Unit 3: Array and Function

Presented By:
Bipin Maharjan

Table of Contents

What is an Array?

- An array can store one or more values in a single variable name.
- Made up of elements
- Each element in the array is assigned its own ID so that it can be easily accessed.
- Each element has key and value
- Element value can be strings, numbers, true or false or can be other array
- Arrays are collections of related values
    - The data submitted from a form
    - The names of students in a class
    - The populations of a list of cities

3 Kinds of Arrays

- Indexed Array
  - Arrays with numeric index
- Associative Array
  - Arrays with named keys
- Multidimensional Array
  - Arrays containing one or more arrays

Indexed Array

- A indexed array stored each element with a numeric ID key.
- 3 ways to write a indexed array.

1st:

```
$fruits = array('apple', 'banana', 'orange');
```

2nd:

```
$names = ['John', 'Jane', 'Bob'];
```

3rd:

```
$primes[0] = 2;

$primes[1] = 3;

$primes[2] = 5;

$primes[3] = 7;
```

Indexed Array: Example

```
<?php

$names = ['John', 'Jane', 'Bob'];


echo "The names are: $names[0], $names[1],
$names[2]";


?>
```

Output:

```
The names are: John, Jane, Bob
```

Associative Array

- An associative array also is a type of array by which you can assign an arbitrary key to every value.
- In an associative array, each key is associated with a value.
- With associative arrays we can use the values as keys and assign values to them.
- In an associative array, the keys are not necessarily numeric, and even when they are numeric, not necessarily in any order.

Associative Array Contd…

- There are three ways for defining the associative array in PHP. Which are as follows:

1st:

```
$ages = array("John" => 35, "Jane" => 30, "Bob" => 25);
```

2nd:

```
$ages = ["John" => 35, "Jane" => 30, "Bob" => 25];
```

3rd:

```
$ages["John"] = 36;

$ages["Jane"] = 31;

$ages["Bob"] = 26;
```

Associative Array: Example

```
<?php

$ages = array("John" => 35, "Jane" => 30, "Bob"
=> 25);

echo "The ages are: " . $ages["John"] . ", " .
$ages["Jane"] . ", " . $ages["Bob"];
```

Output:

```
The ages are: 35, 30, 25
```

Multi-dimensional Array

- A multi-dimensional array is also known as two-dimensional array.
- A multi-dimensional array is an array of arrays i.e. each element in the main array can also be an array.
- And each element in the sub-array can be an array, and so on.
- If an array element value is another array then this is a multidimensional array.

Syntax

```
$arrayName = array ($array1, $array2, $array3);
```

Multi-dimensional Array: Example

```
<?php

$family = array(

    'father' => array('name' => 'John', 'age' => 35),

    'mother' => array('name' => 'Jane', 'age' => 30),

    'son' => array('name' => 'Bob', 'age' => 25)

);

echo "The family is: " . $family['father']['name'] . ", " .
$family['mother']['name'] . ", " . $family['son']['name'];

?>
```

Output:

```
The family is: John, Jane, Bob
```

Array Iteration

- Iteration = looping through array elements.
- PHP provides different ways to access and process all items in an array.
- Common methods:
  - for loop
  - foreach loop

Iterating with for loop

- Works best with indexed arrays.
- Use count() to find the array length.

Example:

```
$fruits = array("Apple", "Banana", "Mango");
for ($i = 0; $i < count($fruits); $i++) {
   echo $fruits[$i] . "<br>";
}
```

Output:

```
Apple, Banana, Mango
```

Iterating with foreach loop

- Easiest and most commonly used.
- Works with indexed & associative arrays.

