OOPS Assignment Hexaware

NAME : THILAK RAAJ R A

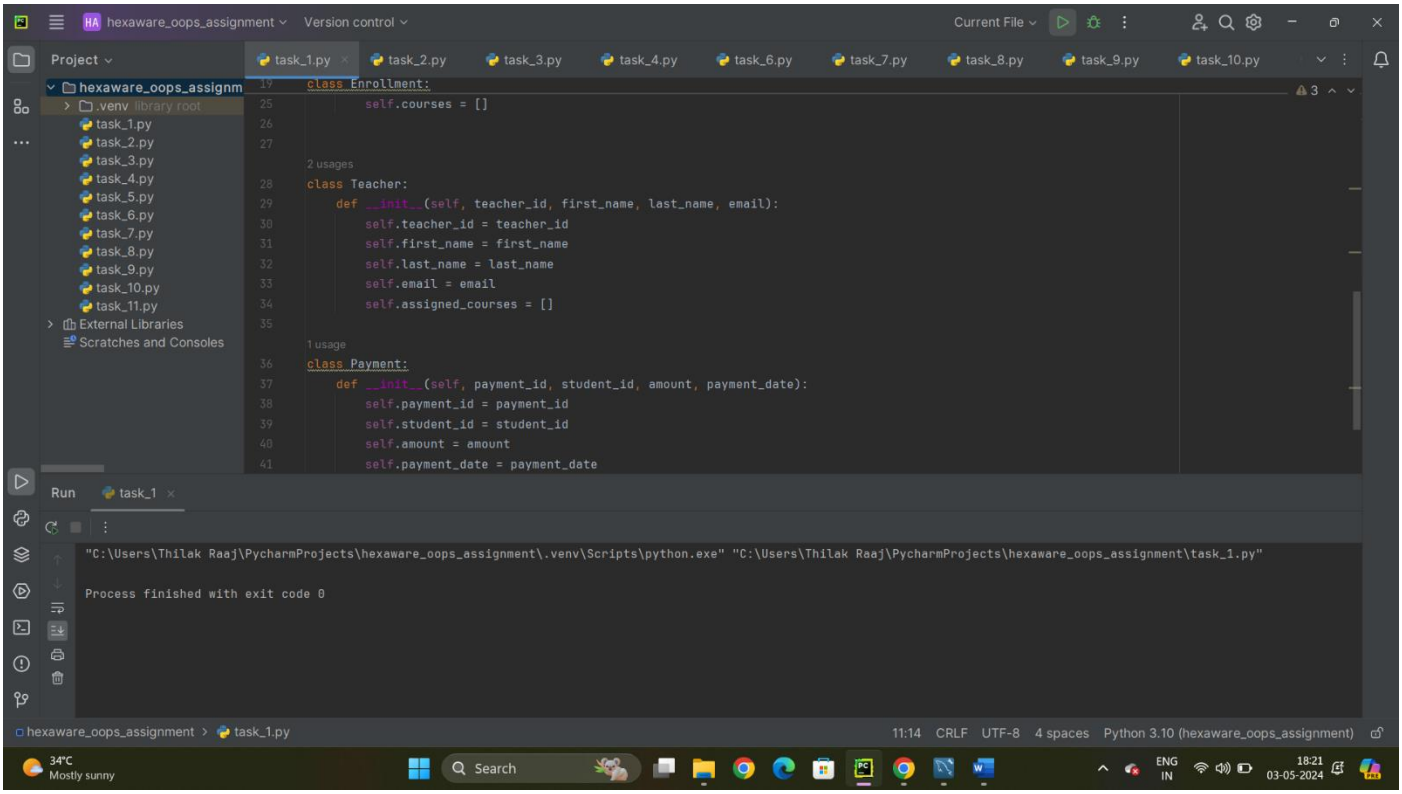
TOPIC : STUDENT INFORMATION SYSTEM

Task 1: Define Classes

Define the following classes based on the domain description: Student class with the following attributes: • Student ID • First Name • Last Name • Date of Birth • Email • Phone Number Course class with the following attributes: • Course ID • Course Name • Course Code • Instructor Name Enrollment class to represent the relationship between students and courses. It should have attributes: • Enrollment ID • Student ID (reference to a Student) • Course ID (reference to a Course) • Enrollment Date Teacher class with the following attributes: • Teacher ID • First Name • Last Name • Email Payment class with the following attributes: • Payment ID • Student ID (reference to a Student) • Amount • Payment Date

CODE:

```
class Student:
    def __init__(self, student_id, first_name, last_name, date_of_birth, email,
phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []
        self.payments = []
class Course:
    def __init__(self, course_id, course_name, course_code, instructor_name):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
        self.enrollments = []

class Enrollment:
    def __init__(self, enrollment_id, student, course, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student = student  # Reference to the student object
        self.course = course  # Reference to the course object
        self.enrollment_date = enrollment_date
        self.courses = []


class Teacher:
    def __init__(self, teacher_id, first_name, last_name, email):
        self.teacher_id = teacher_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.assigned_courses = []

class Payment:
    def __init__(self, payment_id, student_id, amount, payment_date):
        self.payment_id = payment_id
        self.student_id = student_id
        self.amount = amount
        self.payment_date = payment_date
```



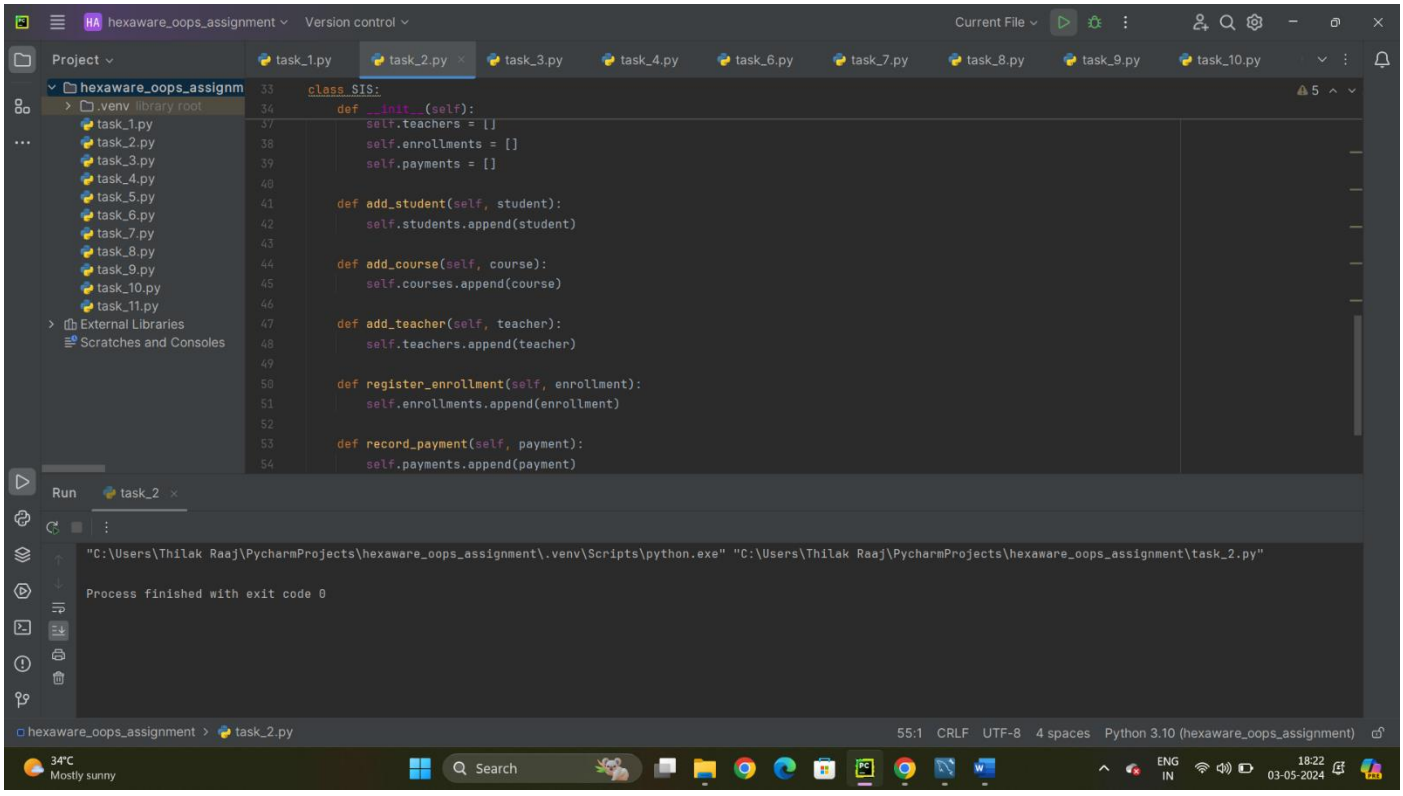
Task 2: Implement Constructors

Implement constructors for each class to initialize their attributes. Constructors are special methods that are called when an object of a class is created. They are used to set initial values for the attributes of the class. Below are detailed instructions on how to implement constructors for each class in your Student Information System (SIS) assignment: Student Class Constructor In the Student class, you need to create a constructor that initializes the attributes of a student when an instance of the Student class is created SIS Class Constructor If you have a class that represents the Student Information System itself (e.g., SIS class), you may also implement a constructor for it. This constructor can be used to set up any initial configuration for the SIS. Repeat the above process for each class Course, Enrollment, Teacher, Payment by defining constructors that initialize their respective attributes.

