Implement OOPs

A Student Information System (SIS) manages information about students, courses, student enrollments, teachers, and payments. Each student can enroll in multiple courses, each course can have multiple students, each course is taught by a teacher, and students make payments for their courses. Students have attributes such as name, date of birth, email, and phone number. Courses have attributes such as course name, course code, and instructor name. Enrollments track which students are enrolled in which courses. Teachers have attributes such as names and email. Payments track the amount and date of payments made by students.

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:

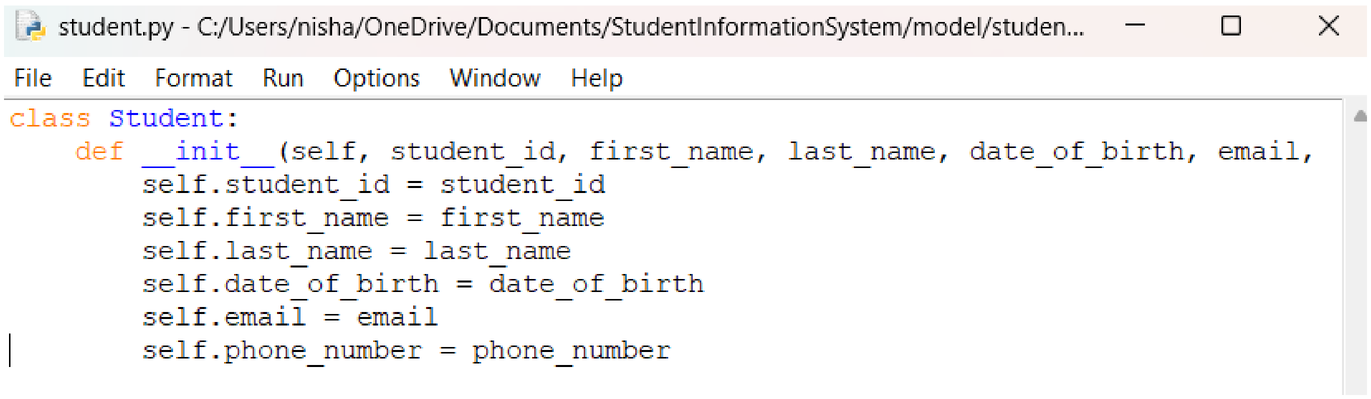
- • Enrollment ID
- • Student ID (reference to a Student)
- • Course ID (reference to a Course)
- • Enrollment Date

Teacher class with the following attributes:

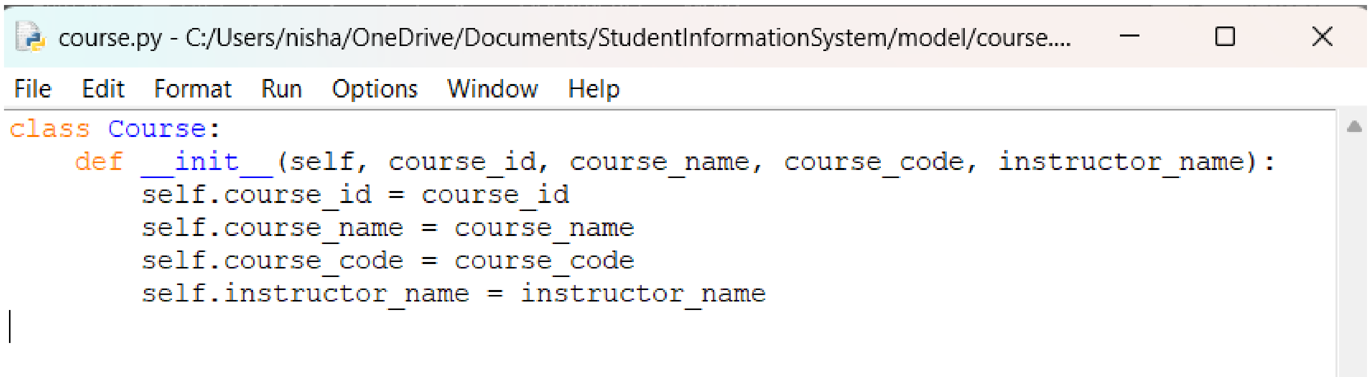
- • Teacher ID
- • First Name
- • Last Name
- • Email

Payment class with the following attributes:

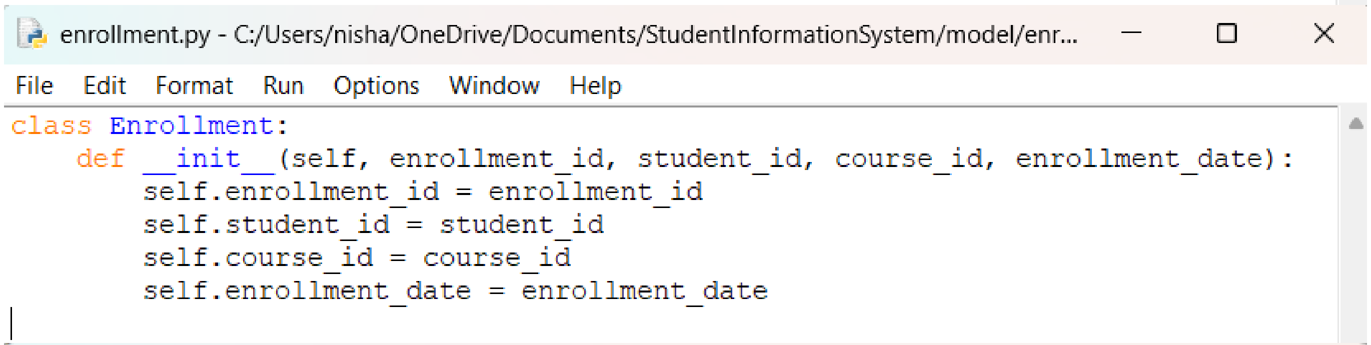
- Payment ID
- Student ID (reference to a Student)
- Amount
- Payment Date


student.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/studen...
File Edit Format Run Options Window Help

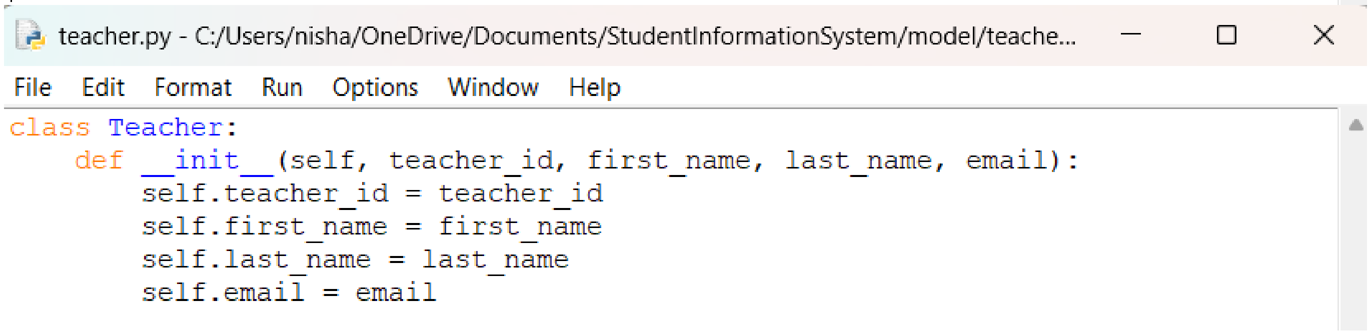
```
class Student:
    def __init__(self, student_id, first_name, last_name, date_of_birth, email,
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
```


course.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/course....

```
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
```

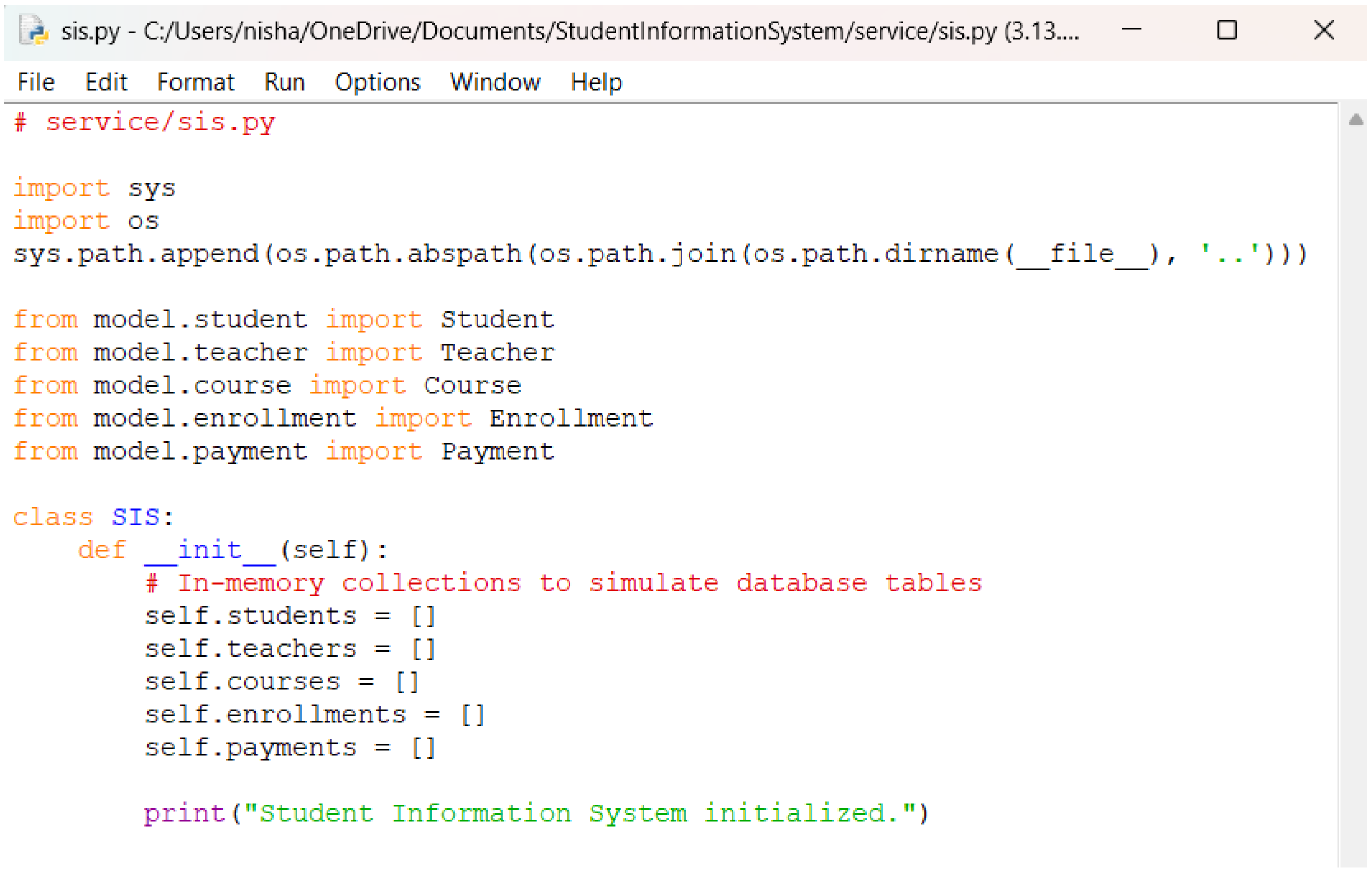

enrollment.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/enr...

```
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
```


teacher.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/teache...

```
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
```

Task2

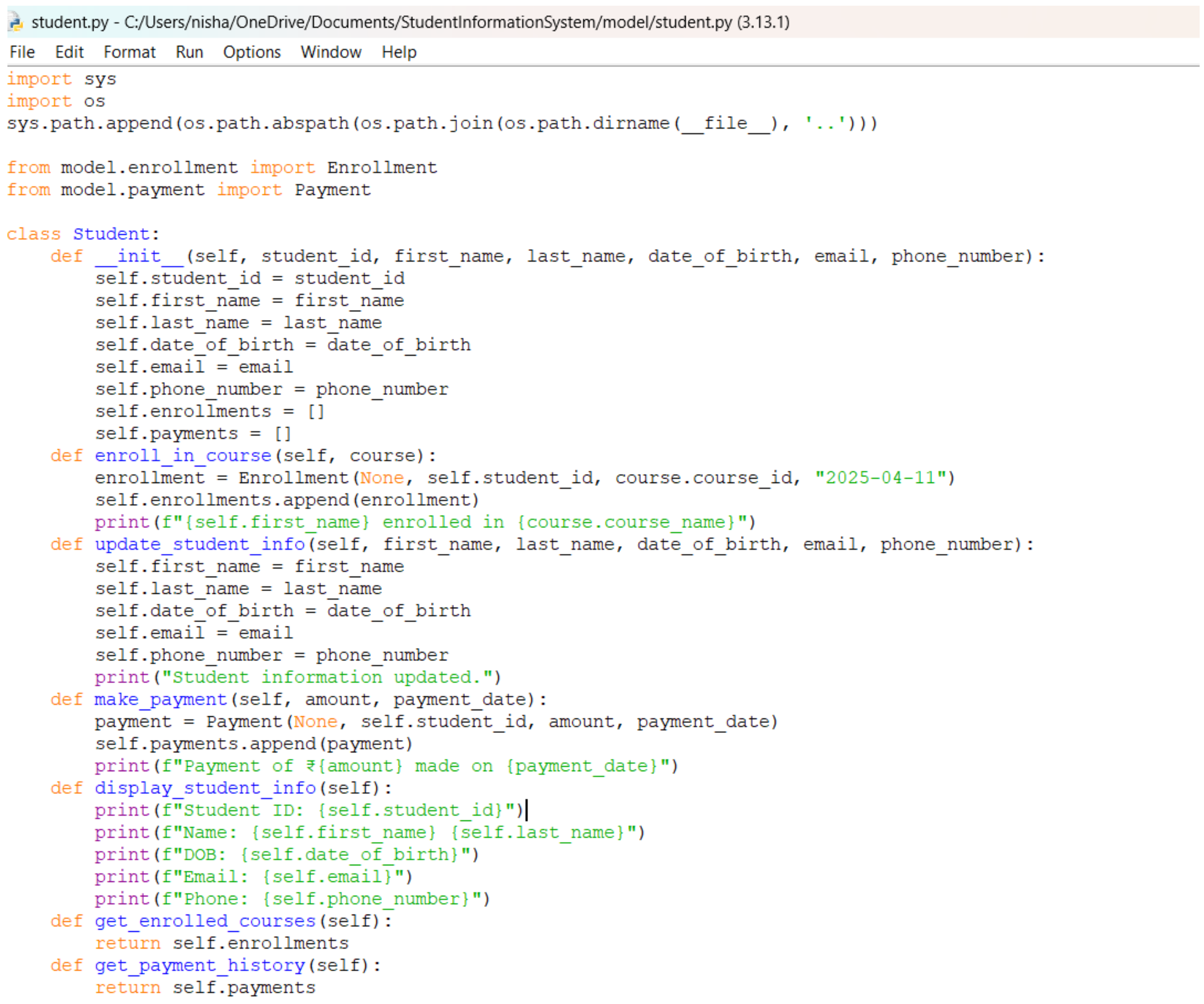

sis.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/service/sis.py (3.13....

```
# service/sis.py

import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from model.student import Student
from model.teacher import Teacher
from model.course import Course
from model.enrollment import Enrollment
from model.payment import Payment

class SIS:
    def __init__(self):
        # In-memory collections to simulate database tables
        self.students = []
        self.teachers = []
        self.courses = []
        self.enrollments = []
        self.payments = []

        print("Student Information System initialized.")
```

Task 3: Implement Methods

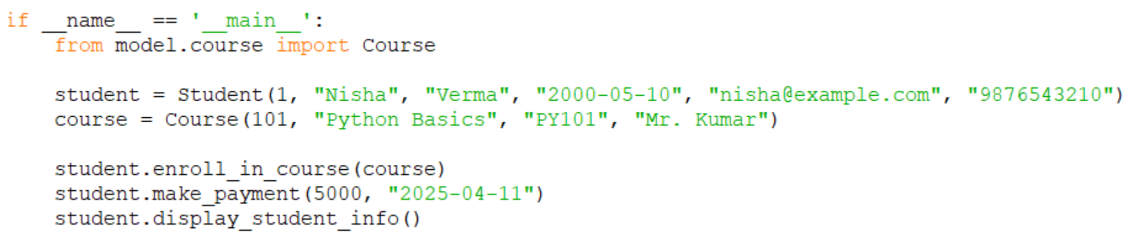
Implement methods in your classes to perform various operations related to the Student Information System (SIS). These methods will allow you to interact with and manipulate data within your system. Below are detailed instructions on how to implement methods in each class:
Implement the following methods 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.

