MODLE-03

Using Functions , Class- Objects, Forms in PHP: Functions in PHP, Function definition, Creating and invoking user-defined functions, Formal parameters versusactual parameters, Function and variable scope, Recursion, Library functions, Date and Time Functions

Strings in PHP: What is String?, Creating and Declaring String, String Functions

1. Functions in PHP:

PHP (Hypertext Preprocessor) is a widely-used open-source server-side scripting language designed for web development. One of the core aspects of PHP, like any programming language, is its ability to handle functions. Functions in PHP are blocks of code designed to perform specific tasks and can be reused multiple times throughout a program.

USES OF FUNCTION IN PHP PROGRAM

- **Modularization**: Functions allow you to break down your code into smaller, reusable units, making your code more organized and easier to maintain.
- **Code Reusability**: Once you define a function, you can call it multiple times from anywhere within your PHP script or even from other PHP files, promoting code reusability and reducing redundancy.
- **Abstraction**: Functions help in abstracting away complex logic, allowing you to focus on high-level functionality without getting bogged down in implementation details.
- **Encapsulation**: Functions encapsulate a specific task or functionality, promoting encapsulation and reducing the likelihood of unintended side effects.
- **Readability**: Well-named functions can make your code more readable and self-explanatory, enhancing its understandability for both yourself and other developers.
- **Parameterization**: Functions can accept parameters, allowing you to customize their behavior based on input values. This makes your code more flexible and adaptable to different scenarios.
- **Return Values**: Functions can return values, enabling them to produce output or results that can be further used or manipulated by other parts of your code.
- **Testing and Debugging**: Functions make it easier to test and debug your code because you can isolate specific functionality and focus on verifying its correctness independently.

FUNCTION DEFINITION:While creating a user defined function we need to keep few thingsin mind:

1. Any name ending with an open and closed parenthesis is afunction.
2. A function name always begins with the keyword function.
3. To call a function we just need to write its name followed by parenthesis.
4. A function name cannot start with a number. It can start with analphabet or underscore.
5. A function name is not case-sensitive.

Syntax:

```
function function_name()
{
        Executable code;
}
```

Example:

```
<?php
        function func()
        {
                echo "This is PHP program using Functions";
        }
        func();
 ?>
```

Output:

```
This is PHP program using Functions
```

SYNTAX:

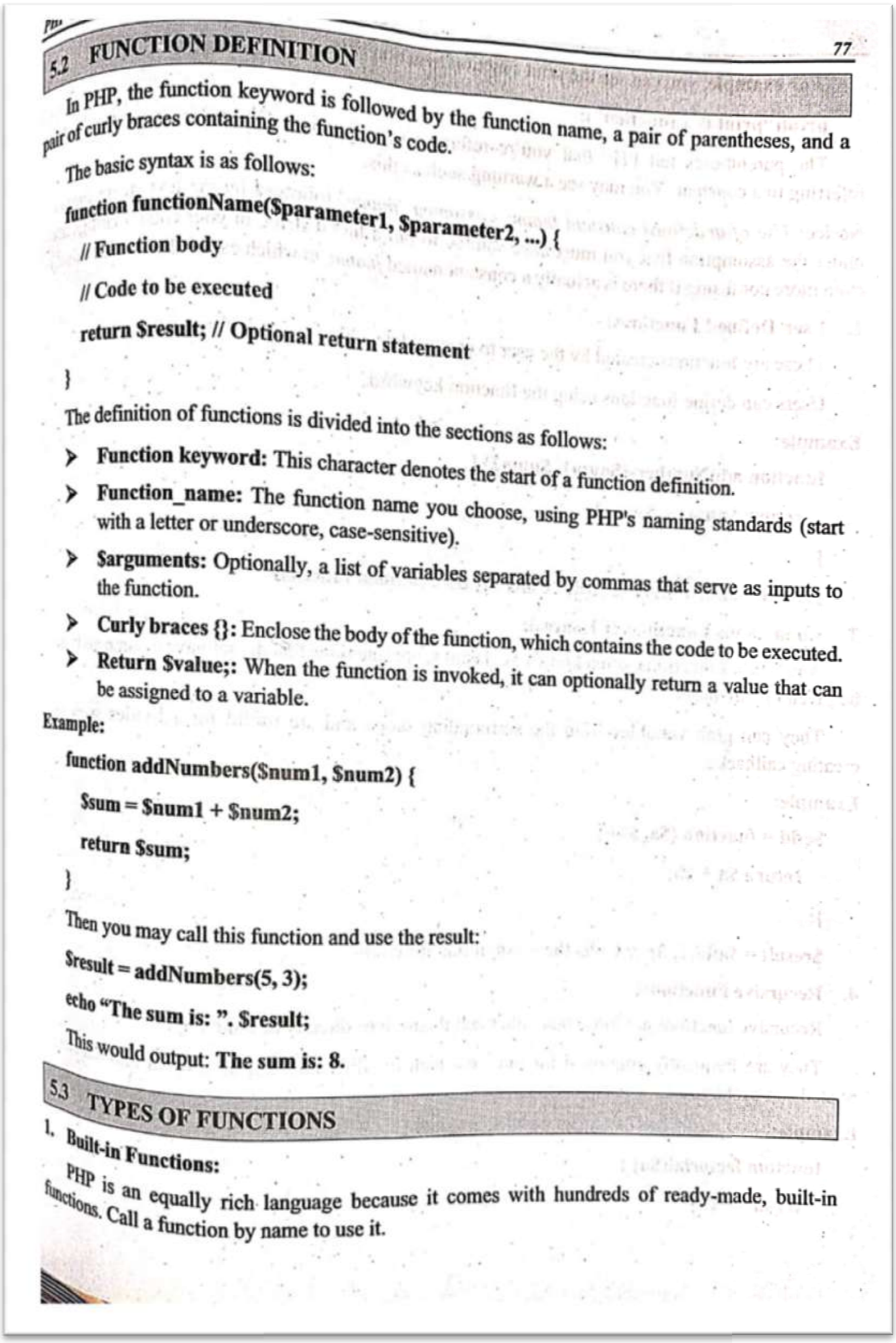
```
function functionName($parameter1, $parameter2, ...) {
   // Function body
   // Code to be executed when the function is called
   // This can include any PHP code, such as variable assignments, conditionals,
loops, etc.

   // Optional: return statement to return a value from the function
```

```
    // return $returnValue;
}
```

Explanation of the syntax elements:

function: This keyword is used to declare a function in PHP.

- **functionName**: This is the name of the function you are defining. It should follow the same naming rules as variables in PHP.
- **$parameter1, $parameter2, ...**: These are optional parameters that the function can accept. Parameters allow you to pass values to the function when it is called.
- **{ ... }**: This block contains the code that defines the behavior of the function. It can include any valid PHP code.
- **return**: This keyword is used to specify the value that the function should return. It is optional and can be omitted if the function does not need to return a value.



5.2 FUNCTION DEFINITION

In PHP, the function keyword is followed by the function name, a pair of parentheses, and a pair of curly braces containing the function's code.

The basic syntax is as follows:

```
function functionName($parameter1, $parameter2, ...) {
  // Function body
  // Code to be executed
  return $result; // Optional return statement
}
```

The definition of functions is divided into the sections as follows:

- **Function keyword:** This character denotes the start of a function definition.
- **Function_name:** The function name you choose, using PHP's naming standards (start with a letter or underscore, case-sensitive).
- **$arguments:** Optionally, a list of variables separated by commas that serve as inputs to the function.
- **Curly braces {}:** Enclose the body of the function, which contains the code to be executed.
- **Return $value;:** When the function is invoked, it can optionally return a value that can be assigned to a variable.