CODE:

```
class Student:
    def __init__(self, student_id, first_name, last_name, date_of_birth, email,
phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
class Course:
    def __init__(self, course_id, course_name, course_code, instructor_name):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
class Enrollment:
    def __init__(self, enrollment_id, student_id, course_id, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student_id = student_id
        self.course_id = course_id
        self.enrollment_date = enrollment_date
class Teacher:
    def __init__(self, teacher_id, first_name, last_name, email):
        self.teacher_id = teacher_id
        self.first_name = first_name
        self.last_name = last_name
```

```
        self.email = email
class Payment:
    def __init__(self, payment_id, student_id, amount, payment_date):
        self.payment_id = payment_id
        self.student_id = student_id
        self.amount = amount
        self.payment_date = payment_date
class SIS:
    def __init__(self):
        self.students = []
        self.courses = []
        self.teachers = []
        self.enrollments = []
        self.payments = []

    def add_student(self, student):
        self.students.append(student)

    def add_course(self, course):
        self.courses.append(course)

    def add_teacher(self, teacher):
        self.teachers.append(teacher)

    def register_enrollment(self, enrollment):
        self.enrollments.append(enrollment)

    def record_payment(self, payment):
        self.payments.append(payment)
```



Task 3: Implement Methods

Implement methods in your classes to perform various operations related to the Student Information System (SIS). These methods will allow you to interact with and manipulate data within your system. Below are detailed instructions on how to implement methods in each class: Implement the following methods in the appropriate classes: Student Class: • EnrollInCourse(course: Course): Enrolls the student in a course. • UpdateStudentInfo(firstName: string, lastName: string, dateOfBirth: DateTime, email: string, phoneNumber: string): Updates the student's information. • MakePayment(amount: decimal, paymentDate: DateTime): Records a payment

made by the student. • DisplayStudentInfo(): Displays detailed information about the student. • GetEnrolledCourses(): Retrieves a list of courses in which the student is enrolled. • GetPaymentHistory(): Retrieves a list of payment records for the student. Course Class: • AssignTeacher(teacher: Teacher): Assigns a teacher to the course. • UpdateCourseInfo(courseCode: string, courseName: string, instructor: string): Updates course information. • DisplayCourseInfo(): Displays detailed information about the course. • GetEnrollments(): Retrieves a list of student enrollments for the course. • GetTeacher(): Retrieves the assigned teacher for the course. Enrollment Class: • GetStudent(): Retrieves the student associated with the enrollment. • GetCourse(): Retrieves the course associated with the enrollment. Teacher Class: • UpdateTeacherInfo(name: string, email: string, expertise: string): Updates teacher information. • DisplayTeacherInfo(): Displays detailed information about the teacher. • GetAssignedCourses(): Retrieves a list of courses assigned to the teacher. Payment Class: • GetStudent(): Retrieves the student associated with the payment. • GetPaymentAmount(): Retrieves the payment amount. • GetPaymentDate(): Retrieves the payment date. SIS Class (if you have one to manage interactions): • EnrollStudentInCourse(student: Student, course: Course): Enrolls a student in a course. • AssignTeacherToCourse(teacher: Teacher, course: Course): Assigns a teacher to a course. • RecordPayment(student: Student, amount: decimal, paymentDate: DateTime): Records a payment made by a student. • GenerateEnrollmentReport(course: Course): Generates a report of students enrolled in a specific course. • GeneratePaymentReport(student: Student): Generates a report of payments made by a specific student. • CalculateCourseStatistics(course: Course): Calculates statistics for a specific course, such as the number of enrollments and total payments. Use the Methods In your driver program or any part of your code where you want to perform actions related to the Student Information System, create instances of your classes, and use the methods you've implemented. Repeat this process for using other methods you've implemented in your classes and the SIS class.

