Unit-6

Introduction to server-side Programming

The techniques of writing the programs whose action takes place on the web server are called server-side programming. Server-side programming is also called server-side scripting. Server- side scripting is a technique used in web development which involves employing scripts on a web server which produce a response customized for each user's (client's) request to the website. server side programming is written in various language such as PHP, ASP, JSP, PYTHON etc.

PHP, which stands for "PHP: Hypertext Preprocessor" is a widely-used Open Source general- purpose scripting language that is especially suited for Web development and can be embedded into HTML. Its syntax draws upon C, Java, and Perl, and is easy to learn. The main goal of the language is to allow web developers to write dynamically generated web pages quickly, but you can do much more with PHP.

File: first.php

```
<!DOCTYPE>
<html>
<body>
<?php
echo "<h2>Hello First PHP</h2>";
?>
</body>
</html>
```

Output:

Hello First PHP

What can PHP do?

PHP is mainly focused on server-side scripting, so you can do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies. But PHP can do much more.

There are three main areas where PHP scripts are used.

•**Server-side scripting.** This is the most traditional and main target field for PHP. You need three things to make this work. The PHP parser (Common Gateway Interface, CGI or server module), a web server and a web browser. You need to run the web server, with a connected PHP installation. You

can access the PHP program output with a web browser, viewing the PHP page through the server. All these can run on your home machine.
•**Command line scripting. You** can make a PHP script to run it without any server or browser. You only need the PHP parser to use it this way.
•**Writing desktop applications.** PHP is probably not the very best language to create a desktop application with a graphical user interface, but if you know PHP very well, and would like to use some advanced PHP features in your client-side applications you can also use PHP-GTK to write such programs.
PHP can be used on all major operating systems, including Linux, many Unix variants (including HP-UX, Solaris and OpenBSD), Microsoft Windows, Mac OS X, RISC OS, and probably others. PHP has also support for most of the web servers today. This includes Apache, Microsoft Internet Information Server, Personal Web Server, Netscape and iPlanet servers, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd, and many others. For the majority of the servers PHP has a module, for the others supporting the CGI standard, PHP can work as a CGI processor.

One of the strongest and most significant features in PHP is its support for a wide range of databases. Writing a database-enabled web page is incredibly simple. The following databases are currently supported:

•dBase
•mSQL
•MySQL
•PostgreSQL
•SQLite
•Sybase
We also have a database abstraction extension (named PDO) allowing you to transparently use any database supported by that extension. Additionally PHP supports ODBC, the Open Database Connection standard, so you can connect to any other database supporting this world standard.

Escaping from HTML

When PHP parses a file, it looks for opening and closing tags, which tell PHP to start and stop interpreting the code between them. Parsing in this manner allows php to be embedded in all sorts of different documents, as everything outside of a pair of opening and closing tags is ignored by the PHP parser. Most of the time you will see php embedded in HTML documents, as in this example.

```
<p>This is going to be ignored.</p>
<?php echo 'While this is going to be parsed.'; ?>
<p>This will also be ignored.</p>
```

Example. PHP Opening and Closing Tags

```
1.<?php echo 'if you want to serve XHTML or XML documents, do like this'; ?>

2.          <script language="php">
echo 'some editors (like FrontPage) don\'t like processing instructions';
</script>
```

3. <? echo 'this is the simplest, an SGML processing instruction'; ?>
<?= expression ?> This is a shortcut for "<? echo expression ?>"

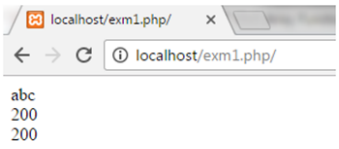
4. <% echo 'You may optionally use ASP-style tags'; %>
<%= $variable; # This is a shortcut for "<% echo . . ." %>

PHP $ and $$ Variables

•The $var (single dollar) is a normal variable with the name var that stores any value like string, integer, float, etc.

