|  |
| --- |
| Experiment No. 4 |
| Creating functions, classes and objects using python |
| Name: Krisha Chikka |
| Roll no. : 31 SE/Div1 |
| Date of Performance: 05/02/2024 |
| Date of Submission: 12/02/2024 |

**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:**class Student:

def \_\_init\_\_(self, name, age, grade):

self.name = name

self.age = age

self.grade = grade

def study(self):

print(f"{self.name} is studying hard.")

def get\_grade(self):

return self.grade

# Creating objects of the Student class

stud1 = Student("Cherry", 19, "A")

stud2 = Student("Veda", 20, "B")

# Accessing object attributes

print(f"{stud1.name} is {stud1.age} years old and got a grade of {stud1.get\_grade()}.")

print(f"{stud2.name} is {stud2.age} years old and got a grade of {stud2.get\_grade()}.")

# Calling object methods

stud1.study()

stud2.study()

**Output:**

Cherry is 19 years old and got a grade of A.

Veda is 20 years old and got a grade of B.

Cherry is studying hard.

Veda is studying hard.

**Conclusion:**

In this experiment, we delved into creating classes and objects in Python. Classes serve as blueprints for creating objects, bundling data and functionality together. We defined a `Student` class with attributes like name, age, and grade, along with methods like `study()` and `get\_grade()`. By instantiating objects of this class, such as `stud1` and `stud2`, we were able to access their attributes and call their methods. This demonstrates how classes and objects organize and encapsulate data and behavior, promoting code reusability and maintainability in Python programs.