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

**Ans .**

Built-in Function vs. User-defined Function in Python:

Built-in Function: Pre-defined by Python. Ready to use without imports.

Example: len("Hello") returns 5.

User-defined Function: Created by programmers for specific tasks.

Example:

def add(a, b):

return a + b

result = add(3, 5)

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

**Ans.**

Passing Arguments to a Function in Python:

To pass arguments to a function in Python, you provide values within the parentheses when calling the function. The function then uses these values to perform its tasks.

Positional Arguments:

Arguments passed based on their position in the function call.

Order matters.

No explicit names used when passing.

Example: print("Hello", "world")

Keyword Arguments:

Arguments passed with their corresponding parameter names.

Order doesn't matter.

Provides clarity and flexibility.

Example: print(end="\n", sep="-")

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

**Ans.**

Purpose of the Return Statement in a Function: The return statement in a function is used to send a value back to the caller of the function. It's used to provide the result of the function's computation or operation.

Multiple Return Statements in a Function: Yes, a function can have multiple return statements. However, once a return statement is executed, the function immediately exits, and the value specified in the return statement is returned to the caller.

Example:

def find\_grade(score):

if score >= 90:

return "A"

elif score >= 80:

return "B"

elif score >= 70:

return "C"

else:

return "F"

student\_score = 85

grade = find\_grade(student\_score)

print("Grade:", grade) # Output: Grade: B

In this example, the find\_grade function has multiple return statements based on the score provided. Depending on the score, a different grade is returned to the caller. Once a return statement is executed, the function exits, and the corresponding value is returned.

**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.**

Lambda Functions in Python: Lambda functions, also known as anonymous functions, are small, inline functions defined using the lambda keyword. They are used for simple operations and are not named like regular functions.

Difference from Regular Functions:

* Syntax: Lambda functions have a concise syntax: lambda arguments: expression. Regular functions use the def keyword and a block of code.
* Name: Lambda functions are anonymous and don't have a formal name.
* Scope: Lambda functions are typically used for small, specific tasks, while regular functions can be more complex and versatile.

Example of a Useful Lambda Function:

# Sorting a list of tuples based on the second element using a lambda function

points = [(2, 5), (1, 8), (4, 3), (3, 6)]

points\_sorted = sorted(points, key=lambda x: x[1])

print(points\_sorted) # Output: [(4, 3), (2, 5), (3, 6), (1, 8)]

In this example, a lambda function is used as the key argument in the sorted function to sort a list of tuples based on their second element. This simplifies the code by avoiding the need to define a separate named function for this specific sorting operation.

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

**Ans.**

Scope in Functions - Local and Global:

Local Scope:

* Variables defined within a function are considered to have local scope.
* They can only be accessed and modified within the function.
* They are not accessible outside of the function.
* Helps in encapsulation and prevents unintentional variable interference.

Global Scope:

* Variables defined outside of any function have global scope.
* They can be accessed and modified from anywhere in the code.
* Changes made to global variables within a function are reflected globally.
* Overuse of global variables can lead to code complexity and potential issues.

Example:

global\_var = 10 # Global variable

def my\_function():

local\_var = 5 # Local variable

print(global\_var) # Accessing global variable

print(local\_var) # Accessing local variable

my\_function()

print(global\_var) # Global variable accessible here

#print(local\_var) # Uncommenting this line would result in an error

In this example, global\_var has global scope and can be accessed both within and outside the function. local\_var has local scope and is only accessible within the function. Attempting to access local\_var outside the function would result in an error.

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

**Ans.**

You can use the "return" statement in a Python function to return multiple values by separating them with commas. These values will be packed into a tuple, which can then be unpacked when the function is called.

Example:

def get\_coordinates():

x = 3

y = 5

return x, y # Returning multiple values as a tuple

x\_coord, y\_coord = get\_coordinates() # Unpacking the returned tuple

print("X Coordinate:", x\_coord)

print("Y Coordinate:", y\_coord)

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

**Ans.**

Python uses a "pass by object reference" approach, which can be confusingly similar to both "pass by value" and "pass by reference" but is actually different:

Pass by Value:

In a "pass by value" approach, a copy of the value is passed to the function.

Changes made to the parameter within the function don't affect the original value.

Commonly seen in languages like C or C++.

Pass by Reference:

In a "pass by reference" approach, a reference (memory location) of the variable is passed to the function.

Changes made to the parameter within the function directly affect the original value.

Commonly seen in languages like C++ with explicit references or pointers.

Pass by Object Reference in Python:

Python uses a concept closer to "pass by object reference."

The reference to an object (like a list or dictionary) is passed, not a copy or direct memory location.

Mutable objects can be modified within the function, and the changes are reflected outside the function.

Immutable objects (like integers or strings) can't be modified this way.

Example:

def modify\_list(lst, num):

lst.append(4) # Modifying the list

num = 100 # Modifying the integer parameter

my\_list = [1, 2, 3]

my\_num = 5

modify\_list(my\_list, my\_num)

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

print("Modified Number:", my\_num) # Output: Modified Number: 5

In this example, the list my\_list is modified within the function, and the change persists outside. However, the integer my\_num remains unchanged since integers are immutable.

**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.**

Here's a Python function that takes a numeric value (integer or decimal) as input and performs the specified operations:

import math

def perform\_operations(x):

log\_result = math.log(x)

exp\_result = math.exp(x)

power\_result = 2 \*\* x

sqrt\_result = math.sqrt(x)

results = {

'Logarithmic (log x)': log\_result,

'Exponential (exp(x))': exp\_result,

'Power (2^x)': power\_result,

'Square Root': sqrt\_result

}

return results

# Test the function with different input values

input\_value = 3.0

results = perform\_operations(input\_value)

for operation, result in results.items():

print(f"{operation}: {result}")

Replace the input\_value with the numeric value you want to test. This function uses the math module to perform the mathematical operations. It returns a dictionary containing the results of each operation.

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

**Ans.**

Here's a Python function that takes a full name as input and returns the first name and last name:

def extract\_names(full\_name):

names = full\_name.split()

if len(names) >= 2:

first\_name = names[0]

last\_name = ' '.join(names[1:])

return first\_name, last\_name

else:

return None, None

# Test the function

full\_name = "John Doe"

first\_name, last\_name = extract\_names(full\_name)

if first\_name and last\_name:

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

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

else:

print("Invalid full name format.")

Replace the full\_name with the full name you want to extract the first name and last name from. This function uses the split method to split the full name into individual words and then combines the words after the first one to extract the last name.