Python Coding Task

1)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
        self._roll_number = roll_number
        self.__marks = marks

    def display_details(self):
        print("Name:", self.name)
        print("Roll Number:", self._roll_number)
        print("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)
```

2)**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("Chandan", 101, 95)

s.name = "Katasani"
print(s.name)

s._roll_number = 202
print(s._roll_number)

try:
    print(s.__marks)  # This will raise an AttributeError
except AttributeError:
    print("Cannot access __marks directly").
```

3)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("Roll Number (protected):", self._roll_number)  # Accessible
        try:
            print("Marks (private):", self.__marks)  # Will raise AttributeError
        except AttributeError:
            print("Cannot access __marks from subclass")
```

4)Use of Name Mangling
Demonstrate how to access the private attribute `__marks` using name mangling technique from outside the class.

```
s = Student("Chandan", 101, 95)
print(s._Student__marks)  # Accessing private attribute via name mangling
```

5)Reflection
Answer the following short questions:
1. Why can't private members be accessed directly?

To protect internal implementation details and prevent accidental modification from outside the class.

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

Protected members signal that attributes should only be accessed within the class and its subclasses.

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

Name mangling adds the class name prefix to private members (e.g., `__marks` becomes `_ClassName__marks`), which prevents unintentional access and allows controlled access when necessary.