

Assignment 2

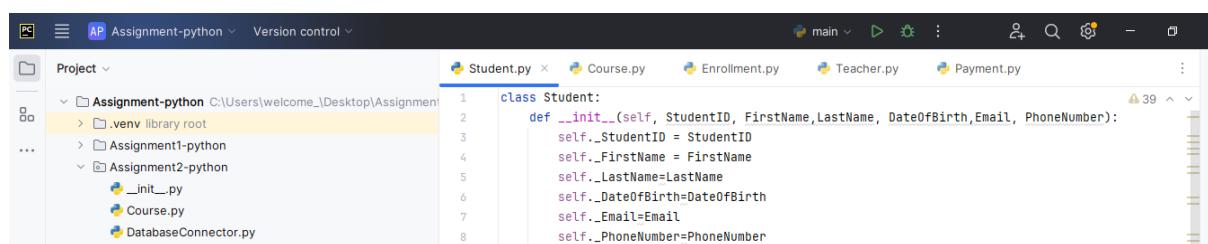
Task 1: Define Classes

Task 2: Implement Constructors

Define the following classes based on the domain description:

Student class with the following attributes:

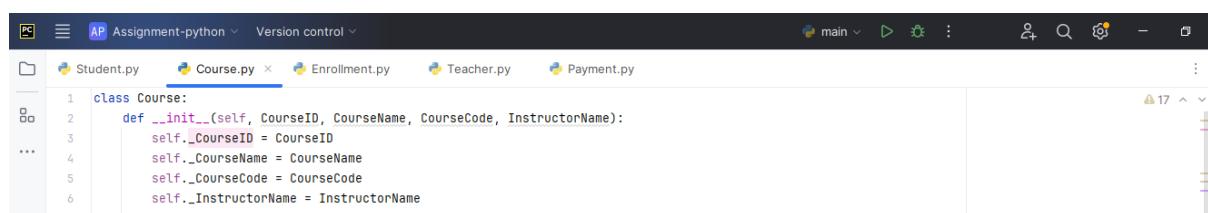
- Student ID
- First Name
- Last Name
- Date of Birth
- Email
- Phone Number



```
1  class Student:  
2      def __init__(self, StudentID, FirstName, LastName, DateOfBirth, Email, PhoneNumber):  
3          self._StudentID = StudentID  
4          self._FirstName = FirstName  
5          self._LastName = LastName  
6          self._DateOfBirth = DateOfBirth  
7          self._Email = Email  
8          self._PhoneNumber = PhoneNumber
```

Course class with the following attributes:

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

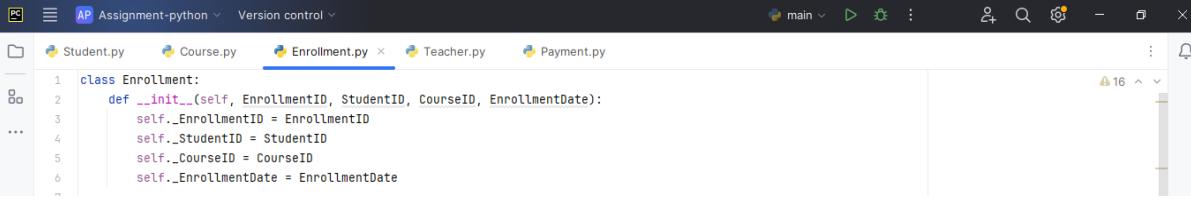


```
1  class Course:  
2      def __init__(self, CourseID, CourseName, CourseCode, InstructorName):  
3          self._CourseID = CourseID  
4          self._CourseName = CourseName  
5          self._CourseCode = CourseCode  
6          self._InstructorName = InstructorName
```

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)

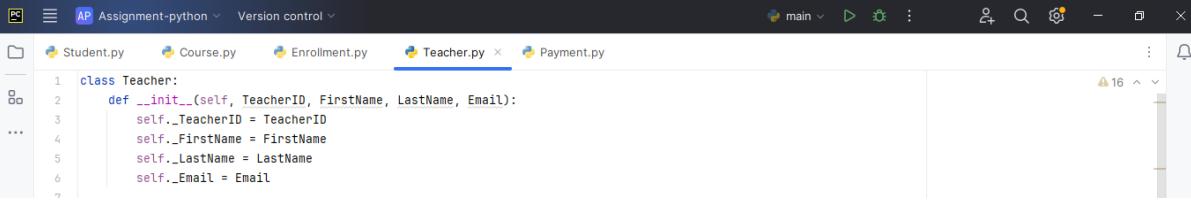
- Enrollment Date



```
1 class Enrollment:
2     def __init__(self, EnrollmentID, StudentID, CourseID, EnrollmentDate):
3         self._EnrollmentID = EnrollmentID
4         self._StudentID = StudentID
5         self._CourseID = CourseID
6         self._EnrollmentDate = EnrollmentDate
```

Teacher class with the following attributes:

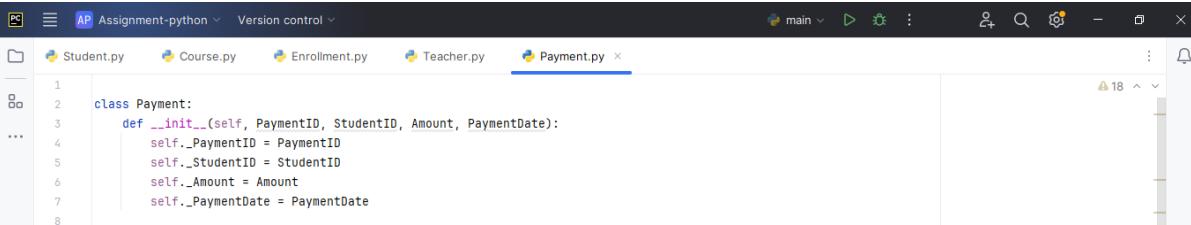
- Teacher ID
- First Name
- Last Name
- Email



```
1 class Teacher:
2     def __init__(self, TeacherID, FirstName, LastName, Email):
3         self._TeacherID = TeacherID
4         self._FirstName = FirstName
5         self._LastName = LastName
6         self._Email = Email
```

Payment class with the following attributes:

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



```
1 class Payment:
2     def __init__(self, PaymentID, StudentID, Amount, PaymentDate):
3         self._PaymentID = PaymentID
4         self._StudentID = StudentID
5         self._Amount = Amount
6         self._PaymentDate = PaymentDate
```

Task 3: Implement Methods

Task 5:

Student Class:

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

```

1 usage
75     def EnrollInCourse(self, course):
76         self._enrolled_courses.append(course)
77         print(f"Student {self.FirstName} {self.LastName} enrolled in {course.CourseName}.")
78
79 s = Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime( year: 2002,
80                                         Email="aniket.biyani@gmail.com", PhoneNumber="1234567890")
81 math_course = Course( CourseID: 101, CourseName: "Mathematics", CourseCode: "MATH101", InstructorName:
82
83 s.EnrollInCourse(math_course)
84
85 ...
86     def UpdateStudentInfo(self, firstName, lastName, dateOfBirth, email, phoneNumber):
87         Student __init__()

```

C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\Student.py
Student Aniket Biyani enrolled in Mathematics.

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

```

1 usage
85     def UpdateStudentInfo(self, firstName, lastName, dateOfBirth, email, phoneNumber):
86         self.FirstName = firstName
87         self.LastName = lastName
88         self.DateOfBirth = dateOfBirth
89         self.Email = email
90         self.PhoneNumber = phoneNumber
91         print("Student information updated successfully.")
92 s = Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime( year: 2002,
93                                         Email="aniket.biyani@gmail.com", PhoneNumber="1234567890")
94 s.UpdateStudentInfo( firstName: "Aniket", lastName: "Biyani", dateOfBirth: (2001, 3, 8),
95                                         email: "aniket.biyani11@gmail.com", phoneNumber: "1234567880")
96
97     def MakePayment(self, amount, paymentDate):
98         payment_record = {"Amount": amount, "PaymentDate": paymentDate}
99         self._payment_history.append(payment_record)
100        print(f"Payment of {amount} made on {paymentDate} recorded successfully.")

```

PEP 8: E128 continuation line under-indented for visual indent
Reformat the file Alt+Shift+Enter More actions... Alt+Enter

C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\Student.py
Student information updated successfully.
Process finished with exit code 0

- **MakePayment(amount: decimal, paymentDate: DateTime):** Records a payment made by the student.

```

1 usage
100    def MakePayment(self, amount, paymentDate):
101        payment_record = {"Amount": amount, "PaymentDate": paymentDate}
102        self._payment_history.append(payment_record)
103        print(f"Payment of {amount} made on {paymentDate} recorded successfully.")
104
105 s = Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime( year: 2002,
106                                         Email="aniket.biyani@gmail.com", PhoneNumber="1234567890")
107 s.MakePayment( amount: 50, paymentDate: "12-12-2023")
108
109     def DisplayStudentInfo(self):
110         print(f"Student ID: {self.StudentID}")

```

C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\Student.py
Payment of 50 made on 12-12-2023 recorded successfully.