Example:

```
function addNumbers($num1, $num2) {
  $sum = $num1 + $num2;
  return $sum;
}
```

Then you may call this function and use the result:

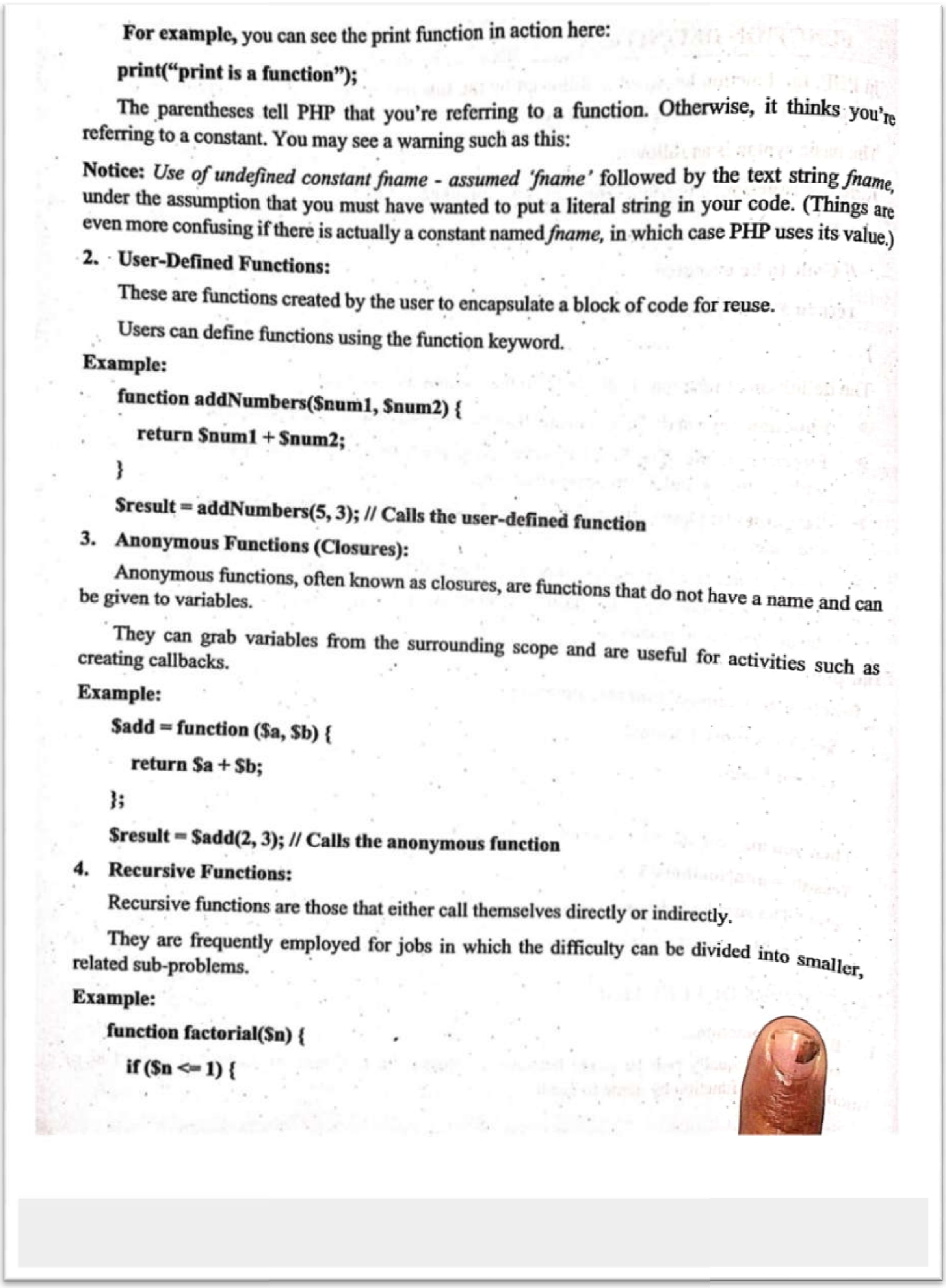
```
$result = addNumbers(5, 3);
echo "The sum is: ". $result;
```

This would output: **The sum is: 8.**

5.3 TYPES OF FUNCTIONS

1. **Built-in Functions:**

PHP is an equally rich language because it comes with hundreds of ready-made, built-in functions. Call a function by name to use it.



For example, you can see the print function in action here:

print("print is a function");

The parentheses tell PHP that you're referring to a function. Otherwise, it thinks you're referring to a constant. You may see a warning such as this:

Notice: *Use of undefined constant fname - assumed 'fname'* followed by the text string *fname,* under the assumption that you must have wanted to put a literal string in your code. (Things are even more confusing if there is actually a constant named *fname,* in which case PHP uses its value.)

2. **User-Defined Functions:**

   These are functions created by the user to encapsulate a block of code for reuse.

   Users can define functions using the function keyword.

Example:

```
function addNumbers($num1, $num2) {
   return $num1 + $num2;
}
$result = addNumbers(5, 3); // Calls the user-defined function
```

3. **Anonymous Functions (Closures):**

   Anonymous functions, often known as closures, are functions that do not have a name and can be given to variables.

   They can grab variables from the surrounding scope and are useful for activities such as creating callbacks.

Example:

```
$add = function ($a, $b) {
   return $a + $b;
};
$result = $add(2, 3); // Calls the anonymous function
```

4. **Recursive Functions:**

   Recursive functions are those that either call themselves directly or indirectly.

   They are frequently employed for jobs in which the difficulty can be divided into smaller, related sub-problems.

Example:

```
function factorial($n) {
   if ($n <= 1) {
        return 1;
    } else {
        return $n * factorial($n - 1);
    }
}
$result = factorial(5); // Calculates the factorial of 5
```

5. **Internal Functions:**

Internal functions are functions that are built-in or provided by PHP extensions. They are different from user-defined functions, which are created by the programmers themselves. Internal functions can be used directly in a script, without any declaration or definition. User-defined functions need to be declared before they can be used.

The GD library functions for image editing and the MySQLi extension functions for MySQL database interface are two examples of PHP extensions that provide internal functions.

- **GD:** This extension allows you to create and manipulate images in various formats, such as JPEG, PNG, GIF, etc. For example, you can use the ***imagecreatefromjpeg*** function to load a JPEG image from a file, and then use the ***imagefilter*** function to apply a grayscale effect to it.
- **MySQLi:** This extension provides an improved way of interacting with MySQL databases, supporting features such as prepared statements, transactions, stored procedures, etc. For example, you can use the ***mysqli_connect*** function to establish a connection to a MySQL server, and then use the ***mysqli_query*** function to execute a SQL statement.

Examples:

```
$image = imagecreate(100, 50);    // Internal GD library function
```

These are some of the most prevalent types of PHP functions. Understanding the various kinds will assist you in writing modular, reusable, and efficient code.

5.4 CREATING AND INVOKING USER-DEFINED FUNCTIONS

In PHP, creating and invoking user-defined functions entails defining a function and then calling it as necessary. Here's a step-by-step procedure:

1. **Creating a User-Defined Function:**

You can create a function using the function keyword, followed by the function name, parameters (if any), and the function body.

Here's a simple example:

```
<?php
function greetUser($name) {
   echo "Hello, $name!";
}
?>
```

In this example, the function greetUser takes a parameter $name and echoes a greeting.

2. **Invoking (Calling) the Function:**

After defining the function, you can call it by using its name followed by parentheses. Provid any required arguments inside the parentheses:

Example:

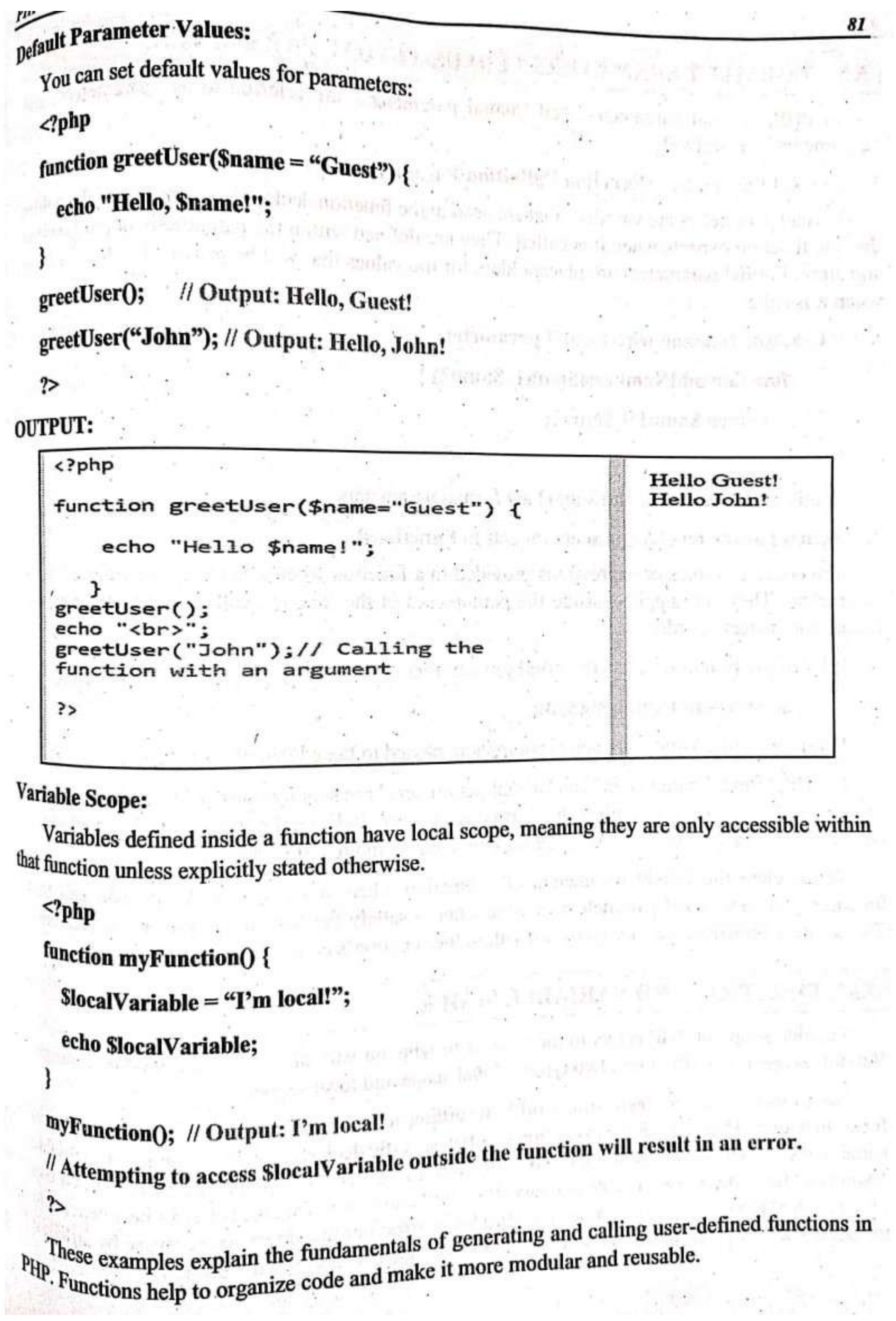
```
<?php
// Calling the function with an argument
greetUser("John");
// Output: Hello, John!
?>
```

Example with Return Statement:

You can also use the return statement to return a value from the function:

```
<?php
function addNumbers($num1, $num2) {
   $sum = $num1 + $num2;
   return $sum;
}
$result = addNumbers(5, 3);
echo "The sum is: ". $result;
// Output: The sum is: 8
?>
```

In this example, the addNumbers function returns the sum of two numbers, and the result is stored in the variable $result.



Default Parameter Values:

You can set default values for parameters:

```
<?php
function greetUser($name = "Guest") {
  echo "Hello, $name!";
}
greetUser();    // Output: Hello, Guest!
greetUser("John"); // Output: Hello, John!
?>
```

OUTPUT:

```
<?php
function greetUser($name="Guest") {
    echo "Hello $name!";
  }
greetUser();
echo "<br>";
greetUser("John");// Calling the
function with an argument
?>
```

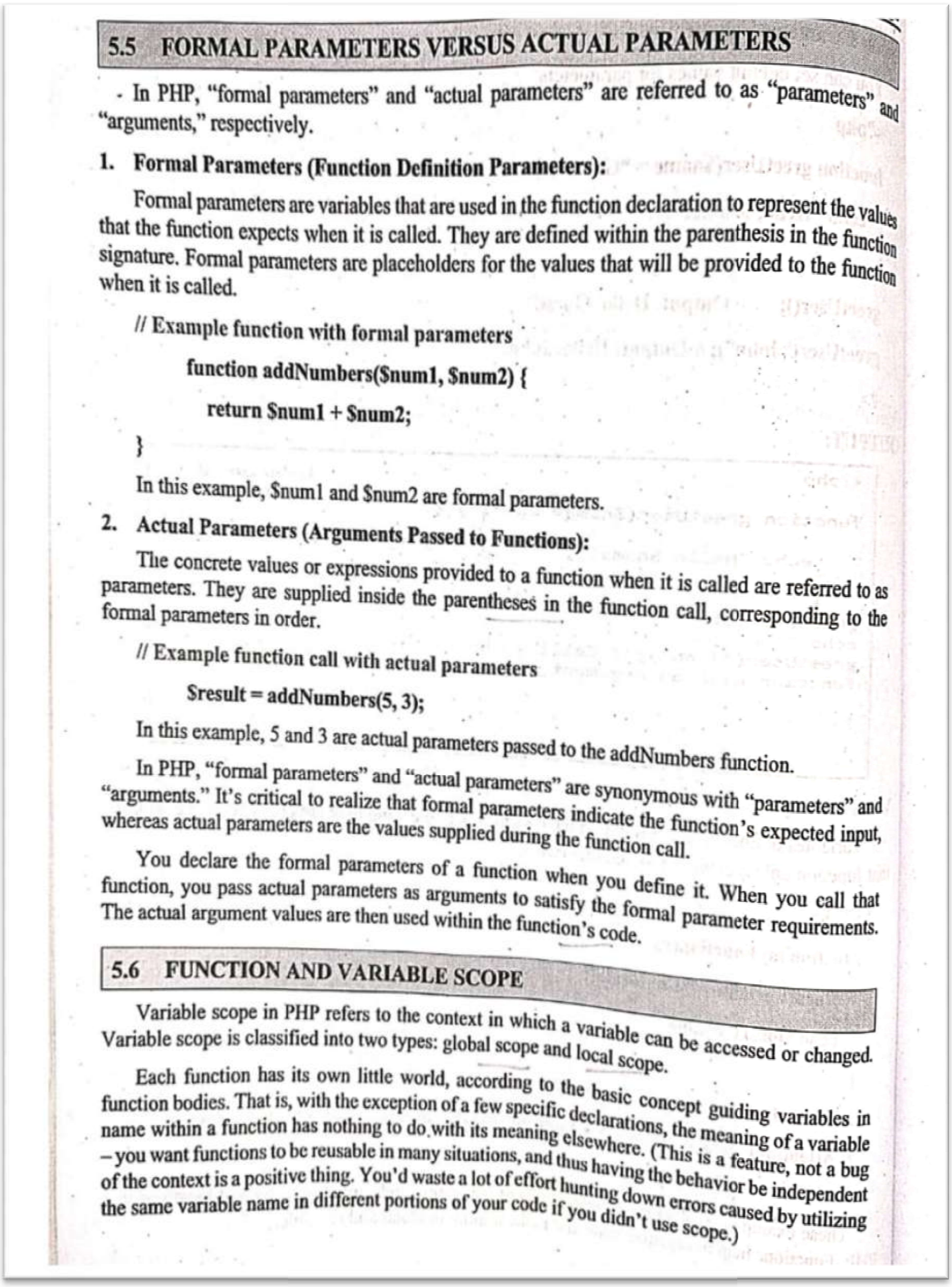
```
Hello Guest!
Hello John!
```

Variable Scope:

Variables defined inside a function have local scope, meaning they are only accessible within that function unless explicitly stated otherwise.

```
<?php
function myFunction() {
  $localVariable = "I'm local!";
  echo $localVariable;
}
myFunction(); // Output: I'm local!
// Attempting to access $localVariable outside the function will result in an error.
?>
```

These examples explain the fundamentals of generating and calling user-defined functions in PHP. Functions help to organize code and make it more modular and reusable.



5.5 FORMAL PARAMETERS VERSUS ACTUAL PARAMETERS

- In PHP, "formal parameters" and "actual parameters" are referred to as "parameters" and "arguments," respectively.

1. **Formal Parameters (Function Definition Parameters):**

   Formal parameters are variables that are used in the function declaration to represent the values that the function expects when it is called. They are defined within the parenthesis in the function signature. Formal parameters are placeholders for the values that will be provided to the function when it is called.

   ```
   // Example function with formal parameters
       function addNumbers($num1, $num2) {
         return $num1 + $num2;
   }
   ```

   In this example, $num1 and $num2 are formal parameters.

2. **Actual Parameters (Arguments Passed to Functions):**

   The concrete values or expressions provided to a function when it is called are referred to as parameters. They are supplied inside the parentheses in the function call, corresponding to the formal parameters in order.

   ```
   // Example function call with actual parameters
       $result = addNumbers(5, 3);
   ```

   In this example, 5 and 3 are actual parameters passed to the addNumbers function.

