Assignment 2

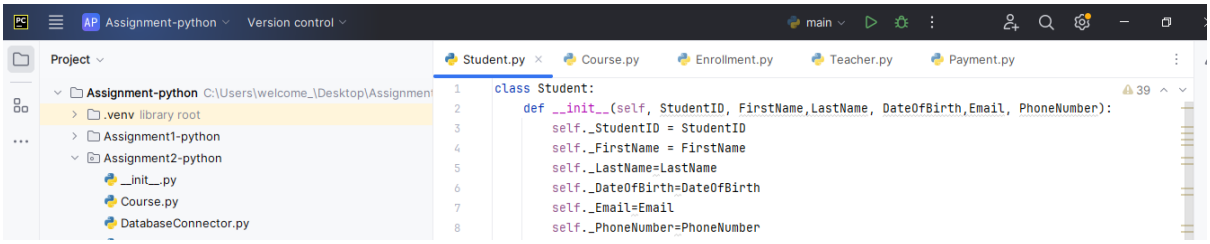
Task 1: Define Classes

Task 2: Implement Constructors

Define the following classes based on the domain description:

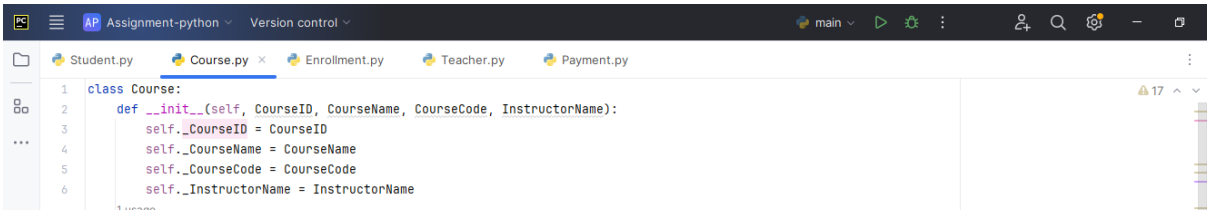
Student class with the following attributes:

- Student ID
- First Name
- Last Name
- Date of Birth
- Email
- Phone Number



Course class with the following attributes:

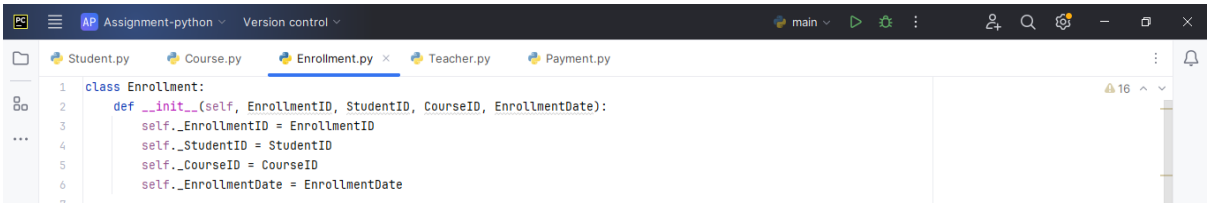
- Course ID
- Course Name
- Course Code
- Instructor Name



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



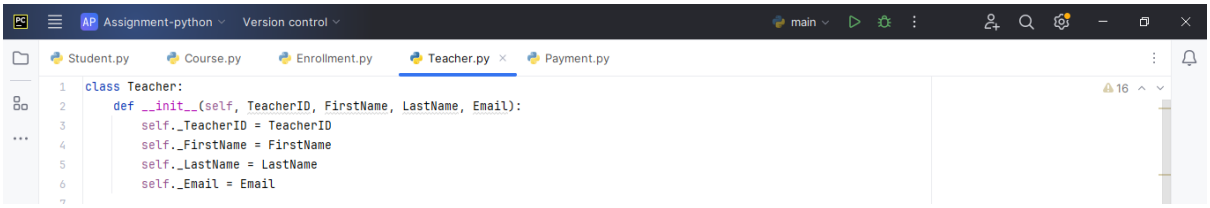
Teacher class with the following attributes:

• Teacher ID

• First Name

• Last Name

• Email



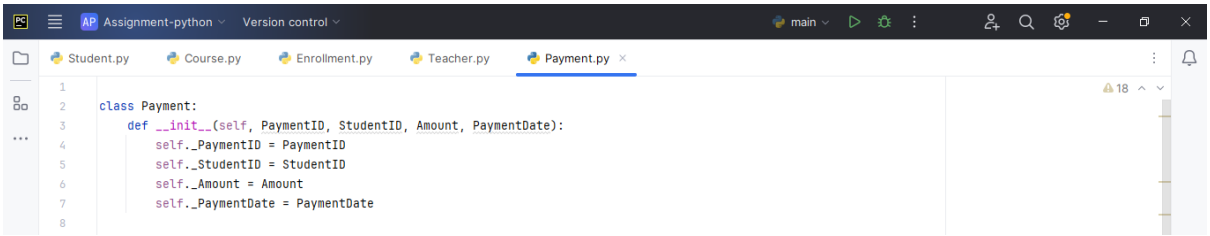
Payment class with the following attributes:

• Payment ID

• Student ID (reference to a Student)

• Amount

• Payment Date



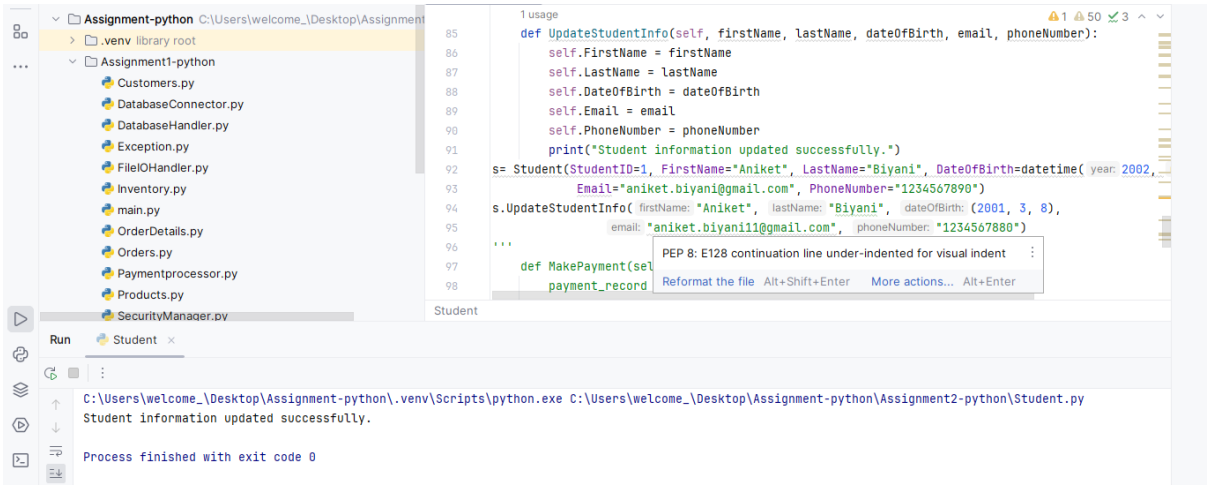
Task 3: Implement Methods

Task 5:

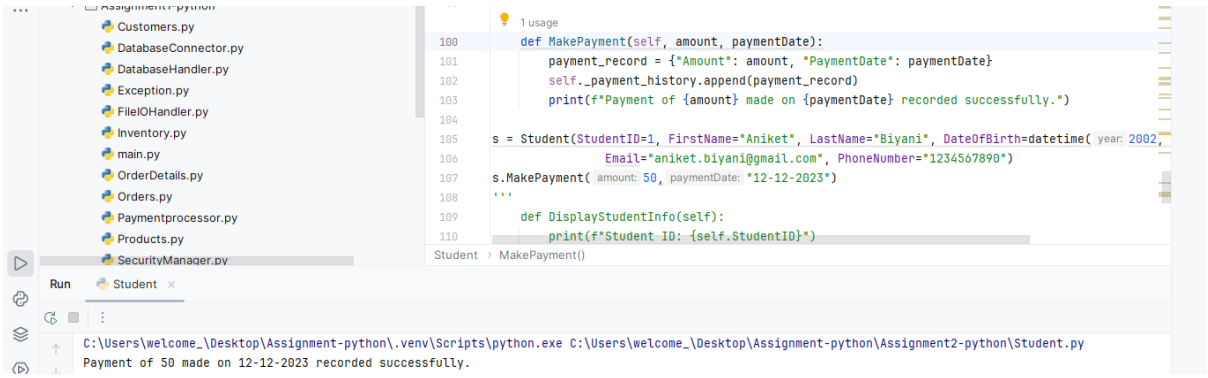
Student Class:

• EnrollInCourse(course: Course): Enrolls the student in a course.

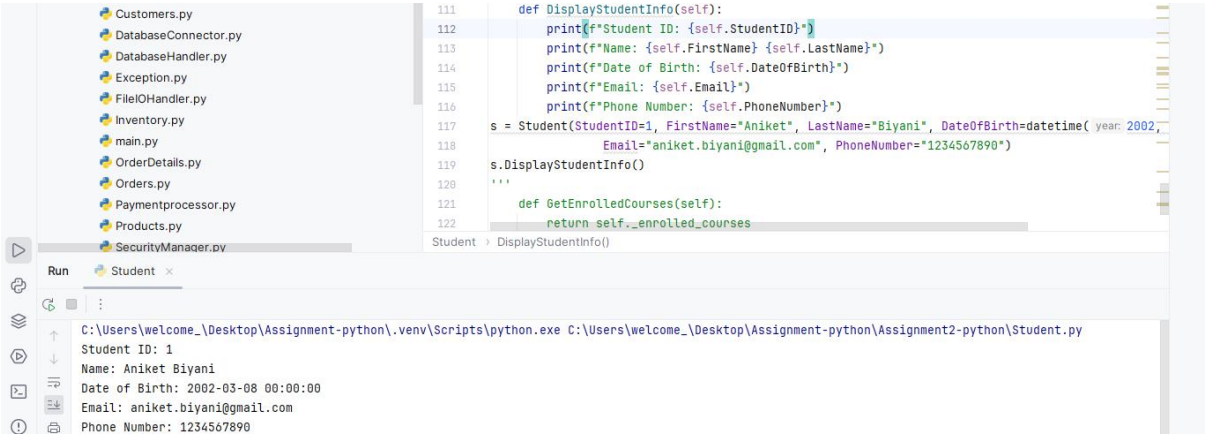
• UpdateStudentInfo(firstName: string, lastName: string, dateOfBirth: DateTime, email: string, phoneNumber: string): Updates the student's information.



• MakePayment(amount: decimal, paymentDate: DateTime): Records a payment made by thestudent.