- **DisplayStudentInfo():** Displays detailed information about the student.

```

    Customers.py
    DatabaseConnector.py
    DatabaseHandler.py
    Exception.py
    FileIOHandler.py
    Inventory.py
    main.py
    OrderDetails.py
    Orders.py
    Paymentprocessor.py
    Products.py
    SecurityManager.py

111     def DisplayStudentInfo(self):
112         print(f"Student ID: {self.StudentID}")
113         print(f"Name: {self.FirstName} {self.LastName}")
114         print(f"Date of Birth: {self.DateOfBirth}")
115         print(f"Email: {self.Email}")
116         print(f"Phone Number: {self.PhoneNumber}")
117
118     s = Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime( year: 2002,
119                                         Email="aniket.biyani@gmail.com", PhoneNumber="1234567890")
120     s.DisplayStudentInfo()
121
122     def GetEnrolledCourses(self):
123         return self._enrolled_courses

```

Run Student

```

C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\Student.py
Student ID: 1
Name: Aniket Biyani
Date of Birth: 2002-03-08 00:00:00
Email: aniket.biyani@gmail.com
Phone Number: 1234567890

```

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

```

    ...
    Assignment1-python
    Customers.py
    DatabaseConnector.py
    DatabaseHandler.py
    Exception.py
    FileIOHandler.py
    Inventory.py
    main.py
    OrderDetails.py
    Orders.py
    Paymentprocessor.py
    Products.py
    SecurityManager.py

124     def GetEnrolledCourses(self):
125         return self._enrolled_courses
126
127
128     s= Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime( year: 2002,
129                                         Email="aniket.biyani@gmail.com", PhoneNumber="1234567890")
130     math_course = Course( CourseID: 101, CourseName: "Mathematics", CourseCode: "MATH101", InstructorName:
131
132     s.EnrollInCourse(math_course)
133     s.GetEnrolledCourses()
134
135     GetPaymentHistory()

```

Run Student

```

C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\Student.py
Student Aniket Biyani enrolled in Mathematics.

```

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

```

    DatabaseHandler.py
    Exception.py
    FileIOHandler.py
    Inventory.py
    main.py
    OrderDetails.py
    Orders.py
    Paymentprocessor.py
    Products.py
    SecurityManager.py

136     def GetPaymentHistory(self):
137         return self._payment_history
138
139     s = Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime( year: 2002,
140                                         Email="aniket.biyani@gmail.com", PhoneNumber="1234567890")
141     s.MakePayment( amount: 50, paymentDate: "12-12-2023")
142     s.GetPaymentHistory()

```

Run Student

```

C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\Student.py
Payment of 50 made on 12-12-2023 recorded successfully.

```

Course Class:

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

```

    Exception.py
    FileIOHandler.py
    Inventory.py
    main.py
    OrderDetails.py
    Orders.py
    Paymentprocessor.py
    Products.py
    SecurityManager.py
    Assignment1-python
    Assignment2-python

54     def AssignTeacher(self, teacher):
55         self._assigned_teacher = teacher
56         print(f"Teacher {teacher.FirstName} {teacher.LastName} assigned to the course {self.CourseName}")
57
58     python_teacher = Teacher( TeacherID: 1, FirstName: "Karthika", LastName: "Thangiraj", Email: "karthikamca
59
60     python_course = Course( CourseID: 101, CourseName: "Python", CourseCode: 202, InstructorName: "Karthika"
61
62     python_course.AssignTeacher(python_teacher)

Course > AssignTeacher()

```

Run Course

```

C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\Course.py
Teacher Karthika Thangiraj assigned to the course Python.

Process finished with exit code 0

```

- `UpdateCourseInfo(courseCode: string, courseName: string, instructor: string)`: Updates course information.

```

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")

```

The screenshot shows a code editor window with a Python script named `Course.py`. The `UpdateCourseInfo` method is defined, taking parameters `courseId`, `courseName`, `courseCode`, and `instructor`. It prints a success message. An instance of the `Course` class is created with `CourseID: 101`, `CourseName: "Python"`, `CourseCode: "202"`, and `InstructorName: "karthika"`. This instance's `UpdateCourseInfo` method is then called with `courseId: 101`, `courseName: "Python+Pyunit"`, `courseCode: "102"`, and `instructor: "Prof.Karthika"`. The output in the terminal shows the message "Course information updated successfully." followed by "Process finished with exit code 0".

- `DisplayCourseInfo()`: Displays detailed information about the course.

```

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()

```

The screenshot shows a code editor window with the `DisplayCourseInfo` method defined, which prints the course ID, course code, course name, and instructor name. An instance of the `Course` class is created with `CourseID: 101`, `CourseName: "Python"`, `CourseCode: "202"`, and `InstructorName: "karthika"`. The `DisplayCourseInfo` method is called on this instance. The output in the terminal shows the course details: "Course ID: 101", "Course Code: 202", "Course Name: Python", and "Instructor: karthika". It also shows "Process finished with exit code 0". A tooltip for PEP 8: E305 is visible, stating "expected 2 blank lines after class or function definition, found 1".

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

```

88     def GetEnrollments(self):
89         return self._enrollments
...
91     student_1 = Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime(year: 2002, month: 8, day: 3), Email="aniket@email.com", PhoneNum
92     student_2 = Student(StudentID=2, FirstName="Abhi", LastName="Surya", DateOfBirth=datetime(year: 2001, month: 2, day: 2), Email="abhi@email.com", PhoneNum
93
94     python_course = Course(CourseID: 101, CourseName: "Python", CourseCode: "202", InstructorName: "karthika")
95     student_1.EnrollInCourse(python_course)
96     student_2.EnrollInCourse(python_course)
97
98     enrollments = python_course.GetEnrollments()
99
100    print("Student Enrollments:")
101    for enrollment in enrollments:
102        print(f"- Student ID: {enrollment.StudentID}, Student Name: {enrollment.FirstName} {enrollment.LastName}")

```

The screenshot shows a code editor window with the `GetEnrollments` method defined, which returns a list of student enrollments. Two student objects, `student_1` and `student_2`, are created with their respective details and enrolled in the `python_course`. The `GetEnrollments` method is called on `python_course`. The output in the terminal shows the student enrollments: "Student Aniket Biyani enrolled in Python.", "Student Abhi Surya enrolled in Python.", and "Student Enrollments:". A tooltip for PEP 8: E305 is visible, stating "expected 2 blank lines after class or function definition, found 1".

- **GetTeacher():** Retrieves the assigned teacher for the course.

```

1 usage
105     def GetTeacher(self):
106         return self._assigned_teacher
107
108 python_teacher = Teacher( TeacherID: 1, FirstName: "Karthika", LastName: "Thangiraj", Email: "karthikamca@gmail.com")
109
110 python_course = Course( CourseID: 101, CourseName: "Python", CourseCode: 202, InstructorName: "karthika")
111
112 python_course.AssignTeacher(python_teacher)
113
114 assigned_teacher = python_course.GetTeacher()

```

The screenshot shows a Python code editor with the following code. The code defines a `Course` class and a `Teacher` class. It creates a `python_course` object and a `python_teacher` object. It then assigns the teacher to the course and retrieves the assigned teacher using the `GetTeacher()` method. The output window shows the result: "Teacher Karthika Thangiraj assigned to the course Python."

Enrollment Class:

- **GetStudent():** Retrieves the student associated with the enrollment.

```

...
58 student_1 = Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime( year: 2002, month: 8, day: 3), Email="aniket@email.com", PhoneNumber="9876543210")
59 student_2 = Student(StudentID=2, FirstName="Abhi", LastName="Surya", DateOfBirth=datetime( year: 2001, month: 2, day: 2), Email="abhi@email.com", PhoneNumber="9876543211")
60
61 python_course = Course( CourseID: 101, CourseName: "Python", CourseCode: "202", InstructorName: "karthika")
62 student_1.EnrollInCourse(python_course)
63 student_2.EnrollInCourse(python_course)
64
65 enrollments= python_course.GetEnrollments()
66
67 print("Enrolled Students:")
68 for student in enrollments:
69     print(f"- Student ID: {student.StudentID}, Student Name: {student.FirstName} {student.LastName}")

```

The screenshot shows a Python code editor with the following code. It defines a `Student` class and a `Course` class. It creates two student objects and enrolls them in a course. Then it retrieves the enrollments and prints the student details. The output window shows the result: "Student Aniket Biyani enrolled in Python. Student Abhi Surya enrolled in Python. Enrolled Students:"

Teacher Class:

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

```

1 usage
52     def UpdateTeacherInfo(self, teacherID, firstName, lastName, email):
53         self.TeacherID=teacherID
54         self.FirstName=firstName
55         self.LastName=lastName
56         self.Email=email
57         print("Teacher information updated successfully.")
58 teacher_info = Teacher( TeacherID: 101, FirstName: "Aniket", LastName: "Biyani", Email: "aniketbiyani@gmail.com")
59 teacher_info.UpdateTeacherInfo( teacherID: 101, firstName: "ms Karthika", lastName: "Thangiraj", email: "karthikamca@gmail.com")

```

The screenshot shows a Python code editor with the following code. It defines a `Teacher` class. It creates a `teacher_info` object and updates its information using the `UpdateTeacherInfo()` method. The output window shows the result: "Teacher information updated successfully."

- **DisplayTeacherInfo():** Displays detailed information about the teacher.



```
63     def DisplayTeacherInfo(self):
64         print(f"Teacher ID: {self.TeacherID}")
65         print(f"First Name Code: {self.FirstName}")
66         print(f"Last Name: {self.LastName}")
67         print(f"Email : {self.Email}")
68     teacher = Teacher( TeacherID: 101, FirstName: "Aniket", LastName: "Biyani", Email: "aniketbiyani@gmail.com")
69     teacher.DisplayTeacherInfo()
70
```

Teacher > DisplayTeacherInfo()

Run Teacher ×

C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\Teacher.py

Teacher ID: 101

First Name Code: Aniket

Last Name: Biyani

Email : aniketbiyani@gmail.com

Payment Class:

- `GetStudent()`: Retrieves the student associated with the payment.
 - `GetPaymentAmount()`: Retrieves the payment amount.
 - `GetPaymentDate()`: Retrieves the payment date.

