UNIVERSITY MANAGMENT

A Python Project Submitted

In

B.TECH (INFORMATION TECHNOLOGY)

by

Ashish Raj (ROLL NO:- 2401330130074)

Anushka Agrawal (ROLL NO:- 2401330130059)

Aditya prakash singh (ROLL NO:- 2401330130022)

Under the Supervision of

Prof. (Dr.) Sarika Agarwal

Professor, Artificial Intelligence

Department of Information Technology

School of Computer Science & Information Technology

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA

(An Autonomous Institute)

Affiliated to

DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, LUCKNOW

(June 2025)

ABSTRACT

This University Management System is a console-based application that helps manage students, faculty, and courses in a university. It demonstrates how Object-Oriented Programming works by using classes and objects to represent real-world entities and their relationships. This system shows how object-oriented programming can model real-world situations by creating classes that represent actual things (people, courses) and defining how they interact with each other. The University class acts like a manager that coordinates all the different parts of the system to work together smoothly.

SYSTEM STRUCTURE:

The system is built using 5 main classes that work together:

1. **Person Class (Base Class):**

   - Stores basic information: ID and name
   - Acts as a parent class for Student and Faculty
   - Methods: __init__, __str__, to_dict()

2. **Student Class (inherits from Person):**

   - Additional info: major and enrolled courses
   - Methods: enroll_course(), drop_course(), display_details()
   - Keeps track of which courses the student is taking

3. **Faculty Class (inherits from Person):**

   - Additional info: department and assigned courses
   - Methods: assign_course(), unassign_course(), display_details()
   - Keeps track of which courses the faculty member teaches

4. **Course Class:**

   - Stores: course code, title, credits, prerequisites
   - Tracks: enrolled students and assigned faculty
   - Methods: add_student_id(), remove_student_id(), assign_faculty_id()

5. **University Class (Main Controller):**

   - Manages all students, faculty, and courses
   - Handles all the main operations like adding, removing, enrolling
   - Saves and loads data from JSON files

HOW THE SYSTEM WORKS:

Data Storage:

- All information is stored in three JSON files: students.json, faculty.json, courses.json

Main Operations:

- Add/Remove Students: Create new student records or delete existing ones
- Add/Remove Faculty: Manage faculty member information
- Add/Remove Courses: Create courses with credits and prerequisites
- Enroll Students: Put students in courses (checks prerequisites first)
- Assign Faculty: Give courses to faculty members to teach
- View Information: See lists of students, faculty, courses, and course rosters

Key Features:

- Prerequisite Checking: Students can't enroll in advanced courses without completing prerequisites
- Data Consistency: When you enroll a student, both the student's record and the course record are updated
- Error Handling: The system checks for mistakes and gives helpful error messages
- Menu Interface: Simple numbered menu system for easy navigation

PROGRAM FLOW:

1. Program starts and loads existing data from JSON files
2. Main menu appears with 16 options (0-15)
3. User selects an option
4. Program calls the appropriate function in the University class
5. Function validates input and performs the operation
6. Data is automatically saved to files
7. User gets feedback about success or errors
8. Program returns to main menu

RELATIONSHIPS BETWEEN CLASSES:

- Student and Faculty both inherit from Person (inheritance)
- University contains collections of Student, Faculty, and Course objects (composition)
- When a student enrolls in a course, both objects are updated to maintain consistency
- Course objects store student IDs and faculty IDs to link to actual Student and Faculty objects

FILE STRUCTURE:

The system creates a 'data' folder with three JSON files:

- students.json: Contains all student information and their enrolled courses
- faculty.json: Contains all faculty information and their assigned courses
- courses.json: Contains all course information, prerequisites, and enrollment lists

MAIN FUNCTIONS IN UNIVERSITY CLASS:

Management Functions:

- add_student(), remove_student()
- add_faculty(), remove_faculty()
- add_course(), remove_course()

Enrollment Functions:

- enroll_student_in_course()
- drop_student_from_course()
- assign_faculty_to_course()
- unassign_faculty_from_course()

Display Functions:

- display_all_students()
- display_all_faculty()
- display_all_courses()
- view_course_roster()

Utility Functions:

- _load_data(): Reads information from JSON files
- _save_data(): Writes information to JSON files
- _verify_data_consistency(): Makes sure all relationships are correct

