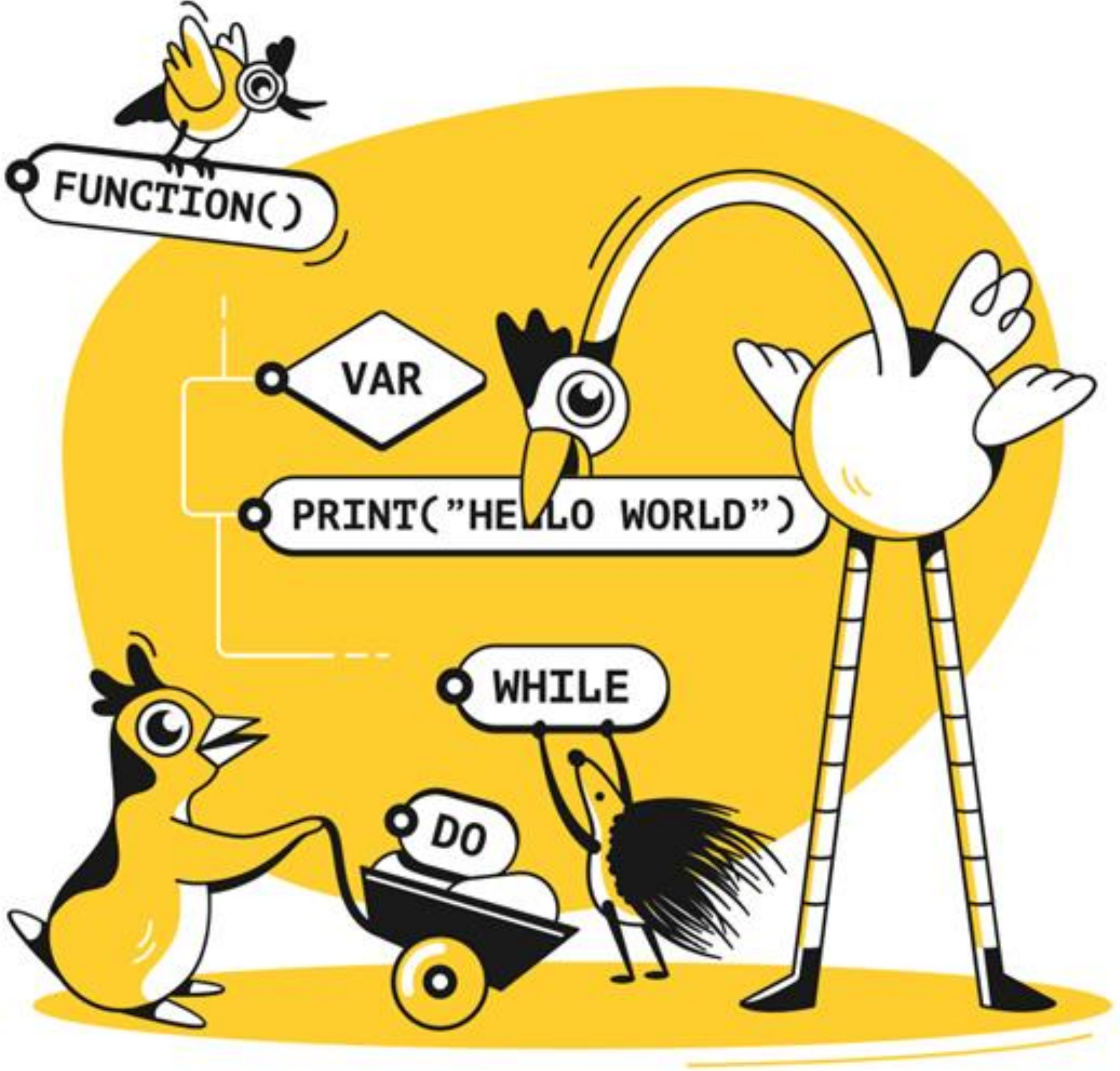
Web Technology II (BIT301)

