**THE BUILDING BLOCKS OF PHP**

**Variables:**

* Definition: Variables are containers that hold values like numbers, strings, objects, arrays, or Booleans.
* Purpose: They allow for dynamic data manipulation without hard-coding values.
* Syntax: Declared with a $ sign followed by the variable name, e.g., $variableName.
* Naming Rules: Must start with a letter or underscore; can include letters, numbers, and underscores; no spaces allowed.
* Example: $num1 = 8; $num2 = 23;

**Importance:**

* Enable creating templates for operations without specific values.
* Use variables to handle data that may change during the script's execution or when passed to another script.

**Local and Global Variables:**

* Local Variables: The value is only present within the function or script where it resides.
* Global Variables: Shared across scripts when defined as global.
  + Example: If $name is global in both scriptA.php and scriptB.php, and these scripts are connected, they share the same value.

**Superglobal Variables:**

* Predefined arrays always available in PHP:
  + $\_GET: Variables provided via the GET method.
  + $\_POST: Variables provided via the POST method.
  + $\_COOKIE: Variables provided via cookies.
  + $\_FILES: Variables provided via file uploads.
  + $\_SERVER: Information like headers, file paths, and script locations.
  + $\_ENV: Variables provided as part of the server environment.
  + $\_REQUEST: Variables via GET, POST, or COOKIE inputs.
  + $\_SESSION: Variables registered in a session.

**Best Practices:**

* Use meaningful variable names for readability and maintainability.
* Superglobals help secure applications by reducing the likelihood of user-injected input.

**Data Types**:

* PHP is loosely typed, meaning it automatically determines data types when values are assigned to variables.
* This flexibility allows variables to hold different types of data at different times.

**Standard Data Types in PHP**:

1. **Boolean**: Represents a value of true or false.
2. **Integer**: A whole number, e.g., 5.
3. **Float (Double)**: A floating-point number, e.g., 3.234.
4. **String**: A collection of characters, e.g., "hello".
5. **Object**: An instance of a class.
6. **Array**: An ordered set of keys and values.
7. **Resource**: Reference to an external resource (e.g., a database).
8. **NULL**: An uninitialized variable.

**Dynamic Typing**:

* PHP allows variables to change types dynamically, which provides flexibility but can lead to errors if a variable holds an unexpected type.

**Testing Variable Types**:

* PHP provides is\_\* functions to test the type of a variable:
  + is\_bool(): Checks if a variable is a Boolean.
  + is\_int(): Checks if a variable is an integer.
  + is\_double(): Checks if a variable is a float.
  + is\_string(): Checks if a variable is a string.
  + is\_array(): Checks if a variable is an array.
  + is\_resource(): Checks if a variable is a resource.
  + is\_null(): Checks if a variable is NULL.
  + is\_numeric(): Checks if a variable is numeric

**Changing Type with** settype():

* **Functionality**: Changes the type of an existing variable.
* **Syntax**: settype($variable, 'new\_type');
* **Example**: Converts a float to various types:

php

$undecided = 3.14;

settype($undecided, 'string'); // Converts to string

settype($undecided, 'integer'); // Converts to integer

settype($undecided, 'double'); // Converts back to double

settype($undecided, 'bool'); // Converts to boolean

**Changing Type by Casting**:

* **Principle**: Creates a copy of the variable with a new type, leaving the original variable unchanged.
* **Syntax**: $newvar = (new\_type) $originalvar;
* **Example**: Casting a variable to different types:

php

$undecided = 3.14;

$holder = (double) $undecided; // Creates a double copy

$holder = (string) $undecided; // Creates a string copy

$holder = (integer) $undecided; // Creates an integer copy

$holder = (boolean) $undecided; // Creates a boolean copy

**Usage**:

* **settype()**: Directly changes the type of the variable.
* **Casting**: Produces a new variable of a specified type, leaving the original variable untouched.

**Example of** settype():

php

$undecided = 3.14;

settype($undecided, 'string');

echo $undecided; // Outputs: "3.14"

settype($undecided, 'integer');

echo $undecided; // Outputs: 3

**Example of Casting**:

php

$undecided = 3.14;

$newvar = (integer) $undecided;

echo $newvar; // Outputs: 3

echo $undecided; // Outputs: 3.14 (original variable remains unchanged)

**Note on Automatic Casting**:

* PHP automatically casts variables when necessary, but this cast is temporary.
* Explicit type setting (using settype() or casting) may be needed to ensure a variable persistently holds a specific type.