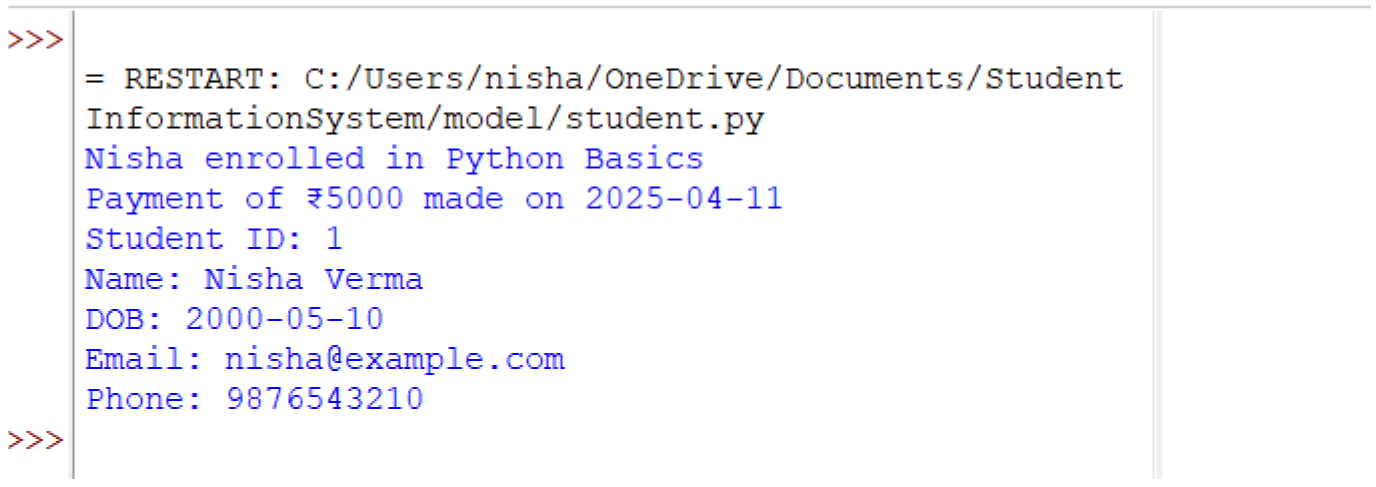

student.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/student.py (3.13.1)
File Edit Format Run Options Window Help

```
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from model.enrollment import Enrollment
from model.payment import Payment

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(None, self.student_id, course.course_id, "2025-04-11")
        self.enrollments.append(enrollment)
        print(f"{self.first_name} enrolled in {course.course_name}")
    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
        print("Student information updated.")
    def make_payment(self, amount, payment_date):
        payment = Payment(None, self.student_id, amount, payment_date)
        self.payments.append(payment)
        print(f"Payment of ₹{amount} made on {payment_date}")
    def display_student_info(self):
        print(f"Student ID: {self.student_id}")
        print(f"Name: {self.first_name} {self.last_name}")
        print(f"DOB: {self.date_of_birth}")
        print(f"Email: {self.email}")
        print(f"Phone: {self.phone_number}")
    def get_enrolled_courses(self):
        return self.enrollments
    def get_payment_history(self):
        return self.payments
```

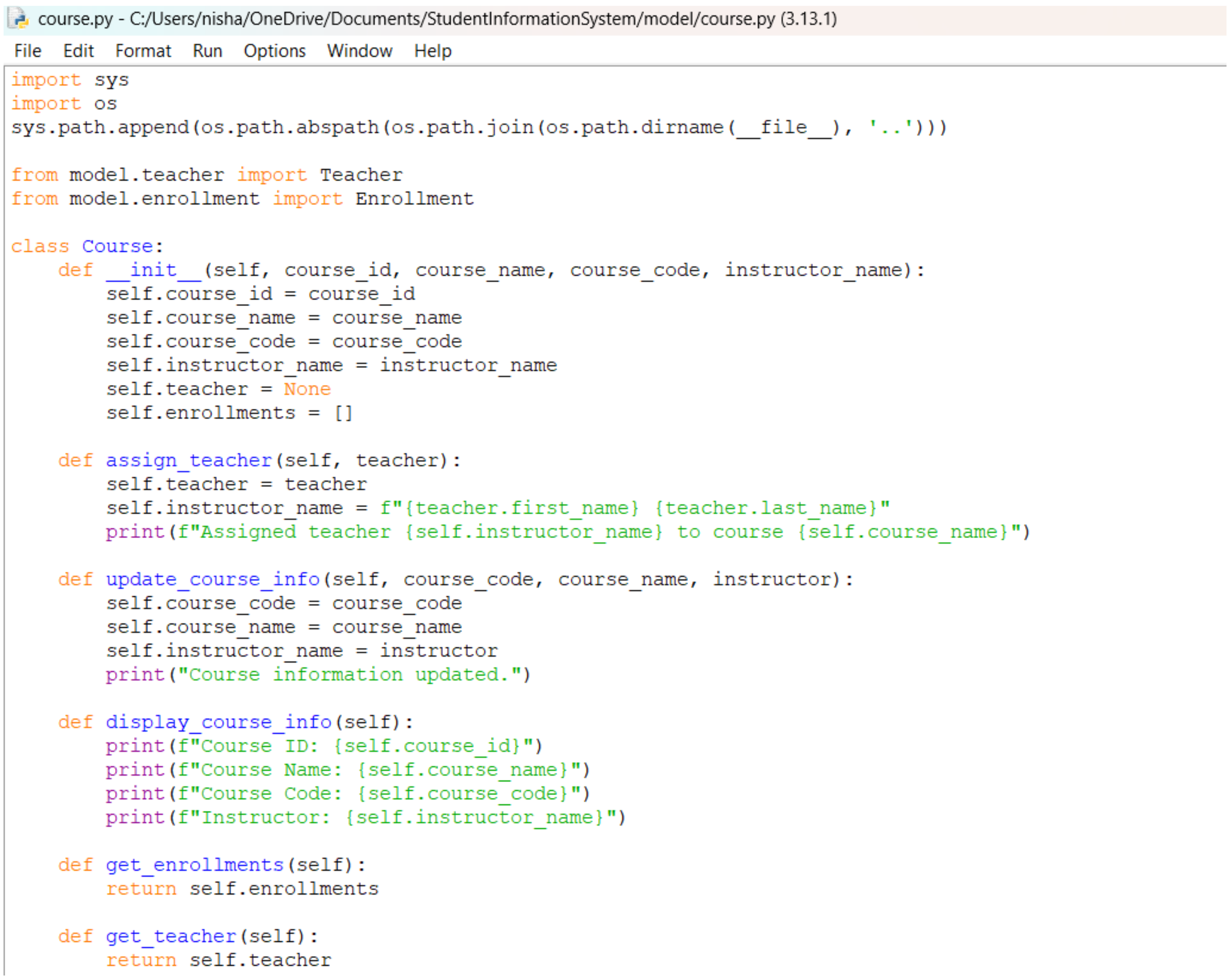
Testing(optional):



```
if __name__ == '__main__':
    from model.course import Course

    student = Student(1, "Nisha", "Verma", "2000-05-10", "nisha@example.com", "9876543210")
    course = Course(101, "Python Basics", "PY101", "Mr. Kumar")

    student.enroll_in_course(course)
    student.make_payment(5000, "2025-04-11")
    student.display_student_info()
```

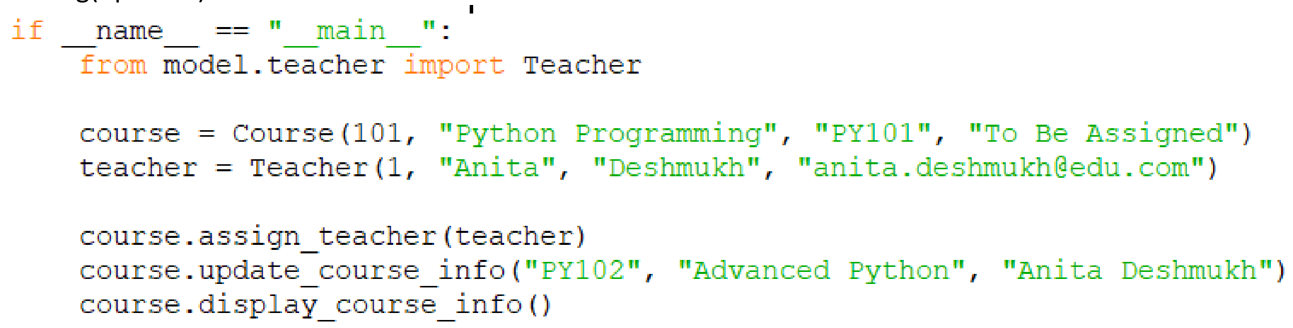


```
>>>
    = RESTART: C:/Users/nisha/OneDrive/Documents/Student
    InformationSystem/model/student.py
    Nisha enrolled in Python Basics
    Payment of ₹5000 made on 2025-04-11
    Student ID: 1
    Name: Nisha Verma
    DOB: 2000-05-10
    Email: nisha@example.com
    Phone: 9876543210
>>>
```

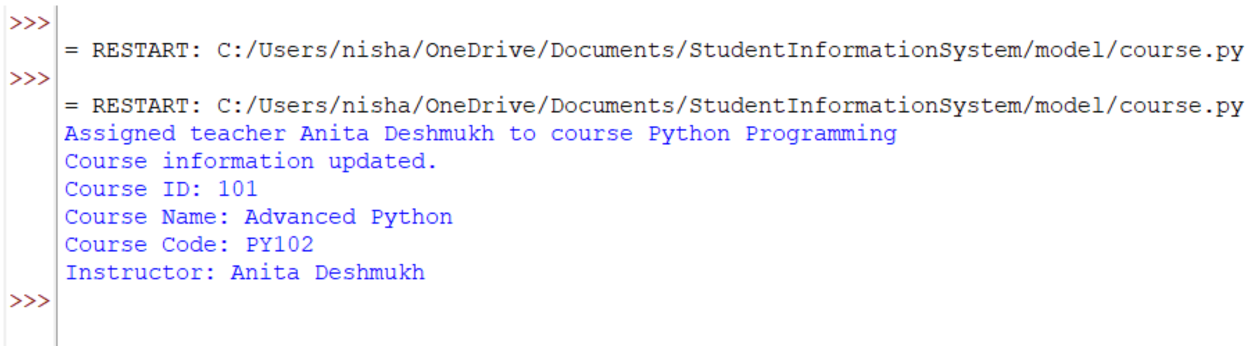

course.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/course.py (3.13.1)

```python
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from model.teacher import Teacher
from model.enrollment import Enrollment

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.teacher = None
        self.enrollments = []

    def assign_teacher(self, teacher):
        self.teacher = teacher
        self.instructor_name = f"{teacher.first_name} {teacher.last_name}"
        print(f"Assigned teacher {self.instructor_name} to course {self.course_name}")

    def update_course_info(self, course_code, course_name, instructor):
        self.course_code = course_code
        self.course_name = course_name
        self.instructor_name = instructor
        print("Course information updated.")

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

    def get_enrollments(self):
        return self.enrollments

    def get_teacher(self):
        return self.teacher
```

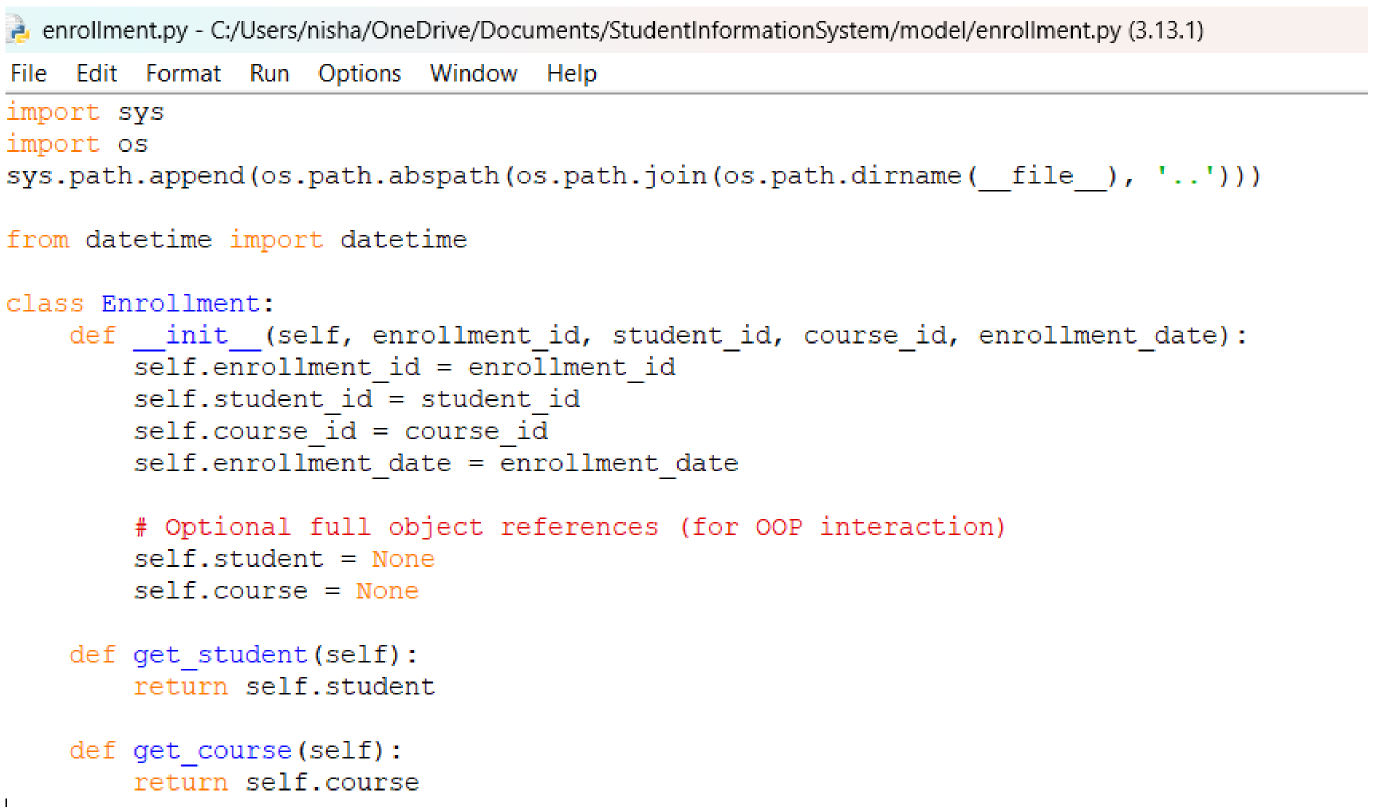
Testing(optional):



```python
if __name__ == "__main__":
    from model.teacher import Teacher

    course = Course(101, "Python Programming", "PY101", "To Be Assigned")
    teacher = Teacher(1, "Anita", "Deshmukh", "anita.deshmukh@edu.com")

    course.assign_teacher(teacher)
    course.update_course_info("PY102", "Advanced Python", "Anita Deshmukh")
    course.display_course_info()
```