•The $$var (double dollar) is a reference variable that stores the value of the $variable inside it.

```
<?php
$x = "abc";
$$x = 200;
echo $x."<br/>";
echo $$x."<br/>";
echo $abc;
?>
```



Instruction separation
As in C , PHP requires instructions to be terminated with a semicolon at the end of each statement. The closing tag of a block of PHP code automatically implies a semicolon; you do not need to have a semicolon terminating the last line of a PHP block. The closing tag for the block will include the immediately trailing newline if one is present.

```
<?php
echo 'This is a test';
?>
<?php echo 'This is a test' ?>
<?php echo 'We omitted the last closing tag';
```

Comments in PHP
Comments in PHP are used to explain and document code. They are ignored by the PHP interpreter and do not affect the execution of the script. PHP supports three types of comments: single-line comments, multi-line comments, and shell-style comments.

Single-Line Comments
C/C++ Style: Use // to start a single-line comment.

```
<?php
// This is a single-line comment
echo 'Hello, World!';
?>
```

Shell Style: Use # to start a single-line comment.

```
<?php
# This is also a single-line comment
echo 'Hello, World!';
?>
```

Multi-Line Comments

C Style: Use /* to start and */ to end a multi-line comment

```
<?php
/*
 This is a multi-line comment
 that spans multiple lines.
*/
echo 'Hello, World!';
?>
```

Comments are useful for:

- Explaining code logic
- Documenting code changes
- Temporarily disabling code for testing purposes

Data Types in PHP

PHP supports several primitive data types, including scalar types, compound types, and special types.

Scalar Types : it holds only single value.

1. **String:** A sequence of characters, enclosed in quotes..

```
<?php
$string = "Hello, World!";
echo $string; // Outputs: Hello, World!
?>
```

2. **Integer:** A non-decimal number.

```
<?php
$int = 42;
echo $int; // Outputs: 42
?>
```

3. **Float (Double):** A number with a decimal point.

```
<?php
$float = 3.14;
echo $float; // Outputs: 3.14
?>
```

4. **Boolean:** Represents two states: true or false.

```
<?php
$bool = true;
echo $bool; // Outputs: 1 (true is displayed as 1)
?>
```

Compound Types

It can hold multiple values. There are 2 compound data types in PHP.

- array
- object

Arrays

PHP array is an ordered map (contains value on the basis of key).
It is used to hold multiple values of similar type in a single variable.

Advantage of PHP Array

Less Code: We don't need to define multiple variables.

Easy to traverse: By the help of single loop, we can traverse all the elements of an array.

Sorting: We can sort the elements of array.

PHP Array Types

There are 3 types of array in PHP.

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

PHP Indexed Array

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

There are two ways to define indexed array:

1st way:

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

2nd way:

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

Example
File: array1.php

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

Output:

Season are: summer, winter, spring and autumn

File: array2.php

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

Output:

Season are: summer, winter, spring and autumn

PHP Associative Array

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

There are two ways to define associative array:

1st way:

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

2nd way:

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

Example
File: arrayassociative1.php

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

Output:
Sonoo salary: 350000

John salary: 450000
Kartik salary: 200000

File: arrayassociative2.php

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

Output:

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

PHP Multidimensional Array

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

Definition

```
$emp = array

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

PHP Multidimensional Array Example

Id	Name	Salary
1	sonoo	400000
2	john	500000
3	rahul	300000

Object-Oriented Programming (OOP) in PHP

Basic Concepts

1. **Class**: A blueprint for creating objects. It defines properties and methods.
2. **Object**: An instance of a class.
3. **Property**: A variable that belongs to a class.
4. **Method**: A function that belongs to a class.

Key Principles

1. **Inheritance**: A class (child class) can inherit properties and methods from another class (parent class) using the extends keyword.

2. **Encapsulation**: Bundling data (properties) and methods into a single unit (class) and restricting access to some components using access modifiers like private, protected, and public.
3. **Polymorphism**: Methods in different classes can have the same name but behave differently, allowing objects to be treated as instances of their parent class rather than their actual class.

Formatted: Font: Not Bold

Formatted: No bullets or numbering

Formatted: No bullets or numbering

Formatted: Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.63 cm + Tab after: 1.27 cm + Indent at: 1.27 cm

Embedding PHP Scripts

PHP can be embedded within HTML to create dynamic web pages. This allows you to mix static HTML with dynamic PHP code, which is executed on the server and the result is sent to the browser.

Basic Syntax

PHP code is embedded in HTML using the `<?php ... ?>` tags.

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Embedding PHP Example</title>
</head>
<body>
    <h1>Welcome to My Website</h1>
    <p>Today's date is: <?php echo date('Y-m-d'); ?></p>
    <p>Random number: <?php echo rand(1, 100); ?></p>
</body>
</html>
```

