**CODING TASK – 20-06-2025**

**Q1. Understanding Access Specifiers**

**Create a class Student with public, protected, and private attributes, and implement the required methods.**

class Student:

def \_\_init\_\_(self, name, roll\_number, marks):

self.name = name # Public

self.\_roll\_number = roll\_number # Protected

self.\_\_marks = marks # Private

def display\_details(self):

print(f"Name: {self.name}")

print(f"Roll Number: {self.\_roll\_number}")

print(f"Marks: {self.\_\_marks}")

def \_update\_roll\_number(self, new\_roll):

self.\_roll\_number = new\_roll

print(f"Updated Roll Number: {self.\_roll\_number}")

def \_\_update\_marks(self, new\_marks):

self.\_\_marks = new\_marks

print(f"Updated Marks: {self.\_\_marks}")

def access\_private\_method(self, new\_marks):

self.\_\_update\_marks(new\_marks)

**Q2. Demonstrate Access**

**Create an object and try accessing/modifying attributes.**

# Create object

student = Student("Lokanya", 101, 95)

# Public: Can access directly

print("\nAccessing Public Attribute:")

student.name = "Lokanya G"

print(student.name)

# Protected: Can access but by convention should be avoided

print("\nAccessing Protected Attribute:")

student.\_roll\_number = 102

print(student.\_roll\_number)

# Private: Direct access will fail

print("\nTrying to Access Private Attribute Directly:")

try:

print(student.\_\_marks)

except AttributeError as e:

print("Error:", e)

**Q3. Inheritance and Access Control**

**Subclass Topper trying to access protected and private members.**

class Topper(Student):

def try\_access(self):

print("\nInside Subclass:")

# Can access protected

print("Accessing protected \_roll\_number:", self.\_roll\_number)

# Cannot access private directly

try:

print("Accessing private \_\_marks:", self.\_\_marks)

except AttributeError as e:

print("Error accessing private attribute:", e)

# Create Topper object

topper = Topper("Anu", 201, 99)

topper.try\_access()

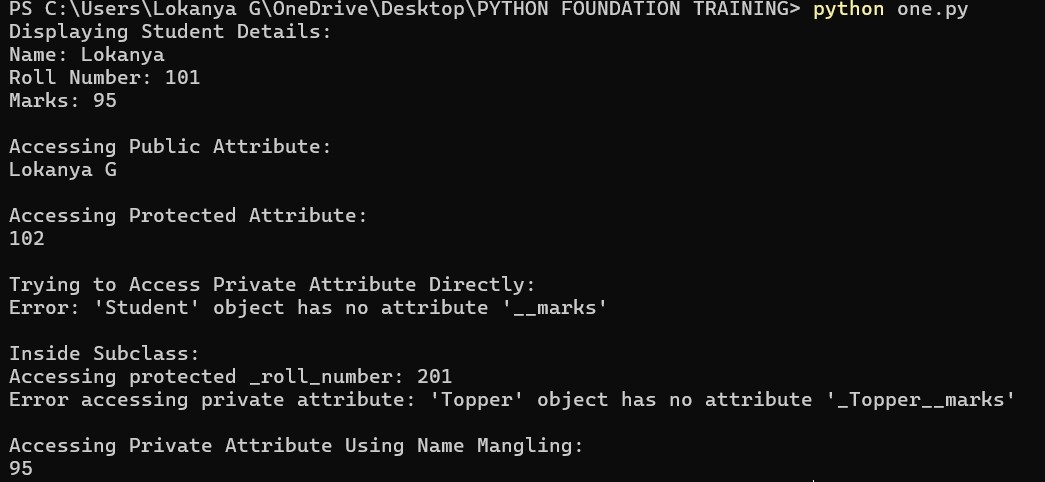
**Q4. Use of Name Mangling**

**Access private attribute \_\_marks using name mangling.**

print("\nAccessing Private Attribute Using Name Mangling:")

print(student.\_Student\_\_marks) # Accessing private variable

**OUTPUT**



**Q5. Reflection**

1. **Why can’t private members be accessed directly?**  
    Private members in Python use double underscores to prevent accidental access and modifications. They are name-mangled to protect internal state and logic.
2. **What is the purpose of using protected members in class design?**  
    Protected members (single underscore) are meant to signal that the attribute is intended for internal use or subclass use, but not for external use directly.
3. **How does name mangling help with private members in Python?**  
    Name mangling adds a prefix \_ClassName to private variables, making it harder (but not impossible) to access them from outside, providing a level of encapsulation.