

# Loops

Looping structures in PHP allow us to execute a block of code repeatedly as long as a certain condition is met.

They enable us to efficiently perform repetitive tasks, iterate through data structures, and automate processes in our code.

- **for**
- **while**
- **do-while**
- **foreach**
  
- **break**
- **continue**

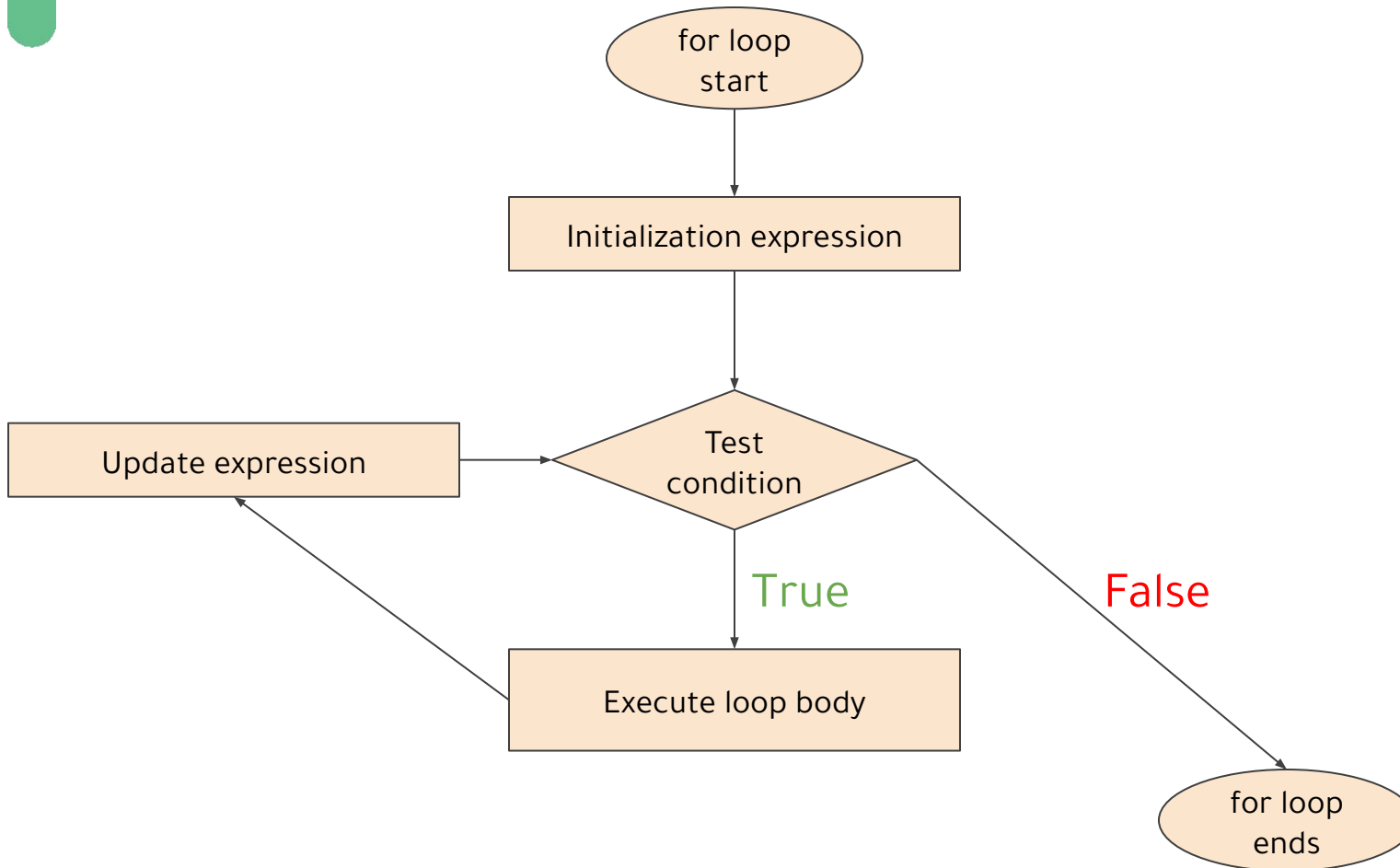
# for Loop

The for loop is used to execute a block of code repeatedly for a specific number of times. It consists of three parts: initialization, condition, and increment/decrement, which control the loop's execution.

```
<?php

for ($i = 0; $i < 5; $i++) {
    echo "Iteration: " . $i . "<br>";
}
```

