```
import ZODB, ZODB.FileStorage
import transaction
from HW11_66011217_Shisa_z_enrollment import Student, Course, Enrollment

storage = ZODB.FileStorage.FileStorage('mydata.fs')
db = ZODB.DB(storage)
connection = db.open()
root = connection.root()

if 'courses' not in root:
    root.courses = {}

if 'students' not in root:
    root.students = {}

course1 = Course(101, 'Computer Programming', 4)
course2 = Course(201, 'Web Programming', 4)
course3 = Course(202, 'Software Engineering Principle', 5)
course4 = Course(301, 'Artificial Intelligence', 3)

student1 = Student(1103, "Mr. Dvalinn Durinson")
student1.enrollCourse(course1, score=75)
student1.enrollCourse(course2, score=81)
student1.enrollCourse(course3, score=81)
student1.enrollCourse(course4, score=57)

root.students[student1.id] = student1
root.courses[course1.id] = course1
root.courses[course2.id] = course2
root.courses[course3.id] = course3
root.courses[course4.id] = course4

transaction.commit()

for course_id, course in root.courses.items():
    course.printDetail()
    print()

for student_id, student in root.students.items():
    student.printTranscript()
    print()

connection.close()
db.close()
```

HW11_66011217_Shisa_z_enrollment.py :

```
import persistent

def grade_to_points(grade):
    grade_map = {
        "A": 4.0,
        "B": 3.0,
        "C": 2.0,
        "D": 1.0,
        "F": 0.0
    }
    return grade_map.get(grade, 0.0)

class Course(persistent.Persistent):
    def __init__(self, id, name="", credit=0):
        self.id = id
        self.name = name
        self.credit = credit
        self.gradeScheme = [
            {"Grade": "A", "min": 80, "max": 100},
            {"Grade": "B", "min": 70, "max": 79},
            {"Grade": "C", "min": 60, "max": 69},
            {"Grade": "D", "min": 50, "max": 59},
            {"Grade": "F", "min": 0, "max": 49}
        ]

    def printDetail(self):
        print(f"ID: {self.id}   Course: {self.name}, Credit: {self.credit}")

    def scoreGrading(self, score):
        for scheme in self.gradeScheme:
            if scheme["min"] <= score <= scheme["max"]:
                return scheme["Grade"]
        return "Invalid Score"

    def setGradeScheme(self, scheme):
        if not isinstance(scheme, list):
            raise ValueError("Grade scheme must be a list of dictionaries.")
        for item in scheme:
            if not isinstance(item, dict):
                raise ValueError("Each grade scheme entry must be a dictionary.")
            if not all(k in item for k in ("Grade", "min", "max")):
                raise ValueError("Each dictionary must contain 'Grade', 'min', and 'max' keys.")
```

```
            if not (isinstance(item["Grade"], str) and isinstance(item["min"], int) and
isinstance(item["max"], int)):
                raise ValueError("'Grade' must be a string and 'min'/'max' must be integers.")
            if item["min"] > item["max"]:
                raise ValueError(f"For grade {item['Grade']}, 'min' cannot be greater than 'max'.")
        self.gradeScheme = scheme

class Student(persistent.Persistent):
    def __init__(self, id, name=""):
        self.id = id
        self.name = name
        self.enrolls = []

    def enrollCourse(self, course, score=0):
        enrollment = Enrollment(self, course, score)
        self.enrolls.append(enrollment)
        return enrollment

    def getEnrollment(self, course):
        for enroll in self.enrolls:
            if enroll.course.id == course.id:
                return enroll
        return None

    def printTranscript(self):
        print(f"Transcript\nID: {self.id}, Name: {self.name}")
        print("Course list:")
        total_credits = 0
        total_points = 0
        for enrollment in self.enrolls:
            course = enrollment.course
            score = enrollment.getScore()
            grade = course.scoreGrading(score)
            print(f"ID: {course.id:<6} Course: {course.name}, Credit: {course.credit}, Score: {score},
Grade: {grade}")
            total_credits += course.credit
            total_points += grade_to_points(grade) * course.credit

        gpa = total_points / total_credits if total_credits > 0 else 0
        print(f"Total GPA is {gpa:.2f}")


class Enrollment(persistent.Persistent):
    def __init__(self, student, course, score=0):
```

```
        self.student = student
        self.course = course
        self.score = score

    def getCourse(self):
        return self.course

    def getScore(self):
        return self.score

    def setScore(self, score):
        if not isinstance(score, int) or not (0 <= score <= 100):
            raise ValueError("Score must be an integer between 0 and 100.")
        self.score = score

    def getGrade(self):
        return self.course.scoreGrading(self.score)

    def printDetail(self):
        print(f"ID: {self.course.id}  Course: {self.course.name}, Credit: {self.course.credit}, Score:
{self.score}, Grade: {self.getGrade()}")
```