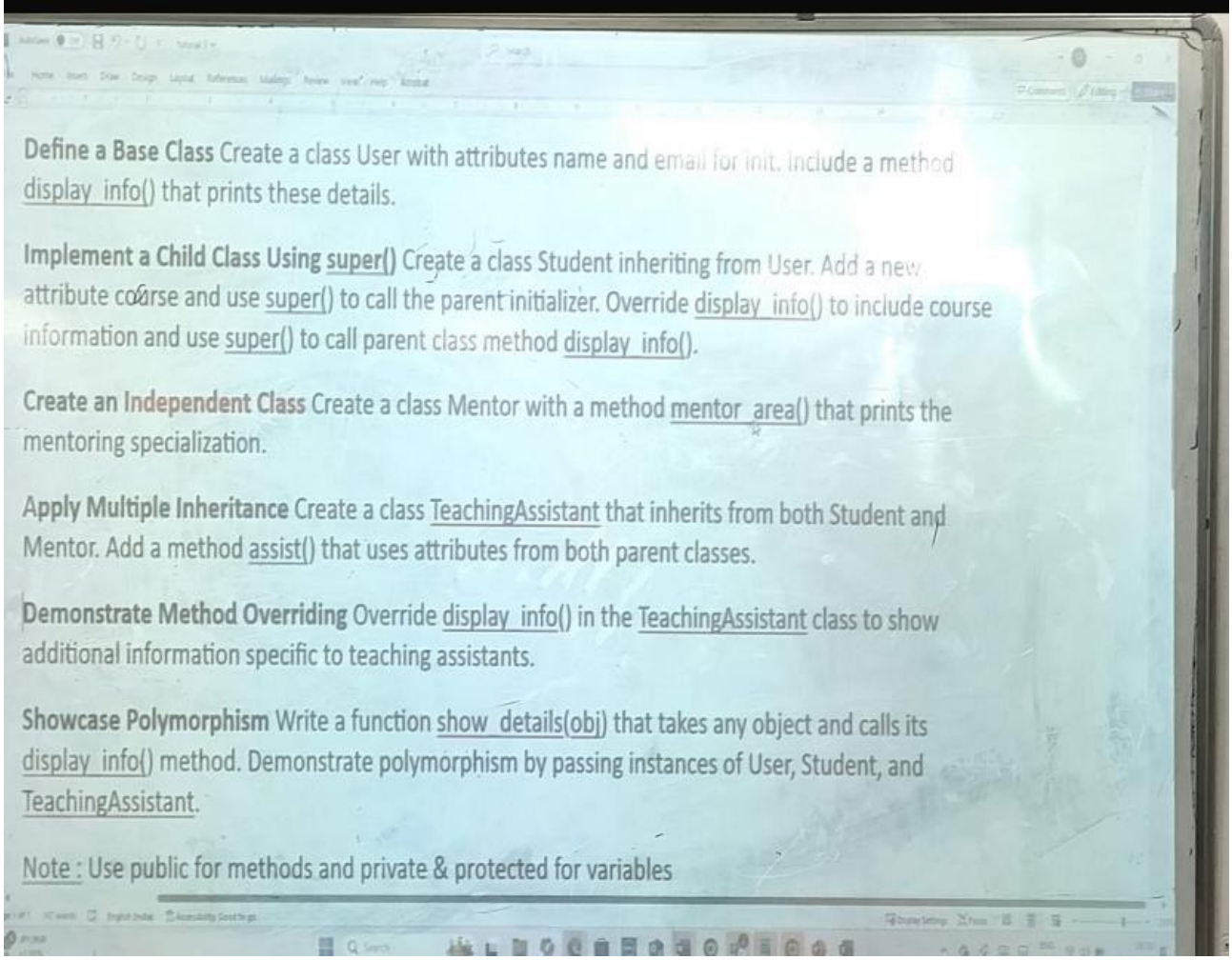

I see the question from the image — it's about implementing **OOP concepts in Python** with:

- **Base class**
- **Child class with `super()`**
- **Independent class**
- **Multiple inheritance**
- **Method overriding**
- **Polymorphism**
- **Variable visibility (public, protected, private)**

Here's the complete **Python code** solution:

```
# Base Class
class User:
    def __init__(self, name, email):
        self._name = name          # Protected variable
        self.__email = email       # Private variable

    def display_info(self):        # Public method
        print(f"Name: {self._name}, Email: {self.__email}")
```

```
# Child Class using super()
class Student(User):
    def __init__(self, name, email, course):
        super().__init__(name, email)
        self._course = course

    def display_info(self):
        super().display_info()
        print(f"Course: {self._course}")


# Independent Class
class Mentor:
    def __init__(self, specialization):
        self._specialization = specialization

    def mentor_area(self):
        print(f"Mentoring Specialization:
{self._specialization}")


# Multiple Inheritance
class TeachingAssistant(Student, Mentor):
    def __init__(self, name, email, course, specialization):
        Student.__init__(self, name, email, course)
        Mentor.__init__(self, specialization)

    def assist(self):
        print(f"{self._name} assists in {self._course}
({self._specialization})")

    # Method Overriding
    def display_info(self):
        super().display_info()
        print(f"Specialization: {self._specialization}")


# Polymorphism demonstration
def show_details(obj):
    obj.display_info()


# Example usage
if __name__ == "__main__":
```

```
    user1 = User("Alice", "alice@example.com")
    student1 = Student("Bob", "bob@example.com", "Python
Programming")
    ta1 = TeachingAssistant("Charlie", "charlie@example.com",
"Data Science", "Machine Learning")

    # Calling methods directly
    print("\n--- Direct Method Calls ---")
    user1.display_info()
    student1.display_info()
    ta1.display_info()
    ta1.assist()

    # Polymorphism in action
    print("\n--- Polymorphism Output ---")
    for person in (user1, student1, ta1):
        show_details(person)
```

Explanation

1. **Public methods** → `display_info`, `assist`, `mentor_area`.
2. **Protected variables** → `_name`, `_course`, `_specialization` (accessible by subclasses).
3. **Private variable** → `__email` (only accessible inside the class).
4. **`super()` usage** → Used in `Student.display_info()` and `TeachingAssistant.display_info()` to call parent implementations.
5. **Multiple inheritance** → `TeachingAssistant(Student, Mentor)` inherits from both.
6. **Method overriding** → `display_info()` in `TeachingAssistant` adds extra details.
7. **Polymorphism** → `show_details()` works with any object having `display_info()`.

If you want, I can also **draw a class diagram** showing relationships between `User`, `Student`, `Mentor`, and `TeachingAssistant` so it's clear for your notes.
Do you want me to include that?