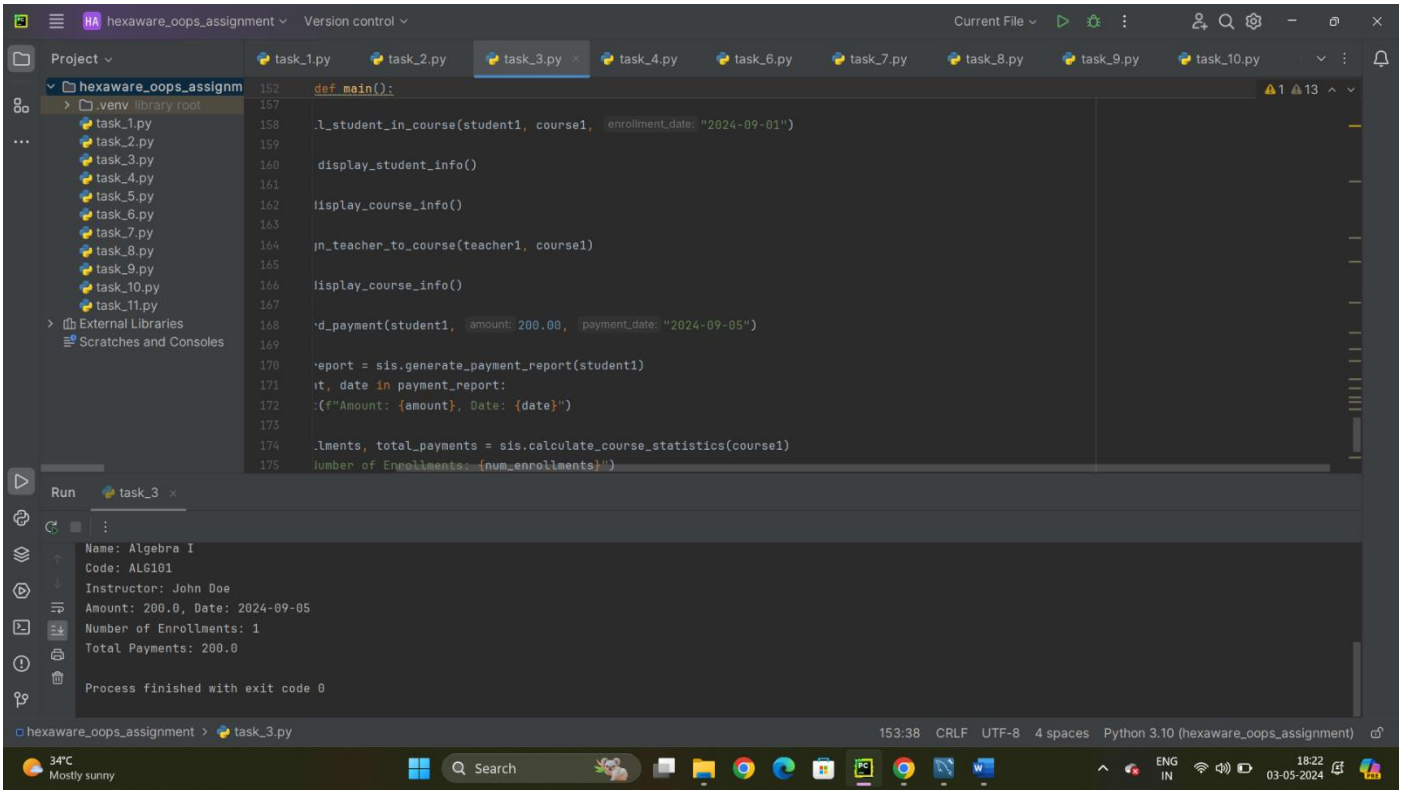
CODE:

```
class Student:
    def __init__(self, student_id, first_name, last_name, date_of_birth, email,
phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []
        self.payments = []

    def enroll_in_course(self, course):
        enrollment = Enrollment(self, course)
        self.enrollments.append(enrollment)
        course.enrollments.append(enrollment)

    def update_student_info(self, first_name, last_name, date_of_birth, email,
phone_number):
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number

    def make_payment(self, amount, payment_date):
        payment = Payment(self, amount, payment_date)
        self.payments.append(payment)

    def display_student_info(self):
        print(f"Student ID: {self.student_id}")
        print(f"Name: {self.first_name} {self.last_name}")
        print(f"Date of Birth: {self.date_of_birth}")
        print(f"Email: {self.email}")
        print(f"Phone Number: {self.phone_number}")

    def get_enrolled_courses(self):
```

```
        return [enrollment.course for enrollment in self.enrollments]

    def get_payment_history(self):
        return [(payment.amount, payment.payment_date) for payment in self.payments]

class Course:
    def __init__(self, course_id, course_name, course_code, instructor_name):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
        self.enrollments = []

    def assign_teacher(self, teacher):
        self.instructor_name = f"{teacher.first_name} {teacher.last_name}"

    def update_course_info(self, course_code, course_name, instructor_name):
        self.course_code = course_code
        self.course_name = course_name
        self.instructor_name = instructor_name

    def display_course_info(self):
        print(f"Course ID: {self.course_id}")
        print(f"Name: {self.course_name}")
        print(f"Code: {self.course_code}")
        print(f"Instructor: {self.instructor_name}")

    def get_enrollments(self):
        return [enrollment.student for enrollment in self.enrollments]

    def get_teacher(self):
        return self.instructor_name

class Enrollment:
    def __init__(self, student, course, enrollment_date):
        self.student = student
        self.course = course
        self.enrollment_date = enrollment_date

    def get_student(self):
        return self.student

    def get_course(self):
        return self.course

class Teacher:
    def __init__(self, teacher_id, first_name, last_name, email):
        self.teacher_id = teacher_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.courses_assigned = []

    def update_teacher_info(self, first_name, last_name, email):
        self.first_name = first_name
        self.last_name = last_name
        self.email = email

    def display_teacher_info(self):
        print(f"Teacher ID: {self.teacher_id}")
        print(f"Name: {self.first_name} {self.last_name}")
        print(f"Email: {self.email}")

    def get_assigned_courses(self):
        return self.courses_assigned

class Payment:
```

```
    def __init__(self, student, amount, payment_date):
        self.student = student
        self.amount = amount
        self.payment_date = payment_date

    def get_student(self):
        return self.student

    def get_payment_amount(self):
        return self.amount

    def get_payment_date(self):
        return self.payment_date

class SIS:
    def __init__(self):
        self.students = []
        self.courses = []
        self.teachers = []
        self.enrollments = []
        self.payments = []

    def enroll_student_in_course(self, student, course, enrollment_date):
        enrollment = Enrollment(student, course, enrollment_date)
        student.enrollments.append(enrollment)
        course.enrollments.append(enrollment)

    def assign_teacher_to_course(self, teacher, course):
        teacher.courses_assigned.append(course)
        course.instructor_name = f"{teacher.first_name} {teacher.last_name}"

    def record_payment(self, student, amount, payment_date):
        payment = Payment(student, amount, payment_date)
        student.payments.append(payment)
        self.payments.append(payment)

    def generate_enrollment_report(self, course):
        enrolled_students = [enrollment.student for enrollment in course.enrollments]
        return enrolled_students

    def generate_payment_report(self, student):
        student_payments = [(payment.amount, payment.payment_date) for payment in
student.payments]
        return student_payments

    def calculate_course_statistics(self, course):
        num_enrollments = len(course.enrollments)
        total_payments = sum([payment.amount for enrollment in course.enrollments for
payment in enrollment.student.payments])
        return num_enrollments, total_payments

def main():
    student1 = Student("S001", "Alice", "Smith", "2005-08-15", "alice@example.com",
"555-0101")
    course1 = Course("C001", "Algebra I", "ALG101", "")
    teacher1 = Teacher("T001", "John", "Doe", "john.doe@example.com")
    sis = SIS()

    sis.enroll_student_in_course(student1, course1, "2024-09-01")

    student1.display_student_info()

    course1.display_course_info()

    sis.assign_teacher_to_course(teacher1, course1)

    course1.display_course_info()
```

```
    sis.record_payment(student1, 200.00, "2024-09-05")

    payment_report = sis.generate_payment_report(student1)
    for amount, date in payment_report:
        print(f"Amount: {amount}, Date: {date}")

    num_enrollments, total_payments = sis.calculate_course_statistics(course1)
    print(f"Number of Enrollments: {num_enrollments}")
    print(f"Total Payments: {total_payments}")

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



Task 4: Exceptions handling and Custom Exceptions Implementing

custom exceptions allows you to define and throw exceptions tailored to specific situations or business logic requirements. Create Custom Exception Classes You'll need to create custom exception classes that are inherited from the System.Exception class or one of its derived classes (e.g., System.ApplicationException). These custom exception classes will allow you to encapsulate specific error scenarios and provide meaningful error messages. Throw Custom Exceptions In your code, you can throw custom exceptions when specific conditions or business logic rules are violated. To throw a custom exception, use the throw keyword followed by an instance of your custom exception class. • DuplicateEnrollmentException: Thrown when a student is already enrolled in a course and tries to enroll again. This exception can be used in the EnrollStudentInCourse method. • CourseNotFoundException: Thrown when a course does not exist in the system, and you attempt to perform operations on it (e.g., enrolling a student or assigning a teacher).• StudentNotFoundException: Thrown when a student does not exist in the system, and you attempt to perform operations on the student (e.g., enrolling in a course, making a payment). • TeacherNotFoundException: Thrown when a teacher does not exist in the system, and you attempt to assign them to a course. • PaymentValidationException: Thrown when there is an issue with payment validation, such as an invalid payment amount or payment date. • InvalidStudentDataException: Thrown when data provided for creating or updating a student is invalid (e.g., invalid date of birth or email format). • InvalidCourseDataException: Thrown when data provided for creating or updating a course is invalid (e.g., invalid course code or instructor name). • InvalidEnrollmentDataException: Thrown when data provided for creating an enrollment is invalid (e.g., missing student or course references). • InvalidTeacherDataException: Thrown when data provided for creating or updating a

teacher is invalid (e.g., missing name or email). • InsufficientFundsException: Thrown when a student attempts to enroll in a course but does not have enough funds to make the payment.

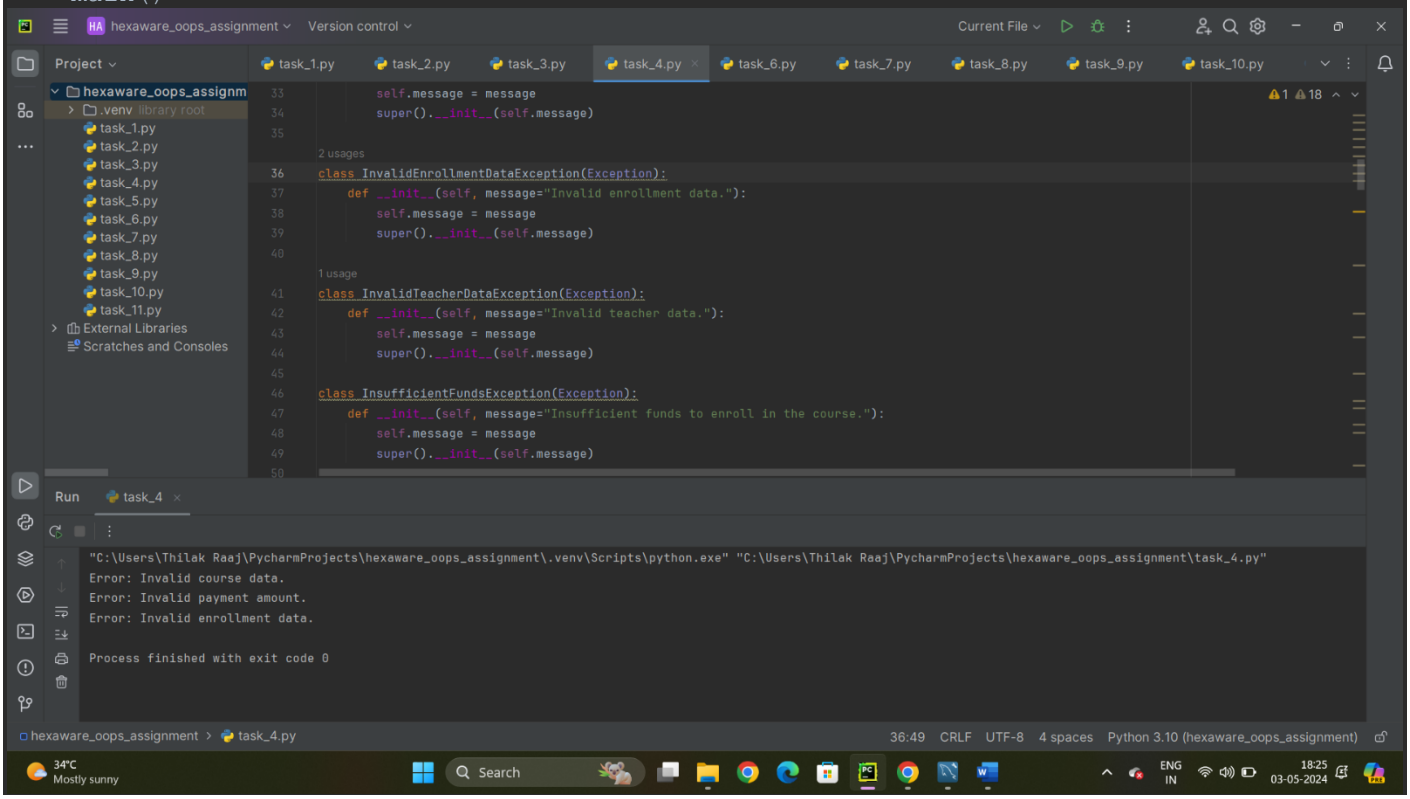
CODE:

```
class DuplicateEnrollmentException(Exception):
    def __init__(self, message="Student is already enrolled in the course."):
        self.message = message
        super().__init__(self.message)

class CourseNotFoundException(Exception):
    def __init__(self, message="Course not found."):
        self.message = message
        super().__init__(self.message)

class StudentNotFoundException(Exception):
    def __init__(self, message="Student not found."):
        self.message = message
        super().__init__(self.message)

class TeacherNotFoundException(Exception):
    def __init__(self, message="Teacher not found."):
        self.message = message
        super().__init__(self.message)

class PaymentValidationException(Exception):
    def __init__(self, message="Payment validation failed."):
        self.message = message
        super().__init__(self.message)

class InvalidStudentDataException(Exception):
    def __init__(self, message="Invalid student data."):
        self.message = message
        super().__init__(self.message)

class InvalidCourseDataException(Exception):
    def __init__(self, message="Invalid course data."):
        self.message = message
        super().__init__(self.message)

class InvalidEnrollmentDataException(Exception):
    def __init__(self, message="Invalid enrollment data."):
        self.message = message
        super().__init__(self.message)

class InvalidTeacherDataException(Exception):
    def __init__(self, message="Invalid teacher data."):
        self.message = message
        super().__init__(self.message)

class InsufficientFundsException(Exception):
    def __init__(self, message="Insufficient funds to enroll in the course."):
        self.message = message
        super().__init__(self.message)


class Student:
    def __init__(self, student_id, first_name, last_name, date_of_birth, email,
phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []
        self.payments = []
```

