PHP Arrays

- **Arrays** are a fundamental data structure in PHP.
- They allow you to store a collection of elements under a single variable name.
- Elements can be of various data types: strings, numbers, objects, or even other arrays.

Key Points:

- **Access:** Elements are accessed using an **index** (numeric for ordered lists) or a **key** (associative for named elements).

Ways to Create Arrays in PHP:

1. Using the `array()` function:

- This is the traditional method for creating arrays.
- You can specify key-value pairs or just values.
- Keys can be strings or integers.

2. Using the short array syntax (`[]`):

- A more concise alternative to the `array()` function.
- Use square brackets `[]` with comma-separated elements.

3. Using the `compact()` function (PHP 4+):

- Creates an associative array from variables.
- Variable names become keys, and their values are assigned.

Benefits:

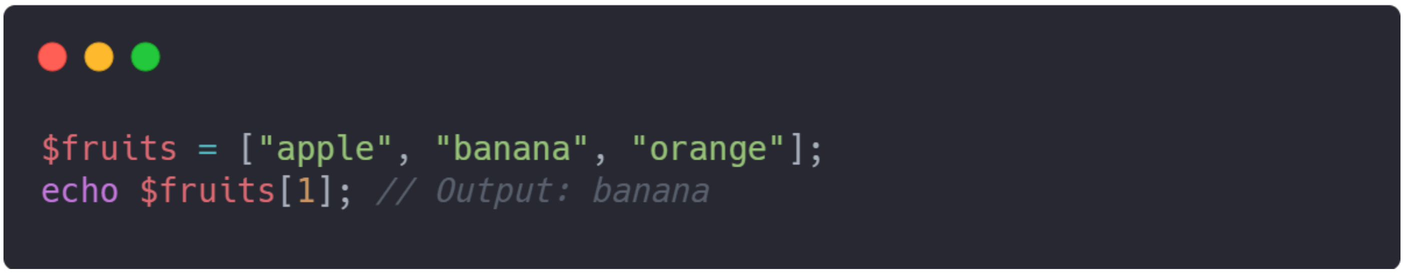
- Organize related data efficiently.
- Simplify complex data structures.
- Improve code readability and maintainability.

Types of Arrays in PHP

PHP offers three primary array types to organize your data effectively:

1. Indexed / Numeric Arrays:

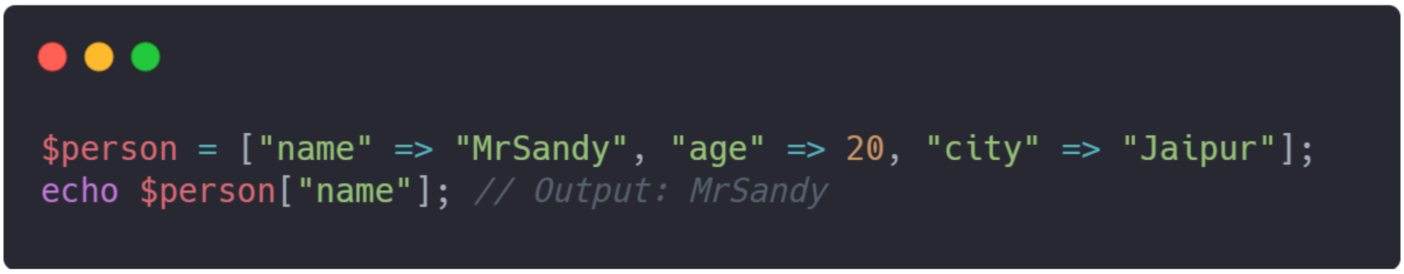
- Ordered collections of values.
- Accessed using numeric indexes starting from 0.
- Example:



```php
$fruits = ["apple", "banana", "orange"];
echo $fruits[1]; // Output: banana
```