Instructor: Prakash Neupane


FUNCTION()
VAR
PRINT("HELLO WORLD")
WHILE
DO

Functions

- **Function Definition:**
- Named block of code
- Performs a specific task

- **Parameters:**
- Accepts input values
- Can act upon these parameters

- **Return Values:**
- Optionally provides a result
- Can return a single value or an array

Functions

- **Compile-Time Efficiency:**
- Compiled only once for the page
- Saves on compilation time

- **Code Reliability:**
- Centralizes bug fixes
- Fixes in one place affect all uses

- **Readability Improvement:**
- Isolates code for specific tasks
- Enhances code organization

Functions

- PHP's Strength: Functions
  - Real power of PHP resides in its functions
- Abundance of Built-in Functions:
  - Over 1000 pre-defined functions
  - Ready-to-use for various tasks
- Custom Function Creation:
  - Ability to craft personalized functions
  - Tailored to specific needs

Functions

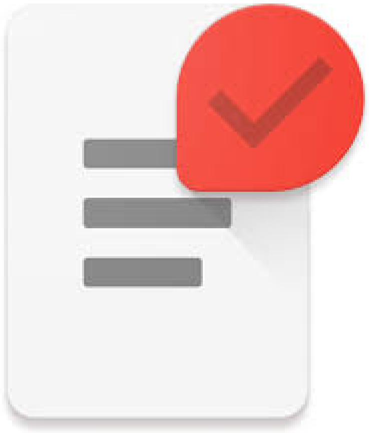
- Built-in
  - PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.
  - Example: Array, Calendar, Date, Directory, zip etc.
- User Defined
  - Besides the built-in PHP functions, it is possible to create your own functions.
    - A function is a block of statements that can be used repeatedly in a program.
    - A function will not execute automatically when a page loads.
    - A function will be executed by a call to the function.

Functions

- Built-in
  - *<?php*
  - *$length =strlen("Good Afternoon!");*
  - *echo $length;*
  - *?>*

- User Defined
- *<?php*
- *function printText( ){*
  - *echo “Good Afternoon!”;*
- *}*
- *printText( );*
- *?>*

Functions



- *Quick Task:*
  - *Create your own functions for displaying your name, address and contact.*

Defining and Calling Functions

- A user-defined function declaration starts with the word function:

- **Syntax**

```
function functionName() {
 // code to be executed;
}
```

Defining and Calling Functions

- A user-defined function declaration starts with the word function:

- **Syntax**

```
function functionName() {
 // code to be executed;
}
functionName();
```

A function name must start with a letter or an underscore. Function names are NOT case-sensitive.

Defining and Calling Functions

- ***Example:***

```
<?php
function foo() {
    return "Foo says: 'Hello, I'm Foo!'";
}

function bar() {
    return "Bar says: 'Hi there, I'm Bar!'";
}

// Calling the functions
echo foo()."\n";
echo bar();
?>
```

Variable Scope

- Variables Without Functions:
  - Variables are global, usable anywhere on the page
  - No separation between page and function variables
- Functions Introduce Isolation:
  - Functions have their own variable scope
  - Variables within a function are distinct from page and other functions
- Variable Access Rules:
  - Variables defined in a function (including parameters) are not accessible outside the function
  - Variables defined outside a function are not accessible by default within the function

Variable Scope

- Illustration: Example demonstrates the isolation of variables within functions

```
<?php
$a = 3;
function foo(){
    – $a += 2;
}
foo();
echo $a;
?>
```

- $a inside the function foo() is distinct from the $a outside the function.
- The outer $a remains 3 on the page, unaffected by the function.
- Inside the foo() function, $a takes on the value 2 but doesn't affect the outer $a.

Variable Scope

- Illustration: Example demonstrates the isolation of variables within functions

```
<?php
$a = 3;
function foo(){
    – $a += 2;
}
foo();
echo $a;
?>
```

- $a inside the function foo() is distinct from the $a outside the function.
- The outer $a remains 3 on the page, unaffected by the function.
- Inside the foo() function, $a takes on the value 2 but doesn't affect the outer $a.