```
    def enroll_in_course(self, course):
        if course in self.get_enrolled_courses():
            raise DuplicateEnrollmentException()
        enrollment = Enrollment(self, course)
        self.enrollments.append(enrollment)
        course.enrollments.append(enrollment)

    def update_student_info(self, first_name, last_name, date_of_birth, email,
phone_number):
        if not first_name or not last_name or not date_of_birth or not email or not
phone_number:
            raise InvalidStudentDataException()
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number

    def make_payment(self, amount, payment_date):
        if amount <= 0:
            raise PaymentValidationException("Invalid payment amount.")
        payment = Payment(self, amount, payment_date)
        self.payments.append(payment)

    def display_student_info(self):
        print(f"Student ID: {self.student_id}")
        print(f"Name: {self.first_name} {self.last_name}")
        print(f"Date of Birth: {self.date_of_birth}")
        print(f"Email: {self.email}")
        print(f"Phone Number: {self.phone_number}")

    def get_enrolled_courses(self):
        return [enrollment.course for enrollment in self.enrollments]

    def get_payment_history(self):
        return [(payment.amount, payment.payment_date) for payment in self.payments]

class Course:
    def __init__(self, course_id, course_name, course_code, instructor_name):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
        self.enrollments = []

    def assign_teacher(self, teacher):
        self.instructor_name = f"{teacher.first_name} {teacher.last_name}"

    def update_course_info(self, course_code, course_name, instructor_name):
        # Validation logic can be added here
        if not course_code or not course_name or not instructor_name:
            raise InvalidCourseDataException()
        self.course_code = course_code
        self.course_name = course_name
        self.instructor_name = instructor_name

    def display_course_info(self):
        print(f"Course ID: {self.course_id}")
        print(f"Name: {self.course_name}")
        print(f"Code: {self.course_code}")
        print(f"Instructor: {self.instructor_name}")

    def get_enrollments(self):
        return [enrollment.student for enrollment in self.enrollments]

    def get_teacher(self):
```

```
        return self.instructor_name

class Enrollment:
    def __init__(self, student, course, enrollment_date):
        # Validation logic can be added here
        if not student or not course:
            raise InvalidEnrollmentDataException()
        self.student = student
        self.course = course
        self.enrollment_date = enrollment_date

    def get_student(self):
        return self.student

    def get_course(self):
        return self.course

class Teacher:
    def __init__(self, teacher_id, first_name, last_name, email):
        self.teacher_id = teacher_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.courses_assigned = []

    def update_teacher_info(self, first_name, last_name, email):

        if not first_name or not last_name or not email:
            raise InvalidTeacherDataException()
        self.first_name = first_name
        self.last_name = last_name
        self.email = email

    def display_teacher_info(self):
        print(f"Teacher ID: {self.teacher_id}")
        print(f"Name: {self.first_name} {self.last_name}")
        print(f"Email: {self.email}")

    def get_assigned_courses(self):
        return self.courses_assigned

class Payment:
    def __init__(self, student, amount, payment_date):

        if amount <= 0:
            raise PaymentValidationException("Invalid payment amount.")
        self.student = student
        self.amount = amount
        self.payment_date = payment_date

    def get_student(self):
        return self.student

    def get_payment_amount(self):
        return self.amount

    def get_payment_date(self):
        return self.payment_date

class SIS:
    def __init__(self):
        self.students = []

    def enroll_student_in_course(self, student, course, enrollment_date):
        enrollment = Enrollment(student, course, enrollment_date)
        student.enrollments.append(enrollment)
        course.enrollments.append(enrollment)
```

```
def main():
    student1 = Student("S001", "Alice", "Smith", "2005-08-15", "alice@example.com",
"555-0101")
    course1 = Course("C001", "Algebra I", "ALG101", "")
    teacher1 = Teacher("T001", "John", "Doe", "john.doe@example.com")
    sis = SIS()

    try:

        sis.enroll_student_in_course(student1, course1, "2024-09-01")
        sis.enroll_student_in_course(student1, course1, "2024-09-01")
    except DuplicateEnrollmentException as e:
        print(f"Error: {e}")

    try:

        course1.update_course_info("", "", "")
    except InvalidCourseDataException as e:
        print(f"Error: {e}")

    try:

        student1.make_payment(-100, "2024-09-05")
    except PaymentValidationException as e:
        print(f"Error: {e}")

    try:
        Enrollment(None, None, None)
    except InvalidEnrollmentDataException as e:
        print(f"Error: {e}")

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



Task 5: Collections

Implement Collections:

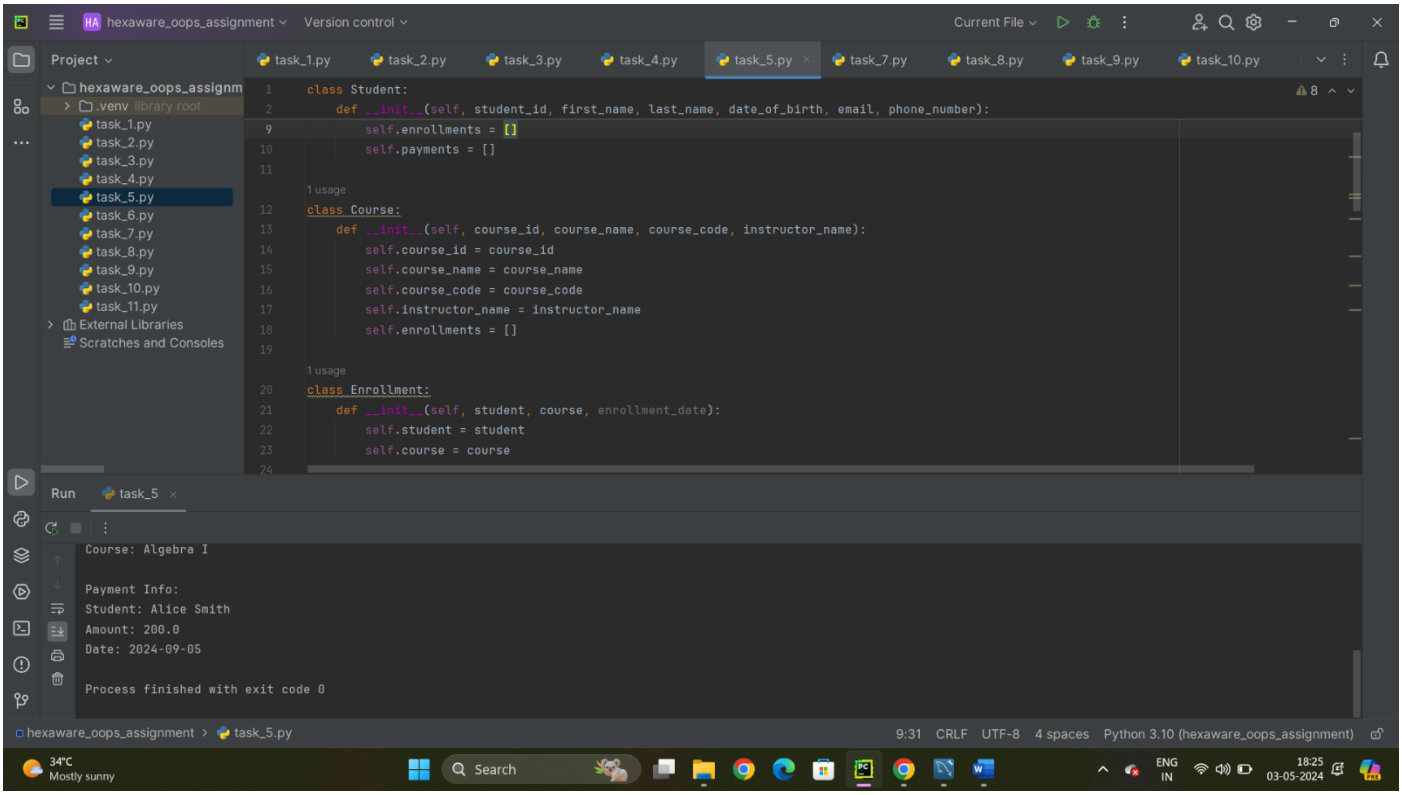
Implement relationships between classes using appropriate data structures (e.g., lists or dictionaries) to maintain associations between students, courses, enrollments, teachers, and payments. These relationships are essential for

the Student Information System (SIS) to track and manage student enrollments, teacher assignments, and payments accurately. Define Class-Level Data Structures You will need class-level data structures within each class to maintain relationships. Here's how to define them for each class: Student Class: Create a list or collection property to store the student's enrollments. This property will hold references to Enrollment objects. Example: List Enrollments { get; set; } Course Class: Create a list or collection property to store the course's enrollments. This property will hold references to Enrollment objects. Example: List Enrollments { get; set; } Enrollment Class: Include properties to hold references to both the Student and Course objects. Example: Student Student { get; set; } and Course Course { get; set; } Teacher Class: Create a list or collection property to store the teacher's assigned courses. This property will hold references to Course objects. Example: List AssignedCourses { get; set; } Payment Class: Include a property to hold a reference to the Student object. Example: Student Student { get; set; } Update Constructor(s) In the constructors of your classes, initialize the list or collection properties to create empty collections when an object is instantiated. Repeat this for the Course, Teacher, and Payment classes, where applicable.

