VARIABLES

In programming, variables are containers or storage locations that hold data, and their values can change during the execution of a program. They serve as symbolic names for values and are used to store and manipulate data within a program. Variables provide a way to label and reference memory locations in the computer's memory.

Creating (Declaring) PHP Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:

Example

```
$x = 5;

$y = "John"
```

In the example above, the variable $x will hold the value 5, and the variable $y will hold the value "John".

Note: When you assign a text value to a variable, put quotes around the value.

Note: Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

PHP Variables

A variable can have a short name (like $x and $y) or a more descriptive name ($age, $carname, $total_volume).

Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)

Output Variables

The PHP echo statement is often used to output data to the screen.

The following example will show how to output text and a variable:

Example

```
$txt = "W3Schools.com";

echo "I love $txt!";
```

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives an option to specify the data type expected when declaring a function, and by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

Variable Types

PHP has no command for declaring a variable, and the data type depends on the value of the variable.

Example

```
$x = 5;      // $x is an integer

$y = "John"; // $y is a string

echo $x;

echo $y;
```

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

Get the Type

To get the data type of a variable, use the var_dump() function.

Example

The var_dump() function returns the data type and the value:

```
$x = 5;

var_dump($x);
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```php
$x = "John";

echo $x;
```

String variables can be declared either by using double or single quotes, but you should be aware of the differences. Learn more about the differences in the PHP Strings chapter.

Assign Multiple Values

You can assign the same value to multiple variables in one line:

Example

All three variables get the value "Fruit":

```php
$x = $y = $z = "Fruit";
```

PHP Variables Scope

In PHP, variables can be declared anywhere in the script.

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

Global and Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

ExampleGet your own PHP Server

Variable with global scope:

```
$x = 5; // global scope


function myTest() {

  // using x inside this function will generate an error

  echo "<p>Variable x inside function is: $x</p>";

}

myTest();


echo "<p>Variable x outside function is: $x</p>";
```

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

Example

Variable with local scope:

```
function myTest() {

  $x = 5; // local scope

  echo "<p>Variable x inside function is: $x</p>";

}

myTest();


// using x outside the function will generate an error
```

```
echo "<p>Variable x outside function is: $x</p>";
```

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared

PHP The global Keyword

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

Example

```
$x = 5;

$y = 10;



function myTest() {

  global $x, $y;

  $y = $x + $y;

}



myTest();

echo $y; // outputs 15
```

PHP also stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

Example

```
$x = 5;

$y = 10;


function myTest() {

  $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];

}


myTest();

echo $y; // outputs 15
```

PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the static keyword when you first declare the variable:

Example

```
function myTest() {

  static $x = 0;

  echo $x;

  $x++;
}


myTest();

myTest();

myTest();
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

PHP echo and print Statements

In PHP, both `echo` and `print` are used to output data to the browser or the standard output. While they share similar functionalities, there are some differences between the two:

1. `echo` Statement:

- **Syntax:**

  ```php

  echo expression1, expression2, ...;

  ```

- **Usage:**

  - `echo` can take multiple parameters, separated by commas.

  - It does not have a return value.

  - It is marginally faster than `print`.

- **Examples:**
  ```php
  $name = "John";
  $age = 25;

  echo "Hello, $name!"; // Output: Hello, John!
  echo "Age: " . $age;   // Output: Age: 25
  ```

2. `print` Statement:

- **Syntax:**
  ```php
  print expression;
  ```

- **Usage:**
  - `print` can only take one argument.
  - It returns 1, so it can be used in expressions.
  - It is slightly slower than `echo`.

- **Examples:**
  ```php
  $name = "Jane";
  ```

```
  $age = 30;

  print "Hello, $name!"; // Output: Hello, Jane!
  print "Age: " . $age;   // Output: Age: 30
```

Choosing Between `echo` and `print`:

- **Return Value:**
  - `echo` does not have a return value.
  - `print` returns 1, allowing it to be used in expressions.

- **Multiple Parameters:**
  - `echo` can take multiple parameters, separated by commas.
  - `print` can only take one argument.

- **Speed:**
  - `echo` is generally faster than `print`, but the difference is usually negligible in real-world scenarios.

- **Common Use:**
  - `echo` is more commonly used in PHP scripts due to its flexibility and efficiency.

Both `echo` and `print` are commonly used to output HTML and other content in PHP scripts. The choice between them often depends on personal preference and specific use cases. In modern PHP development, `echo` is more commonly used for its flexibility and speed.