# for Loop



# while Loop

The while loop is used to execute a block of code repeatedly as long as a specified condition is true. It continues to execute until the condition becomes false.

```
<?php

$i = 0;

while ($i < 5) {
    echo "Iteration: " . $i . "<br>";
    $i++;
}
```

# do-while Loop

The do-while loop is similar to the **while** loop, but it executes the code block at least once, even if the condition is initially false. It then continues to execute as long as the condition remains true.

```
<?php

$i = 0;

do {
    echo "Iteration: " . $i . "<br>";
    $i++;
} while ($i < 5);
```

# foreach Loop

The foreach loop is used to iterate over arrays or other iterable objects. It allows you to access each element without the need for an index or counter.

```
<?php

$fruits = ["Apple", "Banana", "Orange"];

foreach ($fruits as $key=>$fruit) {
    echo $fruit . "<br>";
}
```

# break Statement

The break statement is used to exit a loop prematurely. It allows you to terminate the execution of a loop when a certain condition is met or a specific point is reached. It is commonly used to stop a loop iteration when a desired outcome is achieved.

```
<?php

for ($i = 0; $i < 5; $i++) {
    if ($i == 3) {
        break;
    }
    echo "Iteration: " . $i . "<br>";
}
```

# continue Statement

The continue statement is used to skip the rest of the current iteration and move on to the next iteration of a loop. It allows you to bypass specific code within a loop while continuing the loop's execution.

```
<?php

for ($i = 0; $i < 5; $i++) {
    if ($i == 2) {
        continue;
    }
    echo "Iteration: " . $i . "<br>";
}
```



# Question?

02

**Which of the following is a valid variable name in PHP?**

- a) 1variable
- b) variable\_1
- c) variable-1
- d) &variable

# Question?

03

What is the difference between `print()` and `echo()`?

- `print()` can be used as part of an expression, while `echo()` can't
- `echo()` can be used as part of an expression, while `print()` can't
- `echo()` can be used in the CLI version of PHP, while `print()` can't
- `print()` can be used in the CLI version of PHP, while `echo()` can't
- There's no difference: both functions print out some text!



# Any Questions?



# Arrays

Arrays are a fundamental data structure in PHP that allow you to store multiple values in a single variable. They can hold various types of data, including numbers, strings, and even other arrays.

# Array Types

## Numeric/Indexed Array

0	1	2	3	4
Apple	Banana	Orange	Strawberry	Grape

## Associative Array

name	email	phone	address
John Doe	john@example.com	0912223333	street 123

## Multi-dimensional Array

	0	1	2	3
0	Math	40	40	80
1	Science	50	44	94
2	Geography	35	42	77

# Declaring and Referencing Arrays

Arrays in PHP can be created using different methods. The most common ways are:

- Using the **array()** function
- Using square brackets **[]**

```
<?php
// numeric index, auto assigned key
$arr = array(10, 'abc', 30);
// numeric index, key explicitly set
$arr = array(0 => 10, 1 => 'abc', 2 => 30 );
// associative
$arr = array('name' => 'foo', 'age' => 20);
// short syntax
$arr = ['name' => 'foo', 'age' => 20];
```

# Accessing Array Elements

- Array elements can be accessed using their respective index values. In PHP, array indices start at **0** or key name.
- PHP array keys are case sensitive: **\$arr['A']** and **\$arr['a']** are different elements. Keys may only be a string or an integer. Other variable types are cast into one of these
- If you do not specify a key then PHP will assign an auto-incrementing numeric key.

```
<?php
$arr = array(10, 'abc', 30);
echo $arr[0]; // 10
echo $arr[1]; // abc
echo $arr[2]; // 30

$arr2 = array('foo' => 'value', 'bar' => 'value2');
echo $arr2['foo']; // value
echo $arr2['bar']; // value2
```