The screenshot shows the Microsoft Visual Studio Code interface. The top bar displays the project name "Assignment-python" and the file "Payment.py". The main editor area contains the following Python code:

```
def GetPaymentAmount(self):
    return self.Amount

    Usage
def GetPaymentDate(self):
    return self.PaymentDate

Payment = Payment( PaymentID=101, StudentID=1, Amount=50, PaymentDate="2023-12-12")

# Retrieve payment amount and date
payment_amount = payment.GetPaymentAmount()
payment_date = payment.GetPaymentDate()

# Display payment information
print(f"Payment Amount: {payment_amount}")
print(f"Payment Date: {payment_date}")
```

The bottom panel shows the "Run" tab with the command "Payment" selected. The output window displays the results of running the script:

```
C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\Payment.py
Payment Amount: 50
Payment Date: 2023-12-12
Process finished with exit code 0
```

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.

A screenshot of a code editor window titled "Assignment-python". The current file is "SIS.py". The code defines a class "SIS" with methods for enrollment, teacher assignment, payment recording, and report generation. The interface includes a sidebar with icons for file operations, a status bar at the bottom, and a taskbar at the bottom.

```
6     class SIS:
7         def __init__(self):
8             self._enrollments = []
9             self._teachers_assigned_courses = []
10            self._payments = []
11
12        1 usage
13        def EnrollStudentInCourse(self, student, course):
14            enrollment = {"student": student, "course": course}
15            self._enrollments.append(enrollment)
16            print(f"Student {student.FirstName} enrolled in {course.CourseName}.")
17
18        1 usage
19        def AssignTeacherToCourse(self, teacher, course):
20            teacher_assignment = {"teacher": teacher, "course": course}
21            self._teachers_assigned_courses.append(teacher_assignment)
22            print(f"Teacher {teacher.FirstName} assigned to {course.CourseName}.")
23
24        1 usage
25        def RecordPayment(self, paymentid, studentid, amount, paymentDate):
26            payment = Payment(paymentid, studentid, amount, paymentDate)
27            self._payments.append(payment)
28            print(f"Payment of {amount} recorded successfully.")
29
30        1 usage
31        def GenerateEnrollmentReport(self, course):
32
33
34
35
36
37
38
39
40
41
42
43
44
```

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

A screenshot of a code editor window titled "Assignment-python". The current file is "SIS.py". The code defines methods for payment recording, enrollment reporting, payment reporting, and course statistics calculation. The interface includes a sidebar with icons for file operations, a status bar at the bottom, and a taskbar at the bottom.

```
22
23    1 usage
24    def RecordPayment(self, paymentid, studentid, amount, paymentDate):
25        payment = Payment(paymentid, studentid, amount, paymentDate)
26        self._payments.append(payment)
27        print(f"Payment of {amount} recorded successfully.")
28
29    1 usage
30    def GenerateEnrollmentReport(self, course):
31        enrolled_students = [enrollment["student"] for enrollment in self._enrollments if enrollment["course"] == course]
32        print(f"Enrollment Report for {course.CourseName}:")
33        for student in enrolled_students:
34            print(f"- {student.FirstName} {student.LastName}")
35
36
37
38
39
40    1 usage
41    def GeneratePaymentReport(self, student):
42        student_payments = [payment for payment in self._payments if payment.StudentID == student.StudentID]
43        print(f"Payment Report for {student.FirstName} {student.LastName}:")
44        for payment in student_payments:
45            print(f"- Amount: {payment.amount}, Date: {payment.paymentDate}")
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
```

- **CalculateCourseStatistics(course: Course):** Calculates statistics for a specific course, such as the number of enrollments and total payments.

The screenshot shows a Python development environment with several files open: Teacher.py, Student.py, Payment.py, SIS.py (the current file), and Course.py. The SIS.py file contains code for managing student enrollment, teacher assignments, and payments. The Run tab shows the output of the script execution:

```

Run SIS ×
:
↑ Payment Date: 2023-12-14
↑ Student Aniket enrolled in Python.
↓ Teacher ms Karthika assigned to Python.
Payment of 8000 recorded successfully.
Enrollment Report for Python:
- Aniket Biyani
Process finished with exit code 1

```

The terminal at the bottom shows standard Windows interface elements like the taskbar and system tray.

Task 4: Exceptions handling and Custom Exceptions

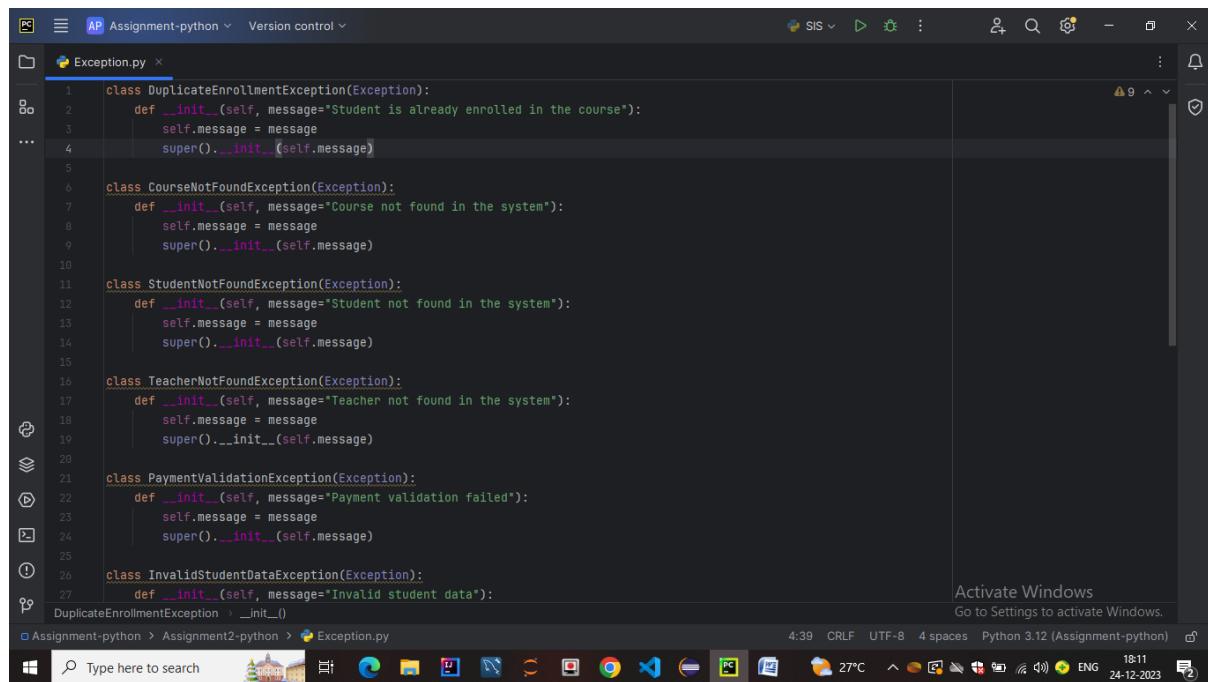
Implementing custom exceptions allows you to define and throw exceptions tailored to specific situations or business logic requirements.

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.

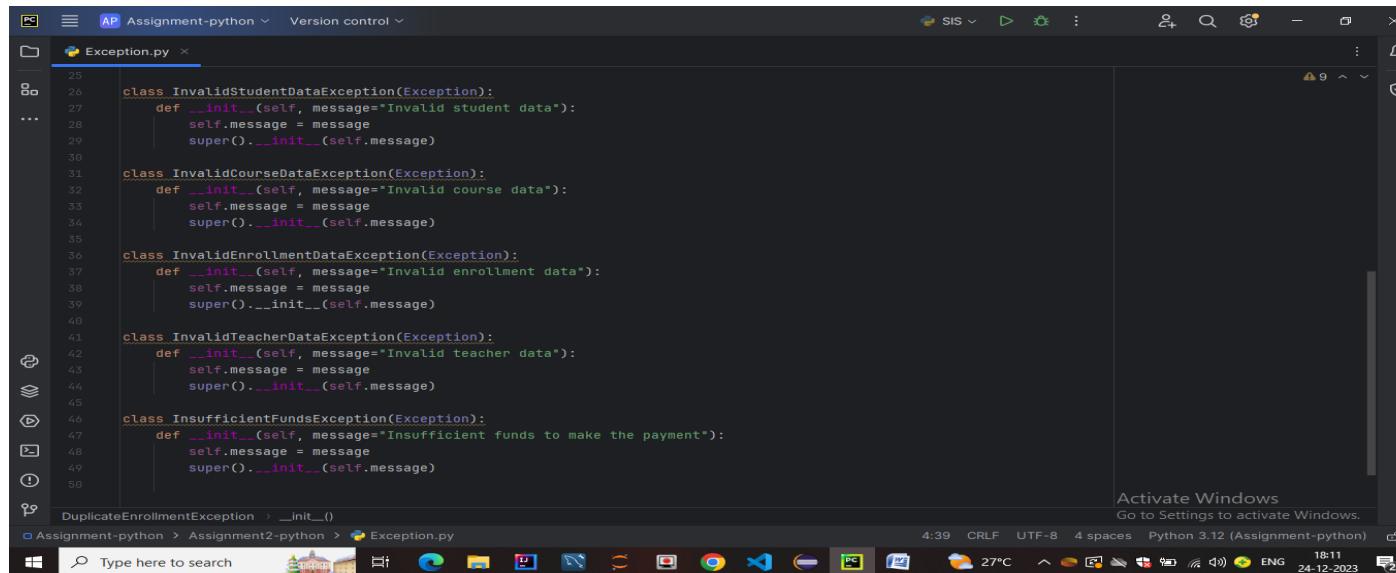
Throw Custom Exceptions In your code, you can throw custom exceptions when specific conditions or business logic rules are violated. To throw a custom exception, use the `throw` keyword followed by an instance of your custom exception class.