2. Associative Arrays

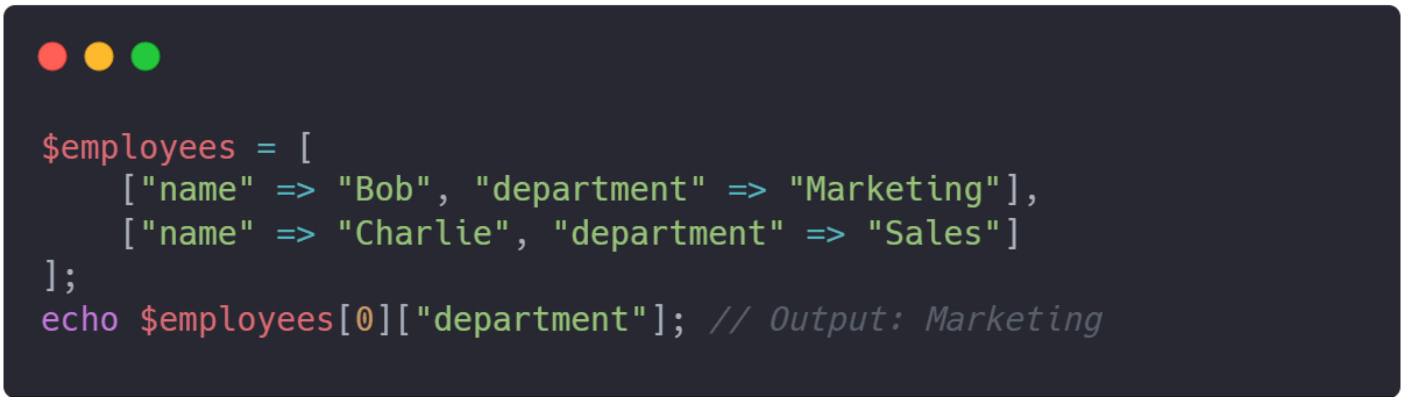
- Unordered collections of key-value pairs.
- Keys can be strings or numbers (strings are recommended for clarity).
- Example:



```php
$person = ["name" => "MrSandy", "age" => 20, "city" => "Jaipur"];
echo $person["name"]; // Output: MrSandy
```

3. Multidimensional Arrays

- Arrays that contain other arrays (nested arrays).
- Accessed using multiple indexes.
- Example:



```php
$employees = [
    ["name" => "Bob", "department" => "Marketing"],
    ["name" => "Charlie", "department" => "Sales"]
];
echo $employees[0]["department"]; // Output: Marketing
```

Key Points:

- Array values can be of any data type: strings, numbers, booleans, objects, even other arrays.
- PHP arrays are flexible and can be dynamically resized as needed.

- Built-in functions like `array_push()`, `array_pop()`, `array_shift()`, and `array_unshift()` help manipulate arrays.

PHP Functions

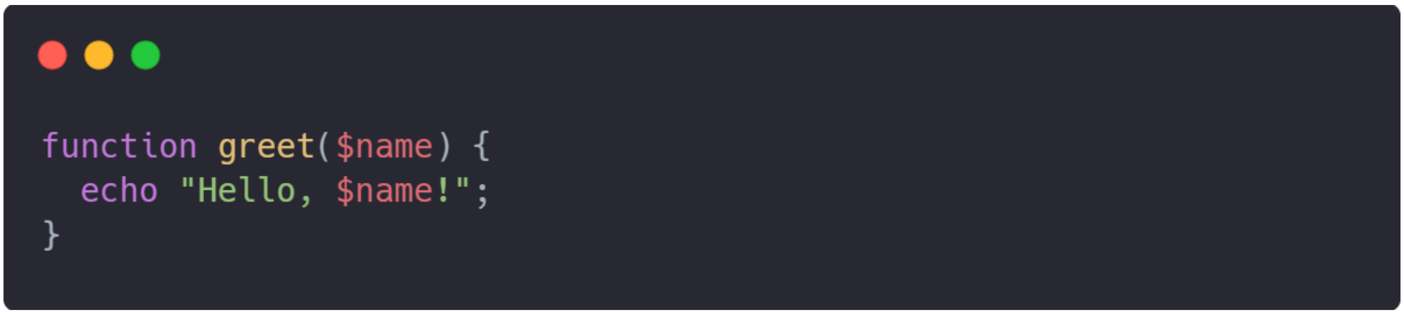
- **Reusable code blocks:** Functions allow you to group a block of code that performs a specific task. This code can be reused throughout your program by simply calling the function.
- **Improved code organization:** Functions break down complex programs into smaller, more manageable pieces, making code easier to read, understand, and maintain.