CODE:

```
class Student:
    def __init__(self, student_id, first_name, last_name, date_of_birth, email,
phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []
        self.payments = []

class Course:
    def __init__(self, course_id, course_name, course_code, instructor_name):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
        self.enrollments = []

class Enrollment:
    def __init__(self, student, course, enrollment_date):
        self.student = student
        self.course = course

class Teacher:
    def __init__(self, teacher_id, first_name, last_name, email):
        self.teacher_id = teacher_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.assigned_courses = []

class Payment:
    def __init__(self, student, amount, payment_date):
        self.student = student
        self.amount = amount
        self.payment_date = payment_date

def main():
    student1 = Student("S001", "Alice", "Smith", "2005-08-15", "alice@example.com",
"555-0101")
    course1 = Course("C001", "Algebra I", "ALG101", "John Doe")
    teacher1 = Teacher("T001", "John", "Doe", "john.doe@example.com")
    payment1 = Payment(student1, 200.00, "2024-09-05")

    enrollment1 = Enrollment(student1, course1, "2024-09-01")

    course1.instructor_name = f"{teacher1.first_name} {teacher1.last_name}"
```

```
    teacher1.assigned_courses.append(course1)

    student1.payments.append(payment1)

    print("Student Info:")
    print(f"Name: {student1.first_name} {student1.last_name}")
    print("Enrollments:")
    for enrollment in student1.enrollments:
        print(f"Course: {enrollment.course.course_name}")

    print("\nCourse Info:")
    print(f"Name: {course1.course_name}")
    print(f"Instructor: {course1.instructor_name}")
    print("Enrollments:")
    for enrollment in course1.enrollments:
        print(f"Student: {enrollment.student.first_name}
{enrollment.student.last_name}")

    print("\nTeacher Info:")
    print(f"Name: {teacher1.first_name} {teacher1.last_name}")
    print("Assigned Courses:")
    for course in teacher1.assigned_courses:
        print(f"Course: {course.course_name}")

    print("\nPayment Info:")
    print(f"Student: {payment1.student.first_name} {payment1.student.last_name}")
    print(f"Amount: {payment1.amount}")
    print(f"Date: {payment1.payment_date}")

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



Task 6: Create Methods for Managing Relationships

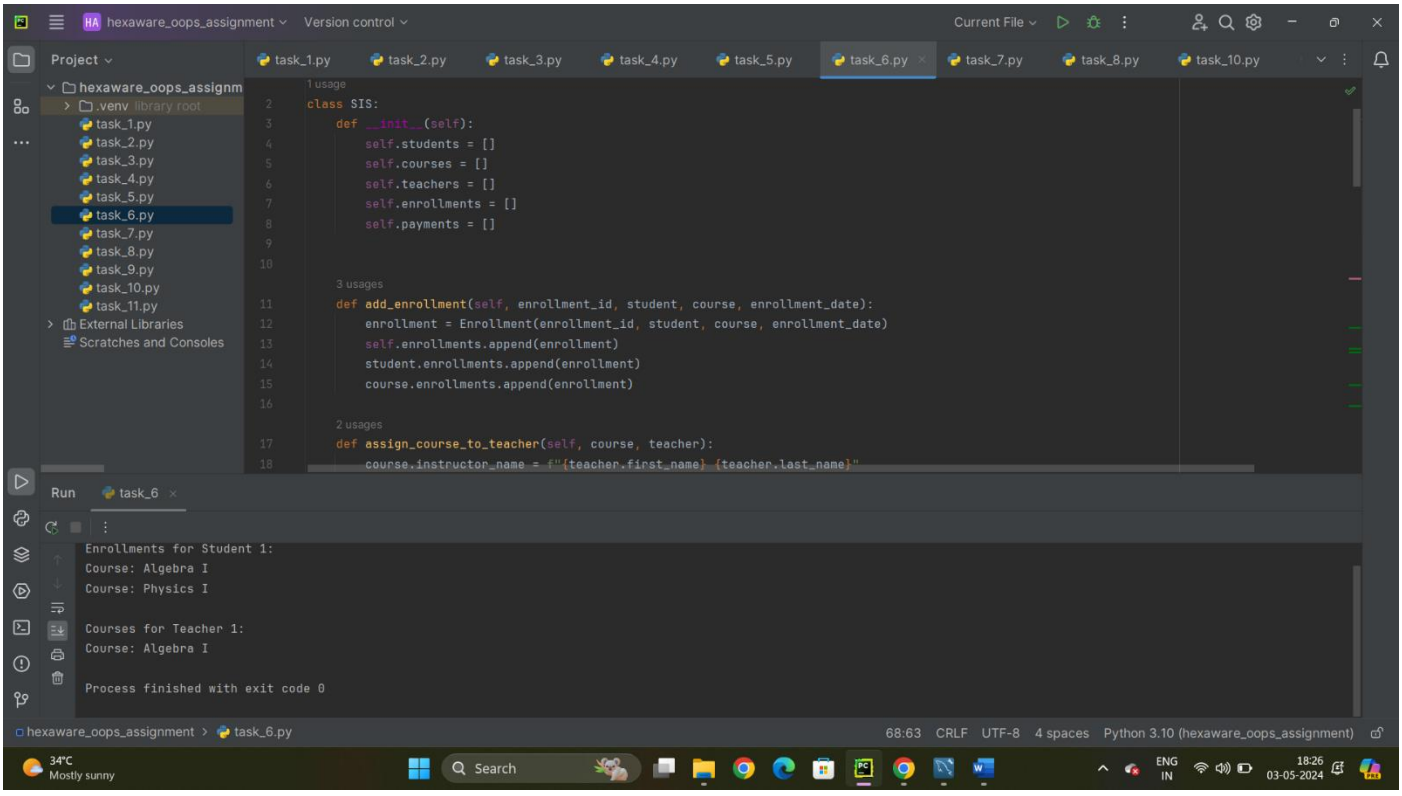
To add, remove, or retrieve related objects, you should create methods within your SIS class or each relevant class. • AddEnrollment(student, course, enrollmentDate): In the SIS class, create a method that adds an enrollment to both the Student's and Course's enrollment lists. Ensure the Enrollment object references the correct Student and Course. • AssignCourseToTeacher(course, teacher): In the SIS class, create a method to assign a course to a teacher. Add the

course to the teacher's AssignedCourses list. • AddPayment(student, amount, paymentDate): In the SIS class, create a method that adds a payment to the Student's payment history. Ensure the Payment object references the correct Student. • GetEnrollmentsForStudent(student): In the SIS class, create a method to retrieve all enrollments for a specific student. • GetCoursesForTeacher(teacher): In the SIS class, create a method to retrieve all courses assigned to a specific teacher. Create a Driver Program A driver program (also known as a test program or main program) is essential for testing and demonstrating the functionality of your classes and methods within your Student Information System (SIS) assignment. In this task, you will create a console application that serves as the entry point for your SIS and allows you to interact with and test your implemented classes and methods. Add References to Your SIS Classes Ensure that your SIS classes (Student, Course, Enrollment, Teacher, Payment) and the SIS class (if you have one to manage interactions) are defined in separate files within your project or are referenced properly. If you have defined these classes in separate files, make sure to include using statements in your driver program to access them: Implement the Main Method In the console application, the Main method serves as the entry point for your program. This is where you will create instances of your classes, call methods, and interact with your Student Information System. In the Main method, you create instances of your classes (e.g., Student, Course, and SIS) and then interact with your Student Information System by calling methods and handling exceptions.

CODE:

```
from task_1 import *
class SIS:
    def __init__(self):
        self.students = []
        self.courses = []
        self.teachers = []
        self.enrollments = []
        self.payments = []


    def add_enrollment(self, enrollment_id, student, course, enrollment_date):
        enrollment = Enrollment(enrollment_id, student, course, enrollment_date)
        self.enrollments.append(enrollment)
        student.enrollments.append(enrollment)
        course.enrollments.append(enrollment)

    def assign_course_to_teacher(self, course, teacher):
        course.instructor_name = f"{teacher.first_name} {teacher.last_name}"
        teacher.assigned_courses.append(course)

    def add_payment(self, payment_id, student, amount, payment_date):
        payment = Payment(payment_id, student, amount, payment_date)
        self.payments.append(payment)
        student.payments.append(payment)

    def get_enrollments_for_student(self, student):
        return [enrollment for enrollment in self.enrollments if enrollment.student ==
student]

    def get_courses_for_teacher(self, teacher):
        return [course for course in self.courses if course.instructor_name ==
f"{teacher.first_name} {teacher.last_name}"]

def main():

    sis = SIS()


    student1 = Student("S001", "Alice", "Smith", "2005-08-15", "alice@example.com",
"555-0101")
    student2 = Student("S002", "Bob", "Jones", "2006-03-20", "bob@example.com", "555-
0202")


    Course1 = Course("C001", "Algebra I", "ALG101", "")
    Course2 = Course("C002", "Physics I", "PHY101", "")
```

```
    teacher1 = Teacher("T001", "John", "Doe", "john.doe@example.com")
    teacher2 = Teacher("T002", "Jane", "Smith", "jane.smith@example.com")


    sis.students.extend([student1, student2])
    sis.courses.extend([Course1, Course2])
    sis.teachers.extend([teacher1, teacher2])


    sis.add_enrollment(1, student1, Course1, "2024-09-01")
    sis.add_enrollment(2, student1, Course2, "2024-09-01")
    sis.add_enrollment(3, student2, Course1, "2024-09-01")


    sis.assign_course_to_teacher(Course1, teacher1)
    sis.assign_course_to_teacher(Course2, teacher2)


    sis.add_payment(1, student1, 200.00, "2024-09-05")
    sis.add_payment(2, student2, 150.00, "2024-09-08")


    print("Enrollments for Student 1:")
    for enrollment in sis.get_enrollments_for_student(student1):
        print(f"Course: {enrollment.course.course_name}")
    print("\nCourses for Teacher 1:")
    for course in sis.get_courses_for_teacher(teacher1):
        print(f"Course: {course.course_name}")

