PHP PROBLEMS AND SOLUTIONS BASED ON W3SCHOOL PHP TUTORIALS

DEFINITION : PHP is a server-side scripting language designed for web development, but it can also be used for general-purpose programming. It is a dynamic and loosely typed language that is commonly used to build dynamic web applications and websites. PHP is embedded within HTML code and executed on the server, generating HTML content that is then sent to the client's web browser. PHP is considered a loosely typed language because it does not require explicit declaration of data types for variables. In other words, you don't need to specify the type of data a variable will hold when you create it. Instead, PHP determines the data type of a variable dynamically based on the value assigned to it at runtime.

For example, in a strongly typed language like C++ or Java, you must declare the data type of a variable explicitly:

```
int age = 30; // Explicitly declaring age as an integer
```

In PHP, you can simply assign a value to a variable without specifying its data type:

```
$age = 30; // No need to declare the data type of $age
```

The variable `$age` is automatically considered an integer because of the value assigned to it. However, you can later change the data type of a variable by assigning it a value of a different type:

```
$age = "thirty"; // Now $age is a string
```

This flexibility can make PHP more forgiving and easier to use for beginners, but it can also lead to unexpected behavior if you're not careful with your variable assignments. While PHP is loosely typed, it's essential to be aware of the data types of your variables and ensure they are used correctly in your code to prevent errors.

Variables ($), Case Sensitivity, Comments, echo & Print

Certainly! Here are ten problems and solutions related to the topics of "PHP Case Sensitivity," "PHP Comments," "PHP Variables ($)," and "PHP echo Statement." Each problem includes a brief explanation and a code example. These are suitable for a PHP developer's interview test or remote job application:

Problem 1 - PHP Case Sensitivity: Write a PHP script to check if two variable names are the same but with different capitalization (e.g., `$myVar` and `$myvar`) and return a boolean result indicating whether they are equal without considering case sensitivity.

```
 $var1 = "myVar";
$var2 = "MYVAR";
$result = strcasecmp($var1, $var2) == 0;
echo $result ? "Variables are equal" : "Variables are not equal";
```

Problem 2 - PHP Comments: Create a PHP script that includes both single-line and multi-line comments to explain the purpose of the script. Then, output a message using `echo` to demonstrate how comments do not affect the script's functionality.

```
 // This is a single-line comment
/*
   This is a multi-line comment
   Used for explaining code.
*/
echo "Hello, PHP!";
```

Problem 3 - PHP Variables ($): Write a PHP function that accepts two arguments, a variable name (as a string) and a value. Create a variable with the provided name and assign the value to it. Then, echo the value of the variable.

```
 function createAndPrintVariable($varName, $value) {
    $$varName = $value;
    echo $$varName;
}

createAndPrintVariable("myVar", "Hello, World!");
// Hello, World!
```

Problem 4 - PHP echo Statement: Create a PHP script that uses the `echo` statement to display a series of numbers from 1 to 10 in a list format.

```
for ($i = 1; $i <= 10; $i++) {
    echo "<li>$i</li>";
}
```

data types, variables, and the difference between echo and print

---

Certainly! Let's dive into the details of the `echo` and `print` statements in PHP, along with examples.

`echo` Statement: The `echo` statement is a language construct in PHP that is used to output one or more strings. It's commonly used for displaying content on a web page. Here are some key points about `echo`:

- `echo` and print can output multiple values separated by commas. but echo is more commonly used for this purpose in practice.
- It does not have a return value; it simply displays content.
- `echo` is slightly faster than `print`.

Example 1 - Using `echo`:

```
 $greeting = "Hello,";
$name = "John";
echo $greeting, " ", $name; // Output: Hello, John
```

`print` Statement: The `print` statement is also used to display content in PHP. Like `echo`, it is used for outputting strings. Here are some key points about `print`:

- `print` can only output a single value (string) at a time.
- It returns a value of 1, which can be used in expressions.
- `print` is marginally slower than `echo`.

Example 2 - Using `print`:

```
 $greeting = "Hello,";
$name = "John";
print $greeting . " " . $name; // Output: Hello, John
```

Comparison:

- Both `echo` and `print` are commonly used for displaying content to users.
- `echo` is often preferred for its speed and flexibility when outputting multiple values.
- `print` can be useful when you want to use its return value in expressions, although this is less common.

Example 3 - Using `print` in an Expression:

```
 $var1 = 5;
$var2 = 3;
$result = print($var1) + $var2; // Output: 53
echo "Result: " . $result; // Output: Result: 53
```

Problem 1 - PHP Data Types: Define and provide examples of PHP's four scalar data types.

```
 $integer = 42;
$float = 3.14;
$string = "Hello, PHP!";
$boolean = true;
```

Problem 2 - PHP Variables: Create a PHP script that declares a variable, assigns an integer value to it, and then prints the variable's value.

```
 $myVar = 5;
echo $myVar;
```

Problem 3 - PHP Variable Scope: Explain the concept of variable scope in PHP. Create a PHP script with both local and global variables and demonstrate how they work.

```php
$globalVar = "I'm global";

function testScope() {
    $localVar = "I'm local";
    echo $localVar;
    global $globalVar;
    echo $globalVar;

    // I'm local
    // I'm global

}

testScope();
```

Problem 4 - PHP String Concatenation: Write a PHP function that concatenates two strings and returns the result. Then, call the function with example strings.

```php
function concatenateStrings($str1, $str2) {
    return $str1 . $str2;
}

$result = concatenateStrings("Hello, ", "PHP!");
echo $result;
```

Problem 5 - PHP Type Conversion: Explain implicit and explicit type conversion in PHP. Provide an example where you convert an integer to a string explicitly.

```
$original = "42";

echo "Original Type: " . gettype($original) . PHP_EOL;
// Original Type: string

// Explicit type casting
$intVal = (int)$original;
echo "Int Type: " . gettype($intVal) . ", Value: $intVal" . PHP_EOL;
// Int Type: integer, Value: 42

$floatVal = (float)$original;
echo "Float Type: " . gettype($floatVal) . ", Value: $floatVal" . PHP_EOL;
// Float Type: double, Value: 42

$boolVal = (bool)$original;
echo "Boolean Type: " . gettype($boolVal) . ", Value: " . ($boolVal ? 'true' : 'false') . PHP_EOL;
// Boolean Type: boolean, Value: true

$stringVal = (string)$original;
echo "String Type: " . gettype($stringVal) . ", Value: $stringVal" . PHP_EOL;
// String Type: string, Value: 42

// Automatic type conversion
$number = 42;
$stringNumber = "42";

$sum = $number + $stringNumber;
echo "Sum Type: " . gettype($sum) . ", Value: $sum" . PHP_EOL;
// Sum Type: integer, Value: 84

$concatenation = $number . $stringNumber;
echo "Concatenation Type: " . gettype($concatenation) . ", Value: $concatenation" . PHP_EOL;
// Concatenation Type: string, Value: 4242

$arrayConversion = (array)$original;
echo "Array Type: " . gettype($arrayConversion) . PHP_EOL;
// Array Type: array

$objectConversion = (object)$original;
echo "Object Type: " . gettype($objectConversion) . PHP_EOL;
// Object Type: object

`PHP_EOL` constant, which represents the end of a line character (either "\n" on Unix/Linux or "\r\n" on Windows).
```

Problem 6 - PHP `echo` vs `print`: Compare the `echo` and `print` statements in PHP in terms of usage and behavior. Provide an example that demonstrates their differences.

```
$var = "Hello, PHP!";
echo $var; // Output: Hello, PHP!
print $var; // Output: Hello, PHP!
```

Problem 7 - PHP Array Data Type: Define an indexed array in PHP with five elements, and then use a loop to print each element.

```
$fruits = array("Apple", "Banana", "Orange", "Grapes", "Cherry");
foreach ($fruits as $fruit) {
    echo $fruit . " ";
}
```

Problem 8 - PHP Associative Array: Create an associative array in PHP that represents a person's details (name, age, email). Then, print out one of the values using its corresponding key.

```
$person = array("name" => "John", "age" => 30, "email" => "john@example.com");
echo "Name: " . $person["name"];
```

Problem 9 - PHP Boolean Data Type: Define a boolean variable in PHP and use it in an `if` statement to check if it's true or false, then echo the result.

```
$isRaining = false;
if ($isRaining) {
    echo "Take an umbrella.";
} else {
    echo "Enjoy the sunshine!";
}
```

Problem 10 - PHP Null Data Type: Create a variable with a `null` value and check its type using the `is_null` function.

```
$nullVar = null;
if (is_null($nullVar)) {
    echo "The variable is null.";
} else {
    echo "The variable is not null.";
}
```

String & Its Methods

Certainly! Here are 20 medium-level problems and solutions related to strings in PHP, along with explanations and examples. These problems are suitable for PHP developer interviews and remote job applications, and they cover a range of string manipulation topics:

Problem 1 - Reverse a String: Write a PHP function to reverse a given string.

```
function reverseString($str) {
    return strrev($str);
}

$result = reverseString("Hello, PHP!");
echo $result; // Output: !PHP ,olleH
```

Problem 2 - Check Palindrome: Create a PHP function to check if a string is a palindrome (reads the same forwards and backward).

```
function isPalindrome($str) {
    $str = strtolower(str_replace(' ', '', $str));
    return $str === strrev($str);
}

$result = isPalindrome("A man a plan a canal Panama");
echo $result ? "Palindrome" : "Not a Palindrome"; // Output: Palindrome
```

Problem 3 - Count Words in a String: Write a PHP function to count the number of words in a given string.

```
function wordCount($words){

    $separate_words = explode(' ',$words);

    $words_array = [];
    echo count($separate_words) . '<br>';
    $limit_words = count($separate_words) > 6 ? array_slice($separate_words,0,5) : $separate_words;

    echo implode(' ', $limit_words);
}
wordCount("A Quick brown fox jumps over the lazy dog");
 // Output: 9
 // Output: A Quick brown fox jumps
```

Problem 4 - Remove Whitespace: Create a PHP function that removes all whitespace from a string.

```
function removeWhitespace($str) {
    return str_replace(' ', '', $str);
}

$result = removeWhitespace("Hello, PHP!");
echo $result; // Output: Hello,PHP!
```

Problem 5 - Replace Substring: Write a PHP function to replace a specific substring with another string in a given string.

```
function replaceSubstring($str, $search, $replace) {
    return str_replace($search, $replace, $str);
}

$result = replaceSubstring("Hello, world!", "world", "PHP");
echo $result; // Output: Hello, PHP!
```

Problem 6 - Extract User Details: Create a PHP function to extract and display all email addresses from a given string.

Simple example

```
// Input text
$text = "Contact us at contact@example.com or support@php.com. Please email john.doe@example.com for more information.";

// Regular expression pattern to match email addresses
$emailPattern = "/[\w\.-]+@[\w\.-]+/";

// Perform the regular expression match and store the results in the $matches array
preg_match_all($emailPattern, $text, $matches);

// $matches[0] contains all the matched email addresses
$emailAddresses = $matches[0];

// Print the extracted email addresses
foreach ($emailAddresses as $email) {
    echo $email . "<br>";
}
```

```
function extractUserDetails($str) {
    // Regular expression patterns to match numbers, usernames, names, and emails
    $numberPattern = "/\d+/"; //  `\d` matches any digit (0-9).
    $usernamePattern = "/@([a-zA-Z0-9_]+)/"; //`@` matches the "@" character literally.
    $namePattern = "/[A-Za-z]+ [A-Za-z]+/";
    $emailPattern = "/[\w\.-]+@[\w\.-]+/"; // matches one or more word characters (letters, digits, or underscores), dots, or hyphen

    preg_match_all($numberPattern, $str, $numbers);  // stores them in the $numbers array.
    preg_match_all($usernamePattern, $str, $usernames); // stores them in the $usernames array.
    preg_match_all($namePattern, $str, $names); // stores them in the $names array.
    preg_match_all($emailPattern, $str, $emails);  // stores them in the $emails array.

    $userDetails = [
        'numbers' => $numbers[0],
        'usernames' => $usernames[1], // Note the use of [1] to get the captured username part
        'names' => $names[0],
        'emails' => $emails[0],
    ];

// Optional. The variable used in this parameter will be populated with an array containing all of the matches that were found

    return $userDetails;
}

$string = "Contact user123 at contact@example.com or support@php.com. John Doe's email is john.doe@example.com.";
$userDetails = extractUserDetails($string);
print_r($userDetails);

Array (
    [numbers] => Array ( [0] => 123 )
    [usernames] => Array ( [0] => user123 )
    [names] => Array ( [0] => John Doe )
    [emails] => Array ( [0] => contact@example.com [1] => support@php.com [2] => john.doe@example.com )
)
```

Problem 7 - Uppercase First Letter of Words: Write a PHP function to capitalize the first letter of each word in a string.

```
function capitalizeWords($str) {
    return ucwords($str);
}

$result = capitalizeWords("hello, php world!");
echo $result; // Output: Hello, Php World!
```

Problem 8 - Truncate String: Create a PHP function that truncates a string to a specified length and adds "..." if it's longer.

```
function truncateString($str, $length) {
    if (strlen($str) > $length) {
        $str = substr($str, 0, $length) . "...";
    }
    return $str;
}

$result = truncateString("This is a long text that should be truncated.", 20);
echo $result; // Output: This is a long text...
```

Problem 9 - Check Anagram: Write a PHP function that checks if two strings are anagrams of each other (contain the same characters in a different order).

```
function areAnagrams($str1, $str2) {
    $str1 = str_replace(' ', '', $str1);
    $str2 = str_replace(' ', '', $str2);
    return count_chars($str1, 1) == count_chars($str2, 1);
}

$result = areAnagrams("listen", "silent");
echo $result ? "Anagrams" : "Not Anagrams"; // Output: Anagrams
```

Problem 10 - Find Substring Occurrences: Create a PHP function that finds all occurrences of a substring in a given string and returns their positions.

```
function findSubstringOccurrences($str, $substring) {
    $positions = [];
    $offset = 0;
    while (($pos = strpos($str, $substring, $offset)) !== false) {
        $positions[] = $pos;
        $offset = $pos + 1;
    }
    return $positions;
}

$result = findSubstringOccurrences("Hello, PHP! PHP is great.", "PHP");
print_r($result); // Output: Array([0] => 7 [1] => 14)
```

Problem 11 - Extract URL Parameters: Write a PHP function to extract and display all URL parameters from a given URL string.

```
function extractURLParameters($url) {
    parse_str(parse_url($url, PHP_URL_QUERY), $params);
    return $params;
}

$url = "https://example.com/page?name=John&age=30";
$parameters = extractURLParameters($url);
print_r($parameters); // Output: Array([name] => John [age] => 30)
```

Problem 12 - Shuffle String: Create a PHP function that shuffles the characters of a string randomly.

```
function shuffleString($str) {
    $arr = str_split($str);
    shuffle($arr);
    return implode('', $arr);
}

$result = shuffleString("Hello, PHP!");
echo $result; // Output: lP, HPeho!
```

Problem 13 - Find Longest Word: Write a PHP function that finds and returns the longest word in a string.

```
function findLongestWord($str) {
    $words = str_word_count($str, 1);
    $longest = "";
    foreach ($words as $word) {
        if (strlen($word) > strlen($longest)) {
            $longest = $word;
        }
    }
    return $longest;
}

$result = findLongestWord("This is a sample sentence with long words.");
echo $result; // Output: sentence
```

Problem 14 - Replace Multiple Characters: Create a PHP function that replaces multiple characters in a string based on a predefined mapping.

```
 function replaceMultipleCharacters($str, $charMap) {
    return strtr($str, $charMap);
}

$charMap = array("a" => "x", "e" => "y", "i" => "z");
$result = replaceMultipleCharacters("This is a test.", $charMap);
echo $result; // Output: Thzs zs y tyst.
```

Problem 15 - Remove Non-Alphanumeric Characters: Write a PHP function that removes all non-alphanumeric characters from a string.

```
 function removeNonAlphanumeric($str) {
    return preg_replace("/[^a-zA-Z0-9]/", "", $str);
}

$result = removeNonAlphanumeric("Hello, @PHP! 123");
echo $result; // Output: HelloPHP123
```

**Problem 16 - Check Sub

string:** Create a PHP function that checks if a given string contains a specific substring.

```
 function containsSubstring($str, $substring) {
    return strpos($str, $substring) !== false;
}

$result = containsSubstring("Hello, PHP!", "PHP");
echo $result ? "Contains substring" : "Does not contain substring"; // Output: Contains substring
```

Problem 17 - Count Vowels and Consonants: Write a PHP function that counts the number of vowels and consonants in a string.

```
 function countVowelsConsonants($str) {
    $str = strtolower($str);
    $vowels = 0;
    $consonants = 0;
    $vowelList = "aeiou";
    $str = str_replace(' ', '', $str);
    for ($i = 0; $i < strlen($str); $i++) {
        if (strpos($vowelList, $str[$i]) !== false) {
            $vowels++;
        } else {
            $consonants++;
        }
    }
    return array("vowels" => $vowels, "consonants" => $consonants);
}

$result = countVowelsConsonants("Hello, PHP!");
print_r($result); // Output: Array([vowels] => 2 [consonants] => 5)
```

Problem 18 - Count Character Occurrences: Create a PHP function that counts the occurrences of each character in a string and returns the result as an associative array.

```
function countCharacterOccurrences($str) {
    $charCount = [];
    $str = str_replace(' ', '', $str);
    for ($i = 0; $i < strlen($str); $i++) {
        $char = $str[$i];
        if (isset($charCount[$char])) {
            $charCount[$char]++;
        } else {
            $charCount[$char] = 1;
        }
    }
    return $charCount;
}

$result = countCharacterOccurrences("Hello, PHP!");
print_r($result); // Output: Array([H] => 1 [e] => 1 [l] => 2 [o] => 1 [P] => 1 [H] => 1)
```

Problem 19 - Reverse Words in a String: Write a PHP function that reverses the order of words in a given string.

```
function reverseWords($str) {
    $words = explode(' ', $str);
    $reversed = implode(' ', array_reverse($words));
    return $reversed;
}

$result = reverseWords("Hello PHP world");
echo $result; // Output: world PHP Hello
```

Problem 20 - Replace Words: Create a PHP function that replaces specific words in a string with predefined replacements.

```
function replaceWords($str, $wordMap) {
    return strtr($str, $wordMap);
}

$wordMap = array("PHP" => "JavaScript", "world" => "universe");
$result = replaceWords("Hello, PHP world!", $wordMap);
echo $result; // Output: Hello, JavaScript universe!
```

Certainly! Here are 10 more medium-level PHP problems and solutions related to strings and other topics, without repeating any previous questions:

Problem 21 - Concatenate Arrays of Strings: Write a PHP function that takes two arrays of strings and concatenates them into a single array of strings.

```
function concatenateStringArrays($arr1, $arr2) {
    return array_merge($arr1, $arr2);
}

$array1 = ["Hello", "PHP"];
$array2 = ["is", "awesome"];
$result = concatenateStringArrays($array1, $array2);
print_r($result); // Output: Array([0] => Hello [1] => PHP [2] => is [3] => awesome)
```

Problem 22 - Title Case: Create a PHP function that converts a string to title case (capitalizes the first letter of each word).

```
function titleCase($str) {
    return ucwords(strtolower($str));
}

$result = titleCase("hello, php world");
echo $result; // Output: Hello, Php World
```

Problem 23 - Extract Domain Name from URL: Write a PHP function that extracts and returns the domain name from a given URL.

```
function extractDomainFromURL($url) {
    $parsed = parse_url($url);
    return $parsed["host"];
}

$url = "https://www.example.com/page";
$result = extractDomainFromURL($url);
echo $result; // Output: www.example.com
```

Problem 24 - Convert Snake Case to Camel Case: Create a PHP function that converts a snake_case string to camelCase.

```
function snakeToCamel($str) {
    return lcfirst(str_replace('_', '', ucwords($str, '_')));
}

$result = snakeToCamel("hello_world_of_php");
echo $result; // Output: helloWorldOfPhp
```

Problem 25 - Generate Random Password: Write a PHP function that generates a random password of a specified length containing both letters and numbers.

```
function generateRandomPassword($length) {
    $chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    $password = "";
    for ($i = 0; $i < $length; $i++) {
        $password .= $chars[rand(0, strlen($chars) - 1)];
    }
    return $password;
}

$result = generateRandomPassword(8);
echo $result; // Output: Random password like "aB3zR7xY"
```

Problem 26 - Check Valid Email: Create a PHP function that checks if a given string is a valid email address.

```
function isValidEmail($email) {
    return filter_var($email, FILTER_VALIDATE_EMAIL) !== false;
}

$result = isValidEmail("john@example.com");
echo $result ? "Valid email" : "Invalid email"; // Output: Valid email
```

Problem 27 - Count Substring Occurrences: Write a PHP function that counts how many times a specific substring appears in a given string.

```
function countSubstringOccurrences($str, $substring) {
    return substr_count($str, $substring);
}

$result = countSubstringOccurrences("PHP is awesome. PHP is powerful.", "PHP");
echo $result; // Output: 2
```

Problem 28 - Replace Words with Links: Create a PHP function that replaces specific words in a string with clickable links.

```
function replaceWordsWithLinks($str, $linkMap) {
    foreach ($linkMap as $word => $link) {
        $str = str_replace($word, "<a href='$link'>$word</a>", $str);
    }
    return $str;
}

$text = "Visit our website for PHP tutorials and more.";
$links = array("PHP" => "https://www.php.net");
$result = replaceWordsWithLinks($text, $links);
echo $result; // Output: Visit our website for <a href='https://www.php.net'>PHP</a> tutorials and more.
```

Problem 29 - Encode and Decode URL: Write PHP functions to encode and decode a URL.

```
function encodeURL($url) {
    return urlencode($url);
}

function decodeURL($encodedUrl) {
    return urldecode($encodedUrl);
}

$url = "https://www.example.com/page?name=John&age=30";
$encoded = encodeURL($url);
$decoded = decodeURL($encoded);
echo "Encoded URL: $encoded<br>";
echo "Decoded URL: $decoded<br>";
```

Problem 30 - Generate Alphabetical Range: Create a PHP function that generates a range of alphabetical characters between two given letters.

```
function generateAlphabeticalRange($start, $end) {
    return implode("", range($start, $end));
}

$result = generateAlphabeticalRange('A', 'F');
echo $result; // Output: ABCDEF
```

Certainly! Here are 10 more medium-level PHP problems and solutions covering a range of topics without repeating any previous questions:

Problem 31 - Find Maximum Word Length: Write a PHP function that finds and returns the length of the longest word in a sentence.

```
function findLongestWordLength($sentence) {
    $words = str_word_count($sentence, 1);
    $maxLength = 0;
    foreach ($words as $word) {
        $wordLength = strlen($word);
        if ($wordLength > $maxLength) {
            $maxLength = $wordLength;
        }
    }
    return $maxLength;
}

$result = findLongestWordLength("PHP is a powerful scripting language.");
echo $result; // Output: 9
```

Problem 32 - Validate Date Format: Create a PHP function that checks if a given string is a valid date in a specified format (e.g., "YYYY-MM-DD").

```
function isValidDateFormat($date, $format = 'Y-m-d') {
    $dateTime = DateTime::createFromFormat($format, $date);
    return $dateTime && $dateTime->format($format) == $date;
}

$result = isValidDateFormat("2023-09-20");
echo $result ? "Valid date" : "Invalid date"; // Output: Valid date
```

Problem 33 - Remove Duplicates from String: Write a PHP function that removes duplicate characters from a string while preserving the order.

```
function removeDuplicates($str) {
    return implode(array_unique(str_split($str)));
}

$result = removeDuplicates("programming");
echo $result; // Output: progamin
```

Problem 34 - Count Line Breaks: Create a PHP function that counts the number of line breaks (newlines) in a given string.

```
function countLineBreaks($str) {
    return substr_count($str, "\n");
}

$text = "This is a\nmulti-line\nstring.";
$result = countLineBreaks($text);
echo $result; // Output: 2
```

Problem 35 - JSON to Array: Write a PHP function that converts a JSON string into an associative array.

```
function jsonToArray($jsonString) {
    return json_decode($jsonString, true);
}

$json = '{"name": "John", "age": 30}';
$array = jsonToArray($json);
print_r($array);
```

Problem 36 - Array to JSON: Create a PHP function that converts an associative array into a JSON string.

```
function arrayToJson($array) {
    return json_encode($array);
}

$array = array("name" => "Jane", "age" => 25);
$json = arrayToJson($array);
echo $json; // Output: {"name":"Jane","age":25}
```

Problem 37 - Remove HTML Tags: Write a PHP function that removes HTML tags from a given string.

```
function removeHtmlTags($str) {
    return strip_tags($str);
}

$html = "<p>This is <b>HTML</b> content.</p>";
$result = removeHtmlTags($html);
echo $result; // Output: This is HTML content.
```

Problem 38 - Sum of Digits in a String: Create a PHP function that calculates the sum of all digits in a string.

```
function sumDigitsInString($str) {
    $digits = preg_replace("/[^0-9]/", "", $str);
    return array_sum(str_split($digits));
}

$result = sumDigitsInString("Hello123World456");
echo $result; // Output: 21
```

Problem 39 - Generate Random String: Write a PHP function that generates a random string of a specified length using characters from a given set.

```
function generateRandomString($length, $charset = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789') {
    $randomString = '';
    for ($i = 0; $i < $length; $i++) {
        $randomString .= $charset[rand(0, strlen($charset) - 1)];
    }
    return $randomString;
}

$result = generateRandomString(8);
echo $result; // Output: Random string like "aB3zR7xY"
```

Problem 40 - Check Balanced Brackets: Create a PHP function that checks if a given string with parentheses is balanced (all open brackets have corresponding closing brackets).

~~~
function areBracketsBalanced($str) {
    $stack = [];
    $brackets = ["(" => ")", "[" => "]", "{" => "}"];

    foreach (str_split($str) as $char) {
        if (array_key_exists($char, $brackets)) {
            array_push($stack, $char);
        } elseif (in_array($char, $brackets)) {
            if (empty($stack) || $brackets[array_pop($stack)] !== $char) {
                return false;
            }
        }
    }

    return empty($stack);
}

$result = areBracketsBalanced("{[()]()}");
echo $result ? "Balanced" : "Not Balanced"; // Output: Balanced

Certainly! Here are 10 more medium-level PHP problems and solutions, each with a brief explanation:

**Problem 41 - URL Slug Generator:**
Create a PHP function that generates a URL-friendly slug from a given string. A slug is a lowercase string with hyphens replacing sp

```php
function generateSlug($str) {
    $str = strtolower(trim($str));
    $str = preg_replace("/[^a-z0-9-]+/", "-", $str);
    return preg_replace("/-+/", "-", $str);
}

$result = generateSlug("Hello, PHP World!");
echo $result; // Output: hello-php-world
~~~

Problem 42 - Find Missing Character: Write a PHP function that finds the missing character from a string containing all the letters of the alphabet except one.

```
function findMissingCharacter($str) {
    $alphabet = "abcdefghijklmnopqrstuvwxyz";
    $missing = "";
    for ($i = 0; $i < strlen($alphabet); $i++) {
        if (strpos($str, $alphabet[$i]) === false) {
            $missing = $alphabet[$i];
            break;
        }
    }
    return $missing;
}

$result = findMissingCharacter("abcdfghijklmnopqrstuvwxyz");
echo $result; // Output: e
```

Problem 43 - Swap Case in a String: Create a PHP function that swaps the case (uppercase to lowercase and vice versa) of letters in a string.

```
function swapCase($str) {
    return strtr($str, "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz", "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
}

$result = swapCase("Hello, PHP World!");
echo $result; // Output: hELLO, php wORLD!
```

Problem 44 - Check Pangram: Write a PHP function that checks if a given string is a pangram (contains all the letters of the alphabet at least once).

```
function isPangram($str) {
    $str = strtolower($str);
    return count(array_unique(str_split(preg_replace("/[^a-z]/", "", $str)))) == 26;
}

$result = isPangram("The quick brown fox jumps over the lazy dog.");
echo $result ? "Pangram" : "Not a Pangram"; // Output: Pangram
```

Problem 45 - Calculate Levenshtein Distance: Create a PHP function that calculates and returns the Levenshtein distance between two strings (the minimum number of single-character edits required to change one word into the other).

```
function levenshteinDistance($str1, $str2) {
    return levenshtein($str1, $str2);
}

$result = levenshteinDistance("kitten", "sitting");
echo $result; // Output: 3
```

Problem 46 - Check Valid IPv4 Address: Write a PHP function that checks if a given string is a valid IPv4 address.

```
function isValidIPv4($ip) {
    return filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV4) !== false;
}

$result = isValidIPv4("192.168.0.1");
echo $result ? "Valid IPv4" : "Invalid IPv4"; // Output: Valid IPv4
```

Problem 47 - Extract Last Word: Create a PHP function that extracts and returns the last word from a string.

```
function extractLastWord($str) {
    $words = str_word_count($str, 1);
    return end($words);
}

$result = extractLastWord("This is the last word.");
echo $result; // Output: word
```

Problem 48 - Check Balanced Parentheses: Write a PHP function that checks if a given string with parentheses is balanced (all open brackets have corresponding closing brackets).

```
function areParenthesesBalanced($str) {
    $stack = [];
    foreach (str_split($str) as $char) {
        if ($char == '(') {
            array_push($stack, $char);
        } elseif ($char == ')') {
            if (empty($stack) || array_pop($stack) != '(') {
                return false;
            }
        }
    }
    return empty($stack);
}

$result = areParenthesesBalanced("(a + b) * (c - d)");
echo $result ? "Balanced" : "Not Balanced"; // Output: Balanced
```

Problem 49 - Find Common Characters: Create a PHP function that finds and returns the common characters shared by two strings.

```
function findCommonCharacters($str1, $str2) {
    return implode("", array_intersect(str_split($str1), str_split($str2)));
}

$result = findCommonCharacters("programming", "language");
echo $result; // Output: amm
```

Problem 50 - Truncate Text: Write a PHP function that truncates a text to a specified length while avoiding cutting words in the middle.

```
function truncateText($text, $maxLength) {
    if (strlen($text) <= $maxLength) {
        return $text;
    }
    $truncated = substr($text, 0, $maxLength);
    return substr($truncated, 0, strrpos($truncated, ' ')) . '...';
}

$result = truncateText("This is a long text that should be truncated.", 20);
echo $result; // Output: This is a long text...
```

Certainly! Here are 10 more medium-level PHP problems and solutions, each with a brief explanation:

Problem 51 - Find First Non-Repeating Character: Write a PHP function that finds and returns the first non-repeating character in a string.

```
function firstNonRepeatingCharacter($str) {
    $charCount = array_count_values(str_split($str));
    foreach (str_split($str) as $char) {
        if ($charCount[$char] === 1) {
            return $char;
        }
    }
    return false;
}

$result = firstNonRepeatingCharacter("programming");
echo $result; // Output: p
```

Problem 52 - Calculate Factorial: Create a PHP function that calculates and returns the factorial of a given integer.

```
function factorial($n) {
    if ($n < 0) {
        return "Invalid input";
    }
    if ($n === 0) {
        return 1;
    }
    return $n * factorial($n - 1);
}

$result = factorial(5);
echo $result; // Output: 120
```

Problem 53 - Generate Random Password with Constraints: Write a PHP function that generates a random password of a specified length with constraints on the characters used (uppercase, lowercase, digits, and symbols).

```
function generateRandomPasswordWithConstraints($length) {
    $uppercase = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    $lowercase = "abcdefghijklmnopqrstuvwxyz";
    $digits = "0123456789";
    $symbols = "!@#$%^&*()";
    $charset = $uppercase . $lowercase . $digits . $symbols;
    $password = "";
    for ($i = 0; $i < $length; $i++) {
        $password .= $charset[rand(0, strlen($charset) - 1)];
    }
    return $password;
}

$result = generateRandomPasswordWithConstraints(10);
echo $result; // Output: Random password with constraints
```

Problem 54 - Reverse Words in a Sentence: Create a PHP function that reverses the order of words in a sentence.

```
function reverseWordsInSentence($sentence) {
    $words = explode(' ', $sentence);
    return implode(' ', array_reverse($words));
}

$result = reverseWordsInSentence("This is a sample sentence.");
echo $result; // Output: sentence. sample a is This
```

Problem 55 - Count Subsequences of 'abc': Write a PHP function that counts the number of subsequences in a given string that match the pattern 'abc'.

```
function countSubsequencesABC($str) {
    $count = 0;
    $aCount = 0;
    $abCount = 0;
    foreach (str_split($str) as $char) {
        if ($char === 'a') {
            $aCount++;
        } elseif ($char === 'b') {
            $abCount += $aCount;
        } elseif ($char === 'c') {
            $count += $abCount;
        }
    }
    return $count;
}

$result = countSubsequencesABC("ababcabcabc");
echo $result; // Output: 16
```

Problem 56 - Validate Credit Card Number: Create a PHP function that validates a credit card number using the Luhn algorithm.

```
function validateCreditCard($cardNumber) {
    $cardNumber = str_replace(' ', '', $cardNumber);
    $digits = str_split(strrev($cardNumber));
    $sum = 0;
    for ($i = 0; $i < count($digits); $i++) {
        $digit = (int)$digits[$i];
        if ($i % 2 == 1) {
            $digit *= 2;
            if ($digit > 9) {
                $digit -= 9;
            }
        }
        $sum += $digit;
    }
    return $sum % 10 === 0;
}

$result = validateCreditCard("4532015112830366");
echo $result ? "Valid" : "Invalid"; // Output: Valid
```

Problem 57 - Find Common Prefix in Strings: Write a PHP function that finds the common prefix among an array of strings.

```
function findCommonPrefix($strs) {
    if (empty($strs)) {
        return "";
    }
    $prefix = $strs[0];
    for ($i = 1; $i < count($strs); $i++) {
        while (strpos($strs[$i], $prefix) !== 0) {
            $prefix = substr($prefix, 0, -1);
            if (empty($prefix)) {
                return "";
            }
        }
    }
    return $prefix;
}

$strings = ["flower", "flour", "flow"];
$result = findCommonPrefix($strings);
echo $result; // Output: "flo"
```

Problem 58 - Remove Nth Node from End of List: Create a PHP function that removes the Nth node from the end of a linked list.

```
class ListNode {
    public $val;
    public $next;
    function __construct($val = 0, $next = null) {
        $this->val = $val;
        $this->next = $next;
    }
}

function removeNthFromEnd($head, $n) {
    $dummy = new ListNode(0);
    $dummy->next = $head;
    $fast = $slow = $dummy;
    for ($i = 0; $i <= $n; $i++) {
        $fast = $fast->next;
    }
    while ($fast !== null) {
        $fast = $fast->next;
        $slow = $slow->next;
    }
    $slow->next = $slow->next->next;
    return $dummy->next;
}

// Usage:
// $head = new ListNode(1, new ListNode(2, new ListNode(3, new ListNode(4, new ListNode(5)))));
// $result = removeNthFromEnd($head, 2);
```

Problem 59 - Rotate Image: Write a PHP function that rotates an N x N matrix representing an image.

```
function rotateImage(&$matrix) {
    $n = count($matrix);
    for ($i = 0; $i < $n / 2; $i++) {
        for ($j = $i; $j < $n - $i - 1; $j++) {
            $temp = $matrix[$i][$j];
            $matrix[$i][$j] = $matrix[$n - $j - 1][$i];
            $matrix[$n - $j - 1][$i] = $matrix[$n - $i - 1][$n - $j - 1];
            $matrix[$n - $i - 1][$n - $j - 1] = $matrix[$j][$n - $i - 1];
            $matrix[$j][$n - $i - 1] = $temp;
        }
    }
```

Integers, Floats, and Number Strings

Certainly! Here are 30 medium-level PHP problems and solutions related to "Integers, Floats, and Number Strings" for PHP developer interviews and remote job applications, without repeating any previous questions:

Problem 1 - Check Even or Odd: Write a PHP function that checks if a given integer is even or odd.

```
function isEven($num) {
    return $num % 2 === 0;
}

$result = isEven(4);
echo $result ? "Even" : "Odd"; // Output: Even
```

Problem 2 - Reverse Integer: Create a PHP function that reverses the digits of an integer.

```
function reverseInteger($num) {
    $reversed = (int)strrev((string)abs($num));
    return ($num < 0) ? -$reversed : $reversed;
}

$result = reverseInteger(12345);
echo $result; // Output: 54321
```

Problem 3 - Palindrome Number: Write a PHP function that checks if an integer is a palindrome (reads the same forwards and backwards).

```
function isPalindrome($num) {
    return $num == reverseInteger($num);
}

$result = isPalindrome(121);
echo $result ? "Palindrome" : "Not Palindrome"; // Output: Palindrome
```

Problem 4 - Fibonacci Sequence: Create a PHP function that generates the Fibonacci sequence up to a specified number of terms.

```
function generateFibonacci($n) {
    $fib = [0, 1];
    while (count($fib) < $n) {
        $fib[] = $fib[count($fib) - 1] + $fib[count($fib) - 2];
    }
    return $fib;
}

$result = generateFibonacci(5);
print_r($result); // Output: [0, 1, 1, 2, 3]
```

Problem 5 - Prime Number Check: Write a PHP function that checks if a given integer is a prime number.

```
function isPrime($num) {
    if ($num <= 1) {
        return false;
    }
    for ($i = 2; $i * $i <= $num; $i++) {
        if ($num % $i === 0) {
            return false;
        }
    }
    return true;
}

$result = isPrime(7);
echo $result ? "Prime" : "Not Prime"; // Output: Prime
```

Problem 6 - Factorial of a Number: Create a PHP function that calculates and returns the factorial of a given integer.

```
function factorial($n) {
    if ($n < 0) {
        return "Invalid input";
    }
    if ($n === 0) {
        return 1;
    }
    return $n * factorial($n - 1);
}

$result = factorial(5);
echo $result; // Output: 120
```

Problem 7 - Decimal to Binary Conversion: Write a PHP function that converts a decimal number to its binary representation.

```
function decimalToBinary($num) {
    return decbin($num);
}

$result = decimalToBinary(10);
echo $result; // Output: 1010
```

Problem 8 - Binary to Decimal Conversion: Create a PHP function that converts a binary number to its decimal representation.

```
function binaryToDecimal($binary) {
    return bindec($binary);
}

$result = binaryToDecimal("1010");
echo $result; // Output: 10
```

Problem 9 - Largest Integer in Array: Write a PHP function that finds and returns the largest integer in an array of numbers.

```
function findLargestInArray($arr) {
    return max($arr);
}

$array = [4, 7, 2, 9, 5];
$result = findLargestInArray($array);
echo $result; // Output: 9
```

Problem 10 - Smallest Integer in Array: Create a PHP function that finds and returns the smallest integer in an array of numbers.

```
function findSmallestInArray($arr) {
    return min($arr);
}

$array = [4, 7, 2, 9, 5];
$result = findSmallestInArray($array);
echo $result; // Output: 2
```

Problem 11 - Check Palindrome String: Write a PHP function that checks if a given string is a palindrome.

```
function isPalindromeString($str) {
    $str = strtolower(preg_replace("/[^A-Za-z0-9]/", '', $str));
    return $str == strrev($str);
}

$result = isPalindromeString("A man, a plan, a canal, Panama");
echo $result ? "Palindrome" : "Not Palindrome"; // Output: Palindrome
```

Problem 12 - Square Root Calculation: Create a PHP function that calculates the square root of a given number.

```
function squareRoot($num) {
    return sqrt($num);
}

$result = squareRoot(16);
echo $result; // Output: 4
```

Problem 13 - Find Largest Common Divisor (GCD): Write a PHP function that finds and returns the greatest common divisor (GCD) of two integers.

```
function findGCD($a, $b) {
    while ($b != 0) {
        $temp = $b;
        $b = $a % $b;
        $a = $temp;
    }
    return $a;
}

$result = findGCD(48, 18);
echo $result; // Output: 6
```

Problem 14 - Calculate Power: Create a PHP function that calculates the result of raising a number to a given power.

```
function calculatePower($base, $exponent) {
    return pow($base, $exponent);
}

$result = calculatePower(2, 3);
echo $result; // Output: 8
```

Problem 15 - Calculate Factorial (Loop Version): Write a PHP function that calculates and returns the factorial of a given integer using a loop.

```
function factorialLoop($n) {
    $result = 1;
    for ($i = 2; $i <= $n; $i++) {
        $result *= $i;
    }
    return $result;
}

$result = factorialLoop(5);
echo $result; // Output: 120
```

Problem 16 - Generate Random Number in a Range: Create a PHP function that generates a random integer within a specified range.

```
function randomInRange($min, $max) {
    return rand($min, $max);
}

$result = randomInRange(1, 10);
echo $result; // Output: Random number in the range
```

Problem 17 - Truncate Floating-Point Number: Write a PHP function that truncates a floating-point number to a specified number of decimal places.

```
function truncateFloat($number, $decimals) {
    return number_format($number, $decimals, '.', '');
}

$result = truncateFloat(3.14159265, 2);



echo $result; // Output: 3.14
```

Problem 18 - Round Floating-Point Number: Create a PHP function that rounds a floating-point number to a specified number of decimal places.

```
function roundFloat($number, $decimals) {
    return round($number, $decimals);
}

$result = roundFloat(3.14159265, 2);
echo $result; // Output: 3.14
```

Problem 19 - Convert Number to Roman Numerals: Write a PHP function that converts an integer to its Roman numeral representation.

```
function intToRoman($num) {
    $roman = "";
    $values = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1];
    $symbols = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"];
    for ($i = 0; $i < count($values); $i++) {
        while ($num >= $values[$i]) {
            $roman .= $symbols[$i];
            $num -= $values[$i];
        }
    }
    return $roman;
}

$result = intToRoman(1984);
echo $result; // Output: MCMLXXXIV
```

Problem 20 - Convert Roman Numerals to Number: Create a PHP function that converts a Roman numeral to its corresponding integer value.

