Web-Programming-Module-3-Important-Topics-PYQs

- 16. Write equivalent PHP statements corresponding to the following:
  - i. Declare an Associative Array Named "ages"
  - ii. Modify the Value Associated with the Key "Megha" to 28
  - iii. Sort the Array According to Values Maintaining Key-Value Relationships and Print
  - iv. Access the Entry Identified by the Key "Alice"
- 17. Design a HTML form to input a string and to display whether it is palindrome or not on form submission using PHP script.
- 18. Write a PHP script to search for a specific string pattern in a text.
- 19. Different types arrays with examples in PHP
  - 1. Indexed Arrays
    - Example:
  - 2. Associative Arrays
    - Example:
  - 3. Multidimensional Arrays
    - Example:

Important Topics & PYQs

1. Write a PHP program to check whether the given number is prime or not.

```php
<?php
// Function to check if a number is prime
function isPrime($num) {
    // A prime number is greater than 1 and divisible only by 1 and itself
    if ($num <= 1) {
        return false; // Numbers less than or equal to 1 are not prime
    }

    // Check for factors from 2 to sqrt($num)
    for ($i = 2; $i <= sqrt($num); $i++) {
        if ($num % $i == 0) {
            return false; // Found a factor, so it's not prime
        }
    }

    return true; // No factors found, the number is prime
}

// Get the number from user input (via command line or form)
$number = 11; // You can replace this with any number or use user input

// Check if the number is prime
if (isPrime($number)) {
    echo $number . " is a prime number.";
} else {
    echo $number . " is not a prime number.";
}
?>
```

2. List out the sorting functions for Arrays in PHP. Illustrate with suitable examples.

1. sort()

Sorts an array in ascending order.

Example:

```
<?php
$array = [4, 2, 8, 1, 5];
sort($array);
print_r($array); // Outputs: [1, 2, 4, 5, 8]
?>
```

2. rsort()

Sorts an array in descending order.

Example:

```
<?php
$array = [4, 2, 8, 1, 5];
rsort($array);
```

```
print_r($array); // Outputs: [8, 5, 4, 2, 1]
?>
```

3. asort()

Sorts an array in ascending order, maintaining the key-value association.

Example:

```
<?php
$array = ["a" => 4, "b" => 2, "c" => 8, "d" => 1];
asort($array);
print_r($array); // Outputs: ["d" => 1, "b" => 2, "a" => 4, "c" => 8]
?>
```

4. ksort()

Sorts an array by its keys in ascending order.

Example:

```
<?php
$array = ["a" => 4, "b" => 2, "c" => 8, "d" => 1];
ksort($array);
print_r($array); // Outputs: ["a" => 4, "b" => 2, "c" => 8, "d" => 1]
?>
```

3. List six primitive data types used in PHP?

In PHP, there are **six primitive (basic) data types**. These are the fundamental types used to store data values. Below is the list along with a brief description of each:

1. Integer

- **Description**: Represents whole numbers (positive or negative) without any decimal points.
- **Example**:

```
$age = 25;  // Integer
```

2. Float (or Double)

- **Description**: Represents numbers with a decimal point, also known as floating-point numbers.
- **Example**:

```
$price = 19.99;  // Float
```

3. String

- **Description**: Represents a sequence of characters or text.
- **Example**:

```
$name = "John Doe";  // String
```

4. Boolean

- **Description**: Represents two possible values: `TRUE` or `FALSE`.
- **Example**:

```
$is_active = true;  // Boolean
```

5. Array

- **Description**: Represents a collection of values or a set of variables, which can be accessed using indexes or keys.
- **Example**:

```
$fruits = array("apple", "banana", "orange");  // Array
```

6. NULL

- **Description**: Represents a variable with no value assigned to it. It is used to indicate the absence of a value.
- **Example**:

```php
$value = null;  // NULL
```

4. *Give 3 string functions used in PHP associated with removal of whitespaces? How will they differ each other?*

In PHP, there are several string functions that can be used to remove whitespaces from strings. Below are **three functions** specifically designed to handle whitespaces, along with explanations of how they differ:

1. trim()

- **Description**: Removes whitespaces (or other predefined characters) from the **beginning and end** of a string.
- **Example**:

```php
$str = "  Hello World!  ";
$trimmed = trim($str);
echo $trimmed;  // Output: "Hello World!"
```

- **Usage**: `trim()` is used when you want to remove unwanted whitespace or characters from both ends of the string, such as leading and trailing spaces, tabs, or newline characters.
- **Key Difference**: It only affects the **start and end** of the string, not spaces between words.