PROGRAM

```
import json
import os
from abc import ABC
from typing import List, Dict, Optional

class Person(ABC):
    def __init__(self, id: str, name: str):
        self._id = id
        self._name = name

    @property
    def id(self) -> str:
        return self._id

    @property
    def name(self) -> str:
        return self._name

    def __str__(self) -> str:
        return f"ID: {self._id}, Name: {self._name}"

    def to_dict(self) -> dict:
        return {'id': self._id , 'name': self._name}

class Student(Person):
    def __init__(self, id: str, name: str, major: str):
        super().__init__(id, name)
        self._major = major
        self._enrolled_course_codes = []

    @property
    def major(self) -> str:
        return self._major

    @major.setter
    def major(self, value: str):
        self._major = value

    @property
    def enrolled_course_codes(self) -> List[str]:
        return self._enrolled_course_codes.copy()

    def enroll_course(self, course_code: str) -> None:
        if course_code not in self._enrolled_course_codes:
            self._enrolled_course_codes.append(course_code)

    def drop_course(self, course_code: str) -> None:
        if course_code in self._enrolled_course_codes:
            self._enrolled_course_codes.remove(course_code)

    def display_details(self) -> str:
        base_details = super().__str__()
        return (f"{base_details}, Major: {self._major}, "
                f"Enrolled Courses: {len(self._enrolled_course_codes)}")

    def to_dict(self) -> dict:
        base_dict = super().to_dict()
        base_dict.update({'type': 'student','major': self._major,
        'enrolled_course_codes': self._enrolled_course_codes})
        return base_dict

class Faculty(Person):
    def __init__(self, id: str, name: str, department: str):
        super().__init__(id, name)
        self._department = department
        self._assigned_course_codes = []

    @property
    def department(self) -> str:
        return self._department

    @department.setter
    def department(self, value: str):
        self._department = value

    @property
    def assigned_course_codes(self) -> List[str]:
        return self._assigned_course_codes.copy()

    def assign_course(self, course_code: str) -> None:
        if course_code not in self._assigned_course_codes:
            self._assigned_course_codes.append(course_code)

    def unassign_course(self, course_code: str) -> None:
        if course_code in self._assigned_course_codes:
            self._assigned_course_codes.remove(course_code)

    def display_details(self) -> str:
        base_details = super().__str__()
        return (f"{base_details}, Department: {self._department}, "
                f"Assigned Courses: {len(self._assigned_course_codes)}")

    def to_dict(self) -> dict:
        base_dict = super().to_dict()
        base_dict.update({'type': 'faculty','department': self._department,
        'assigned_course_codes': self._assigned_course_codes})
        return base_dict

class Course:
    def __init__(self, course_code: str, title: str, credits: int, prerequisites):
        self._course_code = course_code
        self._title = title
        self._credits = credits
        self._prerequisite_codes = prerequisites or []
        self._enrolled_student_ids = []
        self._assigned_faculty_id = None

    @property
    def course_code(self) -> str:
        return self._course_code

    @property
    def title(self) -> str:
        return self._title

    @property
    def credits(self) -> int:
        return self._credits

    @property
    def prerequisite_codes(self) -> List[str]:
        return self._prerequisite_codes.copy()

    @property
    def enrolled_student_ids(self) -> List[str]:
        return self._enrolled_student_ids.copy()

    @property
    def assigned_faculty_id(self) -> Optional[str]:
        return self._assigned_faculty_id

    @assigned_faculty_id.setter
    def assigned_faculty_id(self, value):
        self._assigned_faculty_id = value

    def add_prerequisite(self, prerequisite_code: str) -> None:
        if prerequisite_code not in self._prerequisite_codes:
            self._prerequisite_codes.append(prerequisite_code)

    def add_student_id(self, student_id: str) -> None:
        if student_id not in self._enrolled_student_ids:
            self._enrolled_student_ids.append(student_id)

    def remove_student_id(self, student_id: str) -> None:
        if student_id in self._enrolled_student_ids:
            self._enrolled_student_ids.remove(student_id)

    def assign_faculty_id(self, faculty_id: str) -> None:
        self._assigned_faculty_id = faculty_id

    def unassign_faculty_id(self) -> None:
        self._assigned_faculty_id = None

    def display_details(self) -> str:
        faculty_info = f"Faculty: {self._assigned_faculty_id}" if self._assigned_faculty_id else "Faculty: 
Unassigned"
```

```
        prerequisites_info = f"Prerequisites: {', '.join(self._prerequisite_codes)}" if self._prerequisite_codes else
"Prerequisites: None"
        return (f"Course: {self._course_code} - {self._title}, "
                f"Credits: {self._credits}, "
                f"{prerequisites_info}, "
                f"Enrolled Students: {len(self._enrolled_student_ids)}, "
                f"{faculty_info}")

    def __str__(self) -> str:
        return f"{self._course_code}: {self._title} ({self._credits} credits)"

    def __repr__(self) -> str:
        return f"Course(code='{self._course_code}', title='{self._title}', credits={self._credits})"

    def to_dict(self) -> dict:
        return {'course_code': self._course_code,'title': self._title,'credits': self._credits,
            'prerequisite_codes': self._prerequisite_codes,'enrolled_student_ids': self._enrolled_student_ids,
            'assigned_faculty_id': self._assigned_faculty_id
        }

class University:
    def __init__(self, student_file='data/students.json',
                 faculty_file='data/faculty.json', course_file='data/courses.json'):
        self._students: Dict[str, Student] = {}
        self._faculty: Dict[str, Faculty] = {}
        self._courses: Dict[str, Course] = {}
        os.makedirs('data', exist_ok=True)
        self._student_file = student_file
        self._faculty_file = faculty_file
        self._course_file = course_file
        self._load_data()

    def _load_data(self) -> None:
        try:
            with open(self._student_file, 'r') as f:
                student_data = json.load(f)
                for student_dict in student_data:
                    student = Student(
                        student_dict['id'],
                        student_dict['name'],
                        student_dict['major']
                    )
                    student._enrolled_course_codes = student_dict.get('enrolled_course_codes', [])
                    self._students[student.id] = student
        except FileNotFoundError:
            print(f"No existing {self._student_file} found. Starting with empty student registry.")
        except json.JSONDecodeError:
            print(f"Error reading {self._student_file}. Starting with empty student registry.")
        try:
            with open(self._faculty_file, 'r') as f:
                faculty_data = json.load(f)
                for faculty_dict in faculty_data:
                    faculty = Faculty(faculty_dict['id'],faculty_dict['name'],faculty_dict['department'])
                    faculty._assigned_course_codes = faculty_dict.get('assigned_course_codes', [])
                    self._faculty[faculty.id] = faculty
        except FileNotFoundError:
            print(f"No existing {self._faculty_file} found. Starting with empty faculty registry.")
        except json.JSONDecodeError:
            print(f"Error reading {self._faculty_file}. Starting with empty faculty registry.")
        try:
            with open(self._course_file, 'r') as f:
                course_data = json.load(f)
                for course_dict in course_data:
                    course = Course(course_dict['course_code'],course_dict['title'],course_dict['credits'],
                        course_dict.get('prerequisite_codes', []))
                    course._enrolled_student_ids = course_dict.get('enrolled_student_ids', [])
                    course._assigned_faculty_id = course_dict.get('assigned_faculty_id')
                    self._courses[course.course_code] = course
        except FileNotFoundError:
            print(f"No existing {self._course_file} found. Starting with empty course registry.")
        except json.JSONDecodeError:
            print(f"Error reading {self._course_file}. Starting with empty course registry.")
        self._verify_data_consistency()

    def _verify_data_consistency(self) -> None:
        for student in self._students.values():
            valid_courses = []
            for course_code in student.enrolled_course_codes:
                if course_code in self._courses:
                    valid_courses.append(course_code)
                if student.id not in self._courses[course_code].enrolled_student_ids:
                    self._courses[course_code].add_student_id(student.id)
            student._enrolled_course_codes = valid_courses

        for faculty in self._faculty.values():
            valid_courses = []
            for course_code in faculty.assigned_course_codes:
                if course_code in self._courses:
                    valid_courses.append(course_code)
                    if self._courses[course_code].assigned_faculty_id != faculty.id:
                        self._courses[course_code].assigned_faculty_id = faculty.id
            faculty._assigned_course_codes = valid_courses

        for course in self._courses.values():
            valid_students = []
            for student_id in course.enrolled_student_ids:
                if student_id in self._students:
                    valid_students.append(student_id)
                    if course.course_code not in self._students[student_id].enrolled_course_codes:
                        self._students[student_id].enroll_course(course.course_code)
            course._enrolled_student_ids = valid_students

            if course.assigned_faculty_id and course.assigned_faculty_id not in self._faculty:
                course._assigned_faculty_id = None
            elif course.assigned_faculty_id:
                if course.course_code not in self._faculty[course.assigned_faculty_id].assigned_course_codes:
                    self._faculty[course.assigned_faculty_id].assign_course(course.course_code)
    def _save_data(self) -> None:
        try:
            student_data = [student.to_dict() for student in self._students.values()]
            with open(self._student_file, 'w') as f:
                json.dump(student_data, f, indent=2)

            faculty_data = [faculty.to_dict() for faculty in self._faculty.values()]
            with open(self._faculty_file, 'w') as f:
                json.dump(faculty_data, f, indent=2)

            course_data = [course.to_dict() for course in self._courses.values()]
            with open(self._course_file, 'w') as f:
                json.dump(course_data, f, indent=2)
        except Exception as e:
            print(f"Error saving data: {e}")

    def add_student(self) -> bool:
        print("\n--- Add New Student ---")
        try:
            student_id = input("Enter Student ID: ").strip()
            name = input("Enter Student Name: ").strip()
            if not name:
                print("Student name cannot be empty.")
                return False

            major = input("Enter Student Major: ").strip()
            if not major:
                print("Student major cannot be empty.")
                return False
            student = Student(student_id, name, major)
            self._students[student.id] = student
            self._save_data()
            print(f"Student {name} (ID: {student_id}) added successfully!")
            return True

        except Exception as e:
            print(f"Error adding student: {e}")
            return False

    def remove_student(self) -> bool:
        print("\n--- Remove Student ---")
        try:
            student_id = input("Enter Student ID to remove: ").strip()
            student = self._students[student_id]
            if student.enrolled_course_codes:
                print(f"Cannot remove student {student.name}. Student is enrolled in courses: {',
'.join(student.enrolled_course_codes)}")
                print("Please drop the student from all courses first.")
                return False

            del self._students[student_id]
            self._save_data()
            print(f"Student {student.name} (ID: {student_id}) removed successfully!")
            return True

        except Exception as e:
            print(f"Error removing student: {e}")
            return False

    def add_faculty(self) -> bool:
        print("\n--- Add New Faculty ---")
        try:
            faculty_id = input("Enter Faculty ID: ").strip()
            name = input("Enter Faculty Name: ").strip()
            if not name:
                print("Faculty name cannot be empty.")
                return False

            department = input("Enter Department: ").strip()
            if not department:
                print("Department cannot be empty.")
                return False

            faculty = Faculty(faculty_id, name, department)
            self._faculty[faculty.id] = faculty
            self._save_data()
            print(f"Faculty {name} (ID: {faculty_id}) added successfully!")
            return True

        except Exception as e:
            print(f"Error adding faculty: {e}")
            return False

    def remove_faculty(self) -> bool:
        print("\n--- Remove Faculty ---")
        try:
            faculty_id = input("Enter Faculty ID to remove: ").strip()
            faculty = self._faculty[faculty_id]
            if faculty.assigned_course_codes:
                print(f"Cannot remove faculty {faculty.name}. Faculty is assigned to courses: {',
'.join(faculty.assigned_course_codes)}")
                print("Please unassign the faculty from all courses first.")
                return False

            del self._faculty[faculty_id]
            self._save_data()
            print(f"Faculty {faculty.name} (ID: {faculty_id}) removed successfully!")
            return True

        except Exception as e:
            print(f"Error removing faculty: {e}")
            return False

    def add_course(self) -> bool:
        print("\n--- Add New Course ---")
        try:
            course_code = input("Enter Course Code: ").strip().upper()
            title = input("Enter Course Title: ").strip()
            if not title:
                print("Course title cannot be empty.")
                return False

            credits_str = input("Enter Credits: ").strip()
            try:
                credits = int(credits_str)
                if credits <= 0:
                    print("Credits must be a positive number.")
                    return False
            except ValueError:
                print("Credits must be a valid number.")
                return False
            prereq_input = input("Enter Prerequisites (comma-separated, or press Enter for none): ").strip()
            prerequisites = []
            if prereq_input:
                prerequisites = [code.strip().upper() for code in prereq_input.split(',')]
                for prereq in prerequisites:
                    if prereq not in self._courses:
                        print(f"Warning: Prerequisite course {prereq} does not exist.")

            course = Course(course_code, title, credits, prerequisites)
            self._courses[course.course_code] = course
            self._save_data()
            print(f"Course {course_code} - {title} added successfully!")
            return True

        except Exception as e:
            print(f"Error adding course: {e}")
            return False

    def remove_course(self) -> bool:
        print("\n--- Remove Course ---")
        try:
            course_code = input("Enter Course Code to remove: ").strip().upper()
            course = self._courses[course_code]
            if course.enrolled_student_ids:
                print(f"Cannot remove course {course.title}. Students are enrolled in this course.")
                print("Please drop all students from the course first.")
                return False

            if course.assigned_faculty_id and course.assigned_faculty_id in self._faculty:
                self._faculty[course.assigned_faculty_id].unassign_course(course_code)

            del self._courses[course_code]
            self._save_data()
            print(f"Course {course.title} (Code: {course_code}) removed successfully!")
            return True

        except Exception as e:
            print(f"Error removing course: {e}")
            return False

    def enroll_student_in_course(self) -> bool:
        print("\n--- Enroll Student in Course ---")
        try:
            student_id = input("Enter Student ID: ").strip()
            course_code = input("Enter Course Code: ").strip().upper()
            if not course_code:
                print("Course code cannot be empty.")
                return False

            if course_code not in self._courses:
                print(f"Course with code {course_code} not found.")
                return False

            student = self._students[student_id]
            course = self._courses[course_code]
            if course_code in student.enrolled_course_codes:
                print(f"Student {student.name} is already enrolled in {course.title}.")
                return False
            missing_prereqs = []
            for prereq in course.prerequisite_codes:
                if prereq not in student.enrolled_course_codes:
                    missing_prereqs.append(prereq)
            if missing_prereqs:
                print(f"Cannot enroll student. Missing prerequisites: {', '.join(missing_prereqs)}")
                return False

            # Enroll the student
            student.enroll_course(course_code)
            course.add_student_id(student_id)
            self._save_data()
            print(f"Student {student.name} enrolled in {course.title} successfully!")
            return True

        except Exception as e:
            print(f"Error enrolling student: {e}")
            return False

    def drop_student_from_course(self) -> bool:
        print("\n--- Drop Student from Course ---")
        try:
            student_id = input("Enter Student ID: ").strip()
            course_code = input("Enter Course Code: ").strip().upper()
            if not course_code:
                print("Course code cannot be empty.")
                return False

            if course_code not in self._courses:
                print(f"Course with code {course_code} not found.")
                return False

            student = self._students[student_id]
```

