# 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`.

**SOLUTION**

class Student:

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

        self.name = name

        self.\_roll\_number = roll\_number

        self.\_\_marks = marks

    def display\_details(self):

        print("\n[Q1] 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

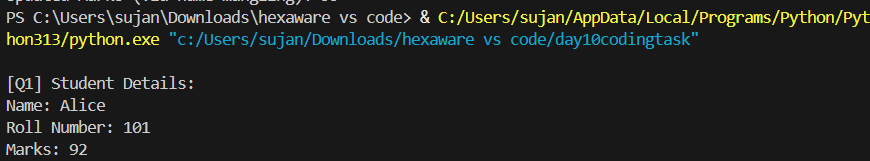
    def access\_private\_method(self, new\_marks):

  self.\_\_update\_marks(new\_marks)

student = Student("Alice", 101, 92)

student.display\_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.

**SOLUTION**

print("\n[Q2] Demonstrating Access to Attributes")

student.name = "Alicia"

print("Updated Name (Public):", student.name)

student.\_roll\_number = 202

print("Updated Roll Number (Protected):", student.\_roll\_number)

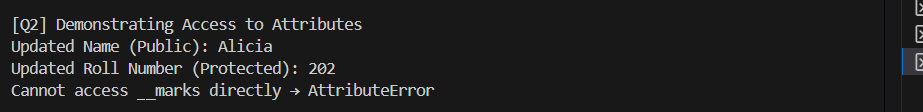
try:

    print("Accessing Marks (Private):", student.\_\_marks)

except AttributeError:

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

**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.

**SOLUTION**

class Topper(Student):

    def try\_access(self):

        print("\n[Q3] Accessing Members from Subclass:")

        print("Accessing \_roll\_number (Protected) from subclass:", self.\_roll\_number)

        try:

            print("Accessing \_\_marks (Private) from subclass:", self.\_\_marks)  # Will raise error

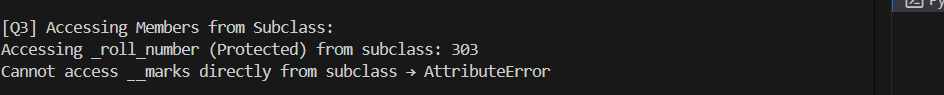
        except AttributeError:

            print("Cannot access \_\_marks directly from subclass → AttributeError")

topper = Topper("Bob", 303, 97)

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.

**SOLUTION**

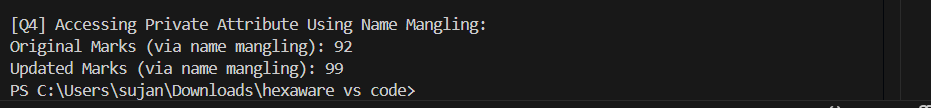
print("\n[Q4] Accessing Private Attribute Using Name Mangling:")

print("Original Marks (via name mangling):", student.\_Student\_\_marks)

student.\_Student\_\_marks = 99

print("Updated Marks (via name mangling):", student.\_Student\_\_marks)

**OUTPUT**

****

## Q5. Reflection

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

In Python, **private members** are defined with **double underscores** (e.g., \_\_marks). This signals that these members are **meant to be hidden** from outside access and only used within the class itself.

Python uses a feature called **name mangling**, where:

self.\_\_marks is internally renamed to: self.\_ClassName\_\_marks → self.\_Student\_\_marks

So, even though Python **does not have strict access control** like Java or C++, this mechanism:

* Helps **prevent accidental modification**
* Encourages **encapsulation** (hiding internal data)

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

Protected members are indicated by a single underscore (e.g., \_roll\_number). This is a convention, not a strict rule, to inform developers that the attribute is intended for use only within the class and its subclasses.

Protected members:

* Help maintain a clean interface for the class
* Allow subclasses to access certain internal attributes or methods
* Avoid unnecessary exposure of internal implementation to external code

They strike a balance between encapsulation and flexibility, especially in inheritance-based designs.

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

Name mangling is Python's way of internally changing the names of private attributes to prevent accidental access. For example:

self.\_\_marks is changed to:

self.\_Student\_\_marks

This approach:

* Makes it harder for external code to accidentally access private members
* Prevents name clashes in subclasses
* Encourages the use of public methods to interact with private data

Name mangling does not make attributes completely inaccessible but helps enforce better design and encapsulation practices.