



You:

1) Justify the following statement using function and argument passing. "PHP is a loosely typed language."



ChatGPT:

The statement "PHP is a loosely typed language" refers to the fact that PHP does not require explicit declaration of data types for variables. In PHP, a variable can change its data type during runtime based on the value assigned to it.

Now, let's justify this statement using function and argument passing:

Consider a function that takes two arguments, adds them, and returns the result. In a loosely typed language like PHP, you can pass arguments of different types (e.g., string and integer) to the function without explicitly specifying their data types.

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

// Example 1: Pass integers
$result1 = add(5, 10); // Result: 15

// Example 2: Pass a string and an integer
$result2 = add("20", 30); // Result: 50 (PHP converts the string to an integer)

// Example 3: Pass a string and a float
$result3 = add("10.5", 5.5); // Result: 16 (PHP converts the string to a float)
```

In these examples, PHP automatically performs type coercion or conversion as needed, allowing you to pass arguments of different types to the add function. This behavior illustrates the loose typing nature of PHP, where the language is flexible in handling different data types without requiring explicit type declarations.

In contrast, in a strictly typed language, you would need to declare the data types of the function parameters, and passing arguments of incompatible types would result in a type error. The fact that PHP allows such flexibility in handling different data types at runtime supports the assertion that PHP is a loosely typed language.



You:

2) Describe sprintf() method of string formatting along with all its parameters.



ChatGPT:

The sprintf() function in PHP is used for formatting strings. It stands for "string print formatted." It works similarly to the printf() function but instead of outputting the formatted string directly, sprintf() returns the formatted string.

Here is the general syntax of the sprintf() function:

```
php
sprintf(format, arg1, arg2, ...)
```

- **format:** A string that specifies the format of the resulting string. It can contain format specifiers that begin with a percent sign (%), which are replaced by the values specified in subsequent arguments.
- **arg1, arg2, ...:** Optional arguments that are inserted into the format string. These can be variables, constants, or expressions.

The sprintf() function supports various format specifiers, each of which corresponds to a specific data type. Some common format specifiers include:

- %s: String
- %d or %i: Signed decimal number
- %f: Floating-point number
- %c: Single character
- %b: Binary representation of a number
- %o: Octal representation of a number
- %x or %X: Hexadecimal representation of a number

Here's an example illustrating the use of sprintf():

```
php
```

```
$name = "John";
$age = 25;

// Using sprintf to format a string
$formattedString = sprintf("Hello, my name is %s and I am %d years old.", $name, $age);

echo $formattedString;
```

In this example, the %s and %d format specifiers are used to insert the values of \$name and \$age into the format string, resulting in the formatted string "Hello, my name is John and I am 25 years old."

It's important to note that sprintf() does not output the formatted string; it returns it. If you want to directly output the formatted string, you can use printf() instead.



You:

3) Write a PHP script to demonstrate various array operators for following arrays: \$a = array("a" => "apple", "b" => "banana"); \$b = array("a" => "pear", "b" => "strawberry", "c" => "cherry");



ChatGPT:

Certainly! Below is a PHP script that demonstrates various array operators using the given arrays \$a and \$b:

```
php
<?php

// Given arrays
$a = array("a" => "apple", "b" => "banana");
$b = array("a" => "pear", "b" => "strawberry", "c" => "cherry");

// Union operator ( + )
$union = $a + $b;
echo "Union of arrays:\n";
print_r($union);
echo "\n";

// Equality operator ( == )
if ($a == $b) {
    echo "Arrays are equal.\n";
} else {
    echo "Arrays are not equal.\n";
}

// Identity operator ( === )
if ($a === $b) {
    echo "Arrays are identical.\n";
} else {
    echo "Arrays are not identical.\n";
}

// Difference operator ( != or <> )
$difference = $a != $b ? "Arrays are different.\n" : "Arrays are not different.\n";
echo $difference;

// Merge operator (array_merge)
$merged = array_merge($a, $b);
echo "Merged array:\n";
print_r($merged);
echo "\n";

// Difference operator (array_diff)
$diff = array_diff($a, $b);
echo "Difference between arrays:\n";
print_r($diff);
echo "\n";

// Intersection operator (array_intersect)
$intersection = array_intersect($a, $b);
echo "Intersection of arrays:\n";
print_r($intersection);
echo "\n";
?>
```

This script showcases various array operators, such as the union operator (+), equality operator (==), identity operator (===), difference operators (!= or <>), and the array functions array_merge, array_diff, and array_intersect. The output will demonstrate the results of each operation on the

given arrays \$a and \$b.



You:

4) Enlist any six string manipulation functions of PHP and explain three of them with appropriate examples.



ChatGPT:

PHP provides a variety of string manipulation functions to perform operations on strings. Here are six common string manipulation functions in PHP:

1. **strlen()** - Returns the length of a string.

