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

Answer: Built-in functions are functions that are pre-defined in Python and are available for use without needing to be explicitly defined. Examples of built-in functions include `print()`, `len()`, `input()`, etc.

# Example of a built-in function

print("Hello, world!")

User-defined functions are functions that are defined by the users to perform specific tasks. These functions need to be explicitly defined before they can be used.

# Example of a user-defined function

def greet(name):

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

greet("Alice")

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

Answer: In Python, arguments can be passed to a function when it is called. There are two ways to pass arguments to a function:

1. Positional Arguments: Positional arguments are passed based on their position or order in the function call. The order in which arguments are passed is crucial.

def add\_numbers(a, b):

return a + b

result = add\_numbers(5, 3)

print(result)

Output: 8

2. Keyword Arguments: Keyword arguments are passed with their corresponding parameter names, and their order does not matter.

def greet(name, age):

print(f"Hello, {name}! You are {age} years old.")

greet(name="Alice", age=30)

Q.3. What is the purpose of the `return` statement in a function? Can a function have multiple return statements? Explain with an example.

Answer: The `return` statement in a function is used to specify the value that the function will return when it is called. The return statement also terminates the function, so any code after the return statement within the same function will not be executed.Yes, a function can have multiple return statements, but only one return statement will be executed when the function is called.

def check\_even\_odd(num):

if num % 2 == 0:

return "Even"

else:

return "Odd"

result = check\_even\_odd(5)

print(result)

Output: Odd

Q.4. What are lambda functions in Python? How are they different from regular functions? Provide an example where a lambda function can be useful.

Answer: Lambda functions, also known as anonymous functions, are functions without a name. They are created using the `lambda` keyword and are typically used for short, simple operations.Regular functions are defined using the `def` keyword and can have a name, arguments, and a block of code.

# Regular function to calculate the square of a number

def square(x):

return x \*\* 2

# Equivalent lambda function to calculate the square of a number

square\_lambda = lambda x: x \*\* 2

Lambda functions are often used when a simple function is required for a short duration, such as in sorting, filtering, or mapping operations.

# Example where a lambda function can be useful

numbers = [1, 5, 2, 8, 3, 7]

# Sorting the list of numbers in ascending order using a lambda function

sorted\_numbers = sorted(numbers, key=lambda x: x)

print(sorted\_numbers)

Output: [1, 2, 3, 5, 7, 8]

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

Answer: Scope refers to the visibility and accessibility of variables within different parts of the code. In Python, functions have their own scope, and variables defined within a function are considered local to that function by default.

- Local Scope: Variables defined within a function are only accessible within that function's block of code. They cannot be accessed from outside the function.

def my\_function():

x = 10 # Local variable within the function

print(x)

my\_function()

Output: 10

# Trying to access the local variable x outside the function will raise an error

print(x) # Raises NameError: name 'x' is not defined

- Global Scope: Variables defined outside any function or at the top level of the code have global scope. They can be accessed from any part of the code, including inside functions.

y = 20 # Global variable

def my\_function():

print(y)

my\_function() # Output: 20

# The global variable y can also be accessed outside the function

print(y)

Output: 20

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

Answer: In Python, you can use the `return` statement to return multiple values from a function by packing them into a data structure like a tuple, list, or dictionary.

def get\_values():

x = 10

y = 20

return x, y # Packing x and y into a tuple

result = get\_values()

print(result) # Output: (10, 20)

# Unpacking the returned tuple

x\_value, y\_value = get\_values()

print(x\_value)

Output: 10

print(y\_value)

Output: 20

Q.7. What is the difference between the "pass by value" and "pass by reference" concepts when it comes to function arguments in Python?

Answer: In Python, all function arguments are passed by object reference, which means that the function receives a reference to the object (variable) passed as an argument.

- Pass by Value: When you pass immutable objects (e.g., int, float, tuple, etc.) to a function, the function receives a copy of the object's value. Any changes made to the parameter inside the function will not affect the original object.

- Pass by Reference: When you pass mutable objects (e.g., list, dictionary, set, etc.) to a function, the function receives a reference to the original object. Any changes made to the parameter inside the function will affect the original object.

# Pass by Value (Immutable)

def modify\_value(x):

x += 10

print("Inside the function:", x)

num = 5

modify\_value(num)

print("Outside the function:", num)

Output: 5 (No change)

# Pass by Reference (Mutable)

def modify\_list(my\_list):

my\_list.append(4)

print("Inside the function:", my\_list)

numbers = [1, 2, 3]

modify\_list(numbers)

print("Outside the function:", numbers)

Output: [1, 2, 3, 4]

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

a. Logarithmic function (log x)

b. Exponential function (exp(x))

c. Power function with base 2 (2^x)

d. Square root

Answer:

import math

def math\_operations(num):

log\_result = math.log(num)

exp\_result = math.exp(num)

power\_result = math.pow(2, num)

square\_root\_result = math.sqrt(num)

print(f"log({num}) = {log\_result}")

print(f"exp({num}) = {exp\_result}")

print(f"2^{num} = {power\_result}")

print(f"√{num} = {square\_root\_result}")

# Test the function

math\_operations(4)

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

Answer:

def extract\_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 = extract\_first\_last\_name(full\_name)

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

Output: First Name: John

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

Output: Last Name: Doe

This function splits the full name using the `split()` method and returns the first and last names as a tuple. You can then unpack the tuple to get the first name and last name separately.