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

ANS: A built in function is already available in Python such as print function, on the other hand user defined functions are created by programmer for code reusability and to reduce the number of code lines.

**Built-in-function:**

Text = “hello world”

print(Text) #Output: hello world

**User-defined -function:**

def add\_number(x, y):

return x + y

result = add\_number(5, 5)

print(result) # Output: 10

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

Ans: You can pass arguments to a function in python by specifying them inside the parenthesis while calling the function.

Please find below code to understand the difference between positional arguments and keyword arguments:

**positional arguments:**

def greet(name, age):

print(f“Hello {name}, you are {age} years old.”)

greet(“Arbaj”,23)

#Output: Hello Arbaj, you are 23 years old.

**keyword arguments:**

def greet(name, age):

print(f“Hello {name}, you are {age} years old.”)

greet(age=23, name=“Arbaj”)

#Output: Hello Arbaj, you are 23 years old.

Q.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 send a value back to caller of the function. It allows the function to compute result and send back to function.

Yes, function can have multiple return statement:

e.g. –

def check\_number(number):

if number > 0:

return “Positive number”

elif number < 0:

return “Negative number”

else:

return “Zero”

Q.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 are defined by lambda keyword and it does not require separate def statement, lambda functions are typically use for short calculations.

**Regular function:**

def square(x):

return x \*\* 2

result = square(5)

print(result) # Output: 25

**Lambda functions:**

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

result = square\_lambda(5)

print(result) #Output: 25

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

Ans: In python Scope refers to the visibility of the variable within different parts of your code. There are mainly two scopes in python:

1. Local scope:

Variable defined inside function are in local variable and they can only be accessed and modified within that specific function. Once function completes the execution the local variable are destroyed.

def my\_function:

x = 10

print(x)

my\_function( ) # Output: 10

1. Global scope: Variables defined outside any function are global variable and they are accessible through out the programme, they can be accessed and modified from any part of code including inside the function.

y = 10

def my\_function:

print(y) # Accessible inside the function

my\_function( ) # Output: 10

print(y) # Accessible outside the function

Q.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 by comma.

e.g –

def calculate(a, b):

sum\_result = a +b

diff\_result = a -b

product\_result = a \* b

return sum\_result, diff\_result, product\_result

# we also use different methods:

Using list:

return [sum\_result, diff\_result, product\_result]

Using dictionary:

return {“sum”: sum\_result, “difference” : diff\_result, “Product”: product\_result}

Q.7) 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:**

When you pass arguments by value to a function, a copy of the variable's value is made, and this copy is used inside the function and changes made to the parameter inside the function do not affect the original variable outside the function.

e.g.

def modify\_value(x):

x += 5

print("Inside function:", x)

y = 10

modify\_value(y)

print("Outside function:", y)

# Output: Outside function: 10

**Pass by reference:**

Python passes references to objects. If the object is mutable (like lists), changes inside the function affect the original object outside the function. Immutable objects (like numbers) behave as pass by value.

e.g.

def modify\_list(lst):

lst.append(4)

print("Inside function:", lst)

my\_list = [1, 2, 3]

modify\_list(my\_list)

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

# Output: Outside function: [1, 2, 3, 4]

Q.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 (2x)

d. square root

Ans:

import math

def math\_operations(value):

# Logarithmic function (log x)

logarithm\_result = math.log(value)

# Exponential function (exp(x))

exponential\_result = math.exp(value)

# Power function with base 2 (2^x)

power\_result = math.pow(2, value)

# Square root

square\_root\_result = math.sqrt(value)

return logarithm\_result, exponential\_result, power\_result, square\_root\_result

math\_operations(4)

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

Ans:

def extract\_names(full\_name):

names\_list = full\_name.list()

first\_name = names\_list[0]

last\_name = names\_list[-1]

return first\_name, last\_name

# Example usage

full\_name = “Arbaj Khan”

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

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

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