```
php
$str = "Hello, World!";
$length = strlen($str);
echo "Length of the string: $length"; // Output: 13
```

2. **strpos()** - Finds the position of the first occurrence of a substring in a string.

```
php
$str = "Hello, World!";
$pos = strpos($str, "World");
echo "Position of 'World': $pos"; // Output: 7
```

3. **substr()** - Returns a portion of a string.

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

4. **str_replace()** - Replaces all occurrences of a substring with another string.

```
php
$str = "Hello, World!";
$newStr = str_replace("World", "PHP", $str);
echo "Modified String: $newStr"; // Output: Hello, PHP!
```

5. **strtolower()** - Converts a string to lowercase.

```
php
$str = "Hello, World!";
$lowercase = strtolower($str);
echo "Lowercase: $lowercase"; // Output: hello, world!
```

6. **str_repeat()** - Repeats a string a specified number of times.

```
php
$str = "Hello, ";
$repeatedStr = str_repeat($str, 3);
echo "Repeated String: $repeatedStr"; // Output: Hello, Hello, Hello,
```

Explanation of three functions:

- **strlen()**: This function returns the length of the given string. In the example, the length of the string "Hello, World!" is calculated using `strlen()`.
- **strpos()**: It finds the position of the first occurrence of a substring within a string. In the example, the position of the substring "World" in the string "Hello, World!" is determined using `strpos()`.
- **substr()**: This function returns a portion of a string based on the specified starting position and length. In the example, a substring starting from position 7 and of length 5 is extracted from the string "Hello, World!" using `substr()`.



You:

Explain the output of following code snippet: • <?php \$price = 2678.996; printf("\$%12.2f\n", \$price); ?> • <?php printf("%10.2f\n", 3.1); printf("%11.3f\n", 30.1); printf("%12.4f\n", 300.1); ?> • <?php \$num = 126; \$course = 'CSE'; \$year = 2021; \$format = 'There are %2\$d students in %1\$s batch in the %3\$d year'; echo sprintf(\$format, \$course, \$num, \$year); ?>



Let's go through each code snippet and explain the output:

1. Code Snippet 1:

```
php
<?php
$price = 2678.996;
printf("\$%12.2f\n\n", $price);
?>
```

Output:

```
ruby
$      2678.99
```

Explanation: In the `printf` statement, the format specifier `%12.2f` is used, which means a floating-point number with a width of 12 characters and 2 decimal places. The value of `$price` is then formatted accordingly, resulting in the output `$2678.99`.

2. Code Snippet 2:

```
php
<?php
printf("%10.2f\n", 3.1);
printf("%11.3f\n", 30.1);
printf("%12.4f\n", 300.1);
?>
```

Output:

```
markdown
      3.10
     30.100
    300.1000
```

Explanation: In each `printf` statement, a floating-point number is formatted with a specific width and precision. The `%10.2f` specifier results in a width of 10 characters and 2 decimal places. Similarly, `%11.3f` and `%12.4f` specify 3 decimal places and 4 decimal places, respectively.

3. Code Snippet 3:

```
php
<?php
$num = 126;
$course = 'CSE';
$year = 2021;
$format = 'There are %2$d students in %1$s batch in the %3$d year';
echo sprintf($format, $course, $num, $year);
?>
```

Output:

```
sql
There are 126 students in CSE batch in the 2021 year
```

Explanation: The `sprintf` function is used to format a string based on the format specifier provided in `$format`. The `%2$d`, `%1$s`, and `%3$d` indicate placeholders for the second argument as an integer, the first argument as a string, and the third argument as an integer, respectively. The values are then substituted in the specified order, resulting in the output "There are 126 students in CSE batch in the 2021 year."



You:

6) Write and explain a demonstrative PHP script to generate array variables for indexed and associative arrays.



Certainly! Below is a PHP script that demonstrates the creation of both indexed and associative arrays. The script initializes and prints both types of arrays, and explanations are provided for each part of the code:

```
php
<?php

// Demonstrative PHP script for generating indexed and associative arrays

// Indexed Array
$colorsIndexed = array("Red", "Green", "Blue", "Yellow", "Orange");