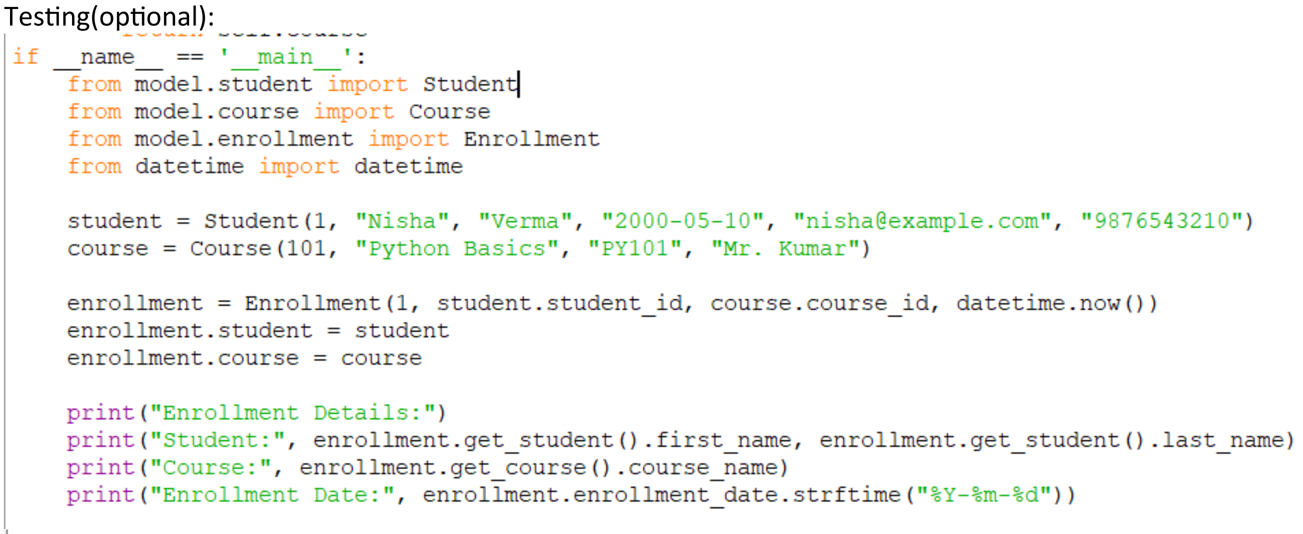


```
>>>
    = RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/course.py
>>>
    = RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/course.py
    Assigned teacher Anita Deshmukh to course Python Programming
    Course information updated.
    Course ID: 101
    Course Name: Advanced Python
    Course Code: PY102
    Instructor: Anita Deshmukh
>>>
```

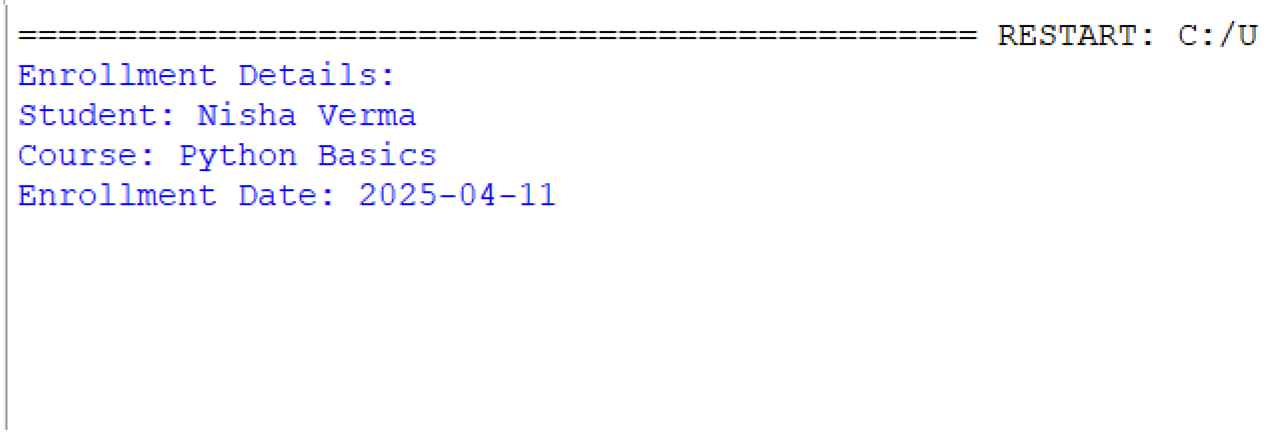

enrollment.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/enrollment.py (3.13.1)
File Edit Format Run Options Window Help

```python
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from datetime import datetime

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

        # Optional full object references (for OOP interaction)
        self.student = None
        self.course = None

    def get_student(self):
        return self.student

    def get_course(self):
        return self.course
```

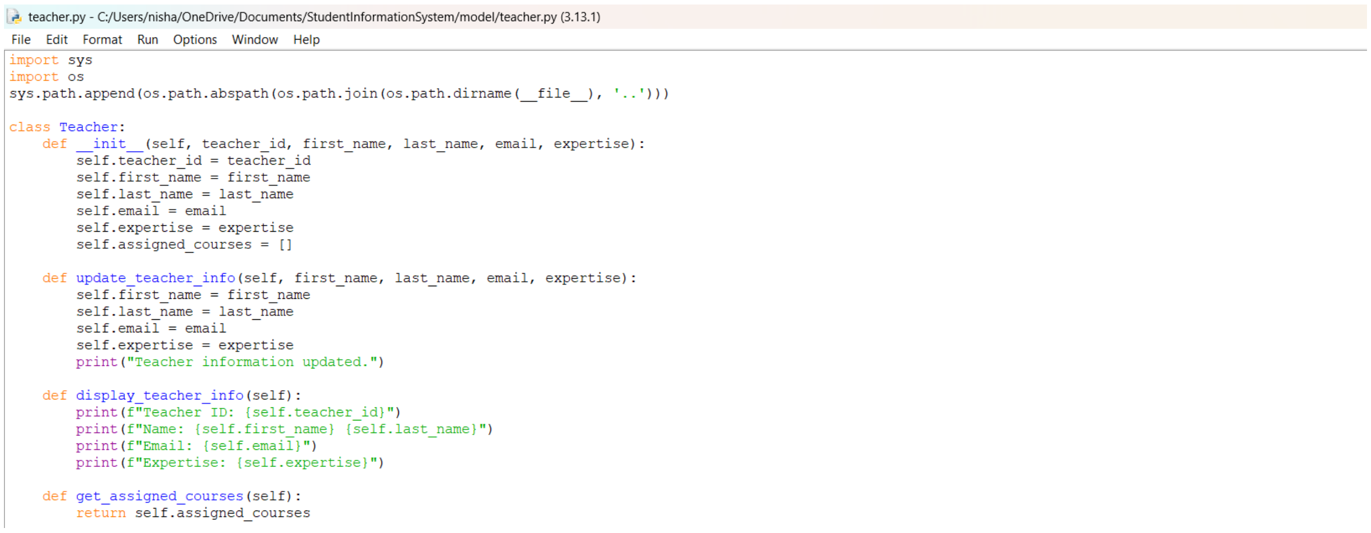
Testing(optional):



```python
if __name__ == '__main__':
    from model.student import Student
    from model.course import Course
    from model.enrollment import Enrollment
    from datetime import datetime

    student = Student(1, "Nisha", "Verma", "2000-05-10", "nisha@example.com", "9876543210")
    course = Course(101, "Python Basics", "PY101", "Mr. Kumar")

    enrollment = Enrollment(1, student.student_id, course.course_id, datetime.now())
    enrollment.student = student
    enrollment.course = course

    print("Enrollment Details:")
    print("Student:", enrollment.get_student().first_name, enrollment.get_student().last_name)
    print("Course:", enrollment.get_course().course_name)
    print("Enrollment Date:", enrollment.enrollment_date.strftime("%Y-%m-%d"))
```

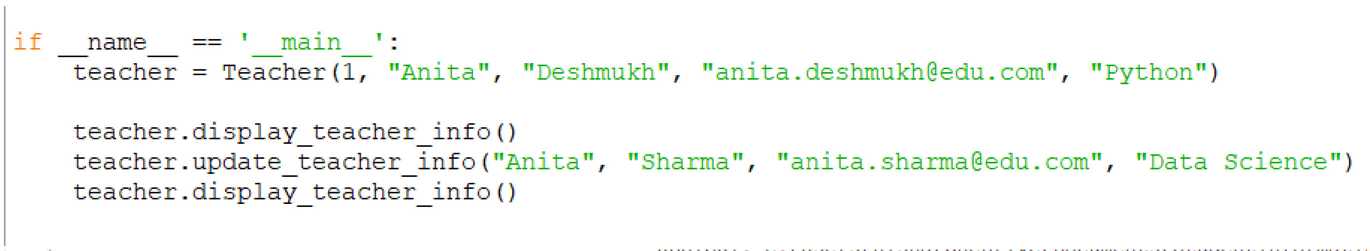


```
================================================ RESTART: C:/U
Enrollment Details:
Student: Nisha Verma
Course: Python Basics
Enrollment Date: 2025-04-11
```

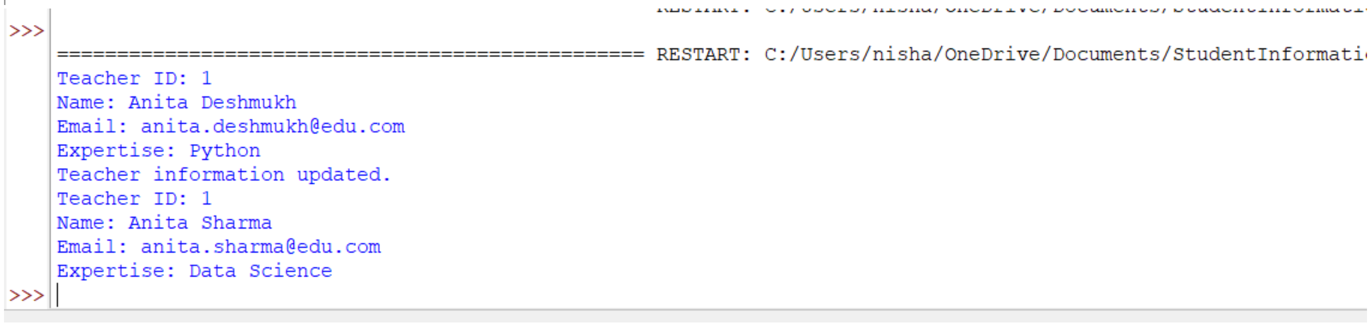

teacher.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/teacher.py (3.13.1)
File Edit Format Run Options Window Help

```
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

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

    def update_teacher_info(self, first_name, last_name, email, expertise):
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.expertise = expertise
        print("Teacher information updated.")

    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}")
        print(f"Expertise: {self.expertise}")

    def get_assigned_courses(self):
        return self.assigned_courses
```

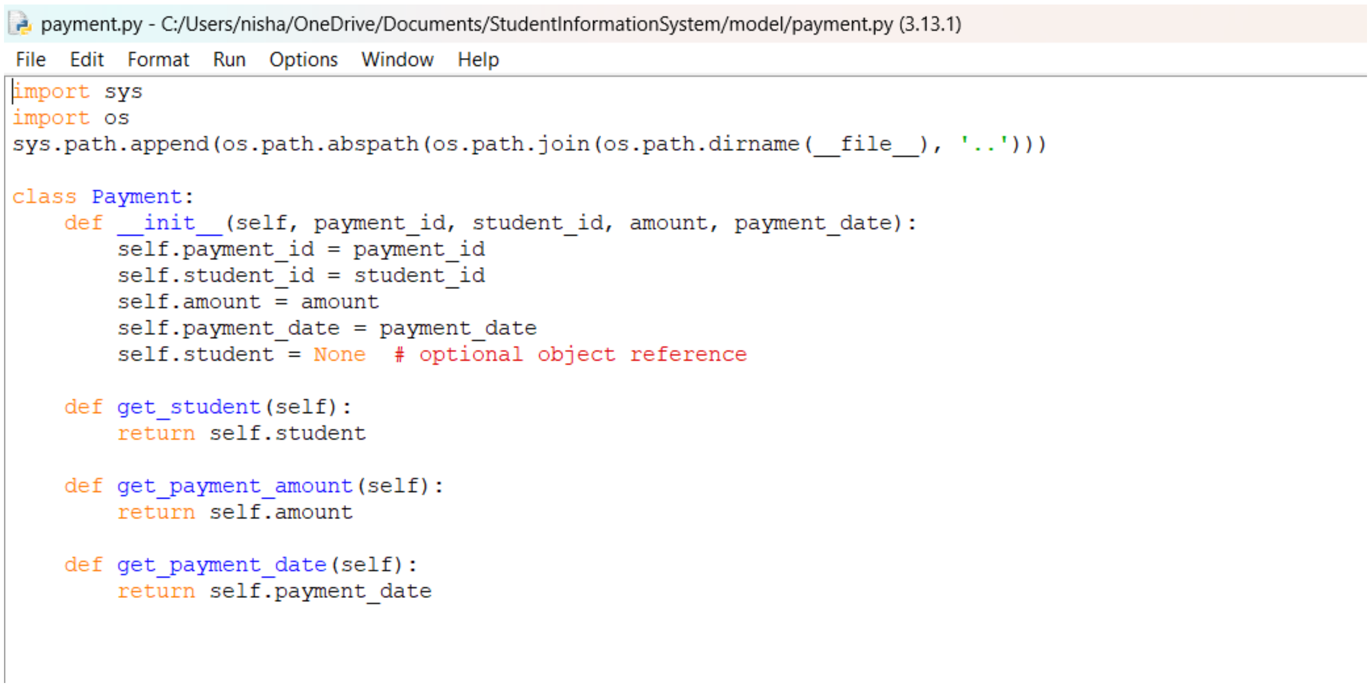
Testing(optional):



```
if __name__ == '__main__':
    teacher = Teacher(1, "Anita", "Deshmukh", "anita.deshmukh@edu.com", "Python")

    teacher.display_teacher_info()
    teacher.update_teacher_info("Anita", "Sharma", "anita.sharma@edu.com", "Data Science")
    teacher.display_teacher_info()
```



```
>>>
    ================================================ RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformati
    Teacher ID: 1
    Name: Anita Deshmukh
    Email: anita.deshmukh@edu.com
    Expertise: Python
    Teacher information updated.
    Teacher ID: 1
    Name: Anita Sharma
    Email: anita.sharma@edu.com
    Expertise: Data Science
>>>
```

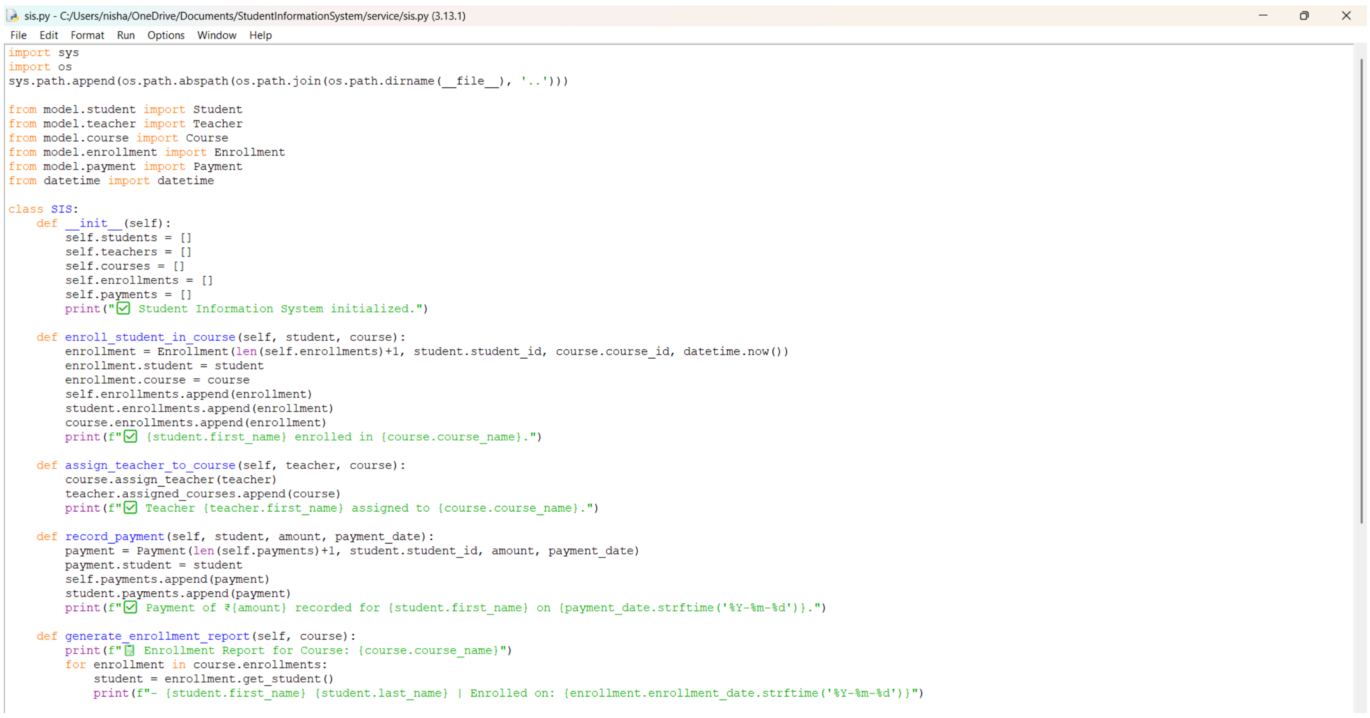

payment.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/payment.py (3.13.1)
File Edit Format Run Options Window Help

```
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

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
        self.student = None  # optional object reference

    def get_student(self):
        return self.student

    def get_payment_amount(self):
        return self.amount

    def get_payment_date(self):
        return self.payment_date
```