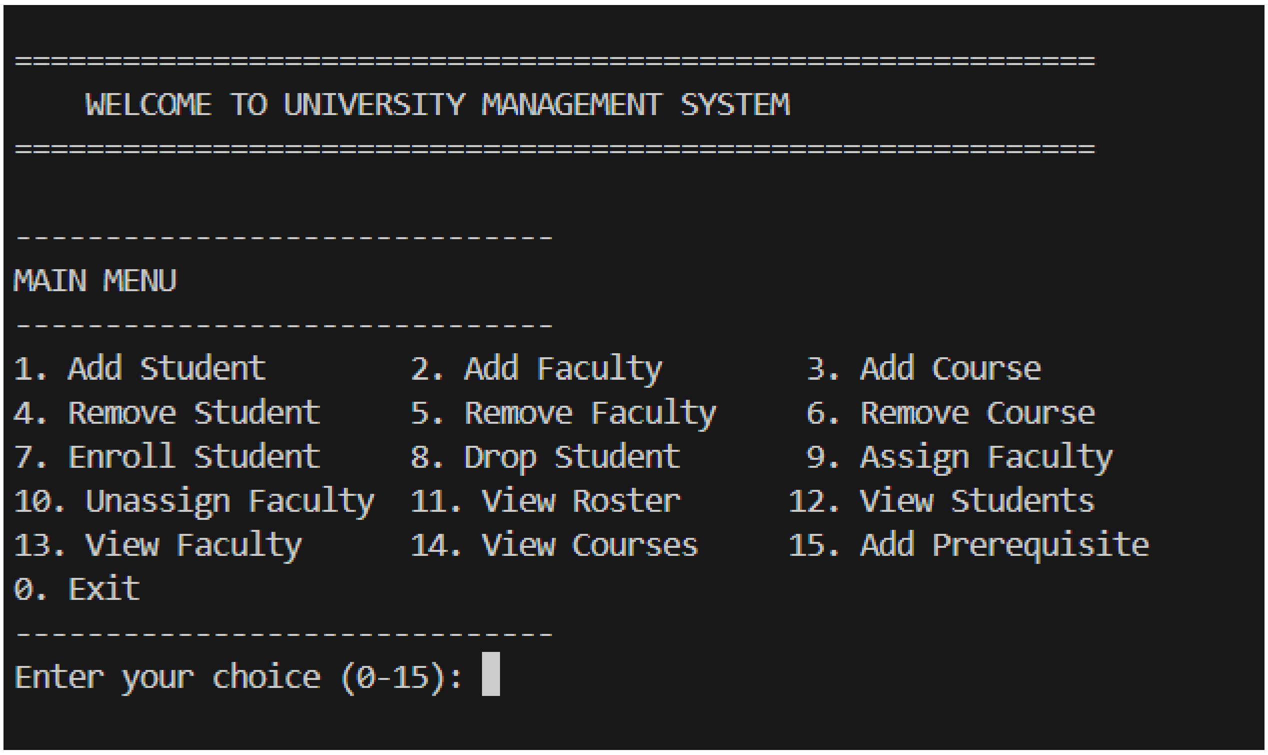
```
            course = self._courses[course_code]

            if course_code not in student.enrolled_course_codes:
                print(f"Student {student.name} is not enrolled in {course.title}.")
                return False

            student.drop_course(course_code)
            course.remove_student_id(student_id)
            self._save_data()
            print(f"Student {student.name} dropped from {course.title} successfully!")
            return True

        except Exception as e:
            print(f"Error dropping student: {e}")
            return False

    def assign_faculty_to_course(self) -> bool:
        print("\n--- Assign Faculty to Course ---")
        try:
            faculty_id = input("Enter Faculty ID: ").strip()
            course_code = input("Enter Course Code: ").strip().upper()
            if not course_code:
                print("Course code cannot be empty.")
                return False

            if course_code not in self._courses:
                print(f"Course with code {course_code} not found.")
                return False

            faculty = self._faculty[faculty_id]
            course = self._courses[course_code]

            # Unassign previous faculty if any
            if course.assigned_faculty_id and course.assigned_faculty_id in self._faculty:
                prev_faculty = self._faculty[course.assigned_faculty_id]
                prev_faculty.unassign_course(course_code)

            # Assign new faculty
            faculty.assign_course(course_code)
            course.assign_faculty_id(faculty_id)
            self._save_data()
            print(f"Faculty {faculty.name} assigned to {course.title} successfully!")
            return True

        except Exception as e:
            print(f"Error assigning faculty: {e}")
            return False

    def unassign_faculty_from_course(self) -> bool:
        print("\n--- Unassign Faculty from Course ---")
        try:
            faculty_id = input("Enter Faculty ID: ").strip()
            course_code = input("Enter Course Code: ").strip().upper()
            if not course_code:
                print("Course code cannot be empty.")
                return False

            if course_code not in self._courses:
                print(f"Course with code {course_code} not found.")
                return False

            faculty = self._faculty[faculty_id]
            course = self._courses[course_code]

            if course.assigned_faculty_id != faculty_id:
                print(f"Faculty {faculty.name} is not assigned to {course.title}.")
                return False

            # Unassign faculty
            faculty.unassign_course(course_code)
            course.unassign_faculty_id()
            self._save_data()
            print(f"Faculty {faculty.name} unassigned from {course.title} successfully!")
            return True

        except Exception as e:
            print(f"Error unassigning faculty: {e}")
            return False

    def view_course_roster(self) -> bool:
        print("\n--- View Course Roster ---")
        try:
            course_code = input("Enter Course Code: ").strip().upper()
            course = self._courses[course_code]
            roster = self.get_course_roster(course_code)

            print(f"\n=== ROSTER FOR {course.title} ({course_code}) ===")

            if course.assigned_faculty_id and course.assigned_faculty_id in self._faculty:
                faculty = self._faculty[course.assigned_faculty_id]
                print(f"Instructor: {faculty.name} ({faculty.department})")
            else:
                print("Instructor: Not assigned")

            print(f"Credits: {course.credits}")

            if course.prerequisite_codes:
                print(f"Prerequisites: {', '.join(course.prerequisite_codes)}")
            else:
                print("Prerequisites: None")

            print(f"\nEnrolled Students ({len(roster)}):")
            if roster:
                for i, student in enumerate(roster, 1):
                    print(f"{i:2d}. {student.name} (ID: {student.id}) - Major: {student.major}")
            else:
                print("No students enrolled.")

            return True

        except Exception as e:
            print(f"Error viewing course roster: {e}")
            return False

    def add_prerequisite(self) -> bool:
        print("\n--- Add Course Prerequisite ---")
        try:
            course_code = input("Enter Course Code: ").strip().upper()
            prerequisite_code = input("Enter Prerequisite Course Code: ").strip().upper()
            if not prerequisite_code:
                print("Prerequisite course code cannot be empty.")
                return False

            if prerequisite_code not in self._courses:
                print(f"Prerequisite course {prerequisite_code} not found.")
                return False

            if prerequisite_code == course_code:
                print("A course cannot be a prerequisite for itself.")
                return False

            course = self._courses[course_code]
            prerequisite_course = self._courses[prerequisite_code]

            if prerequisite_code in course.prerequisite_codes:
                print(f"{prerequisite_course.title} is already a prerequisite for {course.title}.")
                return False

            course.add_prerequisite(prerequisite_code)
            self._save_data()
            print(f"Prerequisite {prerequisite_course.title} added to {course.title} successfully!")
            return True

        except Exception as e:
            print(f"Error adding prerequisite: {e}")
            return False

    def get_course_roster(self, course_code: str) -> List[Student]:
        if course_code not in self._courses:
            return []

        course = self._courses[course_code]
        roster = []

        for student_id in course.enrolled_student_ids:
            if student_id in self._students:
                roster.append(self._students[student_id])

        return roster
```

```
def display_all_students(self) -> None:
    if not self._students:
        print("No students registered.")
        return

    print("\n=== ALL STUDENTS ===")
    for student in self._students.values():
        print(student.display_details())
        if student.enrolled_course_codes:
            print(f"    Enrolled in: {', '.join(student.enrolled_course_codes)}")
        print()

def display_all_faculty(self) -> None:
    if not self._faculty:
        print("No faculty registered.")
        return

    print("\n=== ALL FACULTY ===")
    for faculty in self._faculty.values():
        print(faculty.display_details())
        if faculty.assigned_course_codes:
            print(f"    Teaching: {', '.join(faculty.assigned_course_codes)}")
        print()

def display_all_courses(self) -> None:
    if not self._courses:
        print("No courses registered.")
        return

    print("\n=== ALL COURSES ===")
    for course in self._courses.values():
        print(course.display_details())
        print()

def run(self):
    print("\n" + "="*60)
    print("    WELCOME TO UNIVERSITY MANAGEMENT SYSTEM")
    print("="*60)

    while True:
```

```
print("\n" + "-"*30)
print("MAIN MENU")
print("-"*30)
print("1. Add Student        2. Add Faculty        3. Add Course")
print("4. Remove Student     5. Remove Faculty     6. Remove Course")
print("7. Enroll Student     8. Drop Student       9. Assign Faculty")
print("10. Unassign Faculty  11. View Roster       12. View Students")
print("13. View Faculty      14. View Courses      15. Add Prerequisite")
print("0. Exit")
print("-"*30)

try:
    choice = input("Enter your choice (0-15): ").strip()

    if choice == '0':
        print("\nThank you for using University Management System!")
        break
    elif choice == '1':
        self.add_student()
    elif choice == '2':
        self.add_faculty()
    elif choice == '3':
        self.add_course()
    elif choice == '4':
        self.remove_student()
    elif choice == '5':
        self.remove_faculty()
    elif choice == '6':
        self.remove_course()
    elif choice == '7':
        self.enroll_student_in_course()
    elif choice == '8':
        self.drop_student_from_course()
    elif choice == '9':
        self.assign_faculty_to_course()
    elif choice == '10':
        self.unassign_faculty_from_course()
    elif choice == '11':
        self.view_course_roster()
    elif choice == '12':
```