```
function romanToInt($roman) {
    $values = ["I" => 1, "V" => 5, "X" => 10, "L" => 50, "C" => 100, "D" => 500, "M" => 1000];
    $total = 0;
    $prevValue = 0;
    for ($i = strlen($roman) - 1; $i >= 0; $i--) {
        $currentValue = $values[$roman[$i]];
        if ($currentValue < $prevValue) {
            $total -= $currentValue;
        } else {
            $total += $currentValue;
        }
        $prevValue = $currentValue;
    }
    return $total;
}

$result = romanToInt("MCMLXXXIV");
echo $result; // Output: 1984
```

Problem 21 - Find Duplicates in an Array: Write a PHP function that finds and returns duplicate numbers in an array.

```
function findDuplicates($arr) {
    return array_keys(array_filter(array_count_values($arr), function($count) {
        return $count > 1;
    }));
}

$array = [1, 2, 3, 2, 4, 5, 3];
$result = findDuplicates($array);
print_r($result); // Output: [2, 3]
```

Problem 22 - Calculate Average of Numbers in Array: Create a PHP function that calculates and returns the average of numbers in an array.

```
function calculateAverage($arr) {
    return array_sum($arr) / count($arr);
}

$array = [1, 2, 3, 4, 5];
$result = calculateAverage($array);
echo $result; // Output: 3
```

Problem 23 - Convert Integer to Binary String: Write a PHP function that converts an integer to its binary representation as a string.

```
function intToBinaryString($num) {
    return decbin($num);
}

$result = intToBinaryString(10);
echo $result; // Output: "1010"
```

Problem 24 - Calculate Mean and Median: Create a PHP function that calculates the mean and median of a given array of numbers.

```
function calculateMeanAndMedian($arr) {
    $mean = array_sum($arr) / count($arr);
    sort($arr);
    $middle = floor(count($arr) / 2);
    $median = ($middle % 2 == 0) ? ($arr[$middle - 1] + $arr[$middle]) / 2 : $arr[$middle];
    return ["mean" => $mean, "median" => $median];
}

$array = [1, 2, 3, 4, 5];
$result = calculateMeanAndMedian($array);
print_r($result); // Output: ["mean" => 3, "median" => 3]
```

Problem 25 - Convert Integer to Words: Write a PHP function that converts an integer to its word representation.

```
function intToWords($num) {
    $ones = ["", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"];
    $teens = ["", "eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen", "eighteen", "nineteen"];
    $tens

 = ["", "ten", "twenty", "thirty", "forty", "fifty", "sixty", "seventy", "eighty", "ninety"];

    if ($num == 0) {
        return "zero";
    }

    $words = "";
    if ($num >= 1000) {
        $words .= $ones[($num / 1000) % 10] . " thousand ";
        $num %= 1000;
    }
    if ($num >= 100) {
        $words .= $ones[($num / 100) % 10] . " hundred ";
        $num %= 100;
    }
    if ($num > 0) {
        if ($num >= 20) {
            $words .= $tens[($num / 10) % 10] . " " . $ones[$num % 10];
        } elseif ($num >= 11) {
            $words .= $teens[$num - 10];
        } else {
            $words .= $ones[$num];
        }
    }
    return trim($words);
}

$result = intToWords(12345);
echo $result; // Output: "twelve thousand three hundred forty-five"
```

Problem 26 - Validate ISBN-10: Create a PHP function that validates if a given string is a valid ISBN-10 (International Standard Book Number) format.

```
 function isValidISBN10($isbn) {
    $isbn = str_replace(["-", " "], "", $isbn);
    if (strlen($isbn) !== 10) {
        return false;
    }
    $checksum = 0;
    for ($i = 0; $i < 10; $i++) {
        if ($isbn[$i] === 'X') {
            $checksum += 10;
        } elseif (is_numeric($isbn[$i])) {
            $checksum += $isbn[$i] * (10 - $i);
        } else {
            return false;
        }
    }
    return $checksum % 11 === 0;
}

$result = isValidISBN10("0-306-40615-2");
echo $result ? "Valid ISBN-10" : "Invalid ISBN-10"; // Output: Valid ISBN-10
```

Problem 27 - Calculate Area of Circle: Write a PHP function that calculates the area of a circle given its radius.

```
 function calculateCircleArea($radius) {
    return pi() * pow($radius, 2);
}

$result = calculateCircleArea(5);
echo $result; // Output: 78.539816339745
```

Problem 28 - Convert Float to Fraction: Create a PHP function that converts a floating-point number to a simplified fraction.

```
 function floatToFraction($number) {
    $precision = 1 / 1000000;
    $numerator = 1;
    $denominator = 1;
    while (abs($number - ($numerator / $denominator)) > $precision) {
        if ($number > ($numerator / $denominator)) {
            $numerator++;
        } else {
            $denominator++;
        }
    }
    return [$numerator, $denominator];
}

$result = floatToFraction(0.75);
echo $result[0] . "/" . $result[1]; // Output: 3/4
```

Problem 29 - Calculate Square of a Float: Write a PHP function that calculates the square of a floating-point number.

```
 function squareOfFloat($number) {
    return $number * $number;
}

$result = squareOfFloat(2.5);
echo $result; // Output: 6.25
```

Problem 30 - Largest Common Divisor (GCD) for Array: Create a PHP function that finds and returns the greatest common divisor (GCD) of an array of integers.

```
function findGCDArray($arr) {
    $gcd = $arr[0];
    $count = count($arr);
    for ($i = 1; $i < $count; $i++) {
        $gcd = findGCD($gcd, $arr[$i]);
    }
    return $gcd;
}

$array = [18, 36, 54];
$result = findGCDArray($array);
echo $result; // Output: 18
```

Certainly! Here are ten additional medium-level PHP problems and solutions related to "Integers, Floats, and Number Strings":

Problem 31 - Find the Sum of Digits: Create a PHP function that calculates and returns the sum of the digits of an integer.

```
function sumOfDigits($num) {
    $sum = 0;
    while ($num > 0) {
        $sum += $num % 10;
        $num = (int)($num / 10);
    }
    return $sum;
}

$result = sumOfDigits(12345);
echo $result; // Output: 15
```

Problem 32 - Check Armstrong Number: Write a PHP function that checks if a given number is an Armstrong number (a number that is equal to the sum of its own digits each raised to the power of the number of digits).

```
function isArmstrong($num) {
    $originalNum = $num;
    $numOfDigits = strlen((string)$num);
    $sum = 0;
    while ($num > 0) {
        $digit = $num % 10;
        $sum += pow($digit, $numOfDigits);
        $num = (int)($num / 10);
    }
    return $sum === $originalNum;
}

$result = isArmstrong(153);
echo $result ? "Armstrong" : "Not Armstrong"; // Output: Armstrong
```

Problem 33 - Calculate HCF and LCM: Create a PHP function that calculates the highest common factor (HCF) and lowest common multiple (LCM) of two integers.

```
function calculateHCF($a, $b) {
    while ($b != 0) {
        $temp = $b;
        $b = $a % $b;
        $a = $temp;
    }
    return $a;
}

function calculateLCM($a, $b) {
    return ($a * $b) / calculateHCF($a, $b);
}

$hcf = calculateHCF(12, 18);
$lcm = calculateLCM(12, 18);
echo "HCF: " . $hcf . ", LCM: " . $lcm; // Output: HCF: 6, LCM: 36
```

Problem 34 - Check Happy Number: Write a PHP function that checks if a given number is a happy number (a number that eventually reaches 1 when replaced by the sum of the square of each digit).

```
function isHappy($num) {
    $seen = [];
    while ($num != 1 && !isset($seen[$num])) {
        $seen[$num] = true;
        $sum = 0;
        while ($num > 0) {
            $digit = $num % 10;
            $sum += $digit * $digit;
            $num = (int)($num / 10);
        }
        $num = $sum;
    }
    return $num === 1;
}

$result = isHappy(19);
echo $result ? "Happy" : "Not Happy"; // Output: Happy
```

Problem 35 - Validate Credit Card Number (Luhn Algorithm): Create a PHP function that validates a credit card number using the Luhn algorithm.

```
function isValidCreditCard($cardNumber) {
    $cardNumber = str_replace(' ', '', $cardNumber);
    $digits = str_split(strrev($cardNumber));
    $sum = 0;
    for ($i = 0; $i < count($digits); $i++) {
        $digit = (int)$digits[$i];
        if ($i % 2 == 1) {
            $digit *= 2;
            if ($digit > 9) {
                $digit -= 9;
            }
        }
        $sum += $digit;
    }
    return $sum % 10 === 0;
}

$result = isValidCreditCard("4532015112830366");
echo $result ? "Valid" : "Invalid"; // Output: Valid
```

Problem 36 - Calculate Area of Triangle: Write a PHP function that calculates the area of a triangle given its base and height.

```
function calculateTriangleArea($base, $height) {
    return 0.5 * $base * $height;
}

$result = calculateTriangleArea(5, 8);
echo $result; // Output: 20
```

Problem 37 - Calculate Exponentiation Using Recursion: Create a PHP function that calculates the result of raising a number to a given power using recursion.

```
function power($base, $exponent) {
    if ($exponent == 0) {
        return 1;
    }
    return $base * power($base, $exponent - 1);
}

$result = power(2, 3);
echo $result; // Output: 8
```

Problem 38 - Calculate Median of Two Sorted Arrays: Write a PHP function that finds the median of two sorted arrays.

```
function findMedianSortedArrays($nums1, $nums2) {
    $merged = array_merge($nums1, $nums2);
    sort($merged);
    $count = count($merged);
    if ($count % 2 == 0) {
        return ($merged[$count / 2 - 1] + $merged[$count / 2]) / 2;
    } else {
        return $merged[($count - 1) / 2];
    }
}

$result = findMedianSortedArrays([1, 3], [2]);
echo $result

; // Output: 2.0
```

Problem 39 - Generate Random Prime Number: Create a PHP function that generates a random prime number within a specified range.

```
function generateRandomPrime($min, $max) {
    $primes = [];
    for ($i = max(2, $min); $i <= $max; $i++) {
        if (isPrime($i)) {
            $primes[] = $i;
        }
    }
    return ($primes) ? $primes[array_rand($primes)] : null;
}

function isPrime($num) {
    if ($num <= 1) {
        return false;
    }
    for ($i = 2; $i * $i <= $num; $i++) {
        if ($num % $i === 0) {
            return false;
        }
    }
    return true;
}

$result = generateRandomPrime(10, 50);
echo $result; // Output: Random prime number in the range
```

Math

Problem 1: Find the Factorial

- **Problem:** Calculate the factorial of a given integer.
- **Solution:**

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

Problem 2: Fibonacci Series

- **Problem:** Generate the Fibonacci series up to a given term 'n'.
- **Solution:**

```
function fibonacci($n) {
    $fib = [0, 1];
    for ($i = 2; $i <= $n; $i++) {
        $fib[$i] = $fib[$i - 1] + $fib[$i - 2];
    }
    return $fib;
}
```

Problem 3: Prime Numbers

- **Problem:** Check if a given number is prime.
- **Solution:**

```
function isPrime($n) {
    if ($n <= 1) {
        return false;
    }
    for ($i = 2; $i * $i <= $n; $i++) {
        if ($n % $i == 0) {
            return false;
        }
    }
    return true;
}
```

Problem 4: GCD (Greatest Common Divisor)

- **Problem:** Find the GCD of two numbers.
- **Solution:**

```
function gcd($a, $b) {
    while ($b != 0) {
        $temp = $b;
        $b = $a % $b;
        $a = $temp;
    }
    return $a;
}
```

Problem 5: LCM (Least Common Multiple)

- **Problem:** Find the LCM of two numbers.
- **Solution:**

```
function lcm($a, $b) {
    return ($a * $b) / gcd($a, $b);
}
```

Problem 6: Power Function

- **Problem:** Implement a power function (e.g., x^y).
- **Solution:**

```
function power($x, $y) {
    $result = 1;
    for ($i = 0; $i < $y; $i++) {
        $result *= $x;
    }
    return $result;
}
```

Problem 7: Palindrome Number

- **Problem:** Check if a given number is a palindrome.
- **Solution:**

```
function isPalindrome($n) {
    return $n == strrev($n);
}
```

Problem 8: Count Digits

- **Problem:** Count the number of digits in a given integer.
- **Solution:**

```
function countDigits($n) {
    return strlen((string)$n);
}
```

Problem 9: Sum of Digits

- **Problem:** Calculate the sum of the digits of a number.
- **Solution:**

```
function sumOfDigits($n) {
    $sum = 0;
    while ($n != 0) {
        $sum += $n % 10;
        $n = (int)($n / 10);
    }
    return $sum;
}
```

Problem 10: Prime Factorization

- **Problem:** Find the prime factors of a number.
- **Solution:**

```
function primeFactors($n) {
    $factors = [];
    while ($n % 2 == 0) {
        $factors[] = 2;
        $n = $n / 2;
    }
    for ($i = 3; $i * $i <= $n; $i = $i + 2) {
        while ($n % $i == 0) {
            $factors[] = $i;
            $n = $n / $i;
        }
    }
    if ($n > 2) {
        $factors[] = $n;
    }
    return $factors;
}
```

Problem 11: Factorial Trailing Zeros

- **Problem:** Calculate the number of trailing zeros in the factorial of a number.
- **Solution:**

```
function trailingZeros($n) {
    $count = 0;
    while ($n >= 5) {
        $n = (int)($n / 5);
        $count += $n;
    }
    return $count;
}
```

Problem 12: Binary to Decimal Conversion

- **Problem:** Convert a binary number to its decimal equivalent.
- **Solution:**

```
function binaryToDecimal($binary) {
    return bindec($binary);
}
```

Problem 13: Decimal to Binary Conversion

- **Problem:** Convert a decimal number to its binary equivalent.
- **Solution:**

```
function decimalToBinary($decimal) {
    return decbin($decimal);
}
```

Problem 14: Pascal's Triangle

- **Problem:** Generate Pascal's Triangle up to 'n' rows.
- **Solution:**

```
function generatePascalsTriangle($n) {
    $triangle = [];
    for ($i = 0; $i < $n; $i++) {
        $row = [];
        for ($j = 0; $j <= $i; $j++) {
            if ($j == 0 || $j == $i) {
                $row[] = 1;
            } else {
                $row[] = $triangle[$i - 1][$j - 1] + $triangle[$i - 1][$j];
            }
        }
        $triangle[] = $row;
    }
    return $triangle;
}
```

Problem 15: Square Root

- **Problem:** Find the square root of a number using the Newton-Raphson method.
- **Solution:**

```
function sqrt($x) {
    $guess = $x;
    while (abs($guess * $guess - $x) >= 0.0001) {
        $guess = ($guess + $x / $guess) / 2;
    }
    return $guess;
}
```

Problem 16: Combinations

- **Problem:** Calculate the number of combinations (n choose k).
- **Solution:**

```
function combinations($n, $k) {
    if ($k == 0 || $k == $n) {
        return 1;
    }
    return combinations($n - 1, $k - 1) + combinations($n - 1, $k);
}
```

Problem 17: Permutations

- **Problem:** Generate all permutations of a string.
- **Solution:**

```
function permutations($str) {
    if (strlen($str) == 1) {
        return [$str];
    }
    $perms = [];
    $firstChar = $str[0];
    $rest = substr($str, 1);
    $restPerms = permutations($rest);
    foreach ($restPerms as $perm) {
        for ($i = 0; $i <= strlen($perm); $i++) {
            $perms[] = substr($perm, 0, $i) . $firstChar . substr($perm, $i);
        }
    }
    return $perms;
}
```

Problem 18: Sum of Primes

- **Problem:** Find the sum of all prime numbers up to 'n'.
- **Solution:**

```
function sumOfPrimes($n) {
    $sum = 0;
    for ($i = 2; $i <= $n; $i++) {
        if (isPrime($i)) {
            $sum += $i;
        }
    }
    return $sum;
}
```

Problem 19: Perfect Numbers

- **Problem:** Check if a number is a perfect number.
- **Solution:**

```
function isPerfectNumber($n) {
    $sum = 1;
    for ($i = 2; $i * $i <= $n; $i++) {
        if ($n % $i == 0) {
            $sum += $i + $n / $i;
        }
    }
    return $sum == $n;
}
```

Problem 20: Sum of Digits of a Power

- **Problem:** Calculate the sum of the digits of a number raised to a power.
- **Solution:**

```
function sumOfDigitsOfPower($x, $y) {
    $result = power($x, $y); // You can use the 'power' function from Problem 6.
    return sumOfDigits($result); // You can use the 'sumOfDigits' function from Problem 9.
}
```

Certainly! Here are 20 more math-related problems and solutions in PHP for PHP developers preparing for interviews, with explanations and examples relevant to competitive programming platforms:

Problem 21: Armstrong Number

- **Problem:** Check if a number is an Armstrong number.
- **Solution:**

```
function isArmstrong($n) {
    $sum = 0;
    $temp = $n;
    while ($temp != 0) {
        $digit = $temp % 10;
        $sum += $digit ** strlen((string)$n);
        $temp = (int)($temp / 10);
    }
    return $sum == $n;
}
```

Problem 22: Decimal to Hexadecimal Conversion

- **Problem:** Convert a decimal number to its hexadecimal equivalent.
- **Solution:**

```
function decimalToHexadecimal($decimal) {
    return dechex($decimal);
}
```

Problem 23: Hexadecimal to Decimal Conversion

- **Problem:** Convert a hexadecimal number to its decimal equivalent.
- **Solution:**

```
function hexadecimalToDecimal($hexadecimal) {
    return hexdec($hexadecimal);
}
```

Problem 24: Area of a Circle

- **Problem:** Calculate the area of a circle given its radius.
- **Solution:**

```
function circleArea($radius) {
    return M_PI * $radius * $radius;
}
```

Problem 25: Area of a Triangle

- **Problem:** Calculate the area of a triangle given its base and height.
- **Solution:**

```
function triangleArea($base, $height) {
    return 0.5 * $base * $height;
}
```

Problem 26: Area of a Rectangle

- **Problem:** Calculate the area of a rectangle given its length and width.
- **Solution:**

```
function rectangleArea($length, $width) {
    return $length * $width;
}
```

Problem 27: Area of a Square

- **Problem:** Calculate the area of a square given its side length.
- **Solution:**

```
function squareArea($sideLength) {
    return $sideLength * $sideLength;
}
```

Problem 28: Check if a Number is Perfect Square

- **Problem:** Check if a given number is a perfect square.
- **Solution:**

```
function isPerfectSquare($n) {
    $sqrt = sqrt($n);
    return (int)$sqrt == $sqrt;
}
```

Problem 29: Count Primes in a Range

- **Problem:** Count the number of prime numbers in a given range [a, b].
- **Solution:**

```
function countPrimesInRange($a, $b) {
    $count = 0;
    for ($i = $a; $i <= $b; $i++) {
        if (isPrime($i)) {
            $count++;
        }
    }
    return $count;
}
```

Problem 30: Sum of Natural Numbers

- **Problem:** Calculate the sum of the first 'n' natural numbers.
- **Solution:**

```
function sumOfNaturalNumbers($n) {
    return ($n * ($n + 1)) / 2;
}
```

Problem 31: Collatz Sequence

- **Problem:** Generate the Collatz sequence for a given positive integer.
- **Solution:**

```
function collatzSequence($n) {
    $sequence = [];
    while ($n != 1) {
        $sequence[] = $n;
        if ($n % 2 == 0) {
            $n = $n / 2;
        } else {
            $n = 3 * $n + 1;
        }
    }
    $sequence[] = 1;
    return $sequence;
}
```

Problem 32: HCF (Highest Common Factor) of Array

- **Problem:** Find the HCF of an array of numbers.
- **Solution:**

```
function hcfOfArray($numbers) {
    $hcf = $numbers[0];
    $count = count($numbers);
    for ($i = 1; $i < $count; $i++) {
        $hcf = gcd($hcf, $numbers[$i]); // You can use the 'gcd' function from Problem 4.
    }
    return $hcf;
}
```

Problem 33: LCM (Least Common Multiple) of Array

- **Problem:** Find the LCM of an array of numbers.
- **Solution:**

```
function lcmOfArray($numbers) {
    $lcm = $numbers[0];
    $count = count($numbers);
    for ($i = 1; $i < $count; $i++) {
        $lcm = lcm($lcm, $numbers[$i]); // You can use the 'lcm' function from Problem 5.
    }
    return $lcm;
}
```

Problem 34: Sum of First 'n' Fibonacci Numbers

- **Problem:** Calculate the sum of the first 'n' Fibonacci numbers.
- **Solution:**

```
function sumOfFibonacciNumbers($n) {
    $sum = 0;
    $fib = [0, 1];
    for ($i = 2; $i <= $n; $i++) {
        $fib[$i] = $fib[$i - 1] + $fib[$i - 2];
        $sum += $fib[$i];
    }
    return $sum;
}
```

Problem 35: Number of Divisors

- **Problem:** Calculate the number of divisors of a given integer.
- **Solution:**

```
function countDivisors($n) {
    $count = 0;
    for ($i = 1; $i <= sqrt($n); $i++) {
        if ($n % $i == 0) {
            if ($i == $n / $i) {
                $count++;
            } else {
                $count += 2;
            }
        }
    }
    return $count;
}
```

Problem 36: Check if a Number is a Perfect Cube

- **Problem:** Check if a given number is a perfect cube.
- **Solution:**

```
function isPerfectCube($n) {
    $cubeRoot = round(pow($n, 1/3));
    return $cubeRoot * $cubeRoot * $cubeRoot == $n;
}
```

Problem 37: Check if Two Numbers are Amicable

- **Problem:** Check if two numbers are amicable (pair of numbers where the sum of divisors of each is equal to the other number).
- **Solution:** ```php function areAmicable($a, $b) { return (array_sum(divisors($a)) == $b) && (array_sum(divisors($b)) == $

a); }

Problem 38: Largest Prime Factor
- **Problem:** Find the largest prime factor of a number.
- **Solution:**
```php
function largestPrimeFactor($n) {
    $factor = 2;
    while ($factor * $factor <= $n) {
        if ($n % $factor == 0) {
            $n = $n / $factor;
        } else {
            $factor++;
        }
    }
    return $n;
}
```

Problem 39: Fibonacci Modulo M

- **Problem:** Find the nth Fibonacci number modulo M.
- **Solution:**

```
function fibonacciModuloM($n, $M) {
    $fib = [0, 1];
    for ($i = 2; $i <= $n; $i++) {
        $fib[$i] = ($fib[$i - 1] + $fib[$i - 2]) % $M;
    }
    return $fib[$n];
}
```

Problem 40: Sum of Squares of First 'n' Natural Numbers

- **Problem:** Calculate the sum of the squares of the first 'n' natural numbers.
- **Solution:**

```
function sumOfSquaresOfNaturalNumbers($n) {
    return ($n * ($n + 1) * (2 * $n + 1)) / 6;
}
```

Constants

Certainly! Here are 10 important problems and solutions related to constants in PHP, suitable for PHP developers preparing for remote job interviews. These problems are explained and include code examples with a focus on competitive programming platforms like Codeforces, LeetCode, Kaggle, and CodeChef:

Problem 1: Defining a Constant

- **Problem:** Define a constant in PHP and demonstrate its usage.
- **Solution:**

```
// Define a constant for the value of Pi
define("PI", 3.14159265359);

// Usage example
$radius = 5;
$area = PI * $radius * $radius;
echo "The area of the circle is: " . $area;
```

Problem 2: Case-Insensitive Constants

- **Problem:** Create a case-insensitive constant in PHP.
- **Solution:**

```
// Define a case-insensitive constant
define("GREETING", "Hello, World!", true); // The third argument makes it case-insensitive

// Usage examples
echo GREETING; // Outputs "Hello, World!"
echo greeting; // Outputs "Hello, World!"
```

Problem 3: Using Constants in Switch Statements

- **Problem:** Utilize constants within a switch statement.
- **Solution:**

```
// Define constants for days of the week
define("MONDAY", 1);
define("TUESDAY", 2);
define("WEDNESDAY", 3);
define("THURSDAY", 4);
define("FRIDAY", 5);

$day = MONDAY;

switch ($day) {
    case MONDAY:
        echo "It's Monday!";
        break;
    case TUESDAY:
        echo "It's Tuesday!";
        break;
    // Handle other days...
    default:
        echo "It's another day.";
}
```

Problem 4: Constants as Array Values

- **Problem:** Store constants as values in an array.
- **Solution:**

```
// Define constants for colors
define("RED", "FF0000");
define("GREEN", "00FF00");
define("BLUE", "0000FF");

// Create an array of colors
$colors = [RED, GREEN, BLUE];

// Usage example
echo "The RGB value of Green is: " . $colors[1]; // Outputs "00FF00"
```

Problem 5: Magic Constants

- **Problem:** Explore the usage of magic constants in PHP.
- **Solution:**

```
// Example of using __LINE__ and __FILE__ magic constants
echo "This is line " . __LINE__ . " in file " . __FILE__;
```

Problem 6: Class Constants

- **Problem:** Define and use constants within a PHP class.
- **Solution:**

```
class MathOperations {
    const PI = 3.14159265359;

    public function calculateCircleArea($radius) {
        return self::PI * $radius * $radius;
    }
}

// Usage example
$math = new MathOperations();
echo "The area of the circle is: " . $math->calculateCircleArea(5);
```

Problem 7: Conditional Constants

- **Problem:** Use conditional logic to define constants based on a condition.
- **Solution:**

```
// Define a constant based on a condition
$isProduction = true;
define("API_URL", $isProduction ? "https://production-api.com" : "https://sandbox-api.com");

// Usage example
echo "API URL: " . API_URL;
```

Problem 8: Constants in Namespaces

- **Problem:** Define constants within a PHP namespace.
- **Solution:**

```
namespace MyNamespace;

// Define a constant within the namespace
const GREETING = "Hello from MyNamespace!";

// Usage example
echo GREETING;
```

Problem 9: Checking if a Constant is Defined

- **Problem:** Check if a constant is defined before using it.
- **Solution:**

```
if (defined("SOME_CONSTANT")) {
    echo "SOME_CONSTANT is defined.";
} else {
    echo "SOME_CONSTANT is not defined.";
}
```

Problem 10: Constant Reassignment

- **Problem:** Attempt to reassign a constant and observe the result.
- **Solution:**

```
 // Define a constant
define("MY_CONSTANT", "Initial Value");

// Attempt to reassign the constant (results in a warning)
define("MY_CONSTANT", "New Value");

// Usage example
echo MY_CONSTANT; // Outputs "Initial Value"
```

Operator :

-- Certainly! Here are 10 important problems and solutions related to operators in PHP, suitable for PHP developers preparing for remote job interviews. These problems are explained and include code examples with a focus on competitive programming platforms like Codeforces, LeetCode, Kaggle, and CodeChef:

Problem 1: Arithmetic Operators

- **Problem:** Perform basic arithmetic operations (addition, subtraction, multiplication, division) on two numbers.
- **Solution:**

```
 $a = 10;
$b = 5;

$sum = $a + $b;
$difference = $a - $b;
$product = $a * $b;
$quotient = $a / $b;

echo "Sum: $sum, Difference: $difference, Product: $product, Quotient: $quotient";
```

Problem 2: Increment and Decrement Operators

- **Problem:** Use increment and decrement operators to manipulate variables.
- **Solution:**

```
 $x = 5;

$x++; // Increment $x by 1
$y = ++$x; // Increment $x and assign its new value to $y

$z = 10;
$z--; // Decrement $z by 1
$w = --$z; // Decrement $z and assign its new value to $w

echo "x: $x, y: $y, z: $z, w: $w";
```

Problem 3: Comparison Operators

- **Problem:** Use comparison operators to compare two values.
- **Solution:**

```
 $num1 = 10;
$num2 = 20;

$isEqual = ($num1 == $num2); // Equal to
$isNotEqual = ($num1 != $num2); // Not equal to
$isGreaterThan = ($num1 > $num2); // Greater than
$isLessThan = ($num1 < $num2); // Less than
$isGreaterOrEqual = ($num1 >= $num2); // Greater than or equal to
$isLessOrEqual = ($num1 <= $num2); // Less than or equal to

echo "Equal: $isEqual, Not Equal: $isNotEqual, Greater Than: $isGreaterThan, Less Than: $isLessThan";
```

Problem 4: Logical Operators

- **Problem:** Use logical operators (AND, OR, NOT) to combine conditions.
- **Solution:**

```
$age = 25;
$isStudent = true;

$isAdult = ($age >= 18); // Check if age is 18 or older
$isAdultAndStudent = ($isAdult && $isStudent); // Check if both conditions are true
$isAdultOrStudent = ($isAdult || $isStudent); // Check if at least one condition is true
$isNotStudent = !$isStudent; // Negate the condition

echo "Adult: $isAdult, Adult and Student: $isAdultAndStudent, Adult or Student: $isAdultOrStudent";
```

Problem 5: Ternary Operator

- **Problem:** Use the ternary operator to assign values based on a condition.
- **Solution:**

```
$marks = 75;

$grade = ($marks >= 60) ? "Pass" : "Fail";

echo "Result: $grade";
```

Problem 6: Assignment Operators

- **Problem:** Utilize various assignment operators (+=, -=, *=, /=) to modify variables.
- **Solution:**

```
$total = 10;

$total += 5; // Equivalent to: $total = $total + 5;
$total -= 3; // Equivalent to: $total = $total - 3;
$total *= 2; // Equivalent to: $total = $total * 2;
$total /= 4; // Equivalent to: $total = $total / 4;

echo "Total: $total";
```

Problem 7: Bitwise Operators

- **Problem:** Perform bitwise operations (AND, OR, XOR) on integers.
- **Solution:**

```
$num1 = 5; // Binary: 101
$num2 = 3; // Binary: 011

$bitwiseAnd = $num1 & $num2; // Bitwise AND: 001 (decimal 1)
$bitwiseOr = $num1 | $num2; // Bitwise OR: 111 (decimal 7)
$bitwiseXor = $num1 ^ $num2; // Bitwise XOR: 110 (decimal 6)

echo "Bitwise AND: $bitwiseAnd, Bitwise OR: $bitwiseOr, Bitwise XOR: $bitwiseXor";
```

Problem 8: Conditional Operator Precedence

- **Problem:** Understand the precedence of operators in complex expressions.
- **Solution:**

```
$result = (5 + 3 * 2) / (4 - 2);
// The multiplication and division have higher precedence, so the result is 16 / 2 = 8.

echo "Result: $result";
```

Problem 9: String Concatenation Operator

- **Problem:** Concatenate strings using the string concatenation operator (.) in PHP.
- **Solution:**

```
$firstName = "John";
$lastName = "Doe";

$fullName = $firstName . " " . $lastName;

echo "Full Name: $fullName";
```

Problem 10: Type Comparison Operator

- **Problem:** Use the type comparison operator (===) to compare values and types.
- **Solution:**

```
$num1 = 10;
$num2 = "10";

$isEqual = ($num1 === $num2); // False because values are equal, but types are different

echo "Equal (with type check): " . ($isEqual ? "Yes" : "No");
```

Certainly! Here are 10 more problems and solutions related to operators in PHP, with no repetition:

Problem 11: Null Coalescing Operator

- **Problem:** Use the null coalescing operator (??) to provide a default value for a variable if it is null.
- **Solution:**

```
$name = $_GET['name'] ?? "Guest";

echo "Hello, $name";
```

Problem 12: Spaceship Operator (Combined Comparison Operator)

- **Problem:** Use the spaceship operator (<=>) to compare two values for sorting purposes.
- **Solution:**

```
$array = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5];

usort($array, function($a, $b) {
    return $a <=> $b; // Sort in ascending order
});

print_r($array);
```

Problem 13: Concatenate Assignment Operator

- **Problem:** Use the concatenate assignment operator (.=) to append a string to an existing string variable.
- **Solution:**

```
$message = "Hello, ";
$name = "John";

$message .= $name;

echo $message; // Outputs "Hello, John"
```

Problem 14: Conditional Assignment Operator

- **Problem:** Use the conditional assignment operator (??=) to assign a value to a variable if it is currently null.
- **Solution:**

```
$count = null;
$count ??= 5;

echo "Count: $count"; // Outputs "Count: 5"
```

Problem 15: Bitwise Shift Operators

- **Problem:** Perform bitwise left shift and right shift operations on integers.
- **Solution:**

```
$num = 8; // Binary: 1000

$leftShift = $num << 2; // Bitwise left shift by 2 positions: Binary: 100000 (decimal 32)
$rightShift = $num >> 1; // Bitwise right shift by 1 position: Binary: 100 (decimal 4)

echo "Left Shift: $leftShift, Right Shift: $rightShift";
```

Problem 16: Logical Null Coalescing Operator

- **Problem:** Use the logical null coalescing operator (??) with a condition to provide a default value.
- **Solution:**

```
$isAvailable = true;
$message = $isAvailable ? "Product is available" : "Product is not available";
$status = $isAvailable ?? "Unknown";

echo "Message: $message, Status: $status";
```

Problem 17: Operator Precedence in Expressions

- **Problem:** Understand and use operator precedence to evaluate complex expressions correctly.
- **Solution:**

```
$result = 10 + 5 * 2 > 8 && true || false;
// Operator precedence: Multiplication/Division > Addition/Subtraction > Comparison > Logical AND > Logical OR

echo "Result: " . ($result ? "True" : "False");
```

Problem 18: Exponentiation Operator

- **Problem:** Use the exponentiation operator (**) to calculate the power of a number.
- **Solution:**

```
$base = 2;
$exponent = 3;

$result = $base ** $exponent; // Calculates 2^3 = 8

echo "Result: $result";
```

Problem 19: Chaining Comparison Operators

- **Problem:** Chain multiple comparison operators to create complex conditions.
- **Solution:**

```
$age = 25;

$isTeenager = ($age > 12 && $age < 20);

echo "Is Teenager: " . ($isTeenager ? "Yes" : "No");
```

Problem 20: Identity Operator - Problem: Use the identity operator (===) to compare objects for identity. - Solution: `php $array1 = [1, 2, 3]; $array2 = [1, 2, 3]; $areIdentical = ($array1 === $array2); // False because they are two separate arrays echo "Are Identical: " . ($areIdentical ? "Yes" : "No");` ## If, If Else & Else

Certainly! Here are 20 important problems and solutions related to conditional statements (if, else, and elseif) in PHP, suitable for PHP developers preparing for remote job interviews. These problems are explained and include code examples with a focus on competitive programming platforms like Codeforces, LeetCode, Kaggle, and CodeChef:

Problem 1: Check Even or Odd

- **Problem:** Write a PHP program to check if a number is even or odd.
- **Solution:**

```
$num = 7;

if ($num % 2 == 0) {
    echo "Even";
} else {
    echo "Odd";
}
```

Problem 2: Check Positive, Negative, or Zero

- **Problem:** Write a PHP program to check if a number is positive, negative, or zero.
- **Solution:**

```
$num = -3;

if ($num > 0) {
    echo "Positive";
} elseif ($num < 0) {
    echo "Negative";
} else {
    echo "Zero";
}
```

Problem 3: Find Maximum of Three Numbers

- **Problem:** Write a PHP program to find the maximum of three numbers.
- **Solution:**

```
$a = 5;
$b = 8;
$c = 3;

if ($a >= $b && $a >= $c) {
    echo "Maximum: $a";
} elseif ($b >= $a && $b >= $c) {
    echo "Maximum: $b";
} else {
    echo "Maximum: $c";
}
```

Problem 4: Check Leap Year

- **Problem:** Write a PHP program to check if a given year is a leap year.
- **Solution:**

```
$year = 2024;

if (($year % 4 == 0 && $year % 100 != 0) || ($year % 400 == 0)) {
    echo "Leap Year";
} else {
    echo "Not a Leap Year";
}
```

Problem 5: Calculate Grade

- **Problem:** Write a PHP program to calculate the grade based on a student's score.
- **Solution:**

```
$score = 75;

if ($score >= 90) {
    echo "A";
} elseif ($score >= 80) {
    echo "B";
} elseif ($score >= 70) {
    echo "C";
} elseif ($score >= 60) {
    echo "D";
} else {
    echo "F";
}
```

Problem 6: Check Vowel or Consonant

- **Problem:** Write a PHP program to check if a given character is a vowel or a consonant.
- **Solution:**

```
$char = 'A';

if (in_array(strtoupper($char), ['A', 'E', 'I', 'O', 'U'])) {
    echo "Vowel";
} else {
    echo "Consonant";
}
```

Problem 7: Find the Largest of Four Numbers

- **Problem:** Write a PHP program to find the largest number among four numbers.
- **Solution:**

```
$num1 = 15;
$num2 = 8;
$num3 = 22;
$num4 = 12;

$max = $num1;

if ($num2 > $max) {
    $max = $num2;
}
if ($num3 > $max) {
    $max = $num3;
}
if ($num4 > $max) {
    $max = $num4;
}

echo "Largest Number: $max";
```

Problem 8: Check Prime Number

- **Problem:** Write a PHP program to check if a number is prime.
- **Solution:**

```
$num = 17;
$isPrime = true;

if ($num <= 1) {
    $isPrime = false;
} else {
    for ($i = 2; $i <= sqrt($num); $i++) {
        if ($num % $i == 0) {
            $isPrime = false;
            break;
        }
    }
}

if ($isPrime) {
    echo "Prime";
} else {
    echo "Not Prime";
}
```

Problem 9: Check Palindrome

- **Problem:** Write a PHP program to check if a given string is a palindrome.
- **Solution:**

```
$str = "racecar";
$isPalindrome = true;

$len = strlen($str);
for ($i = 0; $i < $len / 2; $i++) {
    if ($str[$i] != $str[$len - $i - 1]) {
        $isPalindrome = false;
        break;
    }
}

if ($isPalindrome) {
    echo "Palindrome";
} else {
    echo "Not a Palindrome";
}
```

Problem 10: Check if a Year is a Century

- **Problem:** Write a PHP program to check if a given year is a century (ends with 00).
- **Solution:**

```
$year = 1900;

if ($year % 100 == 0) {
    echo "Century";
} else {
    echo "Not a Century";
}
```

Problem 11: Check if a Number is Perfect Square

- **Problem:** Write a PHP program to check if a given number is a perfect square.
- **Solution:**

```
$num = 16;

$sqrt = sqrt($num);

if ($sqrt == (int)$sqrt) {
    echo "Perfect Square";
} else {
    echo "Not a Perfect Square";
}
```

Problem 12: Check if a String Contains a Substring

- **Problem:** Write a PHP program to check if a given string contains a specific substring.
- **Solution:**

```
$haystack = "This is a sample string.";
$needle = "sample";

if (strpos($haystack, $needle) !== false) {
    echo "Substring found";
} else {
    echo "Substring not found";
}
```

Problem 13: Check if a Year is a Leap Year (Alternate Approach)

- **Problem:** Write a PHP program to check if a given year is a leap year using an alternate approach.
- **Solution:**

```
$year = 2024;

if ($year % 4 == 0) {
    if ($year % 100 != 0 || $year % 400 == 0) {
        echo "Leap Year";
    } else {
        echo "Not a Leap Year";
    }
} else {
    echo "Not a Leap Year";
}
```

**Problem 14: Check if a String is a Palindrome

(Alternate Approach)**

- **Problem:** Write a PHP program to check if a given string is a palindrome using an alternate approach.
- **Solution:**

```
$str = "racecar";
$reversed = strrev($str);

if ($str === $reversed) {
    echo "Palindrome";
} else {
    echo "Not a Palindrome";
}
```

Problem 15: Check if a Number is Positive or Non-Positive

- **Problem:** Write a PHP program to check if a number is positive or non-positive.
- **Solution:**

```
$num = -5;

if ($num > 0) {
    echo "Positive";
} else {
    echo "Non-Positive";
}
```

Problem 16: Check if a Number is Negative or Non-Negative

- **Problem:** Write a PHP program to check if a number is negative or non-negative.
- **Solution:**

```
$num = 10;

if ($num < 0) {
    echo "Negative";
} else {
    echo "Non-Negative";
}
```

Problem 17: Calculate Discount Based on Purchase Amount

- **Problem:** Write a PHP program to calculate a discount based on the purchase amount.
- **Solution:**

```
$purchaseAmount = 150;
$discount = 0;

if ($purchaseAmount >= 100) {
    $discount = $purchaseAmount * 0.1; // 10% discount
}

echo "Discount: $discount";
```

Problem 18: Categorize Students Based on Marks

- **Problem:** Write a PHP program to categorize students into "Pass," "Fail," or "Topper" based on their marks.
- **Solution:**

```
$marks = 85;
$category = "";

if ($marks >= 90) {
    $category = "Topper";
} elseif ($marks >= 60) {
    $category = "Pass";
} else {
    $category = "Fail";
}

echo "Category: $category";
```

Problem 19: Check if a Character is a Letter

- **Problem:** Write a PHP program to check if a given character is a letter (alphabetic).
- **Solution:**

```
$char = '9';

if (ctype_alpha($char)) {
    echo "Letter";
} else {
    echo "Not a Letter";
}
```

Problem 20: Check if a Year is a Leap Year (Simplified Approach)

- **Problem:** Write a PHP program to check if a given year is a leap year using a simplified approach.
- **Solution:**

```
$year = 2024;

if ($year % 4 == 0 && ($year % 100 != 0 || $year % 400 == 0)) {
    echo "Leap Year";
} else {
    echo "Not a Leap Year";
}
```

Certainly! Here are 10 more complex if-else problems and solutions in PHP without repetition:

Problem 21: Find the Second Largest Number

- **Problem:** Write a PHP program to find the second largest number among an array of numbers.
- **Solution:**

