1. In Python, what is the difference between a built-in function and a user-defined function? Provide an

example of each.

Ans:

In Python, the main difference between a built-in function and a user-defined function lies in their origins and how they are created.

Built-in Functions:

Built-in functions are predefined functions that are provided by Python as part of its standard library. These functions are readily available for use without requiring any additional coding or importing external modules.

Example:

my\_list = [1, 2, 3, 4, 5]

length = len(my\_list)

print(length) # Output: 5

User-defined Functions:

User-defined functions are functions created by users to perform specific tasks or operations that are not covered by built-in functions. These functions are defined by the user within the program and can be reused multiple times by calling their name.

Example:

def calculate\_average(numbers):

total = sum(numbers)

average = total / len(numbers)

return average

my\_list = [2, 4, 6, 8, 10]

avg = calculate\_average(my\_list)

print(avg) # Output: 6.0

2. How can you pass arguments to a function in Python? Explain the difference between positional

arguments and keyword arguments.

Ans:

In Python, you can pass arguments to a function in two ways: positional arguments and keyword arguments.

Positional Arguments:

Positional arguments are passed to a function based on their position or order. The function receives the arguments in the same order as they are passed in the function call. The number and order of positional arguments must match between the function definition and the function call.

Here's an example of a function that takes two positional arguments:

def greet(name, message):

print(f"Hello {name}! {message}")

# Function call with positional arguments

greet("Alice", "How are you?")

Keyword Arguments:

Keyword arguments are passed to a function using the argument name and an equal sign (=) to assign values. This way, the order of the arguments doesn't matter, and you can selectively provide values for specific arguments, skipping others if needed. Keyword arguments are particularly useful when a function has many optional parameters.

Here's an example using keyword arguments:

def greet(name, message):

print(f"Hello {name}! {message}")

# Function call with keyword arguments

greet(message="How are you?", name="Alice")

3. What is the purpose of the return statement in a function? Can a function have multiple return

statements? Explain with an example.

Ans:

The return statement in a function is used to specify the value that the function should return when it is called. It serves two main purposes:

Terminating the execution: When a return statement is encountered in a function, it immediately ends the function's execution and returns the specified value back to the caller. Any code after the return statement within the function will not be executed.Providing a result: The return statement allows a function to pass a computed value or result back to the caller, which can then be stored in a variable or used in further operations.

Yes, a function can have multiple return statements. The function will terminate and return the specified value as soon as the first return statement is encountered during its execution. Here's an example to illustrate this:

def check\_number(num):

if num > 0:

return "Positive"

elif num < 0:

return "Negative"

else:

return "Zero"

result1 = check\_number(10)

print(result1) # Output: Positive

result2 = check\_number(-5)

print(result2) # Output: Negative

result3 = check\_number(0)

print(result3) # Output: Zero

4. What are lambda functions in Python? How are they different from regular functions? Provide an

example where a lambda function can be useful.

Ans:

In Python, lambda functions, also known as anonymous functions, are small, one-line functions that don't have a name. They are defined using the lambda keyword, followed by a list of parameters, a colon (:), and the expression to be evaluated. The result of the expression is automatically returned. Lambda functions are typically used when you need a simple function for a short period of time and don't want to define a separate named function.

The syntax for a lambda function is as follows:

lambda arguments: expression

5. How does the concept of "scope" apply to functions in Python? Explain the difference between local

scope and global scope.

Ans:

In Python, the concept of "scope" refers to the visibility and accessibility of variables within different parts of a program. It determines where a variable can be accessed and how long it remains available during the execution of a program.

Local Scope:

Local scope refers to the visibility of variables within a specific block of code, such as a function or a loop. Variables defined within a local scope are only accessible within that specific block and are known as local variables. Once the execution of the block ends, the local variables are destroyed and no longer available.

def my\_function():

x = 10

print(x)

my\_function()

Global Scope:

Global scope refers to the visibility of variables throughout the entire program. Variables defined outside any function or block have global scope and are known as global variables. Global variables can be accessed from any part of the program, including inside functions.

For example:

x = 10

def my\_function():

print(x)

my\_function()

6. How can you use the "return" statement in a Python function to return multiple values?

Ans:

In Python, the return statement is typically used to exit a function and optionally return a value. However, you can also use it to return multiple values by separating them with commas. There are a few ways to accomplish this:

Returning a tuple: You can use the return statement to return multiple values as a tuple. Here's an example:

def get\_values():

value1 = 10

value2 = 'hello'

return value1, value2

result = get\_values()

print(result) # Output: (10, 'hello')

7. What is the difference between the "pass by value" and "pass by reference" concepts when it

comes to function arguments in Python?

Ans:

In Python, the concepts of "pass by value" and "pass by reference" can be a bit misleading because the actual behavior depends on the type of the object being passed as an argument. In Python, function arguments are passed by assignment.

When you pass a mutable object, such as a list or a dictionary, as an argument to a function, it is effectively passed by reference. This means that any modifications made to the object within the function will affect the original object outside the function. Here's an example:

def modify\_list(my\_list):

my\_list.append(4)

my\_list = [1, 2, 3]

modify\_list(my\_list)

print(my\_list) # Output: [1, 2, 3, 4]

On the other hand, when you pass an immutable object, such as a number or a string, as an argument to a function, it is effectively passed by value. This means that the function receives a copy of the value, and any modifications made to the object within the function will not affect the original object outside the function. Here's an example:

def modify\_number(num):

num = num + 1

my\_number = 10

modify\_number(my\_number)

print(my\_number) # Output: 10

8. Create a function that can intake integer or decimal value and do following operations:

a. Logarithmic function (log x)

b. Exponential function (exp(x))

c. Power function with base 2 (2x)

d. Square root

Ans:

import math

def perform\_operations(number):

result = {}

# Logarithmic function (log x)

result['log'] = math.log(number)

# Exponential function (exp(x))

result['exp'] = math.exp(number)

# Power function with base 2 (2^x)

result['power'] = 2 \*\* number

# Square root

result['sqrt'] = math.sqrt(number)

return result

9. Create a function that takes a full name as an argument and returns first name and last name.

Ans:

Certainly! Here's an example of a Python function that takes a full name as an argument and returns the first name and last name as separate values:

Ans:

def get\_first\_last\_name(full\_name):

names = full\_name.split()

first\_name = names[0]

last\_name = names[-1]

return first\_name, last\_name

full\_name = "John Doe"

first\_name, last\_name = get\_first\_last\_name(full\_name)

print("First name:", first\_name)

print("Last name:", last\_name)

#Output

First name: John

Last name: Doe