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



Task 7: Database Connectivity

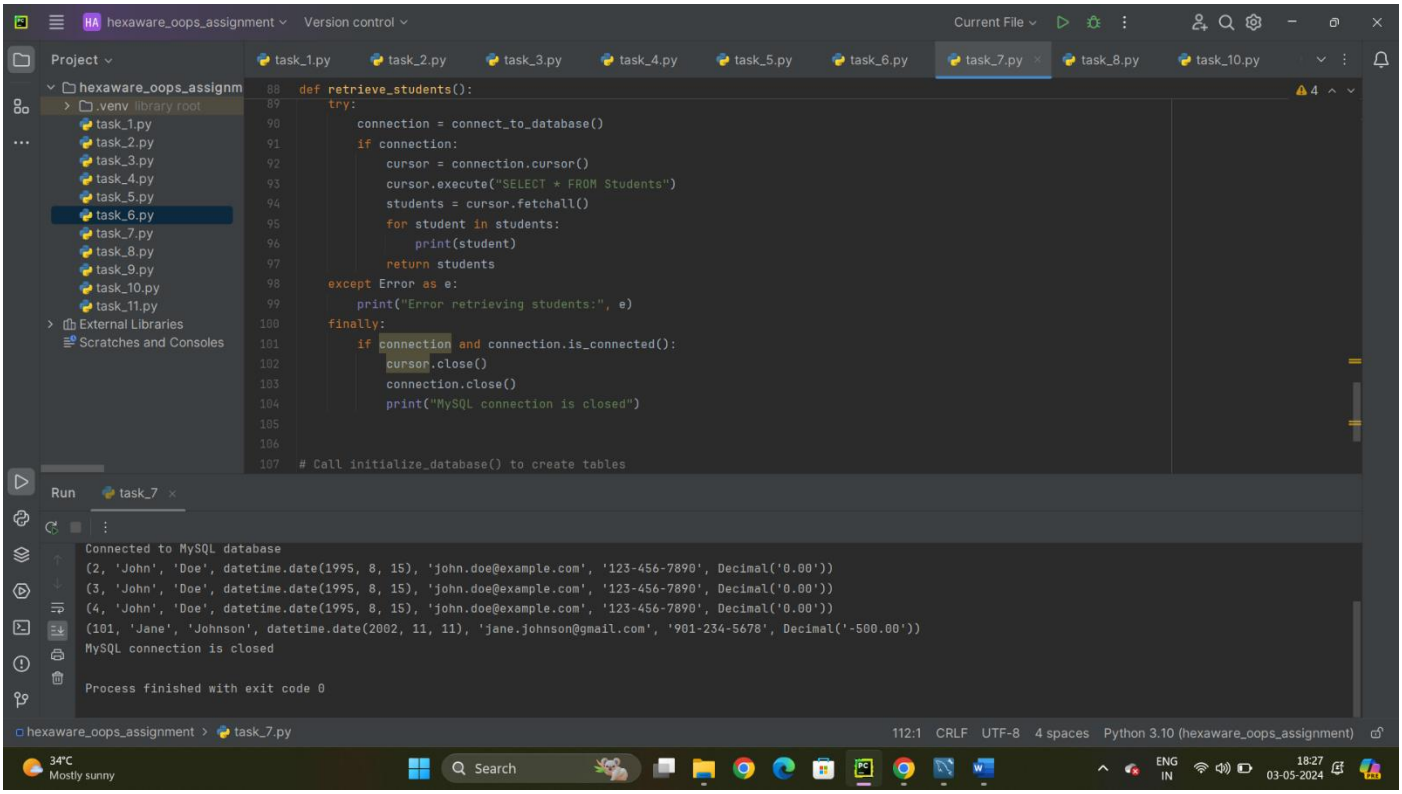
Database Initialization: Implement a method that initializes a database connection and creates tables for storing student, course, enrollment, teacher, and payment information. Create SQL scripts or use code-first migration to create tables with appropriate schemas for your SIS. Data Retrieval: Implement methods to retrieve data from the

database. Users should be able to request information about students, courses, enrollments, teachers, or payments. Ensure that the data retrieval methods handle exceptions and edge cases gracefully. Data Insertion and Updating: Implement methods to insert new data (e.g., enrollments, payments) into the database and update existing data (e.g., student information). Use methods to perform data insertion and updating. Implement validation checks to ensure data integrity and handle any errors during these operations. Transaction Management: Implement methods for handling database transactions when enrolling students, assigning teachers, or recording payments. Transactions should be atomic and maintain data integrity. Use database transactions to ensure that multiple related operations either all succeed or all fail. Implement error handling and rollback mechanisms in case of transaction failures. Dynamic Query Builder: Implement a dynamic query builder that allows users to construct and execute custom SQL queries to retrieve specific data from the database. Users should be able to specify columns, conditions, and sorting criteria. Create a query builder method that dynamically generates SQL queries based on user input. Implement parameterization and sanitation of user inputs to prevent SQL injection.

CODE:

```
import mysql.connector
from mysql.connector import Error


def connect_to_database():
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='root',
            database='SISDB'
        )
        if connection.is_connected():
            print("Connected to MySQL database")
            return connection
    except Error as e:
        print("Error connecting to MySQL database:", e)
        return None


def initialize_database():
    try:
        connection = connect_to_database()
        if connection:
            cursor = connection.cursor()

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS Students (
                    student_id INT AUTO_INCREMENT PRIMARY KEY,
                    first_name VARCHAR(50),
                    last_name VARCHAR(50),
                    date_of_birth DATE,
                    email VARCHAR(100),
                    phone_number VARCHAR(20)
                )
            """)

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS Courses (
                    course_id INT AUTO_INCREMENT PRIMARY KEY,
                    course_name VARCHAR(100),
                    credits INT,
                    teacher_id INT,
                    FOREIGN KEY (teacher_id) REFERENCES Teacher(teacher_id)
                )
            """)

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS Enrollments (
```

```
                    enrollment_id INT AUTO_INCREMENT PRIMARY KEY,
                    student_id INT,
                    course_id INT,
                    enrollment_date DATE,
                    FOREIGN KEY (student_id) REFERENCES Students(student_id),
                    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
                )
            """)

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS Teacher (
                    teacher_id INT AUTO_INCREMENT PRIMARY KEY,
                    first_name VARCHAR(50),
                    last_name VARCHAR(50),
                    email VARCHAR(100)
                )
            """)

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS Payments (
                    payment_id INT AUTO_INCREMENT PRIMARY KEY,
                    student_id INT,
                    amount DECIMAL(10, 2),
                    payment_date DATE,
                    FOREIGN KEY (student_id) REFERENCES Students(student_id)
                )
            """)
            print("Database initialized successfully!")
            connection.commit()
    except Error as e:
        print("Error initializing database:", e)
    finally:
        if connection and connection.is_connected():
            cursor.close()
            connection.close()
            print("MySQL connection is closed")


def retrieve_students():
    try:
        connection = connect_to_database()
        if connection:
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM Students")
            students = cursor.fetchall()
            for student in students:
                print(student)
            return students
    except Error as e:
        print("Error retrieving students:", e)
    finally:
        if connection and connection.is_connected():
            cursor.close()
            connection.close()
            print("MySQL connection is closed")



initialize_database()
retrieve_students()
```

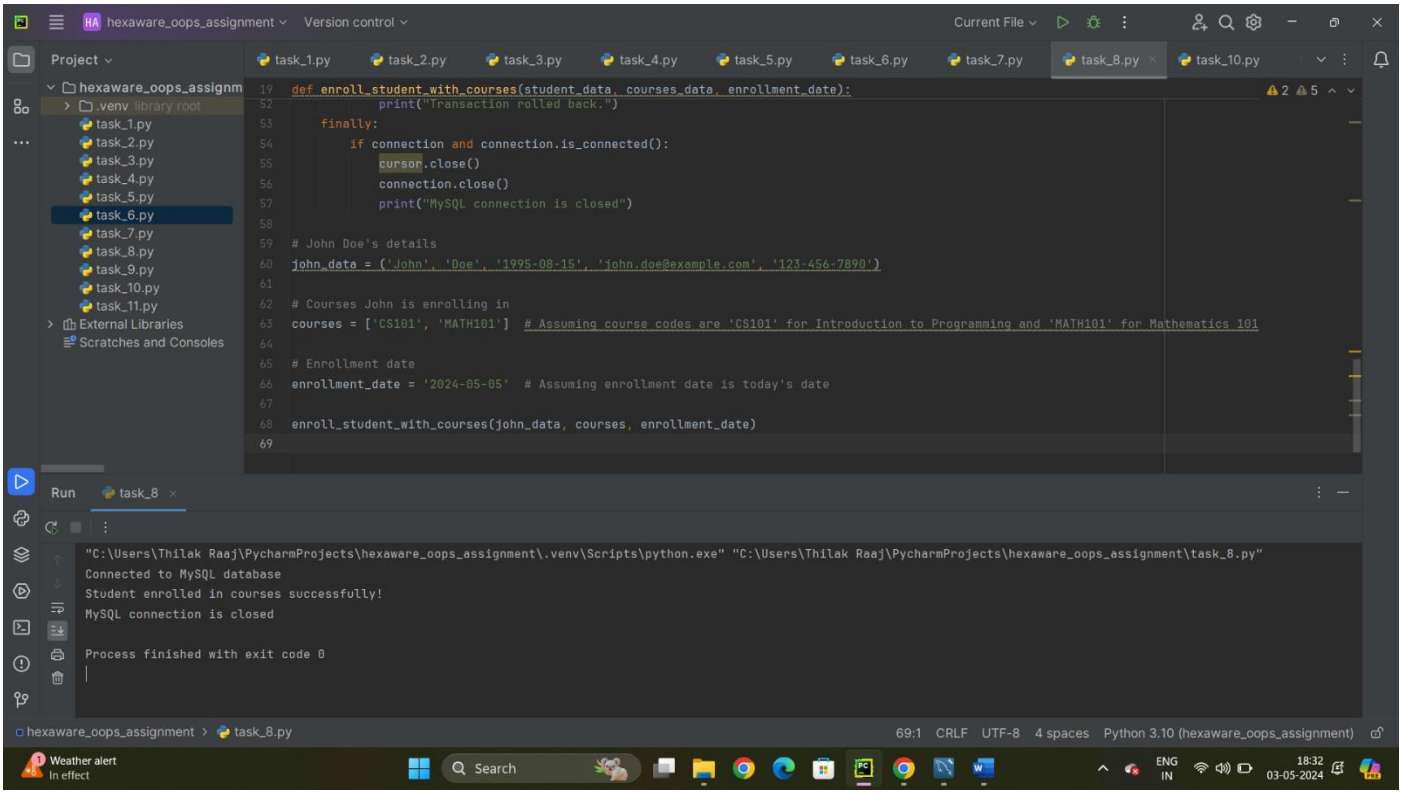


Task 8: Student Enrollment In this task, a new student, John Doe, is enrolling in the SIS. The system needs to record John's information, including his personal details, and enroll him in a few courses. Database connectivity is required to store this information. John Doe's details: • First Name: John • Last Name: Doe • Date of Birth: 1995-08-15 • Email: john.doe@example.com • Phone Number: 123-456-7890 John is enrolling in the following courses: • Course 1: Introduction to Programming • Course 2: Mathematics 101 The system should perform the following tasks: • Create a new student record in the database. • Enroll John in the specified courses by creating enrollment records in the database.

