Assignment 18

**TRADITIONAL OOPS PART 2**

**RESEARCH WORK**

**Q1: What is encapsulation, and how does using private attributes prevent misuse?  
Ans: Encapsulation** or data hiding is bundling of data and methods into a single class. It actually hides the process and provides restriction to some class’s attributes to avoid interference.

**Private Attributes** use name mangling (\_\_attr) to prevent misuse of attributes as name mangling makes it difficult to access the attribute.

**EXAMPLE:**

class Bank:

    def \_\_init\_\_(self, balance):

        self.\_balance = balance

        self.\_\_pin = "1234"

b1 = Bank(12200)

b1.\_balance

b1.\_\_pin

**Q2: Explain the concept of method resolution order (MRO) in multiple inheritance with an example.**

**Ans: Method Resolution Order (MRO)** is the method in python that returns a list of classes in the way they were searched for attributes or methods in a linearized way. In multiple inheritance, mro() method returns a list of classes in which subclasses come before parent classes and no class is visited more than once.

**EXAMPLE:**

class A:

  def letter(self):

    return "A"

class B:

  def letter(self):

    return "B"

class C(A, B):

  pass

c1 = C()

print(c1.letter())

print(C.mro())

**Q3: How does polymorphism facilitate flexible function design across different classes?**

**Ans: Polymorphism** facilitate flexible function design across different classes because in polymorphism a function can be used in different ways which provides flexibility across classes and the function with the same name can be used in different classes and at the end we can call the correct function with a single call.

**EXAMPLE:**

class Cat:

  def speak(self):

    return "Meow!"

class Dog:

  def speak(self):

    return "Waoo!"

def make\_speak(thing):

  return thing.speak()

c1 = Cat()

d1 = Dog()

print(make\_speak(c1))

print(make\_speak(d1))

**Q4: What are the benefits of utilizing super() in inheritance for code maintainability?**

**Ans: super()** function in inheritance is very useful because it is used to inherit methods or attributes from the parent class. It avoids code rewriting in the child class which can cause code duplication and other problems.

**EXAMPLE:**

class Employee:

  def \_\_init\_\_(self, salary):

    self.salary = salary

  def display\_salary(self):

    return self.salary

  def \_\_eq\_\_(self, other):

    if not isinstance(other, Employee):

      print("Wrong type!")

      return NotImplemented

    return self.salary == other.salary

class Manager(Employee):

  def display\_salary(self):

    base\_salary = super().display\_salary()

    bonus = base\_salary \* 0.10

    final\_salary = base\_salary + bonus

    return f"Manager Salary (with 10% bouns): {final\_salary}"

**Q5: How does operator overloading improve readability? Provide a real-world example in code.**

**Ans: Operator Overloading** improves readability in code as we can customize the function of an operator in the specific class in very simple way. We don’t have to right lengthy code to get the desired result, just one time we customize the function of operator we can use it any time within the class.

**EXAMPLE:**

class Employee:

  def \_\_init\_\_(self, salary):

    self.salary = salary

  def display\_salary(self):

    return self.salary

  def \_\_eq\_\_(self, other):

    if not isinstance(other, Employee):

      print("Wrong type!")

      return NotImplemented

    return self.salary == other.salary

In this example we can easily compare the salaries of the employees with the help of operator overloading.