- `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.



```
1 class DuplicateEnrollmentException(Exception):
2     def __init__(self, message="Student is already enrolled in the course"):
3         self.message = message
4         super().__init__(self.message)
5
6 class CourseNotFoundException(Exception):
7     def __init__(self, message="Course not found in the system"):
8         self.message = message
9         super().__init__(self.message)
10
11 class StudentNotFoundException(Exception):
12     def __init__(self, message="Student not found in the system"):
13         self.message = message
14         super().__init__(self.message)
15
16 class TeacherNotFoundException(Exception):
17     def __init__(self, message="Teacher not found in the system"):
18         self.message = message
19         super().__init__(self.message)
20
21 class PaymentValidationException(Exception):
22     def __init__(self, message="Payment validation failed"):
23         self.message = message
24         super().__init__(self.message)
25
26 class InvalidStudentDataException(Exception):
27     def __init__(self, message="Invalid student data"):
28         self.message = message
29         super().__init__(self.message)
30
31 DuplicateEnrollmentException > __init__()
```

- **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



```
25
26 class InvalidStudentDataException(Exception):
27     def __init__(self, message="Invalid student data"):
28         self.message = message
29         super().__init__(self.message)
30
31 class InvalidCourseDataException(Exception):
32     def __init__(self, message="Invalid course data"):
33         self.message = message
34         super().__init__(self.message)
35
36 class InvalidEnrollmentDataException(Exception):
37     def __init__(self, message="Invalid enrollment data"):
38         self.message = message
39         super().__init__(self.message)
40
41 class InvalidTeacherDataException(Exception):
42     def __init__(self, message="Invalid teacher data"):
43         self.message = message
44         super().__init__(self.message)
45
46 class InsufficientFundsException(Exception):
47     def __init__(self, message="Insufficient funds to make the payment"):
48         self.message = message
49         super().__init__(self.message)
50
51 DuplicateEnrollmentException > __init__()
```

Task 5: Collections Implement Collections:

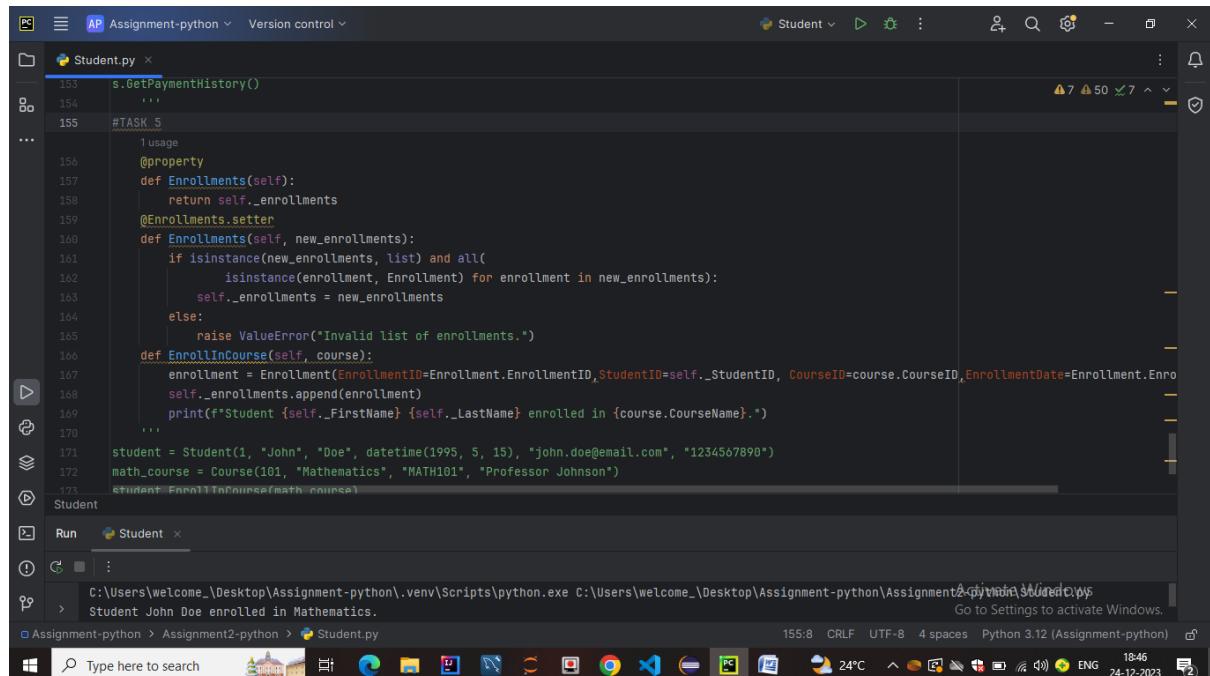
Implement relationships between classes using appropriate data structures (e.g., lists or dictionaries) to maintain associations between students, courses, enrollments, teachers, and payments.

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; }

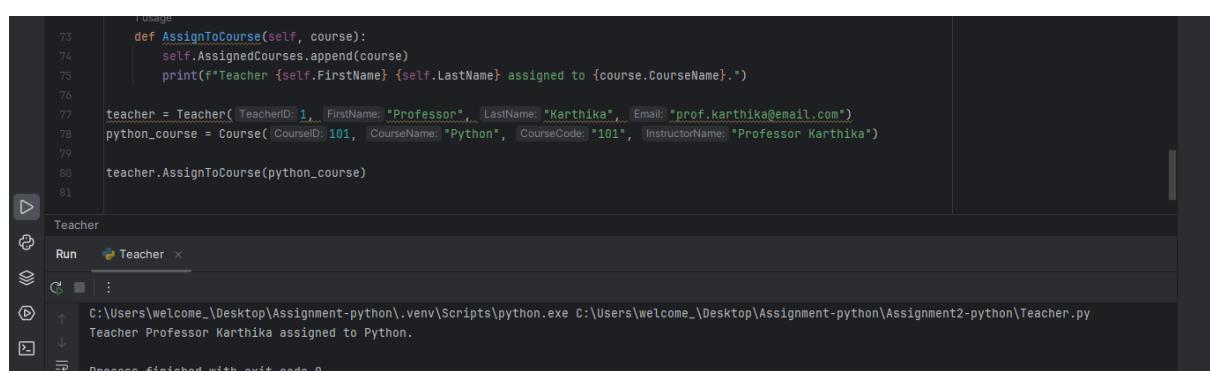


A screenshot of a Windows desktop environment. In the foreground, a code editor window titled "Assignment-python" is open, showing Python code for a "Student" class. The code includes methods for getting and setting enrollment lists, and a main block creating a student and enrolling them in a course. In the background, the taskbar shows several pinned icons (File Explorer, Edge, File Manager, Task View, etc.) and system status indicators like battery level, temperature, and date/time.

```
153     s.GetPaymentHistory()
154     ...
155 #TASK_5
...
156     usage
157     @property
158     def Enrollments(self):
159         return self._enrollments
160     @Enrollments.setter
161     def Enrollments(self, new_enrollments):
162         if isinstance(new_enrollments, list) and all(
163             isinstance(enrollment, Enrollment) for enrollment in new_enrollments):
164             self._enrollments = new_enrollments
165         else:
166             raise ValueError("Invalid list of enrollments.")
167     def EnrollingInCourse(self, course):
168         enrollment = Enrollment(EnrollmentID=EnrollmentID, StudentID=self._StudentID, CourseID=course.CourseID, EnrollmentDate=EnrollmentDate)
169         self._enrollments.append(enrollment)
170         print(f"Student {self._FirstName} {self._LastName} enrolled in {course.CourseName}.")
171     ...
172     student = Student(1, "John", "Doe", datetime(1995, 5, 15), "john.doe@email.com", "1234567890")
173     math_course = Course(101, "Mathematics", "MATH101", "Professor Johnson")
174     student.EnrollingInCourse(math_course)
```

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; }



A screenshot of a Windows desktop environment. In the foreground, a code editor window titled "Assignment2-python" is open, showing Python code for a "Teacher" class. The code includes a method for assigning a course to a teacher. In the background, the taskbar shows several pinned icons (File Explorer, Edge, File Manager, Task View, etc.) and system status indicators like battery level, temperature, and date/time.

```
73     def AssignToCourse(self, course):
74         self.AssignedCourses.append(course)
75         print(f"Teacher {self.FirstName} {self.LastName} assigned to {course.CourseName}.")
76
77     teacher = Teacher( TeacherID=1, FirstName="Professor", LastName="Karthika", Email="prof.karthika@email.com")
78     python_course = Course( CourseID=101, CourseName="Python", CourseCode="101", InstructorName="Professor Karthika")
79
80     teacher.AssignToCourse(python_course)
```

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.

The screenshot shows a code editor with the SIS.py file open. The code defines an `AddEnrollment` method that takes three parameters: `student`, `course`, and `enrollment_date`. It checks if the student and course exist in their respective lists and raises exceptions if they don't. It then creates an `Enrollment` object, increments the enrollment count for both the student and course, and appends the enrollment to both lists. A sample run at the bottom shows creating a student, a course, and then calling the `AddEnrollment` method with those objects and the current date.