CODE:

```
import mysql.connector
from mysql.connector import Error

def connect_to_database():
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='root',
            database='SISDB'
        )
        if connection.is_connected():
            print("Connected to MySQL database")
            return connection
    except Error as e:
        print("Error connecting to MySQL database:", e)
        return None

def enroll_student_with_courses(student_data, courses_data, enrollment_date):
    try:
        connection = connect_to_database()
        if connection:
            cursor = connection.cursor()


            connection.start_transaction()
```

```
            # Insert student
            cursor.execute("""
                INSERT INTO Students (first_name, last_name, date_of_birth, email,
phone_number)
                VALUES (%s, %s, %s, %s, %s)
            """, student_data)

            student_id = cursor.lastrowid


            for course_code in courses_data:
                cursor.execute("""
                    INSERT INTO Enrollments (student_id, course_code, enrollment_date)
                    VALUES (%s, %s, %s)
                """, (student_id, course_code, enrollment_date))


            connection.commit()
            print("Student enrolled in courses successfully!")
    except Error as e:

        print("Error enrolling student:", e)
        if connection and connection.is_connected():
            connection.rollback()
            print("Transaction rolled back.")
    finally:
        if connection and connection.is_connected():
            cursor.close()
            connection.close()
            print("MySQL connection is closed")

john_data = ('John', 'Doe', '1995-08-15', 'john.doe@example.com', '123-456-7890')

courses = ['CS101', 'MATH101']

enrollment_date = '2024-05-05'

enroll_student_with_courses(john_data, courses, enrollment_date)
```

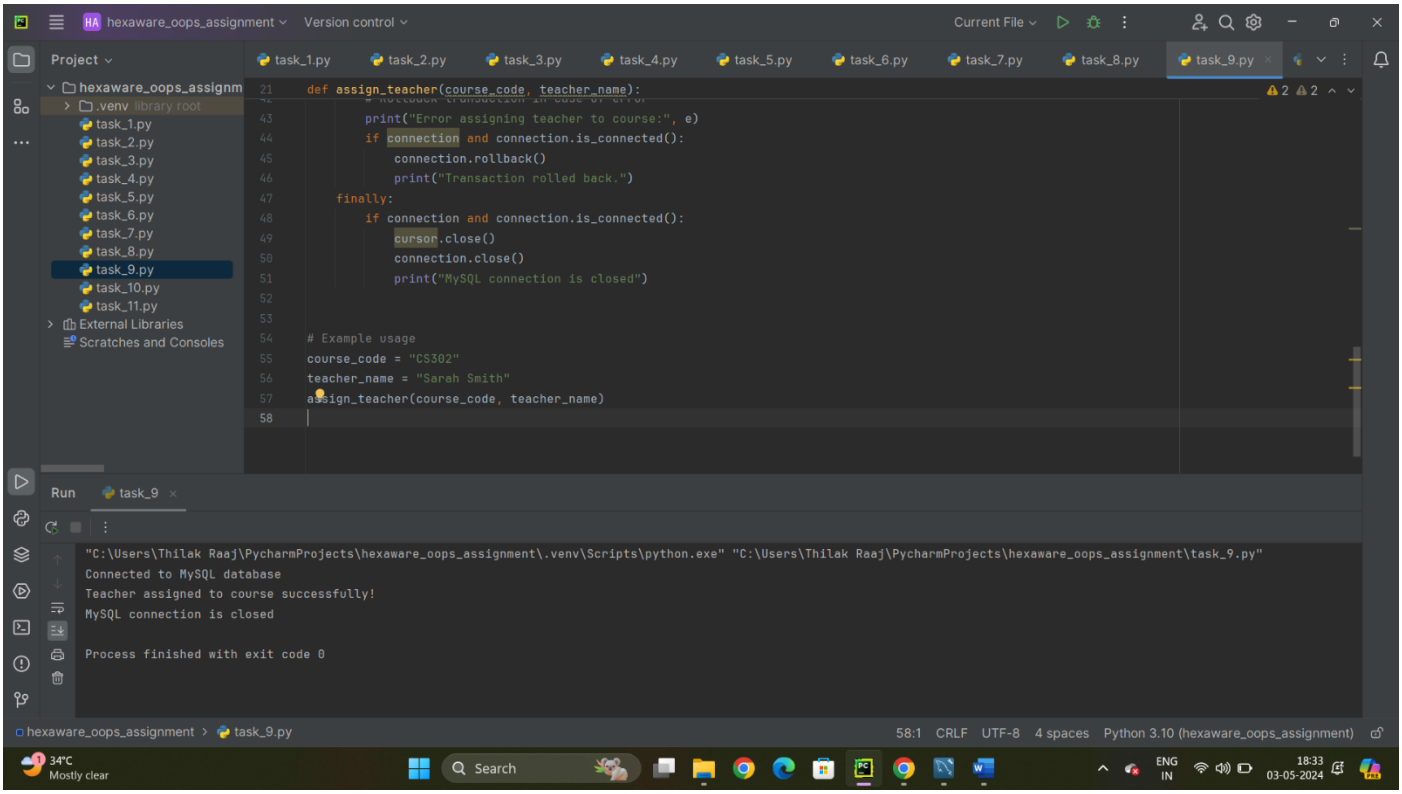


Task 9: Teacher Assignment

In this task, a new teacher, Sarah Smith, is assigned to teach a course. The system needs to update the course record to reflect the teacher assignment. Teacher's Details: • Name: Sarah Smith • Email: sarah.smith@example.com • Expertise: Computer Science Course to be assigned: • Course Name: Advanced Database Management • Course Code: CS302 The system should perform the following tasks: • Retrieve the course record from the database based on the course code. • Assign Sarah Smith as the instructor for the course. • Update the course record in the database with the new instructor information.

CODE:

```
import mysql.connector
from mysql.connector import Error


def connect_to_database():
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='root',
            database='SISDB'
        )
        if connection.is_connected():
            print("Connected to MySQL database")
            return connection
    except Error as e:
        print("Error connecting to MySQL database:", e)
        return None


def assign_teacher(course_code, teacher_name):
    try:
        connection = connect_to_database()
        if connection:
            cursor = connection.cursor()


            connection.start_transaction()
            cursor.execute("""
                UPDATE Courses
                SET instructor_name = %s
                WHERE course_code = %s
            """, (teacher_name, course_code))


            connection.commit()
            print("Teacher assigned to course successfully!")

    except Error as e:

        print("Error assigning teacher to course:", e)
        if connection and connection.is_connected():
            connection.rollback()
            print("Transaction rolled back.")
    finally:
        if connection and connection.is_connected():
            cursor.close()
            connection.close()
            print("MySQL connection is closed")



course_code = "CS302"
teacher_name = "Sarah Smith"
assign_teacher(course_code, teacher_name)
```

