

**Implement OOPs :**

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

**Task 1: Define Classes :**

Define the following classes based on the domain description:

Student class with the following attributes:

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

Course class with the following attributes:

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

Enrollment class to represent the relationship between students and courses. It should have attributes:

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

Teacher class with the following attributes:

- Teacher ID • First Name • Last Name • Email

Payment class with the following attributes:

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

**Source Code:**

```
class Student:
    def
__init__(self, Student_ID=None, First_Name=None, Last_Name=None, DOB=None, Email=None, Phone_NO=None):
    self.Student_ID=Student_ID
    self.First_Name = First_Name
    self.Last_Name = Last_Name
    self.DOB = DOB
    self.Email = Email
    self.Phone_NO = Phone_NO

class Course:
    def
__init__(self, Course_ID=None, Course_Name=None, Course_Code=None, Instructor_Name=None):
    self.Course_ID=Course_ID
    self.Course_Name = Course_Name
    self.Course_Code = Course_Code
```

```
        self.Instructor_Name = Instructor_Name
class Enrollment(Student, Course):
    def __init__(self, Enrollment_ID=None, Student_ID=None, Course_ID=None,
Enrollment_Date=None):
        Student.__init__(self, Student_ID)
        Course.__init__(self, Course_ID)
        self.Enrollment_ID = Enrollment_ID
        self.Enrollment_Date = Enrollment_Date
class Teacher:
    def
__init__(self,Teacher_ID=None,First_Name=None,Last_Name=None,Email=None):
        self.Teacher_ID=Teacher_ID
        self.First_Name = First_Name
        self.Last_Name = Last_Name
        self.Email = Email

class Payment(Student):
    def __init__(self, Payment_ID=None, Student_ID=None, Amount=None,
Payment_Date=None):
        self.Payment_ID = Payment_ID
        Student.__init__(self, Student_ID)
        self.Amount = Amount
        self.Payment_Date = Payment_Date
```

## Task 2: Implement Constructors

Implement constructors for each class to initialize their attributes. Constructors are special methods that are called when an object of a class is created. They are used to set initial values for the attributes of the class. Below are detailed instructions on how to implement constructors for each class in your Student

### Information System (SIS) assignment:

#### Student Class Constructor

In the Student class, you need to create a constructor that initializes the attributes of a student when an instance of the Student class is created

#### SIS Class Constructor

If you have a class that represents the Student Information System itself (e.g., SIS class), you may also implement a constructor for it. This constructor can be used to set up any initial configuration for the SIS.

Repeat the above process for each class Course, Enrollment, Teacher, Payment by defining constructors that initialize their respective attributes.