// Print and display the indexed array
echo "Indexed Array:\n";
print_r($colorsIndexed);
echo "\n";

// Associative Array
$colorsAssociative = array(
    "R" => "Red",
    "G" => "Green",
    "B" => "Blue",
    "Y" => "Yellow",
    "O" => "Orange"
);

// Print and display the associative array
echo "Associative Array:\n";
print_r($colorsAssociative);
echo "\n";

?>
```

Explanation:

1. Indexed Array:

```
php
$colorsIndexed = array("Red", "Green", "Blue", "Yellow", "Orange");
```

This line creates an indexed array named `$colorsIndexed` with elements "Red," "Green," "Blue," "Yellow," and "Orange." Indexed arrays have numeric keys automatically assigned starting from 0.

2. Print and display the indexed array:

```
php
echo "Indexed Array:\n";
print_r($colorsIndexed);
echo "\n";
```

This part prints the label "Indexed Array," followed by the contents of the `$colorsIndexed` array using `print_r()`. The output will show the array elements and their corresponding numeric indices.

3. Associative Array:

```
php
$colorsAssociative = array(
    "R" => "Red",
    "G" => "Green",
    "B" => "Blue",
    "Y" => "Yellow",
    "O" => "Orange"
);
```

This line creates an associative array named `$colorsAssociative` with explicit key-value pairs. The keys ("R," "G," "B," "Y," "O") are associated with their corresponding color values.

4. Print and display the associative array:

```
php
echo "Associative Array:\n";
print_r($colorsAssociative);
echo "\n";
```

This part prints the label "Associative Array," followed by the contents of the `$colorsAssociative` array using `print_r()`. The output will show the array elements and their corresponding keys and values.

When you run this script, you'll see the output displaying both the indexed and associative arrays. The indexed array will show numeric indices, while the associative array will show the specified keys associated with their values.



You:

7) Write and explain a PHP script to demonstrate array-handling operators.



ChatGPT:

Certainly! Below is a PHP script that demonstrates the use of various array-handling operators. The script initializes two arrays and performs operations using different array operators:

```
php
<?php

// PHP script to demonstrate array-handling operators

// Initializing arrays
$array1 = array("a" => "apple", "b" => "banana", "c" => "cherry");
$array2 = array("d" => "date", "e" => "elderberry", "f" => "fig");

// Displaying initial arrays
echo "Array 1:\n";
print_r($array1);
echo "\n";

echo "Array 2:\n";
print_r($array2);
echo "\n";

// Using the + operator (Union)
$unionResult = $array1 + $array2;
echo "Union of arrays (Array 1 + Array 2):\n";
print_r($unionResult);
echo "\n";

// Using the == operator (Equality)
$equalityResult = ($array1 == $array2) ? "Arrays are equal." : "Arrays are not equal.";
echo "Equality of arrays:\n";
echo $equalityResult . "\n";

// Using the === operator (Identity)
$identityResult = ($array1 === $array2) ? "Arrays are identical." : "Arrays are not identical.";
echo "Identity of arrays:\n";
echo $identityResult . "\n";

// Using the != or <> operator (Inequality)
$inequalityResult = ($array1 != $array2) ? "Arrays are different." : "Arrays are not different.";
echo "Inequality of arrays:\n";
echo $inequalityResult . "\n";

// Using the array_merge function
$mergedArray = array_merge($array1, $array2);
echo "Merged array (using array_merge):\n";
print_r($mergedArray);
echo "\n";

?>
```

Explanation:

1. Initializing Arrays:

```
php
$array1 = array("a" => "apple", "b" => "banana", "c" => "cherry");
$array2 = array("d" => "date", "e" => "elderberry", "f" => "fig");
```

Two associative arrays, \$array1 and \$array2, are initialized with different key-value pairs.

2. Union of Arrays (Using the + Operator):

```
php
$unionResult = $array1 + $array2;
```

The + operator is used to perform the union of the arrays. If keys in \$array1 are not present in \$array2, they are added to the result.

3. Equality of Arrays (Using == Operator):

```
php
$equalityResult = ($array1 == $array2) ? "Arrays are equal." : "Arrays are not equal.";
```

The == operator checks if the arrays are equal, i.e., they have the same key-value pairs, regardless of the order.

4. Identity of Arrays (Using === Operator):

```
php
$identityResult = ($array1 === $array2) ? "Arrays are identical." : "Arrays are not identical.";
```

The === operator checks if the arrays are identical, i.e., they have the same key-value pairs in the same order.

5. Inequality of Arrays (Using != or <> Operator):