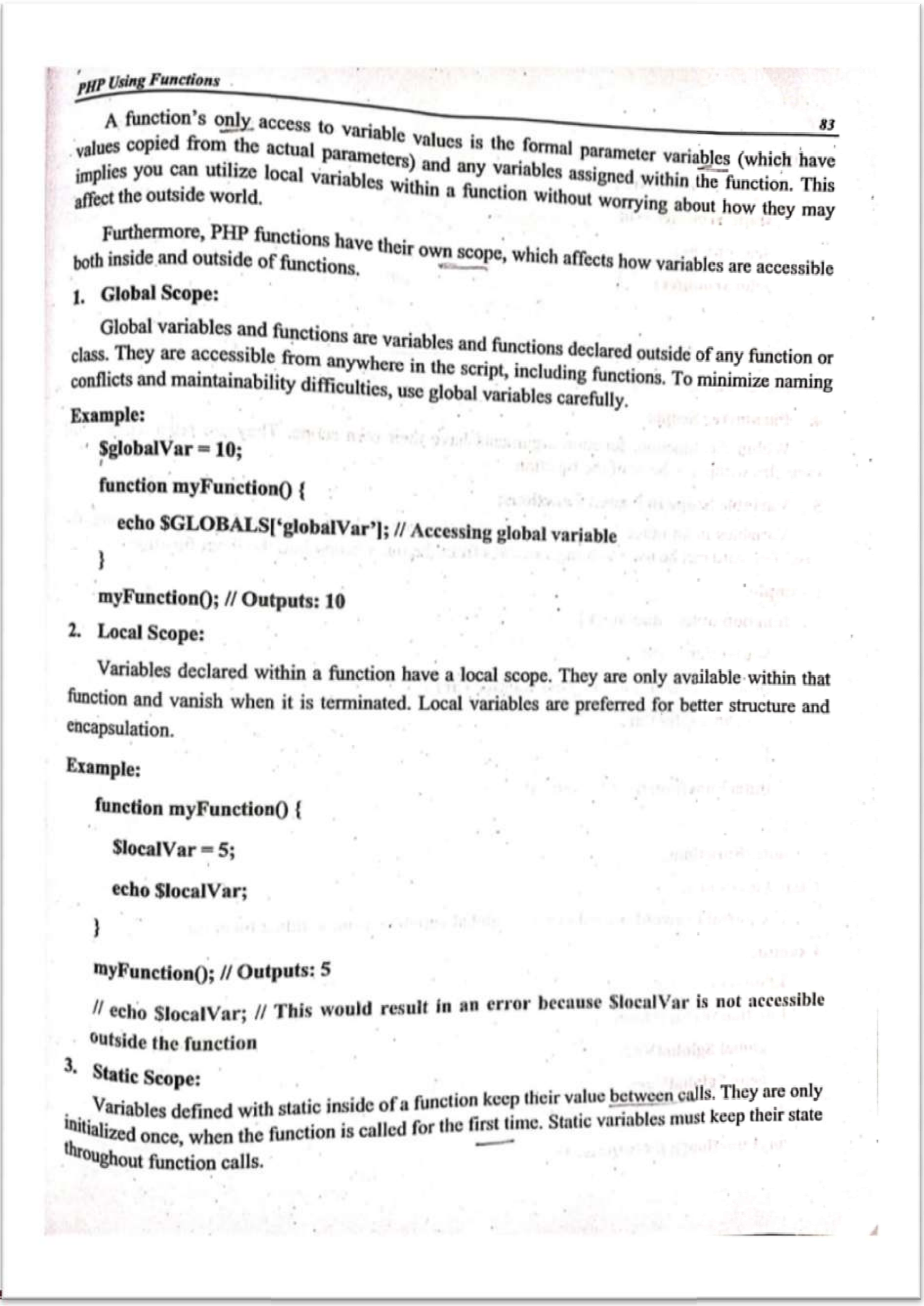
In PHP, "formal parameters" and "actual parameters" are synonymous with "parameters" and "arguments." It's critical to realize that formal parameters indicate the function's expected input, whereas actual parameters are the values supplied during the function call.

You declare the formal parameters of a function when you define it. When you call that function, you pass actual parameters as arguments to satisfy the formal parameter requirements. The actual argument values are then used within the function's code.

5.6 FUNCTION AND VARIABLE SCOPE

Variable scope in PHP refers to the context in which a variable can be accessed or changed. Variable scope is classified into two types: global scope and local scope.

Each function has its own little world, according to the basic concept guiding variables in function bodies. That is, with the exception of a few specific declarations, the meaning of a variable name within a function has nothing to do with its meaning elsewhere. (This is a feature, not a bug – you want functions to be reusable in many situations, and thus having the behavior be independent of the context is a positive thing. You'd waste a lot of effort hunting down errors caused by utilizing the same variable name in different portions of your code if you didn't use scope.)



A function's only access to variable values is the formal parameter variables (which have values copied from the actual parameters) and any variables assigned within the function. This implies you can utilize local variables within a function without worrying about how they may affect the outside world.

Furthermore, PHP functions have their own scope, which affects how variables are accessible both inside and outside of functions.

1. **Global Scope:**

Global variables and functions are variables and functions declared outside of any function or class. They are accessible from anywhere in the script, including functions. To minimize naming conflicts and maintainability difficulties, use global variables carefully.

Example:

```
$globalVar = 10;
function myFunction() {
  echo $GLOBALS['globalVar']; // Accessing global variable
}
myFunction(); // Outputs: 10
```

2. **Local Scope:**

Variables declared within a function have a local scope. They are only available within that function and vanish when it is terminated. Local variables are preferred for better structure and encapsulation.

Example:

```
function myFunction() {
  $localVar = 5;
  echo $localVar;
}
myFunction(); // Outputs: 5
// echo $localVar; // This would result in an error because $localVar is not accessible
outside the function
```

3. **Static Scope:**

Variables defined with static inside of a function keep their value between calls. They are only initialized once, when the function is called for the first time. Static variables must keep their state throughout function calls.

Example:

```
function increment() {
    static $counter = 0;
    $counter++;
    echo $counter;
}
increment(); // Outputs: 1
increment(); // Outputs: 2
```

4. **Parameter Scope:**

Within the function, function arguments have their own scope. They are regarded as local variables within the body of the function.

5. **Variable Scope in Nested Functions:**

Variables in an outer function are not directly accessible in an inner function. However, the 'use' keyword can be used to bring variables from the outer scope into the inner function.

Example:

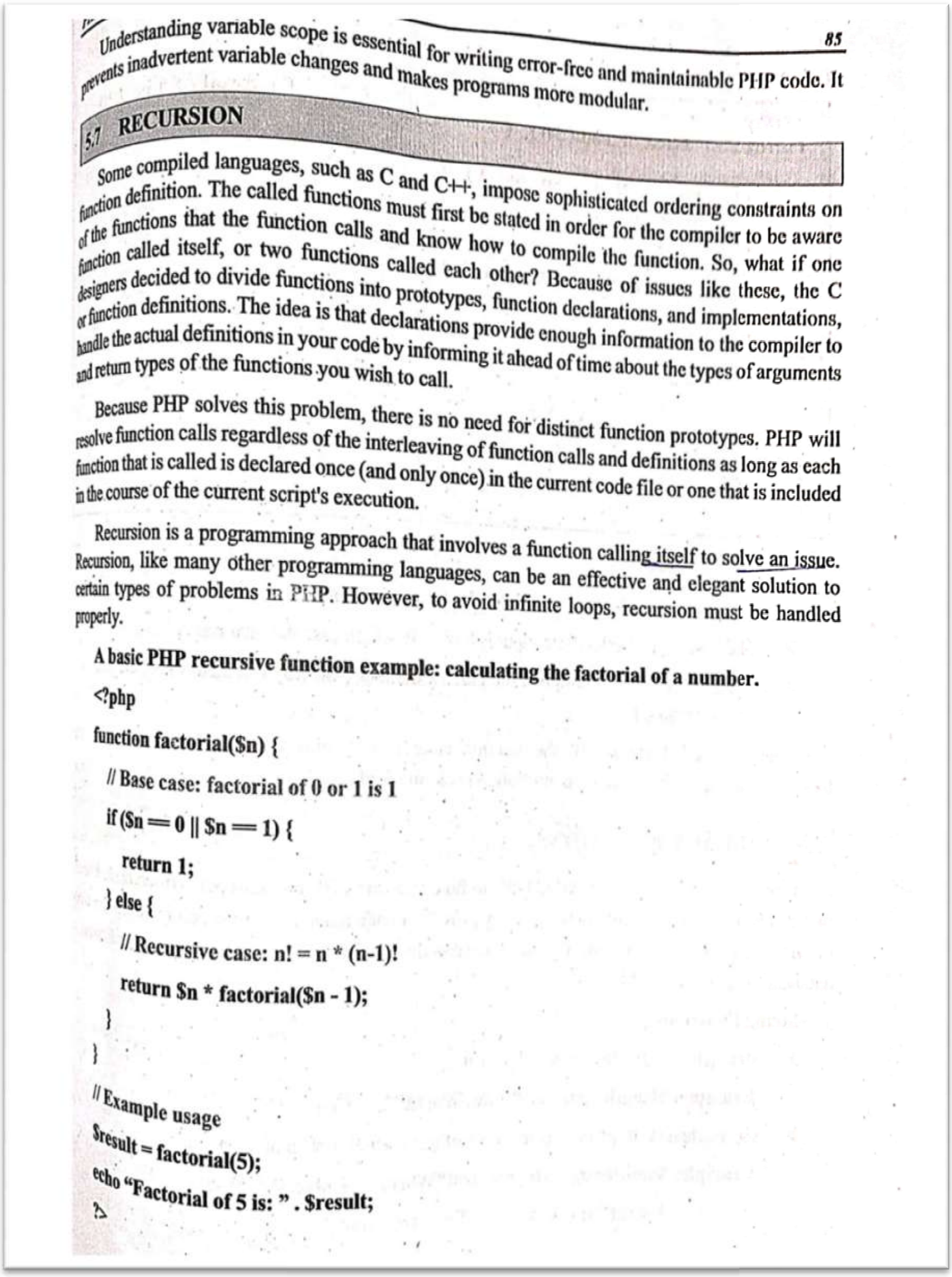
```
function outerFunction() {
    $outerVar = 10;
    function innerFunction() use ($outerVar) {
        echo $outerVar;
    }
    innerFunction(); // Outputs: 10
}
outerFunction();
```