### Source Code:

```
class Student:

    def __init__(self, Student_ID=None, First_Name=None, Last_Name=None, DOB=None,
Email=None, Phone_NO=None):

        self.Student_ID=Student_ID
        self.First_Name = First_Name
        self.Last_Name = Last_Name
        self.DOB = DOB
        self.Email = Email
        self.Phone_NO = Phone_NO

class Course:

    def __init__(self, Course_ID=None, Course_Name=None, Course_Code=None, Instru
ctor_Name=None):

        self.Course_ID=Course_ID
        self.Course_Name = Course_Name
        self.Course_Code = Course_Code
        self.Instructor_Name = Instructor_Name

class Enrollment(Student, Course):

    def __init__(self, Enrollment_ID=None, Student_ID=None, Course_ID=None,
Enrollment_Date=None):

        Student.__init__(self, Student_ID)
        Course.__init__(self, Course_ID)
        self.Enrollment_ID = Enrollment_ID
        self.Enrollment_Date = Enrollment_Date
```

```
class Teacher:
    def __init__(self, Teacher_ID=None, First_Name=None, Last_Name=None, Email=None):
        self.Teacher_ID = Teacher_ID
        self.First_Name = First_Name
        self.Last_Name = Last_Name
        self.Email = Email

class Payment(Student):
    def __init__(self, Payment_ID=None, Student_ID=None, Amount=None,
Payment_Date=None):
        self.Payment_ID = Payment_ID
        Student.__init__(self, Student_ID)
        self.Amount = Amount
        self.Payment_Date = Payment_Date

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

### Task 3: Implement Methods

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

Implement the following methods in the appropriate classes:

#### Student Class:

- `EnrollInCourse(course: Course)`: Enrolls the student in a course.
- `UpdateStudentInfo(firstName: string, lastName: string, dateOfBirth: DateTime, email: string, phoneNumber: string)`: Updates the student's information.
- `MakePayment(amount: decimal, paymentDate: DateTime)`: Records a payment made by the student.
- `DisplayStudentInfo()`: Displays detailed information about the student.
- `GetEnrolledCourses()`: Retrieves a list of courses in which the student is enrolled.
- `GetPaymentHistory()`: Retrieves a list of payment records for the student.

#### Course Class:

- `AssignTeacher(teacher: Teacher)`: Assigns a teacher to the course.
- `UpdateCourseInfo(courseCode: string, courseName: string, instructor: string)`: Updates course information.
- `DisplayCourseInfo()`: Displays detailed information about the course.
- `GetEnrollments()`: Retrieves a list of student enrollments for the course.
- `GetTeacher()`: Retrieves the assigned teacher for the course.

#### Enrollment Class:

- `GetStudent()`: Retrieves the student associated with the enrollment.
- `GetCourse()`: Retrieves the course associated with the enrollment.

#### Teacher Class:

- `UpdateTeacherInfo(name: string, email: string, expertise: string)`: Updates teacher information.
- `DisplayTeacherInfo()`: Displays detailed information about the teacher.
- `GetAssignedCourses()`: Retrieves a list of courses assigned to the teacher.

#### Payment Class:

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

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

- `EnrollStudentInCourse(student: Student, course: Course)`: Enrolls a student in a course.
- `AssignTeacherToCourse(teacher: Teacher, course: Course)`: Assigns a teacher to a course.

- RecordPayment(student: Student, amount: decimal, paymentDate: DateTime): Records a payment made by a student.
- GenerateEnrollmentReport(course: Course): Generates a report of students enrolled in a specific course.
- GeneratePaymentReport(student: Student): Generates a report of payments made by a specific student.
- CalculateCourseStatistics(course: Course): Calculates statistics for a specific course, such as the number of enrollments and total payments.

**Source Code:**

```
from datetime import datetime
class Student:
    def
__init__(self, Student_ID=None, First_Name=None, Last_Name=None, DOB=None, Email=None, Phone_NO=None):
    self.Student_ID=Student_ID
    self.First_Name = First_Name
    self.Last_Name = Last_Name
    self.DOB = DOB
    self.Email = Email
    self.Phone_NO = Phone_NO
    self.enrolled_courses=[]
    self.payment_made=[]
    def EnrollInCourse(self, course):
        if course not in self.enrolled_courses:
            self.enrolled_courses.append(course)
            print(f"Student {self.Student_ID} enrolled in course {course.Course_ID}")
        else:
            print(f"Student {self.Student_ID} is already enrolled in course {course.Course_ID}")

    def UpdateStudentInfo(self, First_Name, Last_Name, DOB, Email, Phone_NO):
        self.First_Name = First_Name
        self.Last_Name = Last_Name
        self.DOB = DOB
        self.Email = Email
        self.Phone_NO = Phone_NO

    def MakePayment(self, payment):
```

```

        self.payment_made.append(payment)
        print(f"Payment of {payment.Amount} made by {self.First_Name} on
{payment.Payment_Date}")

    def DisplayStudentInfo(self):
        print('-' * 50)
        print("Students Detail: ")
        print(f"Student ID: {self.Student_ID}")
        print(f"Name: {self.First_Name} {self.Last_Name}")
        print(f>Date of Birth: {self.DOB}")
        print(f>Email: {self.Email}")
        print(f>Phone Number: {self.Phone_NO}")

    def GetEnrolledCourses(self):
        print('-' * 50)
        print(f"Enrolled Courses for {self.Student_ID}:")
        for course in self.enrolled_courses:
            print(f">{course.Course_ID}-{course.Course_Name}")

    def GetPaymentHistory(self):
        print(f"Payment History for {self.Student_ID}: ")
        for payment in self.payment_made:
            print(f"Payment Amount: {payment.Amount}, Date:
{payment.Payment_Date} ")

class Course:

    def
__init__(self, Course_ID=None, Course_Name=None, Course_Code=None, Instructor_Name=
None):

        self.Course_ID=Course_ID
        self.Course_Name = Course_Name
        self.Course_Code = Course_Code
        self.Instructor_Name = Instructor_Name
        self.course_assigned = []
        self.enrolled_students = []

    def AssignTeacher(self, teacher):
        if teacher not in self.course_assigned:
            self.course_assigned.append(teacher)
            print(f">{teacher.First_Name} has been assigned to the course

```

