# DATE:20/06/25

# Python Coding Task

Time: 30 Minutes

Level: Intermediate

## Q1. Understanding Access Specifiers

Create a class `Student` with the following properties:  
  
Class Requirements:  
1. `name` → Public attribute   
2. `\_roll\_number` → Protected attribute   
3. `\_\_marks` → Private attribute   
  
Implement the following methods:  
- Constructor to initialize all attributes.  
- `display\_details()` → Public method to display all attribute values.  
- `\_update\_roll\_number(new\_roll)` → Protected method to update roll number.  
- `\_\_update\_marks(new\_marks)` → Private method to update marks.  
- `access\_private\_method(new\_marks)` → Public method that uses the private method `\_\_update\_marks`.

**CODE:**

class Student:

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

        self.name=name               # Public attribute

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

        self.\_\_marks=marks           # Private attribute

    def details(self):

        print(f"STUDENT DETAILS:")

        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

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

        self.\_\_marks = new\_marks

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

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

student = Student("Bindhu", "5001", 85)

student.details()

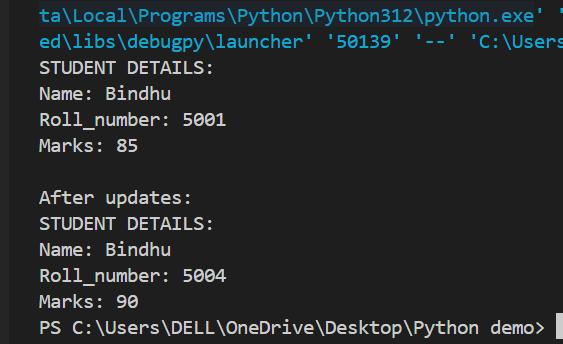
student.\_update\_roll\_number("5004")

student.access\_private\_method(90)

print("\nAfter updates:")

student.details()

**OUTPUT:**



## Q2. Demonstrate Access

In the main section:  
- Create an object of the `Student` class.  
- Modify and print the `name` directly.  
- Modify and print the `\_roll\_number` directly.  
- Try accessing `\_\_marks` directly and observe the result.

**CODE:**

student = Student("Rishi", "5002", 92)

#1

print("INITIAL STUDENT DETAILS:")

student.details()

print("\n")

#2

print("Accessing public name attribute:")

print(f"Original name: {student.name}")

student.name = "Ravi"

print(f"Modified name: {student.name}\n")

#3

print("Accessing protected \_roll\_number attribute:")

print(f"Original roll number: {student.\_roll\_number}")

student.\_roll\_number = "5003"

print(f"Modified roll number: {student.\_roll\_number}\n")

#4

print("\nTrying to access marks directly:")

try:

    print(student.\_\_marks)

except AttributeError:

    print("Failed! Can't access \_\_marks directly")

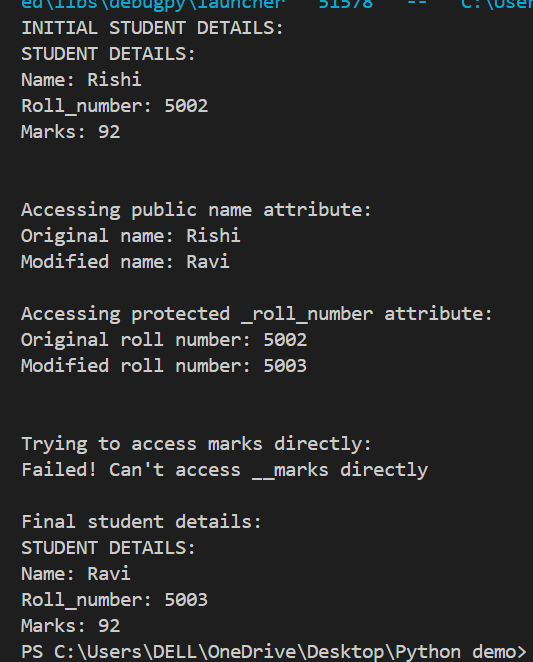
# 5

print("\nFinal student details:")

student.details()

student.details()

**OUTPUT:**



## Q3. Inheritance and Access Control

Create a subclass `Topper` that inherits from `Student` and includes:  
- A method `try\_access()` that attempts to access `\_roll\_number` and `\_\_marks` from the subclass.  
- Show what works and what doesn't.

**CODE:**

class Topper(Student):

    def try\_access(self):

        print("\nTrying to access attributes from Topper subclass:")

        print(f"name: {self.name}")

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

        try:

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

        except AttributeError:

            print("Cannot access \_\_marks directly")

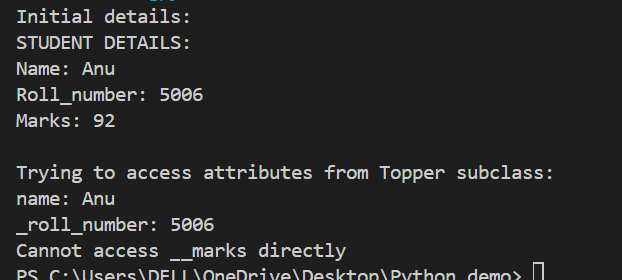
topper = Topper("Anu", "5006", 92)

print("Initial details:")

topper.details()

topper.try\_access()

**OUTPUT:**



## Q4. Use of Name Mangling

Demonstrate how to access the private attribute `\_\_marks` using name mangling technique from outside the class.

**CODE:**

s = Student("Sindhu", "5005", 85)

print("Tying to access private attribute using name mangling:")

print(f"Original marks: {s.\_Student\_\_marks}")

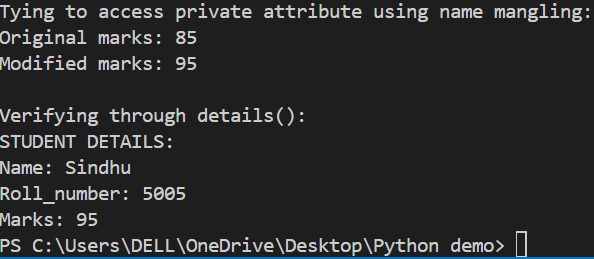
s.\_Student\_\_marks = 95

print(f"Modified marks: {s.\_Student\_\_marks}")

print("\nVerifying through details():")

s.details()

**OUTPUT:**



## Q5. Reflection

Answer the following short questions:  
1. Why can’t private members be accessed directly?

Private members can't be accessed directly because Python intentionally renames them (name mangling) to prevent accidental access from outside the class. This protects internal data that should only be modified by the class's own methods.

2. What is the purpose of using protected members in class design?

Protected members are meant to signal "use within this class or its subclasses only." They’re accessible but serve as a warning to developers to avoid misuse unless extending the class.

3. How does name mangling help with private members in Python?

Name mangling renames private attributes (like \_\_marks to \_Student\_\_marks) to avoid naming conflicts in subclasses and discourage direct access. It’s Python’s way to enforce privacy (though not strictly, as you can still access them if needed).