Types of Functions:

- **Built-in Functions:** PHP provides a large library of pre-written functions for common tasks like string manipulation, math operations, file handling, etc. (e.g., `echo()`, `strlen()`, `sqrt()`)
- **User-Defined Functions:** You can create your own custom functions to perform specific tasks tailored to your program's needs.

Defining a Functions

- Use the `function` keyword followed by the function name and parentheses.
- Function name should start with a letter or underscore.
- Code to be executed goes inside curly braces `{}`.
- **Example:**



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

Calling Functions

- Use the function name followed by parentheses `()`.
- Pass any required arguments (data) within the parentheses, separated by commas.
- **Example:**

```php
greet("MrSandy"); // Output: Hello, MrSandy!
```

Parameter passing / Arguments (Optional)

- Provide data to the function when calling it.
- Function can access and process this data.
- **Example (modified greet function):**

```php
function greet($name, $timeOfDay = "morning") {
  echo "Good $timeOfDay, $name!";
}

greet("Makima", "evening"); // Output: Good evening, Makima!
```

Return Values (Optional)

- Functions can return a value using the `return` statement.
- Returned value can be assigned to a variable or used in expressions.
- **Example (modified greet function):**

```php
function getGreeting($name) {
  return "Hello, $name!";
}

$message = getGreeting("Skywalker");
echo $message; // Output: Hello, Skywalker!
```

Creating Strings

1. Single quotes ('): This is the simplest way to create a string. Ideal for basic text without variables or special characters.

- Example: `$message = 'Hello, world!';`

2. Double quotes ("): Double quotes allow for variable interpolation and interpretation of escape sequences.

- Variable interpolation: `$name = "Gojo"; echo "Hello, $name!";` (Output: Hello, Gojo!)
- Escape sequences: `$path = "C:\\Users\\Sandy\\Documents";` (Escapes the backslash for proper directory path)

3. Heredoc (<<<) (Not important) : Useful for creating multi-line strings with minimal escaping.

- Example: `$content = <<<EOT This is a multi-line string created using heredoc. EOT;`

Accessing Strings

- **Indexing**: Access individual characters using zero-based indexing within square brackets.
  - Example: `$message = "Hello"; echo $message[0];` (Output: H)
- **String functions**: PHP provides various functions for manipulating strings.
  - Example: `strlen($message)` gives the length of the string, `strtoupper($message)` converts to uppercase.
  - Rest of the string functions are mentioned below in String Manipulation Functions.

Additional Key Points:

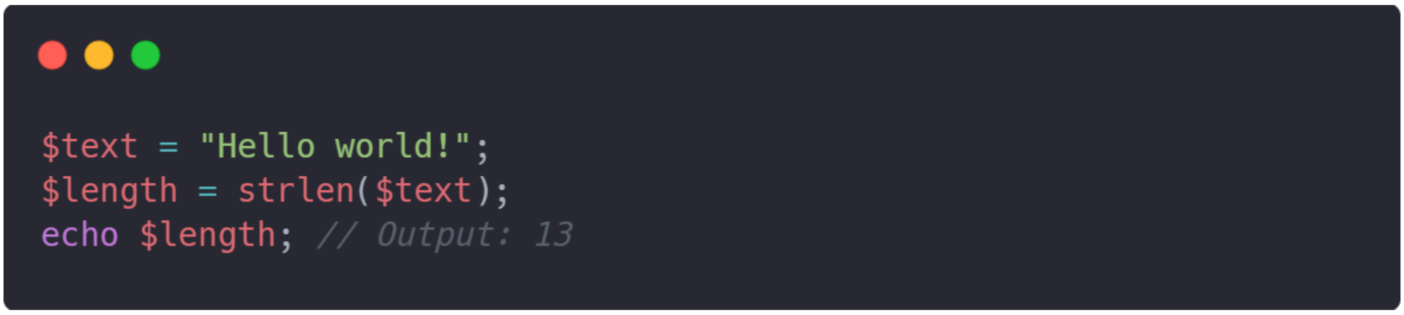
- Single quotes treat everything literally except escape sequences for single quote (') and backslash ().
- Double quotes interpret variables and escape sequences.
- Heredoc is useful for multi-line strings without extra escaping for newlines.
- Indexing allows accessing individual characters within a string.
- String functions provide powerful tools for manipulating strings.