```
php
$inequalityResult = ($array1 != $array2) ? "Arrays are different." : "Arrays are not different.";
```

The != or <> operator checks if the arrays are different, i.e., they have at least one different key-value pair.

6. Merging Arrays (Using array_merge Function):

```
php
$mergedArray = array_merge($array1, $array2);
```

The array_merge function combines the elements of two arrays. If the arrays have the same keys, the later values overwrite the earlier values.

When you run this script, you will see the output demonstrating the results of various array-handling operations.



You:

8) Enlist any six array manipulation functions of PHP and explain any three of them with appropriate examples.



ChatGPT:

Certainly! PHP provides a variety of array manipulation functions. Here are six common ones, along with explanations for three of them:

1. array_push() - Adds one or more elements to the end of an array:

```
php
$fruits = array("apple", "banana");
array_push($fruits, "orange", "grape");
print_r($fruits);
// Output: Array ( [0] => apple [1] => banana [2] => orange [3] => grape )
```

2. array_pop() - Removes and returns the last element from an array:

```
php
$fruits = array("apple", "banana", "orange", "grape");
$lastFruit = array_pop($fruits);
echo "Last fruit: $lastFruit\n";
// Output: Last fruit: grape
```

3. array_merge() - Merges two or more arrays into one:

```
php
$array1 = array("a" => "apple", "b" => "banana");
$array2 = array("c" => "cherry", "d" => "date");
$mergedArray = array_merge($array1, $array2);
print_r($mergedArray);
// Output: Array ( [a] => apple [b] => banana [c] => cherry [d] => date )
```

4. array_slice() - Extracts a portion of an array:

```
php
$numbers = array(1, 2, 3, 4, 5);
$subset = array_slice($numbers, 2, 2);
print_r($subset);
// Output: Array ( [0] => 3 [1] => 4 )
```

5. **array_reverse()** - Reverses the order of elements in an array:

```
php
$colors = array("red", "green", "blue");
$reversedColors = array_reverse($colors);
print_r($reversedColors);
// Output: Array ( [0] => blue [1] => green [2] => red )
```

6. **array_unique()** - Removes duplicate values from an array:

```
php
$numbers = array(1, 2, 3, 2, 4, 5, 1);
$uniqueNumbers = array_unique($numbers);
print_r($uniqueNumbers);
// Output: Array ( [0] => 1 [1] => 2 [2] => 3 [4] => 4 [5] => 5 )
```

Explanation of three functions:

1. array_push():

- `array_push($fruits, "orange", "grape");` adds the elements "orange" and "grape" to the end of the `$fruits` array.

2. array_pop():

- `array_pop($fruits);` removes and returns the last element of the `$fruits` array, which is "grape" in this case.

3. array_merge():

- `$mergedArray = array_merge($array1, $array2);` combines the elements of `$array1` and `$array2` into a new array, where keys are preserved.

These functions provide flexibility and ease in manipulating arrays for various tasks in PHP.



You:

9) Write a PHP script to create a function which will accept below mentioned string as an argument and will return the given output. "I am <i>taking</i> exam of,
Web Designing<p>at UTU." Output: I am taking exam of,
Web Designing<p>at UTU.



ChatGPT:

Certainly! You can create a PHP function to remove specific HTML tags from a given string using the `strip_tags` function. Here's a script for your requirement:

```
php
<?php

function removeTags($inputString) {
    // Specify the allowed tags to keep in the string
    $allowedTags = '<br><p><b>';

    // Use strip_tags to remove unwanted tags
    $outputString = strip_tags($inputString, $allowedTags);

    return $outputString;
}

// Input string
$inputString = "I am <i>taking</i> exam of,<br>Web Designing<p><b>at UTU.</b>";

// Call the function and display the output
$outputString = removeTags($inputString);
echo $outputString;

?>
```

Explanation:

1. The `removeTags` function accepts a string as an argument.
2. Inside the function, the `$allowedTags` variable is set to specify the tags that should not be removed.
3. The `strip_tags` function is used to remove all HTML tags from the input string except the allowed tags specified in `$allowedTags`.
4. The cleaned string is then returned.

When you run this script, it will output the string without the `<i>` and `` tags:

```
css
```


I am taking exam of,
Web Designing<p>at UTU.

This way, you can easily create a reusable function to sanitize or remove specific HTML tags from strings in PHP.



10) Enlist and explain various ways of passing arguments to a function in PHP using an example.



In PHP, you can pass arguments to a function in various ways: by value, by reference, and by using the ... (ellipsis) operator for variable-length argument lists. Here are examples illustrating each method:

1. Pass by Value:

In pass by value, the value of the variable is passed to the function. Any changes made to the parameter inside the function do not affect the original variable.