# Modifying Array Elements

Array elements can be modified by assigning a new value to a specific index.

```
<?php
$numericArray = array(10, 20, 30, 40, 50);
$shortNumericArray = [10, 20, 30, 40, 50];
$associativeNumericArray = ['a' => 10, 'b' => 20, 'c' => 30, 'd' => 40];
$nestedArray = ['10' => ['b' => 10, 'c' => 20], '20' => ['e' => 30, 'f' =>
40]];

// Accessing elements
$firstElement = $numericArray[0];
$thirdElement = $numericArray[2];

// Modifying elements
$numericArray[1] = 25; // Modify the second element to be 25
$numericArray[4] += 10; // Increment the fifth element by 10

// Modifying array
$associativeNumericArray['c'] = 50; // Modify element

// Modifying nested array
$nestedArray['20']['f'] = 50; // Modify nested element
$nestedArray['30'] = ['g' => 60]; // Add new element
```



# Comparing Arrays

It is possible to use the equality `==` and identity `===` operators to compare arrays. When applied to arrays, the equality operator returns true if the arrays have the same keys and values, regardless of their type. The identity operator will only return true if the arrays have the same keys and values, they are in the same order, and they are of the same variable types.

```
<?php
$arr = ['1', '2', '3'];
$brr = [1, 2, 3];
var_dump($arr === $brr); // false
var_dump($arr == $brr); // true
```

# Example Usage of Array Functions

Here's an example demonstrating the usage of array functions:

```
<?php
$numbers = [1, 2, 3, 4, 5];
echo "Total elements: " . count($numbers) . "<br>";
array_push($numbers, 6, 7); // Add 6 and 7 to the end of the array
$lastElement = array_pop($numbers); // Remove the last element
echo "New array: " . implode(", ", $numbers) . "<br>"; // Output: Last array: 1,
2, 3, 4, 5
echo "Last element: " . $lastElement . "<br>"; // Output: Last element: 7
$mergedArray = array_merge($numbers, [8, 9]); // Merge 8 and 9 to the end of the
array
$slicedArray = array_slice($mergedArray, 2, 4); // Slice the array from index 2 to
index 4
$key = array_search(4, $slicedArray); // Find the key of the value 4
echo "Merged array: " . implode(", ", $mergedArray) . "<br>"; // Output: Merged
array: 1, 2, 8, 9
echo "Sliced array: " . implode(", ", $slicedArray) . "<br>"; // Output: Sliced
array: 8, 9
echo "Key of value 4: " . $key . "<br>"; // Output: Key of value 4: 1
```

# Array Functions

PHP provides a rich set of built-in array functions to perform various operations on arrays. These functions make it easier to manipulate and work with array data.

## Common Array Functions

Let's explore some commonly used array functions:

Function	Details
<b>count</b>	Returns the number of elements in an array
<b>array_push</b>	Adds one or more elements to the end of an array
<b>array_pop</b>	Removes and returns the last element of an array
<b>array_merge</b>	Merges two or more arrays into a single array
<b>array_search</b>	Searches for a value in an array and returns the corresponding key
<b>array_sum</b>	Calculate the sum of values in an array

# Array Functions (2)

Function	Details
<b>array_keys</b>	Returns an array containing the keys.
<b>array_values</b>	Returns an array containing the values.
<b>in_array</b>	Checks if a value exists in an array

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

# Question?

03

Are php keys case-sensitive? What will the output of this script be?

```
<?php
$arr1 = ["A" => "apple", "B" => "banana"];
$arr2 = ["a" => "aardvark", "b" => "baboon"];
echo count($arr1 + $arr2);
```

- this produces an error
- 2
- 4
- None of the above

# Question?

04

What will this script output?

```
<?php
$a = array('one', 'two');
$b = array('three', 'four', 'five');
echo count($a + $b);
```

- this produces an error
- 2
- 3
- 5

# Exercise

10 Minutes!

## 01

1. Write a PHP script to display information to 3 students.
2. Print the number of students
3. Use a loop to iterate through the array and print the details for each student.