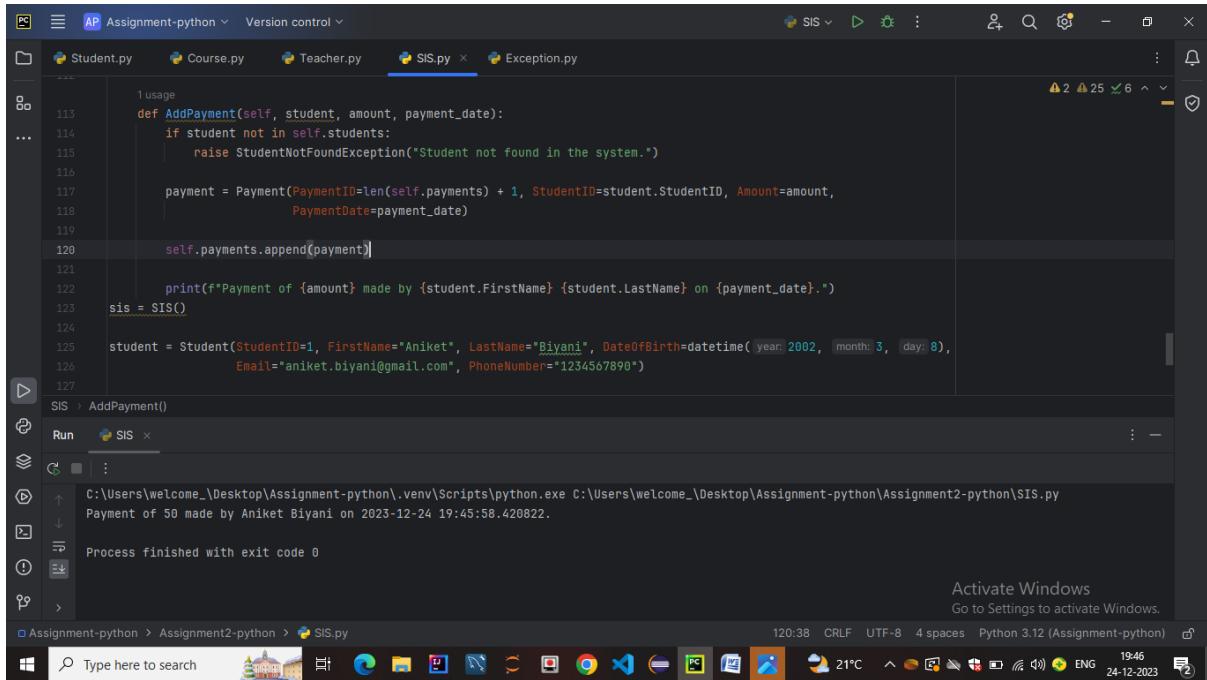
```
69     1 usage
70     def AddEnrollment(self, student, course, enrollment_date):
71         if student not in self.students:
72             raise StudentNotFoundException("Student not found in the system.")
73         if course not in self.courses:
74             raise CourseNotFoundException("Course not found in the system.")
75         enrollment = Enrollment(len(self.enrollments) + 1, student, course, enrollment_date)
76         student.Enrollments.append(enrollment)
77         course.Enrollments.append(enrollment)
78
79         self.enrollments.append(enrollment)
80         print(f"Student course added in enrollment")
81 sis = SIS()
82 student=Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime( year=2002, month=3, day=8),
83                 Email="aniket.biyani@gmail.com", PhoneNumber="1234567890")
84 course = Course( CourseID= 301, CourseName="Python", CourseCode="202", InstructorName="karthika")
85 sis.students.append(student)
86 sis.courses.append(course)
87 sis.AddEnrollment(student, course, datetime.now())
SIS > AddEnrollment() > if course not in self.courses
Run SIS
C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\SIS.py
Student course added in enrollment
Activate Windows
Go to Settings to activate Windows.
Assignment-python > Assignment2-python > SIS.py
74:77 CRLF UTF-8 4 spaces Python 3.12 (Assignment-python) 19:23 23°C 🔍 ENG 24-12-2023
```

- **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.

The screenshot shows a code editor with the SIS.py file open. The code defines an `AssignCourseToTeacher` method that takes two parameters: `course` and `teacher`. It checks if the course and teacher exist in their respective lists and raises exceptions if they don't. It then adds the course to the teacher's `AssignedCourses` list. A sample run at the bottom shows creating a teacher, a course, and then calling the `AssignCourseToTeacher` method with those objects.

```
66     1 usage
67     def AssignCourseToTeacher(self, course, teacher):
68         if course not in self.courses:
69             raise CourseNotFoundException("Course not found in the system.")
70         if teacher not in self.teachers:
71             raise TeacherNotFoundException("Teacher not found in the system.")
72
73         teacher.AssignedCourses.append(course)
74 sis = SIS()
75 teacher = Teacher( TeacherID= 1, FirstName="Professor", LastName="Karthika", Email="prof.karthika@email.com")
76 course = Course( CourseID= 101, CourseName="Python", CourseCode="101", InstructorName="Professor Karthika")
77
78 sis.teachers.append(teacher)
79 sis.courses.append(course)
80
81 sis.AssignCourseToTeacher(course, teacher)
82 print(f"\n{teacher.FirstName} {teacher.LastName}'s Assigned Courses:")
83 for assigned_course in teacher.AssignedCourses:
84     print(f"- {assigned_course.CourseName} ({code: assigned_course.CourseCode})")
Run SIS
C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\SIS.py
Student course added in enrollment
Professor Karthika's Assigned Courses:
- Python (Code: 101)
Activate Windows
Go to Settings to activate Windows.
```

- **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.



```

1 usage
113     def AddPayment(self, student, amount, payment_date):
114         if student not in self.students:
115             raise StudentNotFoundException("Student not found in the system.")
116
117         payment = Payment(PaymentID=len(self.payments) + 1, StudentID=student.StudentID, Amount=amount,
118                           PaymentDate=payment_date)
119
120         self.payments.append(payment)
121
122         print(f"Payment of {amount} made by {student.FirstName} {student.LastName} on {payment_date}.")
123
124     sis = SIS()
125
126     student = Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime(year=2002, month=3, day=8),
127                       Email="aniket.biyani@gmail.com", PhoneNumber="1234567890")

```

SIS > AddPayment()

Run SIS

C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment2-python\SIS.py

Payment of 50 made by Aniket Biyani on 2023-12-24 19:45:58.420822.

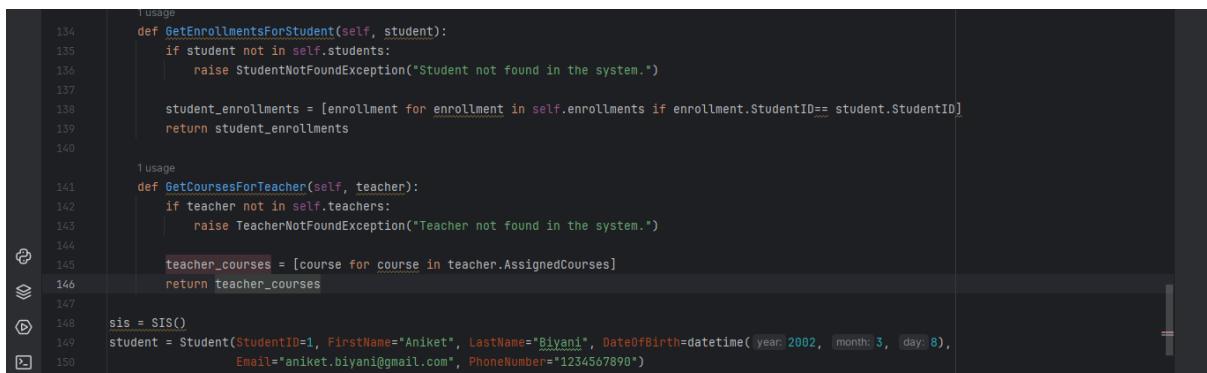
Process finished with exit code 0

Activate Windows
Go to Settings to activate Windows.

120:38 CRLF UTF-8 4 spaces Python 3.12 (Assignment-python) 19:46
24-12-2023

- **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



```

1 usage
134     def GetEnrollmentsForStudent(self, student):
135         if student not in self.students:
136             raise StudentNotFoundException("Student not found in the system.")
137
138         student_enrollments = [enrollment for enrollment in self.enrollments if enrollment.StudentID == student.StudentID]
139
140         return student_enrollments
141
142     def GetCoursesForTeacher(self, teacher):
143         if teacher not in self.teachers:
144             raise TeacherNotFoundException("Teacher not found in the system.")
145
146         teacher_courses = [course for course in teacher.AssignedCourses]
147
148         return teacher_courses
149
150     sis = SIS()
151
152     student = Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime(year=2002, month=3, day=8),
153                       Email="aniket.biyani@gmail.com", PhoneNumber="1234567890")

```

```

148     sis = SIS()
149     student = Student(StudentID=1, FirstName="Aniket", LastName="Biyani", DateOfBirth=datetime( year: 2002, month: 3, day: 8),
150                         Email="aniket.biyani@gmail.com", PhoneNumber="1234567890")
151     sis.students.append(student)
152     teacher = Teacher( TeacherID: 101, FirstName: "Karthika", LastName: "Thangiraj", Email: "karthikamca@gmail.com")
153     sis.teachers.append(teacher)
154     course = Course( CourseID: 101, CourseName: "Python", CourseCode: "101", InstructorName: "Professor Karthika")
155     sis.courses.append(course)
156
157     sis.AddEnrollment(student, course, datetime.now())
158
159     sis.AssignCourseToTeacher(course, teacher)
160     enrollments_for_student = sis.GetEnrollmentsForStudent(student)
161     print(f"Enrollments for {student.FirstName} {student.LastName}:")
162     for enrollment in enrollments_for_student:
163         print(f"- Course: {enrollment.Course.CourseName}, Enrollment Date: {enrollment.EnrollmentDate}")
164
165     courses_for_teacher = sis.GetCoursesForTeacher(teacher)
166     print(f"Courses assigned to {teacher.FirstName}:")
167     for course in courses_for_teacher:
168         print(f"- {course.CourseName}")

Run SIS
C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\SIS.py
Enrollments for Aniket Biyani:
Courses assigned to Karthika:
- Python
Activate Windows
Go to Settings to activate Windows.

```

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.

```

1 from datetime import datetime
2 from Student import Student
3 from Course import Course
4 from Enrollment import Enrollment
5 from SIS import SIS
6
7 usage
8 def main():
9     pass
10
11 if __name__ == "__main__":
12     main()

```

Task 7: Database Connectivity

Database Initialization:

Data Retrieval:

Data Insertion and Updating:

Transaction Management:..