Testing(optional):

```python
if __name__ == '__main__':
    from model.student import Student
    from datetime import datetime

    student = Student(1, "Nisha", "Verma", "2000-05-10", "nisha@example.com", "9876543210")
    payment = Payment(1, student.student_id, 5000, datetime.now())
    payment.student = student

    print("Payment Details:")
    print("Student:", payment.get_student().first_name, payment.get_student().last_name)
    print("Amount Paid: ₹", payment.get_payment_amount())
    print("Date:", payment.get_payment_date().strftime("%Y-%m-%d"))
```

```
================================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/model/payment.py =====
Payment Details:
Student: Nisha Verma
Amount Paid: ₹ 5000
Date: 2025-04-11
```

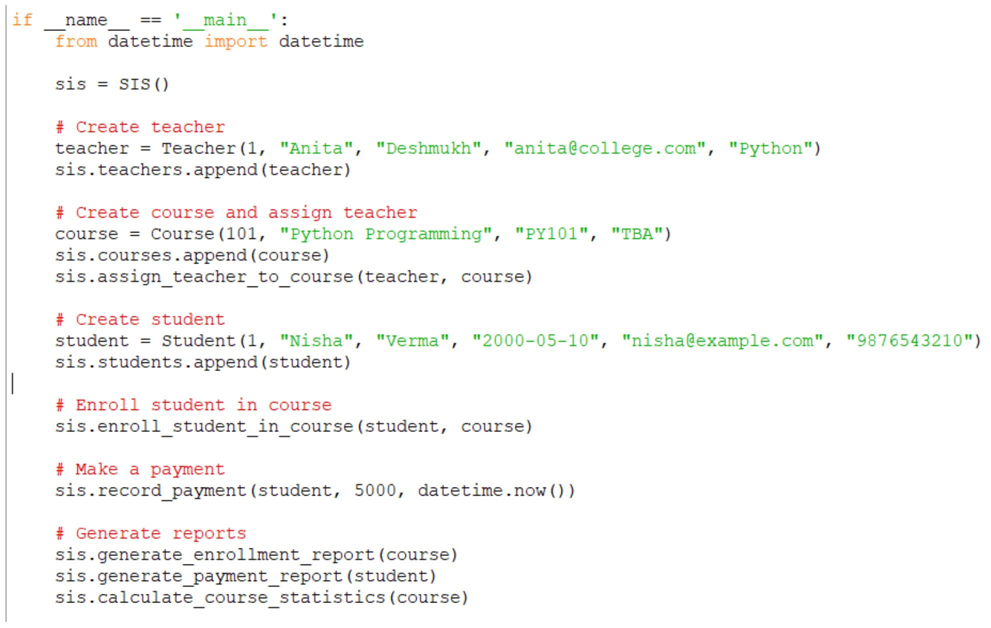

sis.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/service/sis.py (3.13.1)

File Edit Format Run Options Window Help

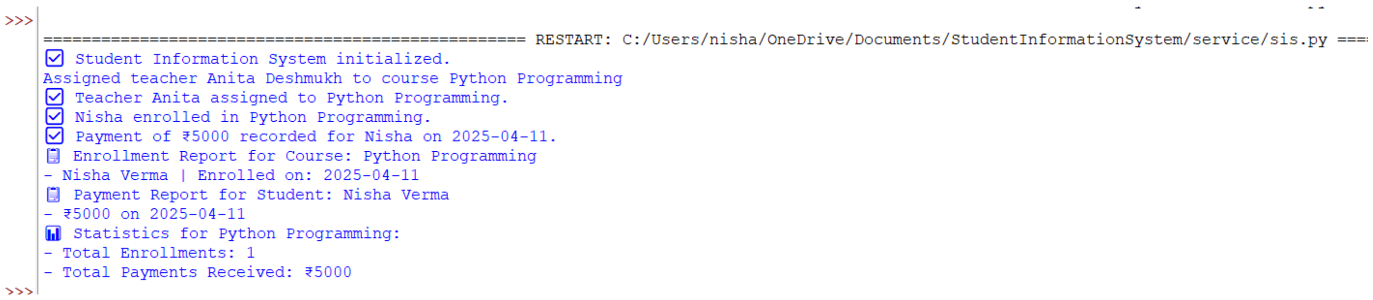
```python
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from model.student import Student
from model.teacher import Teacher
from model.course import Course
from model.enrollment import Enrollment
from model.payment import Payment
from datetime import datetime

class SIS:
    def __init__(self):
        self.students = []
        self.teachers = []
        self.courses = []
        self.enrollments = []
        self.payments = []
        print("✅ Student Information System initialized.")

    def enroll_student_in_course(self, student, course):
        enrollment = Enrollment(len(self.enrollments)+1, student.student_id, course.course_id, datetime.now())
        enrollment.student = student
        enrollment.course = course
        self.enrollments.append(enrollment)
        student.enrollments.append(enrollment)
        course.enrollments.append(enrollment)
        print(f"✅ {student.first_name} enrolled in {course.course_name}.")

    def assign_teacher_to_course(self, teacher, course):
        course.assign_teacher(teacher)
        teacher.assigned_courses.append(course)
        print(f"✅ Teacher {teacher.first_name} assigned to {course.course_name}.")

    def record_payment(self, student, amount, payment_date):
        payment = Payment(len(self.payments)+1, student.student_id, amount, payment_date)
        payment.student = student
        self.payments.append(payment)
        student.payments.append(payment)
        print(f"✅ Payment of ₹{amount} recorded for {student.first_name} on {payment_date.strftime('%Y-%m-%d')}.")

    def generate_enrollment_report(self, course):
        print(f"📋 Enrollment Report for Course: {course.course_name}")
        for enrollment in course.enrollments:
            student = enrollment.get_student()
            print(f"- {student.first_name} {student.last_name} | Enrolled on: {enrollment.enrollment_date.strftime('%Y-%m-%d')}")
```

```python
    def generate_payment_report(self, student):
        print(f"📋 Payment Report for Student: {student.first_name} {student.last_name}")
        for payment in student.payments:
            print(f"- ₹{payment.amount} on {payment.payment_date.strftime('%Y-%m-%d')}")

    def calculate_course_statistics(self, course):
        total_enrollments = len(course.enrollments)
        total_payments = 0

        for enrollment in course.enrollments:
            student = enrollment.get_student()
            for payment in student.payments:
                total_payments += payment.amount

        print(f"📊 Statistics for {course.course_name}:")
        print(f"- Total Enrollments: {total_enrollments}")
        print(f"- Total Payments Received: ₹{total_payments}")
```

Testing(optional):



```python
if __name__ == '__main__':
    from datetime import datetime

    sis = SIS()

    # Create teacher
    teacher = Teacher(1, "Anita", "Deshmukh", "anita@college.com", "Python")
    sis.teachers.append(teacher)

    # Create course and assign teacher
    course = Course(101, "Python Programming", "PY101", "TBA")
    sis.courses.append(course)
    sis.assign_teacher_to_course(teacher, course)

    # Create student
    student = Student(1, "Nisha", "Verma", "2000-05-10", "nisha@example.com", "9876543210")
    sis.students.append(student)

    # Enroll student in course
    sis.enroll_student_in_course(student, course)

    # Make a payment
    sis.record_payment(student, 5000, datetime.now())

    # Generate reports
    sis.generate_enrollment_report(course)
    sis.generate_payment_report(student)
    sis.calculate_course_statistics(course)
```



```
>>>
================================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/service/sis.py ===
☑ Student Information System initialized.
Assigned teacher Anita Deshmukh to course Python Programming
☑ Teacher Anita assigned to Python Programming.
☑ Nisha enrolled in Python Programming.
☑ Payment of ₹5000 recorded for Nisha on 2025-04-11.
📋 Enrollment Report for Course: Python Programming
- Nisha Verma | Enrolled on: 2025-04-11
📋 Payment Report for Student: Nisha Verma
- ₹5000 on 2025-04-11
📊 Statistics for Python Programming:
- Total Enrollments: 1
- Total Payments Received: ₹5000
>>>
```

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.

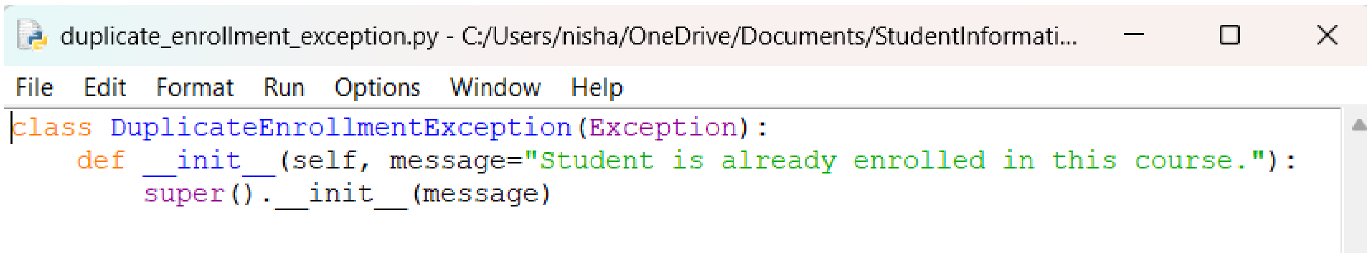
• InvalidStudentDataException: Thrown when data provided for creating or updating a student is invalid (e.g., invalid date of birth or email format).

• InvalidCourseDataException: Thrown when data provided for creating or updating a course is invalid (e.g., invalid course code or instructor name).

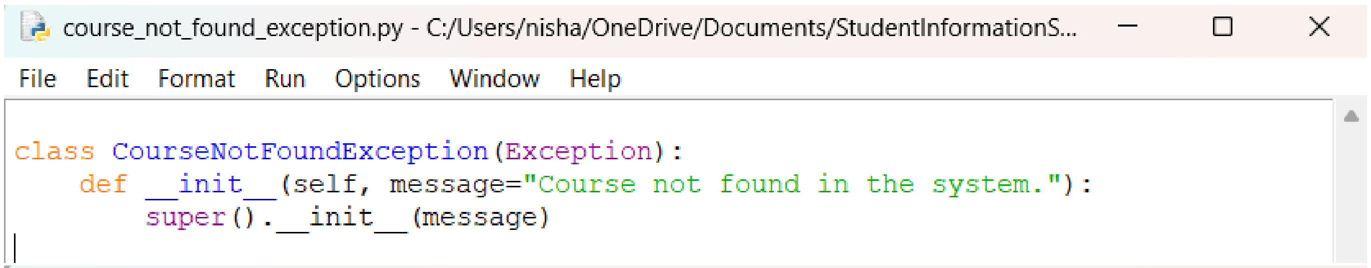
• InvalidEnrollmentDataException: Thrown when data provided for creating an enrollment is invalid (e.g., missing student or course references).

• InvalidTeacherDataException: Thrown when data provided for creating or updating a teacher is invalid (e.g., missing name or email).

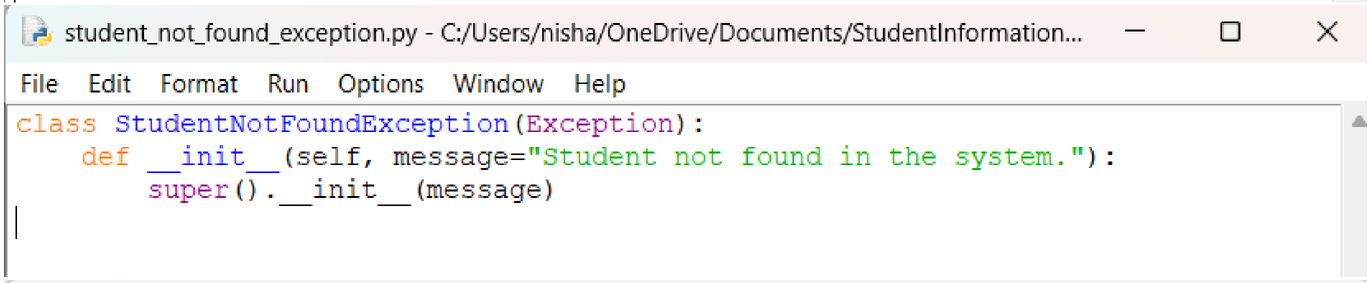
• InsufficientFundsException: Thrown when a student attempts to enroll in a course but does not have enough funds to make the payment.


duplicate_enrollment_exception.py - C:/Users/nisha/OneDrive/Documents/StudentInformati...
File Edit Format Run Options Window Help

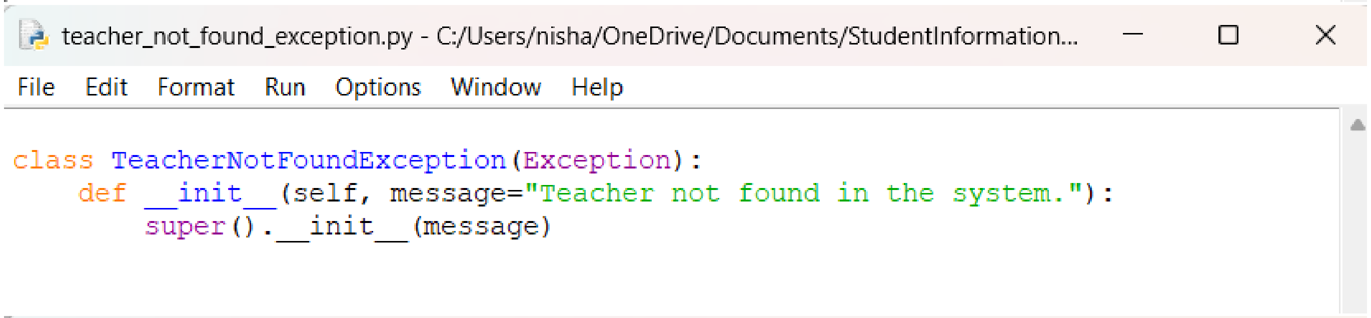
```
class DuplicateEnrollmentException(Exception):
    def __init__(self, message="Student is already enrolled in this course."):
        super().__init__(message)
```


course_not_found_exception.py - C:/Users/nisha/OneDrive/Documents/StudentInformationS...
File Edit Format Run Options Window Help

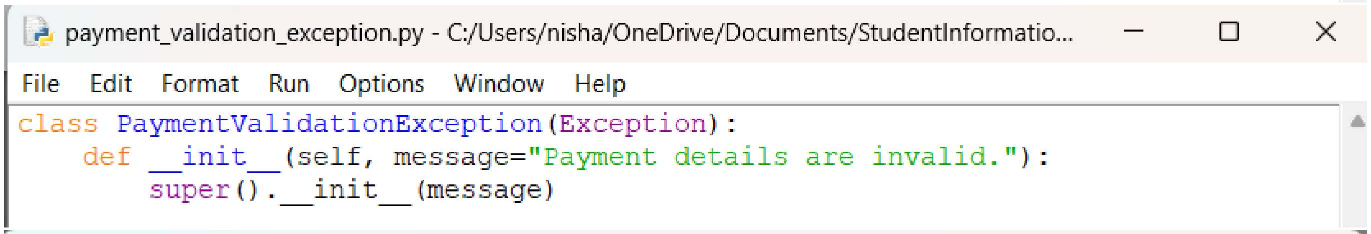
```
class CourseNotFoundException(Exception):
    def __init__(self, message="Course not found in the system."):
        super().__init__(message)
```


student_not_found_exception.py - C:/Users/nisha/OneDrive/Documents/StudentInformation...
File Edit Format Run Options Window Help

