Experiment No. 4
Creating functions, classes and objects using python
Date of Performance:
Date of Submission:

Experiment No. 4

Title: Creating functions, classes and objects using python

Aim: To study and create functions, classes and objects using python

Objective: To introduce functions, classes and objects in python

Theory:

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state.

To understand the need for creating a class let's consider an example, let's say you wanted to track the number of dogs that may have different attributes like breed, age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it's the exact need for classes.

Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

CODE:

```
# Function
def greet(name):
    """Function to greet a person."""
    print("Hello,", name)

# Class
class Dog:
    """A simple class representing a dog."""

    def __init__(self, name, breed):
        """Constructor to initialize dog's name and breed."""
        self.name = name
        self.breed = breed

    def bark(self):
        """Method to make the dog bark."""
        print(self.name, "says: Woof!")

# Creating objects
dog1 = Dog("Buddy", "Labrador")
dog2 = Dog("Max", "German Shepherd")
```

```
# Function call
greet("Alice")


# Object method call
dog1.bark()
dog2.bark()
```

RESULTS:

```
Hello, Alice
Buddy says: Woof!
Max says: Woof!

=== Code Execution Successful ===
```

Conclusion: The implementation of classes, objects, and functions in Python offers a powerful foundation for structuring and organizing code, promoting modularity, and enhancing code reuse. Classes provide blueprints for creating objects with attributes and methods, encapsulating data and behavior within a single unit. Objects instantiated from classes enable the instantiation of multiple instances with unique characteristics, facilitating the modeling of real-world entities and interactions. Functions, on the other hand, encapsulate reusable blocks of code, enhancing code readability and maintainability. Together, these features empower developers to build scalable, maintainable, and efficient Python applications across various domains and use cases.