```
php
<?php

function square($num) {
    return $num * $num;
}

$value = 5;
$result = square($value);

echo "Original value: $value\n";    // Output: Original value: 5
echo "Square: $result\n";          // Output: Square: 25

?>
```

2. Pass by Reference:

In pass by reference, a reference to the variable is passed to the function using the & symbol. Changes made to the parameter inside the function affect the original variable.

```
php
<?php

function squareByReference(&$num) {
    $num = $num * $num;
}

$value = 5;
squareByReference($value);

echo "Original value: $value\n";    // Output: Original value: 25

?>
```

3. Variable-Length Argument Lists:

Using the ... (ellipsis) operator, you can pass a variable number of arguments to a function. The arguments are treated as an array inside the function.

```
php
<?php

function sum(...$numbers) {
    return array_sum($numbers);
}

$result = sum(2, 4, 6, 8);

echo "Sum: $result\n";    // Output: Sum: 20

?>
```

4. Default Values:

You can provide default values for function parameters. If a value is not passed during the function call, the default value is used.

```
php
<?php

function greet($name = "Guest") {
    echo "Hello, $name!\n";
}

greet();           // Output: Hello, Guest!
greet("John");     // Output: Hello, John!

?>
```

5. Mixed Usage:

You can use a combination of pass by value, pass by reference, and variable-length argument lists in a single function.

```
php
<?php

function manipulateValues($a, &$b, ...$rest) {
    $a *= 2;
    $b *= 3;
    $rest = array_map(function($num) { return $num * 10; }, $rest);

    return [$a, $b, $rest];
}

$x = 2;
$y = 5;
$z = 3;

$result = manipulateValues($x, $y, $z, 4, 6);

print_r($result);
// Output: Array ( [0] => 4 [1] => 15 [2] => Array ( [0] => 30 [1] => 50 ) )

?>
```

In this example, \$a is passed by value, \$b by reference, and the remaining arguments by using the ... operator for a variable-length list. The function manipulates these values, and the result is returned as an array.



You:

11) Explain following string functions using appropriate example. a.str_pad() b.str_split() c.str_replace()



ChatGPT:

Certainly! Let's go through each of the string functions you've mentioned with appropriate examples:

a. str_pad()

The str_pad() function is used to pad a string to a certain length with another string. It can add padding on the left, right, or both sides of the original string.

Example:

```
php
<?php

$input = "Hello";
$paddedString = str_pad($input, 10, "*");

echo "Original String: $input\n";
echo "Padded String: $paddedString\n";

?>
```

Output:

```
arduino
```

```
Original String: Hello
Padded String: Hello*****
```

In this example, the original string "Hello" is padded with asterisks (*) on the right side to make its total length 10 characters.

b. str_split()

The str_split() function is used to split a string into an array of substrings, each consisting of a specified number of characters.

Example:

```
php
<?php

$input = "PHP";
$splitArray = str_split($input);

echo "Original String: $input\n";
echo "Split Array: ";
print_r($splitArray);

?>
```

Output:

```
mathematica
Original String: PHP
Split Array: Array ( [0] => P [1] => H [2] => P )
```

In this example, the original string "PHP" is split into an array where each element corresponds to a single character of the string.

c. str_replace()

The str_replace() function is used to replace occurrences of a search string with another string in a given string.

Example:

```
php
<?php

$input = "Hello, World!";
$replacement = "Universe";
$modifiedString = str_replace("World", $replacement, $input);

echo "Original String: $input\n";
echo "Modified String: $modifiedString\n";

?>
```

Output:

```
arduino
Original String: Hello, World!
Modified String: Hello, Universe!
```

In this example, the original string "Hello, World!" has the substring "World" replaced with the string "Universe," resulting in the modified string "Hello, Universe!".

These string functions are useful for various string manipulation tasks in PHP.



12) Write a PHP script to perform ascending order and descending ordersorting of indexed and associative array.



Certainly! Here's a PHP script that demonstrates ascending and descending order sorting for both indexed and associative arrays:

```
php
<?php

// Indexed Array
```

```

$numbersIndexed = array(5, 2, 8, 1, 7);

echo "Original Indexed Array:\n";
print_r($numbersIndexed);

// Ascending order sort for Indexed Array
sort($numbersIndexed);
echo "\nAscending Order Sorted Indexed Array:\n";
print_r($numbersIndexed);