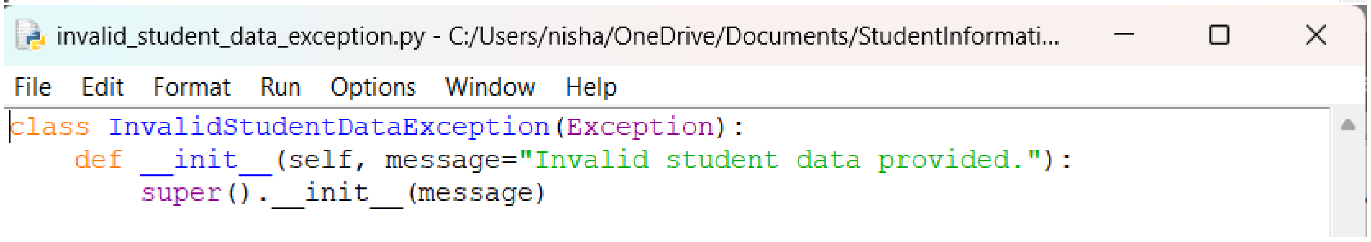
```
class StudentNotFoundException(Exception):
    def __init__(self, message="Student not found in the system."):
        super().__init__(message)
```


teacher_not_found_exception.py - C:/Users/nisha/OneDrive/Documents/StudentInformation...
File Edit Format Run Options Window Help

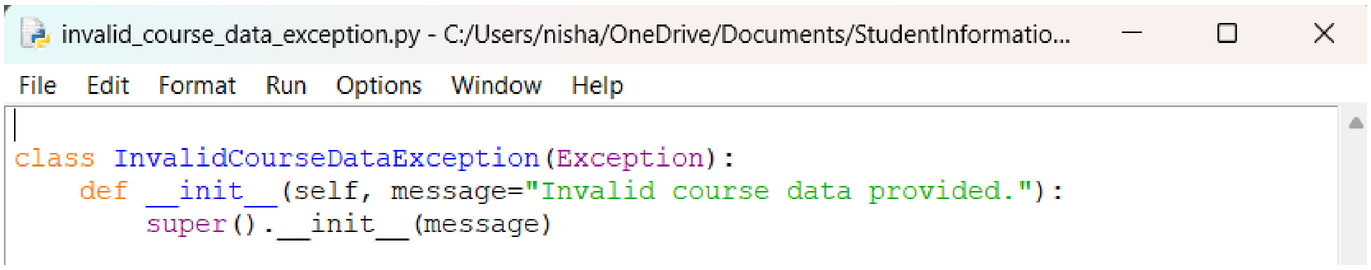
```
class TeacherNotFoundException(Exception):
    def __init__(self, message="Teacher not found in the system."):
        super().__init__(message)
```


payment_validation_exception.py - C:/Users/nisha/OneDrive/Documents/StudentInformatio...
File Edit Format Run Options Window Help

```
class PaymentValidationException(Exception):
    def __init__(self, message="Payment details are invalid."):
        super().__init__(message)
```

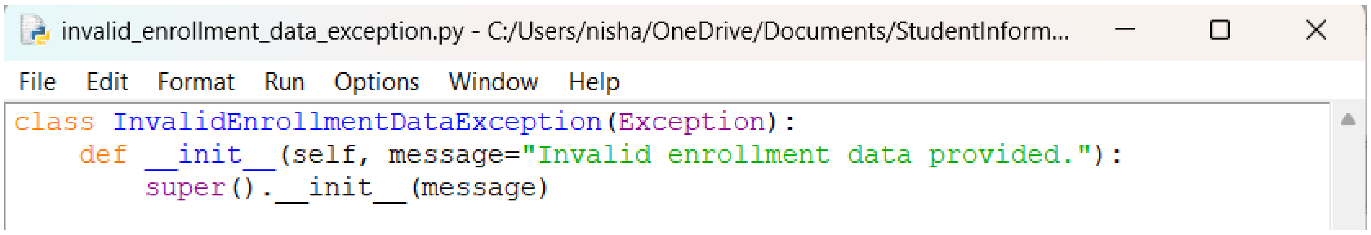

invalid_student_data_exception.py - C:/Users/nisha/OneDrive/Documents/StudentInformati...
File Edit Format Run Options Window Help

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



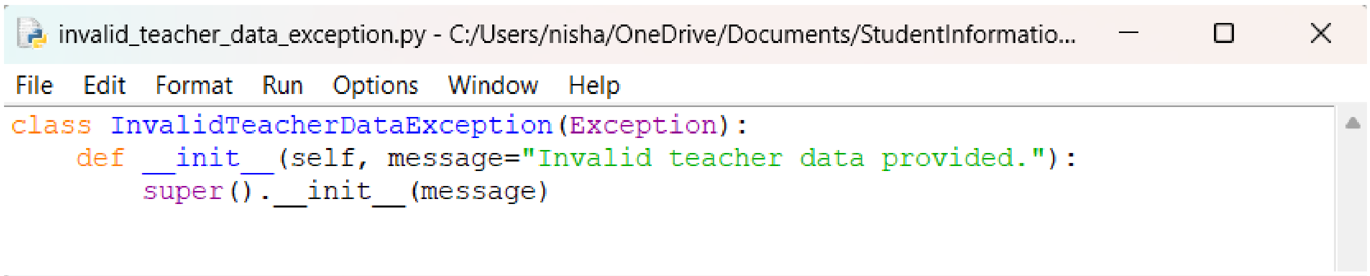
invalid_course_data_exception.py - C:/Users/nisha/OneDrive/Documents/StudentInformatio...

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



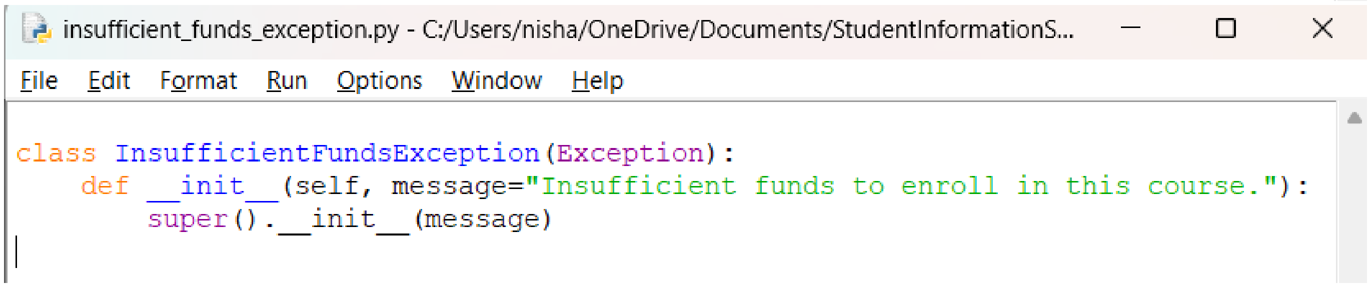
invalid_enrollment_data_exception.py - C:/Users/nisha/OneDrive/Documents/StudentInform...

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



invalid_teacher_data_exception.py - C:/Users/nisha/OneDrive/Documents/StudentInformatio...

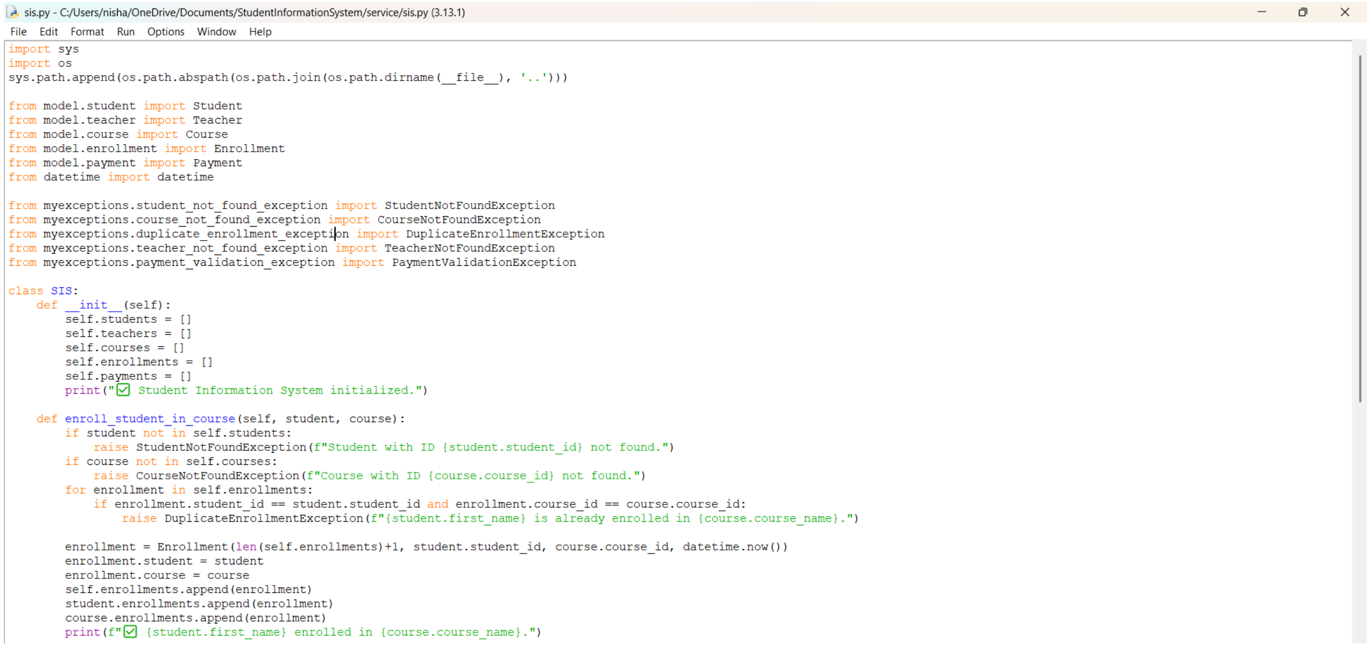
```python
class InvalidTeacherDataException(Exception):
    def __init__(self, message="Invalid teacher data provided."):
        super().__init__(message)
```



insufficient_funds_exception.py - C:/Users/nisha/OneDrive/Documents/StudentInformationS...

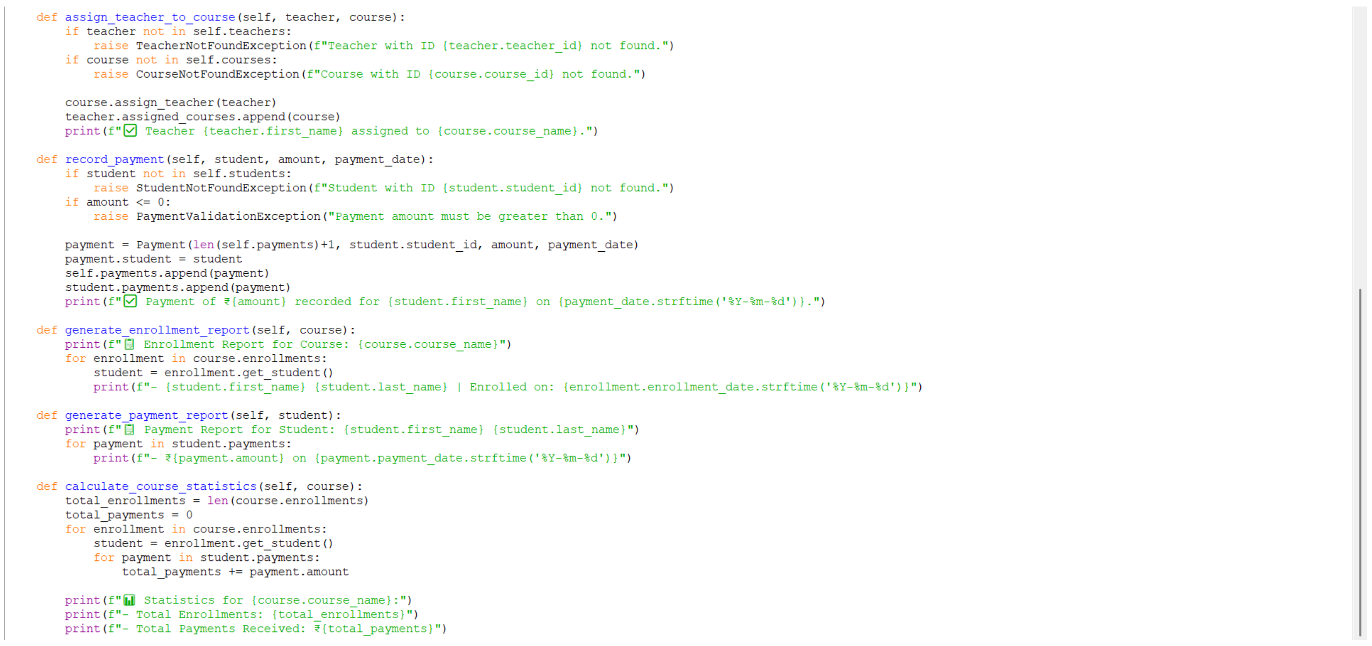
```python
class InsufficientFundsException(Exception):
    def __init__(self, message="Insufficient funds to enroll in this course."):
        super().__init__(message)
```

This is for sis.py updated→to show that exceptions are involved



sis.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/service/sis.py (3.13.1)

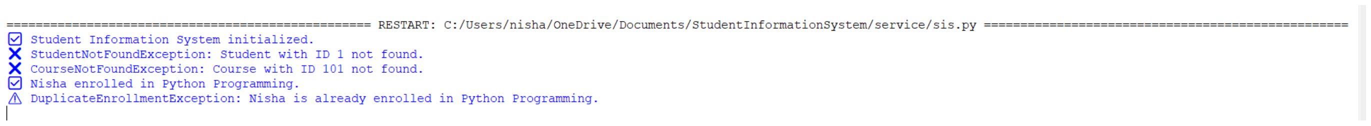
```python
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from model.student import Student
from model.teacher import Teacher
from model.course import Course
from model.enrollment import Enrollment
from model.payment import Payment
from datetime import datetime

from myexceptions.student_not_found_exception import StudentNotFoundException
from myexceptions.course_not_found_exception import CourseNotFoundException
from myexceptions.duplicate_enrollment_exception import DuplicateEnrollmentException
from myexceptions.teacher_not_found_exception import TeacherNotFoundException
from myexceptions.payment_validation_exception import PaymentValidationException

class SIS:
    def __init__(self):
        self.students = []
        self.teachers = []
        self.courses = []
        self.enrollments = []
        self.payments = []
        print("✅ Student Information System initialized.")

    def enroll_student_in_course(self, student, course):
        if student not in self.students:
            raise StudentNotFoundException(f"Student with ID {student.student_id} not found.")
        if course not in self.courses:
            raise CourseNotFoundException(f"Course with ID {course.course_id} not found.")
        for enrollment in self.enrollments:
            if enrollment.student_id == student.student_id and enrollment.course_id == course.course_id:
                raise DuplicateEnrollmentException(f"{student.first_name} is already enrolled in {course.course_name}.")

        enrollment = Enrollment(len(self.enrollments)+1, student.student_id, course.course_id, datetime.now())
        enrollment.student = student
        enrollment.course = course
        self.enrollments.append(enrollment)
        student.enrollments.append(enrollment)
        course.enrollments.append(enrollment)
        print(f"✅ {student.first_name} enrolled in {course.course_name}.")
```



```
    def assign_teacher_to_course(self, teacher, course):
        if teacher not in self.teachers:
            raise TeacherNotFoundException(f"Teacher with ID {teacher.teacher_id} not found.")
        if course not in self.courses:
            raise CourseNotFoundException(f"Course with ID {course.course_id} not found.")

        course.assign_teacher(teacher)
        teacher.assigned_courses.append(course)
        print(f"✅ Teacher {teacher.first_name} assigned to {course.course_name}.")

    def record_payment(self, student, amount, payment_date):
        if student not in self.students:
            raise StudentNotFoundException(f"Student with ID {student.student_id} not found.")
        if amount <= 0:
            raise PaymentValidationException("Payment amount must be greater than 0.")

        payment = Payment(len(self.payments)+1, student.student_id, amount, payment_date)
        payment.student = student
        self.payments.append(payment)
        student.payments.append(payment)
        print(f"✅ Payment of ₹{amount} recorded for {student.first_name} on {payment_date.strftime('%Y-%m-%d')}.")

    def generate_enrollment_report(self, course):
        print(f"📋 Enrollment Report for Course: {course.course_name}")
        for enrollment in course.enrollments:
            student = enrollment.get_student()
            print(f"- {student.first_name} {student.last_name} | Enrolled on: {enrollment.enrollment_date.strftime('%Y-%m-%d')}")

    def generate_payment_report(self, student):
        print(f"📋 Payment Report for Student: {student.first_name} {student.last_name}")
        for payment in student.payments:
            print(f"- ₹{payment.amount} on {payment.payment_date.strftime('%Y-%m-%d')}")

    def calculate_course_statistics(self, course):
        total_enrollments = len(course.enrollments)
        total_payments = 0
        for enrollment in course.enrollments:
            student = enrollment.get_student()
            for payment in student.payments:
                total_payments += payment.amount

        print(f"📊 Statistics for {course.course_name}:")
        print(f"- Total Enrollments: {total_enrollments}")
        print(f"- Total Payments Received: ₹{total_payments}")
```

