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

**Ans 1:** In Python, the main difference between a built-in function and a user-defined function are as follows :

1. Built-in Function:

* Built-in functions are predefined functions provided by Python. They are readily available for use without requiring any additional definitions.
* Examples of built-in functions include print(), len(), sum(), max(), min(), etc.
* print(“Hello World!!”)

1. User-defined Function:

* User-defined functions are functions defined by the user as per their specific requirements.
* They are defined using the def keyword followed by the function name and a block of code to be executed when the function is called.
* For example:

def sum(a,b):

c = a + b

return c

print(sum(10,20))

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

**arguments and keyword arguments**.

**Ans 2:** In Python, you can pass arguments to a function in two main ways: positional arguments and keyword arguments.

1. **Positional Arguments:**

* Positional arguments are passed to a function based on their position or order in the function's parameter list.
* The order in which positional arguments are passed must match the order in which the parameters are defined in the function's definition.
* These arguments are mandatory unless a default value is provided for the corresponding parameter in the function definition.

For example :

def hello(name,surname):

print(f”hello {name} {surname} !!!”)

hello(‘neelam’,’ghavate’)

1. **Keyword Arguments:**

* Keyword arguments are passed to a function with the parameter name explicitly specified along with the argument value.
* Unlike positional arguments, the order of keyword arguments doesn't matter.
* Keyword arguments are useful when you want to make the code more readable by explicitly specifying which argument corresponds to which parameters.

For example:

def hello(name,surname):

print(f”hello {name} {surname} !!!”)

hello(name = ‘neelam’, surname =’ghavate’)

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

**statements? Explain with an example.**

**Ans 3:** The **return** statement in a function serves the purpose of exiting the function and returning a value (or values) back to the caller. It essentially provides a way for a function to send data back to the code that called it. Once a **return** statement is executed, the function stops executing, and the value (or values) specified in the **return** statement are passed back to the caller.

A function can indeed have multiple **return** statements. However, once a **return** statement is encountered, the function immediately exits, and the value specified in that particular **return** statement is returned to the caller. Subsequent **return** statements in the same function will not be executed.

**For example** :

def even\_odd(num):

if num % 2 == 0:

return “Even”

else:

return “Odd”

result = even\_odd(5)

print("The number is:", result)

**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 4:** Lambda functions in Python are anonymous functions that are defined using the lambda keyword instead of the standard **def** keyword used for regular functions. Lambda functions are typically used for short, simple operations where defining a full-fledged function using **def** would be overkill. They are often used in situations where a function is needed for a short period and doesn't need a specific name associated with it.

Example where a lambda function:

numbers = [1, 2, 3, 4, 5] # Using a lambda function to square each number in the list

squared\_numbers = list(map(lambda x: x\*\*2, numbers))

print(squared\_numbers) # Output: [1, 4, 9, 16, 25]

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

**Ans 5:** In Python, the concept of "scope" refers to the region of a program where a particular variable is accessible. It determines the visibility and lifetime of variables. There are primarily two types of scope: local scope and global scope.

**Local Scope:**

* Local scope refers to variables defined within a function. These variables are only accessible within the function where they are defined.
* Variables declared inside a function have local scope, meaning they cannot be accessed from outside the function.
* Local variables are created when the function is called and destroyed when the function exits.

**Example of local scope:**

def my\_function():

x = 10 # Local variable

print("Inside the function, x is:", x)

my\_function()

**Global Scope**:

* Global scope refers to variables defined outside of any function or in the global scope of a module.
* Global variables can be accessed and modified from any part of the code, including within functions.
* However, if a function wants to modify a global variable, it must explicitly declare it using the **global** keyword.

**Example of global scope:**

y = 20 # Global variable

def my\_function():

print("Inside the function, y is:", y)

my\_function()

print("Outside the function, y is:", y)

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

**Ans 6:** In Python, the return statement can be used to return multiple values from a function by returning them as a tuple, list, dictionary, or any other iterable object. When multiple values are returned using return, they are packed into a single object and can be unpacked by the caller.

**For example:**

def calculate(x, y): # Perform some calculations

sum\_result = x + y

difference\_result = x - y

product\_result = x \* y

return sum\_result, difference\_result, product\_result # Call the function

result1, result2, result3 = calculate(5, 3)

print("Sum:", result1) # Output: Sum: 8

print("Difference:", result2) # Output: Difference: 2

print("Product:", result3) # Output: Product: 15

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

**comes to function arguments in Python?**

**Ans 7:** In Python, the terms "pass by value" and "pass by reference" are often used informally to describe how arguments are passed to functions.

**Pass by Value**:

* In pass by value, a copy of the value of the actual parameter (the argument passed to the function) is made and passed to the function.
* The function works with this copy, and any modifications made to the parameter inside the function do not affect the original value outside the function.
* Immutable objects like integers, string, tuples are passed by value-like behavior. When you pass an immutable object to a function, a copy of the object's value is passed. Any modifications made to the parameter inside the function do not affect the original object outside the function.

**Pass by Reference**:

* In pass by reference, instead of making a copy of the value, a reference to the original object is passed to the function.
* Any modifications made to the parameter inside the function affect the original object outside the function as well.
* Mutable objects like List, dictionaries are passed by reference-like behavior. When you pass a mutable object to a function, a reference to the original object is passed. Any modifications made to the parameter inside the function affect the original object outside the function.

**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 (2 x )**

**d. Square root**

**Ans 8:**

import math

def math\_operations(x):

result\_log = math.log(x) # Logarithmic function (log x)

result\_exp = math.exp(x) # Exponential function (exp(x))

result\_power = math.pow(2, x) # Power function with base 2 (2^x)

result\_sqrt = math.sqrt(x) # Square root

return result\_log, result\_exp, result\_power, result\_sqrt # Test the function with an integer value

integer\_value = 4

results\_integer = math\_operations(integer\_value)

print("Integer value:", integer\_value)

print("Logarithmic function:", results\_integer[0])

print("Exponential function:", results\_integer[1])

print("Power function with base 2:", results\_integer[2])

print("Square root:", results\_integer[3])

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

**Ans 9 :**

def split\_name(full\_name): # Split the full name into first name and last name

names = full\_name.split() # Extract the first name and last name

first\_name = names[0]

last\_name = ' '.join(names[1:]) if len(names) > 1 else ''

return first\_name, last\_name # Test the function

full\_name = "John Doe"

first\_name, last\_name = split\_name(full\_name)

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

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