If we want to print average ages, who can do it?

```
<?php
$students = [
    [
        'name' => '...',
        'age' => 20,
        'grade' => 'A',
        'subjects' => ['Math', 'Science', 'English']
    ],
];
```



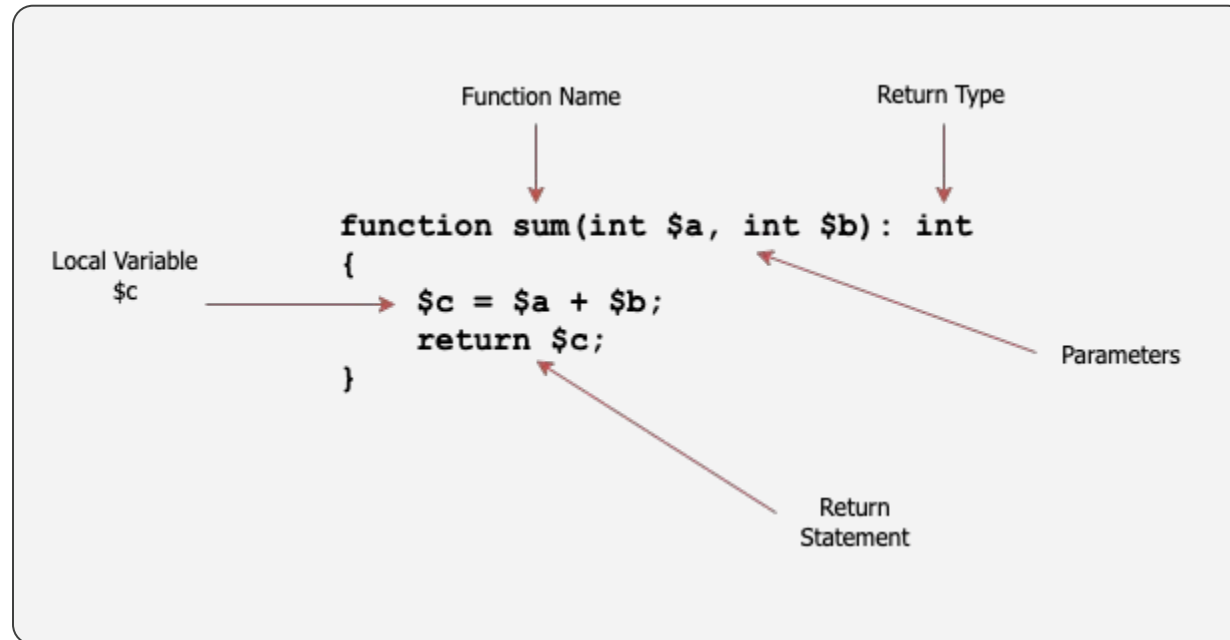
# Functions

Functions are packages of code that can be used to execute a sequence of instructions. Any valid code can be used inside a function, including calls to other functions

Functions in PHP are defined using the **function** keyword, followed by the function name and a pair of parentheses. The code block within curly braces {} contains the instructions executed when the function is called.



# Functions



# Function Arguments

Arguments to a function, also known as parameters, allow you to pass values into the function scope. Arguments are passed as a comma-separated list and are evaluated left to right.

```
<?php
function functionName(string $itemName, array $details) {
    // Function body
}
```

# Function Arguments: Alternate Null Type Syntax

You can prefix the type hint with a question mark to indicate that the variable may either be null or of the specified type. Here's an example:

```
<?php
function myFunc(?string $name) {
    echo "hello, ". $name;
}
myFunc(null);
```

# Function Arguments: Variadics

- The variadic parameters are made available in your function as an array.
- If you are mixing normal fixed parameters with a variadic syntax, then the variadic parameter must be the last parameter in the list of parameters..

```
<?php
function parameterTypeExample($required, $optional = null,
...$variadicParams) {
    echo 'required: ' . $required;
    echo 'optional: ' . $optional;
    echo 'variadicParams: ' . count($variadicParams);
}
```

# Function Arguments: References

By default, PHP passes arguments to functions by value, but it is possible to pass them by reference. You can do this by declaring the argument as a pass by reference, as in this example:

```
<?php
function addOne(&$arg) {
    $arg++;
}
$a = 0;
addOne($a);
echo $a; // 1
```

# Variable Functions

Variable functions are similar in concept to variable variable names. They're easiest to explain with a syntax example:

```
<?php

function foo() {
    echo 'Foo';
}

$var = 'foo';
$var(); // calls foo()
```

# Returns

Using the return statement will prevent further code from executing in your function. If you return from the root scope, then your program will terminate. PHP will return NULL if you do not specify a return value for your function using the return keyword.

```
<?php
function getNumber() {
    return 123;
}
$name = getFullName('Mary', 'Moon');
echo $name;
```

# Type Hinting

Type hinting allows you to specify the variable type that a parameter to a function is expected to be.

```
<?php
function printArray(array $arr) {
    echo "<pre>" . print_r($arr,true) . "</pre>";
}
```

If you pass a parameter of the wrong type, a **TypeError** exception is thrown.



