Unit 4: Server Side Programming with PHP

How PHP fits into a web page?

PHP is a server-side programming language that is commonly used to build dynamic web pages and web applications. PHP scripts are executed on the server side, which means that the code is processed on the web server before being sent to the client's web browser. Here is an example of how PHP code can be embedded in an HTML code . Save the file with extension .php and run with server.

```
<!DOCTYPE html>

<html>

<head>

<title>My Web Page</title>

</head>

<body>

<h1>Welcome to my web page!</h1>

<p><?php echo "Today is " . date("l"); ?></p><?php

$name = "John";

echo "<p>Hello, $name!</p>";

?>

</body>

</html>
```

In this example, we're embedding PHP code in an HTML file using the <?php ?> tags. The date() function is used to display the current day of the week, and the $name variable is used to display a personalized greeting.

Variables and constants

In PHP, variables can be declared and initialized using the following syntax:

```
$variable_name = value;
```

Here's an example:

```
$name = "John Doe"; // string variable

$age = 30; // integer variable

$balance = 100.50; // float variable

$is_active = true; // boolean variable
```

Variable names can contain letters, numbers, and underscores, but cannot start with a number. Variable names are case sensitive in PHP, so $name and $Name would be considered two separate variables. Variables can be

used to store various types of data, including strings, integers, floats, booleans, arrays, and objects. Variables can also be used to store the results of functions, expressions, and other computation.

Operators

In PHP, an operator is a symbol or keyword that performs a specific operation on one or more operands (variables, values, or expressions). Here are some of the most commonly used operators in PHP:

- Arithmetic Operator
  - ✓ '+' Addition
  - ✓ '-' Subtraction
  - ✓ '*' Multiplication
  - ✓ '/' Division
  - ✓ '%' Modulus
  - ✓ '**' Exponentiation (Raise a number to a power)

- Assignment operator
  - ✓ '=' Equal to
  - ✓ '+=' Addition Assignment
  - ✓ '-=' Subtraction Assignment
  - ✓ '*=' Multiplication Assignment
  - ✓ '/=' Division Assignment
  - ✓ '%=' Modulus Assignment
- Comparison Assignment
  - ✓ '==' Equal to
  - ✓ '===' Identical to
  - ✓ '!==' Not Equal to
  - ✓ '>' Greater than
  - ✓ '>' less than

- Logical Operator
  - ✓ '&&' Logical AND
  - ✓ '||' Logical OR
  - ✓ '!' Logical Not
  - ✓ 'XOR' Logical XOR
- String Operator
  - ✓ Concatenation (Join two or more strings together)
- Conditional Operator
  - ✓ '?:' Ternary Operator(return one of two values depending on a condition)
  - ✓ '??' Null coalescing Operator (return the first non-null value among two or more operands

Working with text and numbers

PHP provides a variety of built-in functions and operators for working with both text and numbers. Here are some examples:

Working with Text

➢ strlen() - returns the length of a string.

➢ str_replace() - replaces all occurrences of a string with another string.

➢ substr() - returns a substring from a string.

➢ strpos() - finds the position of the first occurrence of a substring in a string.

➢ strtolower() - converts a string to lowercase.

➢ strtoupper() - converts a string to uppercase.

➢ trim() - removes whitespace from the beginning and end of a string.

➢ explode() - splits a string into an array based on a delimiter.

```
$name = "John Smith";
echo strlen($name); // Output: 10
echo str_replace("Smith", "Doe", $name); // Output: John Doe
echo substr($name, 5); // Output: Smith
echo strpos($name, "Smith"); // Output: 5
echo strtolower($name); // Output: john smith
echo strtoupper($name); // Output: JOHN SMITH
echo trim(" Hello, World! "); // Output: Hello, World!
echo implode(", ", explode(" ", $name)); // Output: John, Smith
```

Working with Numbers

➢ abs() - returns the absolute value of a number.

➢ pow() - raises a number to a power.

➢ sqrt() - returns the square root of a number.

➢ round() - rounds a number to a specified number of decimal places.

➢ ceil() - rounds a number up to the nearest integer.

➢ floor() - rounds a number down to the nearest integer.

➢ rand() - generates a random number.

Example:

```
$num1 = 10;
```

```
$num2 = -5;

echo abs($num2); // Output: 5

echo pow($num1, 2); // Output: 100

echo sqrt($num1); // Output: 3.1622776601684

echo round(3.14159265359, 2); // Output: 3.14

echo ceil(3.14); // Output: 4

echo floor(3.99); // Output: 3

echo rand(1, 10); // Output: a random number between 1 and 10
```

Making decisions with control statements (if, switch, loop)

In PHP, control statements are used to control the flow of execution of a program based on certain conditions. Here are some of the commonly used control statements in PHP:

Conditional Statements

if statement

executes a block of code if a condition is true.

if...else statement

executes a block of code if a condition is true, otherwise it executes another block of code.

if...elseif...else statement

executes a block of code if the first condition is true, otherwise it checks the next condition and so on, until a true condition is found, otherwise it executes the else block.