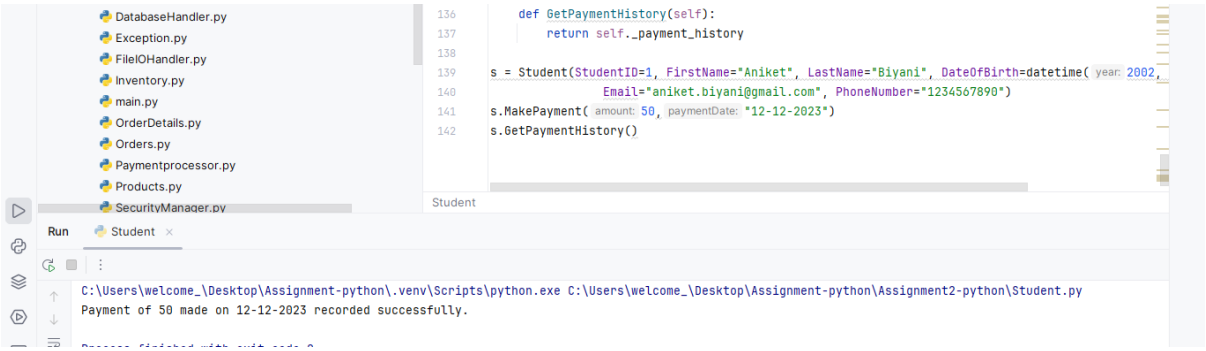


• DisplayStudentInfo(): Displays detailed information about the student.



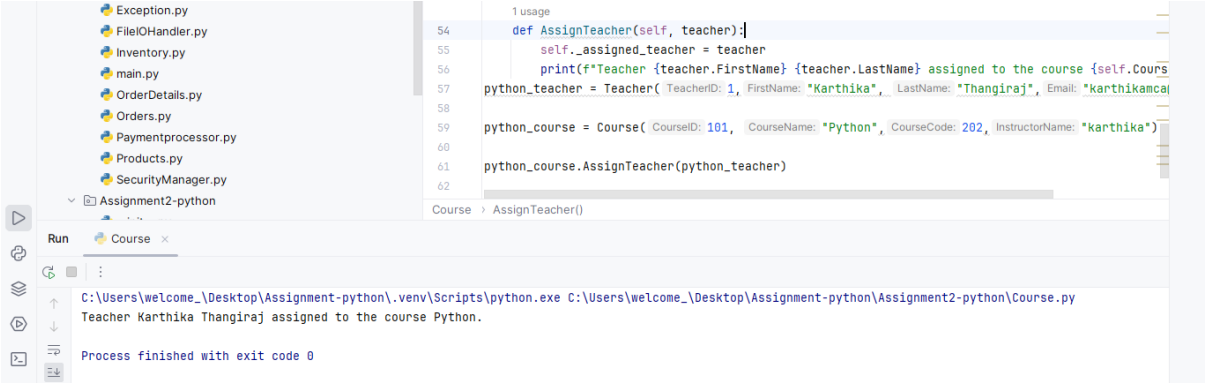
- GetEnrolledCourses(): Retrieves a list of courses in which the student is enrolled.

- GetPaymentHistory(): Retrieves a list of payment records for the student.



Course Class:

- AssignTeacher(teacher: Teacher): Assigns a teacher to the course.

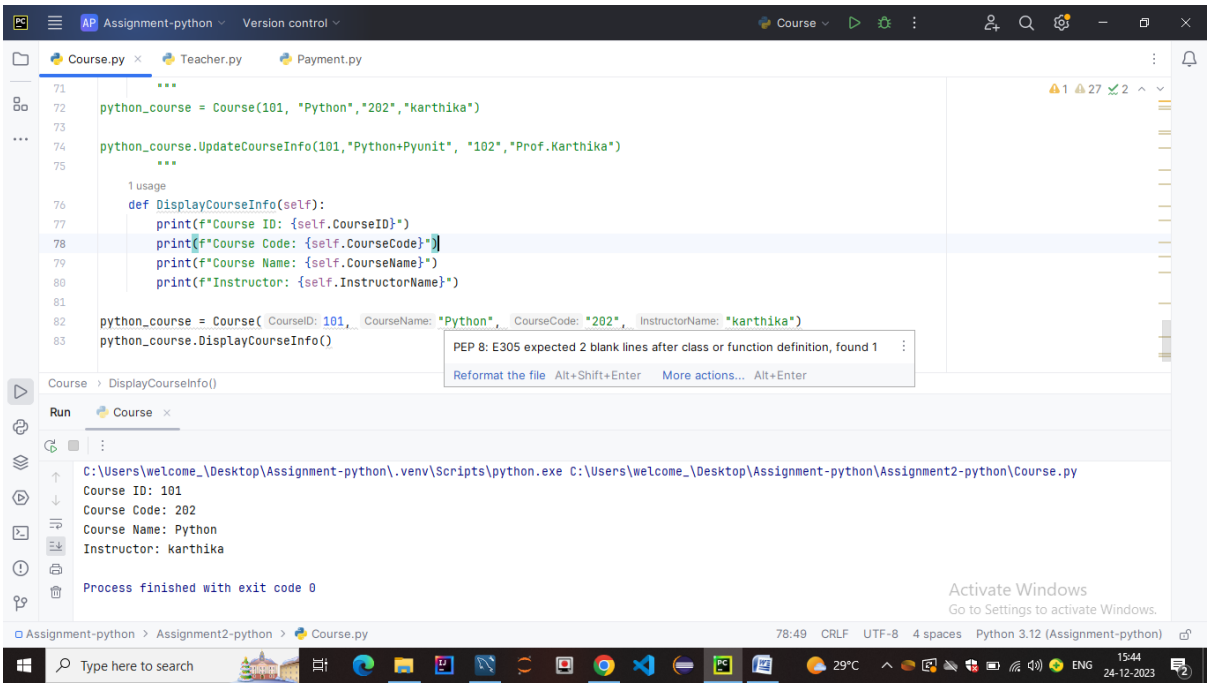


• UpdateCourseInfo(courseCode: string, courseName: string, instructor: string): Updates courseinformation.

```
1 usage
def UpdateCourseInfo(self,courseid,courseName, courseCode, instructor):
    self.CourseID = courseid
    self.CourseName = courseName
    self.CourseCode = courseCode
    self.InstructorName = instructor
    print("Course information updated successfully.")
python_course = Course( CourseID: 101, CourseName: "Python", CourseCode: "202", InstructorName: "karthika")

python_course.UpdateCourseInfo( courseid: 101, courseName: "Python+Pyunit", courseCode: "102", instructor: "Prof.Karthika")
```

```
Course › UpdateCourseInfo()
Run  Course
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment2-python\Course.py
Course information updated successfully.

Process finished with exit code 0
```

• DisplayCourseInfo(): Displays detailed information about the course.



```
Assignment-python  Version control  Course
Course.py  Teacher.py  Payment.py
        """
python_course = Course(101, "Python","202","karthika")

python_course.UpdateCourseInfo(101,"Python+Pyunit", "102","Prof.Karthika")
        """
    1 usage
    def DisplayCourseInfo(self):
        print(f"Course ID: {self.CourseID}")
        print(f"Course Code: {self.CourseCode}")
        print(f"Course Name: {self.CourseName}")
        print(f"Instructor: {self.InstructorName}")

python_course = Course( CourseID: 101, CourseName: "Python", CourseCode: "202", InstructorName: "karthika")
python_course.DisplayCourseInfo()
PEP 8: E305 expected 2 blank lines after class or function definition, found 1
Reformat the file Alt+Shift+Enter  More actions... Alt+Enter
Course › DisplayCourseInfo()
Run  Course
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment2-python\Course.py
Course ID: 101
Course Code: 202
Course Name: Python
Instructor: karthika

Process finished with exit code 0
Activate Windows
Go to Settings to activate Windows.
Assignment-python › Assignment2-python › Course.py  78:49  CRLF  UTF-8  4 spaces  Python 3.12 (Assignment-python)
Type here to search  29°C  ENG  15:44  24-12-2023
```

• GetEnrollments(): Retrieves a list of student enrollments for the course.

```
def GetEnrollments(self):
    return self._enrollments

student_1 = Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime( year: 2002, month: 8, day: 3), Email="aniket@email.com", Pho
student_2 = Student(StudentID=2, FirstName="Abhi", LastName="Surya", DateOfBirth=datetime( year: 2001, month: 2, day: 2), Email="abhi@email.com", PhoneNum

python_course = Course( CourseID: 101, CourseName: "Python", CourseCode: "202", InstructorName: "karthika")
student_1.EnrollInCourse(python_course)
student_2.EnrollInCourse(python_course)

enrollments = python_course.GetEnrollments()

print("Student Enrollments:")
for enrollment in enrollments:
    print(f"- Student ID: {enrollment.StudentID}, Student Name: {enrollment.FirstName} {enrollment.LastName}")
```

```
Run  Course
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment2-python\Course.py
Student Aniket Biyani enrolled in Python.
Student Abhi Surya enrolled in Python.
Student Enrollments:
```

• GetTeacher(): Retrieves the assigned teacher for the course.

Enrollment Class:

• GetStudent(): Retrieves the student associated with the enrollment.

Teacher Class:

• UpdateTeacherInfo(name: string, email: string, expertise: string): Updates teacher information.

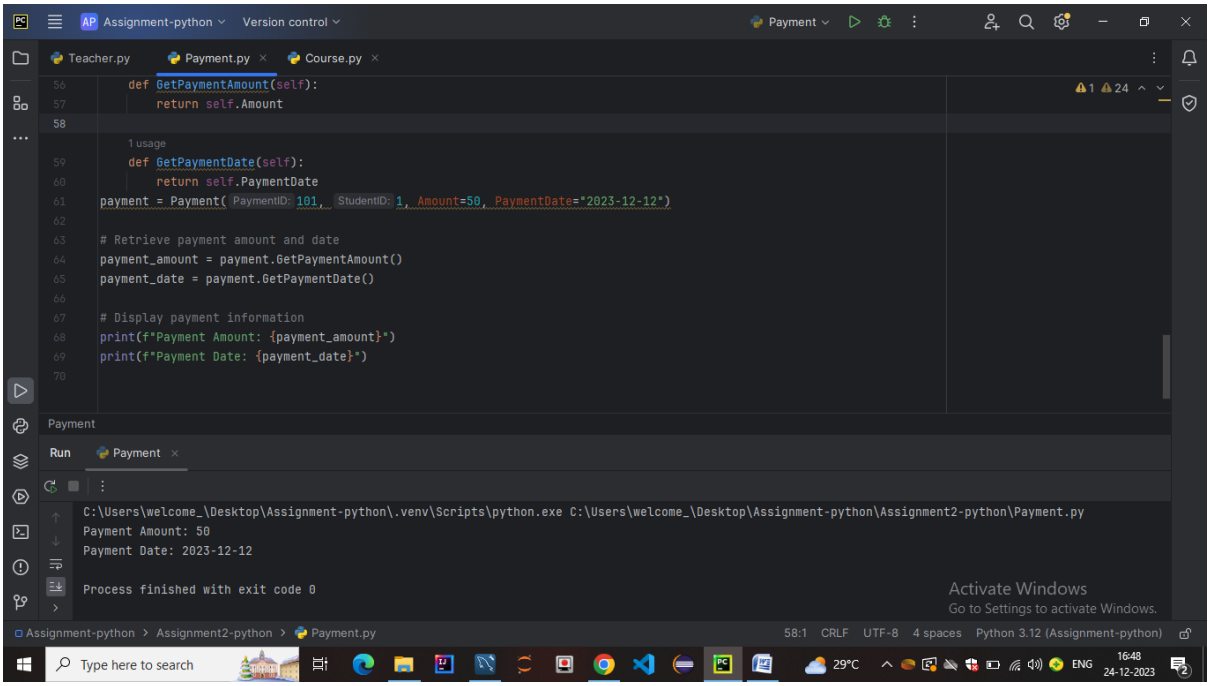
• DisplayTeacherInfo(): Displays detailed information about the teacher.