// Descending order sort for Indexed Array
rsort($numbersIndexed);
echo "\nDescending Order Sorted Indexed Array:\n";
print_r($numbersIndexed);

// Associative Array
$fruitsAssociative = array(
    "a" => "apple",
    "b" => "banana",
    "c" => "cherry",
    "d" => "date"
);

echo "\nOriginal Associative Array:\n";
print_r($fruitsAssociative);

// Ascending order sort for Associative Array by values
asort($fruitsAssociative);
echo "\nAscending Order Sorted Associative Array by Values:\n";
print_r($fruitsAssociative);

// Descending order sort for Associative Array by values
arsort($fruitsAssociative);
echo "\nDescending Order Sorted Associative Array by Values:\n";
print_r($fruitsAssociative);

// Ascending order sort for Associative Array by keys
ksort($fruitsAssociative);
echo "\nAscending Order Sorted Associative Array by Keys:\n";
print_r($fruitsAssociative);

// Descending order sort for Associative Array by keys
krsort($fruitsAssociative);
echo "\nDescending Order Sorted Associative Array by Keys:\n";
print_r($fruitsAssociative);

?>

```

Explanation:

1. For indexed arrays, the `sort()` function is used for ascending order, and `rsort()` is used for descending order.
2. For associative arrays, `asort()` and `arsort()` are used for ascending and descending order sorting by values, respectively.
3. For sorting associative arrays by keys, `ksort()` and `krsort()` are used for ascending and descending order, respectively.

When you run this script, you'll see the original arrays followed by their ascending and descending order sorted versions for both indexed and associative arrays.



13) Explain types of arrays Indexed arrays , Associative arrays & Multidimensional arrays supported by PHP in detail with example



In PHP, arrays are used to store and manage collections of data. There are three main types of arrays: Indexed arrays, Associative arrays, and Multidimensional arrays.

1. Indexed Arrays:

Indexed arrays are arrays where each element is assigned a numeric index starting from 0.

Example:

```

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

```

In this example, the array `$colors` is an indexed array with numeric indices. Elements can be accessed using their index, like `$colors[0]` for "Red."

2. Associative Arrays:

Associative arrays use named keys to index elements, allowing you to create more meaningful indices.

Example:

```
php
$person = array("name" => "John", "age" => 25, "city" => "New York");
```

In this example, the array `$person` is an associative array where each element has a key ("name", "age", "city") associated with its value. You access elements using their keys, like `$person["name"]` for "John."

3. Multidimensional Arrays:

Multidimensional arrays are arrays that contain other arrays. They can be nested to create more complex data structures.

Example:

```
php
$matrix = array(
    array(1, 2, 3),
    array(4, 5, 6),
    array(7, 8, 9)
);
```

In this example, the array `$matrix` is a 2D array (two-dimensional array) representing a matrix. Elements are accessed using two indices, like `$matrix[1][2]` for the value 6.

Example of Associative Multidimensional Array:

```
php
$students = array(
    "Alice" => array("age" => 22, "grade" => "A"),
    "Bob" => array("age" => 20, "grade" => "B"),
    "Charlie" => array("age" => 21, "grade" => "C")
);
```

Here, the `$students` array is an associative multidimensional array. Each student's information is an associative array nested within the main array. Accessing data is done like `$students["Alice"]["age"]` to get the age of Alice.

These types of arrays offer flexibility in organizing and manipulating data in PHP. You can use them based on the specific requirements of your application.



14) Discuss the importance of `declare(strict_types=1)` in PHP and state the output of following code snippet. `<?php declare(strict_types=1); function sum(int $a, $b) { return $a + $b; } echo sum(5, "5 days")."\n\n"; function mul(int $a, int $b) { return $a * $b; } echo mul(5, "5 days"); ?>`



The `declare(strict_types=1);` declaration in PHP is used to enforce strict typing for scalar type declarations (int, float, bool) in function and method parameters. When strict typing is enabled, PHP will check the types of function arguments at runtime, and a `TypeError` will be thrown if the types don't match.

In the given code snippet:

```
php
<?php declare(strict_types=1);