**Practical Application:**

* When users input numbers in a form, they are treated as strings. PHP automatically converts them during arithmetic operations:

**Why Test Variable Types?**

* **Ensure Functionality**: Functions often receive data from various sources. Verifying the data type ensures the function processes the data correctly. For example, a function expecting a resource type may not handle a string appropriately.
* **Prevent Errors**: By checking data types, you can avoid runtime errors and ensure the script behaves as expected.

**Operators and Expressions**:

* **Operators**: Symbols used to manipulate data in variables. They perform actions and usually produce new values.
  + **Operands**: Values on which operators act. Typically, there are two or more operands for one operator.
  + **Example**: (4 + 5), where 4 and 5 are operands and + is the operator, resulting in 9.

**Types of Operators**:

1. **Assignment Operator**:
   * Symbol: =
   * Usage: Assigns the value of the right-side operand to the left-side operand.
   * Example: $name = "Jimbo";
2. **Arithmetic Operators**:
   * Perform basic arithmetic operations:
     + **Addition** (+): 10 + 3 results in 13.
     + **Subtraction** (-): 10 - 3 results in 7.
     + **Division** (/): 10 / 3 results in 3.33.
     + **Multiplication** (\*): 10 \* 3 results in 30.
     + **Modulus** (%): 10 % 3 results in 1.
3. **Concatenation Operator**:
   * Symbol: .
   * Usage: Appends the right-side operand to the left-side operand, treating both as strings.
   * Example: "hello" . " world" results in "hello world".
4. **Combined Assignment Operators**:
   * Combine a standard operator with an assignment operator to transform and assign values in one step.
   * Examples:
     + +=: $x += 5 is equivalent to $x = $x + 5.
     + -=: $x -= 5 is equivalent to $x = $x - 5.
     + /=: $x /= 5 is equivalent to $x = $x / 5.
     + \*=: $x \*= 5 is equivalent to $x = $x \* 5.
     + %=: $x %= 5 is equivalent to $x = $x % 5.
     + .=: $x .= " test" is equivalent to $x = $x . " test".

**Automatically Incrementing and Decrementing**:

* **Post-Increment/Decrement**: $x++ (increments $x by 1 after the operation) and $x-- (decrements $x by 1 after the operation).
* **Pre-Increment/Decrement**: ++$x (increments $x by 1 before the operation) and --$x (decrements $x by 1 before the operation).

**Comparison Operators**:

* Used to perform comparative tests and return Boolean values (true or false).
* **Operators**:
  + == (Equivalence): $x == 5 (false if $x is 4)
  + != (Nonequivalence): $x != 5 (true if $x is 4)
  + === (Identical): $x === 4 (true if $x is 4 and is the same type)
  + > (Greater than): $x > 4 (false if $x is 4)
  + >= (Greater than or equal to): $x >= 4 (true if $x is 4)
  + < (Less than): $x < 4 (false if $x is 4)
  + <= (Less than or equal to): $x <= 4 (true if $x is 4)

**Logical Operators**:

* Used to combine Boolean values and test complex conditions.
* **Operators**:
  + || or or (Or): Returns true if either operand is true.
  + && or and (And): Returns true if both operands are true.
  + xor (Exclusive Or): Returns true if one operand is true, but not both.
  + ! (Not): Returns true if the single operand is not true.

**Operator Precedence**:

* **Precedence Order**: Determines the order in which operators are evaluated in an expression.
  + Higher precedence operators are evaluated before lower precedence operators.
* **List of Operators by Precedence**:
  + Highest precedence: ++, --, (cast)
  + /, \*, %
  + +, -
  + <, <=, =>, >
  + ==, ===, !=
  + &&
  + ||
  + Lowest precedence: =, +=, -=, /=, \*=, %=, .=, and xor or

**Constants**:

* **Purpose**: Constants store values that remain unchanged throughout the execution of a script.
* **Creation**: Use PHP’s built-in define() function to create a constant.
  + **Syntax**: define("CONSTANT\_NAME", value);
  + Example: define("THE\_YEAR", "2012");

**Usage**:

* **Access**: Access constants by their name without a dollar sign.
* **Convention**: Constants are typically named in uppercase letters.