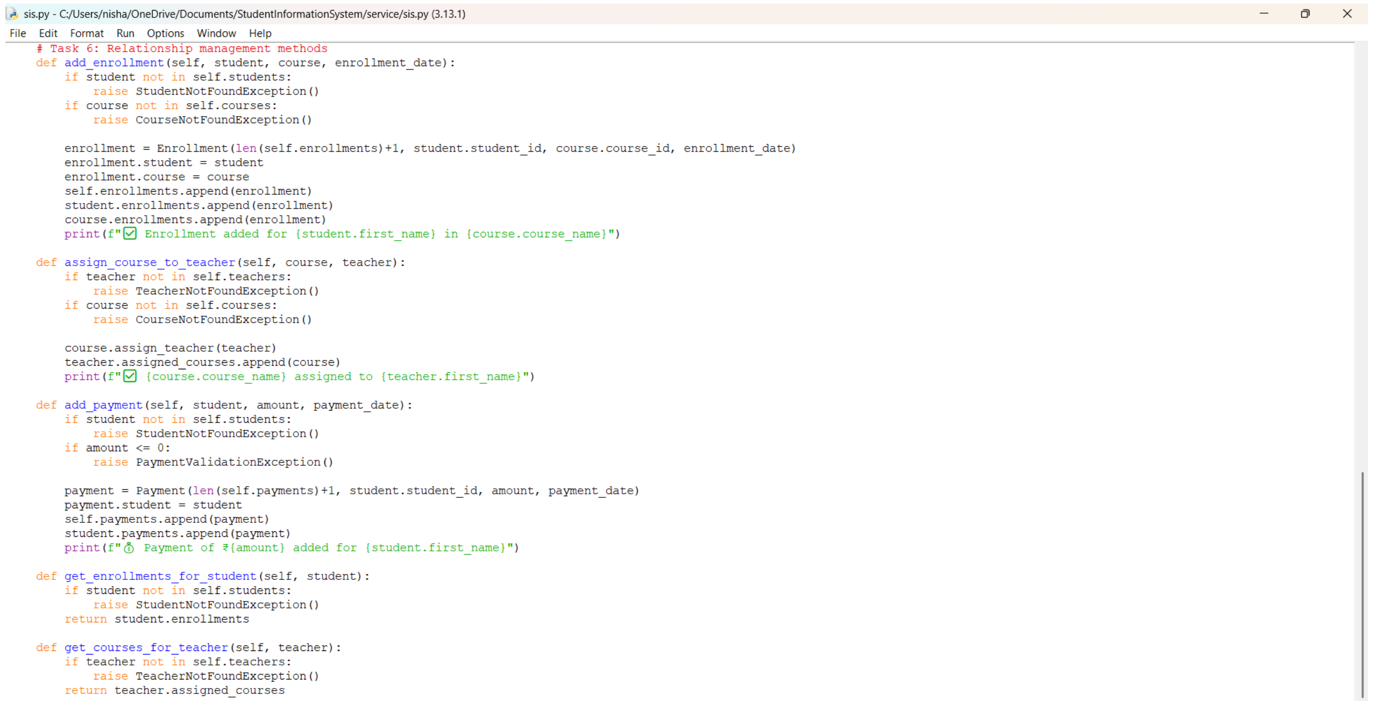
Testing(optional):

```
if __name__ == '__main__':
    from datetime import datetime

    sis = SIS()

    # Create only a student and NOT add to SIS (to trigger exception)
    student = Student(1, "Nisha", "Verma", "2000-05-10", "nisha@example.com", "9876543210")
    course = Course(101, "Python Programming", "PY101", "Unknown")

    try:
        # This will raise StudentNotFoundException
        sis.enroll_student_in_course(student, course)

    except StudentNotFoundException as e:
        print("❌ StudentNotFoundException:", e)

    # Now add the student but not the course
    sis.students.append(student)

    try:
        # This will raise CourseNotFoundException
        sis.enroll_student_in_course(student, course)

    except CourseNotFoundException as e:
        print("❌ CourseNotFoundException:", e)

    # Now add both student and course
    sis.courses.append(course)
    sis.enroll_student_in_course(student, course)  # First enrollment - success ✅

    try:
        # Second time - should raise DuplicateEnrollmentException
        sis.enroll_student_in_course(student, course)

    except DuplicateEnrollmentException as e:
        print("⚠️ DuplicateEnrollmentException:", e)
```



```
================================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/service/sis.py ==================================================
✅ Student Information System initialized.
❌ StudentNotFoundException: Student with ID 1 not found.
❌ CourseNotFoundException: Course with ID 101 not found.
✅ Nisha enrolled in Python Programming.
⚠️ DuplicateEnrollmentException: Nisha is already enrolled in Python Programming.
```

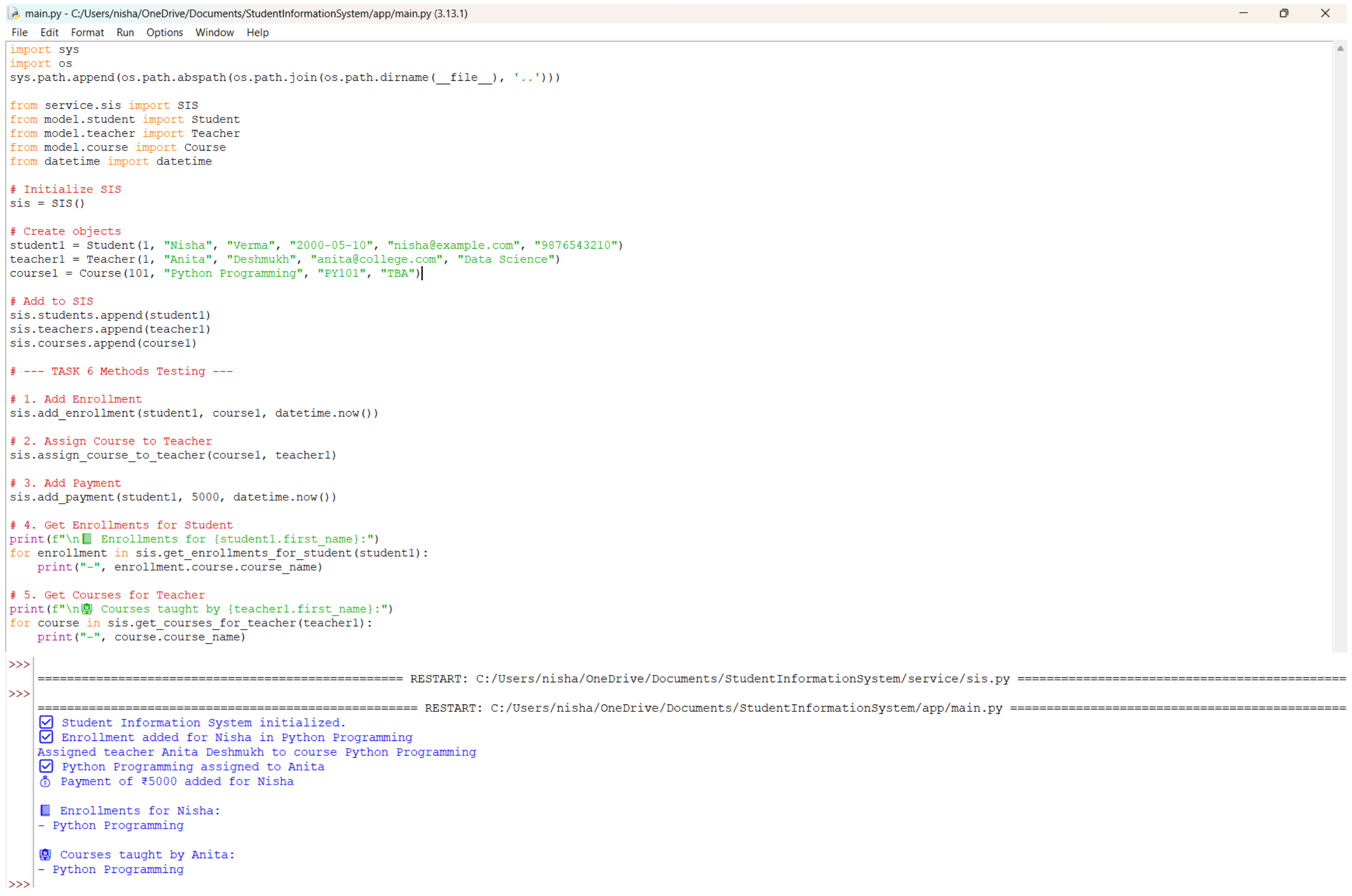
Make the following changes is sis.py, to make all the requirements to match task 6



sis.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/service/sis.py (3.13.1)

```python
    # Task 6: Relationship management methods
    def add_enrollment(self, student, course, enrollment_date):
        if student not in self.students:
            raise StudentNotFoundException()
        if course not in self.courses:
            raise CourseNotFoundException()

        enrollment = Enrollment(len(self.enrollments)+1, student.student_id, course.course_id, enrollment_date)
        enrollment.student = student
        enrollment.course = course
        self.enrollments.append(enrollment)
        student.enrollments.append(enrollment)
        course.enrollments.append(enrollment)
        print(f"✅ Enrollment added for {student.first_name} in {course.course_name}")

    def assign_course_to_teacher(self, course, teacher):
        if teacher not in self.teachers:
            raise TeacherNotFoundException()
        if course not in self.courses:
            raise CourseNotFoundException()

        course.assign_teacher(teacher)
        teacher.assigned_courses.append(course)
        print(f"✅ {course.course_name} assigned to {teacher.first_name}")

    def add_payment(self, student, amount, payment_date):
        if student not in self.students:
            raise StudentNotFoundException()
        if amount <= 0:
            raise PaymentValidationException()

        payment = Payment(len(self.payments)+1, student.student_id, amount, payment_date)
        payment.student = student
        self.payments.append(payment)
        student.payments.append(payment)
        print(f"💰 Payment of ₹{amount} added for {student.first_name}")

    def get_enrollments_for_student(self, student):
        if student not in self.students:
            raise StudentNotFoundException()
        return student.enrollments

    def get_courses_for_teacher(self, teacher):
        if teacher not in self.teachers:
            raise TeacherNotFoundException()
        return teacher.assigned_courses
```

Testing the task 6 implementation in main.py:

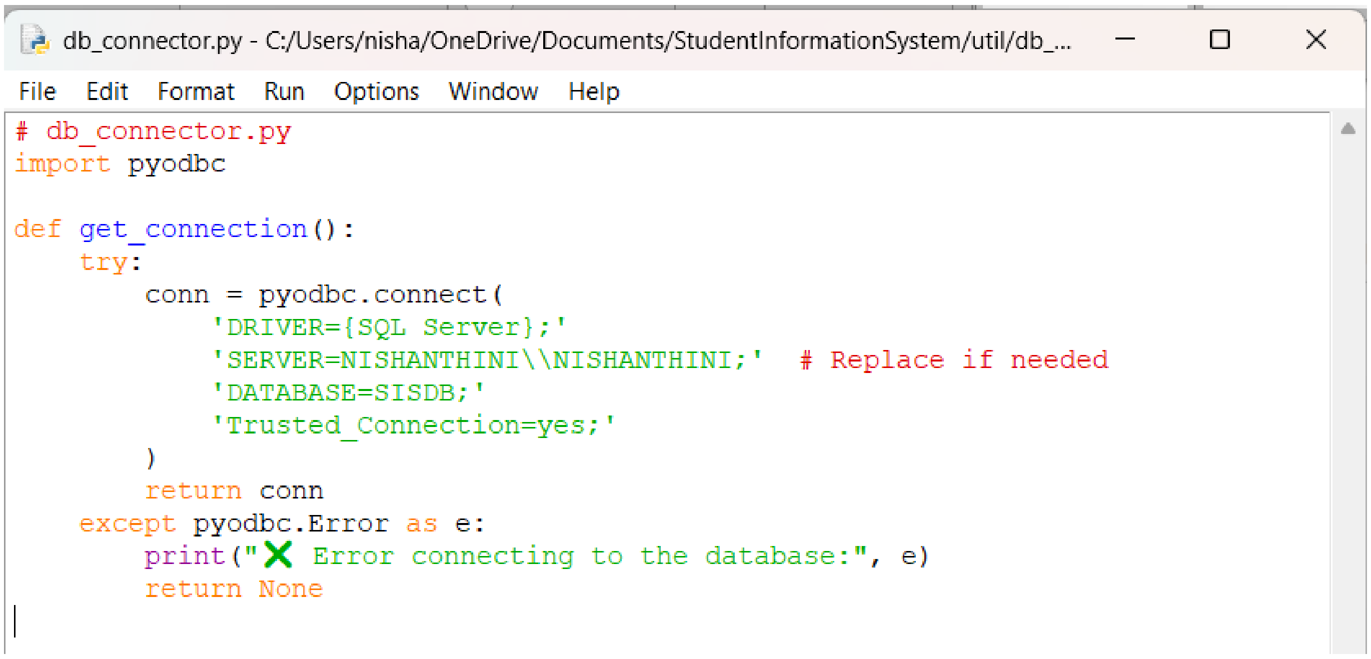


main.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/app/main.py (3.13.1)

```python
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from service.sis import SIS
from model.student import Student
from model.teacher import Teacher
from model.course import Course
from datetime import datetime

# Initialize SIS
sis = SIS()

# Create objects
student1 = Student(1, "Nisha", "Verma", "2000-05-10", "nisha@example.com", "9876543210")
teacher1 = Teacher(1, "Anita", "Deshmukh", "anita@college.com", "Data Science")
course1 = Course(101, "Python Programming", "PY101", "TBA")

# Add to SIS
sis.students.append(student1)
sis.teachers.append(teacher1)
sis.courses.append(course1)

# --- TASK 6 Methods Testing ---

# 1. Add Enrollment
sis.add_enrollment(student1, course1, datetime.now())

# 2. Assign Course to Teacher
sis.assign_course_to_teacher(course1, teacher1)

# 3. Add Payment
sis.add_payment(student1, 5000, datetime.now())

# 4. Get Enrollments for Student
print(f"\n📘 Enrollments for {student1.first_name}:")
for enrollment in sis.get_enrollments_for_student(student1):
    print("-", enrollment.course.course_name)

# 5. Get Courses for Teacher
print(f"\n👩‍🏫 Courses taught by {teacher1.first_name}:")
for course in sis.get_courses_for_teacher(teacher1):
    print("-", course.course_name)
```