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

echo sum(5, "5 days")."\n\n";

function mul(int $a, int $b) {
    return $a * $b;
}

echo mul(5, "5 days");
?>
```

1. sum() function:

- The `sum()` function has strict typing disabled for the second parameter (`$b`), so it will not enforce strict type checking.
- When calling `sum(5, "5 days")`, the string "5 days" is cast to an integer (0 in this case) because strict typing is not enforced.
- The result is `5 + 0`, which is 5.
- The output of `echo sum(5, "5 days")."\n\n";` is 5.

2. mul() function:

- The `mul()` function has strict typing enabled for both parameters (`$a` and `$b`), so it will enforce strict type checking.
- When calling `mul(5, "5 days")`, a `TypeError` will be thrown because the second argument is a string, and strict type checking is enabled.
- The script will terminate execution at this point, and the second `echo` statement will not be reached.

Output:

```
go
5

Fatal error: Uncaught TypeError: Argument 2 passed to mul() must be of the type int, string given in ...:...
```

The `strict_types` declaration is important for writing more robust and predictable code by catching type-related errors during development rather than at runtime. It helps ensure that the types passed to functions and methods match the expected types, reducing the likelihood of unexpected behavior and making the code more maintainable.



You:

15) Define `array_diff()` and `array_merge()` functions. State the output of following code snippet. `<?php $pudding = array('raspberry'=>111, 'peach'=>222, 'chocolate'=>333); $icecream = array('raspberry'=>444, 'peach'=>555, 'chocolate'=>666); print_r(array_diff($icecream, $pudding)); print_r(array_diff($pudding, $icecream)); print_r(array_merge($pudding, $icecream)); print_r(array_merge($icecream, $pudding)); ?>`



ChatGPT:

1. array_diff():

- The `array_diff()` function returns the difference between two arrays, i.e., it returns an array containing all the values from the first array that are not present in any of the other arrays.

2. array_merge():

- The `array_merge()` function merges one or more arrays into a single array. If two or more array elements have the same key, the last one will override the previous ones.

Now, let's analyze the given code snippet:

```
php
<?php
$pudding = array('raspberry'=>111, 'peach'=>222, 'chocolate'=>333);
$icecream = array('raspberry'=>444, 'peach'=>555, 'chocolate'=>666);

print_r(array_diff($icecream, $pudding));
print_r(array_diff($pudding, $icecream));
print_r(array_merge($pudding, $icecream));
print_r(array_merge($icecream, $pudding));
?>
```

Output:

```
csharp
Array
(
    [raspberry] => 444
    [peach] => 555
    [chocolate] => 666
)
Array
(
    [raspberry] => 111
    [peach] => 222
    [chocolate] => 333
)
Array
(
    [raspberry] => 111
    [peach] => 222
    [chocolate] => 333
)
```

```

[raspberry] => 444
[peach] => 555
[chocolate] => 666
)
Array
(
    [raspberry] => 444
    [peach] => 555
    [chocolate] => 666
    [raspberry] => 111
    [peach] => 222
    [chocolate] => 333
)

```

Explanation:

1. `array_diff($icecream, $pudding)`: Returns an array containing the elements from `$icecream` that are not present in `$pudding`. The output has the values from `$icecream` that are not present in `$pudding`.
2. `array_diff($pudding, $icecream)`: Returns an array containing the elements from `$pudding` that are not present in `$icecream`. The output has the values from `$pudding` that are not present in `$icecream`.
3. `array_merge($pudding, $icecream)`: Merges the arrays `$pudding` and `$icecream`. If there are duplicate keys, the values from `$icecream` overwrite the values from `$pudding` for the same keys. The output shows all the merged elements.
4. `array_merge($icecream, $pudding)`: Similar to the previous case, but this time, values from `$pudding` overwrite values from `$icecream` for the same keys. The output shows all the merged elements.



You:

16) Explain following array functions using appropriate example. a. `array_chunk()` b. `array_search()` c. `array_intersect()`



ChatGPT:

Certainly! Let's go through each array function with examples:

a. array_chunk()

The `array_chunk()` function is used to split an array into chunks of a specified size. It returns a multidimensional array containing these chunks.

Example:

```

php
<?php

$numbers = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

// Splitting the array into chunks of size 3
$chunks = array_chunk($numbers, 3);

print_r($chunks);

?>

```

Output:

```

css
Array
(
    [0] => Array ( [0] => 1 [1] => 2 [2] => 3 )
    [1] => Array ( [0] => 4 [1] => 5 [2] => 6 )
    [2] => Array ( [0] => 7 [1] => 8 [2] => 9 )
    [3] => Array ( [0] => 10 )
)

```

In this example, the array `$numbers` is split into chunks of size 3 using `array_chunk()`, resulting in a multidimensional array containing four chunks.

b. array_search()

The `array_search()` function is used to search for a value in an array. It returns the corresponding key if the value is found; otherwise, it returns `false`.

Example:

