Variable Scope

This lesson discusses the scope of variables including the local and global variables.

We'll cover the following

Introduction

The scope of a variable refers to the variable's visibility, i.e., which part of the program can access that variable.

Types of Variables

There are two types of variables depending on the scope (or visibility):

- local variables
- global variables

Local Variables

Variables that are defined within a function are inside a local scope hence are called **local variables**. These variables cannot be accessed outside of the function they are declared in.

```
<?php
function foo()
{
    $number = 10;
    echo $number;
}
foo(); //Will print 10 because text defined inside function is a local variable
?>
```

This code prints **10** because text defined inside a *function* is a *local variable* and cannot be accessed by the script outside the function.

Global Variables

The *variables* defined outside of a *function* are referred to as **global** *variables*.

> A **global** *variable* can be accessed in any part of the program.

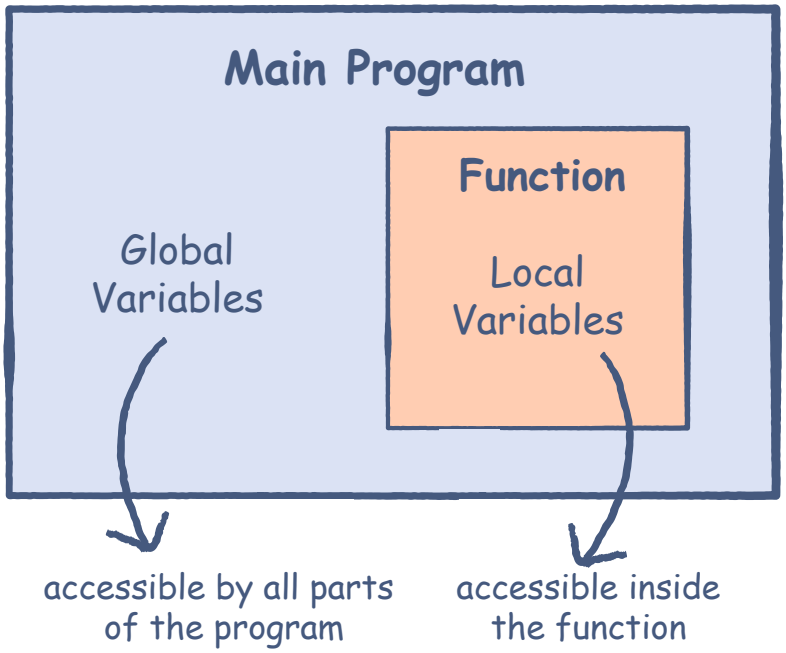
However, in order to modify a *global variable* within a function, we need to use the `global` *keyword*. This is done by placing the `global` *keyword* in front of the *variable*.

Below is an example of how we use **global** *variables*:

```
<?php

$num1 = 5;
$num2 = 2;

function multiply(){
  global $num1; // Accessing global variables from function scope requires this explicit statement
  global $num2;
  $answer = $num1*$num2;
  return $answer;
}

// When in the global scope, regular global variables can be used
// without explicitly stating 'global $variable;'
echo "num1 is: $num1\n";
echo "num2 is: $num2\n";
echo multiply();
?>
```

> **Note:** You cannot assign a value to a variable in the same statement as the `global` *keyword*.

The following figure illustrates the difference between global and local variables:



Using Variable Variables for Function Calls

We discussed variable variables earlier. *Variable variables* are useful for mapping function/method calls. Consider the following code snippet where we use variable variables to call a function `sum` which, as the name suggests, takes two integers and returns their sum.

```
<?php
function sum($x, $y)
{
    return $x + $y;
}
$funcName = 'sum';
echo $funcName(2, 4); // outputs 6;
?>
```

Now let's solve some challenges before moving on to newer concepts.