Payment Class:

• GetStudent(): Retrieves the student associated with the payment.

• GetPaymentAmount(): Retrieves the payment amount.

• GetPaymentDate(): Retrieves the payment date.



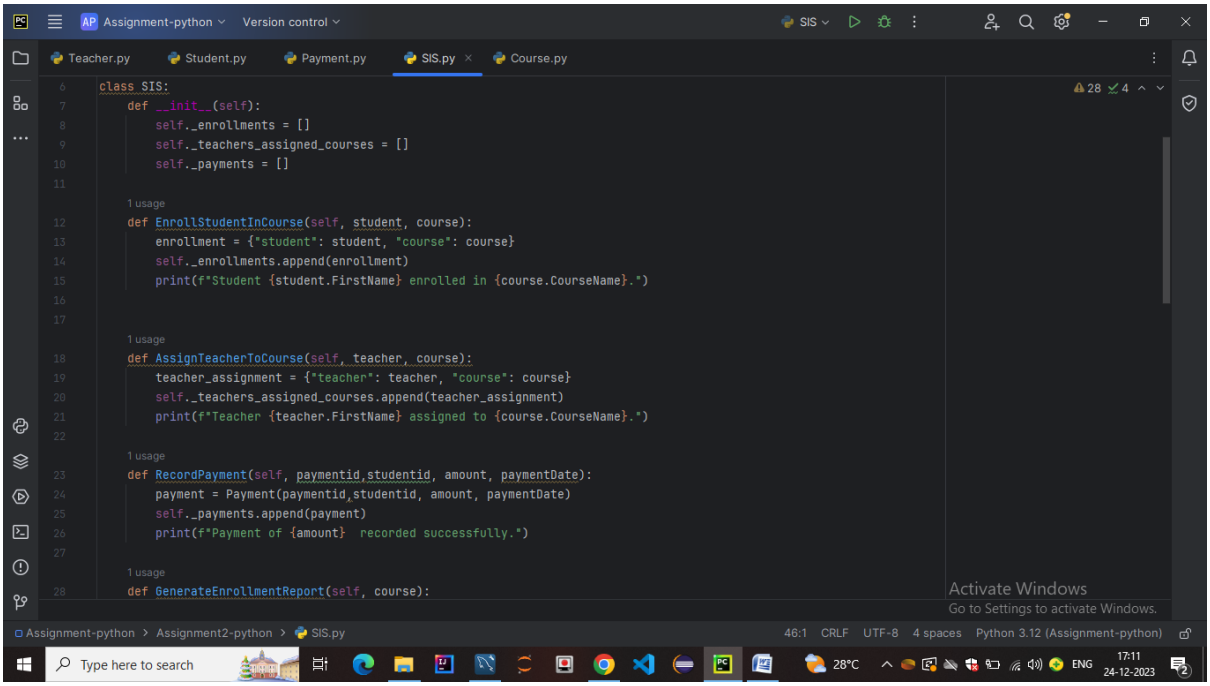
SIS Class (if you have one to manage interactions):

• EnrollStudentInCourse(student: Student, course: Course): Enrolls a student in a course.

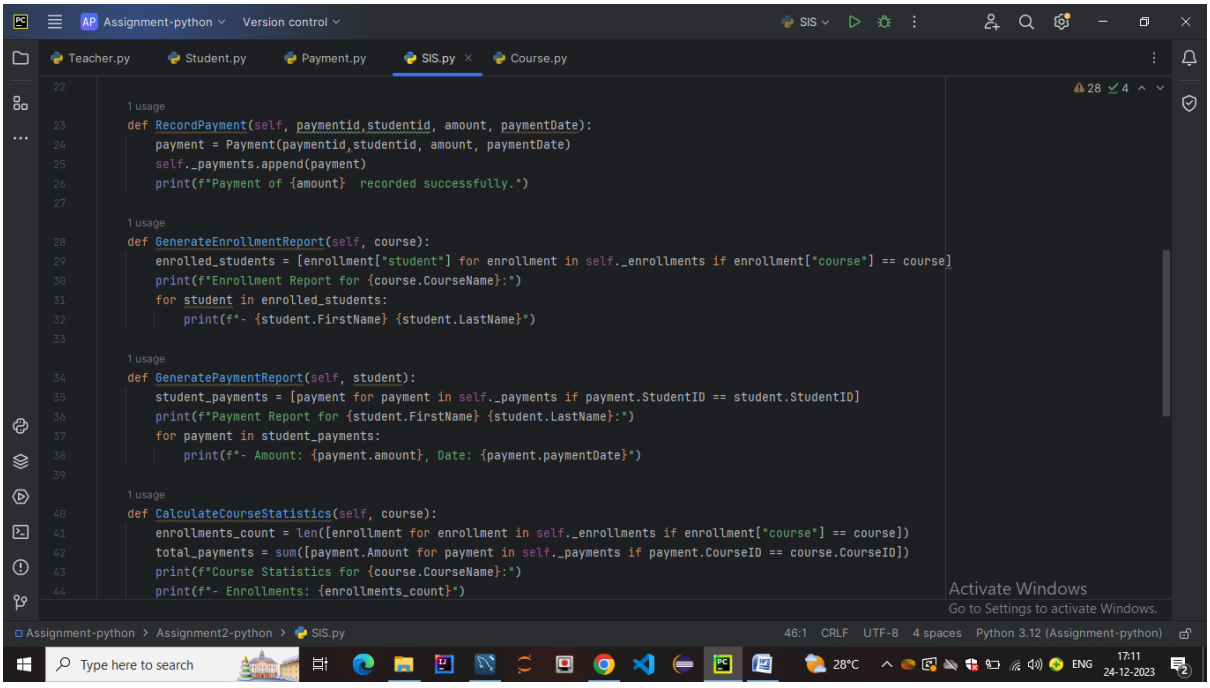
• AssignTeacherToCourse(teacher: Teacher, course: Course): Assigns a teacher to a course.

• 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.

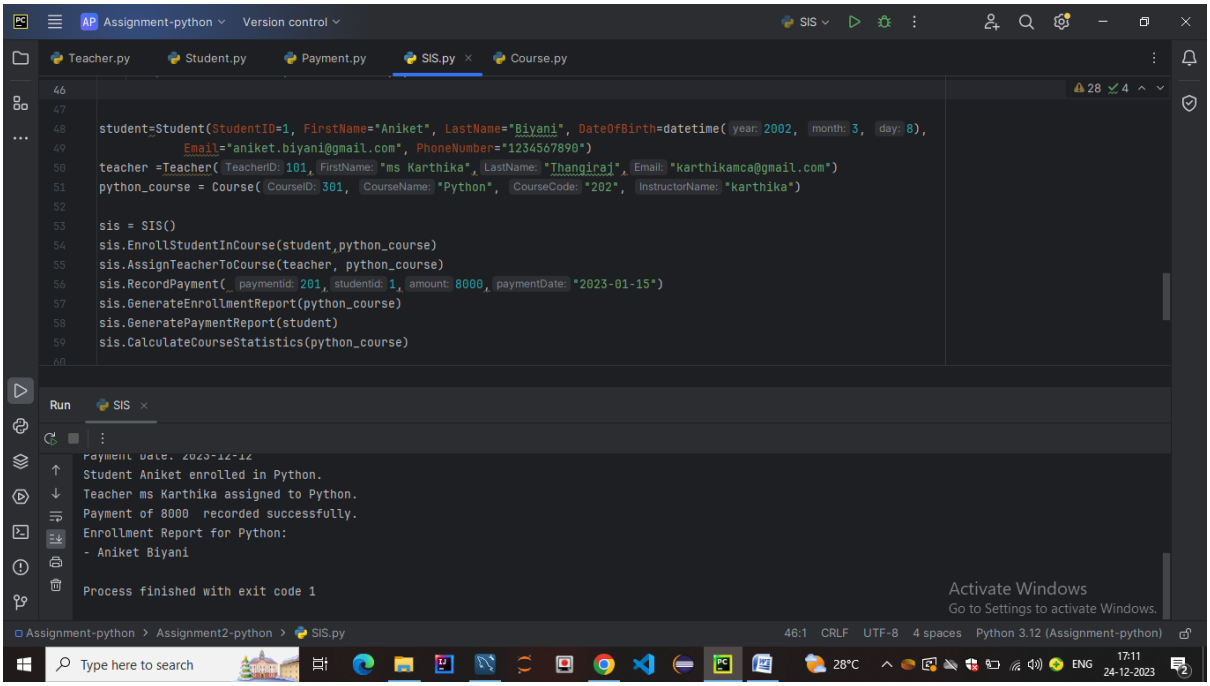


• GeneratePaymentReport(student: Student): Generates a report of payments made by a specific student.



• CalculateCourseStatistics(course: Course): Calculates statistics for a specific course, such as the

number of enrollments and total payments.



Task 4: Exceptions handling and Custom Exceptions

Implementing custom exceptions allows you to define and throw exceptions tailored to specificsituations 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 oneof its derived classes (e.g., System.ApplicationException). These custom exception classes will allow youto encapsulate specific error scenarios and provide meaningful error messages.

Throw Custom ExceptionsIn your code, you can throw custom exceptions when specific conditions or business logic rules areviolated. To throw a custom exception, use the throw keyword followed by an instance of your customexception class.

• DuplicateEnrollmentException: Thrown when a student is already enrolled in a course and triesto enroll again. This exception can be used in the EnrollStudentInCourse method.

• CourseNotFoundException: Thrown when a course does not exist in the system, and you

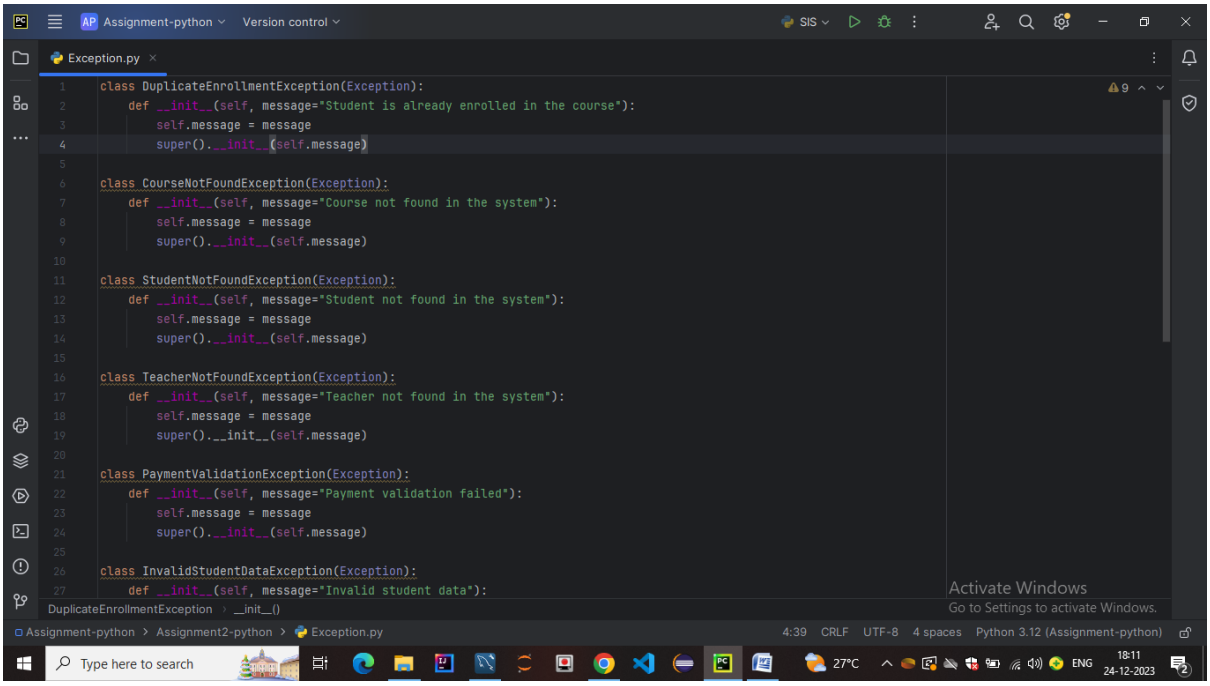
attempt to perform operations on it (e.g., enrolling a student or assigning a teacher).