2. ltrim()

- **Description**: Removes whitespaces (or other predefined characters) from the **left (beginning)** of the string.
- **Example**:

```php
$str = "  Hello World!  ";
$leftTrimmed = ltrim($str);
echo $leftTrimmed;  // Output: "Hello World!  "
```

- **Usage**: `ltrim()` is useful when you want to remove spaces or other unwanted characters only from the **beginning** of the string.
- **Key Difference**: It affects only the **left side** of the string, leaving the right side intact.

3. rtrim()

- **Description**: Removes whitespaces (or other predefined characters) from the **right (end)** of the string.
- **Example**:

```php
$str = "  Hello World!  ";
$rightTrimmed = rtrim($str);
echo $rightTrimmed;  // Output: "  Hello World!"
```

- **Usage**: `rtrim()` is used when you want to remove spaces or other unwanted characters from the **end** of the string.
- **Key Difference**: It affects only the **right side** of the string, leaving the left side intact.

Summary of Differences:

Function	Removes Whitespace From	Example Output
`trim()`	Both ends (start and end)	`"Hello World!"`
`ltrim()`	Left side (beginning)	`"Hello World!  "`
`rtrim()`	Right side (end)	`"  Hello World!"`

5. How to use functions in a PHP program with an example.

In PHP, **functions** are blocks of code that can be called to perform a specific task. Functions help in organizing the code, improving reusability, and making it easier to maintain. Below is an explanation of how to define and use functions in a PHP program with an example.

Steps to Use Functions in PHP

1. **Define a Function**: A function is defined using the `function` keyword, followed by the function name, parentheses `()`, and a block of code enclosed in curly braces `{}`.
2. **Call the Function**: Once the function is defined, you can call it in your program by simply writing its name followed by parentheses.
3. **Pass Arguments to Functions (Optional)**: Functions can accept parameters (arguments) to perform actions on the data passed to them.
4. **Return Values (Optional)**: Functions can return values using the `return` keyword. The return value can be used elsewhere in the program.

Example of a PHP Program Using Functions

```php
<?php

// Step 1: Define a function
function greet($name) {
    // Step 4: Return a greeting message
    return "Hello, " . $name . "!";
}

// Step 2: Call the function
$name = "John";  // Define a variable
$message = greet($name);  // Call the function with the variable as an
argument

// Step 3: Display the returned value
echo $message;  // Output: Hello, John!

?>
```

Explanation:

1. **Function Definition**:
   - The function `greet($name)` is defined with one parameter `$name`.

- Inside the function, the `return` statement creates a greeting message by concatenating `"Hello, "` with the value of `$name`, followed by an exclamation mark.

2. **Function Call**:
    - In the main program, we define a variable `$name = "John"`.
    - We then call the `greet($name)` function, passing the value of `$name` as an argument.
3. **Display the Output**:
    - The function returns a greeting message, which is stored in the `$message` variable.
    - Finally, the `echo` statement outputs the value of `$message`, which is `"Hello, John!"`.

Another Example: Function with Multiple Parameters and No Return Value

```php
<?php

// Function to print a personalized greeting
function printGreeting($firstName, $lastName) {
    echo "Hello, $firstName $lastName!";
}

// Calling the function with two arguments
printGreeting("John", "Doe");  // Output: Hello, John Doe!

?>
```

6. Distinguish between implode and explode function in PHP with suitable examples.

- **`explode()`**: This function splits a string into an array based on a specified delimiter.
  Example:

```php
<?php
$string = "apple,banana,orange";
$array = explode(",", $string);
print_r($array);
/*
Output:
Array
(
    [0] => apple
    [1] => banana
    [2] => orange
)
*/
?>
```

Explanation:

- The `explode(",", $string)` function takes a string and splits it at each comma, creating an array.
- `implode()`: This function joins array elements into a single string, using a specified delimiter.