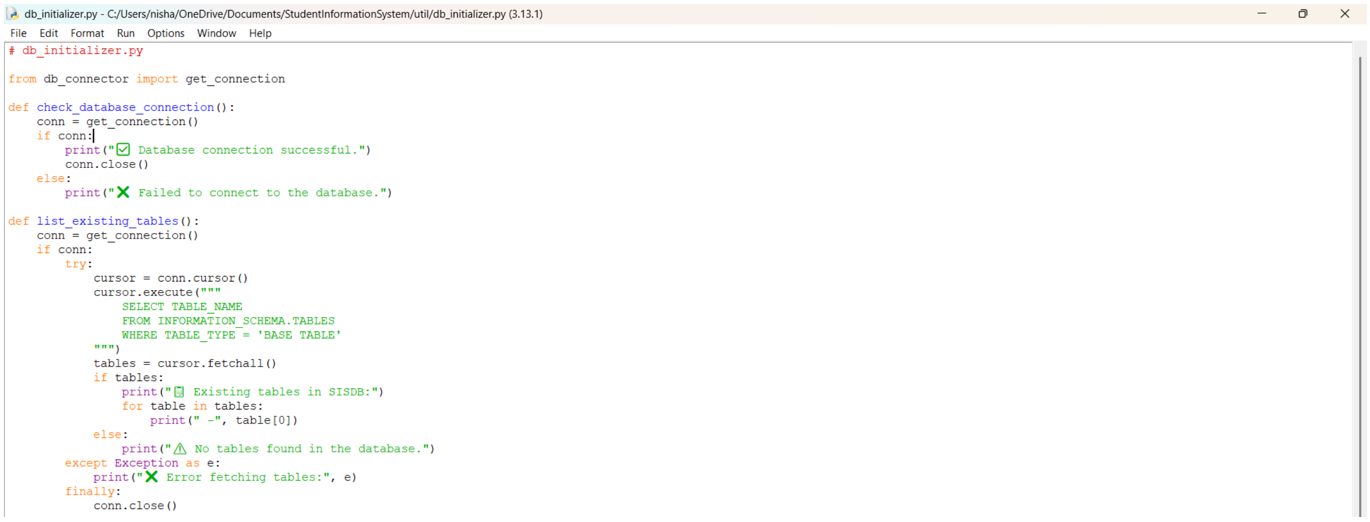
```
>>>
    ==================================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/service/sis.py ==============================================
>>>
    ====================================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/app/main.py ==============================================
    ✅ Student Information System initialized.
    ✅ Enrollment added for Nisha in Python Programming
    Assigned teacher Anita Deshmukh to course Python Programming
    ✅ Python Programming assigned to Anita
    💰 Payment of ₹5000 added for Nisha

    📘 Enrollments for Nisha:
    - Python Programming

    👩‍🏫 Courses taught by Anita:
    - Python Programming
>>>
```

Task 7



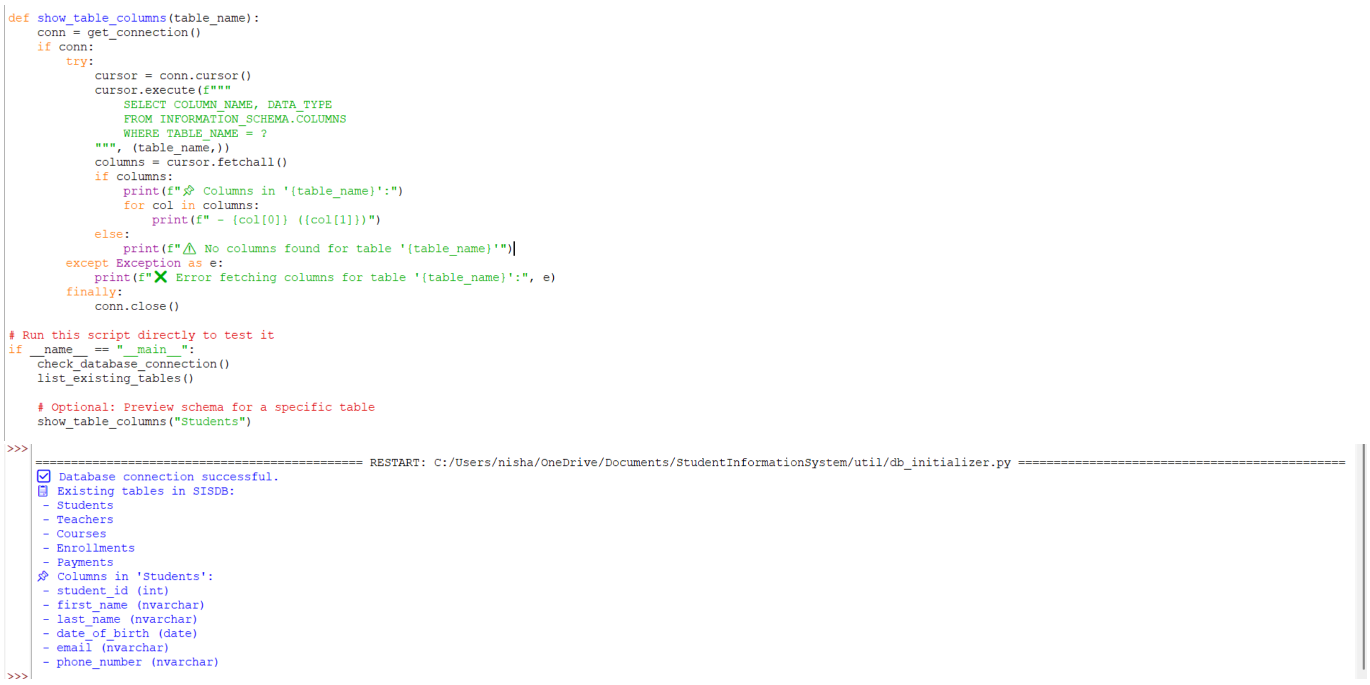
db_connector.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/util/db_...
File Edit Format Run Options Window Help

```python
# db_connector.py
import pyodbc

def get_connection():
    try:
        conn = pyodbc.connect(
            'DRIVER={SQL Server};'
            'SERVER=NISHANTHINI\\NISHANTHINI;'  # Replace if needed
            'DATABASE=SISDB;'
            'Trusted_Connection=yes;'
        )
        return conn
    except pyodbc.Error as e:
        print("❌ Error connecting to the database:", e)
        return None
```



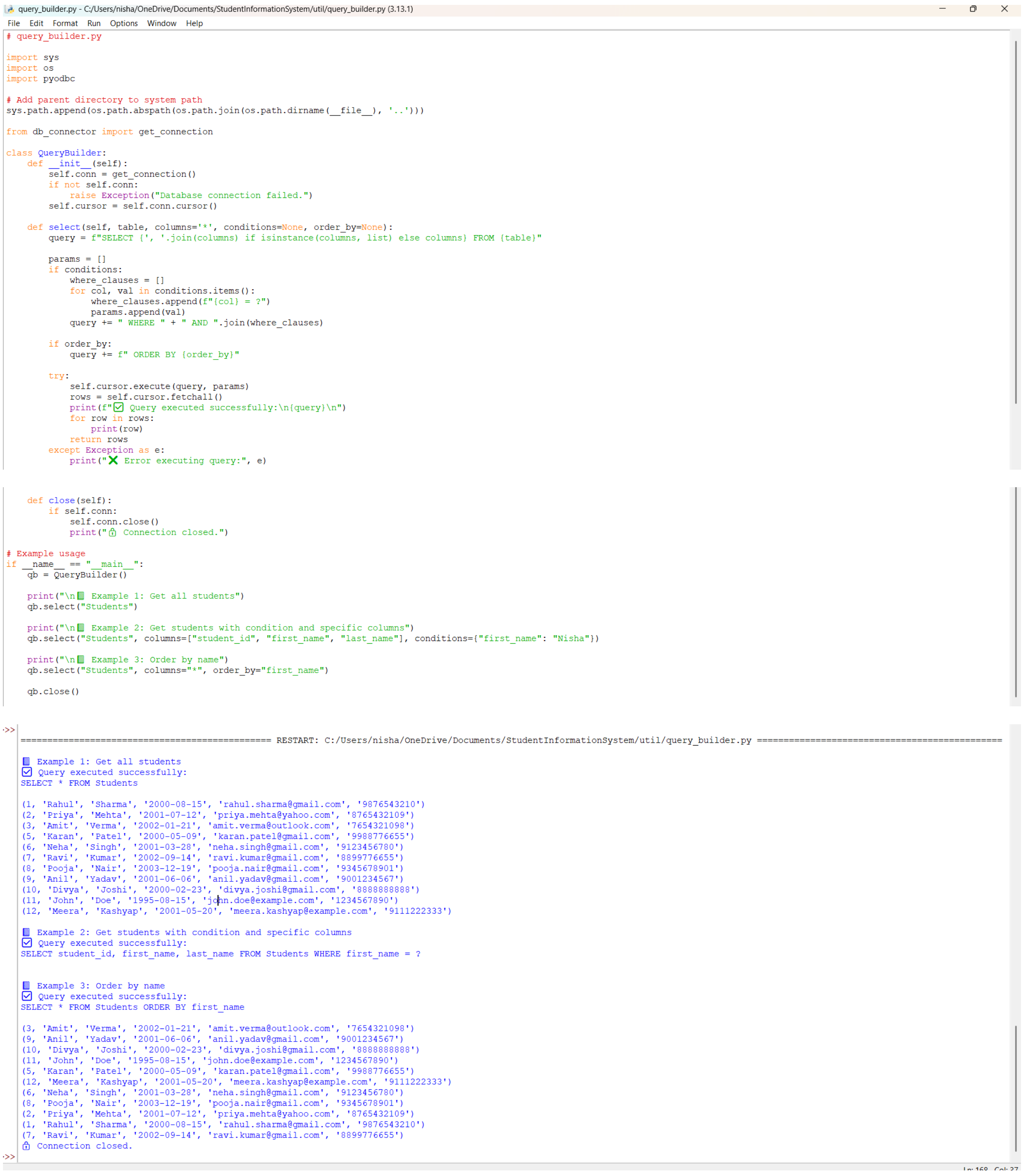
db_initializer.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/util/db_initializer.py (3.13.1)
File Edit Format Run Options Window Help

```python
# db_initializer.py

from db_connector import get_connection

def check_database_connection():
    conn = get_connection()
    if conn:
        print("✅ Database connection successful.")
        conn.close()
    else:
        print("❌ Failed to connect to the database.")

def list_existing_tables():
    conn = get_connection()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute("""
                SELECT TABLE_NAME
                FROM INFORMATION_SCHEMA.TABLES
                WHERE TABLE_TYPE = 'BASE TABLE'
            """)
            tables = cursor.fetchall()
            if tables:
                print("📋 Existing tables in SISDB:")
                for table in tables:
                    print(" -", table[0])
            else:
                print("⚠ No tables found in the database.")
        except Exception as e:
            print("❌ Error fetching tables:", e)
        finally:
            conn.close()
```



```python
def show_table_columns(table_name):
    conn = get_connection()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute(f"""
                SELECT COLUMN_NAME, DATA_TYPE
                FROM INFORMATION_SCHEMA.COLUMNS
                WHERE TABLE_NAME = ?
            """, (table_name,))
            columns = cursor.fetchall()
            if columns:
                print(f"📌 Columns in '{table_name}':")
                for col in columns:
                    print(f" - {col[0]} ({col[1]})")
            else:
                print(f"⚠ No columns found for table '{table_name}'")
        except Exception as e:
            print(f"❌ Error fetching columns for table '{table_name}':", e)
        finally:
            conn.close()

# Run this script directly to test it
if __name__ == "__main__":
    check_database_connection()
    list_existing_tables()

    # Optional: Preview schema for a specific table
    show_table_columns("Students")
```

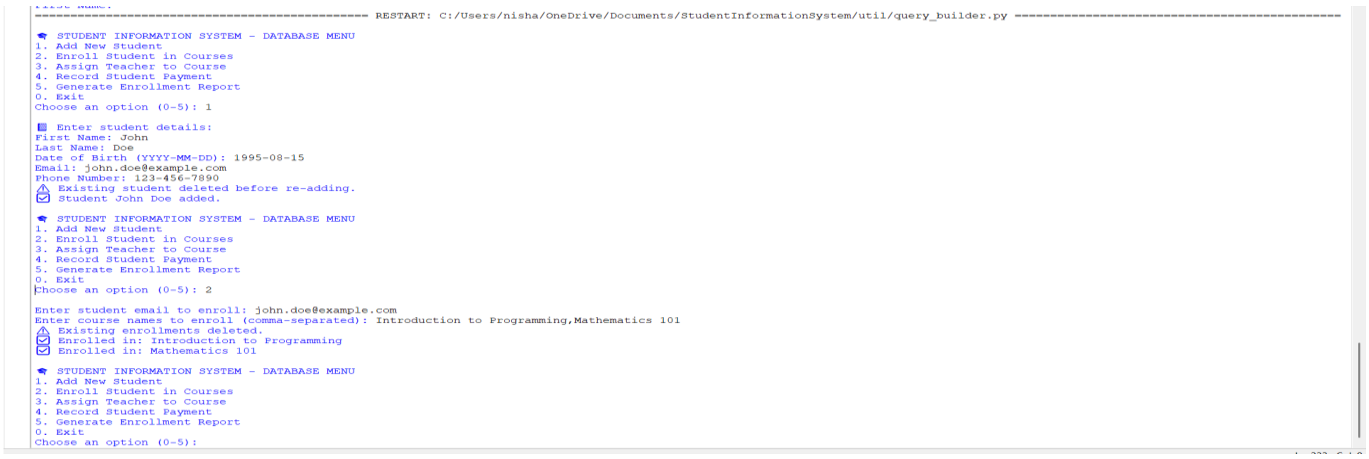
```
>>>
============================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/util/db_initializer.py ==============================================
✅ Database connection successful.
📋 Existing tables in SISDB:
 - Students
 - Teachers
 - Courses
 - Enrollments
 - Payments
📌 Columns in 'Students':
 - student_id (int)
 - first_name (nvarchar)
 - last_name (nvarchar)
 - date_of_birth (date)
 - email (nvarchar)
 - phone_number (nvarchar)
>>>
```


query_builder.py - C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/util/query_builder.py (3.13.1)
File Edit Format Run Options Window Help
Ln: 168 Col: 37

```python
# query_builder.py

import sys
import os
import pyodbc

# Add parent directory to system path
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from db_connector import get_connection

class QueryBuilder:
    def __init__(self):
        self.conn = get_connection()
        if not self.conn:
            raise Exception("Database connection failed.")
        self.cursor = self.conn.cursor()

    def select(self, table, columns='*', conditions=None, order_by=None):
        query = f"SELECT {', '.join(columns) if isinstance(columns, list) else columns} FROM {table}"

        params = []
        if conditions:
            where_clauses = []
            for col, val in conditions.items():
                where_clauses.append(f"{col} = ?")
                params.append(val)
            query += " WHERE " + " AND ".join(where_clauses)

        if order_by:
            query += f" ORDER BY {order_by}"

        try:
            self.cursor.execute(query, params)
            rows = self.cursor.fetchall()
            print(f"✅ Query executed successfully:\n{query}\n")
            for row in rows:
                print(row)
            return rows
        except Exception as e:
            print("❌ Error executing query:", e)


    def close(self):
        if self.conn:
            self.conn.close()
            print("🔒 Connection closed.")

# Example usage
if __name__ == "__main__":
    qb = QueryBuilder()

    print("\n📘 Example 1: Get all students")
    qb.select("Students")

    print("\n📘 Example 2: Get students with condition and specific columns")
    qb.select("Students", columns=["student_id", "first_name", "last_name"], conditions={"first_name": "Nisha"})

    print("\n📘 Example 3: Order by name")
    qb.select("Students", columns="*", order_by="first_name")

    qb.close()
```