Global Keyword:

The global keyword is used to access global variables from within a function.

Example:

```
$globalVar = 15;
function myFunction() {
    global $globalVar;
    echo $globalVar;
}
myFunction(); // Outputs: 15
```



Understanding variable scope is essential for writing error-free and maintainable PHP code. It prevents inadvertent variable changes and makes programs more modular.

5.7 RECURSION

Some compiled languages, such as C and C++, impose sophisticated ordering constraints on function definition. The called functions must first be stated in order for the compiler to be aware of the functions that the function calls and know how to compile the function. So, what if one function called itself, or two functions called each other? Because of issues like these, the C designers decided to divide functions into prototypes, function declarations, and implementations, or function definitions. The idea is that declarations provide enough information to the compiler to handle the actual definitions in your code by informing it ahead of time about the types of arguments and return types of the functions you wish to call.

Because PHP solves this problem, there is no need for distinct function prototypes. PHP will resolve function calls regardless of the interleaving of function calls and definitions as long as each function that is called is declared once (and only once) in the current code file or one that is included in the course of the current script's execution.

Recursion is a programming approach that involves a function calling itself to solve an issue. Recursion, like many other programming languages, can be an effective and elegant solution to certain types of problems in PHP. However, to avoid infinite loops, recursion must be handled properly.

A basic PHP recursive function example: calculating the factorial of a number.

```
<?php
function factorial($n) {
  // Base case: factorial of 0 or 1 is 1
  if ($n == 0 || $n == 1) {
    return 1;
  } else {
    // Recursive case: n! = n * (n-1)!
    return $n * factorial($n - 1);
  }
}
// Example usage
$result = factorial(5);
echo "Factorial of 5 is: " . $result;
?>
```

OUTPUT:

```
<?php
function factorial($n) {

    if ($n == 0 || $n == 1) {
        return 1;
    } else {

        return $n * factorial($n -
1);
    }
}

$result = factorial(5);
echo "Factorial of 5 is: " .
$result;
?>
```

```
Factorial of 5 is: 120
```

In this example:

- The **'factorial'** function calculates the factorial of a number.
- The base case checks if the input is 0 or 1, in which case the factorial is 1.
- The recursive case multiplies the current number **('$n')** by the factorial of the prev number **('$n - 1')**.

When you call **'factorial(5)'**, the function makes a series of recursive calls until it reaches base case, and then the results are multiplied back up the chain.

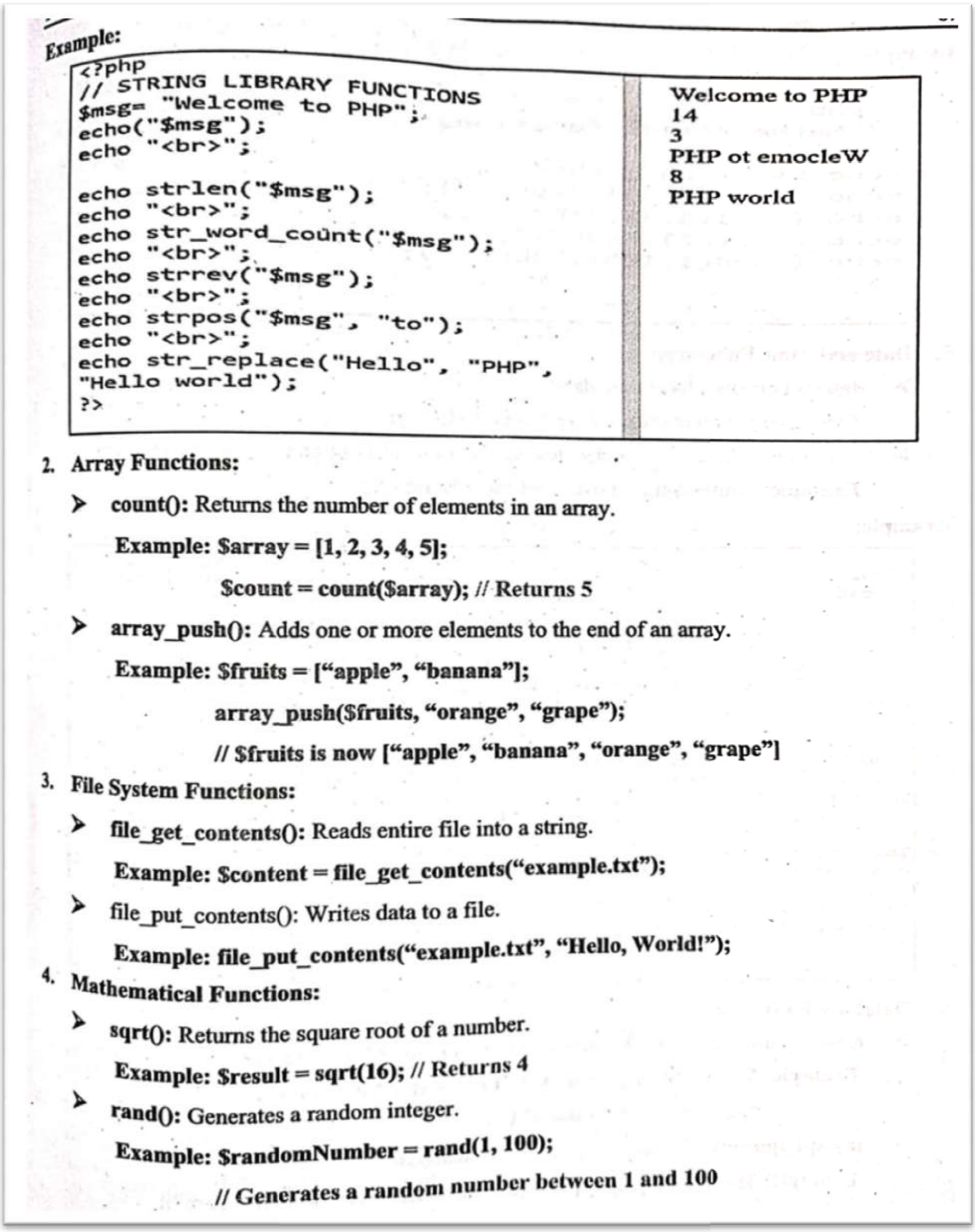
5.8 LIBRARY FUNCTIONS

Library functions, also known as built-in functions, are PHP-included per-written code bl that can be utilized for common tasks. They provide a wide range of features, eliminating the to write code from scratch for a variety of procedures. Everything from string manipulatic database access is covered.

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

     Example: $length = strlen("Hello, World!"); // Returns 13
   - **str_replace():** Replaces occurrences of a substring with another string.

     Example: $newString = str_replace("World", "Universe", "Hello, World!"); // Returns "Hello, Universe!"



