on the internet. PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.) PHP is an open source software PHP is free to download and use.

PHP files can contain text, HTML tags and scripts.PHP files are returned to the browser as plain HTML.PHP files have a file extension of ".php", ".php3", or ".phtml".

PHP enables you to build large, complex, and dynamic websites. PHP can also increase your productivity enormously, both in development time and maintenance time.

Using PHP, you can build websites that do things such as:

- Query a database
- Allow users to upload files
- Create/read files on the server (for example, the files that your users upload)
- Have a "member's area" (i.e. via a login page)
- Have a shopping cart
- Present a customized experience (for example, based on users' browsing history)
- Much, much more

Creating a PHP File:

To create a PHP file, simply do the following:

- Create a new file in your favorite editor
- Type some PHP code
- Save the file with a .php extension

The .php extension tells the web server that it needs to process this as a PHP file. If you accidentally save it with a .html extension the server won't process your PHP code and the browser will just output it all to the screen.

Simple Example:

Every block of PHP code must start with <?php and end with ?>. The following example outputs the text "PHP is easy!" to the screen:

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

```
        <title>PHP Syntax Example</title>
    </head>
    <body>
        <?php
            echo "PHP is easy!";
        ?>
    </body>
</html>
```

Comments

While there is only one type of comment in HTML, PHP has two types. The first type we will discuss is the single line comment. The single line comment tells the interpreter to ignore everything that occurs on that line to the right of the comment. To do a single line comment type "//" or "#" and all text to the right will be ignored by PHP interpreter.

Example:

```
<html>
    <body>
        <?php
            //This is a comment

            /*This isa comment
            block
            */

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


Variables: Introduction

Variables are named "containers" that allow you to store a value or Variables are nothing but identifiers to the memory location to store data. This value can then be used by other parts of the application; simply by referring to the variable's name. For example, the application could display the contents of the variable to the user.

In PHP, variable names must start with a dollar sign ($). For example:

```
<?php
    $myVariable = "PHP is easy!";
    echo $myVariable;
?>
```

Variable Names:

rules:

When creating names for your variables, you need to ensure they comply with the following naming.

o All PHP variable names must start with a letter or underscore (_)"

o Variable names can only contain alpha-numeric characters and underscores (i.e. a-z, A-Z, 0-9, or_)

oVariable names must not contain spaces. For multi-word variable names, either separate the words with an underscore (_) or use capitalization.

- **PHP is a Loosely Typed Language:**

PHP is a loosely type language, which means that you do not need to declare a variable before assigning a value to it .In PHP, the variable is declared automatically when you use it.

For Example,

```
<?php
        $txt="Hello World!";
        $x=16;
?>
```

- **String Variables in PHP:**

A string variable is used to store and manipulate text. String variables are used for values that contains characters. For Example,

```
<?php
    $txt="Hello World";
    echo $txt;
?>
```


- **The Concatenation Operator:**

There is only one string operator in PHP.The concatenation operator (.) is used to put two string values together.To concatenate two string variables together, use the concatenation operator:

```
<?php
    $txt1="Hello World!";
    $txt2="What a nice day!";
    echo $txt1 . " " . $txt2;
?>
```

Output: Hello World! What a nice day!

- **The strlen() function:**

The strlen() function is used to return the length of a string.

```
<?php
        echo strlen("Hello world!");
```

```
?>
```

Output: 12

- **The strpos() function:**

The strpos() function is used to search for character within a string.

```
<?php
        echo strpos("Hello world!","world");
?>
```

Output: 6

Constants: Introduction:

A PHP constant is a variable that cannot be changed after the initial value is assigned during the execution of the script, except for magic constants, which are not really constants.

You create a constant variable using the define () function. The function accepts 2 arguments. The first must be a string and represents the name which will be used to access the constant. A constant name is case-sensitive by default and by convention, are always uppercase. A valid constant name starts with a letter or underscore, followed by any combination of letters, numbers or underscores, however, by convention only capital letters and underscore symbols are used. Constant names must be in quotes when they are defined

The second argument is the value of the constant and can be a string or numbers and can be set explicitly or as as the result of a function or equation.

The scope of a constant is global, which means that you can access constants anywhere in your script without regard to scope.

PHP has many pre-defined constants that can be used in your scripts.