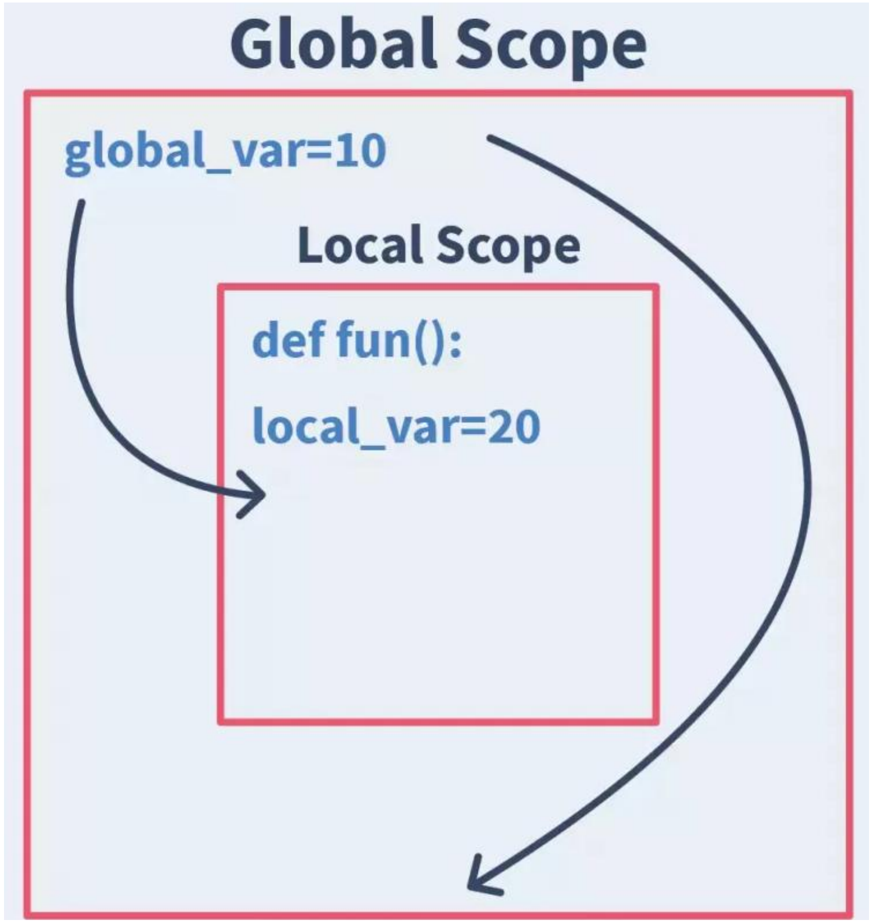
Variable Scope

- Illustration: To fix the bug in above program at line number 5

```
<?php
$a = 3; // Outer $a

function foo() {
   $a = 2; // Inner $a
   return $a;
}

$innerA = foo(); // Returns 2, but the outer $a remains 3

echo "Outer \$a: $a\n"; // Output: Outer $a: 3
echo "Inner \$a: $innerA"; // Output: Inner $a: 2

?>
```

Variable Scope

- Global Scope
- Local Scope



Variable Scope

- Global Scope
  - A variable declared outside a function has a GLOBAL SCOPE and can only be accessed outside a function:

```
<?php
$x = 5; // Global scope variable

function myTest() {
  // Attempting to access $x inside this function will result in an error
  echo "Variable x inside the function is: $x";
}
myTest();

echo "Variable x outside the function is: $x";
?>
```

Variable Scope

- The global Keyword in PHP:
  - **Purpose**:
  - Used to access a global variable from within a function.
  - **Usage**:
  - Prefix variables with the global keyword inside the function.

```php
<?php
$globalVar = 42; // Global variable

function accessGlobal() {
    global $globalVar; // Access the global variable
    echo "The global variable is: $globalVar";
}

accessGlobal(); // Output: The global variable is: 42

?>
```

