# Python Coding Task (20-06-2025)

-**BY GETSY JACINTH**

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

        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 (Protected): {self.\_roll\_number}")

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

        self.\_\_marks = new\_marks

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

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

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

s = Student("Alice", 101, 95)

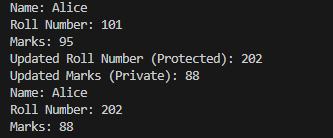
s.display\_details()

s.\_update\_roll\_number(202)

s.access\_private\_method(88)

s.display\_details()

Output screenshot:



## 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:

# Creating object

s = Student("Alice", 101, 95)

# Modify and print public attribute

s.name = "Alicia"

print("Modified Name (Public):", s.name)

# Modify and print protected attribute

s.\_roll\_number = 202

print("Modified Roll Number (Protected):", s.\_roll\_number)

# Try to access private attribute directly

try:

print("Attempting to access \_\_marks directly:", s.\_\_marks)

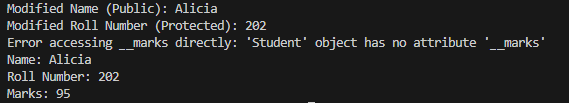
except AttributeError as e:

print("Error accessing \_\_marks directly:", e)

# Display full details using class method

s.display\_details()

output screenshot:



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

super().\_\_init\_\_(name, roll\_number, marks)

def try\_access(self):

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

try:

print("Trying to access Private Attribute from Subclass:", self.\_\_marks)

except AttributeError as e:

print("Error accessing \_\_marks from subclass:", e)

# Create Topper object and call method

t = Topper("Bob", 301, 99)

t.try\_access()

Output Screenshot:



## Q4. Use of Name Mangling

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

Code:

# Access private \_\_marks using name mangling

print("Accessing \_\_marks using name mangling:", s.\_Student\_\_marks)

# Update private \_\_marks using name mangling

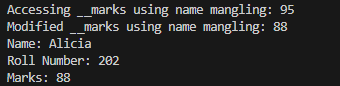
s.\_Student\_\_marks = 88

print("Modified \_\_marks using name mangling:", s.\_Student\_\_marks)

# Confirm with display method

s.display\_details()

Output screenshot:



## Q5. Reflection

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

Answer: Because private members are name-mangled in Python to avoid accidental modification and to enforce encapsulation

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

Answer: Protected members allow access within the class and its subclasses, providing a controlled way to allow extension without exposing the attribute to the outside world.

3. How does name mangling help with private members in Python?  
Answer: It prevents accidental access or overrides by internally renaming private attributes with a pattern \_ClassName\_\_attribute, thus offering a layer of protection without complete restriction.