# Returns:

## Return Type Declarations

We previously looked at how you can declare what variable type your function arguments will be. You can also specify what variable type the function will return.

```
<?php
function getName(string $firstName, string $lastName): string
{
    return $firstName . ' ' . $lastName;
}

$name = getName('Maharah', 'course');

echo $name; // Maharah course
```

# Returns:

## Return Void

If the function is going to return null, you can specify that it will return "void"

```
<?php
function sayHello(): void {
    echo "Hello World";
}
// Hello World
sayWorld();
```

# Variable Scope in Functions

As in other languages, the scope of a PHP variable is the context in which it was defined. PHP has three levels of scope—global, function, and class. Every time a function is called, a new function scope is created.

You can include global scope variables into your function in one of two ways:

```
<?php
$glob = "Global variable";
function myFunction() {
    global $glob;
    $glob = "Changed";
    // $GLOBALS['glob'] = "Changed";
    // echo $glob; // Also can access the global variable inside the function
}
myFunction();
echo $glob; // Changed
```

**Note:** Most coding standards strongly discourage global variables because they introduce problems when writing tests, can introduce weird context problems, and make debugging more difficult.

# Lambda and Closure

A lambda in PHP is an anonymous function that can be stored as a variable.

```
<?php
$lambda = function($a, $b) {
    echo $a + $b;
};
$lambda(1, 2); // 3
```

Another way of putting this is to say that the anonymous function “closes over” variables that are in the scope it was defined in.

In practical syntax in PHP, we define a closure like this:

```
<?php
$string = "Hello Maharah!";
$closure = function() use ($string) {
    echo $string;
};
$closure();
```

# Question?

05

What is the output of the following php script?

```
<?php
function getData($param, ...$extras, $param2) {
    echo $param;
}
getData(1,2,3,"cat");
```

- 1
- cat
- 2
- this produces an error

# Exercise

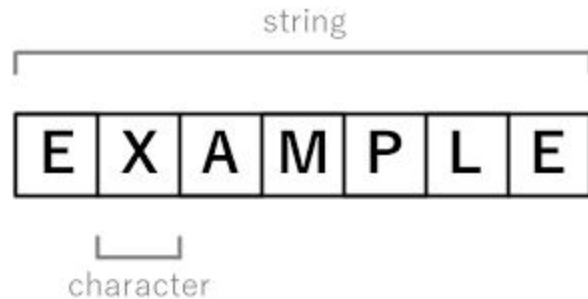
5 Minutes!

02

Create a function named sum that can take a variable number of arguments and returns their sum.

# Strings and Patterns

Strings are sequences of characters used to represent textual data in PHP. They can contain letters, numbers, symbols, and whitespace. PHP provides a variety of string functions that allow you to manipulate and format strings to suit your needs.



# Strings and Patterns

## Declaring Strings

In PHP, strings may be declared either as simple type or complex type. The difference is that complex strings will be evaluated with respect to control characters and variables. Simple strings are declared in 'single quote marks' while complex strings are declared in "double quote marks".

```
<?php
$name = 'Mohamed';
$a = 'Hello $name\n';
$b = "Hello $name\n";
echo $a; // Hello $name\n
echo $b; // Hello Mohamed
```

## Embedding Variables

```
<?php
$name = "Mohamed";
echo 'My name is $name';
echo 'My name is ' . $name;
echo "My name is $name";
echo "My name is {$name}";

// My name is $name
// My name is Mohamed
// My name is Mohamed
// My name is Mohamed
```



# Strings and Patterns

## String Concatenation

String concatenation is the process of combining two or more strings into a single string. In PHP, the concatenation operator is the dot (.) symbol.

```
<?php
$firstName = "Mohamed";
$lastName = "Ahmed";
$fullName = $firstName . " " . $lastName;
echo $fullName; // Output: Mohamed Ahmed
```