Example:

```
<?php
// STRING LIBRARY FUNCTIONS
$msg= "Welcome to PHP";
echo("$msg");
echo "<br>";

echo strlen("$msg");
echo "<br>";
echo str_word_count("$msg");
echo "<br>";
echo strrev("$msg");
echo "<br>";
echo strpos("$msg", "to");
echo "<br>";
echo str_replace("Hello", "PHP",
"Hello world");
?>
```

```
Welcome to PHP
14
3
PHP ot emocleW
8
PHP world
```

2. **Array Functions:**

   ➤ **count():** Returns the number of elements in an array.

   Example: $array = [1, 2, 3, 4, 5];

   $count = count($array); // Returns 5

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

   Example: $fruits = [“apple”, “banana”];

   array_push($fruits, “orange”, “grape”);

   // $fruits is now [“apple”, “banana”, “orange”, “grape”]

3. **File System Functions:**

   ➤ **file_get_contents():** Reads entire file into a string.

   Example: $content = file_get_contents(“example.txt”);

   ➤ file_put_contents(): Writes data to a file.

   Example: file_put_contents(“example.txt”, “Hello, World!”);

4. **Mathematical Functions:**

   ➤ **sqrt():** Returns the square root of a number.

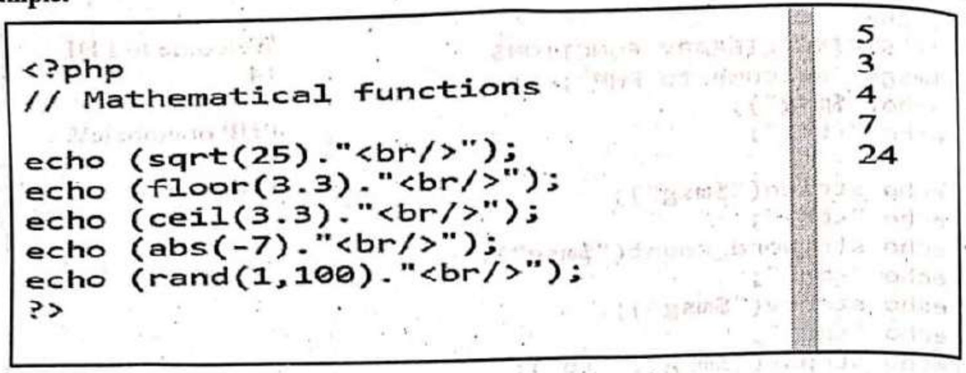
   Example: $result = sqrt(16); // Returns 4

   ➤ **rand():** Generates a random integer.

   Example: $randomNumber = rand(1, 100);

   // Generates a random number between 1 and 100

Example:



```
<?php
// Mathematical functions

echo (sqrt(25)."<br/>");
echo (floor(3.3)."<br/>");
echo (ceil(3.3)."<br/>");
echo (abs(-7)."<br/>");
echo (rand(1,100)."<br/>");
?>
```

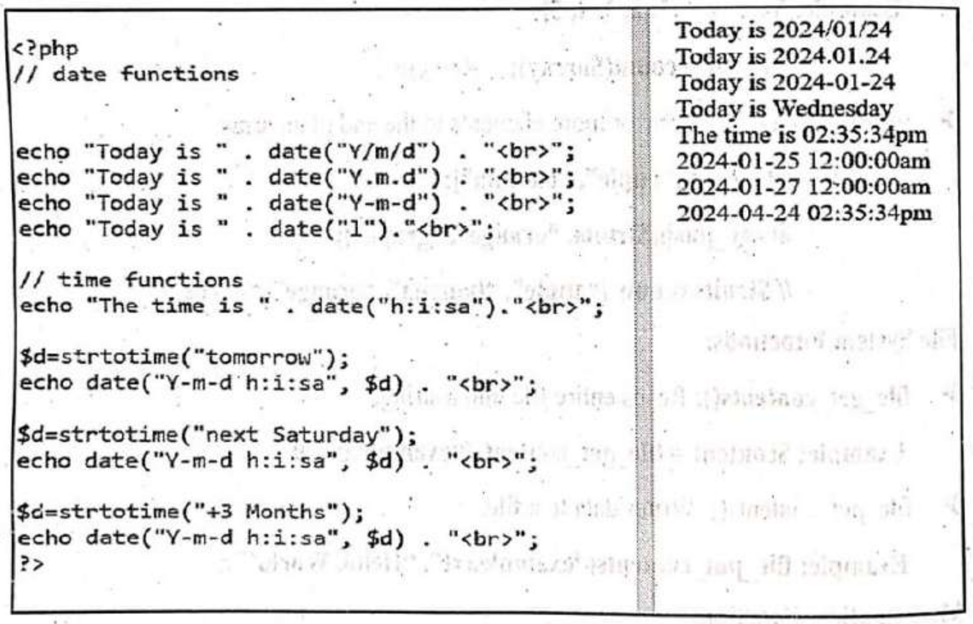
```
5
3
4
7
24
```

5. **Date and Time Functions:**
   - **date():** Formats a local time/date.

     Example: $currentDate = date("Y-m-d H:i:s");
   - **strtotime():** Parses any English textual datetime description into a Unix timestamp.

     Example: $timestamp = strtotime("next Sunday");

Example;



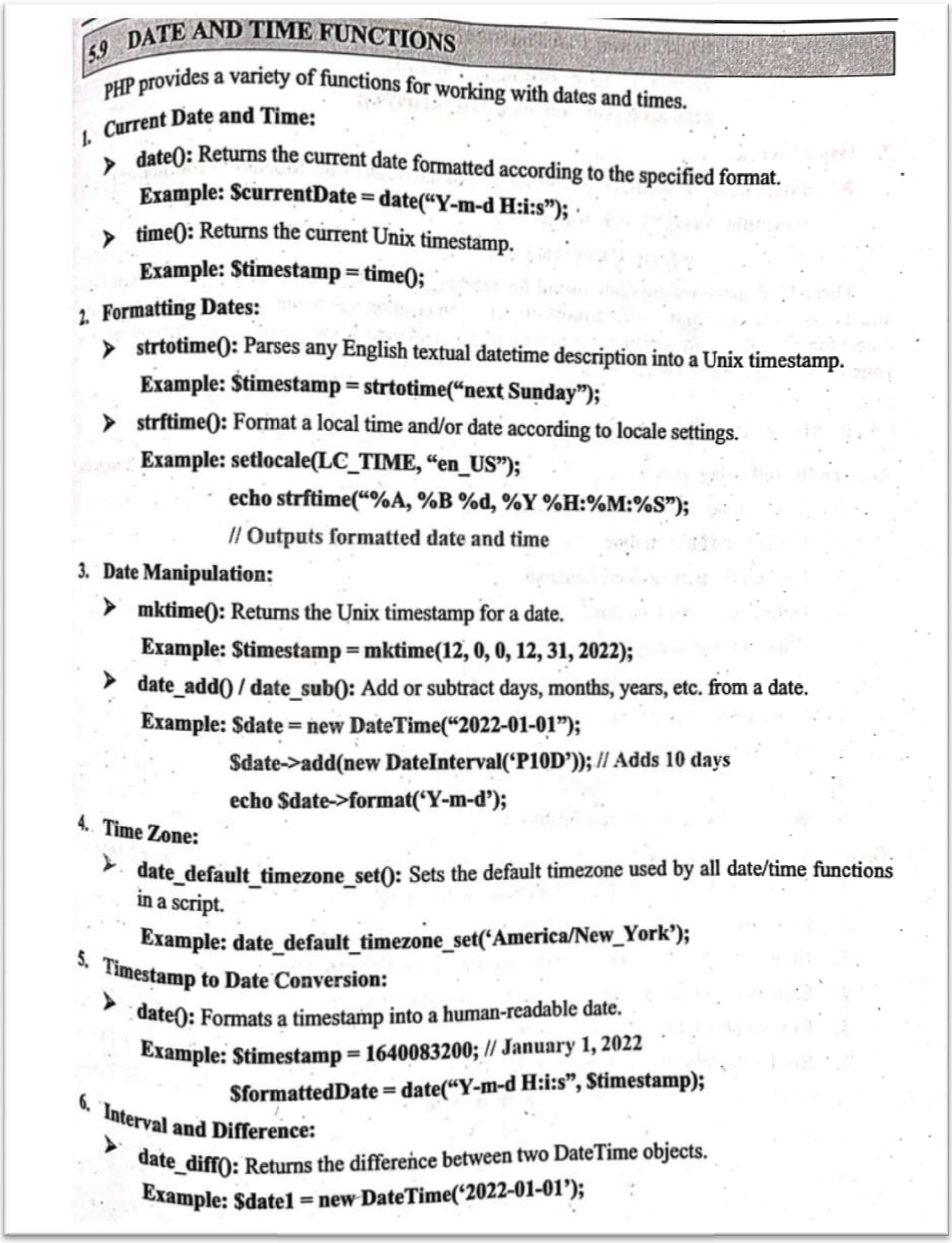
```
<?php
// date functions


echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l")."<br>";

// time functions
echo "The time is " . date("h:i:sa")."<br>";

$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("+3 Months");
echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