String Manipulation Functions

PHP offers a rich set of functions for manipulating strings, making it easy to perform various tasks on text data. Here are some commonly used functions, along with explanations and examples:

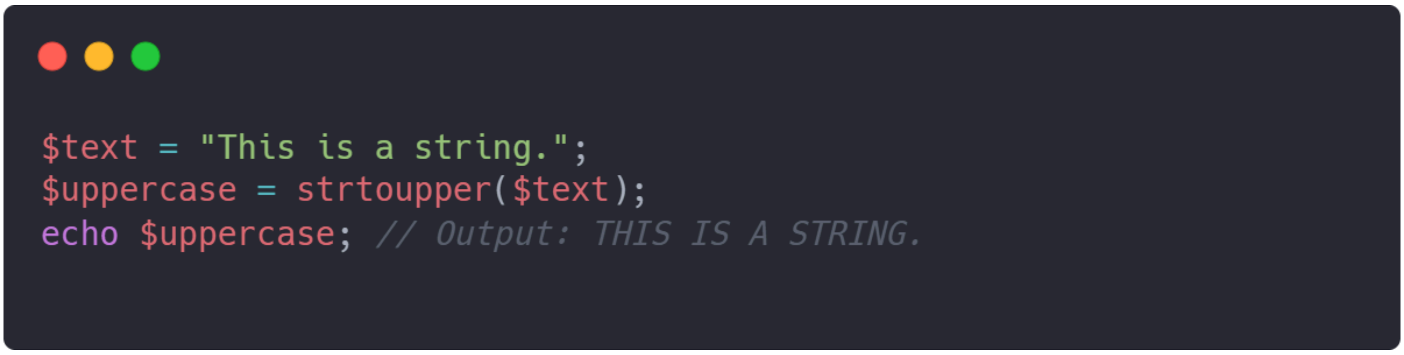
1. Length and Case:

- `strlen($string)`: Returns the number of characters in a string.



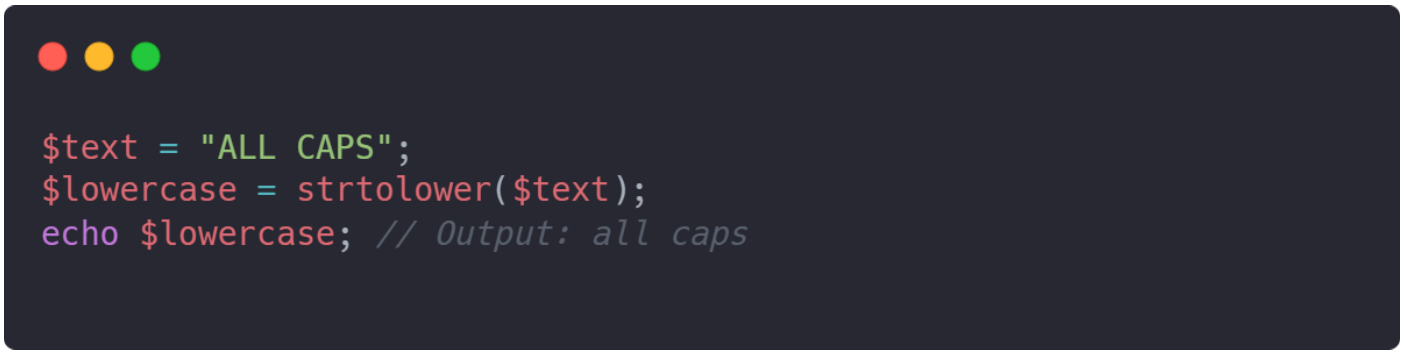
```php
$text = "Hello world!";
$length = strlen($text);
echo $length; // Output: 13
```