Using Short Tags

PHP also supports short tags, which can be enabled in the PHP configuration (php.ini). However, using standard tags (<?php ... ?>) is recommended for better compatibility.

```
<!DOCTYPE html>

<html>

<head>

    <title>Short Tags Example</title>

</head>

<body>

    <h1>Welcome to My Website</h1>

    <p>Today's date is: <?= date('Y-m-d'); ?></p>

    <p>Random number: <?= rand(1, 100); ?></p>

</body>

</html>
```

Mixing PHP with HTML

You can switch between HTML and PHP as needed within a single file.

```
<!DOCTYPE html>

<html>

<head>

    <title>Mixing PHP and HTML</title>

</head>

<body>

    <?php

        $title = "Welcome to My Website";

        $date = date('Y-m-d');

        $number = rand(1, 100);
```

```
    ?>

    <h1><?php echo $title; ?></h1>

    <p>Today's date is: <?php echo $date; ?></p>

    <p>Random number: <?php echo $number; ?></p>

</body>

</html>
```

Basic PHP

1. Variables

Variables in PHP are containers that store data. They are dynamic, meaning their value can change throughout the execution of the script. Variable names are case-sensitive and should start with a $ sign followed by the name (e.g., $variableName).

Example:

```
<?php
// Declaring and initializing variables
$name = "Alice"; // String variable
$age = 25;       // Integer variable
$height = 5.6;   // Float variable

// Concatenating variables in a string
echo "Name: " . $name . "<br>";     // Output: Name: Alice
echo "Age: " . $age . "<br>";       // Output: Age: 25
echo "Height: " . $height . " feet"; // Output: Height: 5.6 feet
?>
```

2. Operators

Operators are symbols that perform operations on variables and values. PHP supports several types of operators:

Arithmetic Operators: Used for mathematical operations.

- + (Addition)
- (Subtraction)
- (Multiplication)
- / (Division)
- % (Modulus)

Comparison Operators: Used to compare values.

- == (Equal to)
- != (Not equal to)

- >(Greater than)
- < (Less than)
- >= (Greater than or equal to)
- <= (Less than or equal to)

Logical Operators: Used to perform logical operations.

- && (Logical AND)
- || (Logical OR)
- ! (Logical NOT)

Example:

```
<?php
// Arithmetic operations
$x = 15;
$y = 4;

$sum = $x + $y;           // Addition
$difference = $x - $y;    // Subtraction
$product = $x * $y;       // Multiplication
$quotient = $x / $y;      // Division
$remainder = $x % $y;     // Modulus

// Comparison
$isGreater = ($x > $y);   // True

// Logical operations
$logicalAnd = ($x > 10 && $y < 5); // True
$logicalOr = ($x > 20 || $y < 5);  // True

// Outputting results
echo "Sum: " . $sum . "<br>";
echo "Difference: " . $difference . "<br>";
echo "Product: " . $product . "<br>";
echo "Quotient: " . $quotient . "<br>";
echo "Remainder: " . $remainder . "<br>";
echo "Is Greater: " . ($isGreater ? 'True' : 'False') . "<br>";
echo "Logical AND: " . ($logicalAnd ? 'True' : 'False') . "<br>";
echo "Logical OR: " . ($logicalOr ? 'True' : 'False');
?>
```

3. Expressions

Expressions are combinations of variables, operators, and values that produce a result. They can include arithmetic, comparison, and logical operations.

```
<?php
// Variables
$a = 8;
$b = 3;
```

```
// Arithmetic expression
$expression1 = ($a * $b) + 7; // (8 * 3) + 7 = 31

// Comparison expression
$expression2 = ($a > $b) ? "True" : "False"; // True

// Logical expression
$expression3 = ($a > 5 && $b < 10) ? "Both conditions are true" : "One or both conditions are false";

// Outputting results
echo "Expression 1 (Arithmetic): " . $expression1 . "<br>";
echo "Expression 2 (Comparison): " . $expression2 . "<br>";
echo "Expression 3 (Logical): " . $expression3;
?>
```