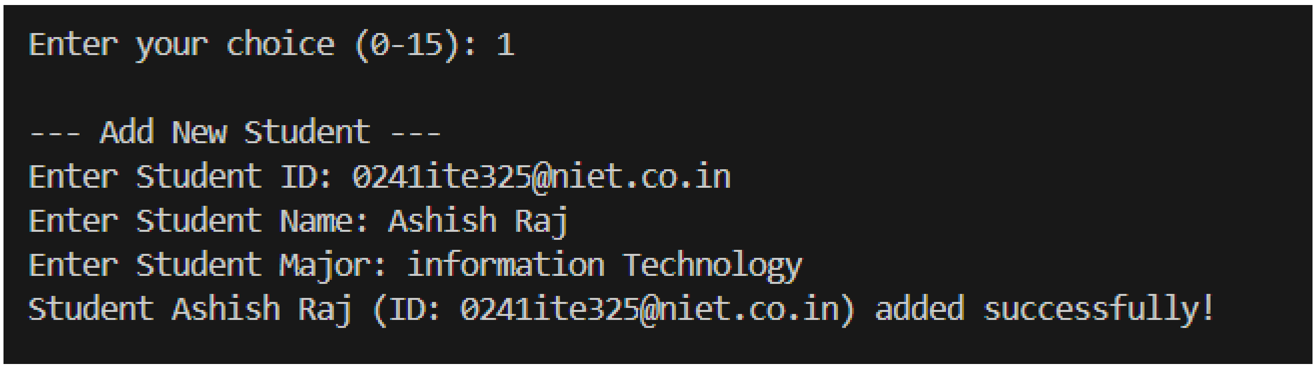
```
                    self.display_all_students()
                elif choice == '13':
                    self.display_all_faculty()
                elif choice == '14':
                    self.display_all_courses()
                elif choice == '15':
                    self.add_prerequisite()
                else:
                    print("Invalid choice. Please enter a number between 0 and 15.")

            except KeyboardInterrupt:
                print("\n\nExiting University Management System...")
                break
            except Exception as e:
                print(f"An error occurred: {e}")

def main():
    university = University()
    university.run()

if __name__ == "__main__":
    main()
```

SNAPSHOT

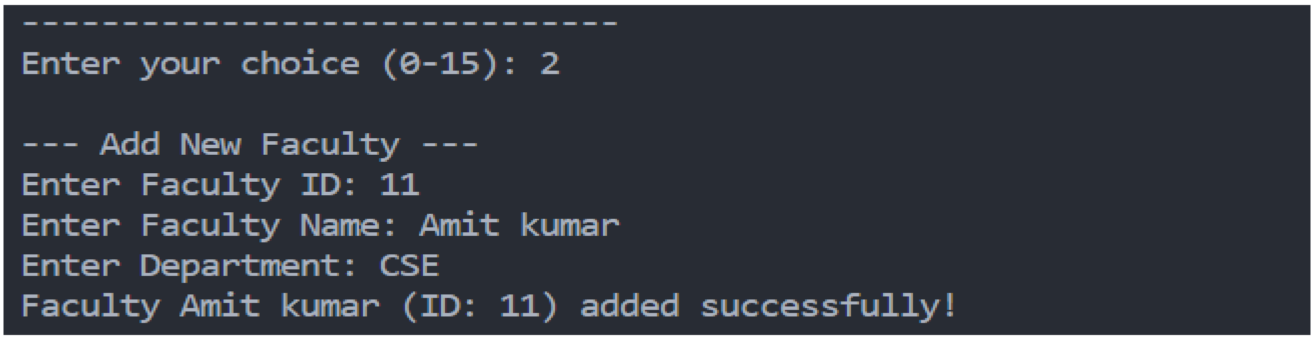


```
============================================================
    WELCOME TO UNIVERSITY MANAGEMENT SYSTEM
============================================================


------------------------------
MAIN MENU
------------------------------
1. Add Student        2. Add Faculty        3. Add Course
4. Remove Student     5. Remove Faculty     6. Remove Course
7. Enroll Student     8. Drop Student       9. Assign Faculty
10. Unassign Faculty  11. View Roster      12. View Students
13. View Faculty      14. View Courses     15. Add Prerequisite
0. Exit
------------------------------
Enter your choice (0-15):
```