• 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 asan invalid payment amount or payment date.



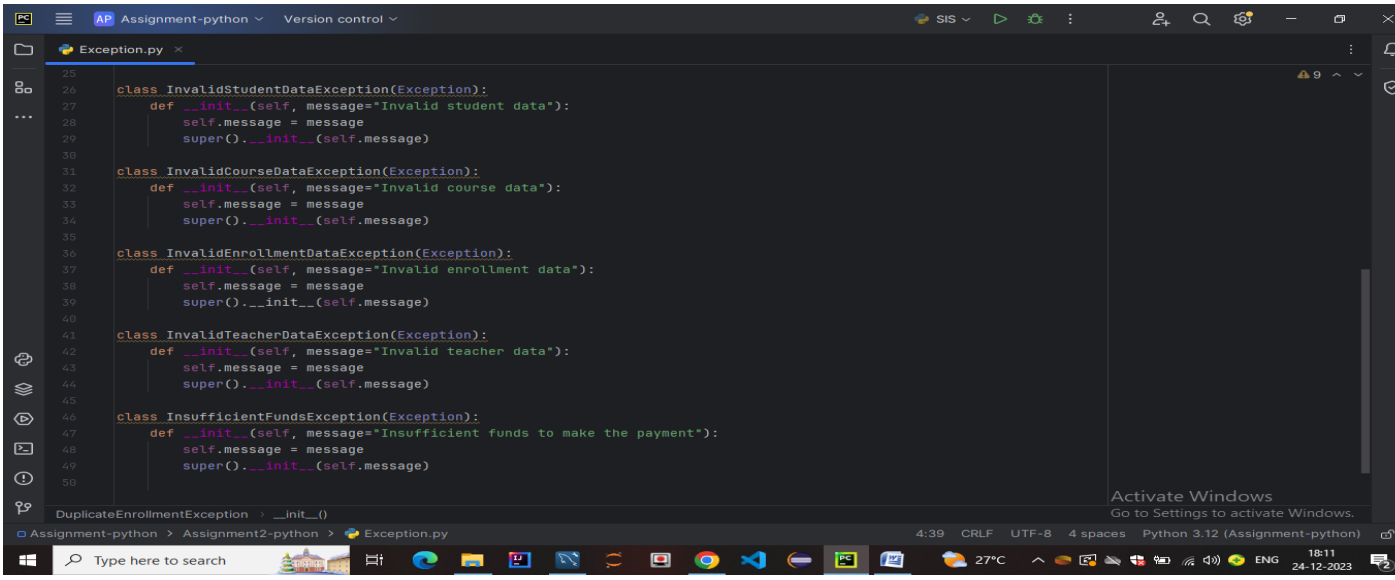
• InvalidStudentDataException: Thrown when data provided for creating or updating a student isinvalid (e.g., invalid date of birth or email format).

• InvalidCourseDataException: Thrown when data provided for creating or updating a course isinvalid (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 isinvalid (e.g., missing name or email).

• InsufficientFundsException: Thrown when a student attempts to enroll in a course but does nothave enough funds to make the payment



Task 5: Collections Implement Collections:

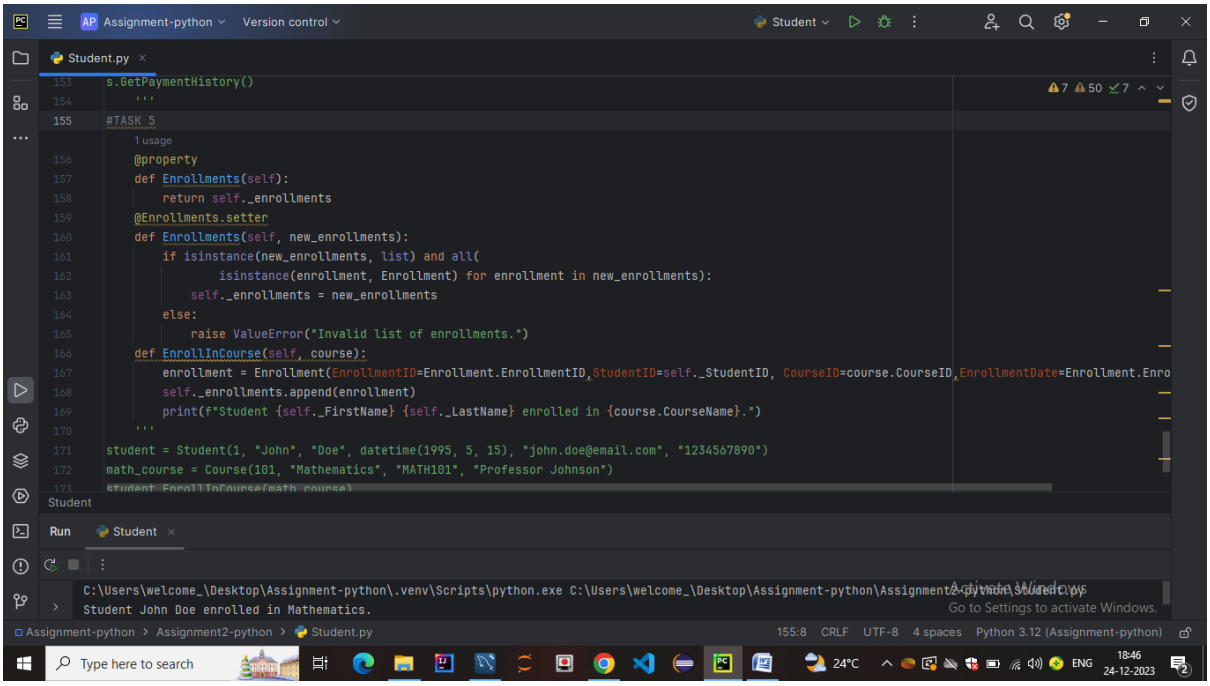
Implement relationships between classes using appropriate data structures (e.g., lists or dictionaries) to maintain associations between students, courses, enrollments, teachers, and payments.

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; }

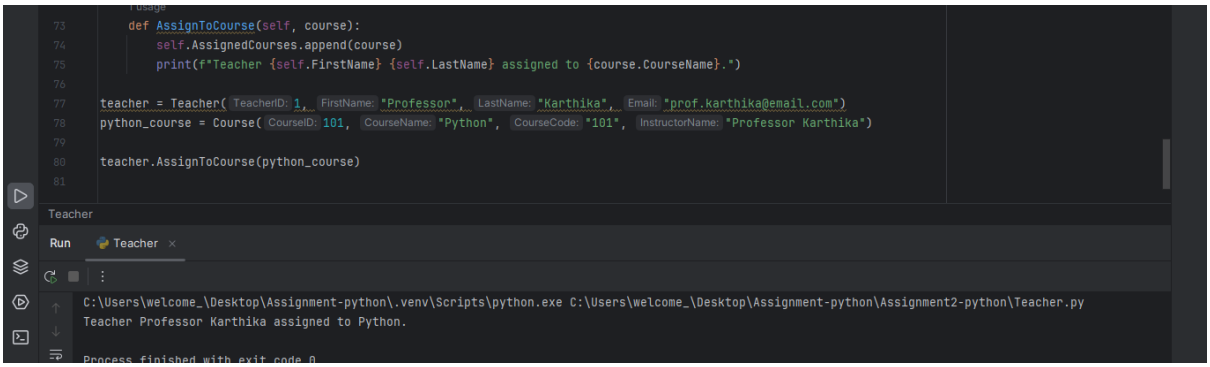
Course Class: Create a list or collection property to store the course's enrollments. This property will hold references to Enrollment objects. Example: List Enrollments { get; set; }

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:

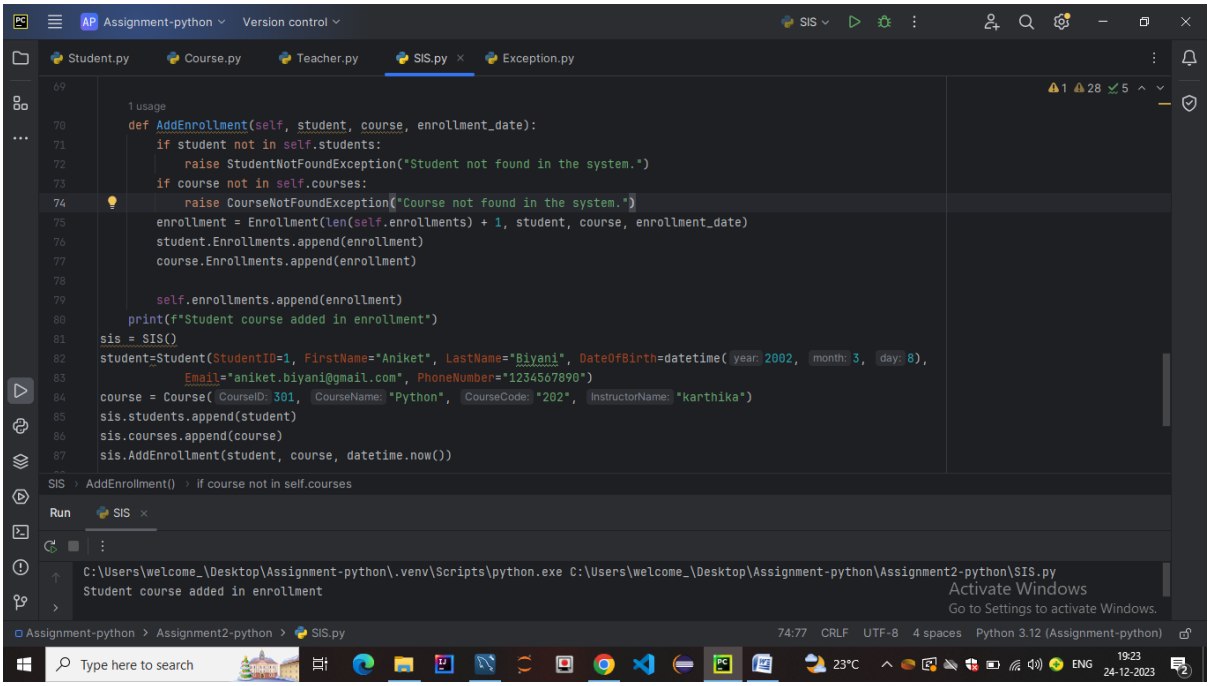
Create a list or collection property to store the teacher's assigned courses. This property will hold references to Course objects. Example: List AssignedCourses { get; set; }