4. Constants

Constants are similar to variables, but once they are set, their values cannot be changed. They are defined using the define() function and do not have the $ symbol.

```
<?php
// Defining constants
define("EARTH_GRAVITY", 9.81);  // Gravity in m/s²
define("PI", 3.14159);        // Value of PI

// Using constants in calculations
$circleCircumference = 2 * PI * 5; // Circumference of a circle with radius 5

// Outputting results
echo "Earth's Gravity: " . EARTH_GRAVITY . " m/s²<br>"; // Output: Earth's Gravity: 9.81 m/s²
echo "Circumference of Circle: " . $circleCircumference . "<br>"; // Output: Circumference of Circle: 31.4159
?>
```

Control Structures

Control structures in PHP are used to control the flow of a script's execution. They include conditionals, loops, and other constructs that allow developers to execute code based on certain conditions or repeatedly execute a block of code.

1. **if Statement**

Executes a block of code if the specified condition is true.

Syntax.

```
if (condition) {
   // code to be executed if condition is true
}
```

Example.

```
$age = 20;
```

```
if ($age >= 18) {
   echo "You are an adult.";
}
```

2. if….else Statement

Executes one block of code if the condition is true, otherwise executes another block of code.

Syntax

```
if (condition) {
   // code to be executed if condition is true
} else {
   // code to be executed if condition is false
}
```

Example

```
$age = 16;
if ($age >= 18) {
   echo "You are an adult.";
} else {
   echo "You are a minor.";
}
```

3. if...elseif...else Statement

Evaluates multiple conditions in sequence and executes the corresponding block of code for the first true condition.

Syntax

```
if (condition1) {
   // code to be executed if condition1 is true
} elseif (condition2) {
   // code to be executed if condition2 is true
} else {
   // code to be executed if both conditions are false
}
```

Example

```
$score = 85;
if ($score >= 90) {
   echo "Grade: A";
} elseif ($score >= 80) {
   echo "Grade: B";
} else {
   echo "Grade: C";
}
```

4. switch Statement

Evaluates an expression against multiple cases and executes the matching block of code.

Syntax

```
switch (n) {
  case value1:
    // code to be executed if n == value1
    break;
  case value2:
    // code to be executed if n == value2
    break;
  default:
    // code to be executed if n is different from all values
}
```

Example

```
$day = "Tuesday";
switch ($day) {
  case "Monday":
    echo "Today is Monday.";
    break;
  case "Tuesday":
    echo "Today is Tuesday.";
    break;
  default:
    echo "Today is neither Monday nor Tuesday.";
}
```

Loops

Like any other language, loop in PHP is used to execute a statement or a block of statements, multiple times until and unless a specific condition is met. This helps the user to save both time and effort of writing the same code multiple times.

PHP supports four types of looping techniques;

- for loop
- while loop
- do-while loop
- foreach loop

1. **for Loop**

Repeatedly executes a block of code for a specified number of iterations.

Syntax

```
for (init; condition; increment) {
  // code to be executed
}
```

Example

```
for ($i = 1; $i <= 5; $i++) {
  echo "The number is: $i <br>";
}
```

2. **while Loop**

Repeatedly executes a block of code as long as the specified condition is true.

Syntax

```
while (condition) {
   // code to be executed as long as condition is true
}
```

Example

```
$i = 1;
while ($i <= 5) {
   echo "The number is: $i <br>";
   $i++;
}
```

3. **do...while Loop**

Executes a block of code once, and then repeats the loop as long as the condition is true.

Syntax

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

Example

```
$i = 1;
do {
   echo "The number is: $i <br>";
   $i++;
} while ($i <= 5);
```

4. **foreach Loop**

Iterates over each element in an array, executing a block of code for each element.

Syntax

```
foreach ($array as $value) {
   // code to be executed
}
```

Example

```
$colors = array("red", "green", "blue", "yellow");
foreach ($colors as $value) {
   echo "$value <br>";
}
```

Break and Continue

1. **break Statement**

Used to exit a loop or switch statement.

```
for ($i = 0; $i < 10; $i++) {
   if ($i == 5) {
      break;
   }
   echo $i . " ";
}
```

2. continue Statement

Used to skip the current iteration of a loop and continue with the next iteration.

```
for ($i = 0; $i < 10; $i++) {
  if ($i == 5) {
     continue;
  }
  echo $i . " ";
}
```

PHP Functions

PHP function is a piece of code that can be reused many times. It can take input as argument list and return value. There are thousands of built-in functions in PHP.