```

{self.Course_Name}")
        else:
            print(f"{teacher.First_Name} has already been assigned to this
course")

    def AssignStudent(self, student):
        if student not in self.course_assigned:
            self.enrolled_students.append(student)
            print(f"{student.First_Name} has been enrolled to the course
{self.Course_Name}")
        else:
            print(f"{student.First_Name} has already been enrolled to this
course")

    def UpdateCourseInfo(self, Course_Name, Course_Code, Instructor_Name):
        self.Course_Name = Course_Name
        self.Course_Code = Course_Code
        self.Instructor_Name = Instructor_Name

    def DisplayCourseInfo(self):
        print('-' * 50)
        print("Course Detail: ")
        print("Course Id: ", self.Course_ID)
        print("Course Name: ", self.Course_Name)
        print("Course Code: ", self.Course_Code)
        print("Instructor Name: ", self.Instructor_Name)

    def GetEnrollments(self, student):
        print("Enrolled Students")
        for student in self.enrolled_students:
            print(f"Student {student.First_Name} has enrolled in
{self.Course_Code} course")

    def GetTeacher(self, teacher):
        print("Assigned Teacher")
        for teacher in self.course_assigned:
            print(f"Teacher {teacher.First_Name} has assigned to
{self.Course_Code} course")

class Enrollment(Student, Course):
    def __init__(self, Enrollment_ID=None, Student_ID=None, Course_ID=None,
Enrollment_Date=None):

```



```

        Student.__init__(self, Student_ID)
        Course.__init__(self, Course_ID)
        self.Enrollment_ID = Enrollment_ID
        self.Enrollment_Date = Enrollment_Date

    def GetStudent(self):
        return self.Student_ID

    def GetCourse(self):
        return self.Course_ID

class Teacher:
    def
__init__(self,Teacher_ID=None,First_Name=None,Last_Name=None,Email=None):
        self.Teacher_ID=Teacher_ID
        self.First_Name = First_Name
        self.Last_Name = Last_Name
        self.Email = Email
        self.assigned_courses=[]

    def DisplayTeacherInfo(self):
        print('-' * 50)
        print("Teacher Info: ")
        print(f"Teacher ID: {self.Teacher_ID}")
        print(f"Name: {self.First_Name} {self.Last_Name}")
        print(f"Email: {self.Email}")

    def UpdateTeacherInfo(self,First_Name,Last_Name,Email):
        self.First_Name = First_Name
        self.Last_Name = Last_Name
        self.Email = Email

    def AssignToCourse(self, course):
        if course not in self.assigned_courses:
            self.assigned_courses.append(course)
            print('-' * 50)
            print(f"Teacher {self.First_Name} assigned to course
{course.Course_Name}")
        else:
            print('-' * 50)
            print(f"Teacher {self.First_Name} is already assigned to course
{course.Course_Name}")

```

```

def GetAssignedCourses(self):
    print('-' * 50)
    print(f"Courses assigned to Teacher {self.First_Name}:")
    for course in self.assigned_courses:
        print(f"{course.Course_ID}-{course.Course_Name}")

class Payment(Student):
    def __init__(self, Payment_ID=None, Student_ID=None, Amount=None,
Payment_Date=None):
        self.Payment_ID = Payment_ID
        Student.__init__(self, Student_ID)
        self.Amount = Amount
        self.Payment_Date = Payment_Date

    def GetStudent(self):
        print(f"{self.Student_ID} has made the payment")

    def GetPaymentAmount(self):
        print(f"Payment of {self.Amount} has been made")

    def GetPaymentDate(self):
        print(f"The Payment was made on {self.Payment_Date} ")

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

    def EnrollStudentInCourse(self, student, course):
        student.EnrollInCourse(course)

    def AssignTeacherToCourse(self, teacher, course):
        course.AssignTeacher(teacher)

    def RecordPayment(self, student, amount, payment_date):
        payment = Payment(Student_ID=student.Student_ID, Amount=amount,
Payment_Date=payment_date)
        student.MakePayment(payment)

```