Example:

```
$num = 10;

if ($num > 0)

{

 echo "Positive number";

}

if ($num % 2 == 0)

{

 echo "Even number";

}

else

{

echo "Odd number";
```

```
}
if ($num > 10) {
 echo "Greater than 10";
}
 elseif ($num == 10)
{
 echo "Equal to 10";
}
else
{
echo "Less than 10";
}
```

Looping Statements

While loop

executes a block of code as long as a condition is true.

```
$num = 1;
while ($num <= 10) {
 echo $num;
 $num++;
}
```

Do...while loop

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

```
$num = 1;
do {
 echo $num;
 $num++;
} while ($num <= 10);
```

For loop

executes a block of code a specified number of times.

```
for ($i = 1; $i <= 10; $i++) {
 echo $i;
}
```

Foreach loop

iterates over the elements of an array or an object.

```
$fruits = array("apple", "banana", "orange");
foreach ($fruits as $fruit) {
 echo $fruit;
}
```

Jump Statements

Break statement

Terminates the current loop or switch statement and transfers control to the statement immediately following the loop or switch.

```
for ($i = 1; $i <= 10; $i++) {
 if ($i == 5) {
 break;
 }
 echo $i; // Output: 1234
}
```

Continue statement

Skips the current iteration of a loop and continues with the next iteration.

```
for ($i = 1; $i <= 10; $i++) {
 if ($i % 2 == 0) {
 continue;
 }
 echo $i; // Output: 13579
}
```

Return statement

Terminates the current function and returns a value to the caller.

```
function add($num1, $num2) {
 return $num1 + $num2;
}
$result = add(2, 3); // Output: 5
```

Working with arrays, strings, datetime and files

In PHP, an array is a data structure that stores a collection of values, such as numbers or

strings. Here are some key points to know about arrays in PHP:

Creating an Array

You can create an array in PHP using the array() function or by using square brackets [] in PHP 5.4 and later:

```
// Using the array() function
$fruits = array("apple", "banana", "orange");
// Using square brackets
$numbers = [1, 2, 3, 4];
```

Accessing Array Elements

You can access the elements of an array using their index. In PHP, arrays are zero-indexed, which means the first element has an index of 0. You can use square brackets to access an element by its index:

```
$fruits = array("apple", "banana", "orange");
echo $fruits[0]; // Output: apple
echo $fruits[1]; // Output: banana
echo $fruits[2]; // Output: orange
```

Associative Arrays

In addition to indexed arrays, PHP also supports associative arrays, which use key-value pairs to store data. You can create an associative array using the array() function or square brackets, and access its elements using their keys:

```
// Using the array() function
$person = array("name" => "John", "age" => 30, "city" => "New York");
```

```
// Using square brackets
$person = ["name" => "John", "age" => 30, "city" => "New York"];
echo $person["name"]; // Output: John
echo $person["age"]; // Output: 30
echo $person["city"]; // Output: New York
```

Array Functions

PHP provides a variety of built-in functions for working with arrays, including:

➢ count() - returns the number of elements in an array.

➢ sort() - sorts an array in ascending order.

➢ rsort() - sorts an array in descending order.

➢ array_push() - adds one or more elements to the end of an array.

➢ array_pop() - removes the last element from an array.

➢ array_merge() - merges two or more arrays into a single array.

Functions

In PHP, a function is a block of code that can be called by name and executed whenever it is needed. Functions can accept arguments and return values, making them a powerful tool for building reusable and modular code.

***Defining a Function**

You can define a function using the function keyword, followed by the name of the function, a set of parentheses that may contain parameters, and a block of code enclosed in curly braces:

```
function functionName($arg1, $arg2, ...) {
 // function body
}
```

***Calling a Function**

Once a function is defined, you can call it by using its name followed by a set of parentheses that may contain arguments:

```
functionName($arg1, $arg2, ...);
```

***Function Arguments**

Functions can accept zero or more arguments, which are passed to the function when it is called. You can specify the arguments that a function accepts by listing their names in the parentheses after the function name:

***Function Return Values**

Functions can return values using the return keyword, which terminates the function and returns a value to the caller:

```
function add($num1, $num2) {
 return $num1 + $num2;
}
$result = add(2, 3); // Output: 5
```