Indexed Array Example:

```
$colors = array("Red", "Green", "Blue");

foreach ($colors as $c) {

   echo $c . "<br>";

}
```

Iterating with foreach loop: Associative Array

```
$marks = array("Ram"=>85, "Sita"=>90, "Hari"=>78);

foreach ($marks as $name => $score) {

   echo "$name : $score <br>";

}
```

PHP Built in Function

- PHP provides large number of predefined functions to perform common tasks that can be called directly.
- Saves time and reduces the need to write custom code.
- Categories:
    - String Function
    - Math Function
    - Date & Time Function
    - Array Function

String Function

https://www.php.net/manual/en/ref.strings.php

Function	**What It Does**
strlen()	Returns the length of a string
strrev()	Reverses a string
strtoupper()	Converts string to uppercase
strtolower()	Converts string to lowercase
ucfirst()	Converts first character to uppercase
ucwords()	Converts first character of each word to uppercase
substr()	Extracts part of a string
str_replace()	Replaces some characters with others in a string
strpos()	Finds the position of the first occurrence of a substring
trim()	Removes whitespace (spaces, tabs, newlines) from both ends

Math Function

https://www.php.net/manual/en/ref.strings.php

Function	What It Does
abs()	Returns absolute value
pow(x, y)	Returns x raised to the power y
sqrt()	Returns square root
round()	Rounds number to nearest integer
ceil()	Rounds number up
floor()	Rounds number down
max()	Returns largest number
min()	Returns smallest number
rand(min, max)	Returns random number between min and max
number_format()	Formats a number with grouped thousands

Data and Time Function

https://www.php.net/manual/en/ref.datetime.php

Function	**What It Does**
date("Y-m-d")	Returns current date
date("h:i:sa")	Returns current time (12-hour format with am/pm)
date("H:i:s")	Returns current time (24-hour format)
time()	Returns current Unix timestamp
mktime()	Returns Unix timestamp for a given date
strtotime()	Converts a human-readable date string into timestamp
date("l")	Returns day of the week (e.g., Monday)
date("F")	Returns month name (e.g., August)

Common Date Format Characters

- Y = 4-digit year (2025)
- y = 2-digit year (25)
- m = Month with leading zero (01-12)
- n = Month without leading zero (1-12)
- M = Short month name (Jan-Dec)
- F = Full month name (January-December)
- d = Day with leading zero (01-31)
- j = Day without leading zero (1-31)
- l = Full day name (Monday-Sunday)
- D = Short day name (Mon-Sun)
- h = 12-hour format (01-12)
- H = 24-hour format (00-23)
- i = Minutes (00-59)
- s = Seconds (00-59)
- A = AM/PM

Array Function

https://www.php.net/manual/en/ref.array.php

Function	**What It Does**
explode()	Splits a string into array elements
implode()	Joins array elements into a string
range()	Generates a number range as an array
min()	Finds the smallest value in an array
max()	Finds the largest value in an array
shuffle()	Randomly rearranges the sequence of elements in an array

Array Function Contd…

Function	**What It Does**
array_slice()	Extracts a segment of an array
array_shift()	Removes an element from the beginning of an array
array_unshift()	Adds an element to the beginning of an array
array_pop()	Removes an element from the end of an array
array_push()	Adds an element to the end of an array
array_unique()	Removes duplicate elements from an array
array_reverse()	Reverses the sequence of elements in an array
array_merge()	Combines two or more arrays

Array Function Contd…

Function	**What It Does**
array_intersect()	Calculates the common elements between two or more arrays
array_diff()	Calculates the difference between two arrays
in_array()	Check if a particular value exists in an array
array_key_exists()	Checks if a particular key exists in an array
sort()	Sorts an array
asort()	Sorts an associative array by value
ksort()	Sorts an associative array by key
rsort()	Reverse-sorts an array
krsort()	Reverse-sorts an associative array by value

User Defined Function

- A function is a block of statements that performs a specific task.
- A function will not execute automatically when a page loads.
- It can be used repeatedly in a program.
- A function will be executed by a call to the function.