- `strtoupper($string)`: Converts all characters in a string to uppercase.



```php
$text = "This is a string.";
$uppercase = strtoupper($text);
echo $uppercase; // Output: THIS IS A STRING.
```

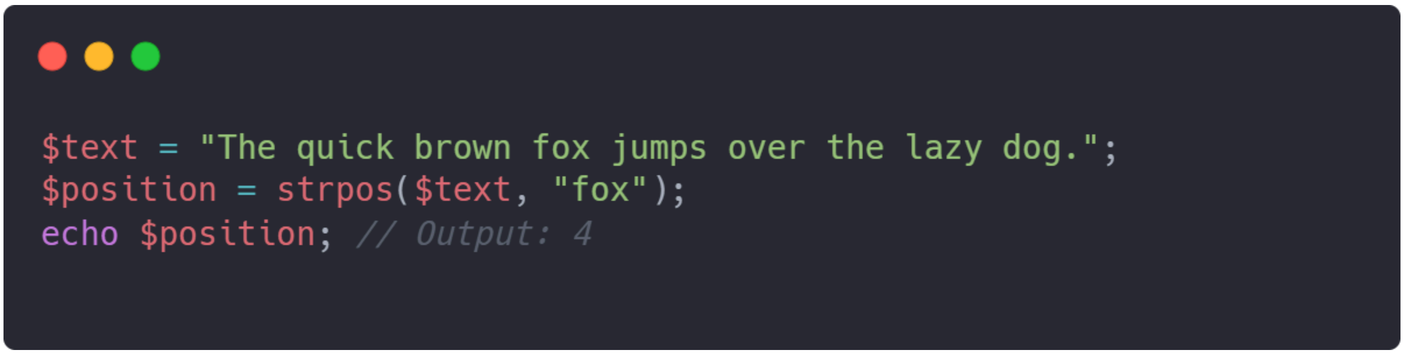
- `strtolower($string)`: Converts all characters in a string to lowercase.



```php
$text = "ALL CAPS";
$lowercase = strtolower($text);
echo $lowercase; // Output: all caps
```

2. Searching and Replacement:

- `strpos($string, $search, $start)`: Finds the first occurrence of a substring within a string.



```php
$text = "The quick brown fox jumps over the lazy dog.";
$position = strpos($text, "fox");
echo $position; // Output: 4
```

- `stripos($string, $search, $start)`: Case-insensitive version of `strpos()`
- `str_replace($search, $replace, $string)`: Replaces all occurrences of a substring with another substring.

```php
$text = "Hello, world!";
$replaced = str_replace("world", "PHP", $text);
echo $replaced; // Output: Hello, PHP!
```

3. Trimming and Splitting:

- `trim($string, $charlist)`: Removes whitespace (or a set of characters) from the beginning and end of a string.

```php
$text = "  Hello world  ";
$trimmed = trim($text);
echo $trimmed; // Output: Hello world
```

- `ltrim($string, $charlist)`: Removes whitespace (or a set of characters) from the beginning of a string.
- `rtrim($string, $charlist)`: Removes whitespace from the end of a string.
- `explode($separator, $string, $limit)`: Converts a string into an array by splitting it at the specified separator.

```php
$text = "This,is,a,comma,separated,string.";
$words = explode(",", $text);
print_r($words);
// Output: Array ( [0] => This [1] => is [2] => a [3] => comma [4]
=> separated [5] => string. )
```

4. Other Useful Functions:

- `substr($string, $start, $length)`: Extracts a portion of a string.

```php
$text = "Hello world!";
$substring = substr($text, 7, 5); // Start at index 7, get 5
characters
echo $substring; // Output: world
```

- `str_repeat($string, $multiplier)`: Repeats a string a specified number of times.

```php
$text = "Ha";
$repeated = str_repeat($text, 3);
echo $repeated; // Output: HaHaHa
```