```
Today is 2024/01/24
Today is 2024.01.24
Today is 2024-01-24
Today is Wednesday
The time is 02:35:34pm
2024-01-25 12:00:00am
2024-01-27 12:00:00am
2024-04-24 02:35:34pm
```

6. **Database Functions:**
   - **mysqli_connect():** Opens a new connection to the MySQL server.

     Example: $conn = mysqli_connect("localhost", "username", "password", "database");
   - **mysqli_query():** Performs a query on the database.

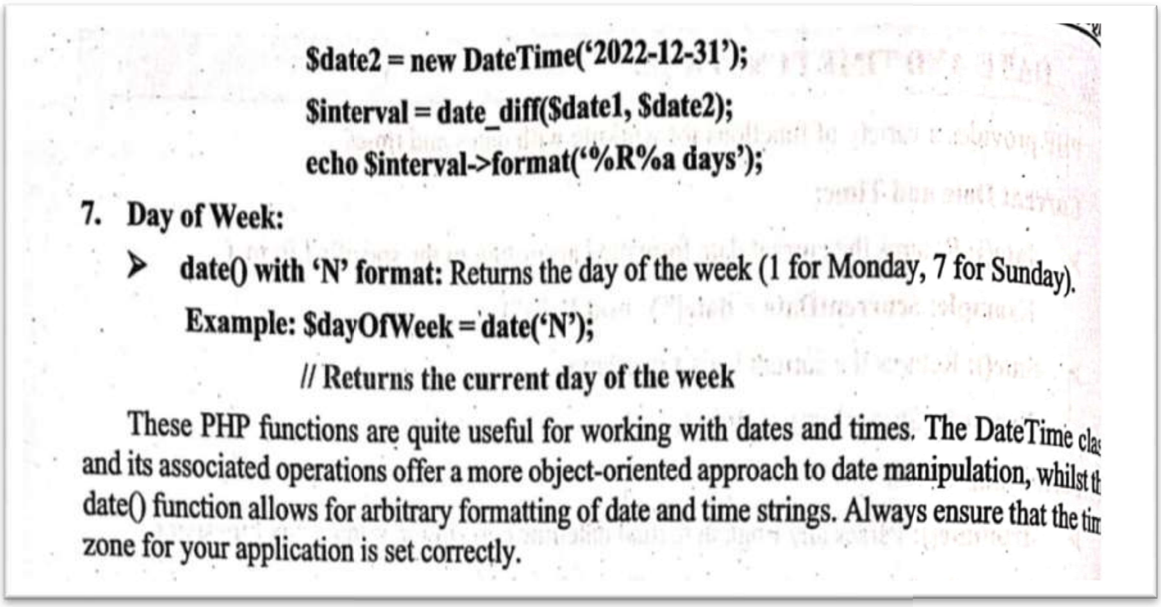
     Example: $result = mysqli_query($conn, "SELECT * FROM users");



5.9 DATE AND TIME FUNCTIONS

PHP provides a variety of functions for working with dates and times.

1. **Current Date and Time:**
   - **date():** Returns the current date formatted according to the specified format.
     Example: $currentDate = date("Y-m-d H:i:s");
   - **time():** Returns the current Unix timestamp.
     Example: $timestamp = time();
2. **Formatting Dates:**
   - **strtotime():** Parses any English textual datetime description into a Unix timestamp.
     Example: $timestamp = strtotime("next Sunday");
   - **strftime():** Format a local time and/or date according to locale settings.
     Example: setlocale(LC_TIME, "en_US");
     echo strftime("%A, %B %d, %Y %H:%M:%S");
     // Outputs formatted date and time
3. **Date Manipulation:**
   - **mktime():** Returns the Unix timestamp for a date.
     Example: $timestamp = mktime(12, 0, 0, 12, 31, 2022);
   - **date_add() / date_sub():** Add or subtract days, months, years, etc. from a date.
     Example: $date = new DateTime("2022-01-01");
     $date->add(new DateInterval('P10D')); // Adds 10 days
     echo $date->format('Y-m-d');
4. **Time Zone:**
   - **date_default_timezone_set():** Sets the default timezone used by all date/time functions in a script.
     Example: date_default_timezone_set('America/New_York');
5. **Timestamp to Date Conversion:**
   - **date():** Formats a timestamp into a human-readable date.
     Example: $timestamp = 1640083200; // January 1, 2022
     $formattedDate = date("Y-m-d H:i:s", $timestamp);
6. **Interval and Difference:**
   - **date_diff():** Returns the difference between two DateTime objects.
     Example: $date1 = new DateTime('2022-01-01');



```
$date2 = new DateTime('2022-12-31');
$interval = date_diff($date1, $date2);
echo $interval->format('%R%a days');
```

7. **Day of Week:**
   - **date() with 'N' format:** Returns the day of the week (1 for Monday, 7 for Sunday).

     Example: $dayOfWeek = date('N');

     // Returns the current day of the week

These PHP functions are quite useful for working with dates and times. The DateTime clas and its associated operations offer a more object-oriented approach to date manipulation, whilst t date() function allows for arbitrary formatting of date and time strings. Always ensure that the tin zone for your application is set correctly.

STRINGS: String is a sequence of characters.For example JSSCOLLEGE' or "JSSCOLLEGE" is string.Everything inside the quotes single(' ') or double(" ") in php can be treated as string. There are 2 ways to specify string in PHP

1. **single quoted strings:** We can create a string by enclosing text in a singlequote. This type of strings does not process special characters inside quotes.

Example

```
<?php
    $str='JSS COLLEGE';
    echo ' welcome to $str ';
?>
```

Output: welcome to $str

In the above program the echo statement prints the variable name rather than printing the contents of the variables.this is because single quote strings in php does not process special characters.Hence the string is unable to identify the $ sign as start of a variable name.

2. **double quoted strings:** we can specify string by enclosing text within double quotes. This type of strings can process special characters inside quotes.

Example

```
<?php
        echo "Hello php I am double quote String\n";
        $str="JSS COLLEGE";
        echo " welcome to $str";
?>
```

Output:

```
Hello php I am double quote String

welcome to JSS COLLEGE
```

In the above program we can see that the double quote string is processing the special characters according to their properties.The \n character is not printed and is considered as a newline .Also instead of the variable name jsscollege is printed

Example

```
<?php
        $str="Hello "php" I am
        double quote String";echo
        $str;
?>
```

Output:

```
Parse error: syntax error, unexpected 'quote'
    (T_STRING) inC:\wamp\www\string1.php on
line 2
```

Example

```
<?php
        $str="Hello \"php\" I am
        double quote String";echo $str;
?>
```

Output:

```
Hello php I am double quote String
```

Note: Using double quote String you can also display variable value on webpage

CONCATENATION OF TWO STRINGS

There are two string operators. The first is the concatenation operator ('.'), which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator ('.='), which appends the argument on the right side to the argument on the left side.

Example1:

```
<?php
        $a = 'Hello';
        $b = 'World!';
        $c = $a.$b;
        echo " $c \n";
?>
```

Output: HelloWorld!

Example2:

```
<?php
        $fname = 'John';
        $lname = 'Carter!';
        $c =
        $fname."
        ".$lname;
        echo " $c
        \n";
?>
```

Output: John Carter!

Example3:

```
<?php
$a = 'Hello';
```

```
$a. = "World!";
echo " $a \n";
?>
```

Output: HelloWorld!

STRING FUNCTIONS IN PHP

PHP have lots of predefined function which is used to perform operation with string some functions are:

1. **strlen():** strlen() function returns the length of a string.

Example

```
<?php
echo strlen("Hello world!");
?>
```

Output: 12

2. **str_word_count():** str_word_count() function is used to count numbers of words in given string.

Example:

```
<?php
        echo str_word_count("Hello world!");
?>
```

Output: 2

3. **strrev():** strrev() function is used to revers any string.

Example