User Defined Function: Syntax

```
//define a function
function myfunction($arg1, arg2, … $argn)
{
        statement 1;
        statement 2;
        …
        …
        return $val;
}
// call function
$result = myfunction($arg1, $arg2, … $argn);
```

User Defined Function Contd…

Function may be defined with optional but any number of arguments. However, same number of arguments must be provided while calling. Function’s body can contain any valid PHP code i.e. conditionals, loops etc. (even other functions or classes may be defined inside a function). After executing statements in the block, program control goes back to the location from which it was invoked irrespective of presence of last statement of function block as return. An expression in front of return statement returns its value to calling environment.

Function: Basic Example

```
<?php
//function definition
function sayHello()
{
   echo "Hello World!";
}

//function call
sayHello();
?>
```

Output:

```
Hello World!
```

Function: With Arguments

```
<?php

// function with arguments

function add($a, $b) {

    echo $a + $b . "<br>";

}


add(10, 20);

add("Hello ", "World!");

?>
```

Output:

```
30

Fatal error: Uncaught TypeError: Unsupported
operand types: string + string in …
```

Function:Return

```
<?php
// function with return value
function getSum($a, $b) {
    return $a + $b;
}
$value = getSum(10, 20);
echo "Sum: $value<br>";
$value = getSum("10", "20");
echo "Sum: $value<br>";
?>
```

Output:

```
Sum: 30
Sum: 30
```

Function: With Default Argument

```
function welcome($name = "Guest") {

    echo "Welcome, $name! <br>";

}

//uses default

welcome();

//overrides default

welcome("John");

?>
```

Output:

```
Welcome, Guest!

Welcome, John!
```

In first call, function is called without passing value. In this case, user argument takes its default value.

Function: With Variable Number of Arguments

It is possible to define a function with ability to receive variable number of arguments. The name of formal argument in function definition is prefixed by … token.

```
<?php
function sum(...$numbers) {
    $sum = 0;
    foreach ($numbers as $number) {
        $sum += $number;
    }
    return $sum;
}
$total = sum(1, 2, 3);
echo "Sum of 1, 2, 3: $total<br>";
echo "Sum of 1, 2, 3, 4: " . sum(1, 2, 3, 4) .
"<br>";
```

Output:

```
Sum of 1, 2, 3: 6

Sum of 1, 2, 3, 4: 10
```

Function: With Variable Number of Arguments

It is also possible to obtain a list of arguments passed to a function with the help of **func_get_args()** function. We can run a PHP loop to traverse each value in the list of arguments passed. In that case the function definition doesn't have a formal argument.

```
function sum() {
    $numbers = func_get_args();
    $sum = 0;
    foreach ($numbers as $number) {
        $sum += $number;
    }
    return $sum;
}
$total = sum(1, 2, 3);
echo "Sum of 1, 2, 3: $total<br>";
```

Output:

```
Sum of 1, 2, 3: 6
```

Function: With Another Function

A function may be defined inside another function’s body block. However, inner function can not be called before outer function has been invoked.

```
<?php
function hello()
{
    echo "Hello World! <br>";
    function greet()
    {
        echo "Welcome to the world of functions!
<br>";
    }
}

//greet();
hello();
greet();
```

Output:

```
Hello World!

Welcome to the world of functions!
```

Remove the comment to call greet() before hello(). Following error message halts the program - Fatal error: Uncaught Error: Call to undefined function hello()

Recursive function

A function that calls itself is called recursive function. Calling itself unconditionally creates infinite loop and results in out of memory error because of stack full. Following program calls factorial() function recursively.

```php
<?php
function factorial($n)
{
    if ($n == 0) {
        return 1;
    } else {
        return $n * factorial($n - 1);
    }
}

echo "Factorial of 5: " . factorial(5) . "<br>";
```

Output:

```
Factorial of 5: 120
```

Variable Scoping

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 Variable Scope

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

```
$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>";
```

Local Variable Scope

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

```
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>";
```

Static Variable Scope

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:

```
function myTest() {
  static $x = 0;
  echo $x;
  $x++;
}
myTest();
myTest();
myTest();
```

Super Global Variables

Some predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

The PHP superglobal variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

Any Questions?