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

**Answer:**

**A built-in function** is a function that is already defined and available as part of a programming language or its standard library. These functions are typically included in the language itself and can be directly used without any additional effort. Examples of built-in functions include print() for printing output, len() for determining the length of a string or list, and range() for generating a sequence of numbers.

**User-defined function** is created by the programmer to fulfill specific requirements. These functions are not predefined in the programming language and need to be explicitly defined by the user. This helps in modularizing code, improving code readability, and reducing redundancy. Here's an example of a user-defined function in Python:

**Program:**

**def Square(n):**

**result = n\* n**

**return result**

**Square(5)**

**Output: 25**

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

**Answer:**

We 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 order in which you provide the arguments when calling the function must match the order in which the function expects them. Here's an example:

**Program:**

**def greet(name, age):**

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

**greet("Vikas", 28)**

**Output: “Hello, Vikas! You are 28 years old.”**

* **Keyword arguments** are passed to a function using their corresponding parameter names. This allows you to specify arguments in any order, as long as you explicitly mention the parameter names. Here's an example

**Program**

**def greet(name, age):**

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

**greet(age= 28,name= "Vikas")**

**Output: “Hello, Vikas! You are 28 years old.”**

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 purpose of the return statement in a function is to specify the value that the function should produce or provide as output when it is called. When a return statement is encountered in a function, it immediately exits the function and returns the specified value to the caller.

The purpose of the return statement in a function is to specify the value that the function should produce or provide as output when it is called. When a return statement is encountered in a function, it immediately exits the function and returns the specified value to the caller.

A function can indeed have multiple return statements, but only one of them will be executed. When a return statement is executed, it terminates the function and any subsequent code in the function will not be executed.

**Program:**

**def grade(score):**

**if score >= 90:**

**return "A"**

**elif score >= 80:**

**return "B"**

**elif score >= 70:**

**return "C"**

**else:**

**return "D"**

**result = grade(85)**

**print(result)**

**Output: “B”**

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 small, one-line functions that do

not have a name. lambda functions can take any number of arguments but can only have a

single expression.

**Syntax:**

*lambda arguments: expression*

**# Regular function to square a number**

**def square(x):**

**return x \*\* 2**

**# Lambda function to square a number**

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

**print(square(5)) # Output: 25**

**print(square\_lambda(5))**

Lambda functions are often used in situations where a small, simple function is required as an

argument to another function, such as in functional programming, when working with higher-order

functions like map(), filter(), or reduce(). They provide a convenient way to define functions on the

fly without the need for a separate function definition.

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

**Answer:**

The concept of "**scope**" refers to the visibility and accessibility of variables within a program. The scope determines where a variable can be accessed and how long it remains valid.

**Local Scope:**

A local scope refers to the region within a function where variables are defined. Variables created within a function have local scope, meaning they are only accessible within that function. These variables are known as local variables. Once the function execution completes, the local variables are destroyed and cannot be accessed from outside the function

**Program:**

**def my\_function():**

**x = 10**

**print(x) # Accessible within the function**

**my\_function()**

**# print(x) # Error: x is not defined outside the function**

**Global Scope:**

The global scope refers to the outermost scope in a Python program. Variables defined outside any function or block have global scope, which means they can be accessed from anywhere in the program, including inside functions. Global variables are accessible within functions, but if you want to modify their values within a function, you need to explicitly declare them as global using the global keyword.

**Program:**

**y = 20 # Global variable**

**def my\_function():**

**global y**

**print(y) # Accessible within the function**

**y = 30 # Modifying the global variable**

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

Answer:

You can return multiple values from a function by packing them into a tuple and returning the tuple. The caller can then unpack the returned tuple into separate variables

**Program**:

**def get\_info():**

**name = "Alice"**

**age = 25**

**city = "New York"**

**return name, age, city**

**# Calling the function and unpacking the returned values**

**person\_name, person\_age, person\_city = get\_info()**

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, the concepts of "pass by value" and "pass by reference" are often used to

describe how function arguments are treated.

**Pass by Value:**

In pass by value, a copy of the value of a variable is passed to a function. Any modifications made to the function parameter within the function do not affect the original variable in the calling code. Essentially, changes made to the function parameter are isolated within the function scope. This concept is commonly seen in languages like C or Java.

**Pass by Reference (Object References in Python):**

In pass by reference or passing object references in Python, the reference to an object is passed to a function. This means that the function parameter and the original variable in the calling code both refer to the same object in memory. If changes are made to the object through the function parameter, they will be reflected in the original variable. This concept is used in Python.

**Program:**

**def modify\_list(lst):**

**lst.append(4) # Modifying the list**

**lst = [1, 2, 3] # Reassigning the list parameter to a new list**

**my\_list = [0]**

**modify\_list(my\_list)**

**print(my\_list) # Output: [0, 4]**

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

a. Logarithmic function (log x)

**import math**

**def logarithm(base, number):**

**try:**

**result = math.log(number, base)**

**return result**

**except ValueError:**

**print("Error: Invalid input for logarithm calculation.")**

**return None**

**logarithm = logarithm(10, 100)**

**print(f"logarithm of number of 100 with base value:10 is {logarithm}")**

b. Exponential function (exp(x))

**import math**

**def exponential(number):**

**try:**

**result = math.exp(number)**

**return result**

**except ValueError:**

**print("Error: Invalid input for logarithm calculation.")**

**return None**

**exp\_value = exponential(2)**

**print(f"Exponential of number is {exp\_value}")**

c. Power function with base 2 (2x)

import math

**def power(number):**

**try:**

**result = 2\*\*number**

**return result**

**except ValueError:**

**print("Error: Invalid input for logarithm calculation.")**

**return None**

**Power\_value = power(2)**

**print(f"Power with base 2 of number is {Power\_value}")**

d. Square root

**import math**

**def sqrt(number):**

**try:**

**result = number\*\*0.5**

**return result**

**except ValueError:**

**print("Error: Invalid input for logarithm calculation.")**

**return None**

**sqrt\_value = sqrt(9)**

**print(f"Square root of number is {sqrt\_value}")**

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

**def full\_name\_giver(text):**

**name\_list=text.split()**

**first\_name= name\_list[0]**

**last\_name= name\_list[1]**

**return first\_name, last\_name**

**full\_name\_giver("vikas sikarwar")**