Variable Scope

- $GLOBALS Array in PHP:
- **Purpose**:
  - Stores all global variables in an associative array.
- **Access**:
  - Variables are stored with their names as keys in the $GLOBALS array.
- **Usage**:
  - Accessible from within functions, allows direct manipulation of global variables.

```
<?php
$globalVar = 42; // Global variable

function accessGlobalWithGLOBALS() {
   $GLOBALS['globalVar']  = $GLOBALS['globalVar']+50;
// Update the global variable via $GLOBALS
}

accessGlobalWithGLOBALS();
echo "The global variable is now: " . $globalVar;
// Output: The global variable is now: 100

?>
```

Variable Scope

- **Local Scope of Variables in Functions:**
  - Variables declared within a function are said to have local scope.
  - Local scope means the variables are accessible only within the function where they are declared.
  - These variables cannot be accessed outside the function.

```
<?php
function myFunction() {
    $localVariable = "I am local!";
    echo $localVariable;
}

myFunction(); // Output: I am local!

// Attempting to access $localVariable here would result in an
    error

?>
```

Variable Scope

- **Preserving Local Variables with 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.
  - By default, local variables in a function are deleted after the function execution is completed.
  - To retain a local variable's value between function calls, use the static keyword when declaring the variable.

```
<?php
function incrementCounter() {
   static $counter = 0; // Declare a static variable


   $counter++; // Increment the static variable
   echo "Counter: $counter\n";
}
for($i=1; $i<5; $i++)
incrementCounter(); // Output: Counter: i
?>
```

Function Parameters

- Functions can expect an arbitrary number of arguments, declared by the function definition.
- There are two different ways to pass parameters to a function.

  -By value

  -By reference

Function Parameters

Passing Parameters by Value:

- When passing parameters by value in PHP, wesend a copy of the original value to the function.
- Any modifications made to the parameter within the function do not affect the original value outside of the function.

Function Parameters

Passing Parameters by Value: Examples

```
<?php
function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, 5);

?>
```

Function Parameters

Passing Parameters by Value: Examples

```
<?php
function square($number) {
$number = $number * $number;
echo "Inside function: $number\n";
}
$originalNumber = 5;
square($originalNumber);
echo "Outside function: $originalNumber";
?>
```

Function Parameters

Passing Parameters by Reference

- Allows a function to directly modify the original variable.
- Useful for altering variable values without returning them.
- Indicated by using an ampersand (&) before the parameter name.
- Changes made to the parameter inside the function affect the original variable.

- Key Benefits:

  - Efficient for large data structures as it avoids copying.

  - Useful for functions with multiple return values.

  - Enables in-place updates without reassignment.

Function Parameters

Passing Parameters by Reference

```
<?php
function square(&$number) {
$number = $number * $number;
echo "Inside function: $number\n";
}
$originalNumber = 5;
square($originalNumber);
echo "Outside function: $originalNumber";
?>
```

Function Parameters

Passing Parameters by Reference

```
<?php
function doubleFirstElement(&$arr) {
    $arr[0] *= 2;
}
$array = [3, 5, 7];
foreach ($array as $value) {
    echo "$value ";
}
doubleFirstElement($array); // $array is now [6, 5, 7]
echo "\n";
// Print the updated
foreach ($array as $value) {
    echo "$value ";
}
```

Function Parameters

Passing Parameters by Reference

```
<?php
function swap(&$a, &$b) {
    $temp = $a;
    $a = $b;
    $b = $temp;
}
$x = 10;
$y = 20;
swap($x, $y); // $x is now 20, and $y is now 10
echo "$x\n";
echo $y;
?>
```

Function Parameters

Default Parameters

In PHP, wecan specify default argument values for function parameters.

Default argument values are used when a value for a particular parameter is not provided during a function call.

This allows you to make certain parameters optional while still providing a default value to be used when no value is passed.