```
$numbers = [8, 12, 5, 19, 3, 7];
$firstMax = PHP_INT_MIN;
$secondMax = PHP_INT_MIN;

foreach ($numbers as $num) {
    if ($num > $firstMax) {
        $secondMax = $firstMax;
        $firstMax = $num;
    } elseif ($num > $secondMax && $num != $firstMax) {
        $secondMax = $num;
    }
}

echo "Second Largest: $secondMax";
```

Problem 22: Check if a Triangle is Equilateral, Isosceles, or Scalene

- **Problem:** Write a PHP program to check if a triangle is equilateral, isosceles, or scalene based on its sides.
- **Solution:**

```
$side1 = 5;
$side2 = 5;
$side3 = 5;

if ($side1 == $side2 && $side2 == $side3) {
    echo "Equilateral Triangle";
} elseif ($side1 == $side2 || $side2 == $side3 || $side1 == $side3) {
    echo "Isosceles Triangle";
} else {
    echo "Scalene Triangle";
}
```

Problem 23: Check if a Number is a Perfect Number

- **Problem:** Write a PHP program to check if a number is a perfect number (sum of its divisors equals the number itself).
- **Solution:**

```
$num = 28;
$sum = 0;

for ($i = 1; $i <= $num / 2; $i++) {
    if ($num % $i == 0) {
        $sum += $i;
    }
}

if ($sum == $num) {
    echo "Perfect Number";
} else {
    echo "Not a Perfect Number";
}
```

Problem 24: Calculate BMI (Body Mass Index)

- **Problem:** Write a PHP program to calculate BMI and categorize it as underweight, normal, overweight, or obese.
- **Solution:**

```
$weight = 70; // in kilograms
$height = 1.75; // in meters

$bmi = $weight / ($height * $height);

if ($bmi < 18.5) {
    echo "Underweight";
} elseif ($bmi >= 18.5 && $bmi < 24.9) {
    echo "Normal";
} elseif ($bmi >= 24.9 && $bmi < 29.9) {
    echo "Overweight";
} else {
    echo "Obese";
}
```

Problem 25: Check if a Number is a Prime Palindrome

- **Problem:** Write a PHP program to check if a number is both prime and a palindrome.
- **Solution:**

```
function isPrime($num) {
    if ($num <= 1) {
        return false;
    }
    for ($i = 2; $i <= sqrt($num); $i++) {
        if ($num % $i == 0) {
            return false;
        }
    }
    return true;
}

function isPalindrome($num) {
    $numStr = (string)$num;
    return $numStr === strrev($numStr);
}

$number = 131;

if (isPrime($number) && isPalindrome($number)) {
    echo "Prime Palindrome";
} else {
    echo "Not a Prime Palindrome";
}
```

Problem 26: Calculate Electricity Bill

- **Problem:** Write a PHP program to calculate the electricity bill based on units consumed and categorize it as per rates.
- **Solution:**

```
$unitsConsumed = 300;
$billAmount = 0;

if ($unitsConsumed <= 50) {
    $billAmount = $unitsConsumed * 0.50;
} elseif ($unitsConsumed <= 150) {
    $billAmount = 50 * 0.50 + ($unitsConsumed - 50) * 0.75;
} elseif ($unitsConsumed <= 250) {
    $billAmount = 50 * 0.50 + 100 * 0.75 + ($unitsConsumed - 150) * 1.20;
} else {
    $billAmount = 50 * 0.50 + 100 * 0.75 + 100 * 1.20 + ($unitsConsumed - 250) * 1.50;
}

echo "Electricity Bill: $billAmount";
```

Problem 27: Check if a Number is a Strong Number

- **Problem:** Write a PHP program to check if a number is a strong number (the sum of the factorial of its digits equals the number itself).
- **Solution:**

```
function factorial($n) {
    if ($n == 0 || $n == 1) {
        return 1;
    }
    return $n * factorial($n - 1);
}

$num = 145;
$sum = 0;
$temp = $num;

while ($temp > 0) {
    $digit = $temp % 10;
    $sum += factorial($digit);
    $temp = (int)($temp / 10);
}

if ($sum == $num) {
    echo "Strong Number";
} else {
    echo "Not a Strong Number";
}
```

Problem 28: Check if a Number is a Harshad Number (Niven Number)

- **Problem:** Write a PHP program to check if a number is a Harshad number (a number that is divisible by the sum of its digits).
- **Solution:**

```
$num = 18;
$sum = 0;
$temp = $num;

while ($temp > 0) {
    $digit = $temp % 10;
    $sum += $digit;
    $temp = (int)($temp / 10);
}

if ($num % $sum == 0) {
    echo "Harshad Number";
} else {
    echo "Not a Harshad Number";
}
```

Problem 29: Check if a Year is a Magic Year

- **Problem:** Write a PHP program to check if a year is a magic year (a year whose day multiplied by month equals the last two digits of the year).
- **Solution:**

```
 $year = 2021;
$day = 10;
$month = 2;

if ($day * $month == $year % 100) {
    echo "Magic Year";
} else {
    echo "Not a Magic Year";
}
```

Problem 30: Check if a Year is a Happy Year

- **Problem:** Write a PHP program to check if a year is a happy year (a year whose sum of squares of digits is a prime number).
- **Solution:**

```
function isPrime($num) {
    if ($num <= 1) {
        return false;
    }
    for ($i = 2; $i <= sqrt($num); $i++) {
        if ($num % $i == 0) {
            return false;
        }
    }
    return true;
}

$year = 2021;
$sumOfSquares = 0;
$temp = $year;

while ($temp > 0) {
    $digit = $temp % 10;
    $sumOfSquares += $digit * $digit;
    $temp = (int)($temp / 10);
}

if (isPrime($sumOfSquares)) {
    echo "Happy Year";
} else {
    echo "Not a Happy Year";
}
```

Problem 31: check different conditions with if

- **Problem:** is_ functions and prints whether they meet the specified condition. You can see how each function works for different data types and conditions in one code view section.
- **Solution:**

```
$variable1 = null;
$variable2 = [1, 2, 3];
$variable3 = "Hello, world!";
$variable4 = true;
$variable5 = 42;
$variable6 = 3.14;
$variable7 = "123";

if (is_null($variable1)) {
    echo "Variable 1 is null.<br>";
} else {
    echo "Variable 1 is not null.<br>";
}

if (is_array($variable2)) {
    echo "Variable 2 is an array.<br>";
} else {
    echo "Variable 2 is not an array.<br>";
}

if (is_string($variable3)) {
    echo "Variable 3 is a string.<br>";
} else {
    echo "Variable 3 is not a string.<br>";
}

if (is_bool($variable4)) {
    echo "Variable 4 is a boolean.<br>";
} else {
    echo "Variable 4 is not a boolean.<br>";
}

if (is_int($variable5)) {
    echo "Variable 5 is an integer.<br>";
} else {
    echo "Variable 5 is not an integer.<br>";
}

if (is_float($variable6)) {
    echo "Variable 6 is a floating-point number.<br>";
} else {
    echo "Variable 6 is not a floating-point number.<br>";
}

if (is_numeric($variable7)) {
    echo "Variable 7 is numeric.<br>";
} else {
    echo "Variable 7 is not numeric.<br>";
}
```

Swhich Case

Problem 1: Days of the Week

- **Problem:** Write a PHP program that takes a number (1 to 7) as input and prints the corresponding day of the week.
- **Solution:**

```
$dayNumber = 3;

switch ($dayNumber) {
    case 1:
        echo "Monday";
        break;
    case 2:
        echo "Tuesday";
        break;
    case 3:
        echo "Wednesday";
        break;
    case 4:
        echo "Thursday";
        break;
    case 5:
        echo "Friday";
        break;
    case 6:
        echo "Saturday";
        break;
    case 7:
        echo "Sunday";
        break;
    default:
        echo "Invalid day number";
}
```

Problem 2: Calculator

- **Problem:** Write a PHP program that performs basic arithmetic operations (addition, subtraction, multiplication, division) based on user input.
- **Solution:**

```
$operator = '+';
$num1 = 10;
$num2 = 5;
$result = 0;

switch ($operator) {
    case '+':
        $result = $num1 + $num2;
        break;
    case '-':
        $result = $num1 - $num2;
        break;
    case '*':
        $result = $num1 * $num2;
        break;
    case '/':
        if ($num2 != 0) {
            $result = $num1 / $num2;
        } else {
            echo "Division by zero is not allowed.";
        }
        break;
    default:
        echo "Invalid operator";
}

echo "Result: $result";
```

Problem 3: Month Name

- **Problem:** Write a PHP program that takes a number (1 to 12) as input and prints the corresponding month name.
- **Solution:**

```php
$monthNumber = 9;

switch ($monthNumber) {
    case 1:
        echo "January";
        break;
    case 2:
        echo "February";
        break;
    case 3:
        echo "March";
        break;
    case 4:
        echo "April";
        break;
    case 5:
        echo "May";
        break;
    case 6:
        echo "June";
        break;
    case 7:
        echo "July";
        break;
    case 8:
        echo "August";
        break;
    case 9:
        echo "September";
        break;
    case 10:
        echo "October";
        break;
    case 11:
        echo "November";
        break;
    case 12:
        echo "December";
        break;
    default:
        echo "Invalid month number";
}
```

Problem 4: Grade Calculation

- **Problem:** Write a PHP program that calculates the grade based on a student's score using switch-case.
- **Solution:**

```php
$score = 85;
$grade = '';

switch (true) {
    case ($score >= 90):
        $grade = 'A';
        break;
    case ($score >= 80):
        $grade = 'B';
        break;
    case ($score >= 70):
        $grade = 'C';
        break;
    case ($score >= 60):
        $grade = 'D';
        break;
    default:
        $grade = 'F';
}

echo "Grade: $grade";
```

Problem 5: Language Selection

- **Problem:** Write a PHP program that displays a greeting message in different languages based on user input.
- **Solution:**

```
$language = 'fr'; // French

switch ($language) {
    case 'en':
        echo "Hello!";
        break;
    case 'fr':
        echo "Bonjour!";
        break;
    case 'es':
        echo "Â¡Hola!";
        break;
    case 'de':
        echo "Hallo!";
        break;
    default:
        echo "Language not supported";
}
```

Problem 6: Day Type (Weekday/Weekend)

- **Problem:** Write a PHP program that determines whether a given day is a weekday or a weekend day.
- **Solution:**

```
$dayNumber = 6; // 6 represents Saturday

switch ($dayNumber) {
    case 1: case 2: case 3: case 4: case 5:
        echo "Weekday";
        break;
    case 6: case 7:
        echo "Weekend";
        break;
    default:
        echo "Invalid day number";
}
```

Problem 7: Country and Currency

- **Problem:** Write a PHP program that takes a country code as input (e.g., "US" for the United States) and prints its currency symbol.
- **Solution:** ```php $countryCode = 'IN'; // India

  switch ($countryCode) { case 'US': echo "Currency: $"; break; case 'IN': echo "Currency: â‚¹"; break; case 'EU': echo

"Currency: â‚¬"; break; default: echo "Currency not found"; }

Problem 8: Menu Selection
- **Problem:** Write a PHP program for a restaurant menu that takes a menu item number as input and displays the selected item.
- **Solution:**
```php
$menuItem = 3; // Represents a specific menu item

switch ($menuItem) {
    case 1:
        echo "You selected: Hamburger";
        break;
    case 2:
        echo "You selected: Pizza";
        break;
    case 3:
        echo "You selected: Salad";
        break;
    case 4:
        echo "You selected: Pasta";
        break;
    default:
        echo "Invalid menu item";
}
```

Problem 9: Determine Working Hours

- **Problem:** Write a PHP program that determines whether a given time is within working hours (9 AM to 5 PM).
- **Solution:**

```
$hour = 14; // 2 PM

switch (true) {
    case ($hour >= 9 && $hour <= 17):
        echo "Working Hours";
        break;
    default:
        echo "Non-Working Hours";
}
```

Problem 10: Clothing Size Conversion

- **Problem:** Write a PHP program that converts clothing sizes from one country's standard to another based on user input.
- **Solution:**

```
$sizeUS = 'M';
$convertedSize = '';

switch ($sizeUS) {
    case 'S':
        $convertedSize = 'XS';
        break;
    case 'M':
        $convertedSize = 'S';
        break;
    case 'L':
        $convertedSize = 'M';
        break;
    case 'XL':
        $convertedSize = 'L';
        break;
    default:
        echo "Size not found";
}

echo "Converted Size: $convertedSize";
```

Problem 11: Determine Age Group

- **Problem:** Write a PHP program that determines which age group a person falls into based on their age.
- **Solution:**

```
$age = 35;
$ageGroup = '';

switch (true) {
    case ($age >= 0 && $age <= 12):
        $ageGroup = 'Child';
        break;
    case ($age >= 13 && $age <= 19):
        $ageGroup = 'Teenager';
        break;
    case ($age >= 20 && $age <= 59):
        $ageGroup = 'Adult';
        break;
    case ($age >= 60):
        $ageGroup = 'Senior';
        break;
    default:
        echo "Invalid age";
}

echo "Age Group: $ageGroup";
```

Problem 12: Determine Zodiac Sign

- **Problem:** Write a PHP program that determines a person's zodiac sign based on their birthdate (month and day).
- **Solution:**

```
$month = 8; // August
$day = 15;
$zodiacSign = '';

switch ($month) {
    case 1: // January
        if ($day <= 20) {
            $zodiacSign = 'Capricorn';
        } else {
            $zodiacSign = 'Aquarius';
        }
        break;
    case 2: // February
        if ($day <= 18) {
            $zodiacSign = 'Aquarius';
        } else {
            $zodiacSign = 'Pisces';
        }
        break;
    // Continue for all months...
    default:
        echo "Invalid birthdate";
}

echo "Zodiac Sign: $zodiacSign";
```

Problem 13: Determine BMI Category

- **Problem:** Write a PHP program that calculates BMI and determines the BMI category based on user input (height and weight).
- **Solution:**

```
$height = 1.75; // in meters
$weight = 70; // in kilograms
$bmi = $weight / ($height * $height);
$bmiCategory = '';

switch (true) {
    case ($bmi < 18.5):
        $bmiCategory = 'Underweight';
        break;
    case ($bmi >= 18.5 && $bmi < 24.9):
        $bmiCategory = 'Normal';
        break;
    case ($bmi >= 25 && $bmi < 29.9):
        $bmiCategory = 'Overweight';
        break;
    case ($bmi >= 30):
        $bmiCategory = 'Obese';
        break;
    default:
        echo "Invalid BMI";
}

echo "BMI: $bmi, Category: $bmiCategory";
```

Problem 14: Determine Vehicle Type

- **Problem:** Write a PHP program that determines the type of vehicle based on user input (car, motorcycle, bicycle, or unknown).
- **Solution:**

```
$vehicle = 'motorcycle';
$vehicleType = '';

switch ($vehicle) {
    case 'car':
        $vehicleType = 'Motorized Vehicle';
        break;
    case 'motorcycle':
        $vehicleType = 'Motorized Vehicle';
        break;
    case 'bicycle':
        $vehicleType = 'Non-Motorized Vehicle';
        break;
    default:
        $vehicleType = 'Unknown Vehicle';
}

echo "Vehicle Type: $vehicleType";
```

Problem 15: Determine Color Name

- **Problem:** Write a PHP program that determines the name of a color based on its hexadecimal code.
- **Solution:**

```
$hexCode = '#FF5733';
$colorName = '';

switch ($hexCode) {
    case '#FF5733':
        $colorName = 'Red';
        break;
    case '#00FF00':
        $colorName = 'Green';
        break;
    case '#0000FF':
        $colorName = 'Blue';
        break;
    default:
        $colorName = 'Unknown Color';
}

echo "Color Name: $colorName";
```

Problem 16: Determine Zodiac Element

- **Problem:** Write a PHP program that determines a person's zodiac element based on their

zodiac sign.

- **Solution:**

```
$zodiacSign = 'Leo';
$zodiacElement = '';

switch ($zodiacSign) {
    case 'Aries':
    case 'Leo':
    case 'Sagittarius':
        $zodiacElement = 'Fire';
        break;
    case 'Taurus':
    case 'Virgo':
    case 'Capricorn':
        $zodiacElement = 'Earth';
        break;
    case 'Gemini':
    case 'Libra':
    case 'Aquarius':
        $zodiacElement = 'Air';
        break;
    case 'Cancer':
    case 'Scorpio':
    case 'Pisces':
        $zodiacElement = 'Water';
        break;
    default:
        echo "Invalid zodiac sign";
}

echo "Zodiac Element: $zodiacElement";
```

Problem 17: Determine Day Type (Weekday/Weekend) for a Month Day

- **Problem:** Write a PHP program that determines whether a given day (e.g., 25th) of a month is a weekday or a weekend day.
- **Solution:**

```
$day = 25;
$month = 6; // June

// Assuming a 7-day week starting from Sunday
$dayOfWeek = ($day + 6) % 7;
$dayType = '';

switch ($dayOfWeek) {
    case 0:
    case 6:
        $dayType = 'Weekend';
        break;
    default:
        $dayType = 'Weekday';
}

echo "Day Type: $dayType";
```

Problem 18: Determine Olympic Medal

- **Problem:** Write a PHP program that determines the Olympic medal (Gold, Silver, Bronze, or None) based on a country's rank.
- **Solution:**

```
$rank = 2;
$medal = '';

switch ($rank) {
    case 1:
        $medal = 'Gold';
        break;
    case 2:
        $medal = 'Silver';
        break;
    case 3:
        $medal = 'Bronze';
        break;
    default:
        $medal = 'None';
}

echo "Medal: $medal";
```

Problem 19: Determine Weather Condition

- **Problem:** Write a PHP program that determines the weather condition (sunny, rainy, cloudy, or stormy) based on user input.
- **Solution:**

```
$weatherCode = 'S';
$weatherCondition = '';

switch ($weatherCode) {
    case 'S':
        $weatherCondition = 'Sunny';
        break;
    case 'R':
        $weatherCondition = 'Rainy';
        break;
    case 'C':
        $weatherCondition = 'Cloudy';
        break;
    case 'ST':
        $weatherCondition = 'Stormy';
        break;
    default:
        echo "Unknown weather code";
}

echo "Weather Condition: $weatherCondition";
```

Problem 20: Determine Vowel or Consonant - Problem: Write a PHP program that determines whether a given character is a vowel or a consonant. - Solution: `php $char = 'A'; $charType = ''; switch (strtoupper($char)) { case 'A': case 'E': case 'I': case 'O': case 'U': $charType = 'Vowel'; break; default: $charType = 'Consonant'; } echo "$charType: $char";` ## LOOPS : While, Do... While, For & Foreach

---

Loop Problems:

Problem 1: Print Numbers from 1 to 10 using a while loop

- **Problem:** Write a PHP program that prints numbers from 1 to 10 using a while loop.
- **Solution:**

```
$i = 1;
while ($i <= 10) {
    echo $i . " ";
    $i++;
}
```

Problem 2: Calculate the Sum of Numbers from 1 to 100 using a for loop

- **Problem:** Write a PHP program that calculates the sum of numbers from 1 to 100 using a for loop.
- **Solution:**

```
$sum = 0;
for ($i = 1; $i <= 100; $i++) {
    $sum += $i;
}
echo "Sum: $sum";
```

Problem 3: Print Even Numbers from 2 to 20 using a do-while loop

- **Problem:** Write a PHP program that prints even numbers from 2 to 20 using a do-while loop.
- **Solution:**

```
$i = 2;
do {
    echo $i . " ";
    $i += 2;
} while ($i <= 20);
```

Problem 4: Find the Factorial of a Number using a for loop

- **Problem:** Write a PHP program that calculates the factorial of a given number using a for loop.
- **Solution:**

```
$num = 5;
$factorial = 1;
for ($i = 1; $i <= $num; $i++) {
    $factorial *= $i;
}
echo "Factorial of $num: $factorial";
```

Problem 5: Calculate the Fibonacci Series

- **Problem:** Write a PHP program to calculate and print the Fibonacci series of a given length using a for loop.
- **Solution:**

```
$length = 10;
$a = 0;
$b = 1;

for ($i = 0; $i < $length; $i++) {
    echo $a . " ";
    $temp = $a + $b;
    $a = $b;
    $b = $temp;
}
```

Problem 6: Print Multiplication Table

- **Problem:** Write a PHP program that prints the multiplication table of a given number using a for loop.
- **Solution:**

```
$num = 7;
for ($i = 1; $i <= 10; $i++) {
    echo "$num x $i = " . ($num * $i) . "\n";
}
```

Problem 7: Calculate the Power of a Number

- **Problem:** Write a PHP program that calculates the power of a number (e.g., 2^3) using a for loop.
- **Solution:**

```
$base = 2;
$exponent = 3;
$result = 1;

for ($i = 1; $i <= $exponent; $i++) {
    $result *= $base;
}

echo "$base^$exponent = $result";
```

Problem 8: Find the GCD (Greatest Common Divisor) of Two Numbers

- **Problem:** Write a PHP program to find the GCD of two numbers using a while loop.
- **Solution:**

```
$num1 = 24;
$num2 = 36;

while ($num2 != 0) {
    $temp = $num2;
    $num2 = $num1 % $num2;
    $num1 = $temp;
}

echo "GCD: $num1";
```

Problem 9: Reverse a Number

- **Problem:** Write a PHP program to reverse a given number using a while loop.
- **Solution:**

```
$number = 12345;
$reversed = 0;

while ($number != 0) {
    $digit = $number % 10;
    $reversed = $reversed * 10 + $digit;
    $number = (int)($number / 10);
}

echo "Reversed Number: $reversed";
```

Problem 10: Check if a Number is Prime using a for loop

- **Problem:** Write a PHP program to check if a given number is prime using a for loop.
- **Solution:**

```
$num = 29;
$isPrime = true;

for ($i = 2; $i <= sqrt($num); $i++) {
    if ($num % $i == 0) {
        $isPrime = false;
        break;
    }
}

if ($isPrime) {
    echo "$num is a prime number";
} else {
    echo "$num is not a prime number";
}
```

Problem 11: Find the Sum of Even Numbers in an Array using foreach

- **Problem:** Write a PHP program to find the sum of even numbers in an array using a foreach loop.
- **Solution:**

```
$numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
$sum = 0;

foreach ($numbers as $number) {
    if ($number % 2 == 0) {
        $sum += $number;
    }
}

echo "Sum of Even Numbers: $sum";
```

Problem 12: Count the Occurrences of a Character in a String using for loop

- **Problem:** Write a PHP program to count the occurrences of a specific character in a string using a for loop.
- **Solution:**

```
$text = "programming";
$charToCount = 'm';
$count = 0;

for ($i = 0; $i < strlen($text); $i++) {
    if ($text[$i] == $charToCount) {
        $count++;
    }
}

echo "Occurrences of '$charToCount': $count";
```

Problem 13: Find the Maximum and Minimum Values in an Array using foreach

- **Problem:** Write a PHP program to find the maximum and minimum values in an array using a foreach loop.
- **Solution:** ```php $numbers = [45, 67, 12, 89, 23, 56]; $max = PHP_INT_MIN; $min = PHP_INT_MAX;

  foreach ($numbers as $number) { if ($number > $max

) { $max = $number; } if ($number < $min) { $min = $number; } }

echo "Maximum Value: $max, Minimum Value: $min";

Problem 14: Check Palindrome Number
- **Problem:** Write a PHP program to check if a given number is a palindrome (reads the same forwards and backwards) using a while
- **Solution:**
```php
$number = 121;
$original = $number;
$reversed = 0;

while ($number != 0) {
    $digit = $number % 10;
    $reversed = $reversed * 10 + $digit;
    $number = (int)($number / 10);
}

if ($original == $reversed) {
    echo "$original is a palindrome";
} else {
    echo "$original is not a palindrome";
}
```

Problem 15: Find the Second Largest Number in an Array using foreach

- **Problem:** Write a PHP program to find the second largest number in an array using a foreach loop.
- **Solution:**

```
$numbers = [10, 25, 5, 40, 15, 30];
$max = PHP_INT_MIN;
$secondMax = PHP_INT_MIN;

foreach ($numbers as $number) {
    if ($number > $max) {
        $secondMax = $max;
        $max = $number;
    } elseif ($number > $secondMax && $number != $max) {
        $secondMax = $number;
    }
}

echo "Second Largest Number: $secondMax";
```

Problem 16: Remove Duplicates from an Array using foreach

- **Problem:** Write a PHP program to remove duplicate elements from an array using a foreach loop.
- **Solution:**

```
$numbers = [5, 2, 7, 2, 8, 5];
$uniqueNumbers = [];

foreach ($numbers as $number) {
    if (!in_array($number, $uniqueNumbers)) {
        $uniqueNumbers[] = $number;
    }
}

echo "Unique Numbers: " . implode(", ", $uniqueNumbers);
```

Problem 17: Find the Longest Word in a Sentence using for loop

- **Problem:** Write a PHP program to find the longest word in a sentence using a for loop.
- **Solution:**

```
$sentence = "The quick brown fox jumps over the lazy dog";
$words = explode(" ", $sentence);
$longestWord = '';

for ($i = 0; $i < count($words); $i++) {
    if (strlen($words[$i]) > strlen($longestWord)) {
        $longestWord = $words[$i];
    }
}

echo "Longest Word: $longestWord";
```

Problem 18: Print a Pyramid Pattern using nested for loops

- **Problem:** Write a PHP program to print a pyramid pattern of asterisks using nested for loops.
- **Solution:**

```
$height = 5;

for ($i = 1; $i <= $height; $i++) {
    for ($j = 1; $j <= $height - $i; $j++) {
        echo " ";
    }
    for ($k = 1; $k <= 2 * $i - 1; $k++) {
        echo "*";
    }
    echo "\n";
}
```

Problem 19: Find the Intersection of Two Arrays using foreach

- **Problem:** Write a PHP program to find the intersection of two arrays using a foreach loop.
- **Solution:**

```
$array1 = [1, 2, 3, 4, 5];
$array2 = [3, 4, 5, 6, 7];
$intersection = [];

foreach ($array1 as $item) {
    if (in_array($item, $array2)) {
        $intersection[] = $item;
    }
}

echo "Intersection: " . implode(", ", $intersection);
```

Problem 20: Find the Largest Palindrome Word in a Sentence

- **Problem:** Write a PHP program to find the largest palindrome word in a sentence using nested foreach loops.
- **Solution:**

```
$sentence = "A man is known by his deeds of level and civic duty.";
$words = explode(" ", $sentence);
$largestPalindrome = '';

foreach ($words as $word) {
    $word = preg_replace("/[^A-Za-z0-9]/", '', $word); // Remove non-alphanumeric characters
    $reversed = strrev($word);

    if (strtolower($word) == strtolower($reversed) && strlen($word) > strlen($largestPalindrome)) {
        $largestPalindrome = $word;
    }
}

echo "Largest Palindrome Word: $largestPalindrome";
```

Problem 21: Calculate the Mean (Average) of Numbers in an Array using foreach

- **Problem:** Write a PHP program to calculate the mean (average) of numbers in an array using a foreach loop.
- **Solution:**

```
$numbers = [12, 24, 36, 48, 60];
$sum = 0;

foreach ($numbers as $number) {
    $sum += $number;
}

$mean = $sum / count($numbers);
echo "Mean: $mean";
```

Problem 22: Check for Prime Numbers in a Range using a for loop

- **Problem:** Write a PHP program to check for prime numbers in a given range using a for loop.
- **Solution:**

```
$start = 10;
$end = 50;

for ($i = $start; $i <= $end; $i++) {
    $isPrime = true;
    for ($j = 2; $j <= sqrt($i); $j++) {
        if ($i % $j == 0) {
            $isPrime = false;
            break;
        }
    }
    if ($isPrime && $i > 1) {
        echo "$i ";
    }
}
```

Problem 23: Generate a Random Password

- **Problem:** Write a PHP program to generate a random password of a given length using a for loop.
- **Solution:**

```
$length = 8;
$characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
$password = '';

for ($i = 0; $i < $length; $i++) {
    $index = rand(0, strlen($characters) - 1);
    $password .= $characters[$index];
}

echo "Random Password: $password";
```

Problem 24: Count Vowels and Consonants in a String using foreach

- **Problem:** Write

a PHP program to count the number of vowels and consonants in a string using a foreach loop.

- **Solution:**

```
$text = "Hello, World!";
$vowels = 'AEIOUaeiou';
$numVowels = 0;
$numConsonants = 0;

foreach (str_split($text) as $char) {
    if (ctype_alpha($char)) {
        if (strpos($vowels, $char) !== false) {
            $numVowels++;
        } else {
            $numConsonants++;
        }
    }
}

echo "Vowels: $numVowels, Consonants: $numConsonants";
```

Problem 25: Calculate Compound Interest using a for loop

- **Problem:** Write a PHP program to calculate compound interest given principal, rate, and time using a for loop.
- **Solution:**

```
$principal = 1000;
$rate = 5; // 5% interest rate
$time = 3; // 3 years
$compoundInterest = $principal;

for ($i = 1; $i <= $time; $i++) {
    $compoundInterest += ($compoundInterest * $rate / 100);
}

echo "Compound Interest after $time years: $compoundInterest";
```

Problem 26: Find the Smallest Positive Missing Number in an Array using a while loop

- **Problem:** Write a PHP program to find the smallest positive missing number in an array using a while loop.
- **Solution:**

```
$numbers = [3, 4, -1, 1];
$n = count($numbers);
$i = 0;

while ($i < $n) {
    if ($numbers[$i] > 0 && $numbers[$i] <= $n && $numbers[$i] != $numbers[$numbers[$i] - 1]) {
        $temp = $numbers[$i];
        $numbers[$i] = $numbers[$temp - 1];
        $numbers[$temp - 1] = $temp;
    } else {
        $i++;
    }
}

for ($i = 0; $i < $n; $i++) {
    if ($numbers[$i] != $i + 1) {
        echo "Smallest Positive Missing Number: " . ($i + 1);
        break;
    }
}
```

Problem 27: Find the Maximum Subarray Sum using Kadane's Algorithm

- **Problem:** Write a PHP program to find the maximum subarray sum using Kadane's algorithm and a for loop.
- **Solution:**

```
$numbers = [-2, 1, -3, 4, -1, 2, 1, -5, 4];
$maxSum = $currentSum = $numbers[0];

for ($i = 1; $i < count($numbers); $i++) {
    $currentSum = max($numbers[$i], $currentSum + $numbers[$i]);
    $maxSum = max($maxSum, $currentSum);
}

echo "Maximum Subarray Sum: $maxSum";
```

Problem 28: Find the Median of Two Sorted Arrays

- **Problem:** Write a PHP program to find the median of two sorted arrays using a for loop.
- **Solution:**

  ```
   $nums1 = [1, 3];
  $nums2 = [2];
  $merged = [];

  $i = 0;
  $j = 0;

  while ($i < count($nums1) && $j < count($nums2)) {
      if ($nums1[$i] < $nums2[$j]) {
          $merged[] = $nums1[$i];
          $i++;
      } else {
          $merged[] = $nums2[$j];
          $j++;
      }
  }

  while ($i < count($nums1)) {
      $merged[] = $nums1[$i];
      $i++;
  }

  while ($j < count($nums2)) {
      $merged[] = $nums2[$j];
      $j++;
  }

  $total = count($merged);
  if ($total % 2 == 0) {
      $median = ($merged[$total / 2 - 1] + $merged[$total / 2]) / 2;
  } else {
      $median = $merged[($total - 1) / 2];
  }

  echo "Median: $median";
  ```

Problem 29: Calculate the LCM (Least Common Multiple) of Two Numbers using a while loop

- **Problem:** Write a PHP program to calculate the LCM of two numbers using a while loop.
- **Solution:** ```php $num1 = 12; $

num2 = 18;

$a = max($num1, $num2); $b = min($num1, $num2);

$lcm = $a;

while ($lcm % $b != 0) { $lcm += $a; }

echo "LCM of $num1 and $num2: $lcm";

Problem 30: Print Pascal's Triangle using a for loop
- **Problem:** Write a PHP program to print Pascal's Triangle of a given height using a for loop.
- **Solution:**
```php
$height = 5;
for ($i = 0; $i < $height; $i++) {
    $number = 1;
    for ($j = 0; $j < ($height - $i - 1); $j++) {
        echo " ";
    }
    for ($j = 0; $j <= $i; $j++) {
        echo $number . " ";
        $number = $number * ($i - $j) / ($j + 1);
    }
    echo "\n";
}
```

Continue & Break

Certainly! Here are 10 medium-level problems and solutions related to "continue" and "break" statements in PHP, suitable for a PHP developer interview test:

1. Problem: Write a PHP program to print all even numbers from 1 to 20 using a "continue" statement.

Solution:

```
for ($i = 1; $i <= 20; $i++) {
    if ($i % 2 != 0) {
        continue;
    }
    echo $i . " ";
}
```

2. Problem: Create a PHP script to find and display the first five prime numbers using a "break" statement when you reach the fifth prime.

Solution:

```
$count = 0;
$num = 2;

while ($count < 5) {
    $isPrime = true;
    for ($i = 2; $i * $i <= $num; $i++) {
        if ($num % $i == 0) {
            $isPrime = false;
            break;
        }
    }
    if ($isPrime) {
        echo $num . " ";
        $count++;
    }
    $num++;
}
```

3. Problem: Write a PHP program to find the sum of all numbers divisible by 3 or 5 between 1 and 100 using "continue" to skip non-divisible numbers.

Solution:

```
$sum = 0;

for ($i = 1; $i <= 100; $i++) {
    if ($i % 3 != 0 && $i % 5 != 0) {
        continue;
    }
    $sum += $i;
}

echo "Sum: " . $sum;
```

4. Problem: Create a PHP script that prints the Fibonacci sequence up to the first occurrence of a number greater than 1000 using a "break" statement.

Solution:

```
$prev = 0;
$current = 1;

while ($current <= 1000) {
    echo $current . " ";
    $temp = $current;
    $current = $current + $prev;
    $prev = $temp;
}
```

5. Problem: Write a PHP program to find and display the first three numbers that are both divisible by 2 and 3 using "break" when you find the third number.

Solution:

```
$count = 0;
$num = 1;

while ($count < 3) {
    if ($num % 2 == 0 && $num % 3 == 0) {
        echo $num . " ";
        $count++;
    }
    $num++;
}
```

6. Problem: Create a PHP script to calculate the factorial of a number, but stop the calculation if it exceeds 1000 using "break."

Solution:

```
$n = 5;
$factorial = 1;

for ($i = 1; $i <= $n; $i++) {
    $factorial *= $i;
    if ($factorial > 1000) {
        break;
    }
}

echo "Factorial: " . $factorial;
```

7. Problem: Write a PHP program to find and display the first three consecutive positive integers that are perfect squares using "break."

Solution:

```
$count = 0;
$num = 1;

while ($count < 3) {
    if (sqrt($num) == (int)sqrt($num)) {
        echo $num . " ";
        $count++;
    }
    $num++;
}
```

8. Problem: Create a PHP script to find the largest prime number less than 100 using "break."

Solution:

```
for ($num = 99; $num >= 2; $num--) {
    $isPrime = true;
    for ($i = 2; $i * $i <= $num; $i++) {
        if ($num % $i == 0) {
            $isPrime = false;
            break;
        }
    }
    if ($isPrime) {
        echo "Largest Prime Less Than 100: " . $num;
        break;
    }
}
```

9. Problem: Write a PHP program to print the numbers from 1 to 50 but skip multiples of 7 using "continue."

Solution:

```
for ($i = 1; $i <= 50; $i++) {
    if ($i % 7 == 0) {
        continue;
    }
    echo $i . " ";
}
```

10. Problem: Create a PHP script to find the smallest positive integer that is divisible by all integers from 1 to 10 using "break."

Solution:

```
$num = 1;

while (true) {
    $isDivisible = true;
    for ($i = 1; $i <= 10; $i++) {
        if ($num % $i != 0) {
            $isDivisible = false;
            break;
        }
    }
    if ($isDivisible) {
        echo "Smallest Integer Divisible by 1 to 10: " . $num;
        break;
    }
    $num++;
}
```

FUNCTIONS

Creating a list of 30 problems and solutions for a PHP developer interview test focused on functions, including medium and hard-level questions, is quite extensive. However, I can provide you with a condensed version of 10 problems and solutions with code examples to get you started. You can then expand upon this list to meet your specific needs.

1. Problem: Write a PHP function to check if a number is prime.

Solution:

```
function isPrime($num) {
    if ($num <= 1) return false;
    for ($i = 2; $i * $i <= $num; $i++) {
        if ($num % $i == 0) return false;
    }
    return true;
}
```

2. Problem: Implement a recursive function to calculate the Fibonacci sequence.

Solution:

```
function fibonacci($n) {
    if ($n <= 1) return $n;
    return fibonacci($n - 1) + fibonacci($n - 2);
}
```

3. Problem: Write a function to find the factorial of a number.

Solution:

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

4. Problem: Create a function to reverse a string.

Solution:

```
function reverseString($str) {
    return strrev($str);
}
```

5. Problem: Write a function to check if a string is a palindrome.

Solution:

```
function isPalindrome($str) {
    $str = strtolower($str);
    return $str == strrev($str);
}
```

6. Problem: Implement a function to calculate the power of a number.

Solution:

```
function power($base, $exponent) {
    return pow($base, $exponent);
}
```

7. Problem: Create a function to find the GCD (Greatest Common Divisor) of two numbers.

Solution:

```
function gcd($a, $b) {
    while ($b != 0) {
        $temp = $b;
        $b = $a % $b;
        $a = $temp;
    }
    return $a;
}
```

8. Problem: Write a function to generate all permutations of a string.

Solution:

```
function permute($str, $l, $r) {
    if ($l == $r) {
        echo $str . "\n";
    } else {
        for ($i = $l; $i <= $r; $i++) {
            $str = swap($str, $l, $i);
            permute($str, $l + 1, $r);
            $str = swap($str, $l, $i);
        }
    }
}

function swap($str, $i, $j) {
    $temp = str_split($str);
    $temp[$i] = $str[$j];
    $temp[$j] = $str[$i];
    return implode("", $temp);
}
```

9. Problem: Implement a function to find the Nth prime number.

Solution:

```
function nthPrime($n) {
    $count = 0;
    $num = 2;
    while ($count < $n) {
        if (isPrime($num)) {
            $count++;
        }
        $num++;
    }
    return $num - 1;
}
```

10. Problem: Write a function to calculate the factorial of a large number efficiently.

Solution:

```
function factorial($n) {
    $result = '1';
    for ($i = 2; $i <= $n; $i++) {
        $result = multiply($result, strval($i));
    }
    return $result;
}

function multiply($a, $b) {
    return strval(intval($a) * intval($b));
}
```

Certainly! Here are 10 more PHP function-related problems and solutions:

11. Problem: Create a function that finds the sum of all even numbers in an array.

Solution:

```
function sumOfEvenNumbers($arr) {
    $sum = 0;
    foreach ($arr as $num) {
        if ($num % 2 == 0) {
            $sum += $num;
        }
    }
    return $sum;
}
```

12. Problem: Write a function to count the number of vowels in a string.

Solution:

```
function countVowels($str) {
    $vowels = "AEIOUaeiou";
    $count = 0;
    for ($i = 0; $i < strlen($str); $i++) {
        if (strpos($vowels, $str[$i]) !== false) {
            $count++;
        }
    }
    return $count;
}
```

13. Problem: Implement a function to check if two strings are anagrams.

Solution:

```
function areAnagrams($str1, $str2) {
    $str1 = str_replace(' ', '', strtolower($str1));
    $str2 = str_replace(' ', '', strtolower($str2));
    return count_chars($str1, 1) == count_chars($str2, 1);
}
```

14. Problem: Create a function to find the square root of a number without using built-in functions.

Solution:

```
function sqrtWithoutBuiltIn($num) {
    $guess = $num;
    $epsilon = 0.00001;

    while (abs($guess * $guess - $num) >= $epsilon) {
        $guess = ($guess + $num / $guess) / 2.0;
    }
    return $guess;
}
```

15. Problem: Write a function to remove duplicates from an array.

Solution:

```
function removeDuplicates($arr) {
    return array_unique($arr);
}
```

16. Problem: Implement a function to find the longest common prefix of an array of strings.

Solution:

```
function longestCommonPrefix($strs) {
    if (empty($strs)) return "";
    $prefix = $strs[0];
    for ($i = 1; $i < count($strs); $i++) {
        while (strpos($strs[$i], $prefix) !== 0) {
            $prefix = substr($prefix, 0, -1);
            if (empty($prefix)) return "";
        }
    }
    return $prefix;
}
```

17. Problem: Create a function to find the maximum sum of a subarray within an array.

Solution:

```
function maxSubArray($nums) {
    $maxSum = $currentSum = $nums[0];
    for ($i = 1; $i < count($nums); $i++) {
        $currentSum = max($nums[$i], $currentSum + $nums[$i]);
        $maxSum = max($maxSum, $currentSum);
    }
    return $maxSum;
}
```

18. Problem: Write a function to reverse a linked list.

Solution:

```
class ListNode {
    public $val;
    public $next;
    function __construct($val) {
        $this->val = $val;
        $this->next = null;
    }
}

function reverseLinkedList($head) {
    $prev = null;
    while ($head !== null) {
        $next = $head->next;
        $head->next = $prev;
        $prev = $head;
        $head = $next;
    }
    return $prev;
}
```

19. Problem: Implement a function to check if a binary tree is a valid binary search tree (BST).

Solution:

```
class TreeNode {
    public $val;
    public $left;
    public $right;
    function __construct($val = 0, $left = null, $right = null) {
        $this->val = $val;
        $this->left = $left;
        $this->right = $right;
    }
}

function isValidBST($root, $min = null, $max = null) {
    if ($root === null) return true;
    if (($min !== null && $root->val <= $min) || ($max !== null && $root->val >= $max)) {
        return false;
    }
    return isValidBST($root->left, $min, $root->val) && isValidBST($root->right, $root->val, $max);
}
```

Problem 20: Function Basics Problem: Create a PHP function called `calculateArea` that calculates the area of a rectangle when given its length and width as parameters. Use this function to find the area of a rectangle with a length of 5 units and a width of 3 units.

Solution:

```
function calculateArea($length, $width) {
    return $length * $width;
}

$area = calculateArea(5, 3);
echo "The area of the rectangle is: " . $area . " square units.";
```

Explanation: This problem tests the candidate's understanding of basic PHP functions.

Problem 21: Function with Default Parameter Problem: Write a PHP function called `calculateDiscount` that calculates the final price of an item after applying a discount. The function should take two parameters: the item's price and the discount percentage (default to 10% if not provided). Calculate the final price for an item with a price of $50.

Solution:

```
function calculateDiscount($price, $discount = 10) {
    return $price - ($price * ($discount / 100));
}

$finalPrice = calculateDiscount(50);
echo "The final price after applying the default discount is: $" . $finalPrice;
```

Explanation: This problem assesses the candidate's knowledge of default function parameters in PHP.

Problem 22: Recursive Function Problem: Implement a PHP function called `factorial` that calculates the factorial of a given positive integer using recursion. Calculate the factorial of 5.

Solution:

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

$result = factorial(5);
echo "The factorial of 5 is: " . $result;
```

Explanation: This problem evaluates the candidate's ability to create recursive functions.