```
<?php
 define ("SECE", "Sri Eshwar College Of Engineering");
 echo SECE;
?>
```


- **Magic Constants:**

Magic constants, as they are called, are not acutally constants but effectively behave like constants and are pre-defined. Magic constants use the same naming convention as PHP constants. The full list of magic constants can be found here.

There are 7 magic constants that change depending on where they are used. They are listed below.

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 with symlinks resolved whereas in older versions it contained relative path under some circumstances.
DIR	The directory of the file. If used inside an include, the directory of the included file is returned. This is equivalent to dirname(FILE). This directory name does not have a trailing slash unless it is the root directory. (Added in PHP 5.3.0.)
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).
NAMESPACE	The name of the current namespace (case-sensitive). This constant is defined in compile-time (Added in PHP 5.3.0)

Data Types:

PHP scripts deal with data in one form or another, usually stored in variables. PHP can work with different types of data. For example, a whole number is said to have an integer data type, while a string of text has a string data type.

- **PHP's scalar data types:**

Scalar data is data that only contains a single value. As of version 6, PHP features 6 scalar data types:

Type	Description	Example values
Integer	A whole number	7, -23
float	A floating point number	7.68, -23.45
string	A sequence of characters	"Hello", "abc123@#$"
unicode	A sequence of Unicode characters	"Hello", "abc123@#$"
binary	A sequence of binary (non-Unicode) characters	"Hello", "abc123@#$"
boolean	Either true or false	true, false

- **PHP's compound data types:**

Compound data can contain multiple values. PHP has 2 compound data types:

- Array - Can hold multiple values indexed by numbers or strings.
- Object - Can hold multiple values (properties), and can also contain methods (functions) for working on properties.

An array or object can contain multiple values, all accessed via one variable name. What's more, those values can themselves be other arrays or objects. This allows you to create quite complex collections of data.

Operators: Operators are used to operate on values.

Arithmetic Operators:

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus	5%2 10%8 10%2	1 2 0

++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Assignment Operators:

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y
%=	x%=y	x=x%y

Comparison Operators:

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
<>	is not equal	5<>8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Operator Precedence:

* / %

Highest Precedence

+ - .

< <= > >=

= = = != =

&&

Statements:

1. PHP Control Statements: if, elseif, else

if ( )

{}

elseif ( )

{}

else

{}

2. Switch:

switch ( )

{

case condition1 break;

case condition2 break;

}

3. while:

Syntax:

while (condition)

{

}

4. do:

do

{

} while (condition);

5. for:

Example use of the for statement:

```
<?php
for ( $n = 1; $n < 10; $n++)
{
echo "$n<BR>";
}   ?>
```

foreach:

Versions of PHP prior to version 4 do not support the foreach statement. The following code should list the contents of the array.

```
<?php
$tree = array("trunk", "branches", "leaves");
foreach ($tree as $part)
  echo "Tree part: $part ";
}
?>
```

break:

Is used to end the execution of a for, switch, or while statement

continue:

This statement is used to skip the rest of the current loop.

```
<?php
for ( $n = 1; $n < 10; $n++)
{
   echo "$n<BR>";
          if ($n == 5) continue;
  echo "This statement is skipped when $n = 5.<BR>";
}
```

?>

The continue statement can be given an optional parameter such as "continue 2" telling it how many levels of loops to skip.

Functions:

Definition:

A function is a block of code that performs a specific task. It has a name and it is reusable. To keep the script from being executed when the page loads, you can put it into a function. A function will be executed by a call to the function. You may call a function from anywhere within a page.

User-defined functions:

A function may be defined using syntax such as the following:

PHP function guidelines:

- Give the function a name that reflects what the function does
- The function name can start with a letter or underscore (not a number)

Example

A simple function that writes my name when it is called:

```
<html>
<body>
<?php
function writeName()
{
echo "Kai Jim Refsnes";
}
echo "My name is ";
writeName();
?>
</body>
</html>
```


Output:

My name is Kai Jim Refsnes

PHP Functions - Adding parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable.Parameters are specified after the function name, inside the parentheses.

Example 1

The following example will write different first names, but equal last name:

```
<html>
<body>
<?php
function writeName($fname)
```

```
{
echo $fname . " Refsnes.<br />";
}
echo "My name is ";
writeName("Kai Jim");
echo "My sister's name is ";
writeName("Hege");
echo "My brother's name is ";
writeName("Stale");
?>
</body>
</html>
```

Output:

My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes.
My brother's name is Stale Refsnes.

PHP Functions - Return values:

To let a function return a value, use the return statement.

Example

```
<html>
<body>
<?php
function add($x,$y)
{
$total=$x+$y;
return $total;
}
echo "1 + 16 = " . add(1,16);
?>
</body>
</html>
```

Output:

1 + 16 = 17

Functions within functions:

```
<?php
function foo()
{
 function bar()
```

```
{
  echo "I don't exist until foo() is called.\n";
}
}
/* We can't call bar() yet
   since it doesn't exist. */
foo();
/* Now we can call bar(),
   foo()'s processesing has
   made it accessible. */
bar();
?>
```

All functions and classes in PHP have the global scope - they can be called outside a function even if they were defined inside and vice versa.

PHP supports passing arguments by value (the default), passing by reference, and default argument values. Variable-length argument lists are supported only in PHP 4 and later; see Variable-length argument lists and the function references for func_num_args(), func_get_arg(), and func_get_args() for more information.

func_num_args:

func_num_args -- Returns the number of arguments passed to the function

Description

int **func_num_args** (void)

Gets the number of arguments passed to the function.

This function may be used in conjunction with func_get_arg() and func_get_args() to allow user-defined functions to accept variable-length argument lists.

Return Values:

Returns the number of arguments passed into the current user-defined function.

Errors/Exceptions:

Generates a warning if called from outside of a user-defined function.

Examples

Example

```
<?php
function
foo()
{
$numargs = func_num_args();
echo "Number of arguments: $numargs\n";
}
foo(1, 2, 3);    // Prints 'Number of arguments: 3'
```

?>

func_get_arg:

func_get_arg -- Return an item from the argument list

Description

mixed **func_get_arg** (int arg_num)

Gets the specified argument from a user-defined function's argument list.

This function may be used in conjunction with **func_get_args()** and **func_num_args()** to allow user-defined functions to accept variable-length argument lists.

Parameters:

arg_num

The argument offset. Function arguments are counted starting from zero.

Return Values

Returns the specified argument, or **FALSE**on error.

Errors/Exceptions

Generates a warning if called from outside of a user-defined function, or if arg_num is greater than the number of arguments actually passed.

Examples

Example 1. func_get_arg() example

```
<?php
function foo()
{
    $numargs = func_num_args();
    echo "Number of arguments: $numargs<br />\n";
    if ($numargs >= 2) {
        echo "Second argument is: " . func_get_arg(1) . "<br />\n";
    }
}
foo (1, 2, 3);
?>
```


func_num_args:

func_num_args -- Returns the number of arguments passed to the function

Description

int **func_num_args** (void)

Gets the number of arguments passed to the function.

This function may be used in conjunction with **func_get_arg()** and **func_get_args()** to allow user-defined functions to accept variable-length argument lists.

Return Values

Returns the number of arguments passed into the current user-defined function.

Errors/Exceptions

Generates a warning if called from outside of a user-defined function.

Examples

Example 1. func_num_args() example

```
<?php
function foo()
{
    $numargs = func_num_args();
    echo "Number of arguments: $numargs\n";
}
foo(1, 2, 3);    // Prints 'Number of arguments: 3'
?>
```

Variable functions

PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it.

Example for Variable function example

