AI ASSISTED CODING

END LAB EXAM

SET-3

QUESTION 1:

Use few-shot prompting to generate Python classes representing a Student and a Course with basic operations.

PROMPT:

Generate a python code with classes named as student and course using few-shot example.

CODE:

```
# ===== Few-Shot Examples =====

# Example 1
# A simple class representing a Book
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def get_description(self):
        return f"{self.title} by {self.author}"

    def rename_title(self, new_title):
        self.title = new_title


# Example 2
# A simple class representing a Library
class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)
```

```
    def count_books(self):
        return len(self.books)

    def list_titles(self):
        return [book.title for book in self.books]


# ===== Target Request (Few-Shot Prompt) =====
# Using the structure from the examples above,
# generate classes for Student and Course.


class Student:
    def __init__(self, name, student_id):
        self.name = name
        self.student_id = student_id
        self.courses = []

    def enroll(self, course):
        # Enroll student in the course
        if course not in self.courses:
            self.courses.append(course)
            course.add_student(self)

    def list_courses(self):
        # Return list of course names
        return [course.name for course in self.courses]


class Course:
    def __init__(self, name, course_code):
        self.name = name
        self.course_code = course_code
        self.students = []

    def add_student(self, student):
        # Add student to course
        if student not in self.students:
            self.students.append(student)

    def count_students(self):
        # Return number of enrolled students
        return len(self.students)
```

```
# ===== Verification Code (Run in Colab) =====

math = Course("Mathematics", "MATH101")
eng = Course("English", "ENG201")

s = Student("Alice", "S001")
s.enroll(math)
s.enroll(eng)

print("Student Name:", s.name)
print("Courses Enrolled:", s.list_courses())
print("Math Student Count:", math.count_students())
print("English Student Count:", eng.count_students())
```

OUTPUT:

```
Student Name: Alice
Courses Enrolled: ['Mathematics', 'English']
Math Student Count: 1
English Student Count: 1
```

EXPLANATION:

- The code defines `Student` and `Course` classes to simulate a university enrollment system.
- The `Student` class manages a student's name, ID, and the courses they are enrolled in.
- The `Course` class tracks its name, course code, and the students registered for it.
- Students can `enroll` in courses, and courses can `add_student`s, establishing a relationship.
- The example demonstrates creating instances, enrolling a student, and printing enrollment details.

QUESTION-2:

Use Copilot to automatically generate docstrings and comments for the classes, and verify clarity.

PROMPT:

To generate a docstrings and inline comments.

CODE:

```
class Student:
    """
    Represents a student in a school or university system.

    Attributes:
        name (str): The full name of the student.
        student_id (str): A unique identifier assigned to the student.
        courses (list[Course]): A list of courses the student is enrolled
in.
    """

    def __init__(self, name, student_id):
        """Initialize a new Student instance with name and ID."""
        self.name = name
        self.student_id = student_id
        self.courses = []  # Stores Course objects

    def enroll(self, course):
        """
        Enroll the student in a course.

        Args:
            course (Course): The course to enroll this student in.
        """
        if course not in self.courses:
            self.courses.append(course)
            course.add_student(self)  # Maintain two-way relationship

    def list_courses(self):
        """
        Get the names of all courses the student is enrolled in.

        Returns:
```

```
            list[str]: List of course names.
        """
        return [course.name for course in self.courses]


class Course:
    """
    Represents an academic course that students can enroll in.

    Attributes:
        name (str): Name of the course.
        course_code (str): Unique course identifier.
        students (list[Student]): List of students enrolled in the course.
    """

    def __init__(self, name, course_code):
        """Initialize a new Course instance."""
        self.name = name
        self.course_code = course_code
        self.students = []  # Stores Student objects

    def add_student(self, student):
        """
        Add a student to the course.

        Args:
            student (Student): The student to add.
        """
        if student not in self.students:
            self.students.append(student)

    def count_students(self):
        """
        Count how many students are enrolled.

        Returns:
            int: Number of enrolled students.
        """
        return len(self.students)


# ===== Verification Code =====

math = Course("Mathematics", "MATH101")
eng = Course("English", "ENG201")
```

```
s = Student("Alice", "S001")
s.enroll(math)
s.enroll(eng)

print("Student Name:", s.name)
print("Courses Enrolled:", s.list_courses())
print("Math Student Count:", math.count_students())
print("English Student Count:", eng.count_students())
```

OUTPUT:

```
Student Name: Alice
Courses Enrolled: ['Mathematics', 'English']
Math Student Count: 1
English Student Count: 1
```