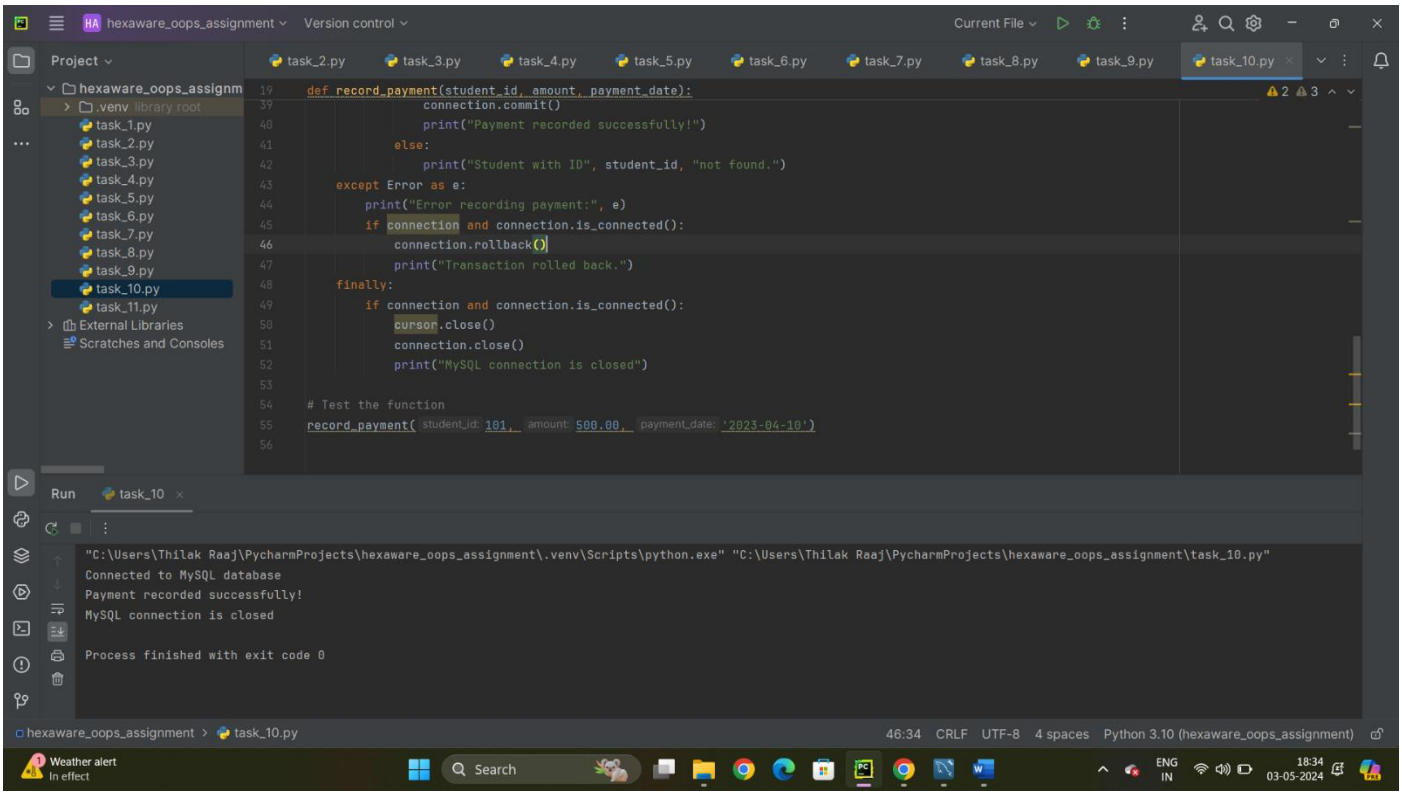


Task 10: Payment Record In this task, a student, Jane Johnson, makes a payment for her enrolled courses. The system needs to record this payment in the database. Jane Johnson's details: • Student ID: 101 • Payment Amount: $500.00 • Payment Date: 2023-04-10 The system should perform the following tasks: • Retrieve Jane Johnson's student record from the database based on her student ID. • Record the payment information in the database, associating it with Jane's student record. • Update Jane's outstanding balance in the database based on the payment amount.

CODE:

```
import mysql.connector
from mysql.connector import Error

def connect_to_database():
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='root',
            database='SISDB'
        )
        if connection.is_connected():
            print("Connected to MySQL database")
            return connection
    except Error as e:
        print("Error connecting to MySQL database:", e)
        return None

def record_payment(student_id, amount, payment_date):
    try:
        connection = connect_to_database()
        if connection:
            cursor = connection.cursor()
            # Check if the student exists
            cursor.execute("SELECT * FROM students WHERE student_id = %s",
(student_id,))
            student = cursor.fetchone()
            if student:
                # Student exists, record the payment
                cursor.execute("""
                    INSERT INTO payments (student_id, amount, payment_date)
                    VALUES (%s, %s, %s)
```

```
                """, (student_id, amount, payment_date))
                cursor.execute("""
                    UPDATE students
                    SET outstanding_balance = outstanding_balance - %s
                    WHERE student_id = %s
                """, (amount, student_id))
                connection.commit()
                print("Payment recorded successfully!")
            else:
                print("Student with ID", student_id, "not found.")
    except Error as e:
        print("Error recording payment:", e)
        if connection and connection.is_connected():
            connection.rollback()
            print("Transaction rolled back.")
    finally:
        if connection and connection.is_connected():
            cursor.close()
            connection.close()
            print("MySQL connection is closed")


record_payment(101, 500.00, '2023-04-10')
```

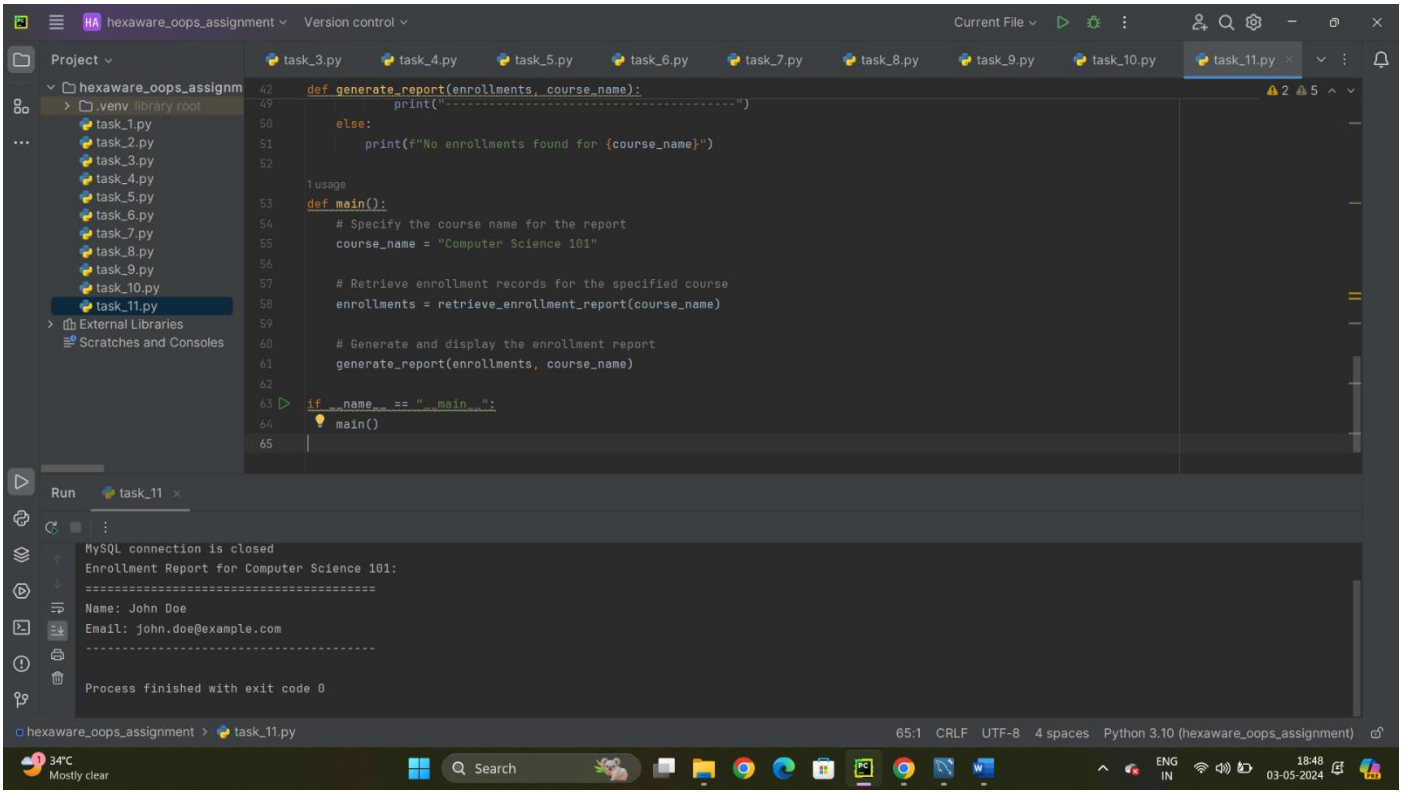


Task 11: Enrollment Report Generation In this task, an administrator requests an enrollment report for a specific course, "Computer Science 101." The system needs to retrieve enrollment information from the database and generate a report. Course to generate the report for: • Course Name: Computer Science 101 The system should perform the following tasks: • Retrieve enrollment records from the database for the specified course. • Generate an enrollment report listing all students enrolled in Computer Science 101. • Display or save the report for the administrator.

CODE:

```
import mysql.connector
from mysql.connector import Error
```

```
def connect_to_database():
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='root',
            database='SISDB'
        )
        if connection.is_connected():
            print("Connected to MySQL database")
            return connection
    except Error as e:
        print("Error connecting to MySQL database:", e)
        return None

def retrieve_enrollment_report(course_name):
    try:
        connection = connect_to_database()
        if connection:
            cursor = connection.cursor()

            cursor.execute("""
                SELECT Students.first_name, Students.last_name, Students.email
                FROM Students
                INNER JOIN Enrollments ON Students.student_id = Enrollments.student_id
                INNER JOIN Courses ON Enrollments.course_code = Courses.course_code
                WHERE Courses.course_name = %s
            """, (course_name,))
            enrollments = cursor.fetchall()
            return enrollments
    except Error as e:
        print("Error retrieving enrollment report:", e)
    finally:
        if connection and connection.is_connected():
            cursor.close()
            connection.close()
            print("MySQL connection is closed")

def generate_report(enrollments, course_name):
    if enrollments:
        print(f"Enrollment Report for {course_name}:")
        print("=====================================")
        for enrollment in enrollments:
            print(f"Name: {enrollment[0]} {enrollment[1]}")
            print(f"Email: {enrollment[2]}")
            print("-------------------------------------")
    else:
        print(f"No enrollments found for {course_name}")

def main():
    # Specify the course name for the report
    course_name = "Computer Science 101"

    # Retrieve enrollment records for the specified course
    enrollments = retrieve_enrollment_report(course_name)

    # Generate and display the enrollment report
    generate_report(enrollments, course_name)

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


hexaware_oops_assignment
Version control
Current File
Project
task_3.py
task_4.py
task_5.py
task_6.py
task_7.py
task_8.py
task_9.py
task_10.py
task_11.py
hexaware_oops_assignm
.venv library root
task_1.py
task_2.py
task_3.py
task_4.py
task_5.py
task_6.py
task_7.py
task_8.py
task_9.py
task_10.py
task_11.py
External Libraries
Scratches and Consoles
def generate_report(enrollments, course_name):
print("----------")
else:
print(f"No enrollments found for {course_name}")
1 usage
def main():
# Specify the course name for the report
course_name = "Computer Science 101"
# Retrieve enrollment records for the specified course
enrollments = retrieve_enrollment_report(course_name)
# Generate and display the enrollment report
generate_report(enrollments, course_name)
if __name__ == "__main__":
main()
Run
task_11
MySQL connection is closed
Enrollment Report for Computer Science 101:
==========
Name: John Doe
Email: john.doe@example.com

Process finished with exit code 0
hexaware_oops_assignment
task_11.py
65:1
CRLF
UTF-8
4 spaces
Python 3.10 (hexaware_oops_assignment)
34°C
Mostly clear
Search
ENG
IN
18:48
03-05-2024