Tishk International University
IT Department
Course Code: IT 349/A

Web Programming

Week #4
Control statements & functions

Fall 2024
Hemin Ibrahim, PhD
hemin.ibrahim@tiu.edu.iq

Overview

- Control Structures
- Loops
- Arrays
- Functions

Objectives

- Understand and apply control structures to manage the flow of code execution and enable decision-making in PHP applications.
- Use loops to efficiently execute repetitive tasks and process data sets within PHP code.
- Master the use of arrays to store, organize, and manipulate collections of data in PHP.
- Develop modular and reusable code by creating functions to encapsulate specific tasks within PHP applications.

Control Structures

- Control flow statements are used to control the flow of execution of statements.
- You can use control flow statements in your programs to conditionally execute
- statements, to repeatedly execute a block of statements.
- Generally, a program is executed sequentially, line by line, and a control structure
- allows you to alter that flow, usually depending on certain conditions.

if statement

- The if keyword is used to check if an expression is true.
- If it is true, a statement is then executed.
- The statement can be a single statement or a compound statement.
- A compound statement consists of multiple statements enclosed by curly brackets.

```php
<?php
    if (condition){
     //code will be executed if specified condition is true
    }
?>
```

If.. elseif.. else statement

```php
<?php
    if (condition){
     //code will be executed if specified condition is true
    }
?>
```

```php
<?php
    if (condition) {
     // code to execute if condition is true
     } elseif (another_condition) {
         // code to execute if another condition is true
     } else {
         // code to execute if all conditions are false
     }
?>
```

if statement - Examples

```php
<?php
    $d = date("D");
    if ($d == "Fri")
     echo "Have a nice weekend!";
?>
```

```php
<?php
    $score = 85;

    if ($score >= 90) {
        echo "Grade: A";
    } elseif ($score >= 80) {
        echo "Grade: B";
    } else {
        echo "Grade: C";
    }
?>
```

Switch

```php
<?php
    switch (variable) {
     case value1:
         // code to execute if variable == value1
         break;
     case value2:
         // code to execute if variable == value2
         break;
     default:
         // code to execute if variable doesn't match any case
     }
?>
```

Switch - Example

```php
<?php
    $d = date("D");
    switch ($d){
    case "Mon":
     echo "Today is Monday";
     break;
    case "Tue":
     echo "Today is Tuesday";
     break;
    case "Wed":
     echo "Today is Wednesday";
     break;
    case "Thu":
     echo "Today is Thursday";
     break;
    case "Fri":
     echo "Today is Friday";
     break;
    case "Sat":
     echo "Today is Saturday";
     break;
    case "Sun":
     echo "Today is Sunday";
     break;
    default:
     echo "Wonder which day is this ?";
    }
?>
```

Loops

- In programming, loops allow you to execute a block of code multiple times. PHP provides several types of loops, each suited for different use cases. Loops are especially useful when working with arrays, databases, and other situations where repetitive tasks are common.

Types of Loops in PHP

PHP supports the following loop structures:

- **for Loop** - Repeats a block of code a specified number of times.
- **while Loop** - Repeats a block of code as long as a condition is true.
- **do...while Loop** - Similar to while, but guarantees the code runs at least once.
- **foreach Loop** - Specially designed for iterating over arrays.

for loop

The for loop is ideal when you know beforehand how many times you need to iterate. It consists of three parts: **initialization**, **condition**, and **increment/decrement.**

```php
<?php
    for ($i = 0; $i < 5; $i++) {
        echo "Iteration: $i\n";
    }
?>
```

while loop

The while loop is useful when the number of iterations is unknown and depends on a condition. It checks the condition before each iteration.

```php
<?php
    $count = 1;
    while ($count <= 5) {
        echo "Count: $count\n";
        $count++;
    }
?>
```

do ... while loop

The do...while loop is similar to the while loop, but it executes the code block at least once, regardless of the condition. The condition is checked after the loop executes.

```php
<?php
    $count = 1;
    do {
        echo "Count: $count\n";
        $count++;
    } while ($count <= 5);
?>
```

foreach loop

The foreach loop is specially designed to iterate over arrays, making it the most efficient choice for array manipulation.

```php
<?php
    foreach ($array as $key => $value) {
        // Code to execute
    }
?>
```

```php
<?php

    $fruits = ["Apple", "Banana", "Cherry"];
    foreach ($fruits as $fruit) {
        echo $fruit . " ";
    }
?>
```

Arrays

- Arrays are one of the most fundamental data structures in PHP, allowing you to store multiple values in a single variable.
- In PHP, arrays are flexible, allowing a **mix of data types** and associative key-value pairs.

