CSE209 – Programming in Python 23CS043

Practical 4

|  |  |  |
| --- | --- | --- |
| **4.1** | | |
| **Aim:** Class, Objects and Inheritance:   Create a class with attributes and methods.   Instantiate an object from a class.   Access and modify the attributes of an object.   Call methods defined in a class using an object.   Access and modify the attributes of an object using getter and setter methods.   Create a subclass that inherits from a superclass.   Override methods in a subclass to provide specific implementations.   Call methods from the superclass using the super() function.   Define and use class variables that are shared among all instances of a class.   Define and use instance variables that are unique to each object.   Create a class that inherit from multiple superclass   Understand and use the method resolution order to determine the order in which base classes are searched. | | |
| **Code:**  *# Class, Objects, and Inheritance Example*  *# 1. Create a class with attributes and methods*  class Animal:  *# Class variable shared among all instances*  species = "Animal"  def \_\_init\_\_(self, name, age):  *# Instance variables unique to each object*  self.name = name  self.age = age  *# Method*  def make\_sound(self):  return "Some generic sound"  *# Getter method*  def get\_name(self):  return self.name  *# Setter method*  def set\_name(self, name):  self.name = name  *# 2. Instantiate an object from a class*  animal = Animal("Buddy", 5)  *# 3. Access and modify the attributes of an object*  print(f"Initial name: {animal.name}")  animal.name = "Max"  print(f"Modified name: {animal.name}")  *# 4. Call methods defined in a class using an object*  print(animal.make\_sound())  *# 5. Access and modify the attributes of an object using getter and setter methods*  print(f"Name using getter: {animal.get\_name()}")  animal.set\_name("Charlie")  print(f"Name after setter: {animal.get\_name()}")  *# 6. Create a subclass that inherits from a superclass*  class Dog(Animal):  def \_\_init\_\_(self, name, age, breed):  *# Call the superclass's \_\_init\_\_ method*  super().\_\_init\_\_(name, age)  self.breed = breed  *# 7. Override methods in a subclass to provide specific implementations*  def make\_sound(self):  return "Woof!"  *# 8. Call methods from the superclass using the super() function*  def get\_species(self):  return super().species  *# 9. Define and use class variables that are shared among all instances of a class*  print(f"Animal species: {Animal.species}")  *# 10. Define and use instance variables that are unique to each object*  dog = Dog("Rex", 3, "Golden Retriever")  print(f"Dog name: {dog.name}, age: {dog.age}, breed: {dog.breed}")  print(dog.make\_sound())  print(f"Dog species: {dog.get\_species()}")  *# 11. Create a class that inherits from multiple superclasses*  class Bird:  def \_\_init\_\_(self, can\_fly):  self.can\_fly = can\_fly  def make\_sound(self):  return "Chirp!"  class Parrot(Animal, Bird):  def \_\_init\_\_(self, name, age, can\_fly):  Animal.\_\_init\_\_(self, name, age)  Bird.\_\_init\_\_(self, can\_fly)  *# Override method*  def make\_sound(self):  return "Squawk!"  *# 12. Understand and use the method resolution order to determine the order in which base classes are searched*  parrot = Parrot("Polly", 2, True)  print(f"Parrot sound: {parrot.make\_sound()}")  *# Check Method Resolution Order (MRO)*  print(Parrot.mro())  **Output Screenshot:** | | |
| **Conclusion/Summary:**  This practical exercise demonstrated essential object-oriented programming (OOP) concepts in Python, including classes, objects, inheritance, method overriding, getters/setters, class and instance variables, multiple inheritance, and method resolution order (MRO). By creating classes like `Animal`, `Dog`, `Bird`, and `Parrot`, we learned how to:  1. Define classes, instantiate objects, and manipulate attributes.  2. Use getters/setters for encapsulation.  3. Implement inheritance and override methods in subclasses.  4. Use `super()` to call superclass methods.  5. Differentiate between class and instance variables.  6. Handle multiple inheritance and understand MRO.  These concepts are fundamental for writing reusable, maintainable, and scalable code. This practical provides a solid foundation for applying OOP principles in real-world Python projects. | | |
| **Student Signature & Date** | **Marks:** | **Evaluator Signature & Date** |