Dynamic Query Builder:

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- Title Bar:** AP Assignment-python > Version control
- File Explorer:** Shows files: Student.py, Course.py, Teacher.py, SIS.py, main.py, DatabaseConnector.py (the active file), and Exception.py.
- Code Editor:** Displays the Python code for the DatabaseConnector class. The code establishes a connection to a MySQL database using the mysql.connector module.
- Status Bar:** Shows the file path: Assignment-python > Assignment2-python > DatabaseConnector.py, and the status: 1:1 CRLF UTF-8 4 spaces Python 3.12 (Assignment-python).
- Bottom Icons:** Includes icons for search, file operations, and system status.

```
1 import mysql.connector
2 from mysql.connector import Error
3
4 class DatabaseConnector:
5     def __init__(self, host, database, user, password):
6         self.host = host
7         self.database = database
8         self.user = user
9         self.password = password
10        self.connection = None
11        self.cursor=None
12    def open_connection(self):
13        try:
14            self.connection = mysql.connector.connect(
15                host=self.host,
16                database=self.database,
17                user=self.user,
18                password=self.password
19            )
20            self.cursor=self.connection.cursor()
21            print("Connected to MySQL database")
22        except Error as e:
23            print(f"Error: {e}")
24
25    def close_connection(self):
26        if self.connection.is_connected():
27            self.connection.close()
```

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:

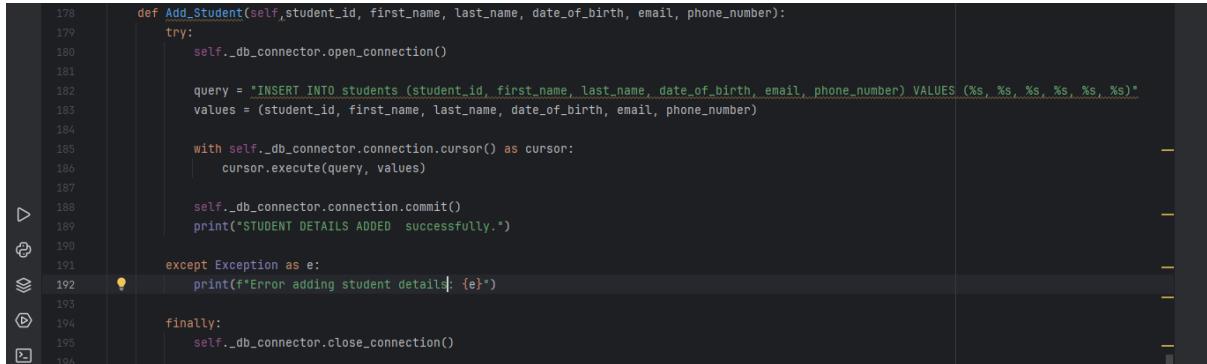
- First Name: John
 - Last Name: Doe
 - Date of Birth: 1995-08-15
 - Email: john.doe@example.com
 - Phone Number: 123-456-7890

John is enrolling in the following courses:

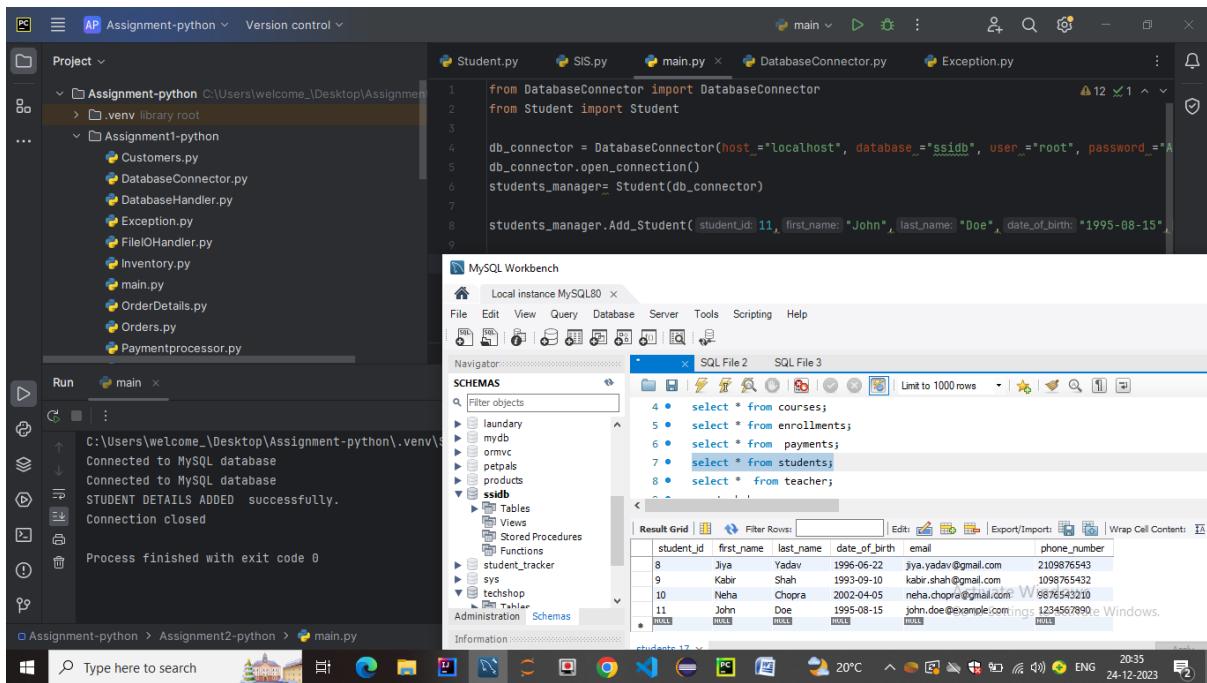
- Course 1: Introduction to Programming
 - Course 2: Mathematics 101

The system should perform the following tasks:

- Create a new student record in the database.



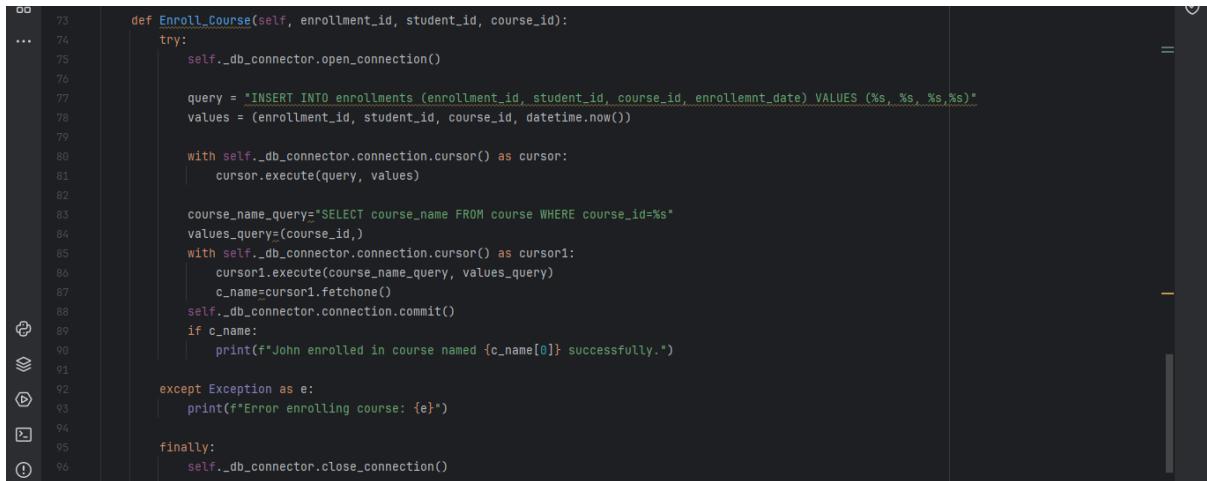
```
178     def Add_Student(self, student_id, first_name, last_name, date_of_birth, email, phone_number):
179         try:
180             self._db_connector.open_connection()
181
182             query = "INSERT INTO students (student_id, first_name, last_name, date_of_birth, email, phone_number) VALUES (%s, %s, %s, %s, %s, %s)"
183             values = (student_id, first_name, last_name, date_of_birth, email, phone_number)
184
185             with self._db_connector.connection.cursor() as cursor:
186                 cursor.execute(query, values)
187
188             self._db_connector.connection.commit()
189             print("STUDENT DETAILS ADDED successfully.")
190
191         except Exception as e:
192             print(f"Error adding student details: {e}")
193
194         finally:
195             self._db_connector.close_connection()
```



Project view showing files like Student.py, SIS.py, main.py, DatabaseConnector.py, and Exception.py. The main.py file contains code to add a student record. The MySQL Workbench window shows a connection to a MySQL database named 'ssidb'. A SQL query is run to select all columns from the 'students' table, resulting in a grid of student data.

student_id	first_name	last_name	date_of_birth	email	phone_number
8	Jiya	Yadav	1996-06-22	jiya.yadav@gmail.com	2109876543
9	Kabir	Shah	1993-09-10	kabir.shah@gmail.com	1098765432
10	Neha	Chopra	2002-04-05	neha.chopra@gmail.com	9876543210
11	John	Doe	1995-08-15	john.doe@example.com	1234567890

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



```
73     def Enroll_Course(self, enrollment_id, student_id, course_id):
74         try:
75             self._db_connector.open_connection()
76
77             query = "INSERT INTO enrollments (enrollment_id, student_id, course_id, enrollemt_date) VALUES (%s, %s, %s, %s)"
78             values = (enrollment_id, student_id, course_id, datetime.now())
79
80             with self._db_connector.connection.cursor() as cursor:
81                 cursor.execute(query, values)
82
83             course_name_query="SELECT course_name FROM course WHERE course_id=%s"
84             values_query=(course_id,)
85             with self._db_connector.connection.cursor() as cursor1:
86                 cursor1.execute(course_name_query, values_query)
87                 c_name=cursor1.fetchone()
88             self._db_connector.connection.commit()
89             if c_name:
90                 print(f"John enrolled in course named {c_name[0]} successfully.")
91
92         except Exception as e:
93             print(f"Error enrolling course: {e}")
94
95         finally:
96             self._db_connector.close_connection()
```

The screenshot shows a development environment with two main windows. On the left is a terminal window titled 'Assignment-python' showing the output of a Python script. The script connects to a MySQL database, enrolls a student in courses, and then disconnects. The terminal also shows the process finished with exit code 0. On the right is the MySQL Workbench interface. It displays the 'Schemas' tree with databases like 'petals', 'products', and 'ssedb'. In the central workspace, there are three tabs: 'SQL File 2', 'SQL File 3', and 'Result Grid'. The SQL tab contains several queries, including `show tables`, `select * from course`, `delete from enrollments where enrollment_id=109;`, and `select * from enrollments`. The Result Grid shows the data from the 'enrollments' table, which has columns 'enrollment_id', 'student_id', 'course_id', and 'enrollment_date'. The data includes rows for enrollment_ids 107, 108, 109, and 110. Below the Result Grid is an 'Output' pane showing the execution history of the queries.