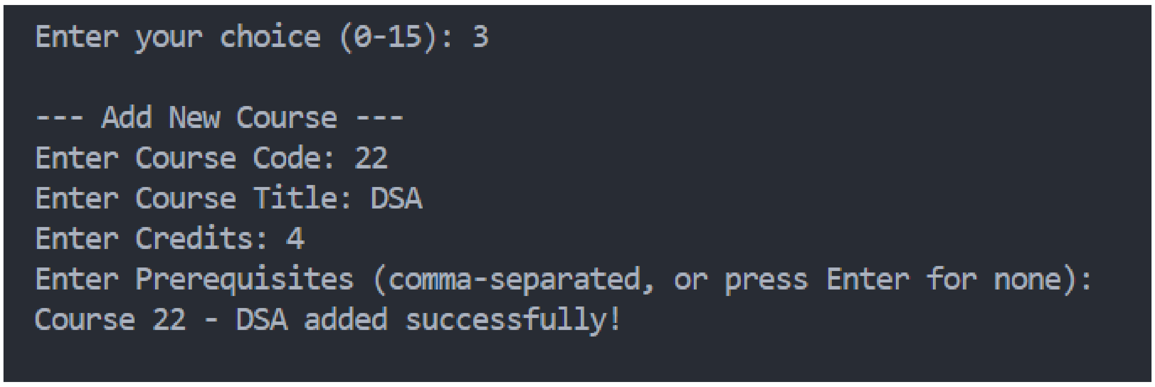
- **Addition of Data:**



```
Enter your choice (0-15): 1

--- Add New Student ---
Enter Student ID: 0241ite325@niet.co.in
Enter Student Name: Ashish Raj
Enter Student Major: information Technology
Student Ashish Raj (ID: 0241ite325@niet.co.in) added successfully!
```

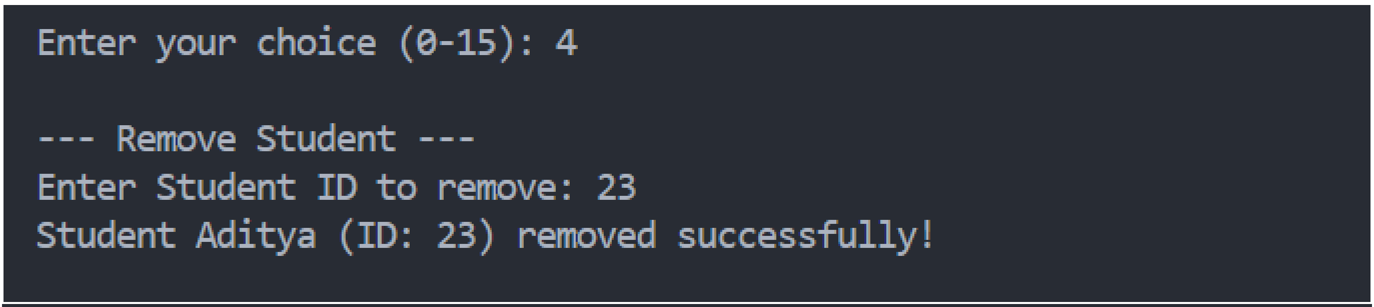


```
------------------------------
Enter your choice (0-15): 2

--- Add New Faculty ---
Enter Faculty ID: 11
Enter Faculty Name: Amit kumar
Enter Department: CSE
Faculty Amit kumar (ID: 11) added successfully!
```

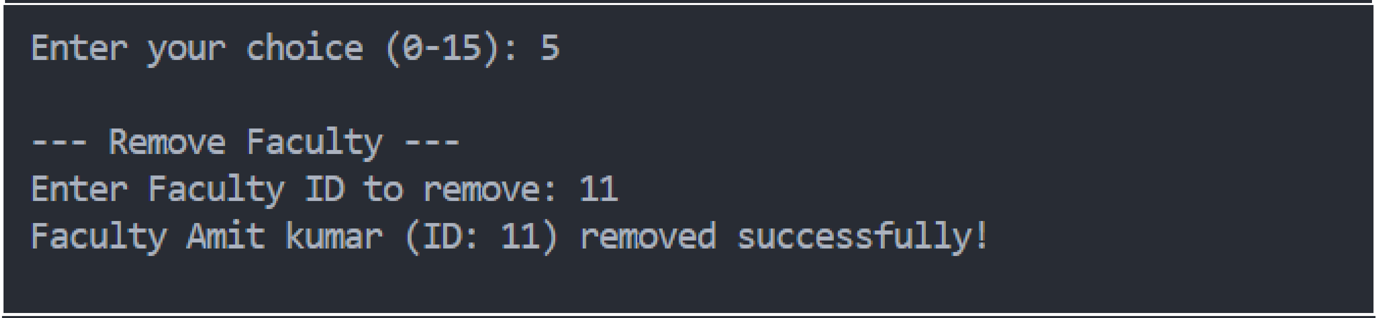


```
Enter your choice (0-15): 3

--- Add New Course ---
Enter Course Code: 22
Enter Course Title: DSA
Enter Credits: 4
Enter Prerequisites (comma-separated, or press Enter for none):
Course 22 - DSA added successfully!
```

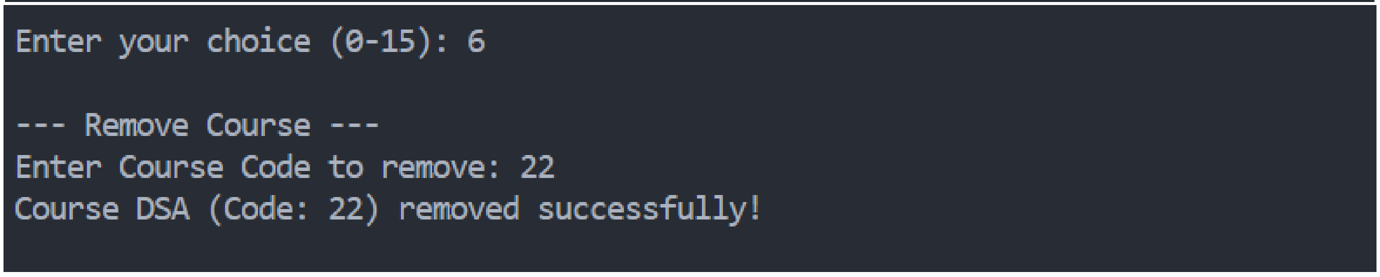
- **Removal of Data:**



```
Enter your choice (0-15): 4

--- Remove Student ---
Enter Student ID to remove: 23
Student Aditya (ID: 23) removed successfully!
```

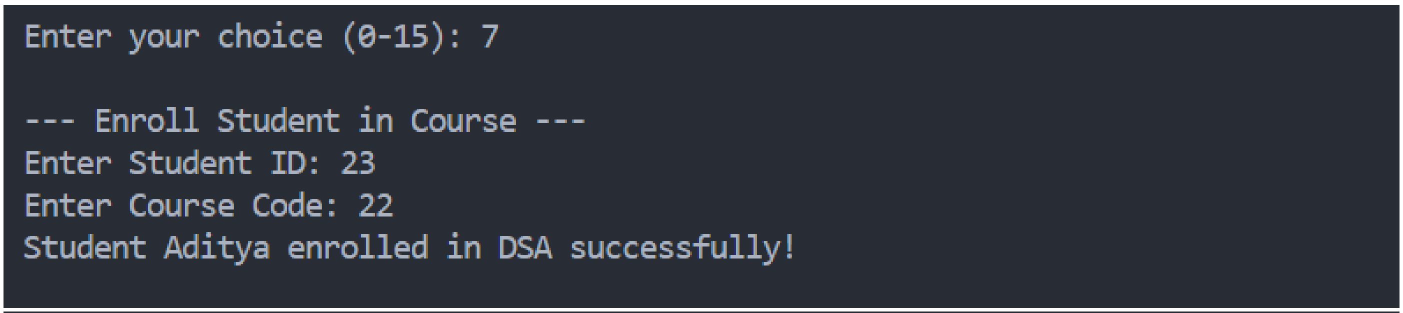


```
Enter your choice (0-15): 5

--- Remove Faculty ---
Enter Faculty ID to remove: 11
Faculty Amit kumar (ID: 11) removed successfully!
```