In PHP, we can define Conditional function, Function within Function and Recursive function also.

Advantage of PHP Functions

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

PHP User-defined Functions

We can declare and call user-defined functions easily. Let's see the syntax to declare user-defined functions.

Syntax:

```
function functionname(){
//code to be executed
}
```

Example

File: function1.php

```
<?php
function sayHello(){
echo "Hello PHP Function";
}
sayHello();//calling function
?>
```

PHP Function Arguments

We can pass the information in PHP function through arguments which is separated by comma.

PHP supports Call by Value (default), Call by Reference, Default argument values and Variable-length argument list.

Let's see the example to pass single argument in PHP function.
File: functionarg.php

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

Output:

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

File: functionarg2.php

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

Output:

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

PHP Call By Reference

Value passed to the function doesn't modify the actual value by default (call by value). But we can do so by passing value as a reference.

By default, value passed to the function is call by value. To pass value as a reference, you need to use ampersand (&) symbol before the argument name.

Let's see a simple example of call by reference in PHP.

File: functionref.php

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

```
echo $str;
?>
```

Output:
Hello Call By Reference

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

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

Hello Rajesh
Hello Sonoo
Hello John

Functions with Return Values

1. **Returning a Value** Functions can return values using the return statement. The returned value can be stored in a variable or used directly.

```
function functionName() {
  // code to be executed
  return $value;
}
```
Example
```
function add($a, $b) {
  return $a + $b;
}
```

2. **Storing the Return Value** The returned value from a function can be stored in a variable or used in an expression.

```
$result = add(3, 5); // $result is 8
echo $result; // Output: 8
```

3.7 Recursion
Recursion is a programming technique where a function calls itself to solve a problem. In PHP, recursive functions are used to solve problems that can be broken down into smaller, similar

subproblems. The key to recursion is having a base case that stops the recursion to prevent infinite loops.
PHP also supports recursive function call like C/C++. In such case, we call current function within function. It is also known as recursion.
It is recommended to avoid recursive function call over 200 recursion level because it may smash the stack and may cause the termination of script.

Basic Structure of a Recursive Function
A recursive function typically includes:

- A base case: The condition under which the function stops calling itself.
- A recursive case: The part where the function calls itself with a modified argument.

The factorial of a non-negative integer nnn is the product of all positive integers less than or equal to nnn. It can be defined recursively as:

- 0!=10! = 10!=1 (base case)
- n!=n×(n−1)!n! = n \times (n-1)!n!=n×(n−1)! (recursive case)

Example 2 : Factorial Number

```
<?php
function factorial($n)
{
  if ($n < 0)
    return -1; /*Wrong value*/
  if ($n == 0)
    return 1; /*Terminating condition*/
  return ($n * factorial ($n -1));
}

echo factorial(5);
?>
```

Output:
120

3.8 String Manipulation:
String manipulation is a common task in PHP, and the language provides a rich set of functions to work with strings. Here are some of the most useful string manipulation functions along with examples:

1. Getting the Length of a String

- **strlen()**: Returns the length of a string.

```
$string = "Hello, World!";
echo strlen($string); // Output: 13
```

2. Changing Case of a String

- **strtoupper()**: Converts all characters in a string to uppercase.
- **strtolower()**: Converts all characters in a string to lowercase.
- **ucfirst()**: Converts the first character of a string to uppercase.