Problem 23: Function with Variable Number of Arguments Problem: Create a PHP function called `calculateAverage` that calculates the average of a variable number of arguments (numbers). Calculate the average of 5, 10, and 15.

Solution:

```
function calculateAverage(...$numbers) {
    $sum = array_sum($numbers);
    $count = count($numbers);
    return $sum / $count;
}

$average = calculateAverage(5, 10, 15);
echo "The average is: " . $average;
```

Explanation: This problem tests the candidate's understanding of functions with a variable number of arguments.

Problem 24: Callback Functions Problem: Create a PHP function called `applyFunction` that takes an array of numbers and a callback function. The `applyFunction` should apply the callback function to each element in the array and return the modified array. Use this function to double all elements in the array `[1, 2, 3, 4, 5]`.

Solution:

```
function applyFunction($array, $callback) {
    return array_map($callback, $array);
}

$numbers = [1, 2, 3, 4, 5];
$doubled = applyFunction($numbers, function ($num) {
    return $num * 2;
});

print_r($doubled);
```

Explanation: This problem assesses the candidate's knowledge of callback functions and array manipulation.

Problem 25: Function Scoping Problem: Explain the concept of variable scoping in PHP functions. Provide an example that demonstrates the difference between local and global scope.

Solution:

```
$globalVar = 10; // Global variable

function testScope() {
    $localVar = 5; // Local variable
    echo "Local variable: " . $localVar . "<br>";
    global $globalVar; // Accessing global variable
    echo "Global variable inside function: " . $globalVar . "<br>";
}

testScope();
echo "Global variable outside function: " . $globalVar;
```

Explanation: This problem evaluates the candidate's understanding of variable scoping within functions.

Problem 26: Function Efficiency Problem: Compare the efficiency of using `echo` within a function to print output versus returning a value and using `echo` outside the function. Provide an explanation and example code.

Solution:

```
// Using echo within a function
function printWithinFunction() {
    echo "Output from within the function.";
}

// Using return and echo outside the function
function returnOutsideFunction() {
    return "Output from the function.";
}

// Usage
printWithinFunction(); // Outputs immediately
$output = returnOutsideFunction();
echo $output; // Outputs after assigning the result
```

Explanation: This problem assesses the candidate's understanding of performance considerations when using `echo` within functions.

Problem 27: Anonymous Functions Problem: Create an anonymous PHP function that calculates the square of a number. Use this anonymous function to calculate the square of 7.

Solution:

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

$result = $square(7);
echo "The square of 7 is: " . $result;
```

Explanation: This problem tests the candidate's knowledge of anonymous functions.

Problem 28: Function Error Handling Problem: Create a PHP function called `divide` that takes two parameters, `dividend` and `divisor`, and returns the result of dividing `dividend` by `divisor`. Handle the case where `divisor` is zero by throwing an exception. Provide an example of using this function with appropriate error handling.

Solution:

```
function divide($dividend, $divisor) {
    if ($divisor === 0) {
        throw new Exception("Division by zero is not allowed.");
    }
    return $dividend / $divisor;
}

try {
    $result = divide(10, 2);
    echo "Result: " . $result;
} catch (Exception $e) {
    echo "Error: " . $e->getMessage();
}
```

Explanation: This problem evaluates the candidate's ability to handle errors in PHP functions.

Problem 29: Recursive Fibonacci Problem: Write a PHP function to calculate the nth Fibonacci number using recursion. Optimize the function to improve its efficiency. Calculate the 10th Fibonacci number.

Solution:

```
function fibonacci($n, $a = 0, $b = 1) {
    if ($n === 0) return $a;
    if ($n === 1) return $b;
    return fibonacci($n - 1, $b, $a + $b);
}

$nthFibonacci = fibonacci(10);
echo "The 10th Fibonacci number is: " . $nthFibonacci;
```

Explanation: This problem assesses the candidate's ability to create an efficient

recursive function for a classic algorithm.

Problem 30: Find Common Elements in Arrays

Problem Statement: Write a PHP function that finds and returns the common elements between two arrays.

Solution:

```
function findCommonElements($arr1, $arr2) {
    return array_intersect($arr1, $arr2);
}

$array1 = [1, 2, 3, 4, 5];
$array2 = [3, 4, 5, 6, 7];
$common = findCommonElements($array1, $array2);
echo "Common elements: " . implode(", ", $common);
```

Problem 31: Calculate Simple Interest

Problem Statement: Create a PHP function to calculate the simple interest given the principal amount, interest rate, and time period.

Solution:

```
function calculateSimpleInterest($principal, $rate, $time) {
    $interest = ($principal * $rate * $time) / 100;
    return $interest;
}

$principal = 1000; // Principal amount in dollars
$rate = 5; // Interest rate per year
$time = 2; // Time period in years
$interest = calculateSimpleInterest($principal, $rate, $time);
echo "Simple Interest: $" . $interest;
```

ARRAYS And Its Methods

Certainly! Here are 30 medium-level problems related to arrays and array built-in methods/functions in PHP, along with solutions and code examples. These problems are designed to help PHP developers prepare for interviews and competitive programming on platforms like Codeforces, LeetCode, Kaggle, and CodeChef.

Problem 1: Find the Maximum Element in an Array

Problem: Write a PHP function to find the maximum element in an array.

Solution:

```
function findMax($arr) {
    $max = $arr[0];
    foreach ($arr as $num) {
        if ($num > $max) {
            $max = $num;
        }
    }
    return $max;
}
```

Problem 2: Remove Duplicates from an Array

Problem: Remove duplicates from an array in PHP.

Solution:

```
function removeDuplicates($arr) {
    return array_unique($arr);
}
```

Problem 3: Reverse an Array

Problem: Reverse the elements of an array in PHP.

Solution:

```
function reverseArray($arr) {
    return array_reverse($arr);
}
```

Problem 4: Find the Intersection of Two Arrays

Problem: Find the intersection of two arrays in PHP.

Solution:

```
function arrayIntersection($arr1, $arr2) {
    return array_intersect($arr1, $arr2);
}
```

Problem 5: Check if Array is Palindrome

Problem: Check if an array is a palindrome (reads the same forwards and backward).

Solution:

```
function isPalindrome($arr) {
    return $arr === array_reverse($arr);
}
```

Problem 6: Rotate an Array

Problem: Rotate an array to the right by k steps.

Solution:

```
function rotateArray($arr, $k) {
    $k %= count($arr);
    $slice1 = array_slice($arr, 0, count($arr) - $k);
    $slice2 = array_slice($arr, count($arr) - $k);
    return array_merge($slice2, $slice1);
}
```

Problem 7: Find Missing Number in Array

Problem: Find the missing number in an array of numbers from 1 to n.

Solution:

```
function findMissingNumber($arr) {
    $n = count($arr) + 1;
    $expectedSum = ($n * ($n + 1)) / 2;
    $actualSum = array_sum($arr);
    return $expectedSum - $actualSum;
}
```

Problem 8: Remove Element from Array

Problem: Remove all instances of a specific element from an array.

Solution:

```
function removeElement($arr, $elem) {
    return array_filter($arr, function ($value) use ($elem) {
        return $value !== $elem;
    });
}
```

Problem 9: Find Third Maximum Number

Problem: Find the third maximum number in an array. If it doesn't exist, return the maximum.

Solution:

```
function thirdMax($arr) {
    $uniqueArr = array_unique($arr);
    rsort($uniqueArr);
    if (count($uniqueArr) >= 3) {
        return $uniqueArr[2];
    } else {
        return $uniqueArr[0];
    }
}
```

Problem 10: Implement Binary Search

Problem: Implement binary search on a sorted array.

Solution:

```
function binarySearch($arr, $target) {
    $left = 0;
    $right = count($arr) - 1;

    while ($left <= $right) {
        $mid = $left + floor(($right - $left) / 2);

        if ($arr[$mid] === $target) {
            return $mid;
        } elseif ($arr[$mid] < $target) {
            $left = $mid + 1;
        } else {
            $right = $mid - 1;
        }
    }

    return -1; // Element not found
}
```

Problem 11: Implement Merge Sort

Problem: Implement the merge sort algorithm for an array.

Solution:

```
function mergeSort($arr) {
    $count = count($arr);
    if ($count <= 1) {
        return $arr;
    }

    $middle = floor($count / 2);
    $left = array_slice($arr, 0, $middle);
    $right = array_slice($arr, $middle);

    $left = mergeSort($left);
    $right = mergeSort($right);

    return merge($left, $right);
}

function merge($left, $right) {
    $result = [];
    while (!empty($left) && !empty($right)) {
        if ($left[0] < $right[0]) {
            $result[] = array_shift($left);
        } else {
            $result[] = array_shift($right);
        }
    }
    return array_merge($result, $left, $right);
}
```

Problem 12: Find Two Sum

Problem: Find two numbers in an array that add up to a specific target sum.

Solution:

```
function twoSum($arr, $target) {
    $complements = [];

    for ($i = 0; $i < count($arr); $i++) {
        $complement = $target - $arr[$i];

        if (isset($complements[$complement])) {
            return [$complements[$complement], $i];
        }

        $complements[$arr[$i]] = $i;
    }

    return null; // No solution found
}
```

Problem 13: Implement Quick Sort

Problem: Implement the quick sort algorithm for an array.

Solution:

```
function quickSort($arr) {
    $count = count($arr);
    if ($count <= 1) {
        return $arr;
    }

    $pivot = $arr[0];
    $left = $right = [];

    for ($i = 1; $i < $count; $i++) {
        if ($arr[$i] < $pivot) {
            $left[] = $arr[$i];
        } else {
            $right[] = $arr[$i];
        }
    }

    $left = quickSort($left);
    $right = quickSort($right);

    return array_merge($left, [$pivot], $right);
}
```

Problem 14: Find Subarray with Maximum Sum

Problem: Find the contiguous subarray with the largest sum.

Solution:

```
function maxSubArray($arr) {
    $maxSum = $currentSum = $arr[0];

    for ($i = 1; $i < count($arr); $i++) {
        $currentSum = max($arr[$i], $currentSum + $arr[$i]);
        $maxSum = max($maxSum, $currentSum);
    }

    return $maxSum;
}
```

Problem 15: Implement Stack using an Array

Problem: Implement a stack data structure using an array with push, pop, and top operations.

Solution:

```
class Stack {
    private $data = [];

    public function push($value) {
        array_push($this->data, $value);
    }

    public function pop() {
        if (!$this->isEmpty()) {
            return array_pop($this->data);
        }
        return null; // Stack is empty
    }

    public function top() {
        if (!$this->isEmpty()) {
            return end($this->data);
        }
        return null; // Stack is empty
    }

    public function isEmpty() {
        return empty($this->data);
    }
}
```

Problem 16: Implement Queue using an Array

Problem: Implement a queue data structure using an array with enqueue, dequeue, and front operations.

Solution:

```
class Queue {
    private $data = [];

    public function enqueue($value) {
        array_push($this->data, $value);
    }

    public function dequeue() {
        if (!$this->isEmpty()) {
            return array_shift($this->data);
        }
        return null; // Queue is empty
    }

    public function front() {
        if (!$this->isEmpty()) {
            return $this->data[0];
        }
        return null; // Queue is empty
    }

    public function isEmpty() {
        return empty($this->data);
    }
}
```

Problem 17: Find Minimum Number of Platforms

Problem: Given arrival and departure times of trains, find the minimum number of platforms needed.

Solution:

```
function minPlatforms($arrivals, $departures) {
    sort($arrivals);
    sort($departures);

    $platformsNeeded = 1;
    $maxPlatforms = 1;

    $i = 1;
    $j = 0;

    while ($i < count($arrivals) && $j < count($departures)) {
        if ($arrivals[$i] <= $departures[$j]) {
            $platformsNeeded++;
            $i++;
            if ($platformsNeeded > $maxPlatforms) {
                $maxPlatforms = $platformsNeeded;
            }
        } else {
            $platformsNeeded--;
            $j++;
        }
    }

    return $maxPlatforms;
}
```

Problem 18: Find Longest Consecutive Sequence

Problem: Find the length of the longest consecutive elements sequence in an unsorted array.

Solution:

```
function longestConsecutive($arr) {
    $set = array_flip($arr);
    $maxStreak = 0;

    foreach ($arr as $num) {
        if (!isset($set[$num - 1])) {
            $currentNum = $num;
            $currentStreak = 1;

            while (isset($set[$currentNum + 1])) {
                $currentNum++;
                $currentStreak++;
            }

            $maxStreak = max($maxStreak, $currentStreak);
        }
    }

    return $maxStreak;
}
```

Problem 19: Find Majority Element

Problem: Find the majority element (element that appears more than n/2 times) in an array.

Solution:

```
function majorityElement($arr) {
    $count = count($arr);
    $majority = null;
    $countMajority = 0;

    foreach ($arr as $num) {
        if ($countMajority === 0) {
            $majority = $num;
            $countMajority = 1;
        } elseif ($majority === $num) {
            $countMajority++;
        } else {
            $countMajority--;
        }
    }

    return $majority;
}
```

Problem 20: Implement Trie (Prefix Tree)

Problem: Implement a trie (prefix tree) data structure in PHP.

Solution:

```
class TrieNode {
    public $children = [];
    public $isEndOfWord = false;
}

class Trie {
    private $root;

    public function __construct() {
        $this->root = new TrieNode();
    }

    public function insert($word) {
        $node = $this->root;
        for ($i = 0; $i < strlen($word); $i++) {
            $char = $word[$i];
            if (!isset($node->children[$char])) {
                $node->children[$char] = new TrieNode();
            }
            $node = $node->children[$char];
        }
        $node->isEndOfWord = true;
    }

    public function search($word) {
        $node = $this->root;
        for ($i = 0; $i < strlen($word); $i++) {
            $char = $word[$i];
            if (!isset($node->children[$char])) {
                return false;
            }
            $node = $node->children[$char];
        }
        return $node->isEndOfWord;
    }

    public function startsWith($prefix) {
        $node = $this->root;
        for ($i = 0; $i < strlen($prefix); $i++) {
            $char = $prefix[$i];
            if (!isset($node->children[$char])) {
                return false;
            }
            $node = $node->children[$char];
        }
        return true;
    }
}
```

Problem 21: Implement a Circular Queue

Problem: Implement a circular queue data structure in PHP with enqueue, dequeue, and isEmpty operations.

Solution:

```
class CircularQueue {
    private $data = [];
    private $front = 0;
    private $rear = -1;
    private $size = 0;

    public function enqueue($value) {
        if ($this->size < count($this->data)) {
            $this->rear = ($this->rear + 1) % count($this->data);
            $this->data[$this->rear] = $value;
            $this->size++;
        }
    }

    public function dequeue() {
        if ($this->size > 0) {
            $value = $this->data[$this->front];
            $this->front = ($this->front + 1) % count($this->data);
            $this->size--;
            return $value;
        }
        return null; // Queue is empty
    }

    public function isEmpty() {
        return $this->size === 0;
    }
}
```

Problem 22: Implement a Min-Heap

Problem: Implement a min-heap data structure in PHP with insert, extractMin, and isEmpty operations.

Solution:

```
class MinHeap {
    private $data = [];

    public function insert($value) {
        array_push($this->data, $value);
        $this->heapifyUp();
    }

    public function extractMin() {
        if (!$this->isEmpty()) {
            $min = $this->data[0];
            $last = array_pop($this->data);

            if (!empty

($this->data)) {
                $this->data[0] = $last;
                $this->heapifyDown();
            }

            return $min;
        }
        return null; // Heap is empty
    }

    public function isEmpty() {
        return empty($this->data);
    }

    private function heapifyUp() {
        $index = count($this->data) - 1;

        while ($index > 0) {
            $parentIndex = ($index - 1) >> 1;
            if ($this->data[$index] < $this->data[$parentIndex]) {
                // Swap with parent
                list($this->data[$index], $this->data[$parentIndex]) = array($this->data[$parentIndex], $this->data[$index]);
                $index = $parentIndex;
            } else {
                break;
            }
        }
    }

    private function heapifyDown() {
        $index = 0;
        $count = count($this->data);

        while ($index < $count) {
            $leftChildIndex = 2 * $index + 1;
            $rightChildIndex = 2 * $index + 2;
            $minIndex = $index;

            if ($leftChildIndex < $count && $this->data[$leftChildIndex] < $this->data[$minIndex]) {
                $minIndex = $leftChildIndex;
            }

            if ($rightChildIndex < $count && $this->data[$rightChildIndex] < $this->data[$minIndex]) {
                $minIndex = $rightChildIndex;
            }

            if ($minIndex !== $index) {
                // Swap with the smaller child
                list($this->data[$index], $this->data[$minIndex]) = array($this->data[$minIndex], $this->data[$index]);
                $index = $minIndex;
            } else {
                break;
            }
        }
    }
}
```

Problem 23: Find Kth Largest Element in an Array

Problem: Find the kth largest element in an unsorted array.

Solution:

```
function findKthLargest($arr, $k) {
    $heap = new MinHeap();

    foreach ($arr as $num) {
        $heap->insert($num);
        if (count($heap) > $k) {
            $heap->extractMin();
        }
    }

    return $heap->extractMin();
}
```

Problem 24: Implement an LRU Cache

Problem: Implement an LRU (Least Recently Used) cache in PHP.

Solution:

```
class LRUCache {
    private $capacity;
    private $cache;

    public function __construct($capacity) {
        $this->capacity = $capacity;
        $this->cache = new SplDoublyLinkedList();
    }

    public function get($key) {
        if (isset($this->cache[$key])) {
            // Move accessed item to the front
            $this->cache->unshift($this->cache->splice($key, 1));
            return $this->cache->top();
        }
        return -1; // Key not found
    }

    public function put($key, $value) {
        if (isset($this->cache[$key])) {
            // Update value and move to the front
            $this->cache->unshift($this->cache->splice($key, 1));
            $this->cache->top()->value = $value;
        } else {
            // Insert a new item
            $this->cache->unshift(new SplDoublyLinkedListNode($value));
            $this->cache->top()->key = $key;

            if (count($this->cache) > $this->capacity) {
                // Remove the least recently used item
                $this->cache->shift();
            }
        }
    }
}
```

Problem 25: Implement Depth-First Search (DFS)

Problem: Implement depth-first search (DFS) for a graph represented as an adjacency list.

Solution:

```
class Graph {
    private $adjacencyList = [];

    public function addEdge($vertex, $neighbor) {
        if (!isset($this->adjacencyList[$vertex])) {
            $this->adjacencyList[$vertex] = [];
        }
        $this->adjacencyList[$vertex][] = $neighbor;
    }

    public function dfs($startVertex) {
        $visited = [];
        $this->dfsHelper($startVertex, $visited);
    }

    private function dfsHelper($vertex, &$visited) {
        $visited[$vertex] = true;
        echo $vertex . " ";

        if (isset($this->adjacencyList[$vertex])) {
            foreach ($this->adjacencyList[$vertex] as $neighbor) {
                if (!isset($visited[$neighbor])) {
                    $this->dfsHelper($neighbor, $visited);
                }
            }
        }
    }
}
```

Problem 26: Implement Breadth-First Search (BFS)

Problem: Implement breadth-first search (BFS) for a graph represented as an adjacency list.

Solution:

```
class Graph {
    private $adjacencyList = [];

    public function addEdge($vertex, $neighbor) {
        if (!isset($this->adjacencyList[$vertex])) {
            $this->adjacencyList[$vertex] = [];
        }
        $this->adjacencyList[$vertex][] = $neighbor;
    }

    public function bfs($startVertex) {
        $visited = [];
        $queue = new SplQueue();
        $queue->enqueue($startVertex);
        $visited[$startVertex] = true;

        while (!$queue->isEmpty()) {
            $vertex = $queue->dequeue();
            echo $vertex . " ";

            if (isset($this->adjacencyList[$vertex])) {
                foreach ($this->adjacencyList[$vertex] as $neighbor) {
                    if (!isset($visited[$neighbor])) {
                        $queue->enqueue($neighbor);
                        $visited[$neighbor] = true;
                    }
                }
            }
        }
    }
}
```

Problem 27: Find Shortest Path in a Graph (Dijkstra's Algorithm)

Problem: Find the shortest path in a weighted graph using Dijkstra's algorithm.

Solution:

```
class Graph {
    private $vertices = [];

    public function addVertex($name) {
        $this->vertices[$name] = [];
    }

    public function addEdge($start, $end, $weight) {
        $this->vertices[$start][] = ['end' => $end, 'weight' => $weight];
        $this->vertices[$end][] = ['end' => $start, 'weight' => $weight]; // For undirected graph
    }

    public function dijkstra($start) {
        $distances = [];
        $visited = [];

        foreach (array_keys($this->vertices) as $vertex) {
            $distances[$vertex] = INF;
            $visited[$vertex] = false;
        }

        $distances[$start] = 0;

        for ($i =

 0; $i < count($this->vertices) - 1; $i++) {
            $minVertex = $this->minDistanceVertex($distances, $visited);
            $visited[$minVertex] = true;

            foreach ($this->vertices[$minVertex] as $neighbor) {
                $newDistance = $distances[$minVertex] + $neighbor['weight'];
                if ($newDistance < $distances[$neighbor['end']]) {
                    $distances[$neighbor['end']] = $newDistance;
                }
            }
        }

        return $distances;
    }

    private function minDistanceVertex($distances, $visited) {
        $minDistance = INF;
        $minVertex = null;

        foreach (array_keys($this->vertices) as $vertex) {
            if (!$visited[$vertex] && $distances[$vertex] < $minDistance) {
                $minDistance = $distances[$vertex];
                $minVertex = $vertex;
            }
        }

        return $minVertex;
    }
}
```

Problem 28: Serialize and Deserialize a Binary Tree

Problem: Implement serialization and deserialization of a binary tree.

Solution:

```
class TreeNode {
    public $val;
    public $left;
    public $right;

    public function __construct($val) {
        $this->val = $val;
        $this->left = null;
        $this->right = null;
    }
}

class BinaryTree {
    public function serialize($root) {
        if ($root === null) {
            return "null,";
        }
        $left = $this->serialize($root->left);
        $right = $this->serialize($root->right);
        return $root->val . "," . $left . $right;
    }

    public function deserialize($data) {
        $nodes = explode(",", $data);
        return $this->deserializeHelper($nodes);
    }

    private function deserializeHelper(&$nodes) {
        $val = array_shift($nodes);
        if ($val === "null") {
            return null;
        }
        $node = new TreeNode((int)$val);
        $node->left = $this->deserializeHelper($nodes);
        $node->right = $this->deserializeHelper($nodes);
        return $node;
    }
}
```

Problem 29: Implement a Trie with Search and Delete Operations

Problem: Implement a trie (prefix tree) data structure with search and delete operations.

Solution:

```php
class TrieNode {
    public $children = [];
    public $isEndOfWord = false;
}

class Trie {
    private $root;

    public function __construct() {
        $this->root = new TrieNode();
    }

    public function insert($word) {
        $node = $this->root;
        for ($i = 0; $i < strlen($word); $i++) {
            $char = $word[$i];
            if (!isset($node->children[$char])) {
                $node->children[$char] = new TrieNode();
            }
            $node = $node->children[$char];
        }
        $node->isEndOfWord = true;
    }

    public function search($word) {
        $node = $this->root;
        for ($i = 0; $i < strlen($word); $i++) {
            $char = $word[$i];
            if (!isset($node->children[$char])) {
                return false;
            }
            $node = $node->children[$char];
        }
        return $node->isEndOfWord;
    }

    public function delete($word) {
        $this->deleteHelper($this->root, $word, 0);
    }

    private function deleteHelper(&$node, $word, $index) {
        if ($index === strlen($word)) {
            if ($node->isEndOfWord) {
                $node->isEndOfWord = false;
            }
            return count($node->children) === 0;
        }

        $char = $word[$index];
        if (!isset($node->children[$char])) {
            return false;
        }

        $shouldDelete = $this->deleteHelper($node->children[$char], $word, $index + 1);

        if ($shouldDelete) {
            unset($node->children[$char]);
            return count($node->children) === 0;
        }

        return false;
    }
}
```

Problem 30: Find All Anagrams in a String

Problem: Find all the starting indices of anagrams of a given string in another string.

Solution:

```
function findAnagrams($s, $p) {
    $result = [];

    $pCount = [];
    foreach (str_split($p) as $char) {
        if (!isset($pCount[$char])) {
            $pCount[$char] = 0;
        }
        $pCount[$char]++;
    }

    $sCount = [];
    $left = 0;
    $right = 0;
    $matched = 0;

    while ($right < strlen($s)) {
        $char = $s[$right];
        if (!isset($sCount[$char])) {
            $sCount[$char] = 0;
        }
        $sCount[$char]++;

        if (isset($pCount[$char]) && $sCount[$char] === $pCount[$char]) {
            $matched++;
        }

        while ($matched === count($pCount)) {
            if ($right - $left + 1 === strlen($p)) {
                $result[] = $left;
            }

            $leftChar = $s[$left];
            if (isset($sCount[$leftChar])) {
                $sCount[$leftChar]--;
                if ($sCount[$leftChar] < $pCount[$leftChar]) {
                    $matched--;
                }
            }

            $left++;
        }

        $right++;
    }

    return $result;
}
```

Problem 31: Find the Minimum Element in an Array

Problem: Write a PHP function to find the minimum element in an array.

Solution:

```
function findMin($arr) {
    return min($arr);
}
```

Problem 32: Calculate the Sum of Array Elements

Problem: Calculate the sum of all elements in an array.

Solution:

```
function arraySum($arr) {
    return array_sum($arr);
}
```

Problem 33: Find the Average Value of Array Elements

Problem: Find the average value of elements in an array.

Solution:

```
function arrayAverage($arr) {
    $count = count($arr);
    if ($count === 0) {
        return 0;
    }
    return array_sum($arr) / $count;
}
```

Problem 34: Check if Array Contains a Specific Element

Problem: Check if a specific element exists in an array.

Solution:

```
function containsElement($arr, $element) {
    return in_array($element, $arr);
}
```

Problem 35: Find the Index of an Element in Array

Problem: Find the index of the first occurrence of a specific element in an array.

Solution:

```
function findElementIndex($arr, $element) {
    return array_search($element, $arr);
}
```

Problem 36: Reverse Words in a String

Problem: Reverse the order of words in a string.

Solution:

```
function reverseWords($str) {
    $words = explode(" ", $str);
    $reversed = array_reverse($words);
    return implode(" ", $reversed);
}
```

Problem 37: Count Vowels in a String

Problem: Count the number of vowels (a, e, i, o, u) in a string.

Solution:

```
function countVowels($str) {
    $vowels = ['a', 'e', 'i', 'o', 'u'];
    $count = 0;
    foreach (str_split($str) as $char) {
        if (in_array(strtolower($char), $vowels)) {
            $count++;
        }
    }
    return $count;
}
```

Problem 38: Remove Whitespace from a String

Problem: Remove all whitespace characters (spaces, tabs, and line breaks) from a string.

Solution:

```
function removeWhitespace($str) {
    return preg_replace('/\s+/', '', $str);
}
```

Problem 39: Merge Two Sorted Arrays

Problem: Merge two sorted arrays into a single sorted array.

Solution:

```
function mergeSortedArrays($arr1, $arr2) {
    return array_merge($arr1, $arr2);
}
```

Problem 40: Check for Palindrome

Problem: Check if a string is a palindrome (reads the same forwards and backward).

Solution:

```
function isPalindromeStr($str) {
    $str = strtolower(preg_replace('/[^A-Za-z0-9]/', '', $str));
    return $str === strrev($str);
}
```

Problem 41: Remove Duplicates from a String

Problem: Remove duplicate characters from a string.

Solution:

```
function removeDuplicatesStr($str) {
    return implode("", array_unique(str_split($str)));
}
```

Problem 42: Check for Anagrams

Problem: Check if two strings are anagrams of each other.

Solution:

```
function areAnagrams($str1, $str2) {
    $str1 = strtolower(preg_replace('/[^A-Za-z]/', '', $str1));
    $str2 = strtolower(preg_replace('/[^A-Za-z]/', '', $str2));
    return count_chars($str1, 1) === count_chars($str2, 1);
}
```

Problem 43: Find the Second Largest Element

Problem: Find the second largest element in an array.

Solution:

```
function findSecondLargest($arr) {
    rsort($arr);
    if (count($arr) >= 2) {
        return $arr[1];
    }
    return null;
}
```

Problem 44: Count Even and Odd Numbers

Problem: Count the number of even and odd numbers in an array.

Solution:

```
function countEvenOdd($arr) {
    $evenCount = 0;
    $oddCount = 0;
    foreach ($arr as $num) {
        if ($num % 2 === 0) {
            $evenCount++;
        } else {
            $oddCount++;
        }
    }
    return ["even" => $evenCount, "odd" => $oddCount];
}
```

Problem 45: Find the Intersection of Two Arrays (Set Intersection)

Problem: Find the intersection of two arrays (common elements).

Solution:

```
function arrayIntersection($arr1, $arr2) {
    return array_intersect($arr1, $arr2);
}
```

Problem 46: Calculate the Product of Array Elements

Problem: Calculate the product of all elements in an array.

Solution:

```
function arrayProduct($arr) {
    return array_product($arr);
}
```

Problem 47: Find the Maximum Occurring Element

Problem: Find the element that occurs the most times in an array.

Solution:

```
function maxOccurringElement($arr) {
    $count = array_count_values($arr);
    arsort($count);
    return key($count);
}
```

Problem 48: Shuffle an Array

Problem: Shuffle the elements of an array randomly.

Solution:

```
function shuffleArray($arr) {
    shuffle($arr);
    return $arr;
}
```

Problem 49: Find the Missing Number in a Range

Problem: Find the missing number in a range from 1 to n.

Solution:

```
function findMissingNumberInRange($arr) {
    $n = max($arr);
    $expectedSum = ($n * ($n + 1)) / 2;
    $actualSum = array_sum($arr);
    return $expectedSum - $actualSum;
}
```

Problem 50: Remove Empty Strings from an Array

Problem: Remove empty strings from an array.

Solution:

```
function removeEmptyStrings($arr) {
    return array_filter($arr, 'strlen');
}
```

Problem 51: Find the Keys of an Array

Problem: Get an array of keys from an associative array.

Solution:

```
function getArrayKeys($assocArray) {
    return array_keys($assocArray);
}
```

Problem 52: Find the Values of an Array

Problem: Get an array of values from an associative array.

Solution:

```
function getArrayValues($assocArray) {
    return array_values($assocArray);
}
```

Problem 53: Reverse an Array

Problem: Reverse the order of elements in an array.

Solution:

```
function reverseArray($arr) {
    return array_reverse($arr);
}
```

Problem 54: Sort an Array in Ascending Order

Problem: Sort an array of numbers in ascending order.

Solution:

```
function sortArrayAscending($arr) {
    sort($arr);
    return $arr;
}
```

Problem 55: Sort an Array in Descending Order

Problem: Sort an array of numbers in descending order.

Solution:

```
function sortArrayDescending($arr) {
    rsort($arr);
    return $arr;
}
```

Problem 56: Extract Unique Values from an Array

Problem: Extract unique values from an array.

Solution:

```
function uniqueValues($arr) {
    return array_unique($arr);
}
```

Problem 57: Merge Two Arrays (Preserve Keys)

Problem: Merge two arrays while preserving keys.

Solution:

```
function mergeArraysPreserveKeys($arr1, $arr2) {
    return $arr1 + $arr2;
}
```

Problem 58: Count the Frequency of Array Elements

Problem: Count the frequency of each element in an array.

Solution:

```
function elementFrequency($arr) {
    return array_count_values($arr);
}
```

Problem 59: Find the First Element of an Array

Problem: Get the first element of an array.

Solution:

```
function firstElement($arr) {
    return reset($arr);
}
```

Problem 60: Find the Last Element of an Array

Problem: Get the last element of an array.

Solution:

```
function lastElement($arr) {
    return end($arr);
}
```

Problem 61: Extract a Range of Elements from an Array

Problem: Extract a range of elements from an array.

Solution:

```
function arraySlice($arr, $start, $length) {
    return array_slice($arr, $start, $length);
}
```

Problem 62: Check if All Elements Satisfy a Condition

Problem: Check if all elements in an array satisfy a given condition.

Solution:

```
function allElementsSatisfyCondition($arr, $condition) {
    return array_reduce($arr, function ($carry, $item) use ($condition) {
        return $carry && $condition($item);
    }, true);
}
```

Problem 63: Sum Values with Initial Value

Problem: Calculate the sum of elements in an array with an initial value.

Solution:

```
function sumWithInitialValue($arr, $initialValue) {
    return array_reduce($arr, function ($carry, $item) {
        return $carry + $item;
    }, $initialValue);
}
```

Problem 64: Extract Elements with Callback Function

Problem: Extract elements from an array based on a callback function.

Solution:

```
function filterWithCallback($arr, $callback) {
    return array_filter($arr, $callback);
}
```

Problem 65: Find the Maximum Value in an Array

Problem: Find the maximum value in an array.

Solution:

```
function findMaxValue($arr) {
    return max($arr);
}
```

Problem 66: Find the Minimum Value in an Array

Problem: Find the minimum value in an array.

Solution:

```
function findMinValue($arr) {
    return min($arr);
}
```

Problem 67: Multiply Corresponding Elements of Two Arrays

Problem: Multiply corresponding elements of two arrays.

Solution:

```
function multiplyArrays($arr1, $arr2) {
    return array_map(function ($a, $b) {
        return $a * $b;
    }, $arr1, $arr2);
}
```

Problem 68: Split a String into an Array

Problem: Split a string into an array using a delimiter.

Solution:

```
function splitString($str, $delimiter) {
    return explode($delimiter, $str);
}
```

Problem 69: Find the Union of Two Arrays

Problem: Find the union of two arrays (combine unique elements from both arrays).

Solution:

```
function arrayUnion($arr1, $arr2) {
    return array_unique(array_merge($arr1, $arr2));
}
```

Problem 70: Find the Difference Between Two Arrays

Problem: Find the difference between two arrays (elements that exist in the first array but not in the second).

Solution:

```
function arrayDifference($arr1, $arr2) {
    return array_diff($arr1, $arr2);
}
```

Problem 71: Find Common Values Between Two Arrays

Problem: Find common values between two arrays.

Solution:

```
function commonValues($arr1, $arr2) {
    return array_intersect($arr1, $arr2);
}
```

Problem 72: Check if an Array is Associative

Problem: Check if an array is associative (contains string keys).

Solution:

```
function isAssociativeArray($arr) {
    return count(array_filter(array_keys($arr), 'is_string')) > 0;
}
```

Problem 73: Check if an Array is Indexed

Problem: Check if an array is indexed (contains only numeric keys).

Solution:

```
function isIndexedArray($arr) {
    return count(array_filter(array_keys($arr), 'is_numeric')) === count($arr);
}
```

Problem 74: Count Occurrences of a Value in an Array

Problem: Count the number of occurrences of a specific value in an array.

Solution:

```
function countOccurrences($arr, $value) {
    return array_count_values($arr)[$value] ?? 0;
}
```

Problem 75: Find the N Largest Elements in an Array

Problem: Find the N largest elements in an array.

Solution:

```
function findNLargest($arr, $n) {
    rsort($arr);
    return array_slice($arr, 0, $n);
}
```

Problem 76: Find the N Smallest Elements in an Array

Problem: Find the N smallest elements in an array.

Solution:

```
function findNSmallest($arr, $n) {
    sort($arr);
    return array_slice($arr, 0, $n);
}
```

Problem 77: Merge Two Arrays (Indexed)

Problem: Merge two arrays with numeric keys (indexed arrays).

Solution:

```
function mergeIndexedArrays($arr1, $arr2) {
    return array_merge($arr1, $arr2);
}
```

Problem 78: Shuffle an Associative Array

Problem: Shuffle the elements of an associative array while preserving key-value associations.

Solution:

```
function shuffleAssociativeArray($arr) {
    $keys = array_keys($arr);
    shuffle($keys);
    $shuffled = [];
    foreach ($keys as $key) {
        $shuffled[$key] = $arr[$key];
    }
    return $shuffled;
}
```

Problem 79: Check if All Elements are Numeric

Problem: Check if all elements in an array are numeric.

Solution:

```
function areAllNumeric($arr) {
    return count(array_filter($arr, 'is_numeric')) === count($arr);
}
```

Problem 80: Find the First Non-Repeating Element

Problem: Find the first non-repeating element in an array.

Solution:

```
function firstNonRepeating($arr) {
    $counts = array_count_values($arr);
    foreach ($arr as $element) {
        if ($counts[$element] === 1) {
            return $element;
        }
    }
    return null;
}
```

Problem 81: Sort an Associative Array by Values

Problem: Sort an associative array by its values in ascending order.

Solution:

```
function sortAssociativeArrayByValues($arr) {
    asort($arr);
    return $arr;
}
```

Problem 82: Sort an Associative Array by Keys

Problem: Sort an associative array by its keys in ascending order.

Solution:

```
function sortAssociativeArrayByKeys($arr) {
    ksort($arr);
    return $arr;
}
```

Problem 83: Check if Array Contains a Subarray

Problem: Check if an array contains another array (subarray).

Solution:

```
function containsSubarray($arr, $subarray) {
    return count(array_intersect($arr, $subarray)) === count($subarray);
}
```

Problem 84: Check if Array is Palindromic

Problem: Check if an array is palindromic (reads the same forwards and backward).

Solution:

```
function isPalindromicArray($arr) {
    return $arr === array_reverse($arr);
}
```

Problem 85: Calculate the Median of an Array

Problem: Calculate the median of a numeric array.

Solution:

```
function calculateMedian($arr) {
    sort($arr);
    $count = count($arr);
    if ($count % 2 === 0) {
        return ($arr[($count - 1) / 2] + $arr[$count / 2]) / 2;
    } else {
        return $arr[($count - 1) / 2];
    }
}
```

Problem 86: Replace Values in an Array

Problem: Replace all occurrences of a specific value in an array with another value.

Solution:

```
function replaceValue($arr, $search, $replace) {
    return array_map(function ($element) use ($search, $replace) {
        return ($element === $search) ? $replace : $element;
    }, $arr);
}
```

Problem 87: Split an Array into Chunks

Problem: Split an array into chunks of a specified size.

Solution:

```
function splitArrayIntoChunks($arr, $chunkSize) {
    return array_chunk($arr, $chunkSize);
}
```

Problem 88: Find the Sum of Squares of Array Elements

Problem: Calculate the sum of the squares of elements in an array.

Solution:

```
function sumOfSquares($arr) {
    return array_sum(array_map(function ($element) {
        return $element * $element;
    }, $arr));
}
```

Problem 89: Find the Product of Even Elements

Problem: Calculate the product of all even elements in an array.

Solution:

```
function productOfEvenElements($arr) {
    return array_reduce(array_filter($arr, function ($element) {
        return $element % 2 === 0;
    }), function ($carry, $element) {
        return $carry * $element;
    }, 1);
}
```

Problem 90: Remove Elements by Value

Problem: Remove all occurrences of a specific value from an array.

Solution:

```
function removeElementsByValue($arr, $value) {
    return array_diff($arr, [$value]);
}
```

Problem 91: Flatten a Multi-dimensional Array

Problem: Flatten a multi-dimensional array into a single-dimensional array.

Solution:

```
function flattenArray($arr) {
    $result = [];
    array_walk_recursive($arr, function ($element) use (&$result) {
        $result[] = $element;
    });
    return $result;
}
```

Problem 92: Find the Mode of an Array

Problem: Find the mode (most frequently occurring value) in an array.

Solution:

```
function findMode($arr) {
    $counts = array_count_values($arr);
    $maxCount = max($counts);
    $modes = array_keys(array_filter($counts, function ($count) use ($maxCount) {
        return $count === $maxCount;
    }));
    return $modes;
}
```

Problem 93: Calculate the Mean of an Array

Problem: Calculate the mean (average) of a numeric array.

Solution:

```
function calculateMean($arr) {
    return array_sum($arr) / count($arr);
}
```

Problem 94: Rotate an Array to the Left

Problem: Rotate an array to the left by a specified number of positions.

Solution:

```
function rotateArrayLeft($arr, $positions) {
    $positions %= count($arr);
    return array_merge(array_slice($arr, $positions), array_slice($arr, 0, $positions));
}
```

Problem 95: Find the Index of the Maximum Element

Problem: Find the index of the maximum element in an array.

Solution:

```
function indexOfMaxElement($arr) {
    return array_search(max($arr), $arr);
}
```

Problem 96: Check if an Array is Sorted

Problem: Check if an array is sorted in ascending order.

Solution:

```
function isSortedAscending($arr) {
    $sortedArr = $arr;
    sort($sortedArr);
    return $arr === $sortedArr;
}
```

Problem 97: Check if an Array is Palindromic

Problem: Check if an array is palindromic (reads the same forwards and backward).

Solution:

```
function isPalindromicArray($arr) {
    return $arr === array_reverse($arr);
}
```

Problem 98: Find the Unique Element in an Array

Problem: Find the unique element in an array where all other elements occur twice.

Solution:

```
function findUniqueElement($arr) {
    $unique = 0;
    foreach ($arr as $element) {
        $unique ^= $element;
    }
    return $unique;
}
```

Problem 99: Count Consecutive Subarrays

Problem: Count the number of consecutive subarrays in an array where the sum of elements is equal to a given target.

Solution:

```
function countConsecutiveSubarrays($arr, $targetSum) {
    $count = 0;
    $currentSum = 0;
    $start = 0;

    for ($end = 0; $end < count($arr); $end++) {
        $currentSum += $arr[$end];

        while ($currentSum > $targetSum) {
            $currentSum -= $arr[$start];
            $start++;
        }

        if ($currentSum === $targetSum) {
            $count++;
        }
    }

    return $count;
}
```

Problem 100: Find the First Missing Positive Integer

Problem: Find the first missing positive integer in an unsorted array.

Solution:

```
function firstMissingPositive($arr) {
    $n = count($arr);

    for ($i = 0; $i < $n; $i++) {
        while ($arr[$i] > 0 && $arr[$

i] <= $n && $arr[$arr[$i] - 1] !== $arr[$i]) {
            list($arr[$i], $arr[$arr[$i] - 1]) = [$arr[$arr[$i] - 1], $arr[$i]];
        }
    }

    for ($i = 0; $i < $n; $i++) {
        if ($arr[$i] !== $i + 1) {
            return $i + 1;
        }
    }

    return $n + 1;
}
```

Problem 101: Find the Longest Increasing Subarray

Problem: Find the length of the longest increasing subarray within an array.

Solution:

```
function longestIncreasingSubarray($arr) {
    $maxLen = $currentLen = 1;

    for ($i = 1; $i < count($arr); $i++) {
        if ($arr[$i] > $arr[$i - 1]) {
            $currentLen++;
        } else {
            $maxLen = max($maxLen, $currentLen);
            $currentLen = 1;
        }
    }

    return max($maxLen, $currentLen);
}
```

Problem 102: Find Common Elements in Three Arrays

Problem: Find the common elements present in three arrays.

Solution:

```
function commonElementsInThreeArrays($arr1, $arr2, $arr3) {
    return array_intersect($arr1, $arr2, $arr3);
}
```

Problem 103: Merge Two Sorted Arrays In-Place

Problem: Merge two sorted arrays in-place without using extra space.

Solution:

```
function mergeSortedArraysInPlace(&$nums1, $m, $nums2, $n) {
    $i = $m - 1;
    $j = $n - 1;
    $k = $m + $n - 1;

    while ($i >= 0 && $j >= 0) {
        if ($nums1[$i] > $nums2[$j]) {
            $nums1[$k] = $nums1[$i];
            $i--;
        } else {
            $nums1[$k] = $nums2[$j];
            $j--;
        }
        $k--;
    }

    while ($j >= 0) {
        $nums1[$k] = $nums2[$j];
        $j--;
        $k--;
    }
}
```

Problem 104: Find the Kth Largest Element in an Array

Problem: Find the kth largest element in an unsorted array.

Solution:

```
function findKthLargest($nums, $k) {
    sort($nums);
    return $nums[count($nums) - $k];
}
```

Problem 105: Find the Kth Smallest Element in an Array

Problem: Find the kth smallest element in an unsorted array.

Solution:

```
function findKthSmallest($nums, $k) {
    sort($nums);
    return $nums[$k - 1];
}
```

Certainly! Here are 20 more array-related problems that do not repeat any of the previous problems:

Problem 106: Find the Maximum Subarray Sum

Problem: Find the maximum sum of a contiguous subarray within an array of integers.

Solution:

```
function maxSubarraySum($arr) {
    $maxSum = $currentSum = $arr[0];

    for ($i = 1; $i < count($arr); $i++) {
        $currentSum = max($arr[$i], $currentSum + $arr[$i]);
        $maxSum = max($maxSum, $currentSum);
    }

    return $maxSum;
}
```

Problem 107: Calculate the Intersection of Two Sorted Arrays

Problem: Calculate the intersection of two sorted arrays.

Solution:

```
function intersectionOfSortedArrays($arr1, $arr2) {
    $result = [];
    $i = $j = 0;

    while ($i < count($arr1) && $j < count($arr2)) {
        if ($arr1[$i] === $arr2[$j]) {
            $result[] = $arr1[$i];
            $i++;
            $j++;
        } elseif ($arr1[$i] < $arr2[$j]) {
            $i++;
        } else {
            $j++;
        }
    }

    return $result;
}
```

Problem 108: Find the Missing Element in a Sorted Array

Problem: Find the missing element in a sorted array of consecutive integers.

Solution:

```
function findMissingElement($arr) {
    $n = count($arr);
    $total = ($n + 1) * ($n + 2) / 2;
    $actualSum = array_sum($arr);
    return $total - $actualSum;
}
```

Problem 109: Remove Duplicates from a Sorted Array

Problem: Remove duplicates from a sorted array in-place.

Solution:

```
function removeDuplicatesFromSortedArray(&$arr) {
    $n = count($arr);
    if ($n === 0) {
        return 0;
    }

    $writeIndex = 0;
    for ($i = 1; $i < $n; $i++) {
        if ($arr[$i] !== $arr[$writeIndex]) {
            $writeIndex++;
            $arr[$writeIndex] = $arr[$i];
        }
    }

    return $writeIndex + 1;
}
```

Problem 110: Rotate an Array to the Right

Problem: Rotate an array to the right by a specified number of positions.

Solution:

```
function rotateArrayRight($arr, $positions) {
    $positions %= count($arr);
    return array_merge(array_slice($arr, -$positions), array_slice($arr, 0, -$positions));
}
```

Problem 111: Count Inversions in an Array

Problem: Count the number of inversions in an array (pairs of elements in reverse order).

Solution:

```
function countInversions($arr) {
    $inversions = 0;
    $n = count($arr);

    for ($i = 0; $i < $n - 1; $i++) {
        for ($j = $i + 1; $j < $n; $j++) {
            if ($arr[$i] > $arr[$j]) {
                $inversions++;
            }
        }
    }

    return $inversions;
}
```

Problem 112: Find the Longest Subarray with Zero Sum

Problem: Find the length of the longest subarray with a sum of zero.

Solution:

```
function longestSubarrayWithZeroSum($arr) {
    $maxLen = 0;
    $sum = 0;
    $sumMap = [];

    for ($i = 0; $i < count($arr); $i++) {
        $sum += $arr[$i];

        if ($sum === 0) {
            $maxLen = max($maxLen, $i + 1);
        }

        if (array_key_exists($sum, $sumMap)) {
            $maxLen = max($maxLen, $i - $sumMap[$sum]);
        } else {
            $sumMap[$sum] = $i;
        }
    }

    return $maxLen;
}
```

Problem 113: Find the Maximum Product Subarray

Problem: Find the maximum product of a contiguous subarray within an array of integers.

Solution:

```
function maxProductSubarray($arr) {
    $maxProduct = $currentMax = $currentMin = $arr[0];

    for ($i = 1; $i < count($arr); $i++) {
        if ($arr[$i] < 0) {
            list($currentMax, $currentMin) = [$currentMin, $currentMax];
        }

        $currentMax = max($arr[$i], $currentMax * $arr[$i]);
        $currentMin = min($arr[$i], $currentMin * $arr[$i]);

        $maxProduct = max($maxProduct, $currentMax);
    }

    return $maxProduct;
}
```

Problem 114: Find All Triplets with Zero Sum

Problem: Find all unique triplets in an array that sum up to zero.

Solution:

```
function threeSumZero($arr) {
    $result = [];
    sort($arr);
    $n = count($arr);

    for ($i = 0; $i < $n - 2; $i++) {
        if ($i === 0 || ($i > 0 && $arr[$i] !== $arr[$i - 1])) {
            $target = 0 - $arr[$i];
            $left = $i + 1;
            $right = $n - 1;

            while ($left < $right) {
                if ($arr[$left] + $arr[$right] === $target) {
                    $result[] = [$arr[$i], $arr[$left], $arr[$right]];
                    while ($left < $right && $arr[$left] === $arr[$left + 1]) $left++;
                    while ($left < $right && $arr[$right] === $arr[$right - 1]) $right--;
                    $left++;
                    $right--;
                } elseif ($arr[$left] + $arr[$right] < $target) {
                    $left++;
                } else {
                    $right--;
                }
            }
        }
    }

    return $result;
}
```

Problem 115: Find All Quadruplets with a Given Sum

Problem: Find all unique quadruplets in an array that sum up to a given target.

Solution:

```
function fourSum($arr, $target) {
    $result = [];
    sort($arr);
    $n = count($arr);

    for ($i = 0; $i < $n - 3; $i++) {
        if ($i ===

 0 || ($i > 0 && $arr[$i] !== $arr[$i - 1])) {
            for ($j = $i + 1; $j < $n - 2; $j++) {
                if ($j === $i + 1 || ($j > $i + 1 && $arr[$j] !== $arr[$j - 1])) {
                    $target2 = $target - $arr[$i] - $arr[$j];
                    $left = $j + 1;
                    $right = $n - 1;

                    while ($left < $right) {
                        if ($arr[$left] + $arr[$right] === $target2) {
                            $result[] = [$arr[$i], $arr[$j], $arr[$left], $arr[$right]];
                            while ($left < $right && $arr[$left] === $arr[$left + 1]) $left++;
                            while ($left < $right && $arr[$right] === $arr[$right - 1]) $right--;
                            $left++;
                            $right--;
                        } elseif ($arr[$left] + $arr[$right] < $target2) {
                            $left++;
                        } else {
                            $right--;
                        }
                    }
                }
            }
        }
    }

    return $result;
}
```

Problem 116: Find the Median of Two Sorted Arrays

Problem: Find the median of two sorted arrays.

Solution:

```
function findMedianSortedArrays($nums1, $nums2) {
    $merged = array_merge($nums1, $nums2);
    sort($merged);
    $n = count($merged);

    if ($n % 2 === 0) {
        $mid1 = $n / 2;
        $mid2 = $mid1 - 1;
        return ($merged[$mid1] + $merged[$mid2]) / 2;
    } else {
        $mid = ($n - 1) / 2;
        return $merged[$mid];
    }
}
```

Problem 117: Find the Maximum Consecutive Ones in Binary Array

Problem: Find the maximum number of consecutive ones in a binary array.

Solution:

```
function maxConsecutiveOnes($arr) {
    $maxCount = 0;
    $currentCount = 0;

    foreach ($arr as $num) {
        if ($num === 1) {
            $currentCount++;
            $maxCount = max($maxCount, $currentCount);
        } else {
            $currentCount = 0;
        }
    }

    return $maxCount;
}
```

Problem 118: Remove Duplicates from Sorted Array II

Problem: Remove duplicates from a sorted array in-place. Allow at most two duplicates.

Solution:

```
function removeDuplicatesFromSortedArray2(&$arr) {
    $n = count($arr);
    if ($n <= 2) {
        return $n;
    }

    $writeIndex = 2;

    for ($i = 2; $i < $n; $i++) {
        if ($arr[$i] !== $arr[$writeIndex - 2]) {
            $arr[$writeIndex] = $arr[$i];
            $writeIndex++;
        }
    }

    return $writeIndex;
}
```

Problem 119: Implement Stack Using Queues

Problem: Implement a stack using two queues.

Solution:

```
class MyStack {
    private $queue1;
    private $queue2;

    function __construct() {
        $this->queue1 = new SplQueue();
        $this->queue2 = new SplQueue();
    }

    function push($x) {
        $this->queue2->enqueue($x);
        while (!$this->queue1->isEmpty()) {
            $this->queue2->enqueue($this->queue1->dequeue());
        }
        $this->queue1 = $this->queue2;
        $this->queue2 = new SplQueue();
    }

    function pop() {
        return $this->queue1->dequeue();
    }

    function top() {
        return $this->queue1->bottom();
    }

    function empty() {
        return $this->queue1->isEmpty();
    }
}
```

Problem 120: Implement Queue Using Stacks

Problem: Implement a queue using two stacks.

Solution:

```
class MyQueue {
    private $stack1;
    private $stack2;

    function __construct() {
        $this->stack1 = new SplStack();
        $this->stack2 = new SplStack();
    }

    function push($x) {
        $this->stack1->push($x);
    }

    function pop() {
        $this->moveStack1ToStack2();
        return $this->stack2->pop();
    }

    function peek() {
        $this->moveStack1ToStack2();
        return $this->stack2->top();
    }

    function empty() {
        return $this->stack1->isEmpty() && $this->stack2->isEmpty();
    }

    private function moveStack1ToStack2() {
        if ($this->stack2->isEmpty()) {
            while (!$this->stack1->isEmpty()) {
                $this->stack2->push($this->stack1->pop());
            }
        }
    }
}
```

Problem 121: Reverse Words in a String

Problem: Given a string, reverse the order of words.

Solution:

```
function reverseWords($s) {
    $words = explode(" ", $s);
    $reversed = array_reverse($words);
    return implode(" ", $reversed);
}
```

Problem 122: Find the Peak Element

Problem: Find the peak element in an array (an element that is greater than its neighbors).

Solution:

```
function findPeakElement($nums) {
    $left = 0;
    $right = count($nums) - 1;

    while ($left < $right) {
        $mid = floor(($left + $right) / 2);
        if ($nums[$mid] > $nums[$mid + 1]) {
            $right = $mid;
        } else {
            $left = $mid + 1;
        }
    }

    return $left;
}
```

Problem 123: Maximum Subarray Product

Problem: Find the maximum product of a contiguous subarray within an array of integers.

Solution:

```
function maxProduct($nums) {
    $maxProduct = $currentMax = $currentMin = $nums[0];

    for ($i = 1; $i < count($nums); $i++) {
        if ($nums[$i] < 0) {
            list($currentMax, $currentMin) = [$currentMin, $currentMax];
        }

        $currentMax = max($nums[$i], $currentMax * $nums[$i]);
        $currentMin = min($nums[$i], $currentMin * $nums[$i]);

        $maxProduct = max($maxProduct, $currentMax);
    }

    return $maxProduct;
}
```

Problem 124: Check for Palindromic Substrings

Problem: Given a string, find and count all palindromic substrings.

Solution:

```
function countPalindromicSubstrings($s) {
    $count = 0;
    $n = strlen($s);

    for ($i = 0; $i < $n; $i++) {
        $count += expandAroundCenter($s, $i, $i);      // Odd length palindromes
        $count += expandAroundCenter($s, $i, $i + 1);  // Even length palindromes
    }

    return $count;
}

function expandAroundCenter($s, $left, $right) {
    $count = 0;

    while ($left >= 0 && $right < strlen($s) && $s[$left] === $s[$right]) {
        $left--;
        $right++;
        $count++;
    }

    return $count;
}
```

Problem 125: Implement a Min Stack

Problem: Implement a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Solution:

```
class MinStack {
    private $stack;
    private $minStack;

    function __construct() {
        $this->stack = new SplStack();
        $this->minStack = new SplStack();
    }

    function push($x) {
        $this->stack->push($x);
        if ($this->minStack->isEmpty() || $x <= $this->minStack->top()) {
            $this->minStack->push($x);
        }
    }

    function pop() {
        if ($this->stack->top() === $this->minStack->top()) {
            $this->minStack->pop();
        }
        $this->stack->pop();
    }

    function top() {
        return $this->stack->top();
    }

    function getMin() {
        return $this->minStack->top();
    }
}
```

Problem 126: Search in Rotated Sorted Array

Problem: Search for a target value in a rotated sorted array.

Solution:

```
function search($nums, $target) {
    $left = 0;
    $right = count($nums) - 1;

    while ($left <= $right) {
        $mid = floor(($left + $right) / 2);

        if ($nums[$mid] === $target) {
            return $mid;
        }

        if ($nums[$left] <= $nums[$mid]) {
            if ($nums[$left] <= $target && $target < $nums[$mid]) {
                $right = $mid - 1;
            } else {
                $left = $mid + 1;
            }
        } else {
            if ($nums[$mid] < $target && $target <= $nums[$right]) {
                $left = $mid + 1;
            } else {
                $right = $mid - 1;
            }
        }
    }

    return -1;
}
```

Problem 127: Longest Consecutive Sequence

Problem: Find the length of the longest consecutive elements sequence in an unsorted array of integers.

Solution:

```
function longestConsecutive($nums) {
    $numSet = array_flip($nums);
    $maxStreak = 0;

    foreach ($numSet as $num => $val) {
        if (!isset($numSet[$num - 1])) {
            $currentNum = $num;
            $currentStreak = 1;

            while (isset($numSet[$currentNum + 1])) {
                $currentNum++;
                $currentStreak++;
            }

            $maxStreak = max($maxStreak, $currentStreak);
        }
    }

    return $maxStreak;
}
```

Problem 128: Find the Celebrity

Problem: You are given a table of people, and one of them is known as a celebrity. The celebrity doesn't know anyone, but everyone knows the celebrity. Find the celebrity in O(n) time.

Solution:

```
function findCelebrity($n) {
    $candidate = 0;

    for ($i = 1; $i < $n; $i++) {
        if (knows($candidate, $i)) {
            $candidate = $i;
        }
    }

    if (isCelebrity($candidate, $n)) {
        return $candidate;
    }

    return -1;
}
```

Problem 129: Sort an Array of 0s, 1s, and 2s (Dutch National Flag Problem)

Problem: Given an array containing only 0s, 1s, and 2s, sort it in-place.

Solution:

```
function sortColors(&$nums) {
    $left = 0;
    $right = count($nums) - 1;
    $curr = 0;

    while ($curr <= $right) {
        if ($nums[$curr] === 0) {
            list($nums[$left], $nums[$curr]) = [$nums[$curr],

 $nums[$left]];
            $left++;
            $curr++;
        } elseif ($nums[$curr] === 2) {
            list($nums[$curr], $nums[$right]) = [$nums[$right], $nums[$curr]];
            $right--;
        } else {
            $curr++;
        }
    }
}
```

Problem 130: Container With Most Water

Problem: Given an array of non-negative integers representing the heights of walls, find the maximum amount of water that can be trapped between two walls.

Solution:

```
function maxArea($height) {
    $maxWater = 0;
    $left = 0;
    $right = count($height) - 1;

    while ($left < $right) {
        $minHeight = min($height[$left], $height[$right]);
        $width = $right - $left;
        $maxWater = max($maxWater, $minHeight * $width);

        if ($height[$left] < $height[$right]) {
            $left++;
        } else {
            $right--;
        }
    }

    return $maxWater;
}
```

Problem 131: Move Zeros to the End of an Array

Problem: Given an array, move all zeros to the end of it while maintaining the relative order of the non-zero elements.

Solution:

```
function moveZeroes(&$nums) {
    $nonZeroIndex = 0;

    foreach ($nums as $num) {
        if ($num !== 0) {
            $nums[$nonZeroIndex++] = $num;
        }
    }

    while ($nonZeroIndex < count($nums)) {
        $nums[$nonZeroIndex++] = 0;
    }
}
```

Problem 132: First Missing Positive

Problem: Given an unsorted integer array, find the smallest missing positive integer.

Solution:

```
function firstMissingPositive($nums) {
    $n = count($nums);

    for ($i = 0; $i < $n; $i++) {
        while ($nums[$i] > 0 && $nums[$i] <= $n && $nums[$nums[$i] - 1] !== $nums[$i]) {
            list($nums[$i], $nums[$nums[$i] - 1]) = [$nums[$nums[$i] - 1], $nums[$i]];
        }
    }

    for ($i = 0; $i < $n; $i++) {
        if ($nums[$i] !== $i + 1) {
            return $i + 1;
        }
    }

    return $n + 1;
}
```

Problem 133: Generate Pascal's Triangle

Problem: Given a non-negative integer numRows, generate the first numRows of Pascal's triangle.

Solution:

```
function generatePascalsTriangle($numRows) {
    $triangle = [];

    for ($i = 0; $i < $numRows; $i++) {
        $row = [];
        for ($j = 0; $j <= $i; $j++) {
            if ($j === 0 || $j === $i) {
                $row[] = 1;
            } else {
                $row[] = $triangle[$i - 1][$j - 1] + $triangle[$i - 1][$j];
            }
        }
        $triangle[] = $row;
    }

    return $triangle;
}
```

Problem 134: Next Permutation

Problem: Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

Solution:

```
function nextPermutation(&$nums) {
    $n = count($nums);
    $i = $n - 2;

    while ($i >= 0 && $nums[$i] >= $nums[$i + 1]) {
        $i--;
    }

    if ($i >= 0) {
        $j = $n - 1;
        while ($j >= 0 && $nums[$j] <= $nums[$i]) {
            $j--;
        }
        list($nums[$i], $nums[$j]) = [$nums[$j], $nums[$i]];
    }

    reverseArray($nums, $i + 1, $n - 1);
}

function reverseArray(&$nums, $start, $end) {
    while ($start < $end) {
        list($nums[$start], $nums[$end]) = [$nums[$end], $nums[$start]];
        $start++;
        $end--;
    }
}
```

Problem 135: Longest Increasing Subsequence

Problem: Find the length of the longest increasing subsequence in an unsorted array of integers.

Solution:

```
function lengthOfLIS($nums) {
    $n = count($nums);
    $dp = array_fill(0, $n, 1);

    for ($i = 1; $i < $n; $i++) {
        for ($j = 0; $j < $i; $j++) {
            if ($nums[$i] > $nums[$j]) {
                $dp[$i] = max($dp[$i], $dp[$j] + 1);
            }
        }
    }

    return max($dp);
}
```

Problem 136: Find Common Elements in Multiple Arrays

Problem: You have multiple arrays, and you need to find the common elements present in all of them.

Solution:

```
function findCommonElements($arrays) {
    if (empty($arrays)) {
        return [];
    }

    $commonElements = $arrays[0];

    for ($i = 1; $i < count($arrays); $i++) {
        $commonElements = array_intersect($commonElements, $arrays[$i]);
    }

    return $commonElements;
}
```

Problem 137: Implement a Circular Buffer

Problem: Implement a circular buffer with a fixed size that overwrites the oldest data when it reaches its limit.

Solution:

```
class CircularBuffer {
    private $size;
    private $buffer;
    private $writeIndex;

    function __construct($size) {
        $this->size = $size;
        $this->buffer = array_fill(0, $size, null);
        $this->writeIndex = 0;
    }

    function write($data) {
        $this->buffer[$this->writeIndex] = $data;
        $this->writeIndex = ($this->writeIndex + 1) % $this->size;
    }

    function read() {
        return $this->buffer;
    }
}
```

Problem 138: Find the Median of a Stream of Integers

Problem: Design a data structure that supports adding integers one at a time and finding the median of the current elements.

Solution:

```
class MedianFinder {
    private $maxHeap; // Left half of elements (lower values)
    private $minHeap; // Right half of elements (higher values)

    function __construct() {
        $this->maxHeap = new SplMaxHeap();
        $this->minHeap = new SplMinHeap();
    }

    function addNum($num) {
        $this->maxHeap->insert($num);
        $this->minHeap->insert($this->maxHeap->extract());

        if ($this->maxHeap->count() < $this->minHeap->count()) {
            $this->maxHeap->insert($this->minHeap->extract());
        }
    }

    function findMedian() {
        if ($this->maxHeap->count() > $this->minHeap->count()) {
            return $this->maxHeap->top();
        }

        return ($this->maxHeap->top() + $this->minHeap->top()) / 2.0;
    }
}
```

Problem 139: Implement a Priority Queue

Problem: Implement a priority queue data structure with methods for insertion and extraction of elements based on their priority.

Solution:

```
class PriorityQueue {
    private $heap;

    function __construct() {
        $this->heap = new SplMinHeap();
    }

    function insert($element, $priority) {
        $this->heap->insert([$priority, $element]);
    }

    function extractMin() {
        if ($this->heap->isEmpty()) {
            return null;
        }

        list($priority, $element) = $this->heap->extract();
        return $element;
    }

    function isEmpty() {
        return $this->heap->isEmpty();
    }
}
```

Problem 140: Implement a Two-Sum Problem with Duplicates

Problem: Implement a function that, given an array of integers and a target sum, returns all pairs of integers in the array that add up to the target sum, including duplicates.

Solution:

```
function findTwoSumPairsWithDuplicates($nums, $target) {
    $pairs = [];
    $numMap = [];

    foreach ($nums as $num) {
        $complement = $target - $num;
        if (isset($numMap[$complement])) {
            $pairs[] = [$num, $complement];
        }
        $numMap[$num]++;
    }

    return $pairs;
}
```

Problem 141: Implement a Sparse Matrix

Problem: Implement a data structure to represent a sparse matrix efficiently.

Solution:

```
class SparseMatrix {
    private $matrix;
    private $rows;
    private $cols;

    function __construct($rows, $cols) {
        $this->rows = $rows;
        $this->cols = $cols;
        $this->matrix = [];
    }

    function set($row, $col, $value) {
        if ($row < 0 || $row >= $this->rows || $col < 0 || $col >= $this->cols) {
            throw new Exception("Invalid row or column index.");
        }

        if ($value !== 0) {
            $this->matrix["$row:$col"] = $value;
        } elseif (isset($this->matrix["$row:$col"])) {
            unset($this->matrix["$row:$col"]);
        }
    }

    function get($row, $col) {
        if ($row < 0 || $row >= $this->rows || $col < 0 || $col >= $this->cols) {
            throw new Exception("Invalid row or column index.");
        }

        return isset($this->matrix["$row:$col"]) ? $this->matrix["$row:$col"] : 0;
    }
}
```

Problem 142: Implement an Interval Merge

Problem: Given a collection of intervals, merge any overlapping intervals.

Solution:

```
function mergeIntervals($intervals) {
    if (empty($intervals)) {
        return [];
    }

    usort($intervals, function($a, $b) {
        return $a[0] - $b[0];
    });

    $merged = [$intervals[0]];

    for ($i = 1; $i < count($intervals); $i++) {
        $current = $intervals[$i];
        $lastMerged = $merged[count($merged) - 1];

        if ($current[0] <= $lastMerged[1]) {
            $lastMerged[1] = max($lastMerged[1], $current[1]);
        } else {
            $merged[] = $current;
        }
    }

    return $merged;
}
```

Problem 143: Implement a Spreadsheet

Problem: Implement a spreadsheet data structure with cells that can contain values or formulas. Formulas can reference other cells.

Solution:

```
class Spreadsheet {
    private $cells;

    function __construct() {
        $this->cells = [];
    }

    function setValue($cell, $value) {
        $this->cells[$cell] = $value;
    }

    function setFormula($cell, $formula) {
        $this->cells[$cell] = $formula;
    }

    function getValue($cell) {
        if (isset($this->cells[$cell])) {
            $value = $this->cells[$cell];
            if (is_numeric($value)) {
                return $value;
            }

 else {
                return $this->evaluateFormula($value);
            }
        }
        return 0; // Default value for empty cell
    }

    function evaluateFormula($formula) {
        // Implement formula evaluation logic here
    }
}
```

Problem 144: Implement a Data Stream Processor

Problem: Implement a data stream processor that continuously processes incoming data points and computes statistics (e.g., moving average, min, max) based on a specified window size.

Solution:

```
class DataStreamProcessor {
    private $data;
    private $windowSize;
    private $windowData;

    function __construct($windowSize) {
        $this->data = [];
        $this->windowSize = $windowSize;
        $this->windowData = [];
    }

    function addDataPoint($value) {
        $this->data[] = $value;

        while (count($this->data) > $this->windowSize) {
            array_shift($this->data);
        }

        $this->updateWindowData();
    }

    function getMovingAverage() {
        return array_sum($this->windowData) / count($this->windowData);
    }

    function getMin() {
        return min($this->windowData);
    }

    function getMax() {
        return max($this->windowData);
    }

    private function updateWindowData() {
        $this->windowData = array_slice($this->data, -1 * $this->windowSize);
    }
}
```

Problem 145: Implement a Task Scheduler

Problem: Implement a task scheduler that schedules tasks with cooldown periods, ensuring that the same task is not executed within the cooldown period.

Solution:

```
function leastInterval($tasks, $n) {
    $taskCount = array_count_values($tasks);
    $maxCount = max($taskCount);

    $idleTime = ($maxCount - 1) * ($n + 1);

    foreach ($taskCount as $count) {
        $idleTime -= min($count, $maxCount - 1);
    }

    $idleTime = max(0, $idleTime);

    return count($tasks) + $idleTime;
}
```

Problem 146: Implement a Circular Linked List

Problem: Implement a circular linked list data structure and its operations (insertion, deletion, traversal) efficiently.

Solution:

```
class Node {
    public $data;
    public $next;

    function __construct($data) {
        $this->data = $data;
        $this->next = null;
    }
}

class CircularLinkedList {
    private $head;

    function __construct() {
        $this->head = null;
    }

    function insert($data) {
        $newNode = new Node($data);

        if ($this->head === null) {
            $newNode->next = $newNode; // Point to itself
            $this->head = $newNode;
        } else {
            $temp = $this->head;
            while ($temp->next !== $this->head) {
                $temp = $temp->next;
            }
            $temp->next = $newNode;
            $newNode->next = $this->head;
        }
    }

    function delete($data) {
        if ($this->head === null) {
            return;
        }

        if ($this->head->data === $data) {
            $temp = $this->head;
            while ($temp->next !== $this->head) {
                $temp = $temp->next;
            }
            if ($temp === $this->head) {
                $this->head = null;
            } else {
                $temp->next = $this->head->next;
                $this->head = $this->head->next;
            }
        } else {
            $prev = null;
            $curr = $this->head;

            do {
                if ($curr->data === $data) {
                    $prev->next = $curr->next;
                    break;
                }
                $prev = $curr;
                $curr = $curr->next;
            } while ($curr !== $this->head);
            } while ($curr !== $this->head);
        }
    }

    function display() {
        if ($this->head === null) {
            return;
        }

        $temp = $this->head;
        do {
            echo $temp->data . " ";
            $temp = $temp->next;
        } while ($temp !== $this->head);
    }
}
```

Problem 147: Implement an Undo/Redo Stack

Problem: Implement an undo/redo stack for a text editor, allowing users to undo and redo text changes.

Solution:

```
class TextEditor {
    private $text;
    private $undoStack;
    private $redoStack;

    function __construct() {
        $this->text = "";
        $this->undoStack = [];
        $this->redoStack = [];
    }

    function insert($text) {
        $this->undoStack[] = $this->text;
        $this->text .= $text;
        $this->redoStack = []; // Clear redo stack
    }

    function delete($numChars) {
        if ($numChars >= strlen($this->text)) {
            $this->undoStack[] = $this->text;
            $this->text = "";
        } else {
            $this->undoStack[] = $this->text;
            $this->text = substr($this->text, 0, -$numChars);
        }
        $this->redoStack = []; // Clear redo stack
    }

    function undo() {
        if (!empty($this->undoStack)) {
            $this->redoStack[] = $this->text;
            $this->text = array_pop($this->undoStack);
        }
    }

    function redo() {
        if (!empty($this->redoStack)) {
            $this->undoStack[] = $this->text;
            $this->text = array_pop($this->redoStack);
        }
    }

    function getText() {
        return $this->text;
    }
}
```

Problem 148: Implement a URL Shortener

Problem: Implement a URL shortening service that converts long URLs into short, shareable links and resolves them back to the original URLs.

Solution:

```
class URLShortener {
    private $urlMap;
    private $shortUrlPrefix;
    private $counter;

    function __construct($shortUrlPrefix = "http://short.ly/") {
        $this->urlMap = [];
        $this->shortUrlPrefix = $shortUrlPrefix;
        $this->counter = 0;
    }

    function shortenURL($longURL) {
        $shortURL = $this->shortUrlPrefix . $this->counter;
        $this->urlMap[$shortURL] = $longURL;
        $this->counter++;
        return $shortURL;
    }

    function resolveShortURL($shortURL) {
        if (isset($this->urlMap[$shortURL])) {
            return $this->urlMap[$shortURL];
        }


        return null;
    }
}
```

Problem 149: Implement a Circular Queue

Problem: Implement a circular queue data structure with methods for enqueuing and dequeuing elements.

Solution:

```php
class CircularQueue {
    private $capacity;
    private $queue;
    private $front;
    private $rear;

    function __construct($capacity) {
        $this->capacity = $capacity;
        $this->queue = array_fill(0, $capacity, null);
        $this->front = $this->rear = -1;
    }

    function enqueue($data) {
        if (($this->rear + 1) % $this->capacity === $this->front) {
            return false; // Queue is full
        }

        if ($this->isEmpty()) {
            $this->front = $this->rear = 0;
        } else {
            $this->rear = ($this->rear + 1) % $this->capacity;
        }

        $this->queue[$this->rear] = $data;
        return true;
    }

    function dequeue() {
        if ($this->isEmpty()) {
            return null; // Queue is empty
        }

        $data = $this->queue[$this->front];

        if ($this->front === $this->rear) {
            $this->front = $this->rear = -1;
        } else {
            $this->front = ($this->front + 1) % $this->capacity;
        }

        return $data;
    }

    function isEmpty() {
        return $this->front === -1 && $this->rear === -1;
    }

    function isFull() {
        return ($this->rear + 1) % $this->capacity === $this->front;
    }
}
```

Problem 150: Implement a Music Playlist

Problem: Implement a music playlist data structure that allows users to add, remove, shuffle, and play songs.

Solution:

```
class MusicPlaylist {
    private $songs;
    private $currentSongIndex;
    private $shuffleMode;

    function __construct() {
        $this->songs = [];
        $this->currentSongIndex = 0;
        $this->shuffleMode = false;
    }

    function addSong($song) {
        $this->songs[] = $song;
    }

    function removeSong($song) {
        $index = array_search($song, $this->songs);
        if ($index !== false) {
            array_splice($this->songs, $index, 1);
            if ($index < $this->currentSongIndex) {
                $this->currentSongIndex--;
            }
        }
    }

    function shuffle() {
        shuffle($this->songs);
        $this->currentSongIndex = 0;
        $this->shuffleMode = true;
    }

    function play() {
        if ($this->shuffleMode) {
            shuffle($this->songs); // Shuffle again for variety
        }
        if ($this->currentSongIndex < count($this->songs)) {
            return $this->songs[$this->currentSongIndex++];
        } else {
            return null; // End of playlist
        }
    }
}
```

SuperGlobals & REGEX

Certainly! Here are 20 medium and hard-level problems related to superglobals and regular expressions in PHP, along with their solutions and code examples:

1. Problem: Create a PHP script that counts the number of times a specific word appears in a given text using regular expressions.

Solution:

```
$text = "This is a test text. Test text is good.";
$word = "test";
$pattern = "/\b" . preg_quote($word, '/') . "\b/i";
preg_match_all($pattern, $text, $matches);
$count = count($matches[0]);
echo "The word '$word' appears $count times in the text.";
```

2. Problem: Write a PHP program to validate an email address using regular expressions.

Solution:

```
$email = "example@email.com";
$pattern = "/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/";
if (preg_match($pattern, $email)) {
    echo "Valid email address.";
} else {
    echo "Invalid email address.";
}
```

3. Problem: Create a PHP script that extracts all URLs from a given HTML document using regular expressions.

Solution:

```
$html = "<a href='http://example.com'>Link 1</a><a href='https://example2.com'>Link 2</a>";
$pattern = "/href=['\"](https?:\/\/.*?)['\"]/i";
preg_match_all($pattern, $html, $matches);
print_r($matches[1]);
```

4. Problem: Write a PHP function that validates a URL, ensuring it starts with "http://" or "https://".

Solution:

```
function validateURL($url) {
    $pattern = "/^(https?:\/\/).+$/";
    return preg_match($pattern, $url) === 1;
}
```

5. Problem: Create a PHP program that validates a date in the format "YYYY-MM-DD" using regular expressions.

Solution:

```
$date = "2023-09-20";
$pattern = "/^\d{4}-\d{2}-\d{2}$/";
if (preg_match($pattern, $date)) {
    echo "Valid date format.";
} else {
    echo "Invalid date format.";
}
```

6. Problem: Write a PHP script that extracts all hashtags (#) from a given text using regular expressions.

Solution:

```
$text = "This is a #sample text with #hashtags.";
$pattern = "/#(\w+)/";
preg_match_all($pattern, $text, $matches);
print_r($matches[1]);
```

7. Problem: Create a PHP function that replaces all occurrences of a specific word with asterisks (*) in a given text.

Solution:

```
function censorWord($text, $word) {
    $pattern = "/\b" . preg_quote($word, '/') . "\b/i";
    return preg_replace($pattern, '***', $text);
}
```

8. Problem: Write a PHP program that extracts all phone numbers in a specific format (e.g., (123) 456-7890) from a given text using regular expressions.

Solution:

```
$text = "Call (123) 456-7890 or (456) 789-1234 for assistance.";
$pattern = "/\(\d{3}\) \d{3}-\d{4}/";
preg_match_all($pattern, $text, $matches);
print_r($matches[0]);
```

9. Problem: Create a PHP script that validates a strong password, including at least one uppercase letter, one lowercase letter, one digit, and one special character.

Solution:

```
function validatePassword($password) {
    $pattern = "/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@#$%^&+=!]).{8,}$/";
    return preg_match($pattern, $password) === 1;
}
```

10. Problem: Write a PHP program that extracts all image file names (e.g., .jpg, .png) from a given directory listing using regular expressions.

Solution:

```
$directoryListing = "image1.jpg, image2.png, script.php, image3.gif";
$pattern = "/\b\w+\.(jpg|png|gif)\b/";
preg_match_all($pattern, $directoryListing, $matches);
print_r($matches[0]);
```

11. Problem: Create a PHP function that counts the number of words in a given text using regular expressions.

Solution:

```
function countWords($text) {
    $pattern = "/\b\w+\b/";
    preg_match_all($pattern, $text, $matches);
    return count($matches[0]);
}
```

12. Problem: Write a PHP script that extracts all valid IP addresses from a given text using regular expressions.

Solution:

```
$text = "IPs: 192.168.1.1, 10.0.0.1, 256.256.256.256, 172.16.0.1";
$pattern = "/\b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\."
          . "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\."
          . "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\."
          . "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b/";
preg_match_all($pattern, $text, $matches);
print_r($matches[0]);
```

13. Problem: Create a PHP program that validates a hexadecimal color code (e.g., #FFA500) using regular expressions.

Solution:

```
$colorCode = "#FFA500";
$pattern = "/^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$/";
if (preg_match($pattern, $colorCode)) {
    echo "Valid color code.";
} else {
    echo "Invalid color code.";
}
```

14. Problem: Write a PHP script that extracts all YouTube video IDs from a given text containing YouTube URLs using regular expressions.

Solution:

```
$text = "Watch this: https://www.youtube.com/watch?v=abc123&feature=xyz";
$pattern = "/watch\?v=([a-zA-Z0-9_-]+)/";
preg_match($pattern, $text, $matches);
echo "Video ID: " . $matches[1];
```

15. Problem: Create a PHP function that removes all HTML tags from a given string using regular expressions.

Solution:

```
function stripHTMLTags($html) {
    return preg_replace("/<.*?>/", "", $html);
}
```

16. Problem: Write a PHP program that validates a credit card number

using regular expressions to match common credit card patterns.

Solution:

```
function validateCreditCard($cardNumber) {
    $pattern = "/^(3[47]\d{13}|4\d{12}(\d{3})?|5[1-5]\d{14}|6011\d{12}|(30[0-5]|36\d{2}|3[89]\d\d)\d{11})$/";
    return preg_match($pattern, $cardNumber) === 1;
}
```

17. Problem: Create a PHP script that extracts all Twitter usernames (e.g., @user) from a given text using regular expressions.

Solution:

```
$text = "Follow @user1 and @user2 for updates.";
$pattern = "/@(\w+)/";
preg_match_all($pattern, $text, $matches);
print_r($matches[1]);
```

18. Problem: Write a PHP program that validates a strong password, including at least 12 characters and a mix of letters, digits, and special characters.

Solution:

```
function validateStrongPassword($password) {
    $pattern = "/^(?=.*[a-zA-Z])(?=.*\d)(?=.*[@#$%^&+=!]).{12,}$/";
    return preg_match($pattern, $password) === 1;
}
```

19. Problem: Create a PHP function that removes all non-alphanumeric characters from a given string using regular expressions.

Solution:

```
function removeNonAlphanumeric($text) {
    return preg_replace("/[^a-zA-Z0-9]/", "", $text);
}
```

20. Problem: Write a PHP script that extracts all mentions (@username) and hashtags (#tag) from a given text using regular expressions.

Solution:

```
$text = "Check out @user1's #awesome post! #coding #php";
$pattern = "/@(\w+)|#(\w+)/";
preg_match_all($pattern, $text, $matches);
$mentions = $matches[1];
$hashtags = $matches[2];
print_r($mentions);
print_r($hashtags);
```

Certainly! Let's explore complex types of superglobals in PHP (e.g., `$_POST`, `$_GET`, `$_FILES`, `$_SESSION`, `$_COOKIE`, and `$_SERVER`) with additional problems and solutions.

Superglobal: `$_POST`

Problem: Create a PHP script that processes a form submission with multiple checkboxes and stores the selected values in an array.

Solution:

```
<form method="post">
    <input type="checkbox" name="fruits[]" value="apple"> Apple
    <input type="checkbox" name="fruits[]" value="banana"> Banana
    <input type="checkbox" name="fruits[]" value="cherry"> Cherry
    <input type="submit" name="submit" value="Submit">
</form>

<?php
if (isset($_POST['submit'])) {
    $selectedFruits = $_POST['fruits'];
    print_r($selectedFruits);
}
?>
```

Superglobal: `$_GET`

Problem: Write a PHP script that displays user-specific data based on a URL parameter.

Solution:

```
// URL: example.com/profile.php?username=johndoe

$username = $_GET['username'];
echo "Welcome, $username!";
```

Superglobal: `$_FILES`

Problem: Create a PHP script that allows users to upload an image file and then displays it on the page.

Solution:

```
<form method="post" enctype="multipart/form-data">
    <input type="file" name="image">
    <input type="submit" name="submit" value="Upload">
</form>

<?php
if (isset($_POST['submit'])) {
    $targetDir = "uploads/";
    $targetFile = $targetDir . basename($_FILES["image"]["name"]);
    move_uploaded_file($_FILES["image"]["tmp_name"], $targetFile);
    echo "Image uploaded successfully!";
    echo "<img src='$targetFile' alt='Uploaded Image'>";
}
?>
```

Superglobal: `$_SESSION`

Problem: Write a PHP script that counts the number of times a user visits a page and displays it.

Solution:

```
session_start();

if (isset($_SESSION['visits'])) {
    $_SESSION['visits']++;
} else {
    $_SESSION['visits'] = 1;
}

echo "Number of visits: " . $_SESSION['visits'];
```

Superglobal: `$_COOKIE`

Problem: Create a PHP script that sets a cookie with a user's preferred language and displays a greeting message in that language.

Solution:

```
if (isset($_COOKIE['preferred_language'])) {
    $language = $_COOKIE['preferred_language'];
} else {
    $language = 'English'; // Default language
    setcookie('preferred_language', $language, time() + 3600 * 24 * 30); // Set cookie for 30 days
}

$greetings = [
    'English' => 'Hello!',
    'Spanish' => 'Â¡Hola!',
    'French' => 'Bonjour!',
    // Add more languages here
];

echo $greetings[$language];
```

Superglobal: `$_SERVER`

Problem: Write a PHP script that displays the user's IP address and user agent.

Solution:

```
$ipAddress = $_SERVER['REMOTE_ADDR'];
$userAgent = $_SERVER['HTTP_USER_AGENT'];

echo "Your IP address is: $ipAddress<br>";
echo "Your user agent is: $userAgent";
```

FORMS : Handling, validation, required, URLs/Email

While these problems aren't directly tied to competitive programming platforms like Codeforces or LeetCode, they cover common real-world scenarios.

1. Problem: Basic HTML Form

- You need to create a simple HTML form that takes a user's name and email.