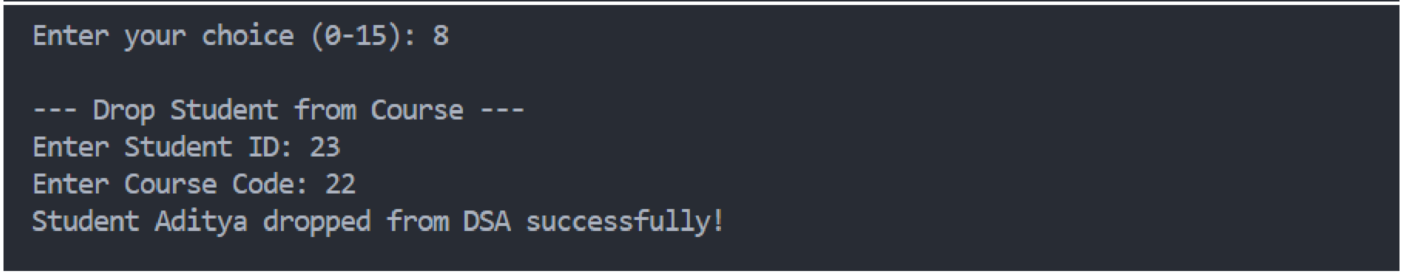


```
Enter your choice (0-15): 6

--- Remove Course ---
Enter Course Code to remove: 22
Course DSA (Code: 22) removed successfully!
```

- **Enroll /Drop Student:**

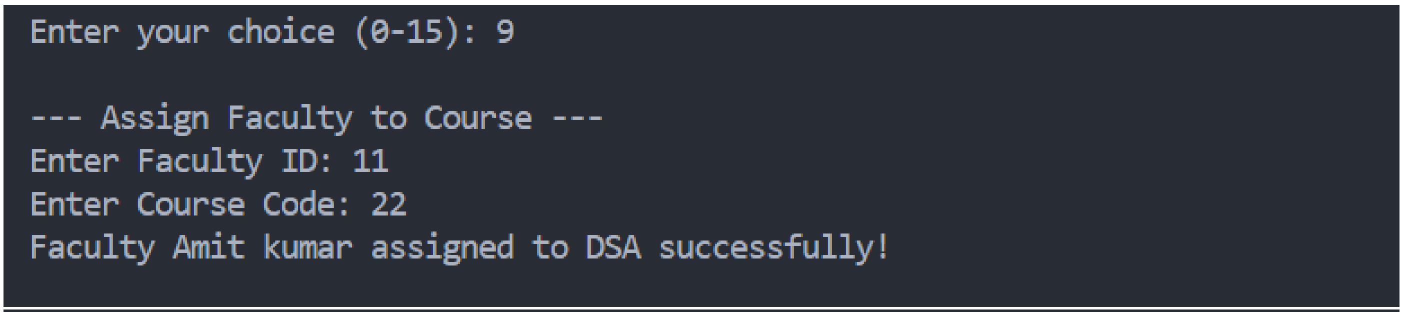


```
Enter your choice (0-15): 7

--- Enroll Student in Course ---
Enter Student ID: 23
Enter Course Code: 22
Student Aditya enrolled in DSA successfully!
```

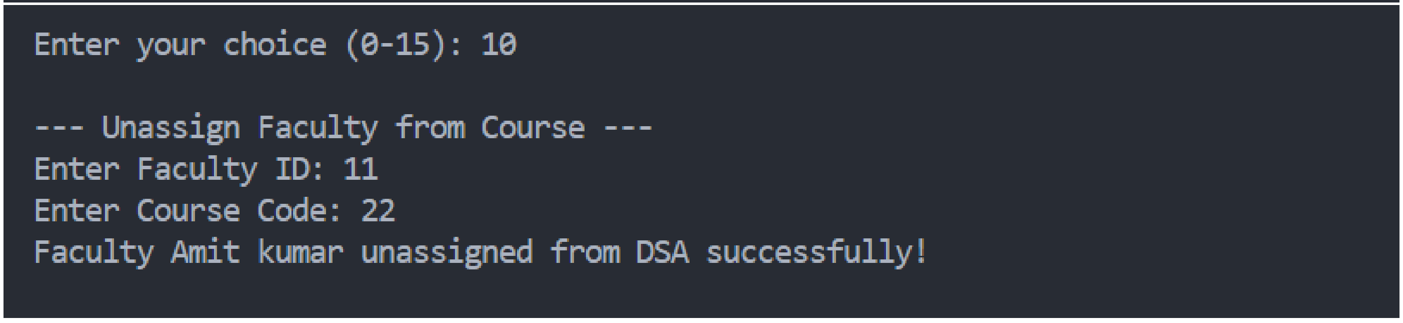


```
Enter your choice (0-15): 8

--- Drop Student from Course ---
Enter Student ID: 23
Enter Course Code: 22
Student Aditya dropped from DSA successfully!
```

- **Assign/Unassign Faculty:**

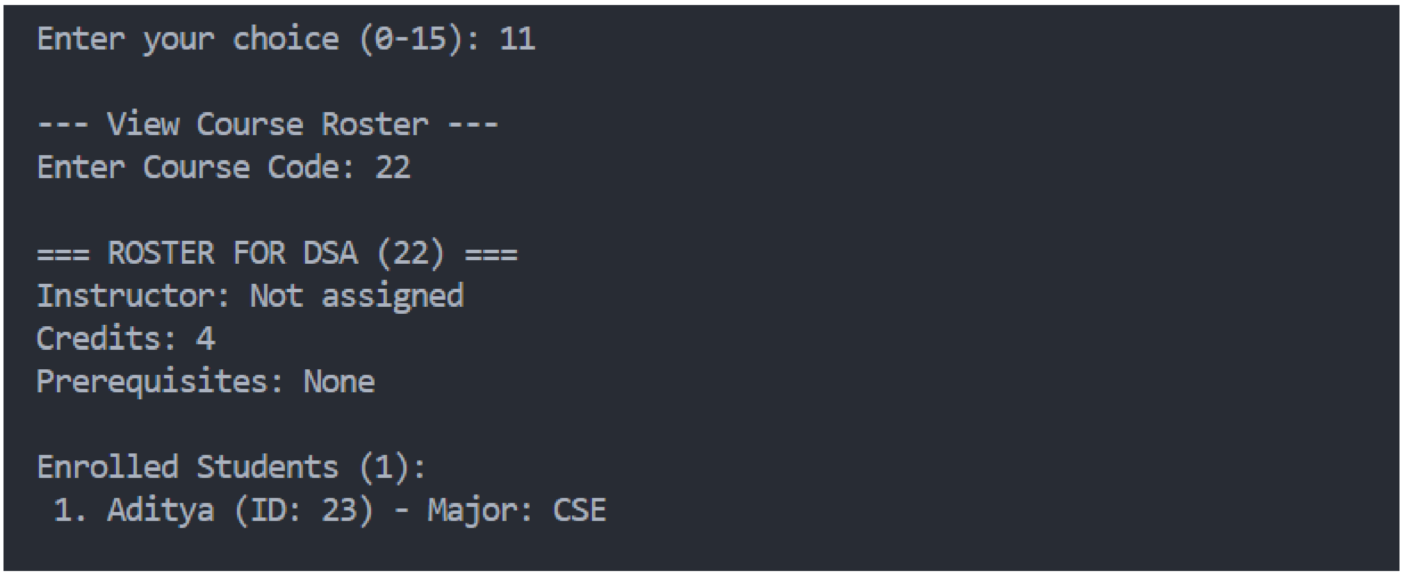


```
Enter your choice (0-15): 9

--- Assign Faculty to Course ---
Enter Faculty ID: 11
Enter Course Code: 22
Faculty Amit kumar assigned to DSA successfully!
```