```
        self.payments.append(payment)

    def GenerateEnrollmentReport(self, course):
        print(f"Enrollment Report for Course {course.Course_Name}:")
        for student in course.enrolled_students:
            print(f"Student ID: {student.Student_ID}, Name: {student.First_Name} {student.Last_Name}")

    def GeneratePaymentReport(self, student):
        print(f"Payment Report for Student {student.Student_ID} - {student.First_Name} {student.Last_Name}:")
        for payment in student.payment_made:
            print(f"Payment Amount: {payment.Amount}, Date: {payment.Payment_Date}")

    def CalculateCourseStatistics(self, course):
        enrolled_students_count = len(course.enrolled_students)
        total_payments = sum(payment.Amount for student in course.enrolled_students
                               for payment in student.payment_made)
        print('-' * 50)
        print(f"Statistics for Course {course.Course_Name}:")
        print(f"Number of Enrollments: {enrolled_students_count}")
        print(f"Total Payments: {total_payments}")
```

## Use the Methods

In your driver program or any part of your code where you want to perform actions related to the Student Information System, create instances of your classes, and use the methods you've implemented.

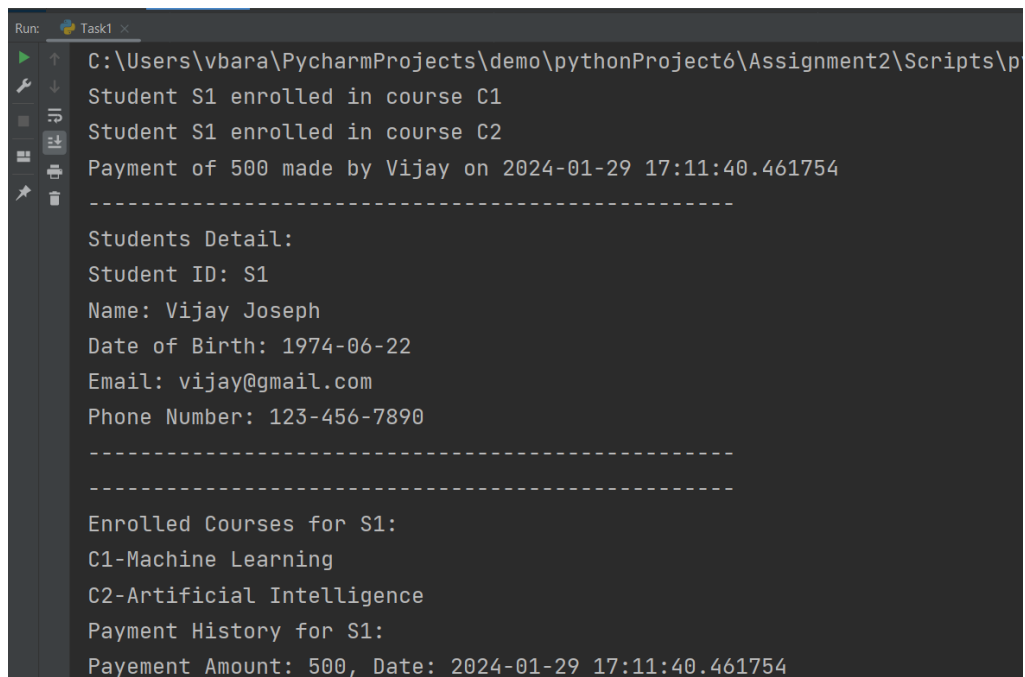
Repeat this process for using other methods you've implemented in your classes and the SIS class.

**Note:** Divided the driver program to use methods which I have implemented for all the respective classes

### Driver Code part 1:

