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 are those function which already exist in python while user defined function are those function which is defined by the user as per requirement.

For eg: Built in function: math,sqrt,len

User defined function: def abc(para):

1. 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 by specifying them within the parentheses when defining or calling the function. There are two main ways to pass arguments: positional arguments and keyword arguments.

Positional argument: Positional arguments are passed to a function based on their position or order. When defining the function, you specify the parameters in the same order as you want the arguments to be passed. When calling the function, you provide the values for these arguments in the same order

def greet(name, age):

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

greet("Alice", 25)

In the example above, the function greet has two positional arguments: name and age. When we call the function greet("Alice", 25), the value "Alice" is assigned to the name parameter, and 25 is assigned to the age parameter.

Keyword Arguments: Keyword arguments are passed to a function based on their parameter names.

def greet(name, age):

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

greet(age=25, name="Alice")

In the updated example, we used keyword arguments to pass the values. The order of the arguments doesn't matter as long as we explicitly mention the parameter names and their values.

1. 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 should be sent back to the caller of the function. It marks the end of the function's execution and returns the specified value to the point where the function was called. The purpose of the return statement is to provide the output or result of the function's computation.

Yes, a function can have multiple return statements

def calculate\_grade(score):

if score >= 90:

return "A"

elif score >= 80:

return "B"

elif score >= 70:

return "C"

elif score >= 60:

return "D"

else:

return "F"

student\_score = 85

grade = calculate\_grade(student\_score)

print("Grade:", grade)

1. 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 are anonymous functions that are defined using the lambda keyword. They are called anonymous because they don't have a formal name like regular functions defined using the def keyword. Lambda functions are often used when a small, one-line function is needed for a specific task and defining a regular function seems unnecessary.

The syntax of a lambda function is as follows: lambda arguments: expression

Lambda functions are anonymous and don't require a formal name.

Lambda functions can only contain a single expression, whereas regular functions can include multiple statements and have a block of code.

Lambda functions are typically used for simple, one-line operations, while regular functions are used for more complex tasks and code reuse.

For eg:

Add=lambda x,y:x+y

N1=int(input(“enter the number:”))

N2= int(input(“enter the number:”))

Print(Add(N1,N2))

1. 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, functions, and other identifiers within a specific region of a program. It determines where and how variables can be accessed, modified, or referenced.

When it comes to functions in Python, each function has its own scope, known as the local scope. The local scope is created when a function is called and is destroyed when the function completes execution. Within the local scope of a function, you can define variables that are only accessible within that function. These variables are called local variables. Local variables are not accessible outside the function and are "forgotten" once the function finishes executing.

def my\_function():

x = 10 # Local variable

print(x)

my\_function() # Output: 10

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

In the above example, the variable x is defined within the my\_function() function. It is accessible and can be used within the function, but outside of the function, attempting to access x will result in a NameError because it is not defined in the global scope.

On the other hand, the global scope refers to the outermost scope of a Python program or module. Variables defined outside any function or class, at the top level of the module, have global scope. Global variables can be accessed and modified from anywhere within the module, including inside functions.

x = 10 # Global variable

def my\_function():

print(x)

my\_function() # Output: 10

print(x) # Output: 10

In this case, x is a global variable. It can be accessed and printed both inside the my\_function() and outside the function, as it is defined in the global scope.

It's important to note that local variables within a function take precedence over global variables with the same name. If a local variable shares the same name as a global variable, the local variable is considered to be in the local scope and shadows the global variable within the function.

x = 10 # Global variable

def my\_function():

x = 20 # Local variable

print(x)

my\_function() # Output: 20

print(x) # Output: 10

In this case, when my\_function() is called, it prints the value of the local variable x, which is 20. The global variable x remains unaffected by the local variable and retains its original value of 10.

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

Ans: In Python, you can use the "return" statement in a function to return multiple values by separating them with commas

def return\_multiple\_values():

value1 = 10

value2 = "Hello"

value3 = [1, 2, 3]

return value1, value2, value3

result = return\_multiple\_values()

print(result) # Output: (10, 'Hello', [1, 2, 3])

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

Ans: Pass by value (immutable objects): When you pass immutable objects like numbers, strings, and tuples as function arguments, they are effectively passed by value.

def modify\_value(x):

x += 1

print("Inside the function:", x)

num = 5

modify\_value(num)

print("Outside the function:", num)

output:

Inside the function: 6

Outside the function: 5

Pass by reference (mutable objects): When you pass mutable objects like lists, dictionaries, or user-defined objects as function arguments, they are effectively passed by reference.

def modify\_list(lst):

lst.append(4)

print("Inside the function:", lst)

my\_list = [1, 2, 3]

modify\_list(my\_list)

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

output:

Inside the function: [1, 2, 3, 4]

Outside the function: [1, 2, 3, 4]

1. Create a function that can intake integer or decimal value and do following operations:
   1. Logarithmic function (log x)
   2. Exponential function (exp(x))
   3. Power function with base 2 (2x)
   4. Square root

Ans: import math

def logarithms\_a(x):

result = math.log(x)

print(result)

logarithms\_a(10)

b) import math

def logarithms\_a(x):

result=math.exp(x)

print(result)

logarithms\_a(10)

c) import math

x = 3

result = math.pow(2, x)

print(result)

d) import math

number = float(input("Enter a number: "))

result = math.sqrt(number)

print("The square root of", number, "is", result)

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

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

name = "John Doe"

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

print("First name:", first)

print("Last name:", last)

Output:

First name: John

Last name: Doe