Types of Arrays in PHP

- **Indexed Arrays**: Arrays with numeric indices.
- **Associative Arrays**: Arrays with named keys.
- **Multidimensional Arrays**: Arrays containing one or more arrays as elements.

Indexed Arrays

- An indexed array uses numeric indices to store values. PHP automatically assigns indices starting from 0.

```php
<?php
    $myArray1 = array("value1", "value2", "value3");
    $myArray2 = ["value1", "value2", "value3"];
?>
```

Indexed Arrays

- or the index can be assigned manually

```php
<?php
    $fruits = array();
    $fruits[0] = "Mango";
    $fruits[1] = "Apple";
    $fruits[2] = "Banana";
    $fruits[3] = "Orange";
?>
```

Indexed Arrays

- Printing array values

```php
<?php
    $cars = array("Volvo", "BMW", "Toyota");
    echo "I like " . $cars[1] . " more than " . $cars[2] . ".";

?>
```

```php
<?php
    $fruits = ["apple", "banana", "cherry", "orange"];
   print_r($fruits);
   // Array ( [0] => apple [1] => banana [2] => cherry [3] => orange )
?>
```

Loop Through an Indexed Array

- To loop through and print all the values of an indexed array, you could use a for loop, like this:

```php
<?php
    $cars = array("Volvo", "BMW", "Toyota");
    $length = count($cars);
    for($i = 0; $i < $length; $i++) {
        echo $cars[$i];
        echo "<br>";
    }
?>
```

Associative Arrays

- Associative arrays use named keys that you assign to each value, making it easier to access data by name rather than by numeric index.

$array = array("key1" => "value1", "key2" => "value2");

```php
<?php
    $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
    // Second way
    $age["Peter"] = "35";
    $age["Ben"] = "37";
    $age["Joe"] = "43";
?>
```

Associative Arrays

- Example

```php
<?php
    $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
    echo "Ben is " . $age['Ben'] . " years old.";
?>
```

Loop Through an Associative Array

- To loop through and print all the values of an associative array, you could use a foreach loop, like this:

```php
<?php
    $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
    foreach($age as $key => $value) {
        echo "Key=" . $key . ", Value=" . $value;
        echo "<br>";
    }
?>
```

Array Functions in PHP

- The count() function is used to return the length (the number of elements) of an array:

```php
<?php
    $numbers = [1, 2, 3, 4, 5];
    echo count($numbers); // Outputs: 5
?>
```

Array Functions in PHP

- in_array(): searches an array for a specific value.

```php
<?php
    $fruits = ["apple", "banana", "cherry", "orange"];
    if (in_array("orange", $fruits)) {
        echo "Orange is in the array";
    } else {
        echo "Orange is not in the array";
    }
?>
```

Array Functions in PHP

- shuffle() Function: The shuffle() function randomizes the order of the elements in the array.

This function assigns new indices for the elements in the array. Existing indices will be removed (See Example below).

```php
<?php
    $fruits = ["apple", "banana", "cherry", "orange"];
    shuffle($fruits);
?>
```

Array Functions in PHP

- shuffle() Function: Example

```php
<?php
    $fruits = ["apple", "banana", "cherry", "orange"];
    print_r($fruits);
    echo "<br>";
    shuffle($fruits);
    print_r($fruits);
?>
```

```
Array ( [0] => apple [1] => banana [2] => cherry [3] => orange )
Array ( [0] => orange [1] => apple [2] => cherry [3] => banana )
```