**Case Sensitivity**:

* By default, constant names are case sensitive.
* To make a constant name case insensitive, pass true as a third argument in define().
  + Example: define("THE\_YEAR", "2012", true);

**Predefined Constants**:

* PHP provides built-in constants:
  + **FILE**: Returns the name of the current file.
  + **LINE**: Returns the current line number.
  + **PHP\_VERSION**: Returns the PHP version.

**Best Practices**:

* Keep constants case sensitive and use uppercase characters for ease of understanding and to follow standard conventions.

**FLOW CONTROL FUNCTIONS IN PHP**

The if statement in PHP allows you to execute a block of code only if a specified condition evaluates to true.

**1. How to use the if statement to execute code if a test expression evaluates to true**

The if statement allows you to execute a block of code only if a specified condition is true.

**Syntax**:

php

if (condition) {

// Code to be executed if the condition is true

}

**2. How to execute alternative blocks of code when the test expression of an if statement evaluates to false**

You can use the else statement to execute a block of code when the if statement's test expression evaluates to false.

**Syntax**:

php

if (condition) {

// Code to be executed if the condition is true

} else {

// Code to be executed if the condition is false

}

**3. How to use the switch statement to execute code based on the value returned by a test expression**

The switch statement allows you to execute one block of code from multiple options based on the value of a test expression.

**Syntax**:

switch (expression) {

case value1:

// Code to be executed if expression equals value1

break;

case value2:

// Code to be executed if expression equals value2

break;

// More cases...

default:

// Code to be executed if expression doesn't match any case

}

**4. How to repeat execution of code using a while statement**

The while statement allows you to execute a block of code repeatedly as long as a specified condition is true.

**Syntax**:

while (condition) {

// Code to be executed as long as the condition is true

}

**5. How to use for statements to make neater loops**

The for statement provides a concise way to create loops, with initialization, condition, and increment/decrement in one line.

**Syntax**:

for (initialization; condition; increment/decrement) {

// Code to be executed for each iteration

}

**6. How to break out of loops**

The break statement allows you to exit a loop before it has completed all its iterations.

**Example**:

php

<?php

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

if ($i > 5) {

break;

}

echo "The number is $i <br/>";

}

?>

**7. How to nest one loop within another**

You can nest loops by placing one loop inside another. This is useful for complex iterations.

**Syntax**:

php

for (initialization; condition; increment/decrement) {

// Outer loop code

for (initialization; condition; increment/decrement) {

// Inner loop code

}

}

**Example**:

php

<?php

for ($i = 1; $i <= 3; $i++) {

for ($j = 1; $j <= 3; $j++) {

echo "i is $i and j is $j <br/>";

}

}

?>

**8. How to use PHP start and end tags within control structures**

You can embed PHP code within HTML using PHP start (<?php) and end (?>) tags. This is useful for mixing HTML with PHP control structures.

**Example with** if **Statement**:

<?php

$showMessage = true;

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>PHP Tags in HTML</title>

</head>

<body>

<?php if ($showMessage): ?>

<p>Hello, this message is shown because the condition is true.</p>

<?php else: ?>

<p>This message is shown when the condition is false.</p>

<?php endif; ?>

</body>

</html>

**Loops in PHP**

Loops are used in PHP to execute a block of code repeatedly as long as a specified condition is true. They are fundamental in programming for tasks that require repetition.

**Types of Loops in PHP:**

1. **while Loop**
2. **do...while Loop**
3. **for Loop**
4. **foreach Loop**

**1. while Loop**

The while loop executes a block of code as long as a specified condition is true.

**Syntax**:

while (condition) {

// Code to be executed

}

**Example**:

<?php

$counter = 1;

while ($counter <= 5) {

echo "Counter is $counter <br/>";

$counter++;

}

?>

**Explanation**:

* The loop continues to execute as long as the $counter is less than or equal to 5.
* $counter++ increments the counter by 1 on each iteration.

**2. do...while Loop**

The do...while loop will always execute the block of code at least once, and then it will repeat the loop as long as the specified condition is true.

**Syntax**:

do {

// Code to be executed

} while (condition);

**Example**:

<?php

$counter = 1;

do {

echo "Counter is $counter <br/>";

$counter++;

} while ($counter <= 5);

?>

**Explanation**:

* The code block runs at least once, regardless of the condition.
* The condition is checked after the execution of the code block.