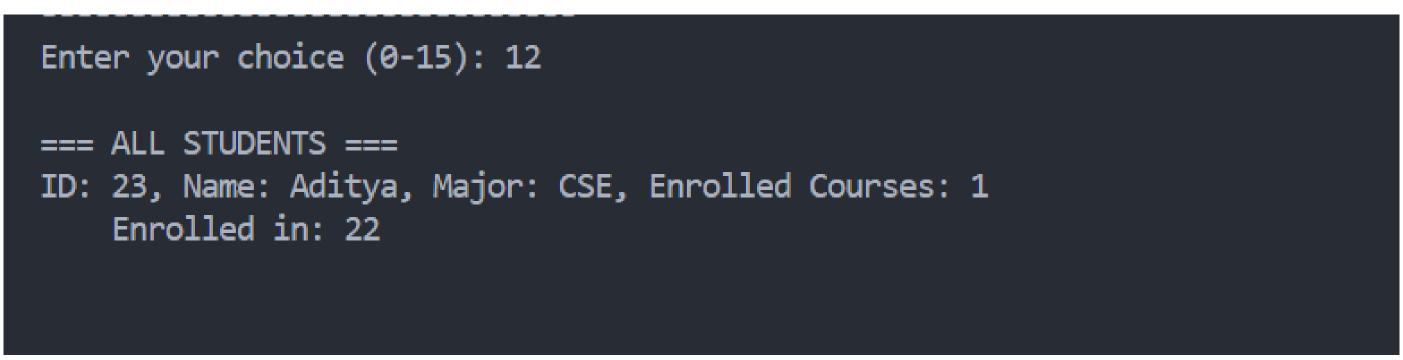
```
Enter your choice (0-15): 10

--- Unassign Faculty from Course ---
Enter Faculty ID: 11
Enter Course Code: 22
Faculty Amit kumar unassigned from DSA successfully!
```

- **View Roster:**



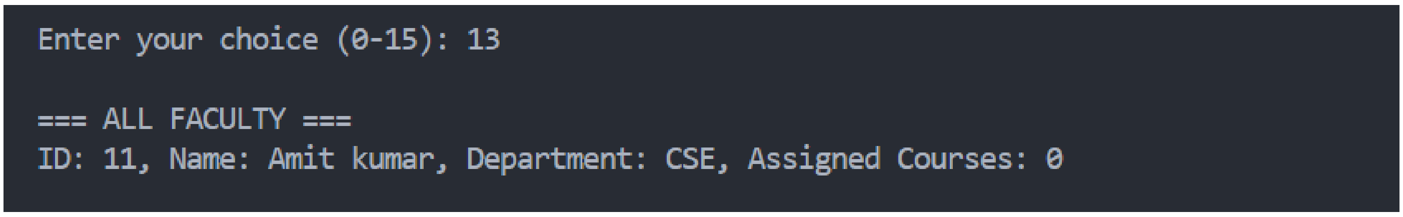
```
Enter your choice (0-15): 11

--- View Course Roster ---
Enter Course Code: 22

=== ROSTER FOR DSA (22) ===
Instructor: Not assigned
Credits: 4
Prerequisites: None

Enrolled Students (1):
 1. Aditya (ID: 23) - Major: CSE
```

- **View Students:**



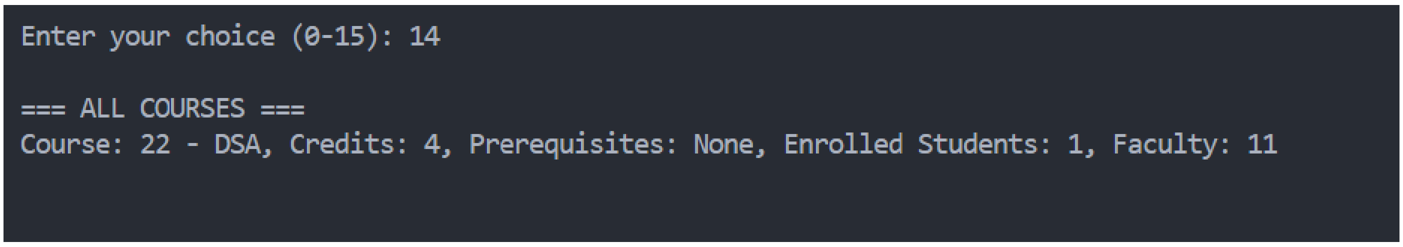
```
Enter your choice (0-15): 12

=== ALL STUDENTS ===
ID: 23, Name: Aditya, Major: CSE, Enrolled Courses: 1
    Enrolled in: 22
```

- **View Faculty:**



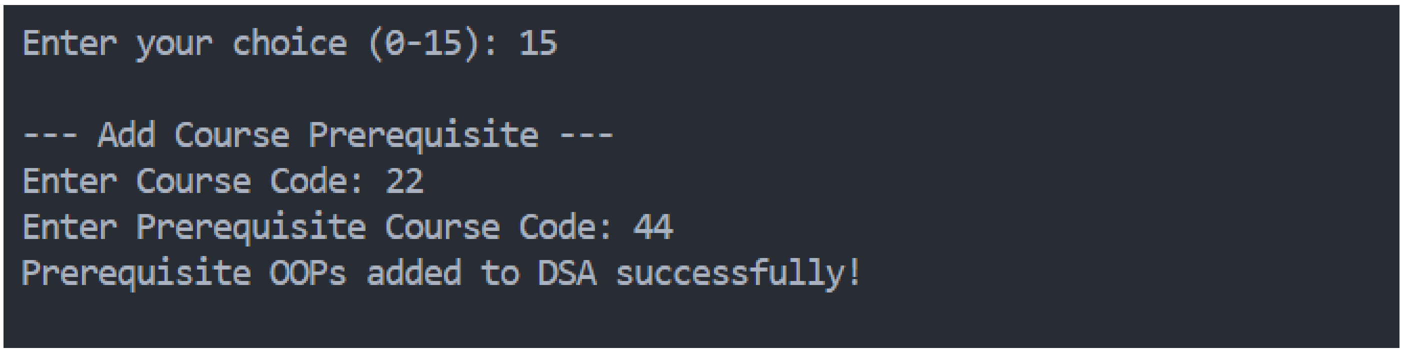
```
Enter your choice (0-15): 13

=== ALL FACULTY ===
ID: 11, Name: Amit kumar, Department: CSE, Assigned Courses: 0
```

- **View Course:**



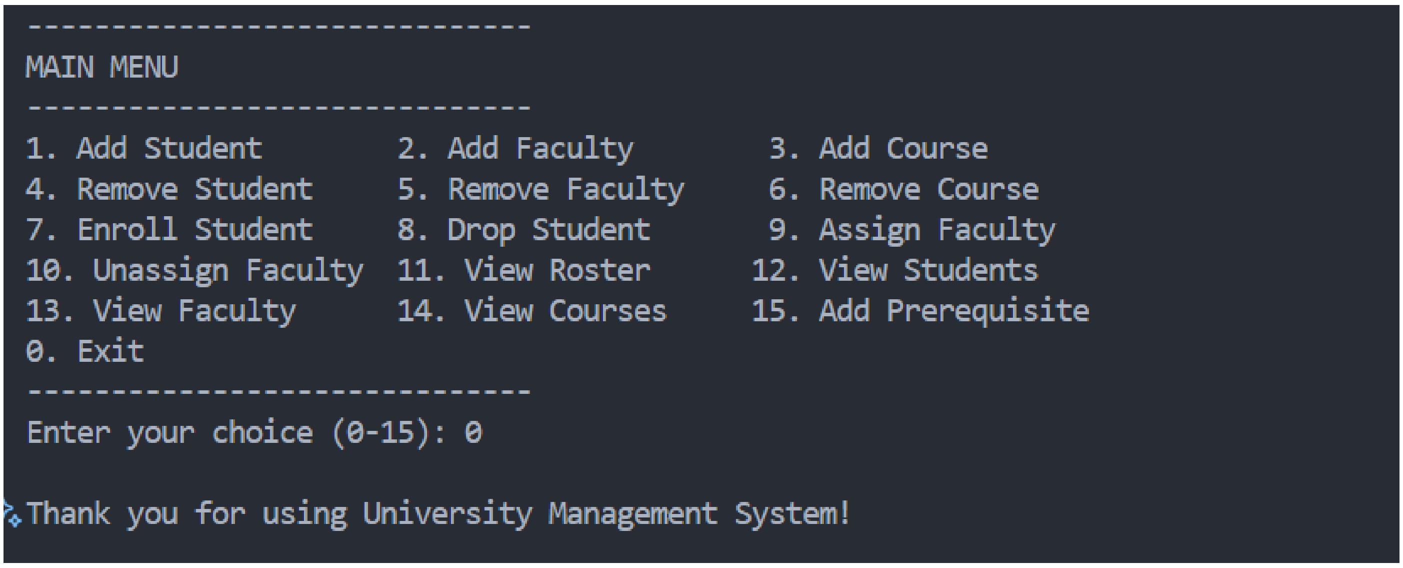
```
Enter your choice (0-15): 14

=== ALL COURSES ===
Course: 22 - DSA, Credits: 4, Prerequisites: None, Enrolled Students: 1, Faculty: 11
```