## Extracting Strings

An individual position in a string can be referenced with the same syntax as an array element. All positions in the string are always zero-based—the first character in the string is position 0.

```
<?php
$string = 'abcdef';
echo $string[0];
// a
```

# Strings and Patterns

## Referencing Characters in Strings

You can reference a position in a string by using either square brackets braces to denote the zero-based integer position you want to reference.

```
<?php
$hello = "world";
echo $hello[0]; // w
```

Writing to a position that is out of range will result in the string being padded with spaces to accommodate the missing section.

```
<?php
$hello = "world";
$hello[10] = "*";
echo $hello; // world   *
```

# Strings and Patterns

## String Length

String length refers to the number of characters in a string. PHP provides the **strlen()** function to determine the length of a string.

```
<?php
$text = "Hello, World!";
$length = strlen($text);
echo $length; // Output: 13
```

## String Case Conversion

PHP offers functions to convert strings between different case formats. **strtolower()** converts a string to lowercase, while **strtoupper()** converts it to uppercase.

```
<?php
$text = "Hello, World!";
$lowercase = strtolower($text);
$uppercase = strtoupper($text);
echo $lowercase; // Output: hello, world!
echo $uppercase; // Output: HELLO, WORLD!
```

# Strings and Patterns

## String Substring

Substring refers to extracting a portion of a string. PHP provides the **substr()** function to extract a substring based on a starting index and optional length.

```
<?php
$text = "Hello, World!";
$substring = substr($text, 7, 5);
echo $substring; // Output: World
```

## String Formatting

PHP offers various string formatting functions to modify the appearance of text. Examples include **trim()** to remove whitespace, **str\_replace()** to replace specific characters, and **sprintf()** to format strings based on placeholders.

```
<?php
$text = " Hello, World! ";
$trimmed = trim($text);
$replaced = str_replace("World", "PHP", $text);
$formatted = sprintf("Welcome, %s!", $trimmed);
echo $trimmed; // Output: Hello, World!
echo $replaced; // Output: Hello, PHP!
echo $formatted; // Output: Welcome, Hello, World!!
```

# Strings and Patterns

## Searching Strings

PHP search parameters have a \$haystack and we are searching for a \$needle. Compare the order of parameters used for **strpos()**

```
<?php
$str = 'abcdef';
echo strpos($str, 'c');
```

## Matching Strings

Comparing strings in PHP should be done with an appropriate level of care when you're trying to match different variable types.

PHP uses the ASCII value of the character to make the comparison. Lowercase letters have a higher ASCII value than capitals, so you can have the situation where lowercase letters are placed after capitals, like this:

```
<?php
$a = "PHP";
$b = "developer";
if ($a > $b) {
    echo "$a > $b";
} else {
    echo "$a < $b";
}
```

ASCII: <https://www.asciitable.com/>

# Strings and Patterns

## Replacing Strings

PHP has three functions for replacing strings.

**str\_replace()** and its case-insensitive version **str\_ireplace()** can be used for basic replacements.

```
<?php
echo str_replace('foo', 'goo', 'Delicious food'); // Delicious good
```

```
<?php
$bodytag = str_ireplace("%body%", "black", "<body text=%BODY%>");
echo $bodytag; // <body text=black>
```



# String Patterns: Regular Expressions

Strings are sequences of characters used to represent textual data in PHP. They can contain letters, numbers, symbols, and whitespace. PHP provides a variety of string functions that allow you to manipulate and format strings to suit your needs.

# Regular Expressions

## Regular Expression Syntax

Regular expressions, often abbreviated as regex, are powerful tools for pattern matching and manipulation of strings. They provide a concise and flexible way to search, validate, and replace specific patterns within text.

Regular expressions consist of a combination of characters and metacharacters that define a pattern to match.

**Delimiters:** enclose the regular expression pattern.

- Most common: / (forward slash)
- Others: #, ~, etc. (useful when pattern contains /)

**Pattern:** defines the search criteria.

- Literal characters match themselves.
- Meta-characters represent special meanings.



# Regular Expressions

## Meta-characters

Meta-characters are interpreted to have a meaning in the search pattern. They need to be escaped if you intend to have them as a literal part of the expression. They are listed in the following table.

### **Anchors:**

match positions within the string.