- **ucwords()**: Converts the first character of each word in a string to uppercase.

```
$string = "hello, world!";
echo strtoupper($string); // Output: HELLO, WORLD!
echo strtolower($string); // Output: hello, world!
echo ucfirst($string); // Output: Hello, world!
echo ucwords($string); // Output: Hello, World!
```

3. Replacing Substrings

- **str_replace()**: Replaces all occurrences of a search string with a replacement string.
- **str_ireplace()**: Case-insensitive version of str_replace().

```
$string = "Hello, World!";
echo str_replace("World", "PHP", $string); // Output: Hello, PHP!
echo str_ireplace("world", "PHP", $string); // Output: Hello, PHP!
```

4. Finding Substrings

- **strpos()**: Finds the position of the first occurrence of a substring.
- **strrpos()**: Finds the position of the last occurrence of a substring.
- **stripos()**: Case-insensitive version of strpos().
- **strripos()**: Case-insensitive version of strrpos().

```
$string = "Hello, World!";
echo strpos($string, "World"); // Output: 7
echo stripos($string, "world"); // Output: 7
```

5. Extracting Substrings

- **substr()**: Returns a portion of a string.
- **substr_replace()**: Replaces a portion of a string with another string.

```
$string = "Hello, World!";
echo substr($string, 7, 5); // Output: World
echo substr_replace($string, "PHP", 7, 5); // Output: Hello, PHP!
```

6. Repeating and Padding Strings

- **str_repeat()**: Repeats a string a specified number of times.
- **str_pad()**: Pads a string to a new length.

```
$string = "Hello";
echo str_repeat($string, 3); // Output: HelloHelloHello
echo str_pad($string, 10, "!"); // Output: Hello!!!!!
```

7. Trimming Strings

- **trim()**: Strips whitespace (or other characters) from the beginning and end of a string.
- **ltrim()**: Strips whitespace (or other characters) from the beginning of a string.
- **rtrim()**: Strips whitespace (or other characters) from the end of a string.

```
$string = "  Hello, World!  ";
echo trim($string); // Output: Hello, World!
echo ltrim($string); // Output: Hello, World!
echo rtrim($string); // Output:   Hello, World!
```

8. Splitting and Joining Strings

- **explode()**: Splits a string into an array by a specified delimiter.
- **implode()**: Joins array elements into a string with a specified delimiter.

```
$string = "Hello, World!";
$array = explode(", ", $string);
print_r($array); // Output: Array ( [0] => Hello [1] => World! )

$array = ["Hello", "World"];
echo implode(", ", $array); // Output: Hello, World
```

Using Regular Expression

Regular expressions in PHP are a powerful tool for pattern matching and text manipulation. PHP provides support for regular expressions using the Perl Compatible Regular Expressions (PCRE) library.

Basic Functions

Here are the key functions used for regular expressions in PHP:

1. **preg_match()**: Performs a regular expression match.
2. **preg_match_all()**: Performs a global regular expression match.
3. **preg_replace()**: Performs a search and replace with a regular expression.
4. **preg_split()**: Splits a string by a regular expression.
5. **preg_grep()**: Returns the elements of an array that match a pattern.

Examples

1. preg_match()

The preg_match() function searches a string for a pattern and returns whether a match was found.

```
$pattern = "/world/i";
$string = "Hello, World!";
if (preg_match($pattern, $string)) {
   echo "Match found!"; // Output: Match found!
} else {
   echo "Match not found.";
}
```

The /i modifier makes the search case-insensitive.

2. preg_match_all()

The preg_match_all() function finds all matches of a pattern in a string.

```
$pattern = "/[a-z]+/i";
$string = "Hello, World! 123";
preg_match_all($pattern, $string, $matches);
print_r($matches);
// Output: Array (  [0] => Hello [1] => World ) )
```

3. preg_replace()

The preg_replace() function performs a search and replace using a regular expression.

```
$pattern = "/world/i";
$replacement = "PHP";
$string = "Hello, World!";
$newString = preg_replace($pattern, $replacement, $string);
echo $newString; // Output: Hello, PHP!
```

4. preg_split()

The preg_split() function splits a string by a regular expression.

```
$pattern = "/[\s,]+/";
$string = "Hello, World! Welcome to PHP.";
$parts = preg_split($pattern, $string);
print_r($parts);
// Output: Array ( [0] => Hello [1] => World! [2] => Welcome [3] => to [4] => PHP. )
```

This example splits the string by spaces and commas.

5. preg_grep()

The preg_grep() function returns array elements that match a pattern.

```
$pattern = "/^a/i";
$array = ["apple", "banana", "apricot", "cherry"];
$result = preg_grep($pattern, $array);
print_r($result);
// Output: Array ( [0] => apple [2] => apricot )
```

This finds all array elements that start with "a".