```
<form method="post" action="process.php">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>
    <input type="submit" value="Submit">
</form>
```

Solution: Create a PHP script (`process.php`) to handle the form submission.

```
<?php
$name = $_POST['name'];
$email = $_POST['email'];
echo "Name: $name<br>Email: $email";
?>
```

2. Problem: Validating Email Address

- Implement PHP code to validate if an entered email address is valid.

Solution: Use `filter_var` to validate the email.

```
$email = $_POST['email'];
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "Valid email address";
} else {
    echo "Invalid email address";
}
```

3. Problem: Handling Checkbox Inputs

- Create a form with multiple checkboxes and process the selected options in PHP.

Solution: Check which checkboxes are selected.

```
$selectedOptions = $_POST['options'];
foreach ($selectedOptions as $option) {
    echo "Selected: $option<br>";
}
```

4. Problem: Handling Radio Buttons

- Build a form with radio buttons and process the selected option.

Solution: Determine the selected radio button.

```
$selectedOption = $_POST['option'];
echo "Selected: $selectedOption";
```

5. Problem: Uploading Files

- Create a form to upload a file and handle file uploads in PHP.

Solution: Use `$_FILES` to handle file uploads.

```
$file = $_FILES['file'];
move_uploaded_file($file['tmp_name'], 'uploads/' . $file['name']);
```

6. Problem: Handling URLs

- Validate and process a URL entered in a form.

Solution: Use `filter_var` to validate the URL.

```
$url = $_POST['url'];
if (filter_var($url, FILTER_VALIDATE_URL)) {
    echo "Valid URL";
} else {
    echo "Invalid URL";
}
```

7. Problem: Handling Dropdown Lists

- Create a form with a dropdown list and process the selected option.

Solution: Retrieve the selected option.

```
$selectedOption = $_POST['dropdown'];
echo "Selected: $selectedOption";
```

8. Problem: Required Fields Validation

- Implement server-side validation for required form fields.

Solution: Check if required fields are empty.

```
$name = $_POST['name'];
$email = $_POST['email'];

if (empty($name) || empty($email)) {
    echo "Name and email are required fields.";
} else {
    // Process the form.
}
```

9. Problem: Redirecting After Form Submission

- Redirect the user to a thank you page after successful form submission.

Solution: Use the `header` function for redirection.

```
header("Location: thank-you.php");
exit();
```

10. Problem: Preventing Cross-Site Scripting (XSS) - Sanitize user input to prevent XSS attacks when displaying input data.

Solution: Use `htmlspecialchars` to escape user input.

```
$name = htmlspecialchars($_POST['name']);
```

Certainly! Here are the next 10 problems related to PHP forms, along with solutions and code examples:

11. Problem: Handling Textarea Input - Create a form with a textarea input and process the user's input.

Solution: Retrieve and display the textarea content.

```
$message = $_POST['message'];
echo "Message: $message";
```

12. Problem: Handling Multiple Select Inputs - Build a form with a multi-select input (selecting multiple options) and process the selected options.

Solution: Retrieve and display the selected options.

```
$selectedOptions = $_POST['options'];
foreach ($selectedOptions as $option) {
    echo "Selected: $option<br>";
}
```

13. Problem: Handling Date Input - Create a form with a date input and process the user's selected date.

Solution: Retrieve and display the selected date.

```
$selectedDate = $_POST['date'];
echo "Selected Date: $selectedDate";
```

14. Problem: Handling Password Input - Implement password validation rules (e.g., length, complexity) for a password input field.

Solution: Check if the password meets the criteria.

```
$password = $_POST['password'];
if (strlen($password) >= 8 && preg_match('/[A-Za-z0-9]/', $password)) {
    echo "Valid password";
} else {
    echo "Invalid password";
}
```

15. Problem: Handling CAPTCHA - Integrate CAPTCHA verification to prevent spam submissions.

Solution: Use a CAPTCHA service like reCAPTCHA and verify the user's response.

```
$recaptchaSecretKey = 'YOUR_SECRET_KEY';
$recaptchaResponse = $_POST['g-recaptcha-response'];
$response = file_get_contents("https://www.google.com/recaptcha/api/siteverify?secret=$recaptchaSecretKey&response=$recaptchaRespons
$responseKeys = json_decode($response, true);

if (intval($responseKeys["success"]) !== 1) {
    echo "CAPTCHA verification failed";
} else {
    // Process the form.
}
```

16. Problem: Handling Dynamic Form Fields - Allow users to add dynamic form fields (e.g., adding multiple email addresses) and process them.

Solution: Use JavaScript to add dynamic fields and process them in PHP.

JavaScript (Add fields):

```
function addEmailField() {
    const container = document.getElementById('email-container');
    const input = document.createElement('input');
    input.type = 'email';
    input.name = 'emails[]';
    container.appendChild(input);
}
```

PHP (Process fields):

```
$emails = $_POST['emails'];
foreach ($emails as $email) {
    echo "Email: $email<br>";
}
```

17. Problem: Form Data Validation Using Regular Expressions - Validate form data using regular expressions (e.g., phone number format).

Solution: Use regular expressions for validation.

```
$phone = $_POST['phone'];
if (preg_match('/^\d{10}$/', $phone)) {
    echo "Valid phone number";
} else {
    echo "Invalid phone number";
}
```

18. Problem: Displaying Error Messages - Display user-friendly error messages for form validation failures.

Solution: Store error messages in an array and display them.

```
$errors = [];

if (empty($name)) {
    $errors[] = "Name is required.";
}

if (empty($email)) {
    $errors[] = "Email is required.";
}

// Display errors if any.
if (!empty($errors)) {
    foreach ($errors as $error) {
        echo "$error<br>";
    }
}
```

19. Problem: Handling Internationalization (i18n) - Allow users to select their preferred language on a form.

Solution: Store and use language preferences in sessions or cookies.

```
$selectedLanguage = $_POST['language'];
setcookie('language', $selectedLanguage, time() + (3600 * 24 * 30), '/');
```

20. Problem: Preventing Duplicate Form Submissions - Implement measures to prevent accidental duplicate form submissions.

Solution: Use the Post-Redirect-Get (PRG) pattern and generate unique tokens.

```
session_start();

if ($_SERVER['REQUEST_METHOD'] === 'POST' && !isset($_SESSION['submitted'])) {
    // Process the form.
    $_SESSION['submitted'] = true;
    // Redirect to a thank you page.
    header("Location: thank-you.php");
    exit();
}
```

FILTER FOR VALIDATION

Problem 1: How to validate an email address using PHP filters?

Solution:

```
$email = "user@example.com";
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "Valid email address.";
} else {
    echo "Invalid email address.";
}
```

Problem 2: How to validate an IP address using PHP filters?

Solution:

```
$ip = "192.168.1.1";
if (filter_var($ip, FILTER_VALIDATE_IP)) {
    echo "Valid IP address.";
} else {
    echo "Invalid IP address.";
}
```

Problem 3: How to sanitize and filter input data to prevent XSS attacks?

Solution:

```
$userInput = "<script>alert('XSS attack');</script>";
$filteredInput = filter_var($userInput, FILTER_SANITIZE_STRING);
echo $filteredInput;
```

Problem 4: How to validate and sanitize an integer input using PHP filters?

Solution:

```
$userInput = "123";
$filteredInput = filter_var($userInput, FILTER_VALIDATE_INT);

if ($filteredInput !== false) {
    echo "Valid integer: " . $filteredInput;
} else {
    echo "Invalid integer.";
}
```

Problem 5: How to validate and sanitize URL input using PHP filters?

Solution:

```
$url = "https://www.example.com";
$filteredUrl = filter_var($url, FILTER_VALIDATE_URL);

if ($filteredUrl !== false) {
    echo "Valid URL: " . $filteredUrl;
} else {
    echo "Invalid URL.";
}
```

Problem 6: How to filter an array of values using a custom filter function in PHP?

Solution:

```
function customFilter($value) {
    return $value > 5;
}

$array = [3, 6, 9, 2, 7];
$filteredArray = array_filter($array, "customFilter");

print_r($filteredArray);
```

Problem 7: How to validate and filter an array of email addresses in PHP?

Solution:

```
$emailArray = ["user@example.com", "invalid-email", "another@example.com"];
$filteredEmails = array_filter($emailArray, function ($email) {
    return filter_var($email, FILTER_VALIDATE_EMAIL);
});

print_r($filteredEmails);
```

Problem 8: How to use a custom filter to validate and filter a list of usernames?

Solution:

```
function validateUsername($username) {
    // Custom validation logic (e.g., alphanumeric, length constraints)
    return preg_match('/^[a-zA-Z0-9_]+$/', $username) && strlen($username) >= 5;
}

$usernames = ["user123", "invalid*", "john_doe"];
$filteredUsernames = array_filter($usernames, "validateUsername");

print_r($filteredUsernames);
```

Problem 9: How to validate and filter an array of dates using PHP filters?

Solution:

```
$dates = ["2022-01-15", "invalid-date", "2023-12-25"];
$filteredDates = array_filter($dates, function ($date) {
    return (bool) strtotime($date);
});

print_r($filteredDates);
```

Problem 10: How to validate and filter an array of values with a custom callback function and return the filtered array in reverse order?

Solution:

```
$values = [10, 15, 20, 5, 30];
$filteredValues = array_filter($values, function ($value) {
    return $value % 2 == 0;
});

$reversedFilteredValues = array_reverse($filteredValues);

print_r($reversedFilteredValues);
```

PHP ADVACED

This Series is followed By W3School Tutorials Side. ## DATE & TIME

1. Problem: Displaying Current Date and Time

- Write PHP code to display the current date and time.

Solution: Use the `date` function to display the current date and time.

```
$currentDateTime = date("Y-m-d H:i:s");
echo "Current Date and Time: $currentDateTime";
```

2. Problem: Formatting Dates

- Format a given date into a user-friendly format, e.g., "January 15, 2023."

Solution: Use the `date` function with a custom format.

```
$inputDate = "2023-01-15";
$formattedDate = date("F j, Y", strtotime($inputDate));
echo "Formatted Date: $formattedDate";
```

3. Problem: Calculating Age

- Calculate a person's age from their birthdate.

Solution: Use the `date_diff` function.

```
$birthdate = "1990-05-20";
$today = date("Y-m-d");
$age = date_diff(date_create($birthdate), date_create($today))->y;
echo "Age: $age years";
```

4. Problem: Adding Days to a Date

- Add a specific number of days to a given date.

Solution: Use the `date` and `strtotime` functions.

```
$inputDate = "2023-03-10";
$daysToAdd = 30;
$newDate = date("Y-m-d", strtotime($inputDate . " +$daysToAdd days"));
echo "New Date: $newDate";
```

5. Problem: Working with Time Zones

- Convert a date and time from one time zone to another.

Solution: Use the `DateTime` class for time zone conversion.

```
$dateTime = new DateTime("2023-06-15 15:00:00", new DateTimeZone("UTC"));
$dateTime->setTimezone(new DateTimeZone("America/New_York"));
echo "Converted Date and Time: " . $dateTime->format("Y-m-d H:i:s");
```

6. Problem: Find the Next Weekday

- Find the next occurrence of a specific weekday (e.g., the next Monday).

Solution: Use a loop to find the next occurrence.

```
$targetDay = 1; // 1 represents Monday
$today = date("N"); // Get the current day of the week (1 = Monday, 7 = Sunday)

$daysToAdd = ($targetDay - $today + 7) % 7;
$nextMonday = date("Y-m-d", strtotime("+$daysToAdd days"));
echo "Next Monday: $nextMonday";
```

7. Problem: Time Difference Between Two Dates

- Calculate the time difference (duration) between two dates and times.

Solution: Use the `DateTime` class to calculate the difference.

```
$startDateTime = new DateTime("2023-07-01 10:00:00");
$endDateTime = new DateTime("2023-07-05 15:30:00");
$interval = $startDateTime->diff($endDateTime);
echo "Duration: " . $interval->format("%d days %h hours %i minutes");
```

8. Problem: Date Range Generator

- Generate a list of dates within a given date range.

Solution: Use a loop to iterate through dates.

```
$startDate = "2023-09-01";
$endDate = "2023-09-10";

$currentDate = $startDate;
while (strtotime($currentDate) <= strtotime($endDate)) {
    echo "$currentDate\n";
    $currentDate = date("Y-m-d", strtotime($currentDate . " +1 day"));
}
```

9. Problem: Daylight Saving Time (DST) Check

- Determine if a specific date and time is during DST.

Solution: Use the `date` and `strtotime` functions.

```
$dateToCheck = "2023-03-15 14:00:00"; // March 15th is during DST in many regions.
$isDST = date("I", strtotime($dateToCheck));

if ($isDST) {
    echo "DST is in effect.";
} else {
    echo "DST is not in effect.";
}
```

10. Problem: Finding the Last Day of the Month - Find the last day of the current month.

Solution: Use the `date` and `strtotime` functions.

```
$lastDayOfMonth = date("Y-m-t");
echo "Last Day of the Month: $lastDayOfMonth";
```

Certainly! Here are the next 10 problems related to date and time in PHP, along with solutions and code examples:

11. Problem: Counting Days Until a Future Date

- Calculate the number of days remaining until a future date.

Solution: Use the `date_diff` function to calculate the difference.

```
$futureDate = new DateTime("2024-12-31");
$today = new DateTime();
$interval = $today->diff($futureDate);
echo "Days until 2024-12-31: " . $interval->format("%a days");
```

12. Problem: Displaying Relative Time (e.g., "2 hours ago")

- Display a user-friendly relative time based on the current time.

Solution: Calculate the time difference and format it.

```
$timestamp = strtotime("2023-09-19 15:30:00");
$currentTimestamp = time();
$timeDifference = $currentTimestamp - $timestamp;

if ($timeDifference < 60) {
    echo "Just now";
} elseif ($timeDifference < 3600) {
    $minutes = floor($timeDifference / 60);
    echo "$minutes minute(s) ago";
} elseif ($timeDifference < 86400) {
    $hours = floor($timeDifference / 3600);
    echo "$hours hour(s) ago";
} else {
    $days = floor($timeDifference / 86400);
    echo "$days day(s) ago";
}
```

13. Problem: Finding Leap Years

- Determine if a given year is a leap year.

Solution: Use the `date` and `strtotime` functions.

```
$yearToCheck = 2024;
if (date("L", strtotime("$yearToCheck-01-01"))) {
    echo "$yearToCheck is a leap year.";
} else {
    echo "$yearToCheck is not a leap year.";
}
```

14. Problem: Generating Timestamps

- Generate a timestamp for a specific date and time.

Solution: Use the `mktime` function.

```
$timestamp = mktime(14, 30, 0, 9, 30, 2023);
echo "Timestamp for 2:30 PM on September 30, 2023: $timestamp";
```

15. Problem: Day of the Week Calculation

- Find the day of the week for a given date (e.g., Monday, Tuesday).

Solution: Use the `date` function with 'l' format.

```
$inputDate = "2023-09-20";
$dayOfWeek = date("l", strtotime($inputDate));
echo "Day of the Week: $dayOfWeek";
```

16. Problem: Age in Months

- Calculate a person's age in months from their birthdate.

Solution: Use the `date_diff` function to calculate months.

```
$birthdate = "1990-05-20";
$today = new DateTime();
$interval = date_diff(date_create($birthdate), $today);
$ageInMonths = $interval->format("%m months");
echo "Age: $ageInMonths";
```

17. Problem: Date Parsing and Formatting from User Input

- Parse and format a date entered by the user into a consistent format.

Solution: Use the `DateTime` class to handle user input.

```
$userInputDate = "09/20/23";
$parsedDate = DateTime::createFromFormat("m/d/y", $userInputDate);
$formattedDate = $parsedDate->format("Y-m-d");
echo "Formatted Date: $formattedDate";
```

18. Problem: Date Range Overlaps

- Check if two date ranges overlap.

Solution: Compare the start and end dates of the two ranges.

```
$range1Start = "2023-09-10";
$range1End = "2023-09-20";
$range2Start = "2023-09-15";
$range2End = "2023-09-25";

if ($range1End >= $range2Start && $range2End >= $range1Start) {
    echo "Date ranges overlap.";
} else {
    echo "Date ranges do not overlap.";
}
```

19. Problem: Finding the Quarter of a Year

- Determine the quarter of the year for a given date.

Solution: Use the `date` function with 'n' format.

```
$inputDate = "2023-07-15";
$quarter = ceil(date("n", strtotime($inputDate)) / 3);
echo "Quarter: Q$quarter";
```

20. Problem: Counting Weekdays in a Date Range - Calculate the number of weekdays (Monday to Friday) in a given date range.

Solution: Use a loop to iterate through the dates and count weekdays.

```
$startDate = "2023-09-01";
$endDate = "2023-09-30";
$countWeekdays = 0;

$currentDate = $startDate;
while (strtotime($currentDate) <= strtotime($endDate)) {
    $dayOfWeek = date("N", strtotime($currentDate));
    if ($dayOfWeek >= 1 && $dayOfWeek <= 5) {
        $countWeekdays++;
    }
    $currentDate = date("Y-m-d", strtotime($currentDate . " +1 day"));
}

echo "Weekdays in September 2023: $countWeekdays";
```

Certainly! Here are 10 additional problems related to date and time in PHP, without repeating any previous problems:

21. Problem: Timezone Conversion for a List of Timestamps

- Convert a list of timestamps from one timezone to another and display the results.

Solution: Use the `DateTime` class to handle timezone conversion.

```
$timestamps = [1632436800, 1632536800, 1632636800]; // Timestamps in UTC
$fromTimeZone = new DateTimeZone("UTC");
$toTimeZone = new DateTimeZone("America/New_York");

foreach ($timestamps as $timestamp) {
    $dateTime = new DateTime("@$timestamp", $fromTimeZone);
    $dateTime->setTimezone($toTimeZone);
    echo "Converted Time: " . $dateTime->format("Y-m-d H:i:s") . "<br>";
}
```

22. Problem: Finding the Nearest Future Date

- Find the nearest future date among a list of dates.

Solution: Iterate through the dates and compare with the current date.

```
$dates = ["2023-10-01", "2023-10-05", "2023-09-25"];
$currentDate = new DateTime();
$nearestFutureDate = null;

foreach ($dates as $date) {
    $dateTime = new DateTime($date);
    if ($dateTime > $currentDate && ($nearestFutureDate === null || $dateTime < $nearestFutureDate)) {
        $nearestFutureDate = $dateTime;
    }
}

if ($nearestFutureDate !== null) {
    echo "Nearest Future Date: " . $nearestFutureDate->format("Y-m-d");
} else {
    echo "No future dates found.";
}
```

23. Problem: Detecting Daylight Saving Time (DST) Changes

- Determine if a specific date falls within a DST transition period.

Solution: Check if a date is within a specific range during DST changes.

```
$dateToCheck = new DateTime("2023-03-12");
$dstStart = new DateTime("2023-03-12 02:00:00");
$dstEnd = new DateTime("2023-11-05 02:00:00");

if ($dateToCheck >= $dstStart && $dateToCheck < $dstEnd) {
    echo "The date falls within DST change period.";
} else {
    echo "The date is not within DST change period.";
}
```

24. Problem: Date Difference in Hours and Minutes

- Calculate the time difference between two dates in hours and minutes.

Solution: Use the `date_diff` function and format the result.

```
$startDateTime = new DateTime("2023-08-15 14:30:00");
$endDateTime = new DateTime("2023-08-15 18:45:00");
$interval = $startDateTime->diff($endDateTime);
$hours = $interval->h;
$minutes = $interval->i;
echo "Time Difference: $hours hours $minutes minutes";
```

25. Problem: Generating Monthly Calendar

- Generate a monthly calendar for a specific month and year.

Solution: Use loops to create a calendar grid.

```
$year = 2023;
$month = 9;
$daysInMonth = cal_days_in_month(CAL_GREGORIAN, $month, $year);

for ($day = 1; $day <= $daysInMonth; $day++) {
    $date = new DateTime("$year-$month-$day");
    echo $date->format("Y-m-d (l)") . "<br>";
}
```

26. Problem: Age Validation with Minimum and Maximum Age

- Validate user age, ensuring it falls within a specified range (e.g., 18 to 65 years old).

Solution: Calculate age and check if it's within the range.

```
$birthdate = "1990-05-20";
$minAge = 18;
$maxAge = 65;

$today = new DateTime();
$interval = $today->diff(new DateTime($birthdate));
$age = $interval->y;

if ($age >= $minAge && $age <= $maxAge) {
    echo "Valid age: $age years";
} else {
    echo "Age is outside the valid range.";
}
```

27. Problem: Finding the Last Friday of the Month

- Find the date of the last Friday in the current month.

Solution: Use loops to search for the last Friday.

```
$currentMonth = date("m");
$currentYear = date("Y");
$lastDayOfMonth = date("t");

for ($day = $lastDayOfMonth; $day >= 1; $day--) {
    $date = new DateTime("$currentYear-$currentMonth-$day");
    $dayOfWeek = $date->format("N"); // 5 represents Friday
    if ($dayOfWeek == 5) {
        echo "Last Friday: " . $date->format("Y-m-d");
        break;
    }
}
```

28. Problem: Birthday Reminder

- Create a script that reminds users of upcoming birthdays within a certain timeframe (e.g., 7 days).

Solution: Compare user birthdays with the current date.

```
$userBirthdates = ["1990-09-25", "1985-09-30", "1995-10-05"];
$reminderDays = 7;
$currentDate = new DateTime();

foreach ($userBirthdates as $birthdate) {
    $birthday = new DateTime($birthdate);
    $interval = $currentDate->diff($birthday);
    $daysUntilBirthday = $interval->days;

    if ($daysUntilBirthday >= 0 && $daysUntilBirthday <= $reminderDays) {
        echo "Reminder: Birthday on " . $birthday->format("Y-m-d") . "<br>";
    }
}
```

29. Problem: Working with UNIX Timestamps

- Convert UNIX timestamps to readable dates and vice versa.

Solution: Use `date` and `strtotime` functions for conversion.

```
$timestamp = 1632441600; // UNIX timestamp
$formattedDate = date("Y-m-d H:i:s", $timestamp);
echo "Formatted Date: $formattedDate";

$dateString = "2023-09-25 14:30:00";
$timestamp = strtotime($dateString);
echo "Timestamp: $timestamp";
```

30. Problem: Sorting Dates - Sort an array of dates in ascending order.

Solution: Use the `usort` function with a custom comparison function.

```
$dates = ["2023-09-20", "2023-09-10", "2023-09-15"];
usort($dates, function ($a, $b) {
    return strtotime($a) - strtotime($b);
});

foreach ($dates as $date) {
    echo
}
```

include, include_once, require, and require_once statements.

---

1. Problem: Basic Include

- Create two PHP files, `header.php` and `footer.php`. Use the `include` statement to include them in an `index.php` file to display a simple webpage with a header and footer.

Solution (`header.php`):

```
<header>
    <h1>Website Header</h1>
</header>
```

Solution (`footer.php`):

```
<footer>
    <p>&copy; 2023 My Website</p>
</footer>
```

Solution (`index.php`):

```
<?php include 'header.php'; ?>
<main>
    <p>Welcome to my website content.</p>
</main>
<?php include 'footer.php'; ?>
```

2. Problem: Include Once

- Create two PHP files, `config.php` and `database.php`, that both define a variable. Use the `include_once` statement to include them in an `index.php` file. Verify that variables are not redefined.

Solution (`config.php`):

```
<?php
$configVar = "Configuration Value";
?>
```

Solution (`database.php`):

```
<?php
$dbVar = "Database Value";
?>
```

Solution (`index.php`):

```
<?php include_once 'config.php'; ?>
<?php include_once 'database.php'; ?>

<p>Config: <?php echo $configVar; ?></p>
<p>Database: <?php echo $dbVar; ?></p>
```

3. Problem: Require vs. Include

- Create a PHP file, `restricted.php`, that contains sensitive information. Use both `include` and `require` statements to include this file in an `index.php` file. Observe the differences in behavior.

Solution (`restricted.php`):

```
<?php
$secretKey = "sensitive_key";
?>
```

Solution (`index.php`):

```
<?php include 'restricted.php'; ?>
<p>Include: <?php echo $secretKey; ?></p>

<?php require 'restricted.php'; ?>
<p>Require: <?php echo $secretKey; ?></p>
```

4. Problem: Error Handling with Require

- Create an `index.php` file that attempts to include a non-existent file using both `include` and `require`. Observe the differences in error handling.

Solution (`index.php`):

```
<?php
include 'nonexistent.php'; // Will generate a warning but continue execution
require 'nonexistent.php'; // Will generate a fatal error and halt execution
echo "This will not be displayed due to the fatal error.";
?>
```

5. Problem: Conditional Include

- Create a PHP file, `features.php`, that contains feature-specific code. In an `index.php` file, include `features.php` conditionally based on a variable value.

Solution (`features.php`):

```
<?php
if ($enableFeature) {
    echo "Feature is enabled!";
}
?>
```

Solution (`index.php`):

```
<?php
$enableFeature = true;
include 'features.php'; // Feature is enabled
?>

<?php
$enableFeature = false;
include 'features.php'; // Feature is disabled
?>
```

6. Problem: Using `require_once` for Configuration

- Create a PHP configuration file, `config.php`, that defines important settings. Use `require_once` to include it in multiple files to ensure configuration is loaded only once.

Solution (`config.php`):

```
<?php
$databaseHost = 'localhost';
$databaseUser = 'user';
$databasePassword = 'password';
$databaseName = 'dbname';
?>
```

Solution (`db.php`):

```
<?php
require_once 'config.php';
// Use $databaseHost, $databaseUser, etc., for database connection
?>
```

Solution (`other.php`):

```
<?php
require_once 'config.php';
// Use configuration variables in other parts of the application
?>
```

7. Problem: Dynamic File Inclusion

- Create an `index.php` file that dynamically includes different content files based on user input (e.g., page parameter).

Solution (`index.php`):

```php
<?php
$page = $_GET['page'] ?? 'home'; // Default to 'home'
$allowedPages = ['home', 'about', 'contact'];

if (in_array($page, $allowedPages)) {
    include "$page.php";
} else {
    echo "Page not found.";
}
?>
```

8. Problem: Code Reusability with Include

- Create reusable components like headers, footers, and sidebars in separate PHP files. Use `include` to assemble them in various pages (e.g., `page1.php`, `page2.php`).

Solution (`header.php`, `footer.php`, `sidebar.php`):

```html
<!-- header.php -->
<header>
    <h1>Website Header</h1>
</header>
```

```html
<!-- footer.php -->
<footer>
    <p>&copy; 2023 My Website</p>
</footer>
```

```html
<!-- sidebar.php -->
<aside>
    <h2>Sidebar</h2>
    <!-- Sidebar content -->
</aside>
```

Solution (`page1.php`, `page2.php`):

```php
<?php include 'header.php'; ?>
<main>
    <p>Welcome to Page 1</p>
    <?php include 'sidebar.php'; ?>
</main>
<?php include 'footer.php'; ?>
```

9. Problem: Autoloading Classes with Include

- Create a simple class autoloader that includes class files automatically when needed.

Solution (`autoloader.php`):

```php
<?php
spl_autoload_register(function ($className) {
    include "classes/$className.php";
});
?>
```

Solution (`MyClass.php`):

```php
<?php
class MyClass {
    // Class code
}
?>
```

Solution (`index.php`):

```
<?php
include 'autoloader.php';

$obj = new MyClass();
// The autoloader includes MyClass.php automatically
?>
```

10. Problem: Combining PHP and HTML in Included Files - Create a PHP file, `content.php`, that contains a mix of PHP and HTML. Include this file in an `index.php` file and display its content.

Solution (`content.php

`):

```
<?php
$title = "Dynamic Content";
$message = "This content is generated using PHP.";
?>

<h1><?php echo $title; ?></h1>
<p><?php echo $message; ?></p>
```

Solution (`index.php`):

```
<?php include 'content.php'; ?>
```

FILE HANDLING

Problem 1: How to open a file in PHP? Solution:

```
$file = fopen("example.txt", "r");
if ($file) {
    // File opened successfully
    // Perform operations here
    fclose($file);
} else {
    // Error handling
    echo "Unable to open file.";
}
```

Problem 2: How to read the contents of a file in PHP? Solution:

```
$file = fopen("example.txt", "r");
if ($file) {
    while (!feof($file)) {
        $line = fgets($file);
        // Process each line
    }
    fclose($file);
} else {
    echo "Unable to open file.";
}
```

Problem 3: How to create a new file in PHP? Solution:

```
$file = fopen("newfile.txt", "w");
if ($file) {
    // File created successfully
    // Write data to the file if needed
    fclose($file);
} else {
    echo "Unable to create file.";
}
```

Problem 4: How to upload a file using PHP and handle it? Solution:

```
if ($_FILES["file"]["error"] == 0) {
    $target_dir = "uploads/";
    $target_file = $target_dir . basename($_FILES["file"]["name"]);

    if (move_uploaded_file($_FILES["file"]["tmp_name"], $target_file)) {
        echo "File uploaded successfully.";
    } else {
        echo "Error uploading file.";
    }
} else {
    echo "File upload error: " . $_FILES["file"]["error"];
}
```

Problem 5: How to check if a file exists in PHP? Solution:

```
$filename = "example.txt";
if (file_exists($filename)) {
    echo "$filename exists.";
} else {
    echo "$filename does not exist.";
}
```

Problem 6: How to read a CSV file and parse its data in PHP? Solution:

```
$file = fopen("data.csv", "r");
if ($file) {
    while (($row = fgetcsv($file)) !== false) {
        // Process CSV data in $row
    }
    fclose($file);
} else {
    echo "Unable to open CSV file.";
}
```

Problem 7: How to append data to an existing file in PHP? Solution:

```
$file = fopen("example.txt", "a");
if ($file) {
    fwrite($file, "New data to append\n");
    fclose($file);
} else {
    echo "Unable to open file for appending.";
}
```

Problem 8: How to list all files in a directory using PHP? Solution:

```
$dir = "/path/to/directory/";
$files = scandir($dir);
foreach ($files as $file) {
    if ($file != "." && $file != "..") {
        echo "$file\n";
    }
}
```

Problem 9: How to count the number of lines in a file using PHP? Solution:

```
$file = fopen("example.txt", "r");
if ($file) {
    $lineCount = 0;
    while (!feof($file)) {
        $line = fgets($file);
        $lineCount++;
    }
    fclose($file);
    echo "Number of lines: $lineCount";
} else {
    echo "Unable to open file.";
}
```

Problem 10: How to create a ZIP archive of multiple files in PHP? Solution:

```
$zip = new ZipArchive;
if ($zip->open('archive.zip', ZipArchive::CREATE) === TRUE) {
    $zip->addFile('file1.txt', 'file1.txt');
    $zip->addFile('file2.txt', 'file2.txt');
    $zip->close();
    echo "ZIP archive created successfully.";
} else {
    echo "Failed to create ZIP archive.";
}
```

Problem 11: How to extract files from a ZIP archive in PHP? Solution:

```
$zip = new ZipArchive;
if ($zip->open('archive.zip') === TRUE) {
    $zip->extractTo('extracted/');
    $zip->close();
    echo "Files extracted successfully.";
} else {
    echo "Failed to extract files from ZIP archive.";
}
```

Problem 12: How to delete a file in PHP? Solution:

```
$fileToDelete = "example.txt";
if (unlink($fileToDelete)) {
    echo "$fileToDelete deleted successfully.";
} else {
    echo "Failed to delete $fileToDelete.";
}
```

Problem 13: How to check the file size in PHP? Solution:

```
$fileSize = filesize("example.txt");
echo "File size: $fileSize bytes";
```

Problem 14: How to copy a file to another location in PHP? Solution:

```
 $sourceFile = "source.txt";
$destinationFile = "destination.txt";
if (copy($sourceFile, $destinationFile)) {
    echo "File copied successfully.";
} else {
    echo "Failed to copy the file.";
}
```

Problem 15: How to read and write binary files in PHP? Solution:

```
 $sourceFile = "binary.bin";
$destinationFile = "copy.bin";
$source = fopen($sourceFile, 'rb');
$destination = fopen($destinationFile, 'wb');

if ($source && $destination) {
    while (!feof($source)) {
        $data = fread($source, 1024);
        fwrite($destination, $data);
    }
    fclose($source);
    fclose($destination);
    echo "Binary file copied successfully.";
} else {
    echo "Failed to copy binary file.";
}
```

Problem 16: How to read a JSON file in PHP and decode its contents? Solution:

```
 $jsonFile = "data.json";
$jsonData = file_get_contents($jsonFile);
$data = json_decode($jsonData, true); // Decoding as an associative array
```

Problem 17: How to write data to a JSON file in PHP? Solution:

```
 $data = array("name" => "John", "age" => 30);
$jsonData = json_encode($data);
file_put_contents("data.json", $jsonData);
```

Problem 18: How to handle file uploads with size and type restrictions in PHP? Solution:

```
 $maxFileSize = 5 * 1024 * 1024; // 5MB
$allowedTypes = array("jpg", "png", "gif");

if ($_FILES["file"]["error"] == 0) {
    $target_dir = "uploads/";
    $target_file = $target_dir . basename($_FILES["file"]["name"]);
    $fileType = strtolower(pathinfo($target_file, PATHINFO_EXTENSION));

    if (in_array($fileType, $allowedTypes) && $_FILES["file"]["size"] <= $maxFileSize) {
        if (move_uploaded_file($_FILES["file"]["tmp_name"], $target_file)) {
            echo "File uploaded successfully.";
        }

 else {
            echo "Error uploading file.";
        }
    } else {
        echo "Invalid file type or size exceeds limit.";
    }
} else {
    echo "File upload error: " . $_FILES["file"]["error"];
}
```

Problem 19: How to read and parse XML files in PHP? Solution:

```
$xmlFile = "data.xml";
$xmlData = simplexml_load_file($xmlFile);
// Access XML data using $xmlData
```

Problem 20: How to create and write data to an XML file in PHP? Solution:

```
$xml = new SimpleXMLElement('<root></root>');
$xml->addChild('name', 'John');
$xml->addChild('age', 30);
$xml->asXML('data.xml');
```

Problem 21: How to check if a file is writable in PHP? Solution:

```
$filename = "example.txt";
if (is_writable($filename)) {
    echo "$filename is writable.";
} else {
    echo "$filename is not writable.";
}
```

Problem 22: How to handle file permissions in PHP (chmod)? Solution:

```
$filename = "example.txt";
if (chmod($filename, 0644)) { // Set read and write permissions for owner, read-only for others
    echo "File permissions updated successfully.";
} else {
    echo "Failed to update file permissions.";
}
```

Problem 23: How to read and write to a remote file using PHP (HTTP/FTP)? Solution:

```
// Reading from a remote file (HTTP)
$remoteFile = "http://example.com/file.txt";
$contents = file_get_contents($remoteFile);

// Writing to a remote file (FTP)
$server = "ftp.example.com";
$ftpUsername = "username";
$ftpPassword = "password";
$remotePath = "/public_html/file.txt";

$connection = ftp_connect($server);
if ($connection && ftp_login($connection, $ftpUsername, $ftpPassword)) {
    if (ftp_put($connection, $remotePath, "localfile.txt", FTP_ASCII)) {
        echo "File uploaded to remote server successfully.";
    } else {
        echo "Failed to upload file.";
    }
    ftp_close($connection);
} else {
    echo "FTP connection failed.";
}
```

Problem 24: How to read and write to a remote file using cURL in PHP? Solution:

```
// Reading from a remote file
$url = "http://example.com/file.txt";
$curl = curl_init($url);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($curl);
curl_close($curl);

// Writing to a remote file
$url = "http://example.com/upload.php";
$curl = curl_init($url);
$data = array("name" => "John", "age" => 30);
curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
$response = curl_exec($curl);
curl_close($curl);
```

Problem 25: How to append data to a remote file using cURL in PHP? Solution:

```
$url = "http://example.com/append.php";
$curl = curl_init($url);
$data = array("newData" => "Appended data");
curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
$response = curl_exec($curl);
curl_close($curl);
```

Problem 26: How to search for a specific pattern in a file using regular expressions in PHP? Solution:

```
$fileContents = file_get_contents("example.txt");
$pattern = "/pattern/";
if (preg_match($pattern, $fileContents)) {
    echo "Pattern found in the file.";
} else {
    echo "Pattern not found in the file.";
}
```

Problem 27: How to rename a file in PHP? Solution:

```
$oldName = "oldfile.txt";
$newName = "newfile.txt";
if (rename($oldName, $newName)) {
    echo "File renamed successfully.";
} else {
    echo "Failed to rename the file.";
}
```

Problem 28: How to calculate the checksum (MD5/SHA1) of a file in PHP? Solution:

```
$filename = "example.txt";
$md5Checksum = md5_file($filename);
$sha1Checksum = sha1_file($filename);
```

Problem 29: How to read and write to a file line by line efficiently in PHP? Solution:

```
$filename = "example.txt";
$readFile = fopen($filename, "r");
$writeFile = fopen("output.txt", "w");

if ($readFile && $writeFile) {
    while (($line = fgets($readFile)) !== false) {
        // Process and write each line to the output file
        fwrite($writeFile, $line);
    }
    fclose($readFile);
    fclose($writeFile);
    echo "File processed line by line.";
} else {
    echo "Error opening files.";
}
```

Problem 30: How to sort and filter file listings in a directory by various criteria using PHP? Solution:

```
$dir = "/path/to/directory/";
$files = scandir($dir);

// Sort files by modification time (most recent first)
array_multisort(array_map('filemtime', $files), SORT_DESC, $files);

// Filter files by extension (e.g., only .txt files)
$filteredFiles = array_filter($files, function ($file) {
    return pathinfo($file, PATHINFO_EXTENSION) == 'txt';
});

// Output the sorted and filtered file list
foreach ($filteredFiles as $file) {
    echo "$file\n";
}
```

Overview for file It seems like you mentioned a function called `fileattime()`, but there is no built-in PHP function by that name. However, I can provide you with more file-related functions in PHP:

1. **File Attributes and Information:**
    - `filemtime()` : Returns the last modification time of a file.
    - `fileatime()` : Returns the last access time of a file.
    - `filectime()` : Returns the creation time of a file.
    - `filesize()` : Returns the size of a file in bytes.

    ```
    $file = "example.txt";
    echo "Last Modified: " . date("F d Y H:i:s.", filemtime($file));
    echo "Last Accessed: " . date("F d Y H:i:s.", fileatime($file));
    echo "Created: " . date("F d Y H:i:s.", filectime($file));
    echo "File Size: " . filesize($file) . " bytes";
    ```

2. **File and Directory Manipulation:**
    - `rename()` : Renames a file or directory.
    - `copy()` : Copies a file.
    - `mkdir()` : Creates a directory.
    - `rmdir()` : Removes a directory.

    ```
    rename("oldfile.txt", "newfile.txt");
    copy("source.txt", "destination.txt");
    mkdir("new_directory");
    rmdir("empty_directory");
    ```

3. **File Permissions:**
    - `chmod()` : Changes file permissions.

    ```
    chmod("file.txt", 0644); // Set read and write permissions for owner, read-only for others
    ```

4. **File Locking:**

- `flock()`: Provides file locking.