Syntax:

```
function functionName($param1, $param2 = defaultValue) {
    // Function code
}
```

Function Parameters

Default Parameters: Examples

```
<?php

function greet($name, $greeting = "Hello") {

    echo "$greeting!, $name!\n";

}



greet("Class"); // Output: Hello, Alice!

greet("All", "Good Afternoon"); // Output: Hi, Bob!



?>
```

```
<?php
function multiply($a, $b = 2) {
    return $a * $b;
}

$result1 = multiply(5);      // $result1 is 10 (default $b is 2)

$result2 = multiply(5, 3);   // $result2 is 15 (explicit $b is 3)
echo $result1;
echo $result2;
?>
```

Return values

In PHP, functions can return values using the return statement.

When a function returns a value, it means it produces a result that can be used in your code.

Syntax:

```
function functionName($param1, $param2) {
    // Function code here
    return $result;         // Return a value
}
```

```
<?php
function add($a, $b) {
    $sum = $a + $b;
    return $sum;
}

$result = add(3, 5); // Call the function and store the result
echo "The sum is: $result"; // Output: The sum is: 8
?>
```

Return values

In PHP, functions can return values using the return statement.

When a function returns a value, it means it produces a result that can be used in your code.

Syntax:

```
function functionName($param1, $param2) {
    // Function code here
    return $result;        // Return a value
}
```

```
<?php
function add($a, $b) {
    $sum = $a + $b;
    return $sum;
}

$result = add(3, 5); // Call the function and store the result
echo "The sum is: $result"; // Output: The sum is: 8
?>
```

Variable Functions

Defining a Variable Function:

We can assign the name of a function to a variable, and then use that variable to call the function.

Syntax:

$functionName = 'functionName';   // Store the function name in a variable

$result = $functionName();   // Call the function using the variable

```
<?php
function greet() {
    echo "Hello, World!\n";
    return 5;
}

$funcName = 'greet'; // Store the function name in a variable
$res=$funcName(); // Call the function using the variable
echo $res;
?>
```

Anonymous Functions

Also known as lambda functions or closures.

Allow creating functions without naming them.

Basic Syntax:
Define using the “function” keyword without a name.

May include parameters and a function body in curly braces.

Simple Anonymous Function:

```
$add = function ($a, $b) {
 ...
};
```

Use for tasks like adding two numbers.

Anonymous Functions

```
<?php
$add = function ($a, $b) {
return $a + $b;
};
$result = $add(3, 5);
// Call the anonymous function
echo $result;
?>
```

```
<?php
$message = "Hello from Global scope!";
$greet = function ($name) use ($message) {
echo "$message $name";
};
$greet("Class"); // Output: Hello from parent
scope! Alice?>
```

Date and Time functions

#current date and time in Nepal

```
<?php
// Set the timezone to Nepal Standard Time
date_default_timezone_set('Asia/Kathmandu');

// Get the current time in Nepal
$currentTimeInNepal = date('Y-m-d H:i:s');

echo "Current time in Nepal is: $currentTimeInNepal";

?>
```

#current date

```
<?php
echo "Today is " . date("Y/m/d");
?>
```

```
<?php
$currentDate = date("Y-m-d H:i:s");
echo $currentDate;
?>
```

#Retrieves the date/time information as an associative array**.**

```
<?php
$dateInfo = getdate();
echo "Current day: " . $dateInfo['weekday'];
?>
```

Date and Time Functions

Example: Time Zone Handling

```
<?php
date_default_timezone_set('America/New_York');
$nyTime = date("Y-m-d H:i:s");
echo " York time: $nyTime";
?>
```

Date and Time Functions

Example: Time Zone Handling

```
<?php
date_default_timezone_set(‘Asia/Kathmandu');
$ktmTime = date("Y-m-d H:i:s");
echo " Kathmandu time: $ktmTime";
?>
```

- Functions

References

- Kevin Tatroe , Peter MacIntyre, Programming PHP: Creating Dynamic Web Pages, O′Reilly, 2021