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:

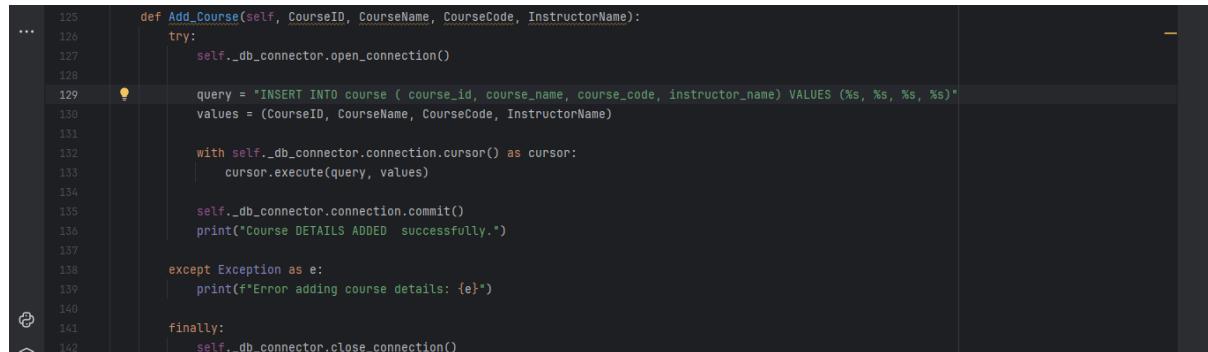
- Name: Sarah Smith • Email: sarah.smith@example.com • Expertise: Computer Science

The screenshot shows a code editor with a Python script for adding a teacher. The script uses a database connector to insert a new teacher record into the 'teacher' table. The teacher details are: TeacherID 311, FirstName 'Sarah', LastName 'Smith', and Email 'sarah.smith@example.com'. The code handles exceptions and ensures the connection is closed after the operation. The code editor shows the entire script from line 86 to 103.

The screenshot shows the MySQL Workbench interface again. The 'Run' tab is active, showing the output of the Python script. It confirms that the teacher DETAILS ADDED successfully. The 'Schemas' tree on the left shows the 'ssedb' database with tables like 'payments', 'students', 'teacher', and 'techshop'. The 'SQL File 3' tab contains several SELECT queries. The 'Result Grid' shows the 'teacher' table with columns 'teacher_id', 'first_name', 'last_name', and 'email'. A new row is added for teacher_id 311, first_name 'Sarah', last_name 'Smith', and email 'sarah.smith@example.com'. The 'Output' pane at the bottom right shows the execution history of the queries.

Course to be assigned:

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

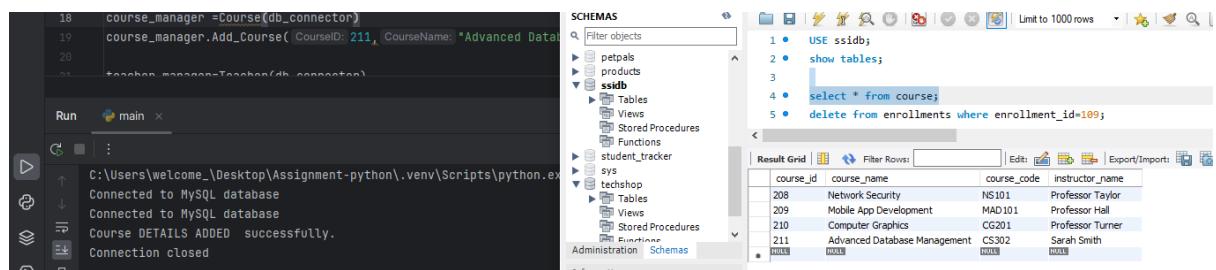


```

125     def Add_Course(self, CourseID, CourseName, CourseCode, InstructorName):
126         ...
127         try:
128             self._db_connector.open_connection()
129
130             query = "INSERT INTO course (course_id, course_name, course_code, instructor_name) VALUES (%s, %s, %s, %s)"
131             values = (CourseID, CourseName, CourseCode, InstructorName)
132
133             with self._db_connector.connection.cursor() as cursor:
134                 cursor.execute(query, values)
135
136             self._db_connector.connection.commit()
137             print("Course DETAILS ADDED successfully.")
138
139         except Exception as e:
140             print(f"Error adding course details: {e}")
141
142         finally:
143             self._db_connector.close_connection()

```

- Assign Sarah Smith as the instructor for the course.



```

18 course_manager = Course(db_connector)
19 course_manager.Add_Course(CourseID=211, CourseName="Advanced Database Management", InstructorName="Sarah Smith")
20
21
22

```

Run main ×

```

C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\main.py
Connected to MySQL database
Connected to MySQL database
Course DETAILS ADDED successfully.
Connection closed

```

SCHEMAS

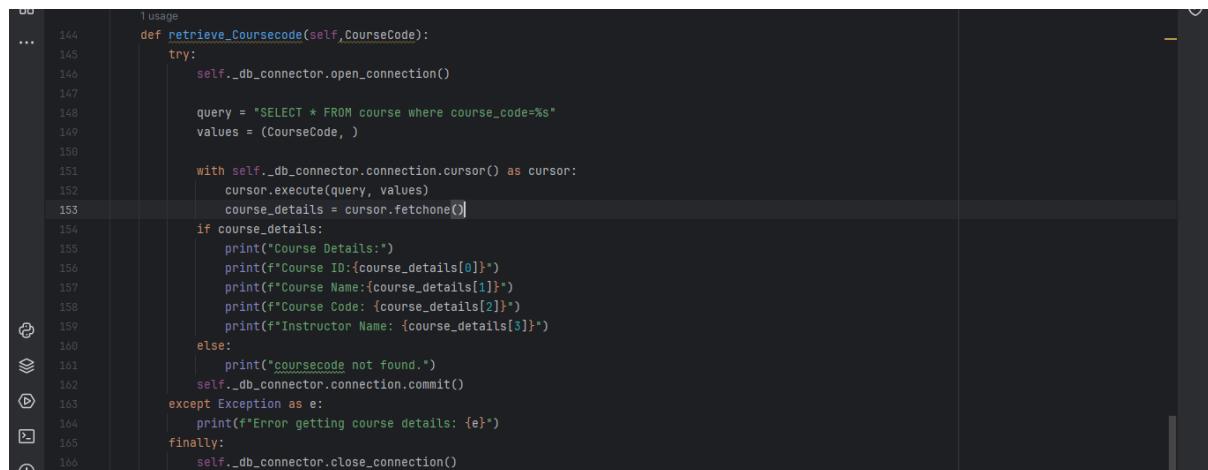
```

1 USE ssidb;
2 show tables;
3
4 select * from course;
5 delete from enrollments where enrollment_id=109;

```

course_id	course_name	course_code	instructor_name
208	Network Security	NS101	Professor Taylor
209	Mobile App Development	MAD101	Professor Hall
210	Computer Graphics	CG201	Professor Turner
211	Advanced Database Management	CS302	Sarah Smith

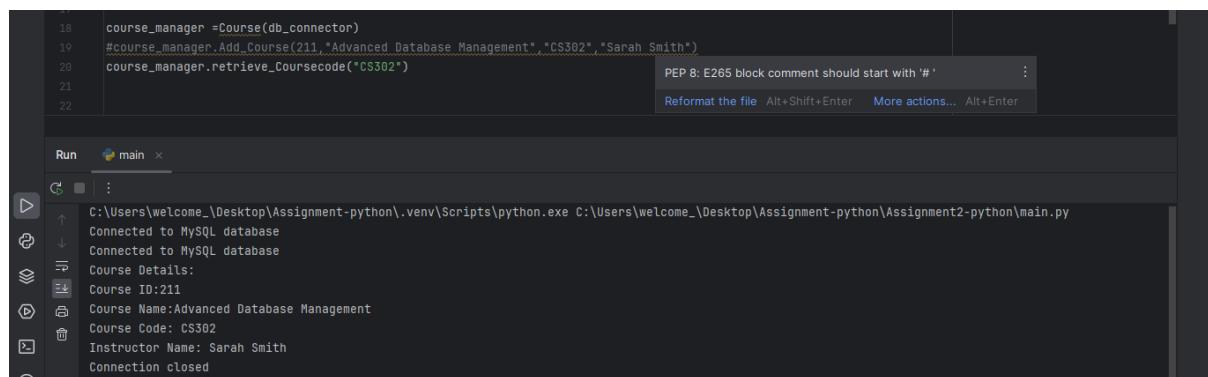
- Retrieve the course record from the database based on the course code.



```

1 Usage
2 def retrieve_Coursecode(self,CourseCode):
3     try:
4         self._db_connector.open_connection()
5
6         query = "SELECT * FROM course WHERE course_code=%s"
7         values = (CourseCode, )
8
9         with self._db_connector.connection.cursor() as cursor:
10             cursor.execute(query, values)
11             course_details = cursor.fetchone()
12
13             if course_details:
14                 print("Course Details:")
15                 print(f"Course ID:{course_details[0]}")
16                 print(f"Course Name:{course_details[1]}")
17                 print(f"Course Code: {course_details[2]}")
18                 print(f"Instructor Name: {course_details[3]}")
19
20             else:
21                 print("coursecode not found.")
22
23         self._db_connector.connection.commit()
24
25     except Exception as e:
26         print(f"Error getting course details: {e}")
27
28     finally:
29         self._db_connector.close_connection()

