Assessment -6

Part A: Function Definitions and Calls

1. Define a function named greet_user that takes a user's name as an argument and prints a greeting message. Write a sample function call to demonstrate its usage.

```
def greet_user(name):
    print("Hello " + name)
greet_user("Aman")
greet_user("Kriti")
```

2. Create a function calculate_area that accepts the radius of a circle as an argument and returns the area. Use the formula area=π×radius2\text{area} = \pi \times \text{radius}^2area=π×radius2 (Hint: You may use the math module to import the value of π).

```
import math
def calculate_area(r):
    area = math.pi*r*r
    print(area)

r = float(input("Enter the radius of circle: "))
calculate_area(r)
```

3. What is the purpose of a return statement in a function? Explain with an example.

   The return statement in a function is used to send a value back to the caller. It serves two primary purposes:

   Return a Value: It allows the function to produce a result that can be used in other parts of the program.

   End the Function Execution: Once a return statement is executed, the function stops further execution.

   value of π).

```
import math
def calculate_area(r):
    area = math.pi*r*r
    print(area)

r = float(input("Enter the radius of circle: "))
calculate_area(r)
```

Part B: Function Arguments and Return Values

4. Write a function find_maximum that takes three numbers as parameters and returns

the largest of the three. Demonstrate the function with an example.

```
def find_maximum(n1,n2,n3):
    return max(n1,n2,n3)

n1 = int(input("Enter first number:"))
n2 = int(input("Enter second number:"))
n3 = int(input("Enter third number:"))
print("The maximul number is: ",find_maximum(n1,n2,n3))
```

5. Explain positional and keyword arguments in Python functions. Provide a code example that includes both types of arguments.

```
'''
Positional Arguments
Arguments are passed in the order in which they are defined in the func-
tion.
The function matches arguments to parameters based on their position.
Keyword Arguments
Arguments are passed by explicitly specifying the parameter name along
with its value.
This allows arguments to be passed in any order.
'''
def describe_person(name, age, city="Unknown"):
    print(f"Name: {name}")
    print(f"Age: {age}")
    print(f"City: {city}")

# Using positional arguments
describe_person("Aman", 20)

# Using only keyword arguments
describe_person(name="Rahul", city="Mumbai", age=30)
```

6. Create a function personal_info that takes a name and age as positional arguments and an optional keyword argument city. If the city argument is not provided, the function should assume the city is "Unknown". Write sample calls for each possible way of calling this function.

```
def personal_info(name, age, city="Unknown"):
    print(f"Name: {name}")
    print(f"Age: {age}")
    print(f"City: {city}")

# Using positional arguments
personal_info("Aman", 20)
#mixing both
personal_info(name="Rahul", age=30, city="Mumbai")
#using keyword
personal_info(name="Rahul", city="Mumbai", age=30)
```

7. Write a function sum_of_squares that accepts any number of arguments and returns the sum of the squares of each argument. Demonstrate how you would call this

function with varying numbers of arguments.

```
def sum_of_square(*n):
    total = 0
    for i in n:
        total += i**2
    return total
n = int(input("Enter the number: "))
print(sum_of_square(1,2,3))
print(sum_of_square(2,4))
```

Part C: Lambda Functions

8. Define a lambda function that takes two arguments, x and y, and returns their product. Call this lambda function with sample values.

```
product = lambda x, y: x * y
result = product(5, 3)
print("The product of 5 and 3 is:", result)
```

9. Write a list of integers, and use a lambda function with the filter function to filter out only the even numbers from the list.

```
my_list = [2,3,4,5,6]
even_number = list(filter(lambda x:x%2==0,my_list))
print(even_number)
```

10. Create a lambda function to sort a list of dictionaries by a specified key. For example, given students = [{"name": "Alice", "age": 24}, {"name": "Bob", "age": 22}, {"name": "Charlie", "age": 23}], sort the list by age using the lambda function.

```
students = [
    {"name": "Alice", "age": 24},
    {"name": "Bob", "age": 22},
    {"name": "Charlie", "age": 23}
]

sorted_students = sorted(students, key=lambda student: student["age"])

print("Sorted by age:", sorted_students)
```