```
php
<?php

$fruits = array("apple", "banana", "cherry", "date");

// Searching for the key of "cherry"
$key = array_search("cherry", $fruits);

if ($key !== false) {
    echo "Key found: $key\n";
} else {
    echo "Key not found.\n";
}

?>
```

Output:

```
mathematica
Key found: 2
```

In this example, the `array_search()` function is used to find the key of the value "cherry" in the array `$fruits`, and it returns the key 2.

c. array_intersect()

The `array_intersect()` function is used to find the intersection of two or more arrays. It returns an array containing the values that are present in all input arrays.

Example:

```
php
<?php

$array1 = array("a", "b", "c", "d");
$array2 = array("b", "d", "e", "f");

// Finding the intersection of two arrays
$intersection = array_intersect($array1, $array2);

print_r($intersection);

?>
```

Output:

```
css
Array
(
    [1] => b
    [3] => d
)
```

In this example, `array_intersect()` is used to find the common elements between `$array1` and `$array2`, resulting in an array containing the values "b" and "d," which are present in both arrays.

These array functions provide useful tools for manipulating and searching arrays in PHP.



You:

17) Write a PHP script to do the following: a. Create a function which will take a string as an argument and will return an array b. Create a function which will take an array as an argument and will return a string.



ChatGPT:

Certainly! Below is a PHP script that demonstrates the creation of two functions:

a. Create a function which will take a string as an argument and will return an array:

```
php
<?php

function stringToArray($inputString) {
```



```
// Use str_split to convert the string into an array of characters
$charArray = str_split($inputString);
return $charArray;
}

// Example usage:
$inputString = "Hello, World!";
$resultArray = stringToArray($inputString);

// Display the result
echo "Original String: $inputString\n";
echo "Array: ";
print_r($resultArray);

?>
```

Output:

```
javascript
Original String: Hello, World!
Array: Array ( [0] => H [1] => e [2] => l [3] => l [4] => o [5] => , [6] => [7] => W [8] => o [9] => r [10] => l [11] => ! )
```

b. Create a function which will take an array as an argument and will return a string:

```
php
<?php

function arrayToString($inputArray) {
    // Use implode to join the array elements into a string
    $resultString = implode("", $inputArray);
    return $resultString;
}

// Example usage:
$inputArray = array("H", "e", "l", "l", "o");
$resultString = arrayToString($inputArray);

// Display the result
echo "Array: ";
print_r($inputArray);
echo "Result String: $resultString\n";

?>
```

Output:

```
javascript
Array: Array ( [0] => H [1] => e [2] => l [3] => l [4] => o )
Result String: Hello
```

In these examples, the `stringToArray()` function converts a string into an array of characters, and the `arrayToString()` function converts an array of characters into a string. These functions can be useful for various tasks involving string and array manipulation.



You:

18) Write a PHP script to count total number of alphabets, digits and special characters in a string.



ChatGPT:

Certainly! Here's a PHP script that counts the total number of alphabets, digits, and special characters in a given string:

```
php
<?php

function countCharacters($inputString) {
    $result = array(
        'alphabets' => 0,
        'digits' => 0,
        'specialCharacters' => 0
    );

    // Loop through each character in the string
```

```
for ($i = 0; $i < strlen($inputString); $i++) {
    $char = $inputString[$i];

    // Check if the character is an alphabet
    if (ctype_alpha($char)) {
        $result['alphabets']++;
    }
    // Check if the character is a digit
    elseif (ctype_digit($char)) {
        $result['digits']++;
    }
    // Check if the character is a special character
    else {
        $result['specialCharacters']++;
    }
}

return $result;
}

// Example usage:
$inputString = "Hello123!@";

// Count characters
$result = countCharacters($inputString);

// Display the result
echo "Original String: $inputString\n";
echo "Alphabets: " . $result['alphabets'] . "\n";
echo "Digits: " . $result['digits'] . "\n";
echo "Special Characters: " . $result['specialCharacters'] . "\n";

?>
```

Output:

```
yaml
Original String: Hello123!@
Alphabets: 5
Digits: 3
Special Characters: 2
```

In this script, the `countCharacters()` function takes a string as input and uses a loop to iterate through each character. The `ctype_alpha()` and `ctype_digit()` functions are then used to check if the character is an alphabet or a digit. If neither condition is met, the character is considered a special character. The counts are then returned in an associative array. The example usage demonstrates how to count alphabets, digits, and special characters in a given string.