- **Add Prerequisites:**



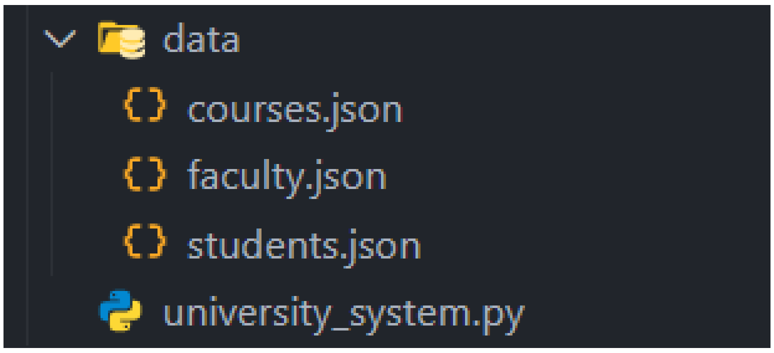
```
Enter your choice (0-15): 15

--- Add Course Prerequisite ---
Enter Course Code: 22
Enter Prerequisite Course Code: 44
Prerequisite OOPs added to DSA successfully!
```

- **Exit:**

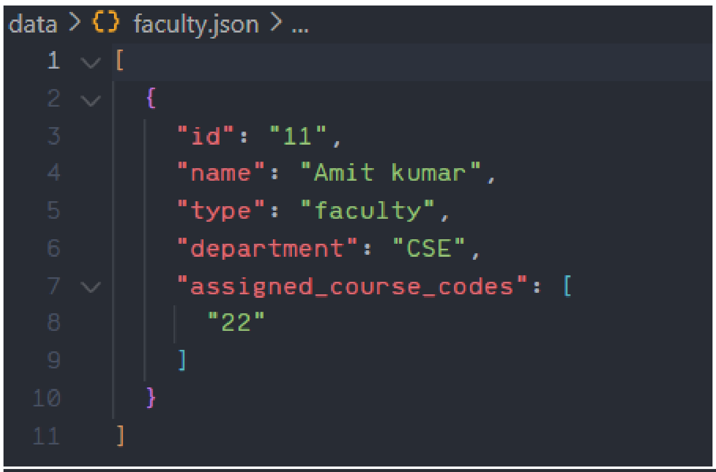


```
------------------------------
MAIN MENU
------------------------------
1. Add Student        2. Add Faculty        3. Add Course
4. Remove Student     5. Remove Faculty     6. Remove Course
7. Enroll Student     8. Drop Student       9. Assign Faculty
10. Unassign Faculty  11. View Roster      12. View Students
13. View Faculty      14. View Courses     15. Add Prerequisite
0. Exit
------------------------------
Enter your choice (0-15): 0

Thank you for using University Management System!
```

- **Data Storage:**

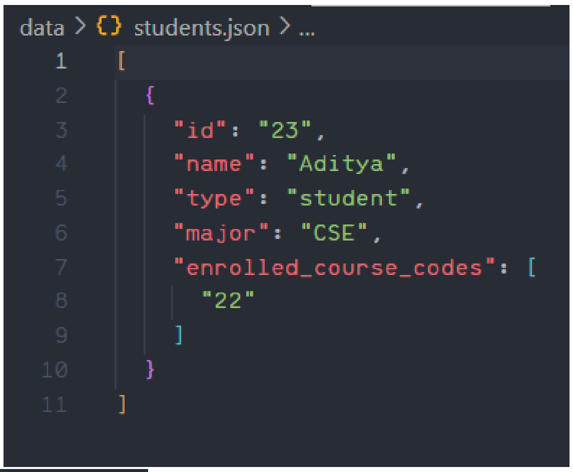


```
data
    courses.json
    faculty.json
    students.json
  university_system.py
```



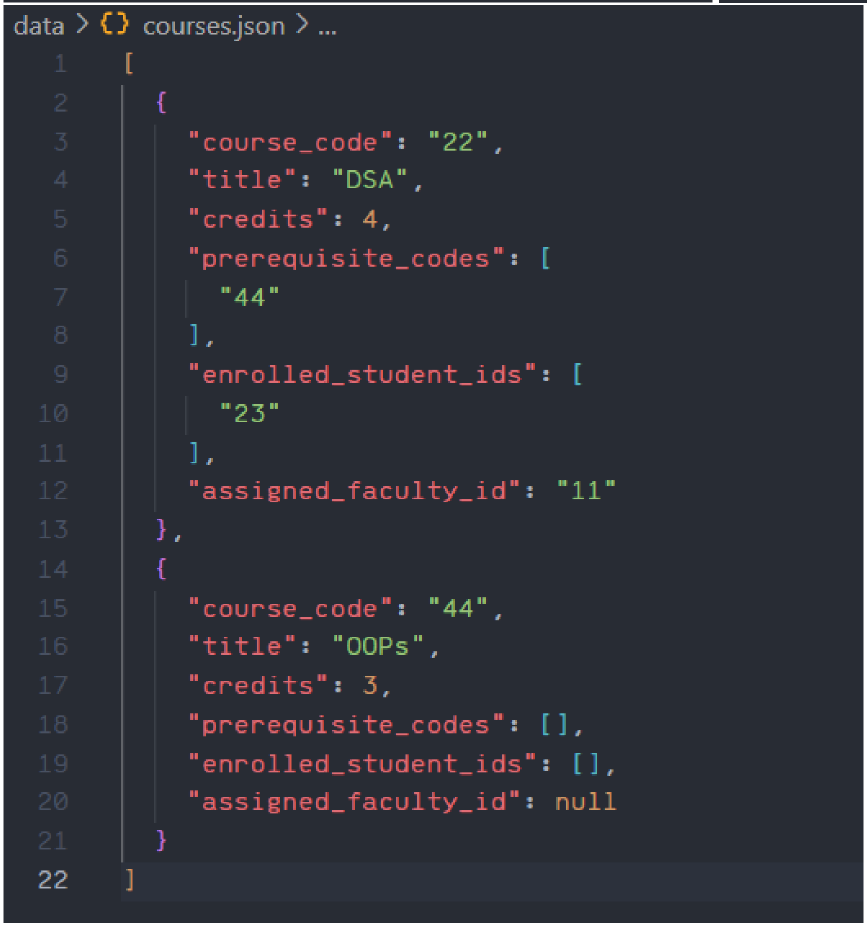
data > faculty.json > ...

```json
[
  {
    "id": "11",
    "name": "Amit kumar",
    "type": "faculty",
    "department": "CSE",
    "assigned_course_codes": [
      "22"
    ]
  }
]
```



data > students.json > ...

```json
[
  {
    "id": "23",
    "name": "Aditya",
    "type": "student",
    "major": "CSE",
    "enrolled_course_codes": [
      "22"
    ]
  }
]
```



data > courses.json > ...

```json
[
  {
    "course_code": "22",
    "title": "DSA",
    "credits": 4,
    "prerequisite_codes": [
      "44"
    ],
    "enrolled_student_ids": [
      "23"
    ],
    "assigned_faculty_id": "11"
  },
  {
    "course_code": "44",
    "title": "OOPs",
    "credits": 3,
    "prerequisite_codes": [],
    "enrolled_student_ids": [],
    "assigned_faculty_id": null
  }
]
```