# 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`.

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"Roll number updated to: {self.\_roll\_number}")

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

self.\_\_marks = new\_marks

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

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

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

## 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.

s = Student("Anas", 101, 88)

s.name = "Aman"

print("Updated Name:", s.name)

s.\_roll\_number = 202

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

## 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.

class Topper(Student):

def try\_access(self):

print("Trying to access 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 in subclass:", e)

t = Topper("Sara", 303, 95)

t.try\_access()

## Q4. Use of Name Mangling

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

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

## Q5. Reflection

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

Private members in Python are **name-mangled** to prevent accidental access or modification, promoting **encapsulation** and data protection.

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

Protected members signal to developers that these attributes/methods are **meant for internal use**, especially by subclasses, but are **not strictly private**.

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

Name mangling renames private members internally to \_ClassName\_\_MemberName, making accidental access harder and helping prevent **name conflicts** in subclasses.