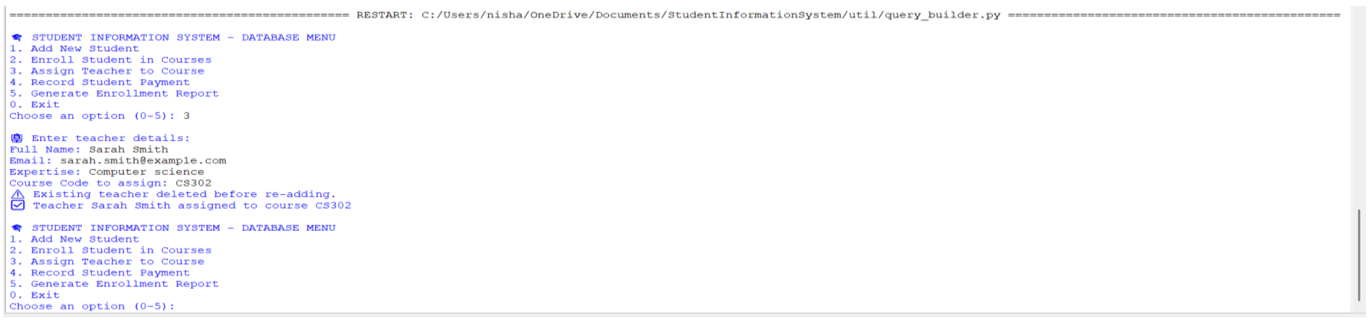
```
>>>
=============================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/util/query_builder.py ==============================================

📘 Example 1: Get all students
✅ Query executed successfully:
SELECT * FROM Students

(1, 'Rahul', 'Sharma', '2000-08-15', 'rahul.sharma@gmail.com', '9876543210')
(2, 'Priya', 'Mehta', '2001-07-12', 'priya.mehta@yahoo.com', '8765432109')
(3, 'Amit', 'Verma', '2002-01-21', 'amit.verma@outlook.com', '7654321098')
(5, 'Karan', 'Patel', '2000-05-09', 'karan.patel@gmail.com', '9988776655')
(6, 'Neha', 'Singh', '2001-03-28', 'neha.singh@gmail.com', '9123456780')
(7, 'Ravi', 'Kumar', '2002-09-14', 'ravi.kumar@gmail.com', '8899776655')
(8, 'Pooja', 'Nair', '2003-12-19', 'pooja.nair@gmail.com', '9345678901')
(9, 'Anil', 'Yadav', '2001-06-06', 'anil.yadav@gmail.com', '9001234567')
(10, 'Divya', 'Joshi', '2000-02-23', 'divya.joshi@gmail.com', '8888888888')
(11, 'John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890')
(12, 'Meera', 'Kashyap', '2001-05-20', 'meera.kashyap@example.com', '9111222333')

📘 Example 2: Get students with condition and specific columns
✅ Query executed successfully:
SELECT student_id, first_name, last_name FROM Students WHERE first_name = ?


📘 Example 3: Order by name
✅ Query executed successfully:
SELECT * FROM Students ORDER BY first_name

(3, 'Amit', 'Verma', '2002-01-21', 'amit.verma@outlook.com', '7654321098')
(9, 'Anil', 'Yadav', '2001-06-06', 'anil.yadav@gmail.com', '9001234567')
(10, 'Divya', 'Joshi', '2000-02-23', 'divya.joshi@gmail.com', '8888888888')
(11, 'John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890')
(5, 'Karan', 'Patel', '2000-05-09', 'karan.patel@gmail.com', '9988776655')
(12, 'Meera', 'Kashyap', '2001-05-20', 'meera.kashyap@example.com', '9111222333')
(6, 'Neha', 'Singh', '2001-03-28', 'neha.singh@gmail.com', '9123456780')
(8, 'Pooja', 'Nair', '2003-12-19', 'pooja.nair@gmail.com', '9345678901')
(2, 'Priya', 'Mehta', '2001-07-12', 'priya.mehta@yahoo.com', '8765432109')
(1, 'Rahul', 'Sharma', '2000-08-15', 'rahul.sharma@gmail.com', '9876543210')
(7, 'Ravi', 'Kumar', '2002-09-14', 'ravi.kumar@gmail.com', '8899776655')
🔒 Connection closed.
>>>
```

Task 8



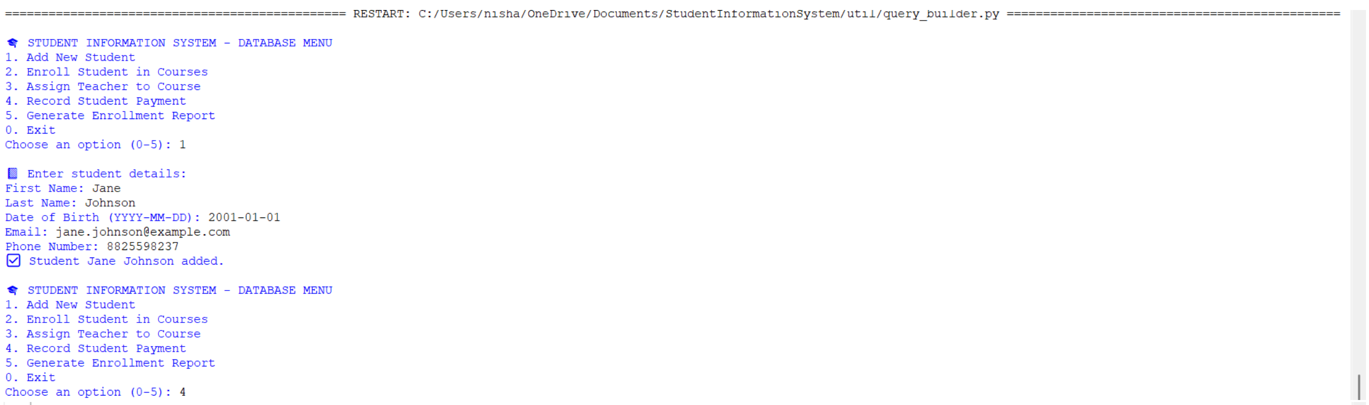
```
============================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/util/query_builder.py ==============================================

🎓 STUDENT INFORMATION SYSTEM - DATABASE MENU
1. Add New Student
2. Enroll Student in Courses
3. Assign Teacher to Course
4. Record Student Payment
5. Generate Enrollment Report
0. Exit
Choose an option (0-5): 1

📘 Enter student details:
First Name: John
Last Name: Doe
Date of Birth (YYYY-MM-DD): 1995-08-15
Email: john.doe@example.com
Phone Number: 123-456-7890
⚠️ Existing student deleted before re-adding.
✅ Student John Doe added.

🎓 STUDENT INFORMATION SYSTEM - DATABASE MENU
1. Add New Student
2. Enroll Student in Courses
3. Assign Teacher to Course
4. Record Student Payment
5. Generate Enrollment Report
0. Exit
Choose an option (0-5): 2

Enter student email to enroll: john.doe@example.com
Enter course names to enroll (comma-separated): Introduction to Programming,Mathematics 101
⚠️ Existing enrollments deleted.
✅ Enrolled in: Introduction to Programming
✅ Enrolled in: Mathematics 101

🎓 STUDENT INFORMATION SYSTEM - DATABASE MENU
1. Add New Student
2. Enroll Student in Courses
3. Assign Teacher to Course
4. Record Student Payment
5. Generate Enrollment Report
0. Exit
Choose an option (0-5):
```

Task 9(before this I have made changes to mssql document)alter command

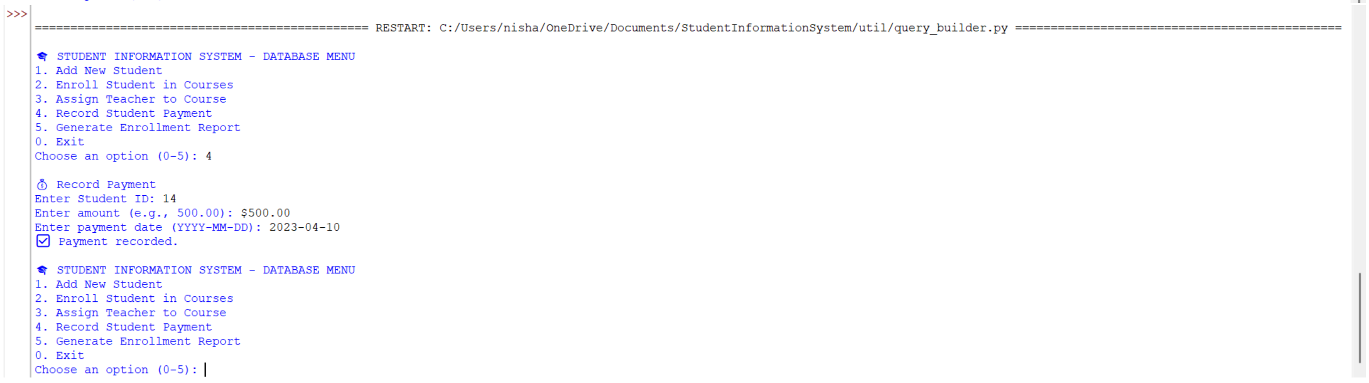


```
============================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/util/query_builder.py ==============================================

🎓 STUDENT INFORMATION SYSTEM - DATABASE MENU
1. Add New Student
2. Enroll Student in Courses
3. Assign Teacher to Course
4. Record Student Payment
5. Generate Enrollment Report
0. Exit
Choose an option (0-5): 3

🧑‍🏫 Enter teacher details:
Full Name: Sarah Smith
Email: sarah.smith@example.com
Expertise: Computer science
Course Code to assign: CS302
⚠️ Existing teacher deleted before re-adding.
✅ Teacher Sarah Smith assigned to course CS302

🎓 STUDENT INFORMATION SYSTEM - DATABASE MENU
1. Add New Student
2. Enroll Student in Courses
3. Assign Teacher to Course
4. Record Student Payment
5. Generate Enrollment Report
0. Exit
Choose an option (0-5):
```

Task 10(insert new student jane johnson with basic details,and then proceed)

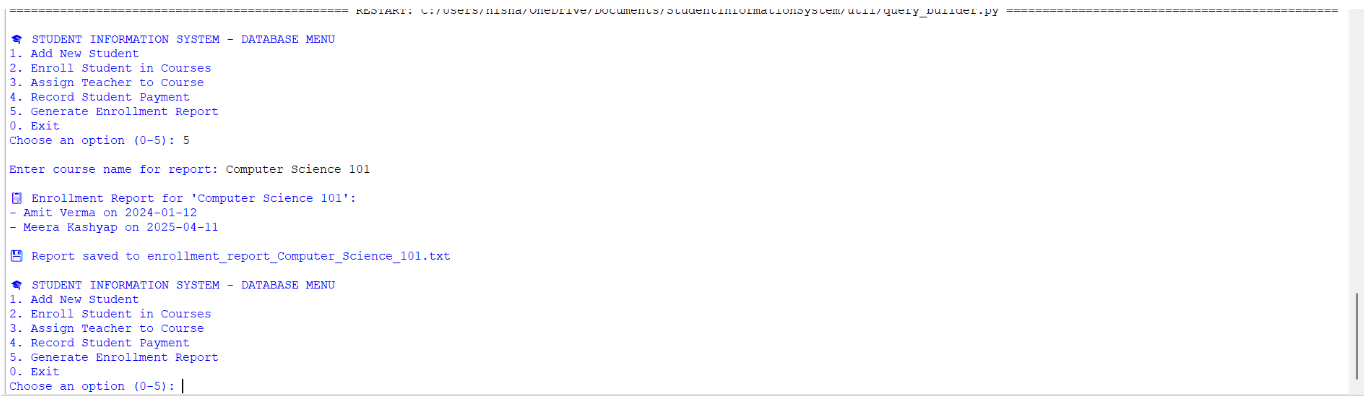


```
============================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/util/query_builder.py ==============================================

🎓 STUDENT INFORMATION SYSTEM - DATABASE MENU
1. Add New Student
2. Enroll Student in Courses
3. Assign Teacher to Course
4. Record Student Payment
5. Generate Enrollment Report
0. Exit
Choose an option (0-5): 1

📘 Enter student details:
First Name: Jane
Last Name: Johnson
Date of Birth (YYYY-MM-DD): 2001-01-01
Email: jane.johnson@example.com
Phone Number: 8825598237
✅ Student Jane Johnson added.

🎓 STUDENT INFORMATION SYSTEM - DATABASE MENU
1. Add New Student
2. Enroll Student in Courses
3. Assign Teacher to Course
4. Record Student Payment
5. Generate Enrollment Report
0. Exit
Choose an option (0-5): 4
>>>
```



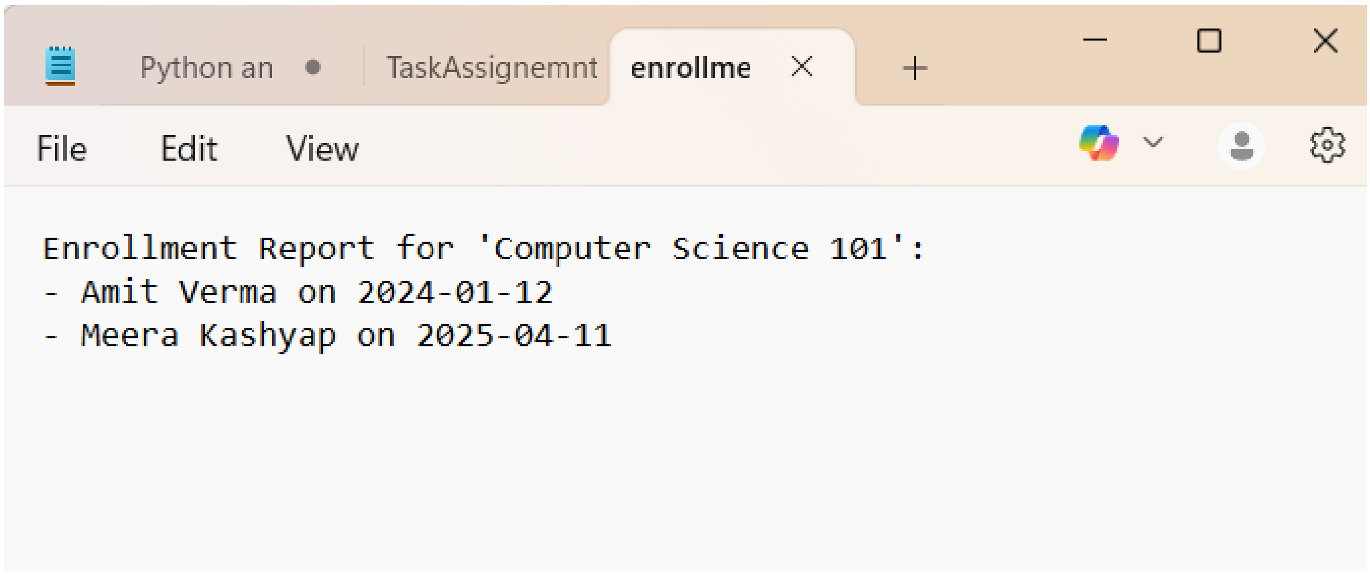
```
============================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/util/query_builder.py ==============================================

🎓 STUDENT INFORMATION SYSTEM - DATABASE MENU
1. Add New Student
2. Enroll Student in Courses
3. Assign Teacher to Course
4. Record Student Payment
5. Generate Enrollment Report
0. Exit
Choose an option (0-5): 4

💰 Record Payment
Enter Student ID: 14
Enter amount (e.g., 500.00): $500.00
Enter payment date (YYYY-MM-DD): 2023-04-10
✅ Payment recorded.

🎓 STUDENT INFORMATION SYSTEM - DATABASE MENU
1. Add New Student
2. Enroll Student in Courses
3. Assign Teacher to Course
4. Record Student Payment
5. Generate Enrollment Report
0. Exit
Choose an option (0-5):
```

TASK 11:



```
============================================== RESTART: C:/Users/nisha/OneDrive/Documents/StudentInformationSystem/util/query_builder.py ==============================================

🎓 STUDENT INFORMATION SYSTEM - DATABASE MENU
1. Add New Student
2. Enroll Student in Courses
3. Assign Teacher to Course
4. Record Student Payment
5. Generate Enrollment Report
0. Exit
Choose an option (0-5): 5

Enter course name for report: Computer Science 101

📋 Enrollment Report for 'Computer Science 101':
- Amit Verma on 2024-01-12
- Meera Kashyap on 2025-04-11

💾 Report saved to enrollment_report_Computer_Science_101.txt

🎓 STUDENT INFORMATION SYSTEM - DATABASE MENU
1. Add New Student
2. Enroll Student in Courses
3. Assign Teacher to Course
4. Record Student Payment
5. Generate Enrollment Report
0. Exit
Choose an option (0-5): 
```

Text file where the enrolment report can be made



```
Enrollment Report for 'Computer Science 101':
- Amit Verma on 2024-01-12
- Meera Kashyap on 2025-04-11
```