- ^: beginning of string
- \$: end of string

### **Quantifiers:**

specify how many times a character can appear.

- \*: zero or more
- +: one or more
- ?: zero or one

### **Character classes:**

match a set of characters.

- [abc]: any of a, b, or c
- [^abc]: any character except a, b, or c
- [a-z]: any lowercase letter

# Regular Expressions

## Pattern Matching

Regular expressions enable pattern matching within strings. By defining a pattern, you can search for specific sequences of characters, such as email addresses, phone numbers, or URLs.

```
<?php
$text = "Hello, my email is john@example.com.";
$pattern = "/\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b/";
preg_match($pattern, $text, $matches);
echo $matches[0]; // Output: john@example.com
```

# Regular Expressions

## Pattern Matching: Example

Here's a simple PHP regular expression that you can use to validate Libyan phone numbers:

```
<?php

$phoneNumber = '+218910000000';
$regex = '/^\+2182[0-9]{7}$/';

if (preg_match($regex, $phoneNumber)) {
    echo 'Valid Libyan phone number.';
} else {
    echo 'Invalid Libyan phone number.';
}
```

# Regular Expressions

## Getting All Matches

So far your expressions are returning just the first occurrence of the matching portion of a search string. Let's say that you want to find all of the matches in the string.

PCRE has a global modifier (more on those later), but PHP uses a separate function called `preg_match_all()` to return all matches.

```
<?php
$subject = "Welcome to <strong>Maharah</strong> <strong>html</strong> text";
$pattern = "/ <strong>([a-zA-Z]+)</strong>/is";
preg_match_all($pattern, $subject, $matches);

echo "<pre>";
print_r($matches[0] ?? []);
/*
Array
(
    [0] => Maharah
    [1] => html
)
*/
```

# Exercise

5 Minutes!

03

**Your goal is to create a PHP script that uses Rexp to check whether a specified phone number is compatible with the specified format for Libyana or Almadar mobile numbers?**

**Valid mobile numbers must have one of the following prefixes:  
091, 092, 093, or 094.**



# HTML Forms

HTML forms are used to collect user input through various form elements such as text fields, checkboxes, radio buttons, and dropdown menus. When a user submits a form, the data is sent to the server for processing.

# HTML Forms

Display form  
to get data:  
- Contacts  
- Registration

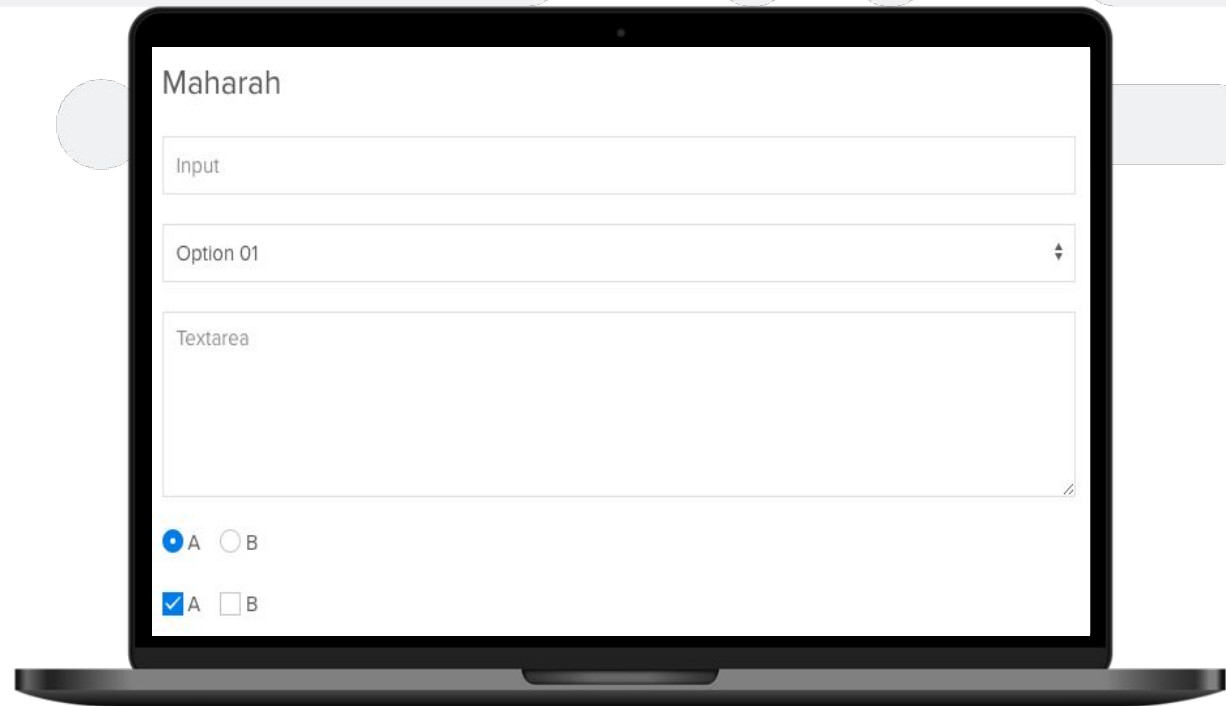
Submitted  
`$_POST`  
and/or `$_GET`  
variables