Example:

```php
<?php
$array = array("apple", "banana", "orange");
$string = implode(", ", $array);
echo $string; // Output: apple, banana, orange
?>
```

Explanation:

- The `implode(", ", $array)` function takes an array and joins its elements into a single string, separated by a comma and space.

7. Write a PHP program to check whether the given number is Armstrong or not.

```php
<?php
// Function to check if a number is Armstrong
function checkArmstrong($num) {
    // Step 1: Find the number of digits
    $originalNum = $num;
    $sum = 0;

    // Step 2: Get each digit of the number
    while ($num > 0) {
        // Get the last digit of the number
        $digit = $num % 10;

        // Add the cube of the digit to the sum
        $sum += $digit * $digit * $digit;

        // Remove the last digit from the number
        $num = (int)($num / 10);
    }

    // Step 3: Check if the sum of cubes is equal to the original number
    if ($sum == $originalNum) {
        return true;  // It's an Armstrong number
    } else {
        return false;  // It's not an Armstrong number
    }
}

// Step 4: Test the function with an example number
$number = 153;

// Step 5: Check if the number is Armstrong
if (checkArmstrong($number)) {
    echo "$number is an Armstrong number.";
} else {
    echo "$number is not an Armstrong number.";
}
?>
```

8. Explain any six string handling functions used in PHP with example.

1. strlen() - Get the Length of a String

The `strlen()` function returns the length (number of characters) of a string.

Example:

```
$string = "Hello, World!";
echo strlen($string);  // Output: 13
```

Explanation: The string "Hello, World!" has 13 characters, so `strlen()` returns 13.

2. strtoupper() - Convert a String to Uppercase

The `strtoupper()` function converts all characters in a string to uppercase.

Example:

```
$string = "hello";
echo strtoupper($string);  // Output: HELLO
```

Explanation: The string "hello" is converted to "HELLO" using `strtoupper()`.

3. strtolower() - Convert a String to Lowercase

The `strtolower()` function converts all characters in a string to lowercase.

Example:

```
$string = "HELLO";
echo strtolower($string);  // Output: hello
```

Explanation: The string "HELLO" is converted to "hello" using `strtolower()`.

4. substr() - Extract a Substring

The `substr()` function extracts a part of a string starting from a specified position and optionally up to a specified length.

Example:

```
$string = "Hello, World!";
echo substr($string, 7, 5);  // Output: World
```

Explanation: The function extracts 5 characters starting from position 7 (the "W" in "World"). The output is "World".

5. strpos() - Find the Position of the First Occurrence of a Substring

The `strpos()` function finds the position of the first occurrence of a substring in a string. It returns `false` if the substring is not found.

Example:

```
$string = "Hello, World!";
echo strpos($string, "World");  // Output: 7
```

Explanation: The substring "World" starts at position 7 in the string, so `strpos()` returns 7.

6. trim() - Remove Whitespace from Both Sides of a String

The `trim()` function removes whitespace (or other specified characters) from both ends of a string.

Example:

```
$string = "  Hello, World!  ";
echo trim($string);  // Output: Hello, World!
```

Explanation: The `trim()` function removes the spaces at the beginning and end of the string, resulting in "Hello, World!".

Summary of Functions:

Function	Purpose	Example
`strlen()`	Returns the length of a string	`strlen("Hello")` → `5`
`strtoupper()`	Converts a string to uppercase	`strtoupper("hello")` → `HELLO`
`strtolower()`	Converts a string to lowercase	`strtolower("HELLO")` → `hello`
`substr()`	Extracts a substring from a string	`substr("Hello", 1, 3)` → `ell`
`strpos()`	Finds the position of the first occurrence of a substring	`strpos("Hello", "e")` → `1`
`trim()`	Removes whitespace from both ends of a string	`trim(" Hello ")` → `Hello`

9. What is the purpose of the implicit arrays $_POST and $_GET in PHP? Consider that a web page displays a form containing two text boxes (named num1 and num2), where the user enters numeric data. Write a PHP script which collects this form data, finds the sum, difference and the product of the two numbers and then displays the same with suitable messages. Assume that the script is to be embedded in the web page specified by the action attribute of the form and that the method used when the form is submitted is GET.