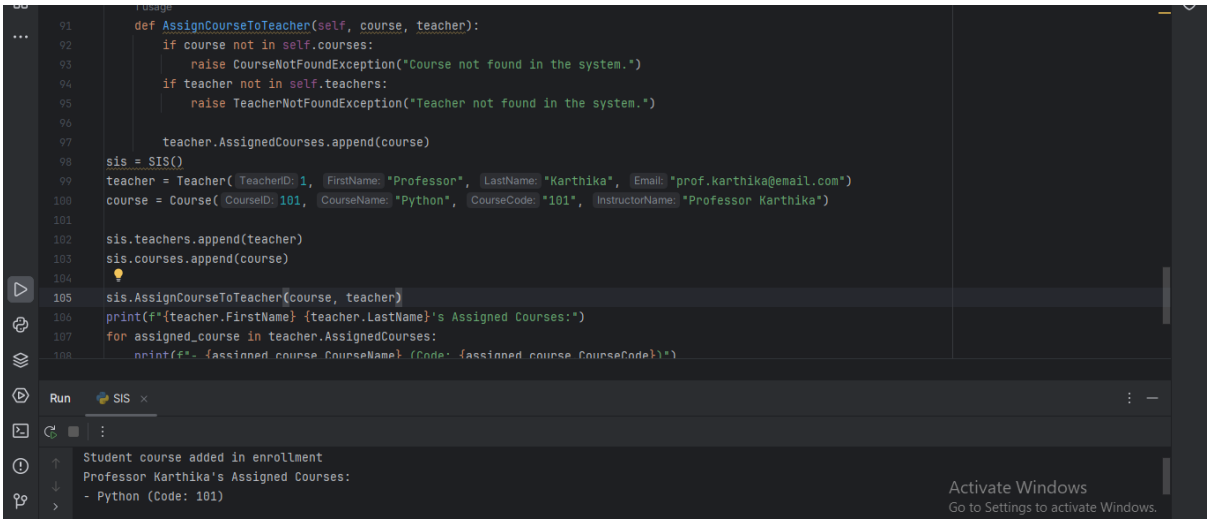
Task 6: Create Methods for Managing Relationships

To add, remove, or retrieve related objects, you should create methods within your SIS class or eachrelevant class.

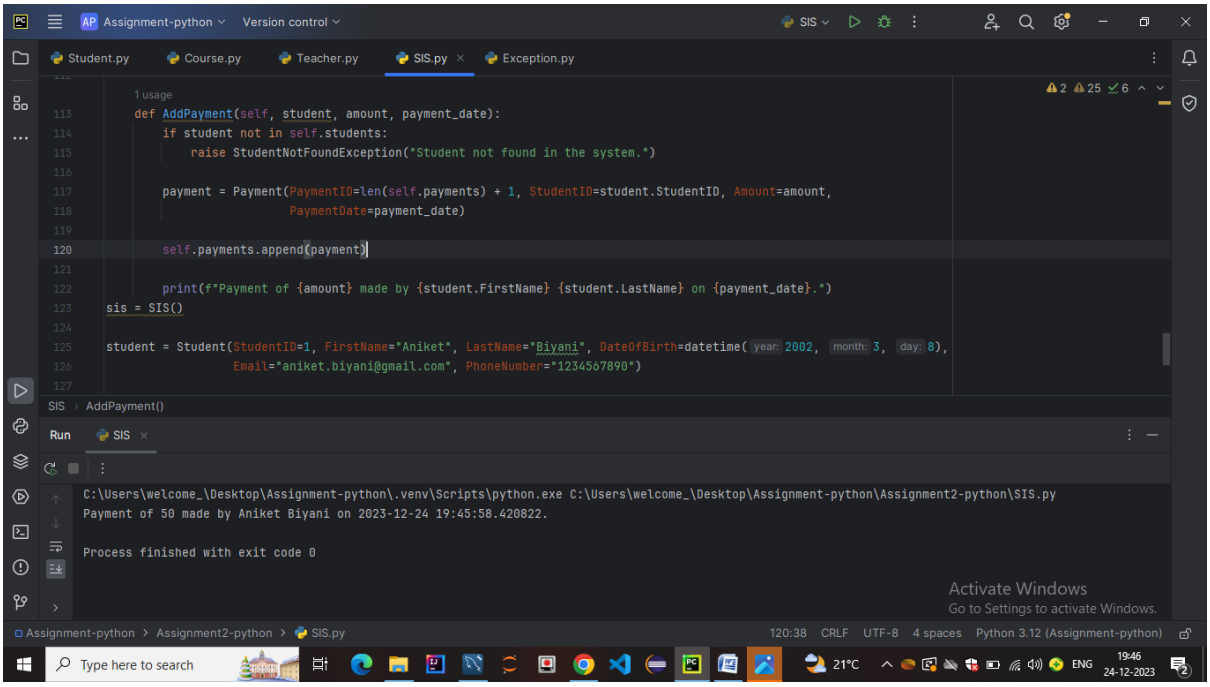
• **AddEnrollment(student, course, enrollmentDate):** In the SIS class, create a method that adds anenrollment to both the Student's and Course's enrollment lists. Ensure the Enrollmentobject references the correct Student and Course.



• **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.

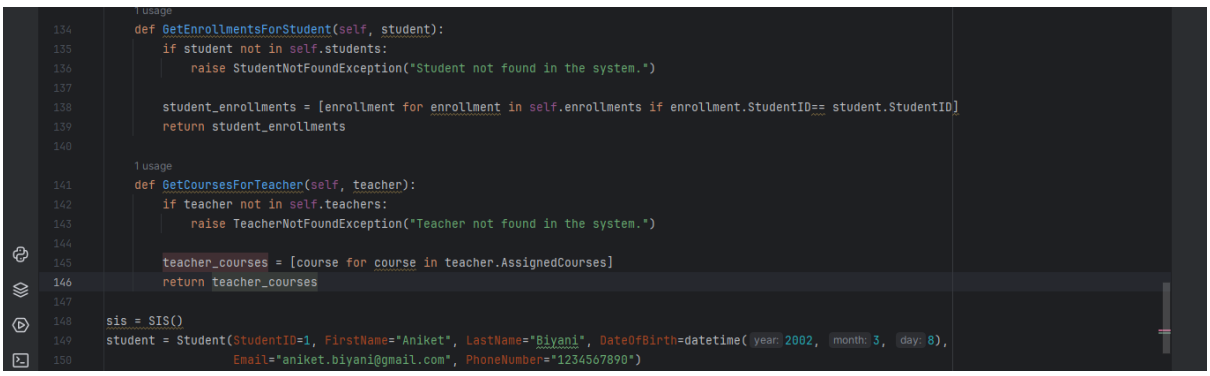


• **AddPayment(student, amount, paymentDate)**: In the SIS class, create a method that adds payment to the Student's payment history. Ensure the Payment object references the correctStudent.

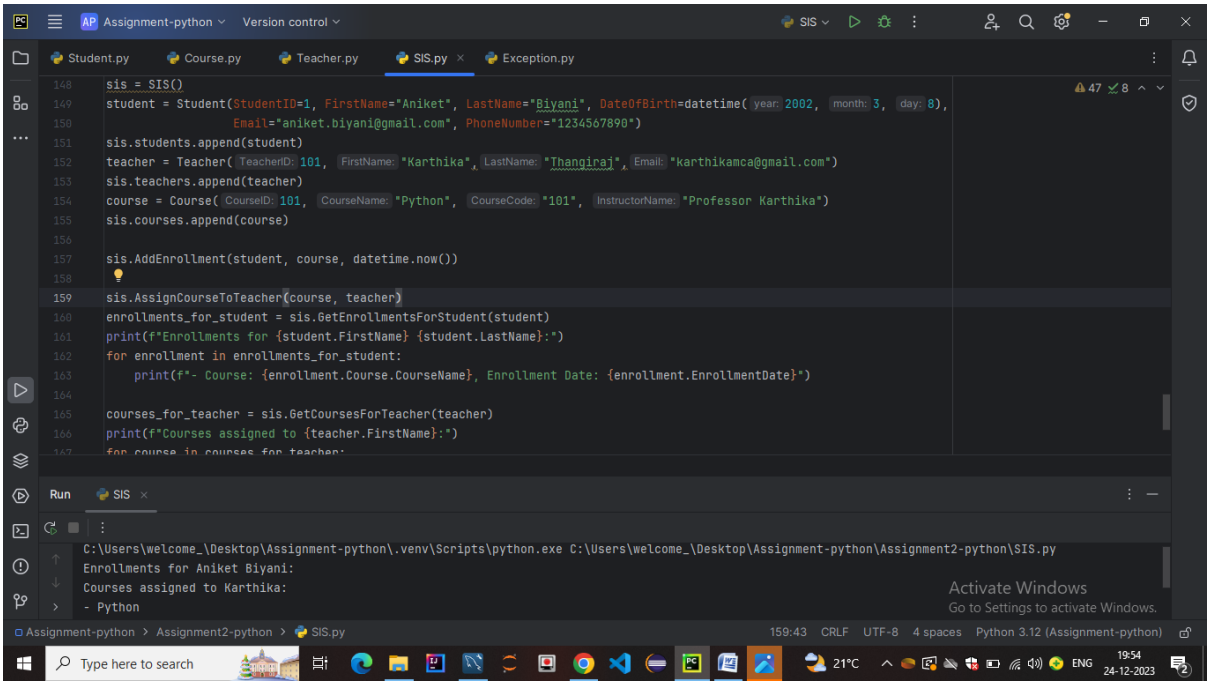


• **GetEnrollmentsForStudent(student):** In the SIS class, create a method to retrieve all enrollmentsfor a specific student.

• **GetCoursesForTeacher(teacher):** In the SIS class, create a method to retrieve all courses assigned to a specific teacher



.



Create a Driver Program

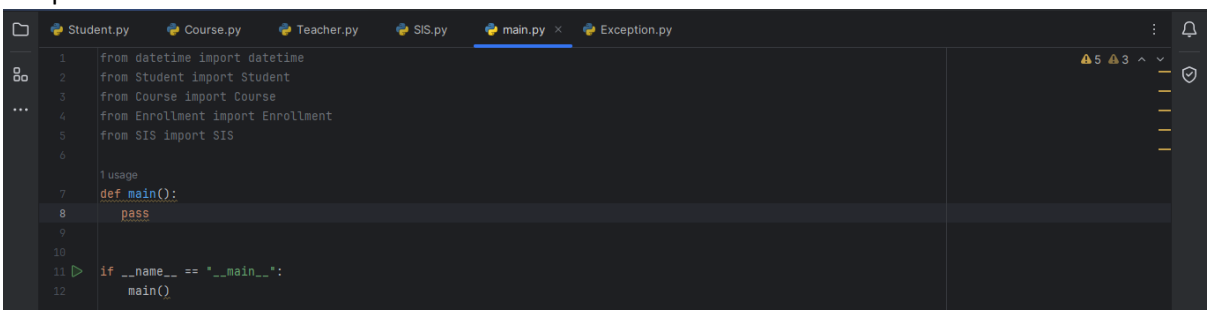
A driver program (also known as a test program or main program) is essential for testing and demonstrating the functionality of your classes and methods within your Student Information System (SIS) assignment. In this task, you will create a console application that serves as the entry point for your 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.