- `str_split($string, $length)`: Splits a string into an array of characters.

```php
$text = "PHP";
$chars = str_split($text);
print_r($chars);
// Output: Array ( [0] => P
```

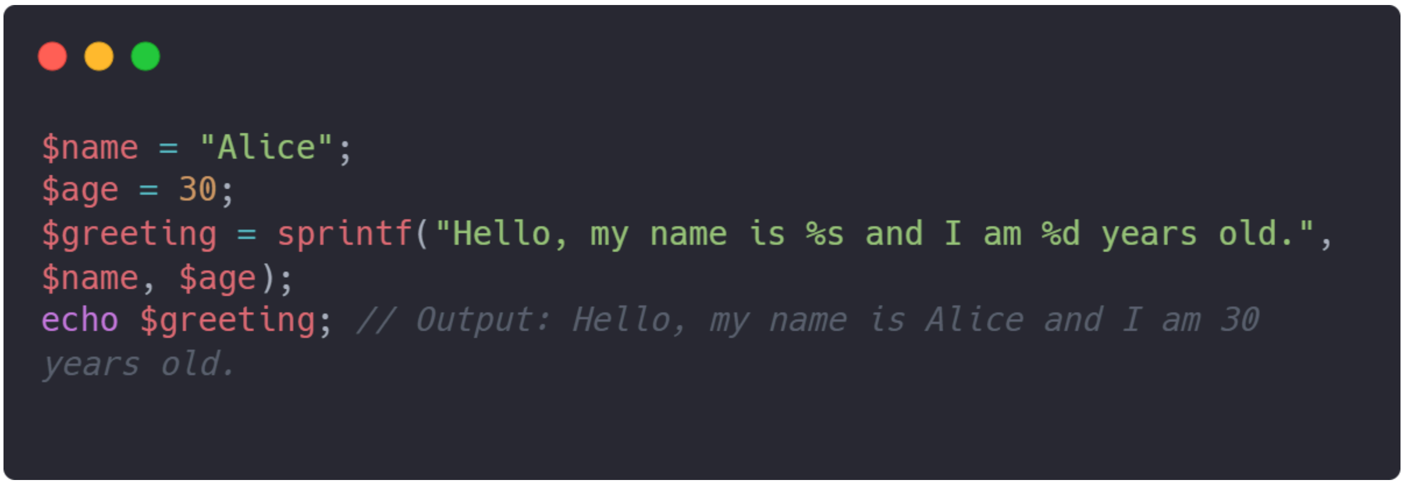
Formatting Strings

1. Using `sprintf()` function:

- Creates a formatted string by inserting placeholders with corresponding values.
- **Syntax:** `sprintf(format_string, arg1, arg2, ...)`
  - `Format_string` : String containing placeholders (`%`) and formatting codes.

    ◦ `arg1, arg2, ...` : Values to be inserted at placeholders (in order).

Example:



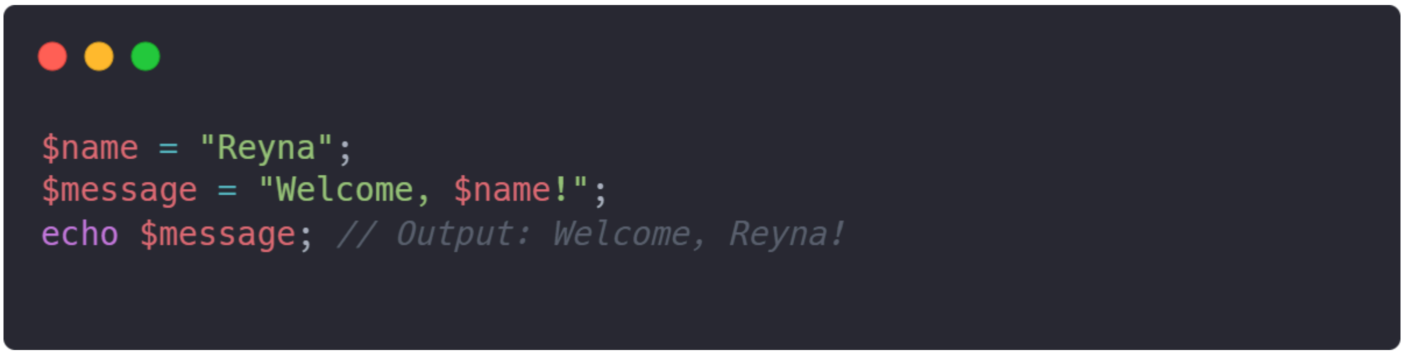
```php
$name = "Alice";
$age = 30;
$greeting = sprintf("Hello, my name is %s and I am %d years old.",
$name, $age);
echo $greeting; // Output: Hello, my name is Alice and I am 30
years old.
```