Purpose of `$_POST` and `$_GET` in PHP:

- `$_POST` and `$_GET` are **superglobal arrays** in PHP that are used to collect data sent from a form submitted by the user.
  - `$_POST` is used to collect form data sent using the **POST** method. It is not visible in the URL and is more secure for sending sensitive data (like passwords).
  - `$_GET` is used to collect form data sent using the **GET** method. The form data is appended to the URL and can be seen in the browser's address bar, making it less secure for sensitive information. However, it is useful for passing data through URLs.

Example PHP Script Using `$_GET`:

Let's say you have a form with two text boxes (`num1` and `num2`) where the user enters numeric values. You want to calculate the sum, difference, and product of these two numbers and display the results.

HTML Form (index.html):

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Calculator</title>
</head>
<body>

    <h2>Simple Calculator</h2>
    <form action="calculate.php" method="GET">
        <label for="num1">Number 1:</label>
        <input type="number" id="num1" name="num1" required><br><br>

        <label for="num2">Number 2:</label>
        <input type="number" id="num2" name="num2" required><br><br>

        <input type="submit" value="Calculate">
    </form>

</body>
</html>
```

PHP Script (calculate.php):

```php
<?php
// Check if the form has been submitted using GET method
if (isset($_GET['num1']) && isset($_GET['num2'])) {
    // Retrieve the numbers from the form data
    $num1 = $_GET['num1'];
    $num2 = $_GET['num2'];

    // Perform calculations
    $sum = $num1 + $num2;
    $difference = $num1 - $num2;
    $product = $num1 * $num2;

    // Display the results
    echo "<h2>Results:</h2>";
    echo "Sum: " . $sum . "<br>";
    echo "Difference: " . $difference . "<br>";
    echo "Product: " . $product . "<br>";
} else {
    // If the form data isn't available
    echo "Please enter both numbers in the form.";
}
?>
```

Explanation:

1. **HTML Form (`index.html`)**:
   - The form contains two text boxes (`num1` and `num2`) where the user will enter numeric data.
   - The `action` attribute specifies the PHP script (`calculate.php`) that will process the form data.
   - The `method="GET"` attribute ensures that the form data is sent via the GET method, which means the data will be appended to the URL.
2. **PHP Script (`calculate.php`)**:
   - The script checks if the form fields `num1` and `num2` are set using `isset($_GET['num1'])` and `isset($_GET['num2'])`.
   - It then retrieves the numeric values entered by the user using `$_GET['num1']` and

`$_GET['num2']`.

- The script calculates the sum, difference, and product of the two numbers.
- The results are then displayed using `echo`.

Example:

- If the user enters `5` for `num1` and `3` for `num2` and submits the form, the URL will look like this:

  ```
  http://yourdomain.com/calculate.php?num1=5&num2=3
  ```

  And the PHP script will output:

  ```
  Results:
  Sum: 8
  Difference: 2
  Product: 15
  ```

10. Why is PHP considered to be dynamically typed?

PHP is considered a **dynamically typed** language because the type of a variable is **determined at runtime**, not at compile time. In other words, you don't need to explicitly declare the type of a variable when you create it. The type is inferred based on the value assigned to the variable during program execution.

Key Features of PHP's Dynamic Typing:

1. **No Explicit Type Declaration**: In PHP, you don't need to specify the type of a variable when you declare it. The variable type is automatically inferred based on the value it holds at runtime.
   Example:

```
$number = 5;  // PHP infers that $number is an integer
$text = "Hello, world!";  // PHP infers that $text is a string
$price = 19.99;  // PHP infers that $price is a float
```

2. **Changing Types During Execution**: A variable's type can change dynamically during execution. PHP allows a variable to hold a value of any type, and it can change that type whenever a new value is assigned.

   Example:

   ```
   $value = 10;    // Initially, $value is an integer
   $value = "PHP"; // Now, $value becomes a string
   $value = 15.5;  // Finally, $value becomes a float
   ```

3. **No Need for Type Conversion**: PHP automatically converts types when necessary (type casting). For example, if you try to add an integer to a string, PHP will attempt to convert the string to a number.

   Example:

   ```
   $a = 10;       // Integer
   $b = "20";     // String
   $sum = $a + $b; // PHP automatically converts $b to an integer,
   resulting in $sum = 30
   echo $sum;     // Outputs 30
   ```

4. **Type Checking at Runtime**: PHP checks the variable's type at the moment the variable is used, rather than at the start of the program. This allows flexibility in handling different types, but it can also lead to runtime errors if types don't match the expected operations.

   Example:

   ```
   $value = "10";  // String
   $value = $value + 5;  // PHP automatically converts the string "10" to
   the integer 10, resulting in $value = 15
   echo $value;  // Outputs 15
   ```

11. Develop a PHP program to print the factorial of a number using a function that accepts number as input and returns the factorial?