```
student1 = Student(Student_ID="S1", First_Name="Vijay", Last_Name="Joseph",
DOB="1974-06-22", Email="vijay@gmail.com", Phone_NO="123-456-7890")
course1 = Course(Course_ID="C1", Course_Name="Machine Learning",
Course_Code="AB101", Instructor_Name="Rajinikanth")
course2 = Course(Course_ID="C2", Course_Name="Artificial Intelligence",
Course_Code="DC102", Instructor_Name="Kamal Haasan")
enrollment1 = Enrollment(Enrollment_ID="E1", Student_ID=student1.Student_ID,
Course_ID=course1.Course_ID, Enrollment_Date="2024-01-29")
teacher1 = Teacher(Teacher_ID="T1", First_Name="Samantha",
Last_Name="Akkineni", Email="samantha@gmail.com")
payment1 = Payment(Payment_ID="P1", Student_ID=student1.Student_ID, Amount=500,
Payment_Date=datetime.now())
student1.EnrollInCourse(course1)
student1.EnrollInCourse(course2)
student1.MakePayment(payment1)
student1.DisplayStudentInfo()
print('-'*50)
student1.GetEnrolledCourses()
student1.GetPaymentHistory()
```

### Output:

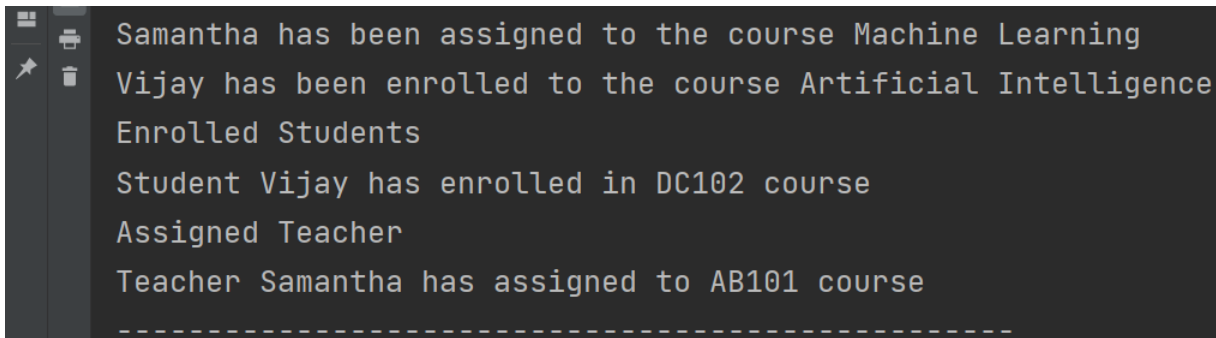


```
Run: Task1 x
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\Scripts\p
Student S1 enrolled in course C1
Student S1 enrolled in course C2
Payment of 500 made by Vijay on 2024-01-29 17:11:40.461754
-----
Students Detail:
Student ID: S1
Name: Vijay Joseph
Date of Birth: 1974-06-22
Email: vijay@gmail.com
Phone Number: 123-456-7890
-----
Enrolled Courses for S1:
C1-Machine Learning
C2-Artificial Intelligence
Payment History for S1:
Payement Amount: 500, Date: 2024-01-29 17:11:40.461754
```

### Driver Code part 2:

```
course1.AssignTeacher(teacher1)
course2.AssignStudent(student1)
course2.GetEnrollments(student1)
course1.GetTeacher(teacher1)
```

### Output:

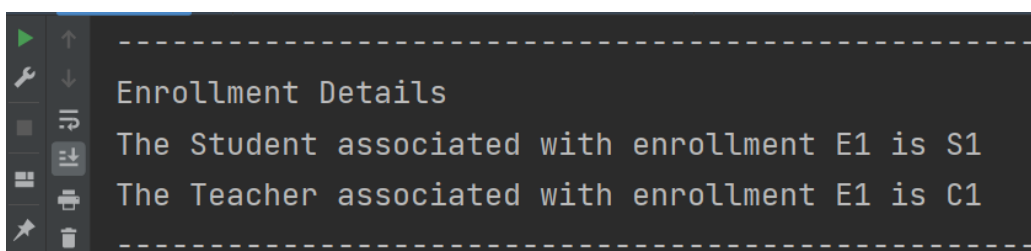


```
Samantha has been assigned to the course Machine Learning
Vijay has been enrolled to the course Artificial Intelligence
Enrolled Students
Student Vijay has enrolled in DC102 course
Assigned Teacher
Teacher Samantha has assigned to AB101 course
-----
```

### Driver Code part 3:

```
enrollment2 = Enrollment(Enrollment_ID="E1",
Student_ID=student1.Student_ID, Course_ID=course1.Course_ID,
Enrollment_Date="2024-01-29")
enrolled_students=enrollment1.GetStudent()
print('-'*50)
print("Enrollment Details")
print(f"The Student associated with enrollment
{enrollment1.Enrollment_ID} is {enrolled_students}")
enrolled_courses=enrollment1.GetCourse()
print(f"The Teacher associated with enrollment
{enrollment1.Enrollment_ID} is {enrolled_courses}")
```

### Output:



```
-----
Enrollment Details
The Student associated with enrollment E1 is S1
The Teacher associated with enrollment E1 is C1
-----
```

### Driver Code part 3:

```
teacher1.AssignToCourse(course2)
teacher1.GetAssignedCourses()
teacher1.DisplayTeacherInfo()
print('-'*50)
```

### Output:

```
-----
Teacher Samantha assigned to course Artificial Intelligence
-----
Courses assigned to Teacher Samantha:
C2-Artificial Intelligence
-----
Teacher Info:
Teacher ID: T1
Name: Samantha Akkineni
Email: samantha@gmail.com
-----
```

### Driver Code part 4:

```
print("Payment Details: ")
payment1.GetStudent()
payment1.GetPaymentAmount()
payment1.GetPaymentDate()
print('-'*50)
```

### Output:

```
-----
Payment Details:
S1 has made the payment
Payment of 500 has been made
The Payment was made on 2024-01-29 19:32:38.555943
-----
```

### Driver Code part 5:

```
sis=SIS()
sis.EnrollStudentInCourse(student1, course1)
sis.AssignTeacherToCourse(teacher1, course1)

sis.RecordPayment(student1, amount=500, payment_date=datetime.now())
```

```
sis.GenerateEnrollmentReport(course2)
sis.GeneratePaymentReport(student1)
sis.CalculateCourseStatistics(course1)
```

**Output:**

```
-----
Student S1 is already enrolled in course C1
Samantha has already been assigned to this course
Payment of 500 made by Vijay on 2024-01-29 19:32:38.555943
-----
Enrollment Report for Course Artificial Intelligence:
Student ID: S1, Name: Vijay Joseph
-----
Payment Report for Student S1 - Vijay Joseph:
Payment Amount: 500, Date: 2024-01-29 19:32:38.555943
Payment Amount: 500, Date: 2024-01-29 19:32:38.555943
-----
Statistics for Course Machine Learning:
Number of Enrollments: 0
Total Payments: 0
Payment of 500 made by Vijay on 2024-01-30 14:02:35.670990
-----
```

## Task 4: Exceptions handling and Custom Exceptions

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

### Create Custom Exception Classes

You'll need to create custom exception classes that are inherited from the `System.Exception` class or one of its derived classes (e.g., `System.ApplicationException`). These custom exception classes will allow you to encapsulate specific error scenarios and provide meaningful error messages.

### Throw Custom Exceptions

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

- `DuplicateEnrollmentException`: Thrown when a student is already enrolled in a course and tries to enroll again. This exception can be used in the `EnrollStudentInCourse` method.
- `CourseNotFoundException`: Thrown when a course does not exist in the system, and you attempt to perform operations on it (e.g., enrolling a student or assigning a teacher).
- `StudentNotFoundException`: Thrown when a student does not exist in the system, and you attempt to perform operations on the student (e.g., enrolling in a course, making a payment).
- `TeacherNotFoundException`: Thrown when a teacher does not exist in the system, and you attempt to assign them to a course.
- `PaymentValidationException`: Thrown when there is an issue with payment validation, such as an invalid payment amount or payment date.
- `InvalidStudentDataException`: Thrown when data provided for creating or updating a student is invalid (e.g., invalid date of birth or email format).
- `InvalidCourseDataException`: Thrown when data provided for creating or updating a course is invalid (e.g., invalid course code or instructor name).
- `InvalidEnrollmentDataException`: Thrown when data provided for creating an enrollment is invalid (e.g., missing student or course references).
- `InvalidTeacherDataException`: Thrown when data provided for creating or updating a teacher is invalid (e.g., missing name or email).
- `InsufficientFundsException`: Thrown when a student attempts to enroll in a course but does not have enough funds to make the payment.

### Source Code:

```
from datetime import datetime

class DuplicateEnrollmentException(Exception):
    def __init__(self, student_id, course_id):
        super().__init__(f"Student {student_id} is already enrolled in course {course_id}")

class CourseNotFoundException(Exception):
    def __init__(self, course_id):
        super().__init__(f"Course {course_id} not found")
```