Process data:  
- Validate  
- Save to DB

# Form Data Handling Process

The handling process of form data using PHP involves the following steps:

1. Creating an HTML form with appropriate form elements and attributes.
2. Configuring the form action to point to a PHP script that will process the submitted data.
3. Retrieving the form data in the PHP script using the **\$\_POST** or **\$\_GET** superglobal arrays.
4. Validating and sanitizing the form data to ensure its integrity and security.
5. Performing any necessary business logic or database operations with the form data.
6. Displaying the processed data or redirecting the user to another page as required.



The image shows a laptop screen displaying a web form titled "Maharah". The form contains the following elements:

- An "Input" text field.
- A dropdown menu labeled "Option 01".
- A "Textarea" for multi-line text input.
- Two radio button options: "A" (selected) and "B".
- Two checkbox options: "A" (checked) and "B".



# HTML Forms

## Form Elements

Dots and space in form field names are converted to underscores. As an example, consider the HTML input tag:

```
<input name="email.address" type="text">
```

The value that it contains will be placed in either `$_GET['email_address']` or `$_POST['email_address']` depending on the forms method.

## Submit button

It is a fundamental element that triggers the submission of the form, enabling the server-side script to handle and process the provided information.

```
<input type="submit" value="Submit">
```

# HTML Forms

## Arrays in HTML Forms

One of the most useful ways that arrays help is in grouping inputs together. Consider a checkbox that can have multiple values:

```
<form action="formhandler.php" method="POST">
  <input type="text" name="name[first]">
  <input type="text" name="name[last]">
  <input type="submit">
</form>
```

This will result in `$_POST` or `$_GET` being an array that looks like this:

```
array(
  'name' => array(
    'first' => '',
    'last' => ''
  )
)
```

# HTML Forms

## Retrieving Form Data in PHP

PHP provides two superglobal arrays to retrieve form data:

- **\$\_POST**: Used for retrieving data sent via the HTTP POST method.
- **\$\_GET**: Used for retrieving data sent via the HTTP GET method or through query parameters in the URL.

```
<?php
$name = $_POST['name'];
$email = $_POST['email'];
```

### Debugging Form Data

To debug the **\$\_POST** data, you can use **print\_r(\$\_POST);** to inspect the data sent through the POST method. This PHP function prints a human-readable representation of the variable, allowing you to examine the structure and contents of the **\$\_POST** array during the debugging process.

```
<?php
print_r($_POST);
// Output
Array
(
    [name] => John Doe
    [email] => john@example.com
    [submit] => Submit
)
```

# HTML Forms

## Validating and Sanitizing Form Data

PHP provides various functions and techniques for form data validation and sanitization, such as:

- Using built-in functions like **filter\_var()** or **preg\_match()** for validating specific data types or patterns.
- Sanitizing data using functions like **filter\_var()** with the **FILTER\_SANITIZE\_\*** constants.

```
<?php
$name = $_POST['name'];
$email = $_POST['email'];
// Validate email format
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "Invalid email address.";
}
$sanitizedName = htmlspecialchars(strip_tags($name));
```

# Exercise

10 Minutes!

04

1. **Create an HTML form that allows users to input the name.**
2. **Create a PHP script to handle the submitted form. Retrieve the entered name**