```php
<?php
// Function to calculate factorial
function calculateFactorial($number) {
    // Initialize the factorial to 1
    $factorial = 1;

    // Loop from 1 to the given number
    for ($i = 1; $i <= $number; $i++) {
        // Multiply the factorial by the current number
        $factorial *= $i;
    }

    // Return the factorial
    return $factorial;
}

// Input number to calculate factorial
$number = 5; // You can change this value to test with different numbers

// Calling the function and displaying the result
echo "The factorial of $number is: " . calculateFactorial($number);
?>
```

12. Why is regular expressions used in PHP? What are the 3 components of a regular expression? Name any two Pearl compatible functions used with regular expressions?

Regular expressions (regex) are used in PHP to **search**, **match**, **replace**, and **validate** patterns in strings. They are incredibly powerful and flexible for text processing, making them ideal for tasks like:

1. **Pattern Matching**: Finding specific patterns in strings, such as matching an email address or a phone number.
2. **Search and Replace**: Replacing certain patterns within a string, for example, changing all occurrences of a word.
3. **Data Validation**: Validating if an input matches a specific format, like checking if a string is a valid email address.
4. **String Parsing**: Extracting specific portions of a string based on patterns, like extracting dates or specific keywords.

Three Components of a Regular Expression

1. **Literal Characters**: These are the basic characters that the regex will match exactly as they appear. For example, `/abc/` will match the string "abc".
2. **Metacharacters**: These are special characters that have a specific meaning in regular expressions. They allow you to define patterns more flexibly. Common metacharacters include:
   - `.` (dot) - Matches any single character except a newline.
   - `^` (caret) - Anchors the match at the start of the string.
   - `$` (dollar sign) - Anchors the match at the end of the string.
   - `[]` (square brackets) - Defines a character class, e.g., `[a-z]` matches any lowercase letter.
   - `|` (pipe) - Represents logical OR, e.g., `abc|xyz` matches either "abc" or "xyz".
3. **Quantifiers**: These specify how many times a character or group should be matched. For example:
   - `*` - Matches 0 or more occurrences.
   - `+` - Matches 1 or more occurrences.
   - `?` - Matches 0 or 1 occurrence.
   - `{n}` - Matches exactly `n` occurrences.

Two Perl-Compatible Functions Used with Regular Expressions in PHP

PHP provides several functions to work with regular expressions, especially Perl-Compatible Regular Expressions (PCRE), which are more powerful and flexible. Here are two commonly used PCRE functions:

1. **`preg_match()`**:

- This function searches a string for a pattern and returns `true` if the pattern is found, and `false` otherwise.
- Example:

```
if (preg_match("/abc/", "abcdef")) {
    echo "Pattern found!";
} else {
    echo "Pattern not found.";
}
```

2. **`preg_replace()`** :
   - This function performs a search and replace operation based on a regular expression pattern. It searches for a pattern and replaces it with a specified string.
   - Example:

```
$result = preg_replace("/cat/", "dog", "I have a cat.");
echo $result;  // Outputs: "I have a dog."
```

13. Write a program to

- Declare an associative array named “items” to store the key-value pairs (“Bread”, 30), (“Butter”, 30), (“Jam”, 35), (“Cheese”, 32).
- Print the array using for each loop
- Sort the array according to values maintaining the key-value relationships and print the sorted key-value pairs.