```
<?php
function foo() {
   echo "In foo()<br />\n";
}
function bar($arg = '')
{
   echo "In bar(); argument was '$arg'.<br />\n";
}
// This is a wrapper function around echo
function echoit($string)
{
   echo $string;
}
$func = 'foo';
$func();        // This calls foo()
$func = 'bar';
$func('test');  // This calls bar()
$func = 'echoit';
$func('test');  // This calls echoit()?>
```


Arrays:

What is an Array?

A variable is a storage area holding a number or text. The problem is, a variable will hold only one value.

PHP array are useful to store multiple data using a single variable. Using index of the PHP array we can access the stored variables. Indexes to an array element can either a number or a string. An array is a list variables. Each item in an array is commonly known as element of the array and can be accessed directly via its index.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1="Saab";
$cars2="Volvo";
$cars3="BMW";
```

An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

Each element in the array has its own index so that it can be easily accessed.

Types :

In PHP, there are three kind of arrays:

- **Numeric array** - An array with a numeric index
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

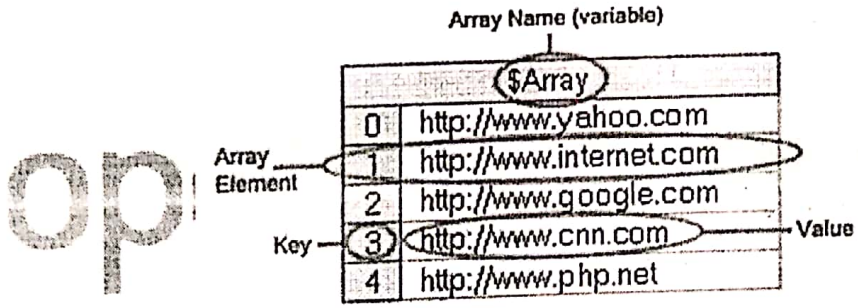

Figure 1.

Numeric Arrays

A numeric array stores each array element with a numeric index. There are two methods to create a numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

2. In the following example we assign the index manually:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

Example

In the following example you access the variable values by referring to the array name and index:

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

The code above will output:

Saab and Volvo are Swedish cars.

Associative Arrays:

An associative array, each ID key is associated with a value.

When storing data about specific named values, a numerical array is not always the best way to do it. With associative arrays we can use the values as keys and assign values to them.

Example 1

In this example we use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Example 2

This example is the same as example 1, but shows a different way of creating the array:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```


The ID keys can be used in a script:

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output: Peter is 32 years old.

Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub- array can be an array, and so on.

Example

In this example we create a multidimensional array, with automatically assigned ID keys:

```
$families    =
array        (
"Griffin"=>arr
ay (
"Peter
",
"Lois"
,
"Mega
n"
),
"Quagmirc"=>arr
ay (
"Glenn"
),
"Brown"=>arr
ay (
"Clevelan
d",
"Loretta",
"Junior"
)
);
```


The array above would look like this if written to the output: Array

```
(
[Griffin]    =>
Array
(
[0]      =>
Peter    [1]
=> Lois [2]
=> Megan
)
[Quagmire] => Array
(
```

[0] => Glenn

)

[Brown] =>

Array

(

[0] => Cleveland

[1] => Loretta

[2] => Junior

)

)

OOP: Object oriented programming is a concept that was created because of the need to overcome the problems that were found with using structured programming techniques. While structured programming uses an approach which is top down, OOP uses an approach which is bottom up.

Main Principal of OOPS are

1. Encapsulation : binding the data and function in a unit called class is called a data encapsulation. this allow the user to hide the information for outside world and doesn't allow the other user to change or modify the internal values of class.
2. Polymorphism: polymorphism is another strong feature of OOPS. it means one term in many forms.
3. Inheritance: another strong property of OOPS. this feature offers to derive a new class from an existing one and acquire all the feature of the existing class the new class which get the feature from existing class called derived class and other class is called base class.

Creating Classes: Basic class definitions begin with the keyword *class*, followed by a class name, followed by a pair of curly braces which enclose the definitions of the properties and methods belonging to the class. The class name can be any valid label which is a not a PHP reserved word. A valid class name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thus: *[a-zA-Z_\x7f-\xff] [a-zA-Z0-9_\x7f-\xff] **.