**3. for Loop**

The for loop is used when you know in advance how many times you want to execute a statement or a block of statements.

**Syntax**:

for (initialization; condition; increment/decrement) {

// Code to be executed

}

**Example**:

<?php

for ($i = 1; $i <= 5; $i++) {

echo "The number is $i <br/>";

}

?>

**Explanation**:

* **Initialization**: Sets the loop counter ($i = 1).
* **Condition**: Runs the loop as long as the condition ($i <= 5) is true.
* **Increment/Decrement**: Modifies the loop counter ($i++).

**4. foreach Loop**

The foreach loop works only on arrays and is used to loop through each key/value pair in an array.

**Syntax**:

foreach ($array as $value) {

// Code to be executed

}

foreach ($array as $key => $value) {

// Code to be executed

}

**Example**:

<?php

$fruits = ["Apple", "Banana", "Cherry"];

foreach ($fruits as $fruit) {

echo "Fruit: $fruit <br/>";

}

?>

**Explanation**:

* Iterates over each element in the $fruits array.
* $fruit contains the value of the current element.

**continue Statement**

The continue statement skips the remaining code inside the current iteration of the loop and jumps to the next iteration.

**Example with** for **Loop**:

<?php

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

if ($i == 5) {

continue; // Skip the rest of the code in this iteration

}

echo "The number is $i <br/>";

}

?>

**Explanation**:

* The loop runs from 1 to 10.
* When $i is 5, the continue statement is executed, skipping the echo statement for that iteration and moving to the next iteration.

**Use Cases:**

* break: Useful when you need to exit the loop as soon as a certain condition is met.
  + Example: Searching for a specific value in an array and stopping once you find it.
* continue: Useful when you want to skip over certain values or iterations but still continue looping.
  + Example: Filtering out certain values while processing a list.

**WORKING WITH FUNCTIONS**

**How to Define and Call Functions from Within Your Scripts**

**Defining a Function**

To define a function, you use the function keyword followed by the function name and a pair of parentheses. The code that makes up the function is placed within curly braces.

**Syntax**:

function functionName() {

// Code to be executed

}

**Calling a Function**

To call a function, simply use its name followed by parentheses.

**Example**:

<?php

greet(); // Outputs: Hello, World!

?>

**How to Pass Values to Functions and Receive Values in Return**

**Passing Values**

You can pass values (also called arguments) to a function by specifying them within the parentheses in the function definition.

**Syntax**:

function functionName($parameter) {

// Code to be executed

}

**Receiving Values in Return**

You can use the return keyword to return a value from a function.

**Example**:

<?php

function add($a, $b) {

return $a + $b;

}

$result = add(5, 3); // $result is now 8

echo $result; // Outputs: 8

?>

**How to Call a Function Dynamically Using a String Stored in a Variable**

You can call a function dynamically by storing its name in a variable and using the variable to call the function.

**Example**:

<?php

function greet() {

echo "Hello, World!";

}

$functionName = "greet";

$functionName(); // Outputs: Hello, World!

?>

**How to Access Global Variables from Within a Function**

To access global variables from within a function, you need to use the global keyword.

**Example**:

<?php

$globalVar = 10;

function testScope() {

global $globalVar;

echo $globalVar; // Outputs: 10

}

testScope();

?>

**How to Give a Function a “Memory”**

You can give a function a "memory" by using static variables. Static variables retain their value between function calls.

**Example**:

<?php

function counter() {

static $count = 0;

$count++;

echo $count;

}

counter(); // Outputs: 1

counter(); // Outputs: 2

counter(); // Outputs: 3

?>

**How to Pass Data to Functions by Reference**

You can pass data to functions by reference using the & symbol. This allows the function to modify the original variable.

**Example**:

<?php

function addOne(&$value) {

$value++;

}

$number = 5;

addOne($number);

echo $number; // Outputs: 6

?>

**How to Verify That a Function Exists Before Calling It**

You can use the function\_exists() function to check if a function has been defined before calling it.

**Example**:

<?php

if (function\_exists('greet')) {

greet(); // Outputs: Hello, World!

} else {

echo "Function does not exist.";

}

function greet() {

echo "Hello, World!";

}

?>