```
<?php
// Declare an associative array with items and their prices
$items = array(
    "Bread" => 30,
    "Butter" => 30,
    "Jam" => 35,
    "Cheese" => 32
);
```

```php
// Print the original array using foreach loop
echo "Original Array:\n";
foreach ($items as $key => $value) {
    echo "$key => $value\n";
}

// Sort the array based on values (ascending order)
asort($items);

// Print the sorted array using foreach loop
echo "\nSorted Array by Values:\n";
foreach ($items as $key => $value) {
    echo "$key => $value\n";
}
?>
```

14. Write a PHP program to compute the sum of the positive integers up to 100 using do while loop.

```php
<?php
// Initialize variables
$sum = 0;
$i = 1;

// Do-while loop to compute sum
do {
    $sum += $i;  // Add current value of $i to sum
    $i++;          // Increment $i by 1
} while ($i <= 100); // Continue loop until $i is greater than 100

// Output the result
echo "The sum of the positive integers up to 100 is: $sum";
?>
```

15. Explain the loops used in PHP with example.

The primary loops in PHP are:

1. `for` **loop**
2. `while` **loop**
3. `do-while` **loop**
4. `foreach` **loop** (used specifically for arrays)

1. `for` loop

The `for` loop is used when you know the number of iterations beforehand. It consists of three parts:

- Initialization: sets the counter to a starting value.
- Condition: checks if the counter is within a certain range.
- Increment/Decrement: updates the counter after each iteration.

Syntax:

```
for (initialization; condition; increment/decrement) {
    // Code to be executed
}
```

Example:

```php
<?php
for ($i = 1; $i <= 5; $i++) {
    echo "Iteration number: $i <br>";
}
?>
```

Explanation:

- **Initialization**: `$i = 1` (Starts the loop with `$i` equal to 1).
- **Condition**: `$i <= 5` (The loop will run as long as `$i` is less than or equal to 5).
- **Increment**: `$i++` (Increments `$i` by 1 after each iteration).

Output:

```
Iteration number: 1
Iteration number: 2
Iteration number: 3
Iteration number: 4
Iteration number: 5
```

2. `while` loop

The `while` loop is used when you want to repeat a block of code as long as a condition is true. It checks the condition before each iteration.

Syntax:

```
while (condition) {
    // Code to be executed
}
```

Example:

```
<?php
$i = 1;
while ($i <= 5) {
    echo "Iteration number: $i <br>";
    $i++; // Increment the counter
}
?>
```

Explanation:

- The loop starts with `$i = 1` and continues running as long as `$i` is less than or equal to 5.
- After each iteration, `$i` is incremented.

Output:

```
Iteration number: 1
Iteration number: 2
Iteration number: 3
Iteration number: 4
Iteration number: 5
```

3. `do-while` loop

The `do-while` loop is similar to the `while` loop, but it guarantees that the block of code will run at least once, even if the condition is false initially. It checks the condition after the loop executes, which means the loop will always run at least once.

Syntax:

```
do {
    // Code to be executed
} while (condition);
```

Example:

```php
<?php
$i = 1;
do {
    echo "Iteration number: $i <br>";
    $i++; // Increment the counter
} while ($i <= 5);
?>
```

Explanation:

- The code inside the `do` block runs first, and then the condition `$i <= 5` is checked after the first execution.

Output:

```
Iteration number: 1
Iteration number: 2
Iteration number: 3
Iteration number: 4
Iteration number: 5
```

4. `foreach` loop

The `foreach` loop is specifically used for iterating over arrays. It simplifies the process of accessing each element in an array without the need for using a counter.

Syntax:

```
foreach ($array as $value) {
    // Code to be executed with $value (element of array)
}
```

Or, to get both key and value:

```
foreach ($array as $key => $value) {
    // Code to be executed with $key and $value
}
```

Example:

```php
<?php
$fruits = ["Apple", "Banana", "Orange", "Grapes"];

foreach ($fruits as $fruit) {
    echo "I love $fruit <br>";
}
?>
```

Explanation:

- `$fruits` is an array, and the `foreach` loop will iterate over each element in the array.
- Each element is assigned to `$fruit` and printed out in the loop.

Output:

```
I love Apple
I love Banana
I love Orange
I love Grapes
```

Example with Key-Value Pair (Associative Array):