Task 7: Database Connectivity

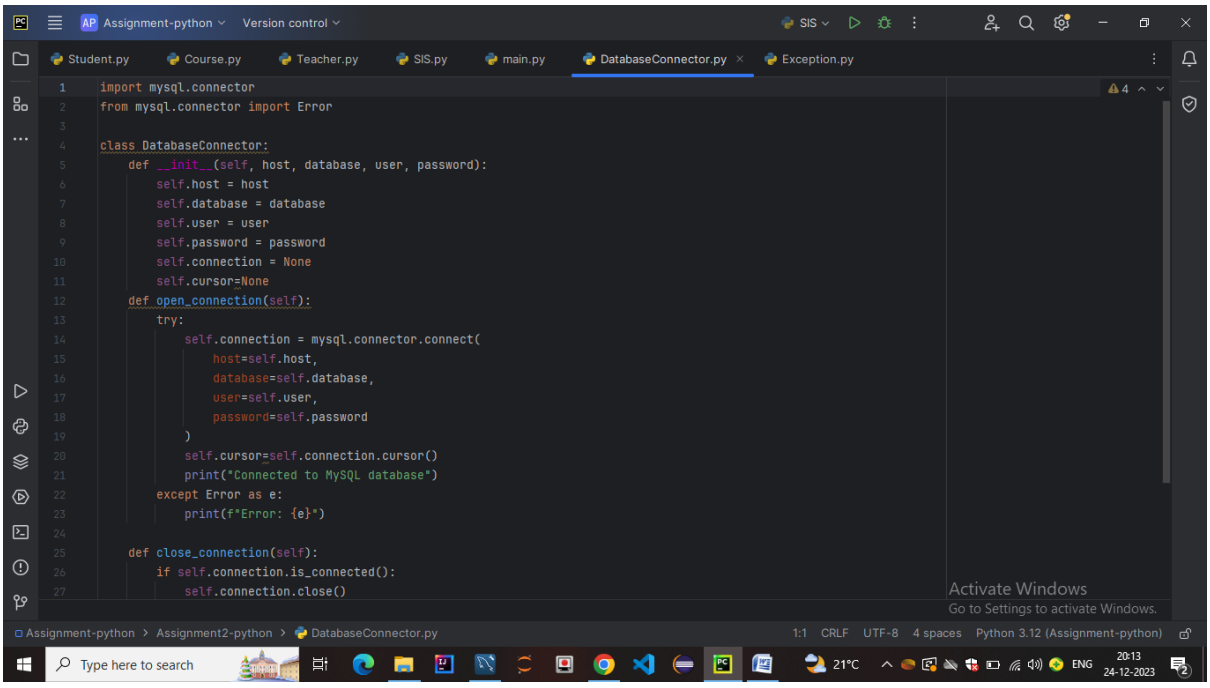
Database Initialization:

Data Retrieval:

Data Insertion and Updating:

Transaction Management:.

Dynamic Query Builder:



Task 8: Student Enrollment

In this task, a new student, John Doe, is enrolling in the SIS. The system needs to record John'sinformation, 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

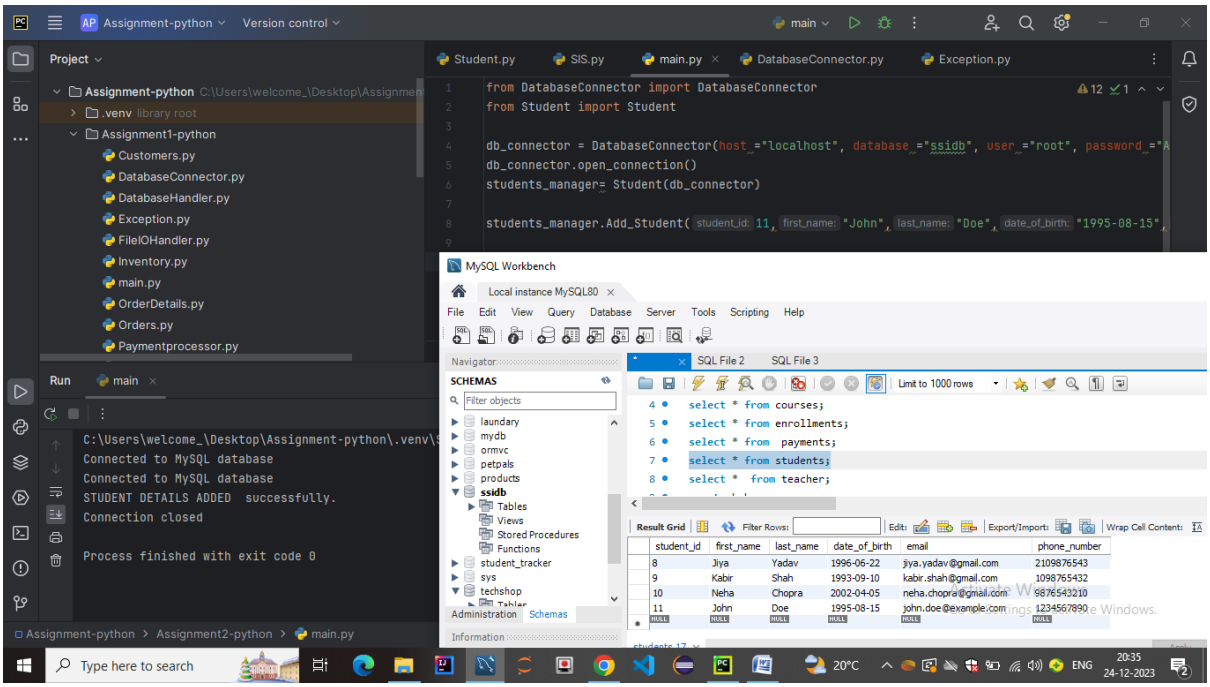
John is enrolling in the following courses:

- Course 1: Introduction to Programming
- Course 2: Mathematics 101

The system should perform the following tasks:

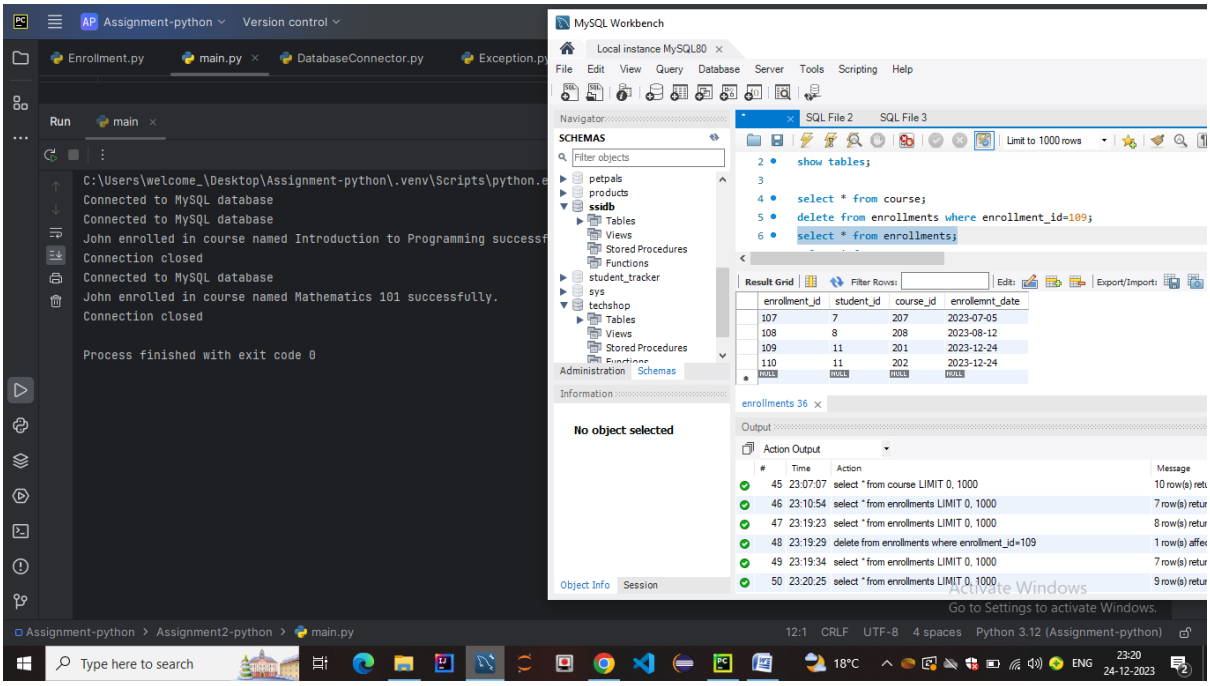
- Create a new student record in the database.

```python
def Add_Student(self,student_id, first_name, last_name, date_of_birth, email, phone_number):
    try:
        self._db_connector.open_connection()

        query = "INSERT INTO students (student_id, first_name, last_name, date_of_birth, email, phone_number) VALUES (%s, %s, %s, %s, %s, %s)"
        values = (student_id, first_name, last_name, date_of_birth, email, phone_number)

        with self._db_connector.connection.cursor() as cursor:
            cursor.execute(query, values)

        self._db_connector.connection.commit()
        print("STUDENT DETAILS ADDED  successfully.")

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

    finally:
        self._db_connector.close_connection()
```



- Enroll John in the specified courses by creating enrollment records in the database.

```python
def Enroll_Course(self, enrollment_id, student_id, course_id):
    try:
        self._db_connector.open_connection()

        query = "INSERT INTO enrollments (enrollment_id, student_id, course_id, enrollemnt_date) VALUES (%s, %s, %s,%s)"
        values = (enrollment_id, student_id, course_id, datetime.now())

        with self._db_connector.connection.cursor() as cursor:
            cursor.execute(query, values)

        course_name_query="SELECT course_name FROM course WHERE course_id=%s"
        values_query=(course_id,)
        with self._db_connector.connection.cursor() as cursor1:
            cursor1.execute(course_name_query, values_query)
            c_name=cursor1.fetchone()
        self._db_connector.connection.commit()
        if c_name:
            print(f"John enrolled in course named {c_name[0]} successfully.")

    except Exception as e:
        print(f"Error enrolling course: {e}")

    finally:
        self._db_connector.close_connection()
```



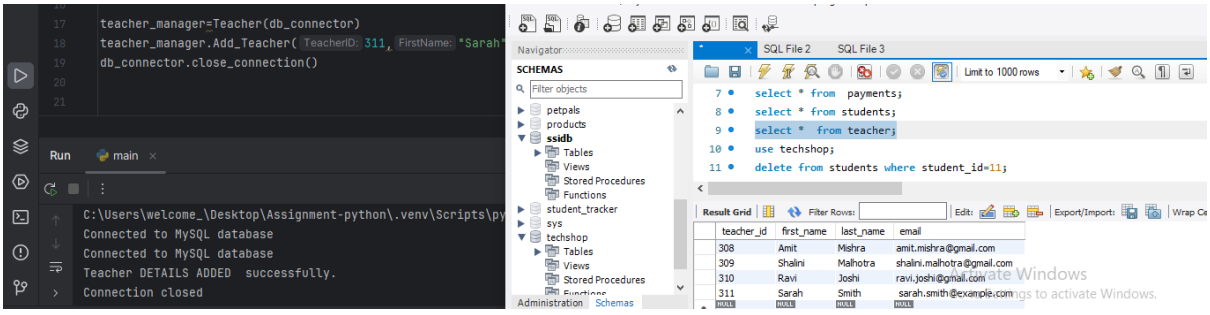
Task 9: Teacher Assignment

In this task, a new teacher, Sarah Smith, is assigned to teach a course. The system needs to update thecourse record to reflect the teacher assignment.