```
<?php
        echo strrev("Hello world!");
?>
```

Output:!dlrow olleH

4. **strtolower():** strtolower() function is used to convert uppercase latter intolowercase latter.

Example

```
<?php
        $str="Hello friends i am HITESH";
        $str=strtol
        ower($str)
        ; echo
        $str;
?>
```

Output: hello friends i am hitesh

5. **strtoupper():** PHP strtoupper() function is used to convert lowercase latter intouppercase latter.

Example

```
<?php
$str="Hello friends i AM Hitesh";
$str=str toupper($str); echo $str;
?>
```

Output: HELLO FRIENDS I AM HITESH

6. **ucwords():** ucwords() function is used to convert first letter of every word intoupper case.

Example

```
<?php
$str="hello friends i am hitesh";
$str=ucwords($str); echo $str;
?>
```

Output: Hello Friends I Am Hitesh

7. **ucfirst():** ucfirst() function returns string converting first character into uppercase. It doesn't change the case of other characters.

Example

```
<?php
function firstUpper($string)
{
return(ucfirst($string));
}
$string="welcome to JSSCOLLEGE"; echo (firstUpper($string));
```

```
?>
```

Output: Welcome to JSSCOLLEGE

8. **lcfirst():** lcfirst() function is used to convert first character into lowercase. Itdoesn't change the case of other characters.

Example

```
<?php
function firstLower($string)
{
return(lcfirst($string));
}
$string="WELCOME to Jss"; echo (firstLower($string));
?>
```

9. **Output:** wELCOME to Jss

strstr()(or) **strchr():** The strchr() function searches for the first occurrence of astring inside another string.

Syntax: strchr(string,search)

Example:

```
<?php
echo strchr("Hello world!","world");
?>
```

output: world!

Example: Search a string for the ASCII value of "o" and return the rest of thestring:

```
<?php
echo strchr("Hello world!",111);
?>
```

Output: o world!

Example:

```
<?php
$originalStr = "jss for students";
$searchStr = "jss" ;
```

```
echo strchr($originalStr, $searchStr);
?>
```

Output: jss for students

Program 2: Program to demonstrate strchr() fucntion when word is not found.

```
<?php
$originalStr = "geeks for geeks";
$searchStr = "gfg" ;
echo strchr($originalStr, $searchStr);
?>
```

Output: -NIL-

10. **strcmp():**Compare two strings with case-sensitive manner

Syntax:	strcmp(string1,string2)

This function returns:

0 - if the two strings are equal

<0 - if string1 is less than string2

>0 - if string1 is greater than string2

Example:

```
<?php
echo strcmp("Hello","Hello"); echo "<br>";
echo strcmp("Hello","hELLo");
?>
```

Output:

```
0
-1
```

Example:

```
<?php
echo strcmp("Hello world!","Hello world!")."<br>";
echo strcmp("Hello world!","Hello")."<br>"; // string1 is greater than string2 echo
strcmp("Hello world!","Hello world! Hello!")."<br>"; // string1 is less than string2
?>
```

Output:

```
0
7
```

-7

11. **strcasecmp():**Compare two strings with case-insensitive manner

Syntax: strcasecmp(string1,string2)

This function returns:

0 - if the two strings are equal

<0 - if string1 is less than string2

>0 - if string1 is greater than string2

Example: <?php

echo strcasecmp("Hello","HELLO"); echo strcasecmp("Hello","hELLo");

?>

Output: 0

0

Example:

<?php

echo strcasecmp("Hello world!","HELLO WORLD!"); // The two strings are equal echo strcasecmp("Hello world!","HELLO"); // String1 is greater than string2 echo strcasecmp("Hello world!","HELLO WORLD! HELLO!"); // String1 is less than string2

?>

Output: 0

7

-7

strncmp(): The strncmp() is used to compare first n character of two strings. This function is case-sensitive which points that capital and small cases will be treated differently, during comparison.

Syntax: strncmp($str1, $str2, $len) This function returns:

0 - if the two strings are equal

<0 - if string1 is less than string2

>0 - if string1 is greater than string2

Example:

<?php

```
$str1 = " JSS college mysore 571125 ";
$str2 = "SS college mysore ooty road mysore";
// Both the strings are equal
$test=strncmp($str1, $str2, 16 );
echo "$test";
?>
```

Output:-51

12. **strpos()**:It enables searching particular text within a string. It works simply by matching the specific text in a string. If found, then it returns the specific position. If not found at all, then it will return “False”.

Syntax: Strpos(string,text);

Example

```
<?php
echo strpos(“Welcome to Cloudways”,”Cloudways”);
?>
```

Output: 11

13. **Str_replace():** It used for replacing specific text within a string.

Syntax: Str_replace(string to be replaced,text,string)

Example

```
<?php
echo str_replace(“cloudways”, “the programming world”, “Welcome to
cloudways”);
?>
```

Output: Welcome to the programming world

14. **str_repeat():** This function is used for repeating a string a specific number of times.

Syntax: Str_repeat(string,repeat)

Example

```
<?php
echo str_repeat(“=”,13);
?>
```

Output: =============

15. **substr():**This function you can display or extract a string from a particular position.

Syntax:substr(string,start,length)

Example

```
<?php
  echo substr("Welcome to Cloudways",6)."<br>";
  echo substr("Welcome to Cloudways",0,10)."<br>";
  ?>
```

Output:

```
e to Cloudways
Welcome to
```

16. **chunk_split():**The chunk_split() function splits a string into a series of smallerparts.

Syntax: chunk_split(string,length,end)

Example: Split the string after each sixth character and add a "..." after each split:

```
<?php
$str = "Hello world!";
echo chunk_split($str,6,"...");
?>
```

Output:

```
Hello ...world!...
```

17. **trim():**Removes whitespace or other characters from both sides of a string Parameter Description string Required.

Specifies the string to check charlist Optional. Specifies which characters to remove from the string.

If omitted, all of the following characters are removed: "\0" - NULL

"\t" - tab

"\n" - new line

"\x0B" - vertical tab

"\r" - carriage return

" " - ordinary white space

Example:

```
<?php
```

```
echo trim(" testing ")."<br>";
echo trim(" testing ", " teng");
?>
```

Output: testing sti

18. **ltrim():** Removes whitespace or other characters from the left side of a string

Syntax: ltrim(string,charlist)

```
<?php
$str = "Hello World!"; echo  $str  . "<br>"; echo ltrim($str,"Hello");
?>
```

Output: Hello World!
World!

19. **rtrim():** Removes whitespace or other characters from the right side of a string

Syntax: rtrim(string,charlist)

```
<?php
$str = "Hello World!"; echo $str . "<br>";
echo rtrim($str,"World!");
?>
```

Output: Hello World!
Hello

Write a PHP program to count number of characters in a given string.

```
<?php
$text="w3resource";
$search_char="r";
$count="0";
for($x="0"; $x< strlen($text); $x++)
{
$count=$count+1;
}
echo $count."\n";
?>
```

Function **Description**

chop()	Removes whitespace or other characters
	from the rightend of a string
chr()	Returns a character from a specified
ASCII value count_chars()	Returns information about characters
used in a stringecho()	Outputs one or more strings
explode()	Breaks a string into an array
fprintf()	Writes a formatted string to a specified output stream
parse_str()	Parses a query string into variables
print()	Outputs one or more strings
printf()	Outputs a formatted string
sprintf()	Writes a formatted string to a variable
sscanf()	Parses input from a string according to a format
str_getcsv()	Parses a CSV string into an array
str_ireplace()	Replaces some characters in a string
(case-insensitive)str_pad()	Pads a string to a new length
str_replace()	Replaces some characters in a string
(case-sensitive)str_shuffle()	Randomly shuffles all characters in
a string str_split()	Splits a string into an array
strcspn()	Returns the number of characters found
	in a string before any part of some
	specified characters are found
stripos()	Returns the position of the first
	occurrence of a stringinside another
	string (case-insensitive)
stristr()	Finds the first occurrence of a string
	inside anotherstring (case-insensitive)
strrchr()	Finds the last occurrence of a string
	inside anotherstring
strripos()	Finds the position of the last occurrence of a string
	inside another string (case-insensitive)
strrpos()	Finds the position of the last

	occurrence of a stringinside another string (case-sensitive)
strspn()	Returns the number of characters found in a string that contains only characters from a specified charlist
substr()	Returns a part of a string
substr_compare()	Compares two strings from a specified start position (binary safe and optionally case-
sensitive) substr_count()	Counts the number of times a
substring occurs in a	
	string
substr_replace()	Replaces a part of a string with another string