```



```

18 course_manager = Course(db_connector)
19 #course_manager.Add_Course(211, "Advanced Database Management", "CS302", "Sarah Smith")
20 course_manager.retrieve_Coursecode("CS302")
21
22

```

Run main ×

```

C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\Assignment2-python\main.py
Connected to MySQL database
Connected to MySQL database
Course Details:
Course ID:211
Course Name:Advanced Database Management
Course Code: CS302
Instructor Name: Sarah Smith
Connection closed

```

PEP 8: E265 block comment should start with '#':

Reformat the file Alt+Shift+Enter More actions... Alt+Enter

```

167     def update_Course_instructorname(self, InstructorName, CourseID):
168         try:
169             self._db_connector.open_connection()
170
171             query = "UPDATE Course SET instructor_name=%s where course_id=%s"
172             values = (InstructorName, CourseID)
173
174             with self._db_connector.connection.cursor() as cursor:
175                 cursor.execute(query, values)
176
177             self._db_connector.connection.commit()
178             print("Course instructor UPDATED successfully.")
179
180         except Exception as e:
181             print(f"Error updating course details: {e}")
182
183         finally:
184             self._db_connector.close_connection()

```

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

The screenshot shows the MySQL Workbench interface. On the left, a code editor window displays Python code for updating a course's instructor. On the right, a results grid shows a list of courses with columns for course_id, course_name, course_code, and instructor_name. A SQL query editor window is also visible in the background.

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	Professor Karthika

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

```

76     def Add_Payment(self, PaymentID, StudentID, Amount, PaymentDate):
77         try:
78             self._db_connector.open_connection()
79
80             query = "INSERT INTO payments (payment_id, student_id, amount, payment_date) VALUES (%s, %s, %s, %s)"
81             values = (PaymentID, StudentID, Amount, PaymentDate)
82
83             with self._db_connector.connection.cursor() as cursor:
84                 cursor.execute(query, values)
85
86             self._db_connector.connection.commit()
87             print("Payment DETAILS ADDED successfully.")
88
89         except Exception as e:
90             print(f"Error adding payment details: {e}")
91
92         finally:
93             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.

Screenshot of MySQL Workbench showing a successful payment insertion and a query results grid.

```

30 payments_manager=Payment(db_connector)
31 payments_manager.Add_Payment( PaymentID: 411, StudentID: 11, Amount: 500 )
32
33 db_connector.close_connection()

```

Run main

Connected to MySQL database
Payment DETAILS ADDED Successfully.
Connection closed

Process finished with exit code 0

Result Grid | Filter Rows: | Edit: | Export/Import | Result Grid | Form Editor

payment_id	student_id	amount	payment_date
406	6	6500	2023-06-30
407	7	8000	2023-07-05
408	8	7500	2023-08-12
409	9	4800	2023-09-08
410	10	9000	2023-10-22
411	11	500	2023-04-10

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

Screenshot of PyCharm showing Python code for retrieving a student record by ID.

```

94     def retrieve_studentrecord(self,StudentID):
95         try:
96             self._db_connector.open_connection()
97
98             query = "SELECT * FROM students where student_id=%s"
99             values = (StudentID,)
100
101            with self._db_connector.connection.cursor() as cursor:
102                cursor.execute(query, values)
103                student_details = cursor.fetchone()
104
105                if student_details:
106                    print("student_details:")
107                    print(f"Student ID:{student_details[0]}")
108                    print(f"Student First Name:{student_details[1]}")
109                    print(f"Student Last Name:{student_details[2]}")
110                    print(f"Date Of Birth: {student_details[3]}")
111                    print(f"Email: {student_details[4]}")
112                    print(f"Phone no:{student_details[5]}")
113
114                else:
115                    print("student id not found.")
116
117            self._db_connector.connection.commit()
118
119        except Exception as e:
120            print(f"Error getting course details: {e}")
121
122        finally:
123            self._db_connector.close_connection()

```

Screenshot of MySQL Workbench showing retrieved student record details.

```

30 payments_manager=Payment(db_connector)
31 #payments_manager.Add_Payment(411,11,500,"2023-04-10")
32 payments_manager.retrieve_studentrecord(101)

```

Run main

Connected to MySQL database
Connected to MySQL database
student_details:
Student ID:101
Student First Name:Jane
Student Last Name:Johnson
Date Of Birth: 2000-08-09
Email: janejohnson@gmail.com
Phone no:1234567888
Connection closed

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

Screenshot of PyCharm showing Python code for updating student balance.

```

119     def update_payment_amount(self,Amount,StudentID):
120         try:
121             self._db_connector.open_connection()
122
123             query = "UPDATE Payments SET Amount=%s where student_id=%s"
124             values = (Amount,StudentID)
125
126             with self._db_connector.connection.cursor() as cursor:
127                 cursor.execute(query, values)
128
129             self._db_connector.connection.commit()
130             print("Student balance UPDATED successfully.")
131
132         except Exception as e:
133             print(f"Error updating student balance details: {e}")
134
135         finally:
136             self._db_connector.close_connection()

```

The screenshot shows a Python development environment. On the left is a file tree with files like `Exception.py`, `FileIOHandler.py`, `Inventory.py`, `main.py`, etc. In the center is a code editor with the following Python code:

```

30 payments_manager=Payment(db_connector)
31 #payments_manager.Add_Payment(411,11_500,"2023-04-10")
32 #payments_manager.retrieve_studentrecord(101)
33 payments_manager.update_payment_amount( Amount: 5000, StudentID: 101)
34
35 db_connector.close_connection()
36

```

Below the code editor is a terminal window showing the command `python.exe` running the script `main.py` from the path `C:\Users\welcome\Desktop\Assignment-python\venv\Scripts`. The terminal output indicates a successful connection to MySQL and an update to the student balance.

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.

The screenshot shows a Python development environment. On the left is a file tree with files like `Exception.py`, `FileIOHandler.py`, `Inventory.py`, `main.py`, etc. In the center is a code editor with the following Python code:

```

99 def retrieve_enrollment_record(self, course_id):
100     try:
101         self._db_connector.open_connection()
102
103         query = "SELECT * FROM enrollments where course_id=%s"
104         values = (course_id,)
105
106         with self._db_connector.connection.cursor() as cursor:
107             cursor.execute(query, values)
108             enrollment_details = cursor.fetchall()
109
110             if enrollment_details:
111                 print("Enrollment details:")
112                 for enroll in enrollment_details:
113                     print(f"Enrollment ID:{enroll[0]}")
114                     print(f"Student ID:{enroll[1]}")
115                     print(f"Enrollment Date:{enroll[3]}")
116             else:
117                 print("no enrollments found.")
118
119         self._db_connector.connection.commit()
120     except Exception as e:
121         print(f"Error getting enrollment details: {e}")
122     finally:
123         self._db_connector.close_connection()

```

Below the code editor is a terminal window showing the command `python.exe` running the script `main.py` from the path `C:\Users\welcome\Desktop\Assignment-python\venv\Scripts`. The terminal output shows the retrieved enrollment details for course ID 101.

The screenshot shows a Python development environment. On the left is a file tree with files like `Exception.py`, `FileIOHandler.py`, `Inventory.py`, `main.py`, etc. In the center is a code editor with the following Python code:

```

18 enrollments_manager.retrieve_enrollment_record(202)
19
20 course_manager =Course(db_connector)

```

Below the code editor is a terminal window showing the command `python.exe` running the script `main.py` from the path `C:\Users\welcome\Desktop\Assignment-python\venv\Scripts`. The terminal output shows the retrieved enrollment details for course ID 102.

- Generate an enrollment report listing all students enrolled in Computer Science 101.

The screenshot shows a code editor with two panes. The left pane displays a file tree for a project named 'Codingch2-CareerHub'. The right pane shows a Python script with line numbers 123 to 146. The script defines a method `retrieve_studentenroll_course_record` that connects to a database, executes a query to fetch enrollment details, and prints the results or an error message.

```
def retrieve_studentenroll_course_record(self, course_id):
    try:
        self._db_connector.open_connection()

        query = "SELECT * FROM enrollments WHERE course_id=%s"
        values = (course_id,)

        with self._db_connector.connection.cursor() as cursor:
            cursor.execute(query, values)
            enrollment_details = cursor.fetchall()
            if enrollment_details:
                print("Enrollment details:")
                for enroll in enrollment_details:
                    print(f"Enrollment ID:{enroll[0]}")
                    print(f"Student ID:{enroll[1]}")
                    print(f"Enrollment Date:{enroll[3]}")
            else:
                print("no enrollments found.")

    except Exception as e:
        print(f"Error getting enrollment details: {e}")
    finally:
        self._db_connector.close_connection()
```

- Display or save the report for the administrator.

The screenshot shows a terminal window with a command-line interface. The user runs the script `main.py` and provides the argument `212` to the function `enrollments_manager.retrieve_enrollment_record`. The output shows a connection to the MySQL database and a message indicating no enrollment records were found.

```
OrderDetails.py
Orders.py
Paymentprocessor.py
Products.py
Run main x
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment2-python\main.py
19 enrollments_manager.retrieve_enrollment_record(212)
20
Connected to MySQL database
Connected to MySQL database
no enrollments found.
Connection closed
Process finished with exit code 0
```