Teacher's Details:

• Name: Sarah Smith• Email: sarah.smith@example.com• Expertise: Computer Science

```
def Add_Teacher(self, TeacherID, FirstName, LastName, Email):
    try:
        self._db_connector.open_connection()

        query = "INSERT INTO teacher ( teacher_id, first_name, last_name, email) VALUES (%s, %s, %s, %s)"
        values = (TeacherID, FirstName, LastName, Email)

        with self._db_connector.connection.cursor() as cursor:
            cursor.execute(query, values)

        self._db_connector.connection.commit()
        print("Teacher DETAILS ADDED  successfully.")

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

    finally:
        self._db_connector.close_connection()
```

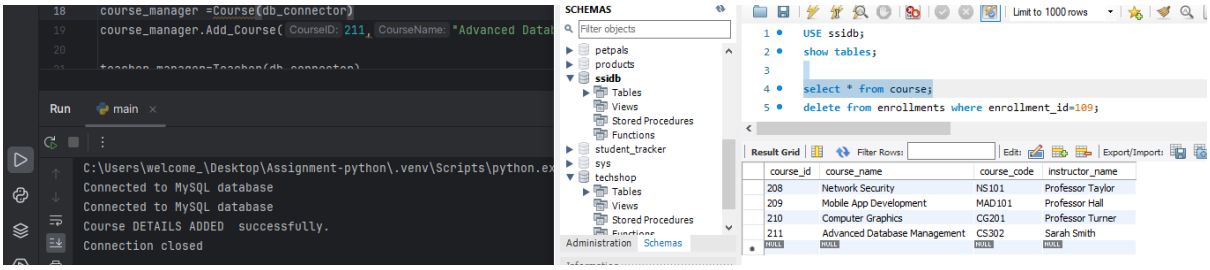


Course to be assigned:

- Course Name: Advanced Database Management
- Course Code: CS302

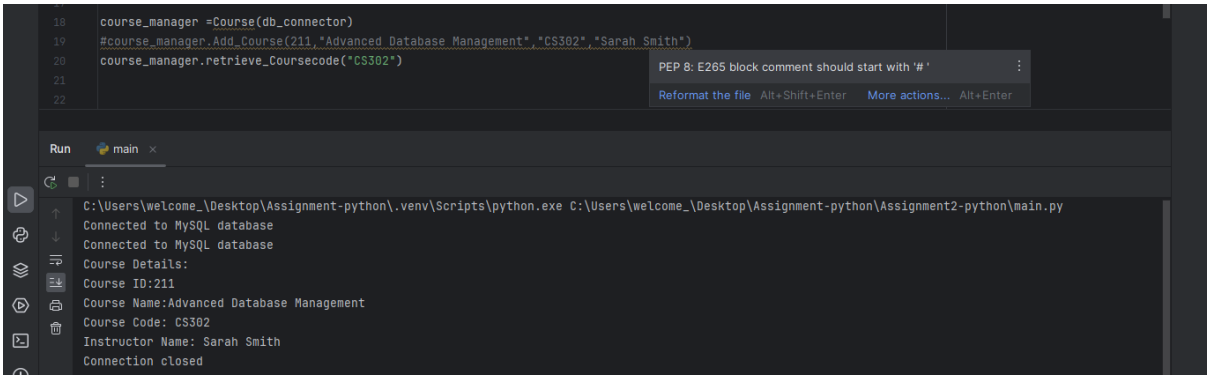
```python
def Add_Course(self, CourseID, CourseName, CourseCode, InstructorName):
    try:
        self._db_connector.open_connection()

        query = "INSERT INTO course ( course_id, course_name, course_code, instructor_name) VALUES (%s, %s, %s, %s)"
        values = (CourseID, CourseName, CourseCode, InstructorName)

        with self._db_connector.connection.cursor() as cursor:
            cursor.execute(query, values)

        self._db_connector.connection.commit()
        print("Course DETAILS ADDED  successfully.")

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

    finally:
        self._db_connector.close_connection()
```

- Assign Sarah Smith as the instructor for the course.



- Retrieve the course record from the database based on the course code.

```python
def retrieve_Coursecode(self,CourseCode):
    try:
        self._db_connector.open_connection()

        query = "SELECT * FROM course where course_code=%s"
        values = (CourseCode, )

        with self._db_connector.connection.cursor() as cursor:
            cursor.execute(query, values)
            course_details = cursor.fetchone()
        if course_details:
            print("Course Details:")
            print(f"Course ID:{course_details[0]}")
            print(f"Course Name:{course_details[1]}")
            print(f"Course Code: {course_details[2]}")
            print(f"Instructor Name: {course_details[3]}")
        else:
            print("coursecode not found.")
        self._db_connector.connection.commit()
    except Exception as e:
        print(f"Error getting course details: {e}")
    finally:
        self._db_connector.close_connection()
```



```python
def update_Course_instructorname(self,InstructorName,CourseID):
    try:
        self._db_connector.open_connection()

        query = "UPDATE Course SET instructor_name=%s where course_id=%s"
        values = (InstructorName,CourseID )

        with self._db_connector.connection.cursor() as cursor:
            cursor.execute(query, values)

        self._db_connector.connection.commit()
        print("Course instructor UPDATED successfully.")

    except Exception as e:
        print(f"Error updating course details: {e}")

    finally:
        self._db_connector.close_connection()
```

- Update the course record in the database with the new instructor information.

```python
course_manager =Course(db_connector)
#course_manager.Add_Course(211,"Advanced Database Management",
#course_manager.retrieve_Coursecode("CS302")
course_manager.update_Course_instructorname( InstructorName: "Prof
```

```
Run  main
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.
Connected to MySQL database
Connected to MySQL database
Course instructor UPDATED successfully.
Connection closed

Process finished with exit code 0
```

File Edit View Query Database Server Tools Scripting Help

Navigator: SCHEMAS

Filter objects

- petpals
- products
- ssidb
  - Tables
  - Views
  - Stored Procedures
  - Functions
- student_tracker
- sys
- techshop
  - Tables
  - Views
  - Stored Procedures
  - Functions
- ticketbookingsystem

Administration Schemas

Information

SQL File 2 SQL File 3

Limit to 1000 rows

```
select * from course;
delete from enrollments where enrollment_id=109;
select * from enrollments;
select * from  payments;
select * from students;
select *  from teacher;
```

Result Grid | Filter Rows: | Edit: | Export/Impo

course_id	course_name	course_code	instructor_name
206	Software Engineering	SE101	Professor Miller
207	Artificial Intelligence	AI201	Professor Wilson
208	Network Security	NS101	Professor Taylor
209	Mobile App Development	MAD101	Professor Hall
210	Computer Graphics	CG201	Professor Turner
211	Advanced Database Management	CS302	Prof karthika

Result Grid | Form Editor

Task 10: Payment Record

In this task, a student, Jane Johnson, makes a payment for her enrolled courses. The system needs torecord this payment in the database.

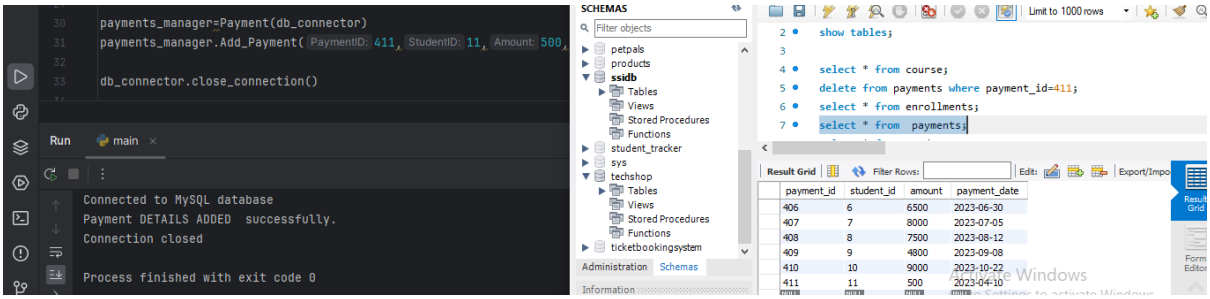
Jane Johnson's details:

- Student ID: 101
- Payment Amount: $500.00
- Payment Date: 2023-04-10

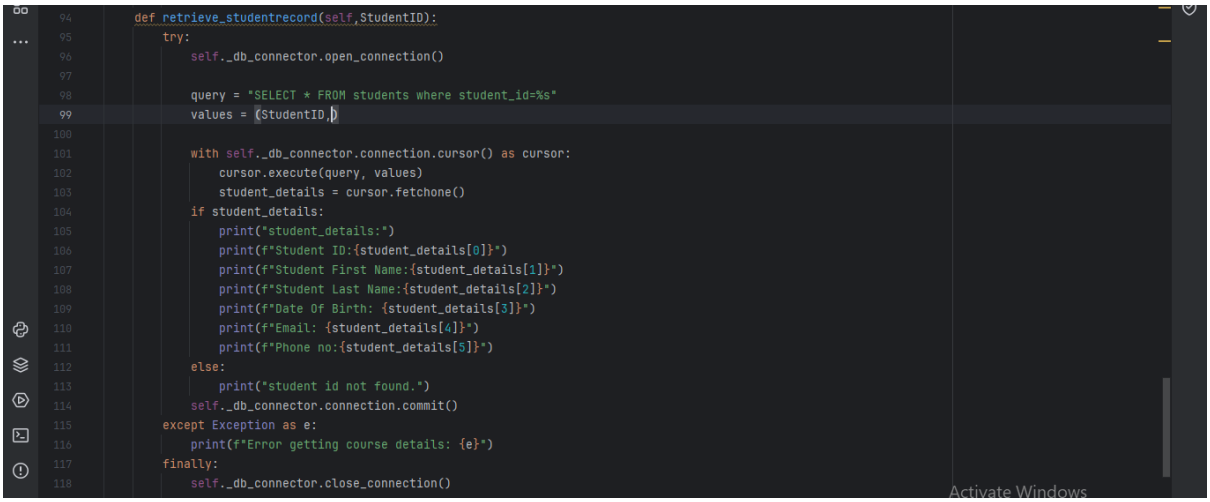
```python
def Add_Payment(self, PaymentID, StudentID, Amount,PaymentDate):
    try:
        self._db_connector.open_connection()

        query = "INSERT INTO payments (payment_id, student_id, amount,payment_date) VALUES (%s, %s, %s, %s)"
        values = (PaymentID, StudentID, Amount,PaymentDate)

        with self._db_connector.connection.cursor() as cursor:
            cursor.execute(query, values)

        self._db_connector.connection.commit()
        print("Payment DETAILS ADDED  successfully.")

    except Exception as e:
        print(f"Error adding payment details: {e}")

    finally:
        self._db_connector.close_connection()
```

