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 functions in Python are functions that are already predefined in the Python programming language. They are readily available for use without the need for additional import statements. These functions provide basic functionality and can be used in various contexts. Examples of built-in functions in Python include `print() `len() `type() and `max()`.

Input: -

print("Hello world!")

Output: -

Hello world!

User-defined functions on the other hand are functions that are created by the users themselves. These functions are defined using the `def` keyword followed by a function name and a block of code that specifies what the function does. User-defined functions allow users to define reusable blocks of code that can be called multiple times throughout a program.

Input: -

def square(number):

return number \*\* 2

result = square(5)

print(result)

Output: -

25

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

Ans. Positional Arguments: Positional arguments are passed to a function based on their position or order. The function defines parameters and the arguments are passed in the same order when calling the function. The number and order of the arguments must match the parameters specified in the function definition.

Input: -

def greet(name age):

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

greet("Alice”, “25”)

Output: -

Hello Alice! You are 25 years old.

Keyword Arguments: Keyword arguments are passed to a function based on the parameter names. In this case the arguments are not dependent on their position but rather identified by their parameter names. This allows you to pass only the desired arguments without the need to pass all the arguments.

Input: -

def greet(name age):

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

greet(name="Alice age=25)

Output: -

Hello Alice! You are 25 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.

Ans. The purpose of the return statement in a function is to specify the value that the function should output or return when it is called. It is used to terminate the execution of the function and provide the result back to the caller. Yes a function can have multiple return statements. Once a return statement is encountered the function execution will stop and the specified value will be returned. This allows the function to have different return paths based on certain conditions.

Input: -

def maximum(a b):

if a > b:

return a

else:

return b

print(maximum(5 10))

Output: -

10

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 can be defined without a name. They are typically used when a small one-time function is needed without the need to define a separate function explicitly. Lambda functions can take any number of arguments but can only have a single expression. They are different from regular functions as they do not require a return statement cannot contain any statements or complex logic and are typically written in a single line.

Input: -

numbers = [5 -2 10 -8 3]

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

print(sorted\_numbers)

Output: -

[-2 3 5 -8 10]

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 region within a program where a variable or name is accessible. Local scope refers to variables that are defined within a specific function. These variables can only be accessed and modified within that function. Global scope on the other hand refers to variables that are defined outside of any function making them accessible from anywhere within the program. The main difference between local and global scope is their visibility. Local variables are only visible within the function they are defined in while global variables can be accessed from any part of the program.

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

Ans. the "return" statement in a function is used to specify the value(s) that will be passed back to the caller of the function. Although it may seem that "return" can only return a single value it can actually be used to return multiple values in the form of tuples lists or any other iterable data structure. To return multiple values simply separate the values with commas in the "return" statement. For example you can write:

def get\_coordinates():

x = 5

y = 10

return x y

In this example the function `get\_coordinates()` returns two values `x` and `y as a tuple `(5 10)`. The caller of the function can then use tuple unpacking to access these values separately.

x\_coord y\_coord = get\_coordinates()

Now `x\_coord` will be assigned the value 5 and `y\_coord` will be assigned the value 10.

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

Ans. In Python function arguments are passed by assignment. However it's essential to understand the difference between "pass by value" and "pass by reference" in order to grasp this concept. In "pass by value the function creates a local copy of the argument value. Any modifications made to the argument inside the function won't affect the original value outside the function. This is the case for immutable objects like numbers and strings. In contrast in "pass by reference the function receives a reference to the original object. Any modifications made to the argument inside the function will directly affect the original value outside the function. This applies to mutable objects like lists and dictionaries.

So although Python behaves like "pass by value" for immutable objects it's more accurate to say that Python uses "pass by object reference."

1. 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 (2x)

d. Square root

Ans.

Input: -

import math

def perform\_operations(x):

result = {}

result['log'] = math.log(x) # Logarithmic function (log x)

result['exp'] = math.exp(x) # Exponential function (exp(x))

result['power'] = 2 \*\* x # Power function with base 2 (2^x)

result['sqrt'] = math.sqrt(x) # Square root

return result

input\_value = 5

output = perform\_operations(input\_value)

print(output)

Output: -

'log': 1.6094379124341003

'exp': 148.4131591025766

'power': 32

'sqrt': 2.23606797749979

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

Ans. The `split()` method returns a list of strings where each string represents an individual word. By convention the first word in the list would be considered the first name and the last word would be considered the last name.

Input: -

def get\_first\_and\_last\_name(full\_name):

names = full\_name.split()

first\_name = names[0]

last\_name = names[-1] # -1 represents the last element in a list

return first\_name last\_name

name = "John Doe"

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

print("First Name”, first\_name)

print("Last Name”, last\_name)

Output: -

First Name: John

Last Name: Doe