```
$file = fopen("example.txt", "w");
if (flock($file, LOCK_EX)) {
    // Exclusive lock acquired
    fwrite($file, "Locked text");
    flock($file, LOCK_UN); // Release the lock
} else {
    echo "Unable to acquire lock.";
}
fclose($file);
```

These are additional file-related functions in PHP that you can use for various file operations, including getting file attributes, manipulating files and directories, managing permissions, and file locking.

These problems and solutions cover a range of file handling scenarios in PHP and should be helpful for PHP developers at different skill levels preparing for interviews or competitive programming challenges.

COOKIE & SESSION

Certainly, here are 10 professional problems related to PHP cookies and sessions, along with solutions and code examples. These are suitable for various levels of PHP developers and can be helpful in remote job interviews, especially in the context of competitive programming platforms like Codeforces, LeetCode, Kaggle, and CodeChef.

Problem 1: How to set a cookie in PHP?

Solution:

```
// Set a cookie that expires in 1 hour
setcookie("user", "John", time() + 3600, "/");
```

Problem 2: How to retrieve and display a cookie value in PHP?

Solution:

```
if (isset($_COOKIE["user"])) {
    echo "Welcome, " . $_COOKIE["user"];
} else {
    echo "Cookie not set.";
}
```

Problem 3: How to delete a cookie in PHP?

Solution:

```
// Set the cookie's expiration time to the past
setcookie("user", "", time() - 3600, "/");
```

Problem 4: How to use sessions in PHP to store and retrieve user data?

Solution:

```
// Start a session
session_start();

// Store data in the session
$_SESSION["username"] = "John";

// Retrieve data from the session
if (isset($_SESSION["username"])) {
    echo "Welcome, " . $_SESSION["username"];
} else {
    echo "Session data not set.";
}

// End the session
session_destroy();
```

Problem 5: How to set session timeout in PHP?

Solution:

```
// Start a session with a custom timeout (e.g., 30 minutes)
session_set_cookie_params(1800);
session_start();
```

Problem 6: How to check if a user is logged in using sessions?

Solution:

```
session_start();

if (isset($_SESSION["loggedin"]) && $_SESSION["loggedin"] === true) {
    echo "You are logged in.";
} else {
    echo "Please log in.";
}
```

Problem 7: How to implement a secure login system using PHP sessions and passwords hashing?

Solution:

```
session_start();

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Validate username and password (check database)
    $username = $_POST["username"];
    $password = $_POST["password"];
    $hashedPassword = password_hash("user_password", PASSWORD_DEFAULT);

    if (password_verify($password, $hashedPassword)) {
        $_SESSION["loggedin"] = true;
        echo "Login successful.";
    } else {
        echo "Login failed.";
    }
}
```

Problem 8: How to protect against session fixation attacks in PHP?

Solution:

```
session_start();

if (!isset($_SESSION["initiated"])) {
    session_regenerate_id();
    $_SESSION["initiated"] = true;
}
```

Problem 9: How to use PHP sessions to implement a shopping cart functionality?

Solution:

```
session_start();

// Add an item to the shopping cart
if (isset($_POST["add_to_cart"])) {
    $item_id = $_POST["item_id"];
    $quantity = $_POST["quantity"];

    // Store item and quantity in the session
    $_SESSION["cart"][$item_id] = $quantity;
}

// Display the shopping cart contents
if (isset($_SESSION["cart"])) {
    foreach ($_SESSION["cart"] as $item_id => $quantity) {
        echo "Item ID: $item_id, Quantity: $quantity<br>";
    }
} else {
    echo "Cart is empty.";
}
```

Problem 10: How to implement user logout functionality using PHP sessions?

Solution:

```
session_start();

// Check if the user is logged in
if (isset($_SESSION["loggedin"]) && $_SESSION["loggedin"] === true) {
    // Unset all session variables
    $_SESSION = array();

    // Destroy the session
    session_destroy();

    echo "Logged out successfully.";
} else {
    echo "You are not logged in.";
}
```

FILTERS FOR VALIDATION DATA

Certainly, here are 10 professional problems related to PHP filters and advanced filtering with validation, along with solutions and code examples. These problems cover various filtering scenarios and are suitable for PHP developers at different levels preparing for interviews or competitive programming challenges.

Problem 1: How to validate an email address using PHP filters?

Solution:

```
$email = "user@example.com";
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "Valid email address.";
} else {
    echo "Invalid email address.";
}
```

Problem 2: How to validate an IP address using PHP filters?

Solution:

```
$ip = "192.168.1.1";
if (filter_var($ip, FILTER_VALIDATE_IP)) {
    echo "Valid IP address.";
} else {
    echo "Invalid IP address.";
}
```

Problem 3: How to sanitize and filter input data to prevent XSS attacks?

Solution:

```
$userInput = "<script>alert('XSS attack');</script>";
$filteredInput = filter_var($userInput, FILTER_SANITIZE_STRING);
echo $filteredInput;
```

Problem 4: How to validate and sanitize an integer input using PHP filters?

Solution:

```
$userInput = "123";
$filteredInput = filter_var($userInput, FILTER_VALIDATE_INT);

if ($filteredInput !== false) {
    echo "Valid integer: " . $filteredInput;
} else {
    echo "Invalid integer.";
}
```

Problem 5: How to validate and sanitize URL input using PHP filters?

Solution:

```
$url = "https://www.example.com";
$filteredUrl = filter_var($url, FILTER_VALIDATE_URL);

if ($filteredUrl !== false) {
    echo "Valid URL: " . $filteredUrl;
} else {
    echo "Invalid URL.";
}
```

Problem 6: How to filter an array of values using a custom filter function in PHP?

Solution:

```
function customFilter($value) {
    return $value > 5;
}

$array = [3, 6, 9, 2, 7];
$filteredArray = array_filter($array, "customFilter");

print_r($filteredArray);
```

Problem 7: How to validate and filter an array of email addresses in PHP?

Solution:

```
$emailArray = ["user@example.com", "invalid-email", "another@example.com"];
$filteredEmails = array_filter($emailArray, function ($email) {
    return filter_var($email, FILTER_VALIDATE_EMAIL);
});

print_r($filteredEmails);
```

Problem 8: How to use a custom filter to validate and filter a list of usernames?

Solution:

```
function validateUsername($username) {
    // Custom validation logic (e.g., alphanumeric, length constraints)
    return preg_match('/^[a-zA-Z0-9_]+$/', $username) && strlen($username) >= 5;
}

$usernames = ["user123", "invalid*", "john_doe"];
$filteredUsernames = array_filter($usernames, "validateUsername");

print_r($filteredUsernames);
```

Problem 9: How to validate and filter an array of dates using PHP filters?

Solution:

```
$dates = ["2022-01-15", "invalid-date", "2023-12-25"];
$filteredDates = array_filter($dates, function ($date) {
    return (bool) strtotime($date);
});

print_r($filteredDates);
```

Problem 10: How to validate and filter an array of values with a custom callback function and return the filtered array in reverse order?

Solution:

```
$values = [10, 15, 20, 5, 30];
$filteredValues = array_filter($values, function ($value) {
    return $value % 2 == 0;
});

$reversedFilteredValues = array_reverse($filteredValues);

print_r($reversedFilteredValues);
```

callback functions, JSON handling, and exception handling.

Certainly, here are 20 professional problems related to PHP callback functions, JSON, and exceptions, along with solutions and code examples. These problems cover various scenarios and are suitable for PHP developers at different levels preparing for interviews or competitive programming challenges.

Problem 1: How to create and use a callback function in PHP?

Solution:

```
function applyCallback($callback, $value) {
    return $callback($value);
}

$double = function ($x) {
    return $x * 2;
};

$result = applyCallback($double, 5);
echo $result; // Output: 10
```

Problem 2: How to sort an array of strings in PHP using a custom callback function?

Solution:

```
$fruits = ["apple", "banana", "cherry"];

usort($fruits, function ($a, $b) {
    return strlen($a) - strlen($b);
});

print_r($fruits); // Output: ["apple", "banana", "cherry"]
```

Problem 3: How to decode a JSON string in PHP and access its values?

Solution:

```
$jsonString = '{"name": "John", "age": 30}';
$data = json_decode($jsonString, true);

echo $data["name"]; // Output: John
echo $data["age"];  // Output: 30
```

Problem 4: How to encode an array into JSON format in PHP?

Solution:

```
$data = ["name" => "John", "age" => 30];
$jsonString = json_encode($data);

echo $jsonString; // Output: {"name":"John","age":30}
```

Problem 5: How to handle JSON parsing errors and exceptions in PHP?

Solution:

```
$jsonString = '{"name": "John", "age": 30,}';
try {
    $data = json_decode($jsonString, true, 512, JSON_THROW_ON_ERROR);
    echo $data["name"];
} catch (JsonException $e) {
    echo "JSON Error: " . $e->getMessage();
}
```

Problem 6: How to use a callback function with `array_map()` in PHP?

Solution:

```
$numbers = [1, 2, 3, 4, 5];

$double = function ($n) {
    return $n * 2;
};

$doubledNumbers = array_map($double, $numbers);
print_r($doubledNumbers); // Output: [2, 4, 6, 8, 10]
```

Problem 7: How to filter an array using a custom callback function with `array_filter()` in PHP?

Solution:

```
$numbers = [1, 2, 3, 4, 5];

$isEven = function ($n) {
    return $n % 2 == 0;
};

$evenNumbers = array_filter($numbers, $isEven);
print_r($evenNumbers); // Output: [2, 4]
```

Problem 8: How to handle exceptions in PHP and create custom exception classes?

Solution:

```
class CustomException extends Exception {}

try {
    throw new CustomException("Custom exception message");
} catch (CustomException $e) {
    echo "Caught exception: " . $e->getMessage();
}
```

Problem 9: How to use `try`, `catch`, and `finally` blocks in PHP for exception handling?

Solution:

```
try {
    // Code that may throw an exception
    throw new Exception("An error occurred");
} catch (Exception $e) {
    echo "Caught exception: " . $e->getMessage();
} finally {
    echo "This will always execute.";
}
```

Problem 10: How to throw an exception with a custom message and code in PHP?

Solution:

```
try {
    $age = 15;
    if ($age < 18) {
        throw new Exception("You must be 18 or older to access this content.", 403);
    }
} catch (Exception $e) {
    echo "Error {$e->getCode()}: {$e->getMessage()}";
}
```

Problem 11: How to create a custom error handler in PHP for uncaught exceptions?

Solution:

```
function customErrorHandler($errno, $errstr, $errfile, $errline) {
    echo "Error: [$errno] $errstr at $errfile:$errline";
}

set_error_handler("customErrorHandler");

try {
    // Code that may throw an exception
    throw new Exception("An error occurred");
} catch (Exception $e) {
    // This will not catch the exception, but customErrorHandler will handle errors
    echo "Caught exception: " . $e->getMessage();
}
```

Problem 12: How to create a custom error handler in PHP for errors and exceptions?

Solution:

```
function customErrorHandler($errno, $errstr, $errfile, $errline) {
    echo "Error: [$errno] $errstr at $errfile:$errline";
}

function customExceptionHandler($e) {
    echo "Exception: " . $e->getMessage();
}

set_error_handler("customErrorHandler");
set_exception_handler("customExceptionHandler");

try {
    // Code that may throw an exception
    throw new Exception("An error occurred");
} catch (Exception $e) {
    // This will catch the exception
    echo "Caught exception: " . $e->getMessage();
}
```

Problem 13: How to use a callback function to filter and transform an array of strings in PHP?

Solution:

```
$names = ["John", "Alice", "Bob"];

$formatName = function ($name) {
    return "Hello, " . ucfirst($name);
};

$greetings = array_map($formatName, $names);
print_r($greetings);
```

Problem 14: How to handle and log exceptions in a PHP application?

Solution:

```
class Logger {
    public static function logException(Exception $e) {
        error_log("Exception: " . $e->getMessage());
    }
}

try {
    // Code that may throw an exception
    throw new Exception("An error occurred");
} catch (Exception $e) {
    Logger::logException($e);
    echo "Caught exception: " . $e->getMessage();
}
```

Problem 15: How to validate and filter an array of user inputs using callback functions in PHP?

Solution:

```
$userInputs = ["John", "invalid*", "Alice"];

$isValidInput = function ($input) {
    return preg_match('/^[a-zA-Z]+$/', $input);
};

$filteredInputs = array_filter($userInputs, $isValidInput);
print_r($filteredInputs);
```

Problem 16: How to throw a custom exception when validating user inputs in PHP?

Solution:

```
class InvalidInputException extends Exception {}

function validateInput($input) {
    if (!preg_match('/^[a-zA-Z]+$/', $input)) {
        throw new InvalidInputException("Invalid input: $input");
    }
}

$userInputs = ["John", "invalid*", "

Alice"];

try {
    foreach ($userInputs as $input) {
        validateInput($input);
    }
} catch (InvalidInputException $e) {
    echo "Caught exception: " . $e->getMessage();
}
```

Problem 17: How to create and use a custom filter function to transform an array in PHP?

Solution:

```
function customFilter($value) {
    return $value * 2;
}

$numbers = [1, 2, 3, 4, 5];
$transformedNumbers = array_map("customFilter", $numbers);
print_r($transformedNumbers);
```

Problem 18: How to use the `array_reduce()` function with a custom callback to calculate the sum of an array in PHP?

Solution:

```
$numbers = [1, 2, 3, 4, 5];

$sumCallback = function ($carry, $item) {
    return $carry + $item;
};

$sum = array_reduce($numbers, $sumCallback);
echo $sum; // Output: 15
```

Problem 19: How to filter an array of products by price range using a custom callback in PHP?

Solution:

```
$products = [
    ["name" => "Product A", "price" => 10],
    ["name" => "Product B", "price" => 20],
    ["name" => "Product C", "price" => 30],
];

$filterByPrice = function ($product) {
    return $product["price"] >= 20;
};

$filteredProducts = array_filter($products, $filterByPrice);
print_r($filteredProducts);
```

Problem 20: How to use a custom exception class and handle exceptions in PHP when parsing JSON data?

Solution:

```
 class JSONParseException extends Exception {}

$jsonString = '{"name": "John", "age": 30,}';
try {
    $data = json_decode($jsonString, true, 512, JSON_THROW_ON_ERROR);
    echo $data["name"];
} catch (JsonException $e) {
    throw new JSONParseException("JSON parsing error: " . $e->getMessage());
} catch (JSONParseException $e) {
    echo "Caught exception: " . $e->getMessage();
}
```

PHP OOP

OOP stands for Object-Oriented Programming, and it is a programming paradigm used in PHP (and many other programming languages) for organizing and structuring code. Object-oriented programming is based on the concept of "objects," which are instances of classes. Here's a brief overview of OOP in PHP:

PHP OOP : Class,objects, constructor, destructor,inheritance, polymorphism, encapsulation, and abstraction.

---

1. **Creating a Class**: Define a PHP class to represent a basic object and demonstrate its instantiation.
2. **Constructor Usage**: Explain the purpose of a constructor in PHP OOP and provide an example.
3. **Destructor Function**: Discuss the role of a destructor and show how it's used in PHP.
4. **Inheritance Basics**: Create a base class and a derived class demonstrating inheritance.
5. **Method Overriding**: Implement method overriding in PHP OOP.
6. **Access Modifiers**: Explain public, private, and protected access modifiers and provide examples.
7. **Encapsulation**: Showcase encapsulation in a PHP class.
8. **Abstraction**: Create an abstract class and demonstrate how it's used.
9. **Interfaces**: Define an interface and implement it in a class.
10. **Polymorphism**: Explain polymorphism and give an example using method overloading.
11. **Static Methods**: Discuss static methods in PHP classes and show their usage.
12. **Constants in Classes**: Define constants within a class and explain their significance.
13. **Class Properties**: Create class properties and demonstrate their usage.
14. **Object Cloning**: Explain object cloning in PHP and provide a code example.
15. **Dependency Injection**: Discuss the concept of dependency injection in PHP OOP.
16. **Magic Methods**: Explain __get, __set, __isset, and __unset magic methods.
17. **Abstract Methods**: Define abstract methods in an abstract class and implement them.
18. **Final Classes and Methods**: Discuss final classes and methods in PHP OOP.
19. **Traits**: Define a trait and use it in multiple classes.
20. **Method Chaining**: Show how method chaining can be implemented in PHP.
21. **Type Hinting**: Explain type hinting in method parameters and return values.
22. **Late Static Binding**: Discuss late static binding and provide an example.
23. **Exception Handling**: Handle exceptions in PHP OOP code.
24. **File Handling with Classes**: Create a class for file manipulation (e.g., reading and writing).
25. **Database Interaction**: Develop a class for database interaction (CRUD operations).
26. **User Authentication**: Implement user authentication using PHP OOP.
27. **Pagination Class**: Create a pagination class for handling large datasets.
28. **Logger Class**: Build a logger class for debugging and error tracking.
29. **Dependency Management**: Explain and implement dependency management in PHP OOP.
30. **Unit Testing**: Discuss the importance of unit testing in PHP OOP and provide a basic example.

Class,objects, constructor, destructor,and inheritance

Problem 1: Creating a Class Problem Statement: Create a PHP class called `Person` with properties `name` and `age`, and a constructor to initialize these properties.

Solution:

```
class Person {
    public $name;
    public $age;

    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }
}
```

Problem 2: Inheritance Problem Statement: Create a subclass `Student` that inherits from `Person` and adds a `studentId` property. Override the constructor to include `studentId`.

Solution:

```
class Student extends Person {
    public $studentId;

    public function __construct($name, $age, $studentId) {
        parent::__construct($name, $age);
        $this->studentId = $studentId;
    }
}
```

Problem 3: Destructor Problem Statement: Implement a destructor in the `Person` class that displays a message when an object is destroyed.

Solution:

```
class Person {
    public $name;
    public $age;

    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }

    public function __destruct() {
        echo "The object is destroyed.";
    }
}
```

Problem 4: Access Modifiers Problem Statement: Add private access modifiers to the properties in the `Person` class and create public getter and setter methods to access and modify them.

Solution:

```
class Person {
    private $name;
    private $age;

    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }

    public function getName() {
        return $this->name;
    }

    public function setName($name) {
        $this->name = $name;
    }

    public function getAge() {
        return $this->age;
    }

    public function setAge($age) {
        $this->age = $age;
    }
}
```

Problem 5: Static Method Problem Statement: Create a static method in the `Person` class to count the number of `Person` objects created.

Solution:

```
class Person {
    private $name;
    private $age;
    private static $count = 0;

    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
        self::$count++;
    }

    public static function getCount() {
        return self::$count;
    }
}
```

Problem 6: Abstract Class Problem Statement: Create an abstract class `Shape` with an abstract method `area()` . Create two subclasses `Circle` and `Rectangle` that implement the `area()` method.

Solution:

```
abstract class Shape {
    abstract public function area();
}

class Circle extends Shape {
    private $radius;

    public function __construct($radius) {
        $this->radius = $radius;
    }

    public function area() {
        return 3.14 * $this->radius * $this->radius;
    }
}

class Rectangle extends Shape {
    private $length;
    private $width;

    public function __construct($length, $width) {
        $this->length = $length;
        $this->width = $width;
    }

    public function area() {
        return $this->length * $this->width;
    }
}
```

Problem 7: Interface Problem Statement: Create an interface `Resizable` with a method `resize($percentage)` and implement it in the `Circle` and `Rectangle` classes.

Solution:

```
interface Resizable {
    public function resize($percentage);
}

class Circle extends Shape implements Resizable {
    // ... (previous code)

    public function resize($percentage) {
        $this->radius *= ($percentage / 100);
    }
}

class Rectangle extends Shape implements Resizable {
    // ... (previous code)

    public function resize($percentage) {
        $this->length *= ($percentage / 100);
        $this->width *= ($percentage / 100);
    }
}
```

Problem 8: Magic Methods Problem Statement: Implement the `__toString()` magic method in the `Person` class to return a string representation of the object.

Solution:

```
class Person {
    private $name;
    private $age;

    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }

    public function __toString() {
        return "Name: {$this->name}, Age: {$this->age}";
    }
}
```

Problem 9: Encapsulation Problem Statement: Encapsulate the properties of a `Car` class and create methods to start and stop the car.

Solution:

```
class Car {
    private $isRunning = false;

    public function start() {
        $this->isRunning = true;
    }

    public function stop() {
        $this->isRunning = false;
    }

    public function isRunning() {
        return $this->isRunning;
    }
}
```

Problem 10: Dependency Injection Problem Statement: Implement dependency injection in a `Logger` class that logs messages to a file.

Solution:

```
class Logger {
    private $logFile;

    public function __construct($logFile) {
        $this->logFile = $logFile;
    }

    public function log($message) {
        file_put_contents($this->logFile, $message . "\n", FILE_APPEND);
    }
}
```

Problem 11: Singleton Pattern Problem Statement: Implement the Singleton pattern for a `DatabaseConnection` class to ensure only one database connection is created.

Solution:

```
class DatabaseConnection {
    private static $instance;

    private function __construct() {
        // Private constructor to prevent direct instantiation.
    }

    public static function getInstance() {
        if (!self::$instance) {
            self::$instance = new DatabaseConnection();
        }
        return self::$instance;
    }

    // Rest of the database connection methods...
}
```

Problem 12: Exception Handling Problem Statement: Create a custom exception class `CustomException` and throw it when a specific condition is not met in your code.

Solution:

```
class CustomException extends Exception {}

function someFunction($value) {
    if ($value < 0) {
        throw new CustomException("Value cannot be negative.");
    }
    // Rest of the function...
}
```

Problem 13: Namespaces Problem Statement: Use namespaces to organize classes into

a `Utils` namespace and access them accordingly.

Solution:

```
namespace Utils;

class Helper {
    public static function doSomething() {
        // ...
    }
}
```

Problem 14: Traits Problem Statement: Create a trait `LoggerTrait` that provides logging capabilities and use it in multiple classes.

Solution:

```
trait LoggerTrait {
    public function log($message) {
        // Log the message...
    }
}

class User {
    use LoggerTrait;
    // ...
}

class Order {
    use LoggerTrait;
    // ...
}
```

Problem 15: Dependency Injection Container Problem Statement: Implement a Dependency Injection Container that can resolve and inject dependencies into classes.

Solution:

```
class Container {
    private $bindings = [];

    public function bind($abstract, $concrete) {
        $this->bindings[$abstract] = $concrete;
    }

    public function resolve($abstract) {
        if (isset($this->bindings[$abstract])) {
            $concrete = $this->bindings[$abstract];
            if (is_callable($concrete)) {
                return $concrete();
            }
            return new $concrete;
        }
        throw new Exception("Binding not found for {$abstract}");
    }
}

// Usage:
$container = new Container();
$container->bind('Logger', function () {
    return new Logger();
});

$logger = $container->resolve('Logger');
```

Problem 16: Factory Pattern Problem Statement: Implement a Factory pattern to create different types of objects.

Solution:

```
interface Product {
    public function getName();
}

class ConcreteProductA implements Product {
    public function getName() {
        return "Product A";
    }
}

class ConcreteProductB implements Product {
    public function getName() {
        return "Product B";
    }
}

class ProductFactory {
    public static function createProduct($type) {
        if ($type === 'A') {
            return new ConcreteProductA();
        } elseif ($type === 'B') {
            return new ConcreteProductB();
        }
        throw new InvalidArgumentException("Invalid product type");
    }
}
```

Problem 17: Observer Pattern Problem Statement: Implement the Observer pattern to notify multiple objects when a subject's state changes.

Solution:

```
interface Observer {
    public function update($data);
}

class Subject {
    private $observers = [];

    public function addObserver(Observer $observer) {
        $this->observers[] = $observer;
    }

    public function notifyObservers($data) {
        foreach ($this->observers as $observer) {
            $observer->update($data);
        }
    }
}
```

Problem 18: Command Pattern Problem Statement: Implement the Command pattern to encapsulate a request as an object and parameterize clients with queues, requests, and operations.

Solution:

```
interface Command {
    public function execute();
}

class Light {
    public function on() {
        echo "Light is on.";
    }

    public function off() {
        echo "Light is off.";
    }
}

class LightOnCommand implements Command {
    private $light;

    public function __construct(Light $light) {
        $this->light = $light;
    }

    public function execute() {
        $this->light->on();
    }
}

class LightOffCommand implements Command {
    private $light;

    public function __construct(Light $light) {
        $this->light = $light;
    }

    public function execute() {
        $this->light->off();
    }
}
```

Problem 19: Strategy Pattern Problem Statement: Implement the Strategy pattern to define a family of algorithms, encapsulate each one, and make them interchangeable.

Solution:

```
interface PaymentStrategy {
    public function pay($amount);
}

class CreditCardPayment implements PaymentStrategy {
    public function pay($amount) {
        echo "Paid $amount via Credit Card.";
    }
}

class PayPalPayment implements PaymentStrategy {
    public function pay($amount) {
        echo "Paid $amount via PayPal.";
    }
}

class ShoppingCart {
    private $paymentMethod;

    public function setPaymentMethod(PaymentStrategy $paymentMethod) {
        $this->paymentMethod = $paymentMethod;
    }

    public function checkout($amount) {
        $this->paymentMethod->pay($amount);
    }
}
```

Problem 20: Decorator Pattern Problem Statement: Implement the Decorator pattern to add behavior to individual objects, either statically or dynamically, without affecting the behavior of other objects from the same class.

Solution:

```
interface Coffee {
    public function cost();
}

class SimpleCoffee implements Coffee {
    public function cost() {
        return 5;
    }
}

class MilkDecorator implements Coffee {
    private $coffee;

    public function __construct(Coffee $coffee) {
        $this->coffee = $coffee;
    }

    public function cost() {
        return $this->coffee->cost() + 2;
    }
}

class SugarDecorator implements Coffee {
    private $coffee;

    public function __construct(Coffee $coffee) {
        $this->coffee = $coffee;
    }

    public function cost() {
        return $this->coffee->cost() + 1;
    }
}
```

Problem 21: State Pattern Problem Statement: Implement the State pattern to allow an object to alter its behavior when its internal state changes.

Solution:

```
interface State {
    public function doAction(Context $context);
}

class Context {
    private $state;

    public function setState(State $state) {
        $this->state = $state;
    }

    public function getState() {
        return $this->state;
    }

    public function request() {
        $this->state->doAction($this);
    }
}

class ConcreteStateA implements State {
    public function doAction(Context $context) {
        echo "State A";
        $context->setState(new ConcreteStateB());
    }
}

class ConcreteStateB implements State {
    public function doAction(Context $context) {
        echo "State B";
        $context->setState(new ConcreteStateA());
    }
}
```

Problem 22: Bridge Pattern Problem Statement: Implement the Bridge pattern to separate an objectâ€™s abstraction from its implementation so that the two can vary independently.

Solution:

```
 interface DrawingAPI {
    public function drawCircle($x, $y, $radius);
}

class DrawingAPI1 implements DrawingAPI {
    public function drawCircle($x, $y, $radius) {
        echo "API1.circle at ($x,$y) with radius $radius.";
    }
}

class DrawingAPI2 implements DrawingAPI {
    public function drawCircle($x, $y, $radius) {
        echo "API2.circle at ($x,$y) with radius $radius.";
    }
}

abstract class Shape {
    protected $drawingAPI;

    protected function __construct(DrawingAPI $drawingAPI) {
        $this->drawingAPI = $drawingAPI;
    }

    abstract public function draw();
}

class CircleShape extends Shape {
    private $x, $y, $radius;

    public function __construct($x, $y, $radius,

 DrawingAPI $drawingAPI) {
        parent::__construct($drawingAPI);
        $this->x = $x;
        $this->y = $y;
        $this->radius = $radius;
    }

    public function draw() {
        $this->drawingAPI->drawCircle($this->x, $this->y, $this->radius);
    }
}
```

Problem 23: Chain of Responsibility Pattern Problem Statement: Implement the Chain of Responsibility pattern to pass requests along a chain of handlers, allowing multiple objects to handle the request.

Solution:

```php
abstract class Logger {
    protected $nextLogger;

    public function setNextLogger(Logger $nextLogger) {
        $this->nextLogger = $nextLogger;
    }

    abstract public function logMessage($level, $message);
}

class ConsoleLogger extends Logger {
    public function logMessage($level, $message) {
        if ($level === 'CONSOLE') {
            echo "Console: $message\n";
        } elseif ($this->nextLogger !== null) {
            $this->nextLogger->logMessage($level, $message);
        }
    }
}

class FileLogger extends Logger {
    public function logMessage($level, $message) {
        if ($level === 'FILE') {
            echo "File: $message\n";
        } elseif ($this->nextLogger !== null) {
            $this->nextLogger->logMessage($level, $message);
        }
    }
}

class ErrorLogger extends Logger {
    public function logMessage($level, $message) {
        if ($level === 'ERROR') {
            echo "Error: $message\n";
        } elseif ($this->nextLogger !== null) {
            $this->nextLogger->logMessage($level, $message);
        }
    }
}
```

Problem 24: Visitor Pattern Problem Statement: Implement the Visitor pattern to represent an operation to be performed on the elements of an object structure.

Solution:

```
interface Visitor {
    public function visitElementA(ElementA $elementA);
    public function visitElementB(ElementB $elementB);
}

interface Element {
    public function accept(Visitor $visitor);
}

class ElementA implements Element {
    public function accept(Visitor $visitor) {
        $visitor->visitElementA($this);
    }
}

class ElementB implements Element {
    public function accept(Visitor $visitor) {
        $visitor->visitElementB($this);
    }
}

class ConcreteVisitor implements Visitor {
    public function visitElementA(ElementA $elementA) {
        echo "Visited ElementA\n";
    }

    public function visitElementB(ElementB $elementB) {
        echo "Visited ElementB\n";
    }
}
```

Problem 25: Memento Pattern Problem Statement: Implement the Memento pattern to capture and externalize an object's internal state so that the object can be restored to that state later.

Solution:

```
class Memento {
    private $state;

    public function __construct($state) {
        $this->state = $state;
    }

    public function getState() {
        return $this->state;
    }
}

class Originator {
    private $state;

    public function setState($state) {
        $this->state = $state;
    }

    public function getState() {
        return $this->state;
    }

    public function saveToMemento() {
        return new Memento($this->state);
    }

    public function restoreFromMemento(Memento $memento) {
        $this->state = $memento->getState();
    }
}

class Caretaker {
    private $mementos = [];

    public function addMemento(Memento $memento) {
        $this->mementos[] = $memento;
    }

    public function getMemento($index) {
        return $this->mementos[$index];
    }
}
```

Problem 26: Proxy Pattern Problem Statement: Implement the Proxy pattern to provide a surrogate or placeholder for another object to control access to it.

Solution:

```
interface Image {
    public function display();
}

class RealImage implements Image {
    private $filename;

    public function __construct($filename) {
        $this->filename = $filename;
        $this->loadImageFromDisk();
    }

    private function loadImageFromDisk() {
        echo "Loading image: {$this->filename}\n";
    }

    public function display() {
        echo "Displaying image: {$this->filename}\n";
    }
}

class ProxyImage implements Image {
    private $realImage;
    private $filename;

    public function __construct($filename) {
        $this->filename = $filename;
    }

    public function display() {
        if ($this->realImage === null) {
            $this->realImage = new RealImage($this->filename);
        }
        $this->realImage->display();
    }
}
```

Problem 27: Composite Pattern Problem Statement: Implement the Composite pattern to compose objects into tree structures to represent part-whole hierarchies.

Solution:

```
interface Graphic {
    public function draw();
}

class Circle implements Graphic {
    public function draw() {
        echo "Draw a circle.\n";
    }
}

class Square implements Graphic {
    public function draw() {
        echo "Draw a square.\n";
    }
}

class CompositeGraphic implements Graphic {
    private $graphics = [];

    public function add(Graphic $graphic) {
        $this->graphics[] = $graphic;
    }

    public function draw() {
        foreach ($this->graphics as $graphic) {
            $graphic->draw();
        }
    }
}
```

Problem 28: Flyweight Pattern Problem Statement: Implement the Flyweight pattern to minimize memory usage or computational expenses by sharing as much as possible with similar objects.

Solution:

```
class Flyweight {
    private $sharedState;

    public function __construct($sharedState) {
        $this->sharedState = $sharedState;
    }

    public function operation($uniqueState) {
        echo "Shared: {$this->sharedState}, Unique: {$uniqueState}\n";
    }
}

class FlyweightFactory {
    private $flyweights = [];

    public function getFlyweight($sharedState) {
        if (!isset($this->flyweights[$sharedState])) {
            $this->flyweights[$sharedState] = new Flyweight($sharedState);
        }
        return $this->flyweights[$sharedState];
    }
}
```

Problem 29: Interpreter Pattern Problem Statement: Implement the Interpreter pattern to define a grammar for interpreting a language and provide an interpreter for the language.

Solution:

```
interface Expression {
    public function interpret($context);
}

class TerminalExpression implements Expression {
    private $data;

    public function __construct($data) {
        $this->data = $data;
    }

    public function interpret($context) {
        return strpos($context, $this->data) !== false;
    }
}

class OrExpression implements Expression {
    private $expr1;
    private $expr2;

    public function __construct(Expression $expr1, Expression $expr2

) {
        $this->expr1 = $expr1;
        $this->expr2 = $expr2;
    }

    public function interpret($context) {
        return $this->expr1->interpret($context) || $this->expr2->interpret($context);
    }
}
```

Problem 30: Mediator Pattern Problem Statement: Implement the Mediator pattern to define an object that encapsulates how a set of objects interact.

Solution:

```
class User {
    private $name;
    private $mediator;

    public function __construct($name, Mediator $mediator) {
        $this->name = $name;
        $this->mediator = $mediator;
    }

    public function getName() {
        return $this->name;
    }

    public function sendMessage($message) {
        $this->mediator->sendMessage($this, $message);
    }
}

class ChatMediator {
    private $users = [];

    public function addUser(User $user) {
        $this->users[] = $user;
    }

    public function sendMessage(User $user, $message) {
        foreach ($this->users as $u) {
            if ($u !== $user) {
                echo "{$user->getName()} sends message to {$u->getName()}: $message\n";
            }
        }
    }
}
```

Polymorphism, Encapsulation, and Abstraction

Problem 1: Polymorphism Problem Statement: Create a PHP class hierarchy for different geometric shapes (e.g., Circle, Rectangle, Triangle). Implement a `calculateArea()` method in each class and demonstrate polymorphism by calculating the area of various shapes.

Solution:

```
abstract class Shape {
    abstract public function calculateArea();
}

class Circle extends Shape {
    private $radius;

    public function __construct($radius) {
        $this->radius = $radius;
    }

    public function calculateArea() {
        return 3.14 * $this->radius * $this->radius;
    }
}

class Rectangle extends Shape {
    private $width;
    private $height;

    public function __construct($width, $height) {
        $this->width = $width;
        $this->height = $height;
    }

    public function calculateArea() {
        return $this->width * $this->height;
    }
}

class Triangle extends Shape {
    private $base;
    private $height;

    public function __construct($base, $height) {
        $this->base = $base;
        $this->height = $height;
    }

    public function calculateArea() {
        return 0.5 * $this->base * $this->height;
    }
}
```

Problem 2: Encapsulation Problem Statement: Create a `BankAccount` class with private properties (`balance`, `accountNumber`) and methods (`deposit`, `withdraw`) to encapsulate the account's details and transactions.

Solution:

```
 class BankAccount {
    private $balance = 0;
    private $accountNumber;

    public function __construct($accountNumber) {
        $this->accountNumber = $accountNumber;
    }

    public function deposit($amount) {
        $this->balance += $amount;
    }

    public function withdraw($amount) {
        if ($amount <= $this->balance) {
            $this->balance -= $amount;
        } else {
            echo "Insufficient balance.";
        }
    }
}
```

Problem 3: Abstraction Problem Statement: Create an abstract class `Vehicle` with abstract methods (`start` , `stop`) and two concrete subclasses (`Car` , `Motorcycle`) implementing these methods.

Solution:

```
 abstract class Vehicle {
    abstract public function start();
    abstract public function stop();
}

class Car extends Vehicle {
    public function start() {
        echo "Car started.\n";
    }

    public function stop() {
        echo "Car stopped.\n";
    }
}

class Motorcycle extends Vehicle {
    public function start() {
        echo "Motorcycle started.\n";
    }

    public function stop() {
        echo "Motorcycle stopped.\n";
    }
}
```

Problem 4: Inheritance and Polymorphism Problem Statement: Create a base class `Animal` with properties (`name` , `species`) and a method `speak()` . Implement two subclasses (`Dog` , `Cat`) inheriting from `Animal` and override the `speak()` method to make each animal produce a unique sound.

Solution:

```
class Animal {
    protected $name;
    protected $species;

    public function __construct($name, $species) {
        $this->name = $name;
        $this->species = $species;
    }

    public function speak() {
        return "Unknown sound";
    }
}

class Dog extends Animal {
    public function speak() {
        return "Woof!";
    }
}

class Cat extends Animal {
    public function speak() {
        return "Meow!";
    }
}
```

Problem 5: Interface and Abstraction Problem Statement: Create an interface `Drawable` with a method `draw()`. Implement this interface in two classes (`Circle`, `Rectangle`) and demonstrate how to use these objects interchangeably.

Solution:

```
interface Drawable {
    public function draw();
}

class Circle implements Drawable {
    public function draw() {
        echo "Drawing a circle.\n";
    }
}

class Rectangle implements Drawable {
    public function draw() {
        echo "Drawing a rectangle.\n";
    }
}

function drawShape(Drawable $shape) {
    $shape->draw();
}
```

Problem 6: Encapsulation and Setter-Getter Problem Statement: Create a `Person` class with private properties (`name`, `age`) and implement setter and getter methods for these properties.

Solution:

```
class Person {
    private $name;
    private $age;

    public function setName($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function setAge($age) {
        if ($age >= 0) {
            $this->age = $age;
        } else {
            echo "Invalid age.";
        }
    }

    public function getAge() {
        return $this->age;
    }
}
```

Problem 7: Polymorphism with Inheritance Problem Statement: Create a base class `Fruit` with a method `taste()` and two subclasses (`Apple` , `Banana`) that inherit from `Fruit` and override the `taste()` method to describe their taste.

Solution:

```
class Fruit {
    public function taste() {
        return "Generic fruit taste";
    }
}

class Apple extends Fruit {
    public function taste() {
        return "Sweet and crisp";
    }
}

class Banana extends Fruit {
    public function taste() {
        return "Creamy and slightly sweet";
    }
}
```

Problem 8: Abstraction and Inheritance Problem Statement: Create an abstract class `Animal` with properties (`name` , `species`) and an abstract method `makeSound()` . Implement two subclasses (`Dog` , `Cat`) inheriting from `Animal` and implement the `makeSound()` method to make each animal produce a unique sound.

Solution:

```
abstract class Animal {
    protected $name;
    protected $species;

    public function __construct($name, $species) {
        $this->name = $name;
        $this->species = $species;
    }

    abstract public function makeSound();
}

class Dog extends Animal {
    public function makeSound() {
        return "Woof!";
    }
}

class Cat extends Animal {
    public function makeSound() {
        return "Meow!";
    }
}
```

Problem 9: Interface and Polymorphism Problem Statement: Create an interface `Playable` with methods (`play`, `pause`, `stop`). Implement this interface in two classes (`MusicPlayer`, `VideoPlayer`) and demonstrate polymorphism by using the same interface to control both players.

Solution:

```
interface Playable {
    public function play();
    public function pause();
    public function stop();
}

class Music

Player implements Playable {
    public function play() {
        echo "Music playing.\n";
    }

    public function pause() {
        echo "Music paused.\n";
    }

    public function stop() {
        echo "Music stopped.\n";
    }
}

class VideoPlayer implements Playable {
    public function play() {
        echo "Video playing.\n";
    }

    public function pause() {
        echo "Video paused.\n";
    }

    public function stop() {
        echo "Video stopped.\n";
    }
}
```

Problem 10: Abstraction and Inheritance Problem Statement: Create an abstract class `Employee` with properties (`name`, `salary`) and an abstract method `calculateBonus()`. Implement two subclasses (`Manager`, `Developer`) inheriting from `Employee` and implement the `calculateBonus()` method differently for each.

Solution:

```
abstract class Employee {
    protected $name;
    protected $salary;

    public function __construct($name, $salary) {
        $this->name = $name;
        $this->salary = $salary;
    }

    abstract public function calculateBonus();
}

class Manager extends Employee {
    public function calculateBonus() {
        return $this->salary * 0.2;
    }
}

class Developer extends Employee {
    public function calculateBonus() {
        return $this->salary * 0.1;
    }
}
```

Problem 11: Interface and Abstraction Problem Statement: Create an interface `Drawable` with a method `draw()`. Implement this interface in two classes (`Circle`, `Rectangle`) and demonstrate how to use these objects interchangeably with a generic drawing function.

Solution:

```
interface Drawable {
    public function draw();
}

class Circle implements Drawable {
    public function draw() {
        echo "Drawing a circle.\n";
    }
}

class Rectangle implements Drawable {
    public function draw() {
        echo "Drawing a rectangle.\n";
    }
}

function drawShape(Drawable $shape) {
    $shape->draw();
}
```

Problem 12: Encapsulation and Getter Problem Statement: Create a `Product` class with private properties (`name`, `price`) and implement a `getInfo()` method that returns a formatted string with the product's name and price.

Solution:

```
class Product {
    private $name;
    private $price;

    public function __construct($name, $price) {
        $this->name = $name;
        $this->price = $price;
    }

    public function getInfo() {
        return "Product: {$this->name}, Price: {$this->price}";
    }
}
```

Problem 13: Polymorphism with Inheritance Problem Statement: Create a base class `Shape` with a method `draw()` and two subclasses (`Circle`, `Rectangle`) that inherit from `Shape` and override the `draw()` method to describe how to draw each shape.

Solution:

```
class Shape {
    public function draw() {
        return "Drawing a generic shape";
    }
}

class Circle extends Shape {
    public function draw() {
        return "Drawing a circle";
    }
}

class Rectangle extends Shape {
    public function draw() {
        return "Drawing a rectangle";
    }
}
```

Problem 14: Abstraction and Inheritance Problem Statement: Create an abstract class `Document` with properties (`title`, `content`) and an abstract method `generateHTML()`. Implement two subclasses (`Article`, `Video`) inheriting from `Document` and implement the `generateHTML()` method differently for each.

Solution:

```
abstract class Document {
    protected $title;
    protected $content;

    public function __construct($title, $content) {
        $this->title = $title;
        $this->content = $content;
    }

    abstract public function generateHTML();
}

class Article extends Document {
    public function generateHTML() {
        return "<h1>{$this->title}</h1><p>{$this->content}</p>";
    }
}

class Video extends Document {
    public function generateHTML() {
        return "<iframe src='{$this->content}'></iframe>";
    }
}
```

Problem 15: Interface and Polymorphism Problem Statement: Create an interface `Logger` with a method `log($message)`. Implement this interface in two classes (`FileLogger`, `DatabaseLogger`) and demonstrate how to use these loggers interchangeably.

Solution:

```
interface Logger {
    public function log($message);
}

class FileLogger implements Logger {
    public function log($message) {
        file_put_contents('log.txt', $message . "\n", FILE_APPEND);
    }
}

class DatabaseLogger implements Logger {
    public function log($message) {
        // Log to the database...
    }
}

function logMessage(Logger $logger, $message) {
    $logger->log($message);
}
```

Problem 16: Encapsulation and Setter-Getter Problem Statement: Create a `Student` class with private properties (`name`, `age`) and implement setter and getter methods for these properties.

Solution:

```
class Student {
    private $name;
    private $age;

    public function setName($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function setAge($age) {
        if ($age >= 0) {
            $this->age = $age;
        } else {
            echo "Invalid age.";
        }
    }

    public function getAge() {
        return $this->age;
    }
}
```

Problem 17: Polymorphism with Inheritance Problem Statement: Create a base class `Vehicle` with a method `drive()` and two subclasses (`Car`, `Motorcycle`) that inherit from `Vehicle` and override the `drive()` method to describe how each vehicle drives.

Solution:

```
class Vehicle {
    public function drive() {
        return "Driving a generic vehicle";
    }
}

class Car extends Vehicle {
    public function drive() {
        return "Driving a car";
    }
}

class Motorcycle extends Vehicle {
    public function drive() {
        return "Riding a motorcycle";
    }
}
```

Problem 18: Abstraction and Inheritance Problem Statement: Create an abstract class `Employee` with properties (`name`, `salary`) and an abstract method `calculateBonus()`. Implement two subclasses (`Manager

`, ` Developer) `inheriting from ` Employee `and implement the` calculateBonus()` method differently for each.

Solution:

```
abstract class Employee {
    protected $name;
    protected $salary;

    public function __construct($name, $salary) {
        $this->name = $name;
        $this->salary = $salary;
    }

    abstract public function calculateBonus();
}

class Manager extends Employee {
    public function calculateBonus() {
        return $this->salary * 0.2;
    }
}

class Developer extends Employee {
    public function calculateBonus() {
        return $this->salary * 0.1;
    }
}
```

Problem 19: Interface and Polymorphism Problem Statement: Create an interface `Playable` with methods (`play`, `pause`, `stop`). Implement this interface in two classes (`MusicPlayer`, `VideoPlayer`) and demonstrate polymorphism by using the same interface to control both players.

Solution:

```
interface Playable {
    public function play();
    public function pause();
    public function stop();
}

class MusicPlayer implements Playable {
    public function play() {
        echo "Music playing.\n";
    }

    public function pause() {
        echo "Music paused.\n";
    }

    public function stop() {
        echo "Music stopped.\n";
    }
}

class VideoPlayer implements Playable {
    public function play() {
        echo "Video playing.\n";
    }

    public function pause() {
        echo "Video paused.\n";
    }

    public function stop() {
        echo "Video stopped.\n";
    }
}
```

Problem 20: Encapsulation and Getter-Setter Problem Statement: Create a `Customer` class with private properties (`name` , `email`) and implement getter and setter methods for these properties.

Solution:

```
class Customer {
    private $name;
    private $email;

    public function setName($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function setEmail($email) {
        $this->email = $email;
    }

    public function getEmail() {
        return $this->email;
    }
}
```

Problem 21: Polymorphism with Inheritance Problem Statement: Create a base class `Animal` with a method `makeSound()` and two subclasses (`Dog` , `Cat`) that inherit from `Animal` and override the `makeSound()` method to describe their sound.

Solution:

```
class Animal {
    public function makeSound() {
        return "Generic animal sound";
    }
}

class Dog extends Animal {
    public function makeSound() {
        return "Woof!";
    }
}

class Cat extends Animal {
    public function makeSound() {
        return "Meow!";
    }
}
```

Problem 22: Abstraction and Inheritance Problem Statement: Create an abstract class `Document` with properties (`title`, `content`) and an abstract method `generateHTML()`. Implement two subclasses (`Article`, `Video`) inheriting from `Document` and implement the `generateHTML()` method differently for each.

Solution:

```
abstract class Document {
    protected $title;
    protected $content;

    public function __construct($title, $content) {
        $this->title = $title;
        $this->content = $content;
    }

    abstract public function generateHTML();
}

class Article extends Document {
    public function generateHTML() {
        return "<h1>{$this->title}</h1><p>{$this->content}</p>";
    }
}

class Video extends Document {
    public function generateHTML() {
        return "<iframe src='{$this->content}'></iframe>";
    }
}
```

Problem 23: Interface and Polymorphism Problem Statement: Create an interface `Logger` with a method `log($message)`. Implement this interface in two classes (`FileLogger`, `DatabaseLogger`) and demonstrate how to use these loggers interchangeably.

Solution:

```
interface Logger {
    public function log($message);
}

class FileLogger implements Logger {
    public function log($message) {
        file_put_contents('log.txt', $message . "\n", FILE_APPEND);
    }
}

class DatabaseLogger implements Logger {
    public function log($message) {
        // Log to the database...
    }
}

function logMessage(Logger $logger, $message) {
    $logger->log($message);
}
```

Problem 24: Encapsulation and Getter-Setter Problem Statement: Create a `Book` class with private properties (`title`, `author`) and implement getter and setter methods for these properties.

Solution:

```
class Book {
    private $title;
    private $author;

    public function setTitle($title) {
        $this->title = $title;
    }

    public function getTitle() {
        return $this->title;
    }

    public function setAuthor($author) {
        $this->author = $author;
    }

    public function getAuthor() {
        return $this->author;
    }
}
```

Problem 25: Polymorphism with Inheritance Problem Statement: Create a base class `Shape` with a method `draw()` and two subclasses (`Circle`, `Rectangle`) that inherit from `Shape` and override the `draw()` method to describe how to draw each shape.

Solution:

```
class Shape {
    public function draw() {
        return "Drawing a generic shape";
    }
}

class Circle extends Shape {
    public function draw() {
        return "Drawing a circle";
    }
}

class Rectangle extends Shape {
    public function draw() {
        return "Drawing a rectangle";
    }
}
```

Problem 26: Abstraction and Inheritance Problem Statement: Create an abstract class `Employee` with properties (`name`, `salary`) and an abstract method `calculateBonus()`. Implement two subclasses (`Manager`, `Developer`) inheriting from `Employee` and implement the `calculateBonus()` method differently for each.

Solution:

```
abstract class Employee {
    protected $name;
    protected $salary;

    public function __construct($name, $salary) {
        $this->name = $name;
        $this->salary = $salary;
    }

    abstract public function calculateBonus();
}

class Manager extends Employee {
    public function calculateBonus() {
        return $this->salary * 0.2;
    }
}

class Developer extends Employee {
    public function calculateBonus() {
        return $this->salary * 0.1;
    }
}
```

Problem 27: Interface and Polymorphism Problem Statement: Create an interface `Playable` with methods (`play`, `pause`, `stop`). Implement this interface in two classes (`MusicPlayer`, `VideoPlayer`) and demonstrate polymorphism by using the same interface to control both players.

Solution:

```
interface Playable {
    public function play();
    public function pause();
    public function stop();
}

class MusicPlayer implements Playable {
    public function play() {
        echo "Music playing.\n";
    }

    public function pause() {


        echo "Music paused.\n";
    }

    public function stop() {
        echo "Music stopped.\n";
    }
}

class VideoPlayer implements Playable {
    public function play() {
        echo "Video playing.\n";
    }

    public function pause() {
        echo "Video paused.\n";
    }

    public function stop() {
        echo "Video stopped.\n";
    }
}
```

Problem 28: Encapsulation and Getter-Setter Problem Statement: Create a `Customer` class with private properties (`name` , `email`) and implement getter and setter methods for these properties.

Solution:

```
class Customer {
    private $name;
    private $email;

    public function setName($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function setEmail($email) {
        $this->email = $email;
    }

    public function getEmail() {
        return $this->email;
    }
}
```

Problem 29: Polymorphism with Inheritance Problem Statement: Create a base class `Animal` with a method `makeSound()` and two subclasses (`Dog` , `Cat`) that inherit from `Animal` and override the `makeSound()` method to describe their sound.

Solution:

```
class Animal {
    public function makeSound() {
        return "Generic animal sound";
    }
}

class Dog extends Animal {
    public function makeSound() {
        return "Woof!";
    }
}

class Cat extends Animal {
    public function makeSound() {
        return "Meow!";
    }
}
```

Problem 30: Abstraction and Inheritance Problem Statement: Create an abstract class `Document` with properties (`title`, `content`) and an abstract method `generateHTML()`. Implement two subclasses (`Article`, `Video`) inheriting from `Document` and implement the `generateHTML()` method differently for each.

Solution:

```
abstract class Document {
    protected $title;
    protected $content;

    public function __construct($title, $content) {
        $this->title = $title;
        $this->content = $content;
    }

    abstract public function generateHTML();
}

class Article extends Document {
    public function generateHTML() {
        return "<h1>{$this->title}</h1><p>{$this->content}</p>";
    }
}

class Video extends Document {
    public function generateHTML() {
        return "<iframe src='{$this->content}'></iframe>";
    }
}
```

OOP : More Problem

Problem 1: Singleton Pattern Problem Statement: Implement the Singleton design pattern in PHP to ensure that a class has only one instance and provide a global point of access to it.

Solution:

```
 class Singleton {
     private static $instance;

     private function __construct() {
         // Private constructor to prevent direct instantiation
     }

     public static function getInstance() {
         if (self::$instance === null) {
             self::$instance = new self();
         }
         return self::$instance;
     }
}
```

Problem 2: Factory Method Pattern Problem Statement: Implement the Factory Method design pattern in PHP to create objects of different types based on a specified condition.

Solution:

```
 interface Product {
    public function getName();
}

class ConcreteProductA implements Product {
    public function getName() {
        return "Product A";
    }
}

class ConcreteProductB implements Product {
    public function getName() {
        return "Product B";
    }
}

interface Factory {
    public function createProduct();
}

class ConcreteFactoryA implements Factory {
    public function createProduct() {
        return new ConcreteProductA();
    }
}

class ConcreteFactoryB implements Factory {
    public function createProduct() {
        return new ConcreteProductB();
    }
}
```

Problem 3: Dependency Injection Problem Statement: Implement dependency injection in PHP to inject a database connection object into a `UserRepository` class.

Solution:

```
class DatabaseConnection {
    // Database connection details here
}

class UserRepository {
    private $db;

    public function __construct(DatabaseConnection $db) {
        $this->db = $db;
    }

    public function getUserById($id) {
        // Use $this->db to fetch user data
    }
}
```

Problem 4: Observer Pattern Problem Statement: Implement the Observer design pattern in PHP. Create a `Subject` class that maintains a list of observers and notifies them when its state changes.

Solution:

```
interface Observer {
    public function update($data);
}

class Subject {
    private $observers = [];

    public function addObserver(Observer $observer) {
        $this->observers[] = $observer;
    }

    public function setState($data) {
        // Change the state
        $this->notifyObservers($data);
    }

    private function notifyObservers($data) {
        foreach ($this->observers as $observer) {
            $observer->update($data);
        }
    }
}
```

Problem 5: Strategy Pattern Problem Statement: Implement the Strategy design pattern in PHP to define a family of algorithms, encapsulate each one, and make them interchangeable.

Solution:

```
interface PaymentStrategy {
    public function pay($amount);
}

class CreditCardPayment implements PaymentStrategy {
    public function pay($amount) {
        // Implement credit card payment logic
    }
}

class PayPalPayment implements PaymentStrategy {
    public function pay($amount) {
        // Implement PayPal payment logic
    }
}

class ShoppingCart {
    private $paymentStrategy;

    public function setPaymentStrategy(PaymentStrategy $paymentStrategy) {
        $this->paymentStrategy = $paymentStrategy;
    }

    public function checkout($amount) {
        $this->paymentStrategy->pay($amount);
    }
}
```

Problem 6: Decorator Pattern Problem Statement: Implement the Decorator design pattern in PHP to add behavior to individual objects, either statically or dynamically, without affecting their behavior of other objects from the same class.

Solution:

```
interface Coffee {
    public function cost();
}

class SimpleCoffee implements Coffee {
    public function cost() {
        return 5; // Base cost of simple coffee
    }
}

class MilkDecorator implements Coffee {
    private $coffee;

    public function __construct(Coffee $coffee) {
        $this->coffee = $coffee;
    }

    public function cost() {
        return $this->coffee->cost() + 2; // Cost of milk
    }
}

class SugarDecorator implements Coffee {
    private $coffee;

    public function __construct(Coffee $coffee) {
        $this->coffee = $coffee;
    }

    public function cost() {
        return $this->coffee->cost() + 1; // Cost of sugar
    }
}
```

Problem 7: Template Method Pattern Problem Statement: Implement the Template Method design pattern in PHP to define the skeleton of an algorithm in the base class and let subclasses override specific steps of the algorithm without changing its structure.

Solution:

```
abstract class ReportTemplate {
    public function generateReport() {
        $this->openFile();
        $this->writeHeader();
        $this->writeContent();
        $this->writeFooter();
        $this->closeFile();
    }

    abstract protected function openFile();
    abstract protected function writeHeader();
    abstract protected function writeContent();
    abstract protected function writeFooter();
    abstract protected function closeFile();
}

class PDFReport extends ReportTemplate {
    protected function openFile() {
        // Open PDF file
    }

    protected function writeHeader() {
        // Write PDF header
    }

    protected function writeContent() {
        // Write PDF content
    }

    protected function writeFooter() {
        // Write PDF footer
    }

    protected function closeFile() {
        // Close PDF file
    }
}

class ExcelReport extends ReportTemplate {
    protected function openFile() {
        // Open Excel file
    }

    protected function writeHeader() {
        // Write Excel header
    }

    protected function writeContent() {
        // Write Excel content
    }

    protected function writeFooter() {
        // Write Excel footer
    }

    protected function closeFile() {
        // Close Excel file
    }
}
```

Problem 8: Command Pattern Problem Statement: Implement the Command design pattern in PHP to encapsulate a request as an object, thereby allowing for parameterization of clients with queues, requests, and operations.

Solution:

```
 interface Command {
    public function execute();
}

class Light {
    public function turnOn() {
        // Turn on the light
    }

    public function turnOff() {
        // Turn off the light
    }
}

class LightOnCommand implements Command {
    private $light;

    public function __construct(Light $light) {
        $this->light = $light;
    }

    public function execute() {
        $this->light->turnOn();
    }
}

class LightOffCommand implements Command {
    private $light;

    public function __construct(Light $light) {
        $this->light = $light;
    }

    public function execute() {
        $this->light->turnOff();
    }
}

class RemoteControl {
    private $commands = [];

    public function setCommand($slot, Command $command) {
        $this->commands[$slot] = $command;
    }

    public function pressButton($slot) {
        if (isset($this->commands[$slot])) {
            $this->commands[$slot]->execute();
        }
    }
}
```

**Problem 9:

Proxy Pattern** **Problem Statement:** Implement the Proxy design pattern in PHP to provide a surrogate or placeholder for another object to control access to it.

Solution:

```
interface Image {
    public function display();
}

class RealImage implements Image {
    private $filename;

    public function __construct($filename) {
        $this->filename = $filename;
        $this->loadImageFromDisk();
    }

    private function loadImageFromDisk() {
        // Load image data from disk
    }

    public function display() {
        // Display the loaded image
    }
}

class ProxyImage implements Image {
    private $realImage;
    private $filename;

    public function __construct($filename) {
        $this->filename = $filename;
    }

    public function display() {
        if ($this->realImage === null) {
            $this->realImage = new RealImage($this->filename);
        }
        $this->realImage->display();
    }
}
```

Problem 10: Chain of Responsibility Pattern Problem Statement: Implement the Chain of Responsibility design pattern in PHP to pass a request along a chain of handlers, each processing the request or passing it to the next handler in the chain.

Solution:

```
abstract class Handler {
    protected $successor;

    public function setSuccessor(Handler $successor) {
        $this->successor = $successor;
    }

    abstract public function handleRequest($request);
}

class ConcreteHandlerA extends Handler {
    public function handleRequest($request) {
        if ($request === 'A') {
            // Handle the request
        } elseif ($this->successor !== null) {
            $this->successor->handleRequest($request);
        }
    }
}

class ConcreteHandlerB extends Handler {
    public function handleRequest($request) {
        if ($request === 'B') {
            // Handle the request
        } elseif ($this->successor !== null) {
            $this->successor->handleRequest($request);
        }
    }
}

class ConcreteHandlerC extends Handler {
    public function handleRequest($request) {
        if ($request === 'C') {
            // Handle the request
        } elseif ($this->successor !== null) {
            $this->successor->handleRequest($request);
        }
    }
}
```

OOPS : nterfaces, traits, namespaces, and iterables

Sure, here are 10 professional and real-life problems related to PHP OOP concepts such as interfaces, traits, namespaces, and iterables, along with detailed explanations and code examples:

Problem 1: Creating an Interface Problem Statement: Create an interface `Shape` with a method `calculateArea()` that must be implemented by classes representing different shapes.

Solution:

```
interface Shape {
    public function calculateArea();
}

class Circle implements Shape {
    private $radius;

    public function __construct($radius) {
        $this->radius = $radius;
    }

    public function calculateArea() {
        return 3.14 * $this->radius * $this->radius;
    }
}

class Rectangle implements Shape {
    private $length;
    private $width;

    public function __construct($length, $width) {
        $this->length = $length;
        $this->width = $width;
    }

    public function calculateArea() {
        return $this->length * $this->width;
    }
}
```

Problem 2: Using Traits for Reusability Problem Statement: Create a trait `Loggable` with a method `log($message)` that can be used in multiple classes to log messages.

Solution:

```
trait Loggable {
    public function log($message) {
        echo "Log: $message\n";
    }
}

class User {
    use Loggable;
    // Other user-related methods and properties
}

class Product {
    use Loggable;
    // Other product-related methods and properties
}
```

Problem 3: Defining a Namespace Problem Statement: Define a namespace `MyApp` and create a class `Logger` inside this namespace. Instantiate the `Logger` class outside the namespace.

Solution:

```
namespace MyApp;

class Logger {
    public function log($message) {
        echo "Log: $message\n";
    }
}

// Instantiate the Logger class
$logger = new Logger();
```

Problem 4: Using Multiple Traits Problem Statement: Create two traits, `Loggable` and `Serializable`, and use them in a class `Item` to provide logging and serialization capabilities.

Solution:

```
trait Loggable {
    public function log($message) {
        echo "Log: $message\n";
    }
}

trait Serializable {
    public function serialize() {
        return json_encode($this);
    }
}

class Item {
    use Loggable, Serializable;
    // Other item-related methods and properties
}
```

Problem 5: Implementing an Iterator Problem Statement: Create a class `MyIterator` that implements the `Iterator` interface. It should be able to iterate over an array of values.

Solution:

```
class MyIterator implements Iterator {
    private $position = 0;
    private $data = [];

    public function __construct($data) {
        $this->data = $data;
    }

    public function current() {
        return $this->data[$this->position];
    }

    public function key() {
        return $this->position;
    }

    public function next() {
        ++$this->position;
    }

    public function rewind() {
        $this->position = 0;
    }

    public function valid() {
        return isset($this->data[$this->position]);
    }
}

$data = [1, 2, 3, 4, 5];
$iterator = new MyIterator($data);

foreach ($iterator as $key => $value) {
    echo "Key: $key, Value: $value\n";
}
```

Problem 6: Using Namespaces in a Real-Life Scenario Problem Statement: Create two classes, `User` and `Product`, in separate namespaces, and demonstrate how to use them together.

Solution:

```
namespace MyApp\User;

class User {
    public function getName() {
        return "John Doe";
    }
}

namespace MyApp\Product;

class Product {
    public function getProductName() {
        return "Sample Product";
    }
}

// Usage
$user = new \MyApp\User\User();
$product = new \MyApp\Product\Product();

echo "User: " . $user->getName() . "\n";
echo "Product: " . $product->getProductName() . "\n";
```

Problem 7: Combining Traits and Interfaces Problem Statement: Create a trait `Loggable` with a method `log($message)` and an interface `Serializable` with a method `serialize()`. Create a class `Item` that uses both the trait and the interface.

Solution:

```
trait Loggable {
    public function log($message) {
        echo "Log: $message\n";
    }
}

interface Serializable {
    public function serialize();
}

class Item implements Serializable {
    use Loggable;

    public function serialize() {
        return json_encode($this);
    }
}
```

Problem 8: Namespace Alias Problem Statement: Use a namespace alias to simplify the use of a class from a deeply nested namespace.

Solution:

```
use MyApp\Product\Some\Very\Long\Namespace\Product as ShortProduct;

$product = new ShortProduct();
```

Problem 9: Using Iterables in Functions Problem Statement: Create a function `calculateSum` that accepts an iterable of numbers and returns their sum.

Solution:

```
function calculateSum(iterable $numbers) {
    $sum = 0;


    foreach ($numbers as $number) {
        $sum += $number;
    }
    return $sum;
}

$data = [1, 2, 3, 4, 5];
$total = calculateSum($data);
echo "Sum: $total\n";
```

Problem 10: Namespace and Class Autoloading Problem Statement: Set up an autoloading mechanism using namespaces and demonstrate how it can automatically load a class.

Solution:

```
spl_autoload_register(function ($class) {
    // Convert namespace separator to directory separator
    $classPath = str_replace("\\", DIRECTORY_SEPARATOR, $class);

    // Include the class file
    include_once $classPath . '.php';
});

// Now you can use classes without explicitly including them
$user = new \MyApp\User\User();
```

OOPS With static & final methods

Problem 1: Static Method and Property Problem Statement: Create a class `Counter` with a static property `count` and a static method `increment()` to increment the count. Demonstrate how to use the static property and method.

Solution:

```
class Counter {
    public static $count = 0;

    public static function increment() {
        self::$count++;
    }
}

// Usage
Counter::increment();
echo Counter::$count; // Output: 1
```

Problem 2: Static Factory Method Problem Statement: Create a class `Logger` with a static factory method `createLogger($type)` that returns an instance of a logger based on the provided type (e.g., "file" or "database").

Solution:

```
class Logger {
    public static function createLogger($type) {
        if ($type === 'file') {
            return new FileLogger();
        } elseif ($type === 'database') {
            return new DatabaseLogger();
        }
    }
}

// Usage
$fileLogger = Logger::createLogger('file');
$databaseLogger = Logger::createLogger('database');
```

Problem 3: Static Factory Method and Singleton Problem Statement: Enhance the previous problem by making the logger instances singleton using static methods.

Solution:

```
class Logger {
    private static $fileLoggerInstance;
    private static $databaseLoggerInstance;

    public static function getFileLogger() {
        if (self::$fileLoggerInstance === null) {
            self::$fileLoggerInstance = new FileLogger();
        }
        return self::$fileLoggerInstance;
    }

    public static function getDatabaseLogger() {
        if (self::$databaseLoggerInstance === null) {
            self::$databaseLoggerInstance = new DatabaseLogger();
        }
        return self::$databaseLoggerInstance;
    }
}

// Usage
$fileLogger = Logger::getFileLogger(); // Returns the same instance on subsequent calls
$databaseLogger = Logger::getDatabaseLogger(); // Returns the same instance on subsequent calls
```

Problem 4: Final Method Problem Statement: Create a class `Shape` with a final method `calculateArea()` that cannot be overridden by subclasses. Implement two subclasses (`Circle` , `Rectangle`) and use the final method.

Solution:

```
class Shape {
    final public function calculateArea() {
        // Common area calculation logic
    }
}

class Circle extends Shape {
    // Cannot override calculateArea()
}

class Rectangle extends Shape {
    // Cannot override calculateArea()
}
```

Problem 5: Static Property and Method with Inheritance Problem Statement: Create a base class `Vehicle` with a static property `totalCount` to keep track of the total number of vehicles created. Implement two subclasses (`Car` , `Motorcycle`) that inherit from `Vehicle` and use the static property to count instances.

Solution:

```
class Vehicle {
    public static $totalCount = 0;

    public function __construct() {
        self::$totalCount++;
    }
}

class Car extends Vehicle {
    // Inherits static property $totalCount
}

class Motorcycle extends Vehicle {
    // Inherits static property $totalCount
}

// Usage
$car1 = new Car();
$car2 = new Car();
$motorcycle1 = new Motorcycle();

echo Vehicle::$totalCount; // Output: 3 (total number of vehicles)
```

Problem 6: Static Method with Inheritance Problem Statement: Create a base class `MathOperation` with a static method `add($a, $b)` that adds two numbers. Implement two subclasses (`Addition` , `Subtraction`) that inherit from `MathOperation` and use the static method.

Solution:

```
class MathOperation {
    public static function add($a, $b) {
        return $a + $b;
    }
}

class Addition extends MathOperation {
    // Inherits static method add()
}

class Subtraction extends MathOperation {
    // Inherits static method add()
}

// Usage
$result1 = Addition::add(5, 3); // Output: 8
$result2 = Subtraction::add(10, 2); // Output: 12
```

Problem 7: Static Property and Method with Abstract Class Problem Statement: Create an abstract class `Product` with a static property `totalProducts` to keep track of the total number of products created. Implement two subclasses (`Book` , `Toy`) that inherit from `Product` and use the static property to count instances.

Solution:

```
abstract class Product {
    public static $totalProducts = 0;

    public function __construct() {
        self::$totalProducts++;
    }
}

class Book extends Product {
    // Inherits static property $totalProducts
}

class Toy extends Product {
    // Inherits static property $totalProducts
}

// Usage
$book1 = new Book();
$toy1 = new Toy();
$toy2 = new Toy();

echo Product::$totalProducts; // Output: 3 (total number of products)
```

Problem 8: Static Method with Interface Problem Statement: Create an interface `Shape` with a static method `describe()` that provides a description of shapes. Implement two classes (`Circle` , `Rectangle`) that implement the `Shape` interface and use the static method.

Solution:

```
interface Shape {
    public static function describe();
}

class Circle implements Shape {
    public static function describe() {
        return "This is a circle.";
    }
}

class Rectangle implements Shape {
    public static function describe() {
        return "This is a rectangle.";
    }
}

// Usage
echo Circle::describe(); // Output: "This is a circle."
echo Rectangle::describe(); // Output: "This is a rectangle."
```

Problem 9: Final Class Problem Statement: Create a final class `Utility` that cannot be extended. Demonstrate the use of a final class.

Solution:

```
final class Utility {
    public static function doSomething() {
        // Utility methods
    }
}

// Cannot extend Utility class
// class CustomUtility extends Utility {}

// Usage
Utility::doSomething();
```

Problem 10: Static Property and Method with Final Class Problem Statement: Create a final class `Config` with a static property `settings` to store configuration settings and a static method `get($key)` to retrieve a specific setting. Demonstrate the use of a final class with static properties and methods.

Solution:

```
final class Config {
    private static $settings = [];

    public static function get($key) {
        return isset(self::$settings[$key]) ? self::$settings[$key] : null;
    }

    public static function set($key, $value) {
        self::$settings[$key] = $value;
    }
}

// Usage
Config::set('siteName', 'My Website');
echo Config::get('siteName'); // Output: "My Website"
```

PHP WITH MYSQL : 3 TYPES :- MySQL. MySQLi. PDO

PHP with a MySQL database using three different methods: MySQL, MySQLi, and PDO. These problems are designed to help PHP developers prepare for job interviews and cover various aspects of database connectivity. Each problem includes a detailed explanation and a code example.

STEP : 1

Problem 1: Basic Connection (MySQL) *Problem*: Write a PHP script to establish a basic MySQL database connection.

Solution:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysql_connect($servername, $username, $password);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysql_error());
}
echo "Connected successfully";
mysql_close($conn);
?>
```

Problem 2: Basic Connection (MySQLi) *Problem*: Write a PHP script to establish a basic MySQLi database connection.

Solution:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
$conn->close();
?>
```

Problem 3: Basic Connection (PDO) *Problem*: Write a PHP script to establish a basic PDO database connection.

Solution:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
    $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
}
catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
?>
```

Problem 4: Execute a Query (MySQL) *Problem*: Write a PHP script to execute a basic SELECT query using MySQL.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = mysql_connect($servername, $username, $password);

if (!$conn) {
    die("Connection failed: " . mysql_error());
}

mysql_select_db($dbname, $conn);

$sql = "SELECT * FROM users";
$result = mysql_query($sql);

if (mysql_num_rows($result) > 0) {
    while ($row = mysql_fetch_assoc($result)) {
        echo "Name: " . $row["name"]. " - Email: " . $row["email"]. "<br>";
    }
} else {
    echo "No results found";
}

mysql_close($conn);
?>
```

Problem 5: Execute a Query (MySQLi) *Problem*: Write a PHP script to execute a basic SELECT query using MySQLi.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT * FROM users";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "Name: " . $row["name"]. " - Email: " . $row["email"]. "<br>";
    }
} else {
    echo "No results found";
}

$conn->close();
?>
```

Problem 6: Execute a Query (PDO) *Problem*: Write a PHP script to execute a basic SELECT query using PDO.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql = "SELECT * FROM users";
    $stmt = $conn->prepare($sql);
    $stmt->execute();

    $result = $stmt->fetchAll(PDO::FETCH_ASSOC);

    if (count($result) > 0) {
        foreach ($result as $row) {
            echo "Name: " . $row["name"]. " - Email: " . $row["email"]. "<br>";
        }
    } else {
        echo "No results found";
    }
} catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
?>
```

STEP 2 :

Here are the next set of problems and solutions related to connecting PHP with a MySQL database using MySQL, MySQLi, and PDO.

Problem 7: Insert Data (MySQL) *Problem*: Write a PHP script to insert data into a MySQL database using the MySQL method.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = mysql_connect($servername, $username, $password);

if (!$conn) {
    die("Connection failed: " . mysql_error());
}

mysql_select_db($dbname, $conn);

$name = "John";
$email = "john@example.com";

$sql = "INSERT INTO users (name, email) VALUES ('$name', '$email')";

if (mysql_query($sql)) {
    echo "Record inserted successfully";
} else {
    echo "Error: " . mysql_error();
}

mysql_close($conn);
?>
```

Problem 8: Insert Data (MySQLi) *Problem*: Write a PHP script to insert data into a MySQL database using the MySQLi method.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$name = "John";
$email = "john@example.com";

$sql = "INSERT INTO users (name, email) VALUES (?, ?)";
$stmt = $conn->prepare($sql);
$stmt->bind_param("ss", $name, $email);

if ($stmt->execute()) {
    echo "Record inserted successfully";
} else {
    echo "Error: " . $stmt->error;
}

$conn->close();
?>
```

Problem 9: Insert Data (PDO) *Problem*: Write a PHP script to insert data into a MySQL database using the PDO method.

Solution:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $name = "John";
    $email = "john@example.com";

    $sql = "INSERT INTO users (name, email) VALUES (:name, :email)";
    $stmt = $conn->prepare($sql);

    $stmt->bindParam(':name', $name);
    $stmt->bindParam(':email', $email);

    if ($stmt->execute()) {
        echo "Record inserted successfully";
    } else {
        echo "Error: " . $stmt->errorInfo()[2];
    }
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
?>
```

Problem 10: Update Data (MySQL) *Problem*: Write a PHP script to update data in a MySQL database using the MySQL method.

Solution:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = mysql_connect($servername, $username, $password);

if (!$conn) {
    die("Connection failed: " . mysql_error());
}

mysql_select_db($dbname, $conn);

$newEmail = "new_email@example.com";
$id = 1;

$sql = "UPDATE users SET email='$newEmail' WHERE id=$id";

if (mysql_query($sql)) {
    echo "Record updated successfully";
} else {
    echo "Error: " . mysql_error();
}

mysql_close($conn);
?>
```

Problem 11: Update Data (MySQLi) *Problem*: Write a PHP script to update data in a MySQL database using the MySQLi method.

Solution:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$newEmail = "new_email@example.com";
$id = 1;

$sql = "UPDATE users SET email=? WHERE id=?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("si", $newEmail, $id);

if ($stmt->execute()) {
    echo "Record updated successfully";
} else {
    echo "Error: " . $stmt->error;
}

$conn->close();
?>
```

Problem 12: Update Data (PDO) *Problem*: Write a PHP script to update data in a MySQL database using the PDO method.

Solution:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $newEmail = "new_email@example.com";
    $id = 1;

    $sql = "UPDATE users SET email=:email WHERE id=:id";
    $stmt = $conn->prepare($sql);

    $stmt->bindParam(':email', $newEmail);
    $stmt->bindParam(':id', $id);

    if ($stmt->execute()) {
        echo "Record updated successfully";
    } else {
        echo "Error: " . $stmt->errorInfo()[2];
    }
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
?>
```

STEP 3 :

Here are the next set of problems and solutions related to connecting PHP with a MySQL database using MySQL, MySQLi, and PDO.

Problem 13: Delete Data (MySQL) *Problem*: Write a PHP script to delete data from a MySQL database using the MySQL method.

Solution:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = mysql_connect($servername, $username, $password);

if (!$conn) {
    die("Connection failed: " . mysql_error());
}

mysql_select_db($dbname, $conn);

$id = 1;

$sql = "DELETE FROM users WHERE id=$id";

if (mysql_query($sql)) {
    echo "Record deleted successfully";
} else {
    echo "Error: " . mysql_error();
}

mysql_close($conn);
?>
```

Problem 14: Delete Data (MySQLi) *Problem*: Write a PHP script to delete data from a MySQL database using the MySQLi method.

Solution:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$id = 1;

$sql = "DELETE FROM users WHERE id=?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("i", $id);

if ($stmt->execute()) {
    echo "Record deleted successfully";
} else {
    echo "Error: " . $stmt->error;
}

$conn->close();
?>
```

Problem 15: Delete Data (PDO) *Problem*: Write a PHP script to delete data from a MySQL database using the PDO method.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $id = 1;

    $sql = "DELETE FROM users WHERE id=:id";
    $stmt = $conn->prepare($sql);

    $stmt->bindParam(':id', $id);

    if ($stmt->execute()) {
        echo "Record deleted successfully";
    } else {
        echo "Error: " . $stmt->errorInfo()[2];
    }
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
?>
```

Problem 16: Error Handling (MySQL) *Problem*: Write a PHP script to handle errors when connecting to a MySQL database using the MySQL method.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

$conn = mysql_connect($servername, $username, $password);

if (!$conn) {
    die("Connection failed: " . mysql_error());
} else {
    echo "Connected successfully";
}

mysql_close($conn);
?>
```

Problem 17: Error Handling (MySQLi) *Problem*: Write a PHP script to handle errors when connecting to a MySQL database using the MySQLi method.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

$conn = new mysqli($servername, $username, $password);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
} else {
    echo "Connected successfully";
}

$conn->close();
?>
```

Problem 18: Error Handling (PDO) *Problem*: Write a PHP script to handle errors when connecting to a MySQL database using the PDO method.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
    $conn = new PDO("mysql:host=$servername", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch (PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
$conn = null;
?>
```

STEP 4 :

Here are more problems and solutions related to connecting PHP with a MySQL database using MySQL, MySQLi, and PDO.

Problem 19: Prepared Statements (MySQL) *Problem*: Write a PHP script to perform a basic SELECT query using prepared statements with MySQL.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = mysql_connect($servername, $username, $password);

if (!$conn) {
    die("Connection failed: " . mysql_error());
}

mysql_select_db($dbname, $conn);

$id = 1;

$stmt = mysql_prepare($conn, "SELECT name, email FROM users WHERE id = ?");
mysql_stmt_bind_param($stmt, "i", $id);
mysql_stmt_execute($stmt);
mysql_stmt_bind_result($stmt, $name, $email);
mysql_stmt_fetch($stmt);

echo "Name: " . $name . " - Email: " . $email;

mysql_close($conn);
?>
```

Problem 20: Prepared Statements (MySQLi) *Problem*: Write a PHP script to perform a basic SELECT query using prepared statements with MySQLi.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$id = 1;

$stmt = $conn->prepare("SELECT name, email FROM users WHERE id = ?");
$stmt->bind_param("i", $id);
$stmt->execute();
$stmt->bind_result($name, $email);
$stmt->fetch();

echo "Name: " . $name . " - Email: " . $email;

$stmt->close();
$conn->close();
?>
```

Problem 21: Prepared Statements (PDO) *Problem*: Write a PHP script to perform a basic SELECT query using prepared statements with PDO.

Solution:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $id = 1;

    $stmt = $conn->prepare("SELECT name, email FROM users WHERE id = :id");
    $stmt->bindParam(':id', $id);
    $stmt->execute();

    $result = $stmt->fetch(PDO::FETCH_ASSOC);

    if ($result) {
        echo "Name: " . $result["name"] . " - Email: " . $result["email"];
    } else {
        echo "No results found";
    }
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
?>
```

Problem 22: Transactions (MySQLi) *Problem*: Write a PHP script to perform a transaction using MySQLi to insert multiple records atomically.

Solution:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$conn->autocommit(false);

$sql1 = "INSERT INTO users (name, email) VALUES ('John', 'john@example.com')";
$sql2 = "INSERT INTO users (name, email) VALUES ('Jane', 'jane@example.com')";

if ($conn->query($sql1) === TRUE && $conn->query($sql2) === TRUE) {
    $conn->commit();
    echo "Transaction committed";
} else {
    $conn->rollback();
    echo "Transaction rolled back: " . $conn->error;
}

$conn->autocommit(true);
$conn->close();
?>
```

Problem 23: Transactions (PDO) *Problem*: Write a PHP script to perform a transaction using PDO to insert multiple records atomically.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $conn->beginTransaction();

    $sql1 = "INSERT INTO users (name, email) VALUES ('John', 'john@example.com')";
    $sql2 = "INSERT INTO users (name, email) VALUES ('Jane', 'jane@example.com')";

    $conn->exec($sql1);
    $conn->exec($sql2);

    $conn->commit();
    echo "Transaction committed";
} catch (PDOException $e) {
    $conn->rollBack();
    echo "Transaction rolled back: " . $e->getMessage();
}
$conn = null;
?>
```

STEP 5 :

Problem 24: Fetch All Records (MySQL) *Problem*: Write a PHP script to fetch and display all records from a MySQL database using the MySQL method.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = mysql_connect($servername, $username, $password);

if (!$conn) {
    die("Connection failed: " . mysql_error());
}

mysql_select_db($dbname, $conn);

$sql = "SELECT * FROM users";
$result = mysql_query($sql);

if (mysql_num_rows($result) > 0) {
    while ($row = mysql_fetch_assoc($result)) {
        echo "Name: " . $row["name"]. " - Email: " . $row["email"]. "<br>";
    }
} else {
    echo "No results found";
}

mysql_close($conn);
?>
```

Problem 25: Fetch All Records (MySQLi) *Problem*: Write a PHP script to fetch and display all records from a MySQL database using the MySQLi method.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT * FROM users";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "Name: " . $row["name"]. " - Email: " . $row["email"]. "<br>";
    }
} else {
    echo "No results found";
}

$conn->close();
?>
```

Problem 26: Fetch All Records (PDO) *Problem*: Write a PHP script to fetch and display all records from a MySQL database using the PDO method.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql = "SELECT * FROM users";
    $stmt = $conn->prepare($sql);
    $stmt->execute();

    $result = $stmt->fetchAll(PDO::FETCH_ASSOC);

    if (count($result) > 0) {
        foreach ($result as $row) {
            echo "Name: " . $row["name"]. " - Email: " . $row["email"]. "<br>";
        }
    } else {
        echo "No results found";
    }
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
?>
```

Problem 27: Database Backup (MySQL) *Problem*: Write a PHP script to backup a MySQL database using the MySQL method.

Solution:

```php
<?php
$databaseName = "myDB";
$username = "username";
$password = "password";

$backupFileName = 'backup.sql';

$command = "mysqldump --user=$username --password=$password --databases $databaseName > $backupFileName";

exec($command);

echo "Database backup completed to $backupFileName";
?>
```

Problem 28: Database Backup (MySQLi) *Problem*: Write a PHP script to backup a MySQL database using the MySQLi method.

Solution:

```php
<?php
$databaseName = "myDB";
$username = "username";
$password = "password";

$backupFileName = 'backup.sql';

$conn = new mysqli("localhost", $username, $password, $databaseName);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$command = "mysqldump --user=$username --password=$password --databases $databaseName > $backupFileName";

exec($command);

echo "Database backup completed to $backupFileName";

$conn->close();
?>
```

Problem 29: Database Backup (PDO) *Problem*: Write a PHP script to backup a MySQL database using the PDO method.

Solution:

```php
<?php
$databaseName = "myDB";
$username = "username";
$password = "password";

$backupFileName = 'backup.sql';

try {
    $conn = new PDO("mysql:host=localhost;dbname=$databaseName", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $command = "mysqldump --user=$username --password=$password --databases $databaseName > $backupFileName";

    exec($command);

    echo "Database backup completed to $backupFileName";
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
?>
```

Problem 30: Search Functionality (MySQLi) *Problem*: Write a PHP script to implement a search functionality using the MySQLi method to search for users by name in a database.

Solution:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$searchTerm = "John";

$sql = "SELECT * FROM users WHERE name LIKE '%$searchTerm%'";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "Name: " . $row["name"]. " - Email: " . $row["email"]. "<br>";
    }
} else {
    echo "No results found";
}

$conn->close();
?>
```