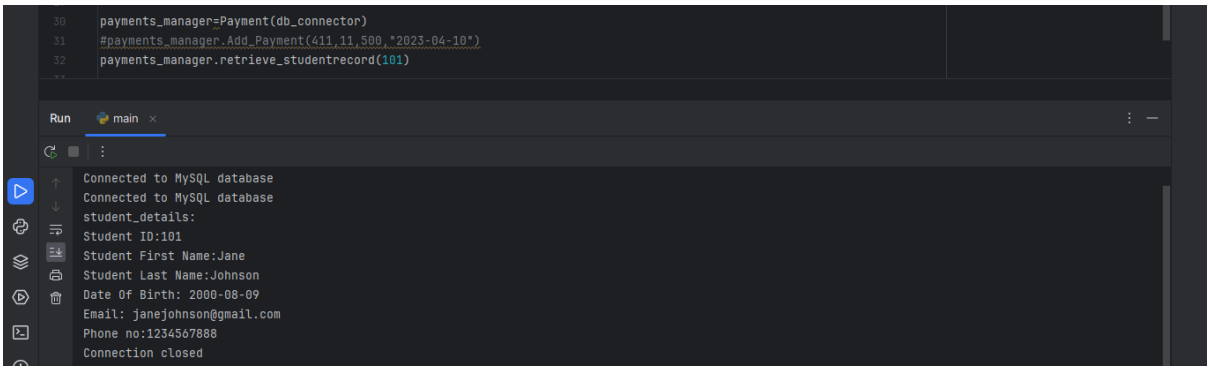
The system should perform the following tasks:

- Record the payment information in the database, associating it with Jane's student record.

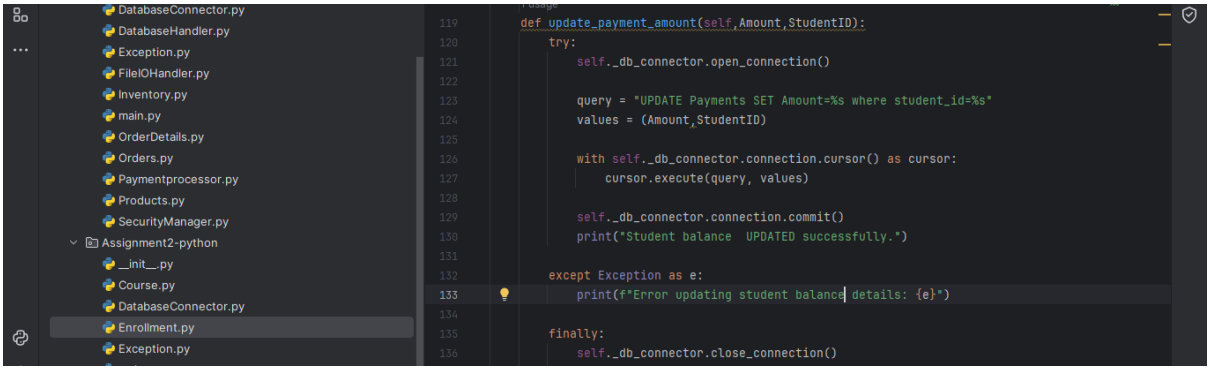


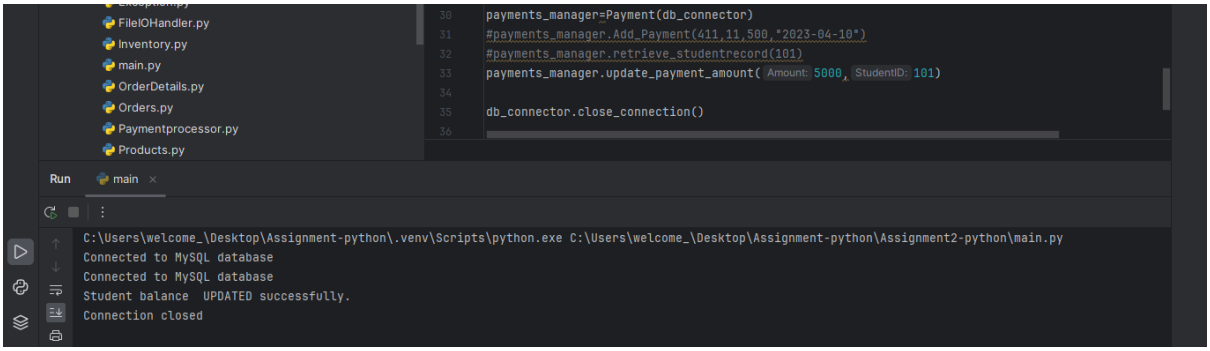
- Retrieve Jane Johnson's student record from the database based on her student ID.





- Update Jane's outstanding balance in the database based on the payment amount.





Task 11: Enrollment Report Generation

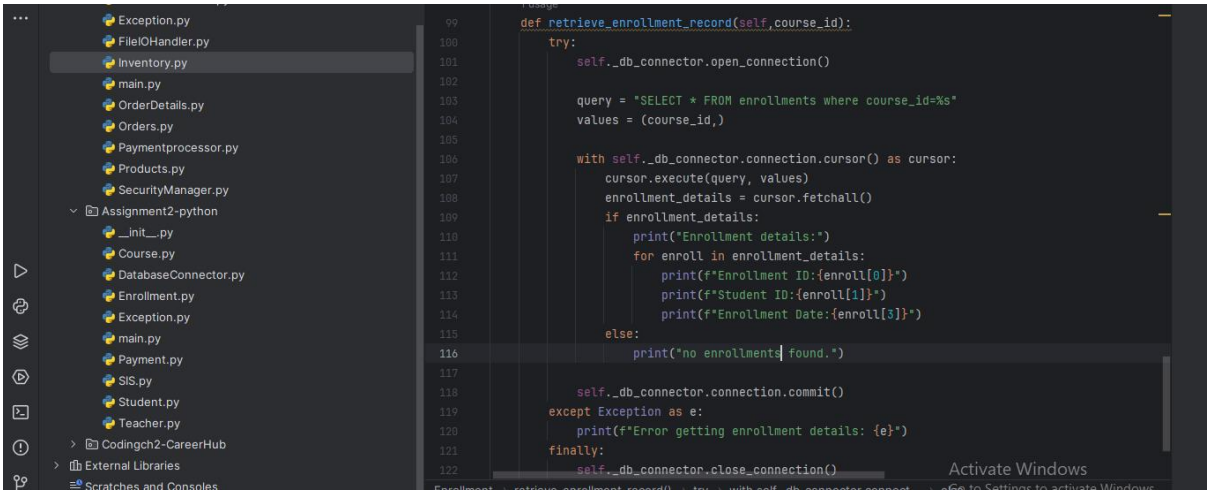
In this task, an administrator requests an enrollment report for a specific course, "Computer Science101." The system needs to retrieve enrollment information from the database and generate a report.

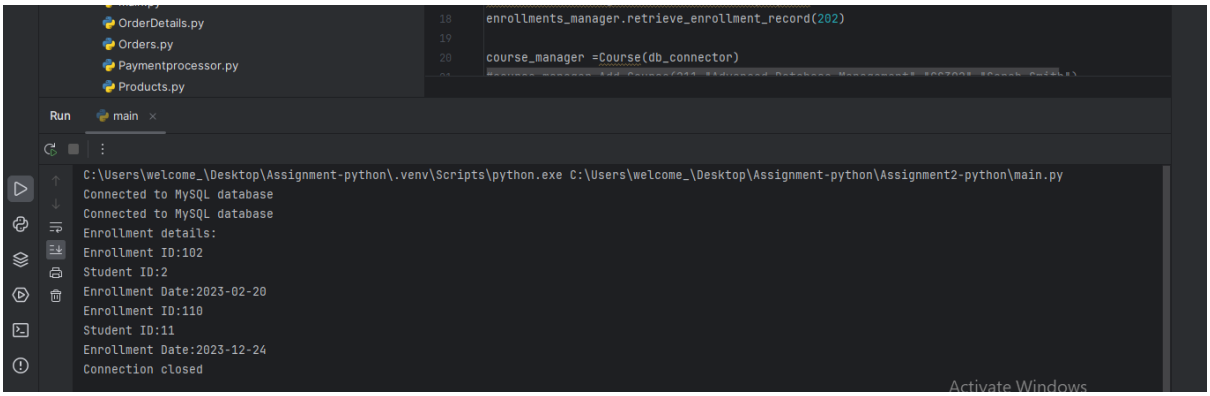
Course to generate the report for:

• Course Name: Computer Science 101

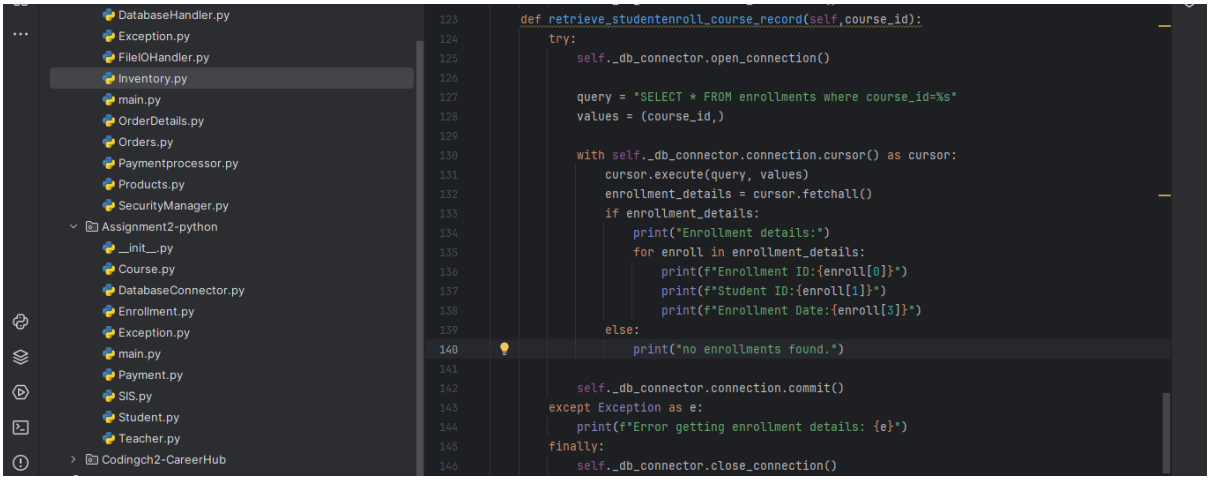
The system should perform the following tasks:

• 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.