A class may contain its own constants, variables (called "properties"), and functions (called "methods").

Example for Simple Class Definition:

```
<?php
class SimpleClass
{
// property declaration
public $var = 'a default value';
// method declaration
public function displayVar() {
echo $this->var;
```

```
}
}
?>
```

Creating Objects:

It's much easier to create objects and use them than it is to define object classes, so before we discuss how to define classes. To create an object of a given class, use the new keyword:

$object = new *Class*;

Assuming that a Person class has been defined, here's how to create a Person object:

```
$rasmus = new Person;
```

Do not quote the class name, or you'll get a compilation error:

```
$rasmus = new 'Person'; // does not work
```

Some classes permit you to pass arguments to the new call. The class's documentation should say whether it accepts arguments. If it does, you'll create objects like this:

```
$object = new Person('Fred', 35);
```

The class name does not have to be hardcoded into your program. You can supply the class name through a variable:

```
$class = 'Person';
$object = new $class;
// is equivalent to
$object = new Person;
```

Specifying a class that doesn't exist causes a runtime error.

Variables containing object references are just normal variables—they can be used in the same ways as other variables. Of particular note is that variable variables work with objects, as shown here:

```
$account = new Account;
$object = 'account'
${$object}->init(50000, 1.10); // same as $account->init
```


String Manipulation and regular expression:

In PHP, and every other flavor of Web programming, a string is a variable contained between quotes with a literal value. For example, $number = "2" is a string variable, whereas $number = 2 is not. The first value is seen as text only, the latter as a numeric value. Simply put, a string is a text variable.

1.**ucwords** - this is used to convert a phrase or sentence into a title format.

```
<?p
hp
```

```
$string ="string manipulation in
php";
echo
ucwords($string);
?
>
```

2. **strtoupper** – this is used to capitalize every letter in a string.

```
<?p
hp
$string ="string manipulation in
php";
echo
strtoupper($string);
?
>
```

3. **strtolower** – this is used to bring all letters in a string to lowercase.

```
<?p
hp
$string ="String Manipulation in
PHP";
echo
strtolower($string);
?
>
```


4. **str_replace** – this function is used to replace a certain substring in a string

```
<?php
$string = "This tutorial is very useful for newbies";
$newstring = str_replace("useful", "difficult", $string );
echo $newstring;
?>
```

The str_replace() function takes 3 parameters. The first parameter is the piece of text you want to replace. The second is the text that will replace the original. The third parameter tells PHP what value to do the find and replace on.

5. **strpos** – find position of first occurrence of a string

Syntax: int **strpos** (string $haystack , mixed $needle [, int $offset = 0])

```
<?php
```

```
$string = "Hello World";
echo "The word World is at position number: ".strpos($string,"World");
?>
```

6. **substr** – Return part of a string

Syntax: string **substr** (string $string , int $start [, int $length])

```
<?php
$string = "Hello World";
$substring = substr($string,6);
echo $substring;
?>
```

The example above will display the word "World".

7.strlwr:

Prototype: char *strlwr(char *)

This function converts the given string to lowercase and returns the same.

8.strupr:

Prototype: char *strupr(char *)

This function converts the given string to UPPERCASE and returns it.

9.strncat:

Prototype: strncat(char *str1, const char,*str2, int n)

It appends first 'n' characters of str2 to the end of str1.

10.strncpy:

Prototype: int strncmp(char *str1, const char *str2, int n)

This function compares first 'n' characters of str1 with str2, it returns 0 if compare was successful.

11.stricmp:

Prototype: int stricmp(const char *str1, const char *str2)

It compares two strings str1 and str2 without regard to case and returns 0 on being successful.

12.strnicmp:

Prototype: int strnicmp(const char *str1, const char *str2, int n)

This function compares first 'n' characters of str1 with str2 without regard to case.

13.strrev:

Prototype: char *strrev(char *)

This function reverses the given string and returns it.

14.stcstr:

Prototype: char *strstr(const char *str1, const char *str2)

This function returns the pointer to where the first occurrence of string str2 is found inside str1.