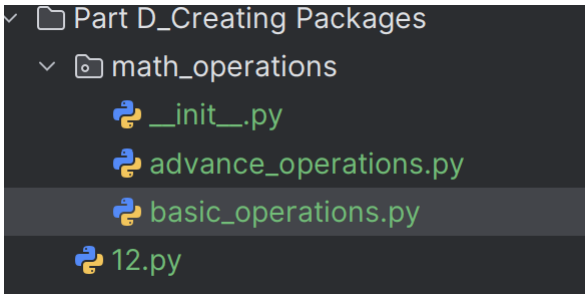
Part D: Creating Packages

11. Explain the purpose of an __init__.py file in a package. Why is it necessary?

The __init__.py file is a special Python script used to mark a directory as a package. It allows package initialization, executes code when the package is imported, and controls the visibility of modules and functions in the package. While optional in Python 3.3 and later, it is essential

for organizing and managing package behavior effectively.

12. Create a simple package structure for a library called math_operations with two modules: basic_operations and advanced_operations. Include sample functions in each module, such as add and multiply in basic_operations, and power and factorial in advanced_operations. Provide code snippets for defining these modules.




```
#basic.operations.py……
def add(a, b):
    return a + b

def multiply(a, b):
    return a * b


#advance_operations.py------------------------------
def power(base, exponent):
    return base ** exponent

def factorial(n):
    if n == 0 or n == 1:
        return 1
    fact = 1
    for i in range(2, n + 1):
        fact *= i
    return fact


# main.py…….
from math_operations import add, multiply, power, factorial

print(add(5, 3))
print(multiply(5, 3))
print(power(2, 3))
print(factorial(5))
```

13. After creating the math_operations package, explain how to install the package locally and import the basic_operations module in a Python script. Provide an

example script that calls the add function from basic_operations.

```
math_operations/
    __init__.py
    basic_operations.py
    advanced_operations.py
setup.py
```

```
# setup.py
from setuptools import setup, find_packages

setup(
    name='math_operations',
    version='1.0',
    packages=find_packages(),
    description='A simple library for basic and advanced mathematical operations',
    author='Your Name',
    author_email='your.email@example.com',
)
```

```
pip install -e .
```

```
# example_script.py
from math_operations.basic_operations import add

# Use the add function
result = add(10, 5)
print(f"The sum of 10 and 5 is: {result}")
```

Part E: Importing and Using Modules

14. What is the difference between import module_name and from module_name import specific_function? Demonstrate with an example using the math module.

1. import module_name

- This imports the entire module.
- You access functions, classes, or variables in the module using the dot notation (module_name.function_name).
- Useful when you need to use multiple functions or want to avoid polluting the namespace.

```
import math

# Access functions with the module prefix
result = math.sqrt(16)  # Square root of 16
pi_value = math.pi      # Value of π


print(f"Square root of 16 is: {result}")  # Output: 4.0
print(f"Value of pi is: {pi_value}")     # Output: 3.141592653589793
```

2. from module_name import specific_function

- This imports specific functions, classes, or variables from the module.
- You can use the imported function directly without the module prefix.

```
from math import sqrt, pi
# Access the imported functions directly
result = sqrt(16)  # Square root of 16
pi_value = pi      # Value of π
print(f"Square root of 16 is: {result}")  # Output: 4.0
print(f"Value of pi is: {pi_value}")     # Output: 3.141592653589793
```

15. Write a Python script that imports the random module and generates a random integer between 1 and 100. Use a function to encapsulate the logic and print the random number.

```
import random

def generate_random_number():
    random_number = random.randint(1, 100)
    print(f"The random number is: {random_number}")

generate_random_number()
```