Example: Email Validation

Here's an example of using regular expressions to validate an email address.

```
$email = "test@example.com";
$pattern = "/^[a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/";
if (preg_match($pattern, $email)) {
   echo "Valid email address."; // Output: Valid email address.
} else {
   echo "Invalid email address.";
}
```

- /^: Asserts the position at the start of the string.
- **[a-zA-Z0-9._%-]+**: Matches one or more characters that can be lowercase or uppercase letters (`a-zA-Z`), digits (`0-9`), dots (`.`), underscores (_), percent signs (`%`), or hyphens (`-`).
- **@**: Matches the `@` symbol.
- **[a-zA-Z0-9.-]+**: Matches one or more characters that can be lowercase or uppercase letters (`a-zA-Z`), digits (`0-9`), dots (`.`), or hyphens (`-`).
- **.**: Escapes the dot (`.`) character, matching a literal dot.
- **[a-zA-Z]{2,6}**: Matches 2 to 6 characters that can be lowercase or uppercase letters (`a-zA-Z`), representing the domain suffix (e.g., com, net, org).
- **$/**: Asserts the position at the end of the string.

Exception Handling in PHP
Exception handling is a powerful mechanism of PHP, which is used to handle runtime errors (runtime errors are called exceptions). So that the normal flow of the application can be maintained.
The main purpose of using exception handling is **to maintain the normal execution of the application**.

What is an Exception?
An exception is an unexcepted outcome of a program, which can be handled by the program itself. Basically, an exception disrupts the normal flow of the program. But it is different from an error because an exception can be handled, whereas an error cannot be handled by the program itself. In other words, - "An unexpected result of a program is an exception, which can be handled by the program itself." Exceptions can be thrown and caught in PHP.

Why needs Exception Handling?
PHP provides a powerful mechanism, exception handling. It allows you to handle runtime errors such as IOException, SQLException, ClassNotFoundException, and more. A most popular example of exception handling is - divide by zero exception, which is an arithmetic exception.

Exception handling is almost similar in all programming languages. It changes the normal flow of the program when a specified error condition occurs, and this condition is known as exception.
PHP offers the following keywords for this purpose:

try -
The try block contains the code that may have an exception or where an exception can arise. When an exception occurs inside the try block during runtime of code, it is caught and resolved in catch block. The try block must be followed by catch or finally block. A try block can be followed by minimum one and maximum any number of catch blocks.

catch -
The catch block contains the code that executes when a specified exception is thrown. It is always used with a try block, not alone. When an exception occurs, PHP finds the matching catch block.

throw -
It is a keyword used to throw an exception. It also helps to list all the exceptions that a function throws but does not handle itself.
Remember that each throw must have at least one "catch".

finally -
The finally block contains a code, which is used for clean-up activity in PHP. Basically, it executes the essential code of the program.

Basic Concepts
1. **Throwing an Exception**: You can throw an exception using the throw statement.
2. **Catching an Exception**: You can catch an exception using try and catch blocks.

3. **Finally Block**: An optional finally block can be used to execute code regardless of whether an exception was thrown or not.

Examples

Throwing and Catching Exceptions

```
function divide($a, $b) {
   if ($b == 0) {
      throw new Exception("Division by zero."); // Throwing an exception
   }
   return $a / $b;
}

try {
   echo divide(10, 2); // This will work
   echo divide(10, 0); // This will throw an exception
} catch (Exception $e) {
   echo 'Caught exception: ',  $e->getMessage(), "\n"; // Catching the exception
}
```

Using Multiple Catch Blocks

You can use multiple catch blocks to handle different types of exceptions.

```
class CustomException extends Exception {}

try {
   throw new CustomException("Custom exception occurred.");
} catch (CustomException $e) {
   echo 'Caught custom exception: ',  $e->getMessage(), "\n";
} catch (Exception $e) {
   echo 'Caught general exception: ',  $e->getMessage(), "\n";
}
```

The Finally Block

A finally block is optional and can be used to execute code regardless of whether an exception was thrown or not.

```
try {
   echo divide(10, 0);
} catch (Exception $e) {
   echo 'Caught exception: ',  $e->getMessage(), "\n";
} finally {
   echo "This is the finally block.\n";
}
```