Array Functions in PHP

- implode(): The implode() function returns a string from the elements of an array.

```php
<?php
    $fruits = ["apple", "banana", "cherry", "orange"];
    $texts = implode(",",$fruits);
    echo $texts; //apple,banana,cherry,orange


?>
```

Array Functions in PHP

- explode(): The explode() function breaks a string into an array.

```php
<?php
    $names = "Ali Aram Kani Kurdistan Milan Shan";
    $myArray = explode(" ", $names);
    echo $myArray[0]; // Outputs: Ali
    echo "<br>";
    echo $myArray[3]; // Outputs: Kurdistan


?>
```

Array Functions in PHP

- array_merge(): allows you to append one array into another. Think of it as concatenation for arrays.

```php
<?php
    $names1 = array("Kardo", "Azad", "Hama");
    $names2 = ["Hardi", "Zara", "Bestun"];
    $names = array_merge($names1, $names2);
?>
```

Deleting Array Elements

If you want to delete an element from an array you can simply use the **unset()** function. The following example shows how to delete an element from an associative array and indexed array.

```php
<?php
    $names1 = array("Kardo", "Azad", "Hama","Hardi", "Zara", "Bestun");
    unset($names1[1]);
?>
```

Array ([0] => Kardo [2] => Hama [3] => Hardi [4] => Zara [5] => Bestun)

?

Deleting Array Elements

If you see the above example carefully you will find that the unset() function didn't reindex the array after deleting the value from the indexed array. To fix this you can use the **array_splice()** function. It takes three parameters: an array, offset (where to start), and length (number of elements to be removed). Let's see how it actually works:

```php
<?php
    $names1 = array("Kardo", "Azad", "Hama","Hardi", "Zara", "Bestun");
    array_splice($names1, 3,1);
?>
```

PHP Sorting Arrays

PHP comes with a number of built-in functions designed specifically for sorting array elements in different ways like alphabetically or numerically in ascending or descending order. Here we'll explore some of these functions most commonly used for sorting arrays.

- sort() and **r**sort() — For sorting **indexed** arrays.
- **a**sort() and **ar**sort() — For sorting **associative** arrays by **value**.
- **k**sort() and **kr**sort() — For sorting **associative** arrays by **key**.

r = **reverse**
a = **associative**
k = **key**

sort() and rsort()

```php
<?php

    $names = array("Aram", "Kareem", "Zana", "Bahjat");
    sort($names);
    print_r($names);
    // Array ( [0] => Aram [1] => Bahjat [2] => Kareem [3] => Zana )

?>
```

```php
<?php

    $names = array("Aram", "Kareem", "Zana", "Bahjat");
    rsort($names);
    print_r($names);
    // Array ( [0] => Zana [1] => Kareem [2] => Bahjat [3] => Aram )

?>
```

asort() and ksort()

```php
<?php

    $age = array("zana" => "55", "kareem" => "17", "bahjat" => "43", "aram" => "40");
    asort($age);
    print_r($age);
    // Array ( [kareem] => 17 [aram] => 40 [bahjat] => 43 [zana] => 55 )


?>
```

```php
<?php

    $age = array("zana" => "55", "kareem" => "17", "bahjat" => "43", "aram" => "40");
    ksort($age);
    print_r($age); // Array ( [aram] => 40 [bahjat] => 43 [kareem] => 17 [zana] => 55 )

?>
```

Function

- In PHP, a function is a block of code that performs a specific task. Functions help you organize and reuse code by encapsulating logic in a single place, making your code more modular, readable, and maintainable. You can call a function multiple times within your code without rewriting the logic.

```php
<?php

    function functionName() {
        // Code to execute
    }

?>
```

Function - example

```php
<?php

    function greet() {
        echo "Hello, World!";
    }

    greet();
?>
```

```php
<?php

    function greet($name) {
        echo "Hello, $name!";
    }

    greet("Kardo"); // Outputs: Hello, Kardo!
?>
```

```php
<?php
Click to collapse the range.

    function add($a, $b) {
        return $a + $b;
    }

    $result = add(5, 10);
    echo $result; // Outputs: 15
?>
```

Thank You