Formatting Codes: (Within placeholders `%`)

- Specify how to format the corresponding argument.
- Common codes:
    - `%s`: String
    - `%d`: Signed decimal number (integer)
    - `%f`: Floating-point number
    - `%c`: Single character
    - `%x`: Hexadecimal number (lowercase)
    - `%X`: Hexadecimal number (uppercase)
- `printf()` function for direct output formatting (doesn't return a string like `sprintf()`).

2. String Interpolation (Double-quoted strings):

- Embed variables directly within double-quoted strings.
- **Syntax:** `$variable_name` inside double quotes.
- **Example:**



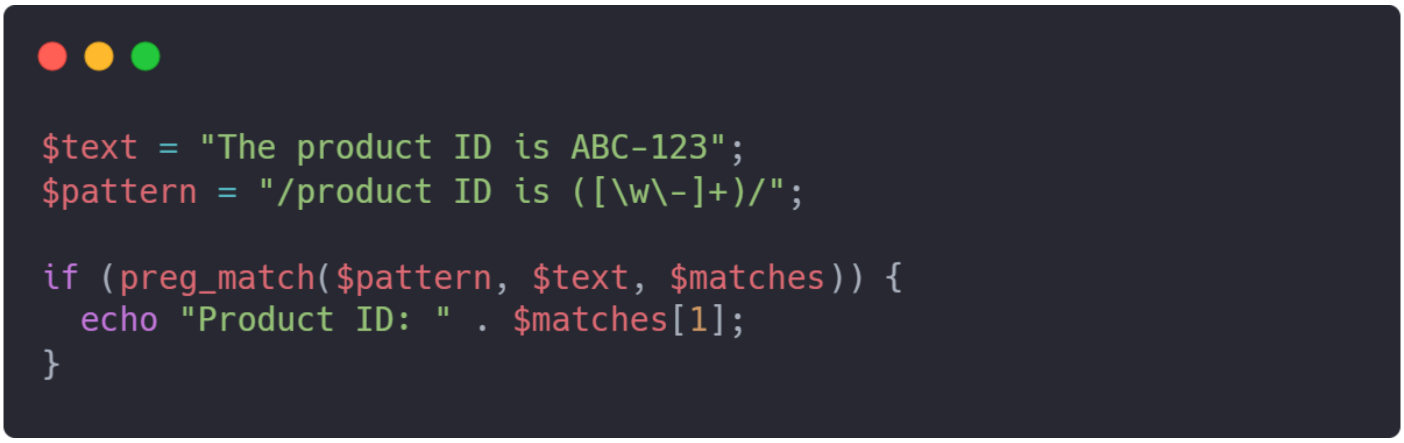
```php
$name = "Reyna";
$message = "Welcome, $name!";
echo $message; // Output: Welcome, Reyna!
```

Pattern Matching

Pattern matching involves searching for specific patterns or regular expressions within a string. PHP provides functions like preg_match() for pattern matching.

1. Regular Expressions (Regex):

- Powerful tool for string manipulation and pattern searching.
- Syntax defines patterns for character sequences.
- Functions like `preg_match()`, `preg_match_all()`, `preg_replace()`, and `preg_split()` facilitate various operations.
- Example:



```php
$text = "The product ID is ABC-123";
$pattern = "/product ID is ([\w\-]+)/";

if (preg_match($pattern, $text, $matches)) {
  echo "Product ID: " . $matches[1];
}
```