```php
<?php
$items = ["Bread" => 30, "Butter" => 40, "Jam" => 35];

foreach ($items as $item => $price) {
    echo "$item costs $price <br>";
}
?>
```

Output:

```
Bread costs 30
Butter costs 40
Jam costs 35
```

16. Write equivalent PHP statements corresponding to the following:

- i.Declare an associative array named “ages” to store the key-value pairs (“Harry”, 21), (“Alice”, 20), (“Megha”, 22), (“Bob”, 19).
- ii. Modify the value associated with the key “Megha” to 28.
- iii. Sort the array according to values maintaining the key-value relationships and print the sorted key-value pairs.
- iv. The entry identified by the key “Alice”.

i. Declare an Associative Array Named "ages"

```php
<?php
$ages = array(
    "Harry" => 21,
    "Alice" => 20,
    "Megha" => 22,
    "Bob" => 19
);
?>
```

ii. Modify the Value Associated with the Key "Megha" to 28

```
<?php
$ages["Megha"] = 28;
?>
```

iii. Sort the Array According to Values Maintaining Key-Value Relationships and Print

```
<?php
asort($ages); // Sort by values while maintaining keys

foreach ($ages as $name => $age) {
    echo "$name: $age<br>";
}
?>
```

iv. Access the Entry Identified by the Key "Alice"

```
<?php
echo "Alice's age is: " . $ages["Alice"];
?>
```

17. Design a HTML form to input a string and to display whether it is palindrome or not on form submission using PHP script.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Palindrome Check</title>
</head>
<body>

    <h2>Palindrome Checker</h2>

    <!-- HTML Form to input a string -->
    <form method="post">
        <label for="inputString">Enter a String:</label>
        <input type="text" id="inputString" name="inputString" required>
        <button type="submit" name="submit">Check</button>
    </form>

    <?php
    // PHP Script to check if the string is a palindrome
    if (isset($_POST['submit'])) {
        // Get the input string from the form
        $inputString = $_POST['inputString'];

        // Check if the string is the same forwards and backwards
        if ($inputString === strrev($inputString)) {
            echo "<p>'$inputString' is a palindrome!</p>";
        } else {
            echo "<p>'$inputString' is not a palindrome.</p>";
        }
    }
    ?>

</body>
</html>
```

18. Write a PHP script to search for a specific string pattern in a text.

```
<?php
// Sample text
$textInput = "The quick brown fox jumps over the lazy dog";
```

```
// The pattern you want to search for
$searchPattern = "fox";

// Use preg_match to check if the pattern exists in the text
if (preg_match("/$searchPattern/", $textInput)) {
    echo "The pattern '$searchPattern' was found in the text!";
} else {
    echo "The pattern '$searchPattern' was NOT found in the text.";
}
?>
```

19. Different types arrays with examples in PHP

In PHP, there are three main types of arrays:

1. **Indexed Arrays**
2. **Associative Arrays**
3. **Multidimensional Arrays**

1. Indexed Arrays

An indexed array is an array where each element is associated with an index (numeric key). The index is automatically assigned starting from 0.

Example:

```
<?php
// Indexed array
$fruits = ["Apple", "Banana", "Cherry"];

// Accessing elements
echo $fruits[0]; // Outputs: Apple
echo $fruits[1]; // Outputs: Banana
?>
```

2. Associative Arrays

An associative array is an array where each element is associated with a custom key (a string or other type) instead of an automatically assigned index.

Example:

```php
<?php
// Associative array
$person = [
    "name" => "John",
    "age" => 25,
    "city" => "New York"
];

// Accessing elements
echo $person["name"]; // Outputs: John
echo $person["age"];  // Outputs: 25
?>
```

3. Multidimensional Arrays

A multidimensional array is an array that contains other arrays. It can be thought of as an array of arrays.

Example:

```php
<?php
// Multidimensional array
$contacts = [
    "John" => ["email" => "john@example.com", "phone" => "1234567890"],
    "Jane" => ["email" => "jane@example.com", "phone" => "0987654321"]
];

// Accessing elements
echo $contacts["John"]["email"]; // Outputs: john@example.com
echo $contacts["Jane"]["phone"]; // Outputs: 0987654321
?>
```