```
class StudentNotFoundException(Exception):
    def __init__(self, student_id):
        super().__init__(f"Student {student_id} not found")

class TeacherNotFoundException(Exception):
    def __init__(self, teacher_id):
        super().__init__(f"Teacher {teacher_id} not found")

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

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

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

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

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

class InsufficientFundsException(Exception):
    def __init__(self, student_id, course_id):
        super().__init__(f"Student {student_id} does not have enough funds to enroll in course {course_id}")

class Student:
    def
__init__(self, Student_ID=None, First_Name=None, Last_Name=None, DOB=None, Email=None, Phone_NO=None):
        self.Student_ID=Student_ID
        self.First_Name = First_Name
        self.Last_Name = Last_Name
```

```

        self.DOB = DOB
        self.Email = Email
        self.Phone_NO = Phone_NO
        self.enrolled_courses=[]
        self.payment_made=[]
    def EnrollInCourse(self,course):
        if course not in self.enrolled_courses:
            self.enrolled_courses.append(course)
            print(f"Student {self.Student_ID} enrolled in course
{course.Course_ID}")
        else:
            print(f"Student {self.Student_ID} is already enrolled in course
{course.Course_ID}")

    def UpdateStudentInfo(self,First_Name,Last_Name,DOB,Email,Phone_NO):
        self.First_Name = First_Name
        self.Last_Name = Last_Name
        self.DOB = DOB
        self.Email = Email
        self.Phone_NO = Phone_NO

    def MakePayment(self,payment):
        self.payment_made.append(payment)
        print(f"Payment of {payment.Amount} made by {self.First_Name} on
{payment.Payment_Date}")

    def DisplayStudentInfo(self):
        print('-' * 50)
        print("Students Detail: ")
        print(f"Student ID: {self.Student_ID}")
        print(f"Name: {self.First_Name} {self.Last_Name}")
        print(f>Date of Birth: {self.DOB}")
        print(f>Email: {self.Email}")
        print(f"Phone Number: {self.Phone_NO}")

    def GetEnrolledCourses(self):
        print('-' * 50)
        print(f"Enrolled Courses for {self.Student_ID}:")
        for course in self.enrolled_courses:
            print(f"{course.Course_ID}-{course.Course_Name}")

```

```

def GetPaymentHistory(self):
    print(f"Payment History for {self.Student_ID}: ")
    for payment in self.payment_made:
        print(f"Payment Amount: {payment.Amount}, Date:
{payment.Payment_Date} ")

class Course:

    def
__init__(self, Course_ID=None, Course_Name=None, Course_Code=None, Instructor_Name=
None):

        self.Course_ID=Course_ID
        self.Course_Name = Course_Name
        self.Course_Code = Course_Code
        self.Instructor_Name = Instructor_Name
        self.course_assigned = []
        self.enrolled_students = []

    def AssignTeacher(self, teacher):
        if teacher not in self.course_assigned:
            self.course_assigned.append(teacher)
            print(f"{teacher.First_Name} has been assigned to the course
{self.Course_Name}")
        else:
            print(f"{teacher.First_Name} has already been assigned to this
course")

    def AssignStudent(self, student):
        if student not in self.course_assigned:
            self.enrolled_students.append(student)
            print(f"{student.First_Name} has been enrolled to the course
{self.Course_Name}")
        else:
            print(f"{student.First_Name} has already been enrolled to this
course")

    def UpdateCourseInfo(self, Course_Name, Course_Code, Instructor_Name):
        self.Course_Name = Course_Name
        self.Course_Code = Course_Code
        self.Instructor_Name = Instructor_Name

```

```

def DisplayCourseInfo(self):
    print('-' * 50)
    print("Course Detail: ")
    print("Course Id: ",self.Course_ID)
    print("Course Name: ", self.Course_Name)
    print("Course Code: ", self.Course_Code)
    print("Instructor Name: ", self.Instructor_Name)

def GetEnrollments(self,student):
    print("Enrolled Students")
    for student in self.enrolled_students:
        print(f"Student {student.First_Name} has enrolled in
{self.Course_Code} course")
    def GetTeacher(self,teacher):
        print("Assigned Teacher")
        for teacher in self.course_assigned:
            print(f"Teacher {teacher.First_Name} has assigned to
{self.Course_Code} course")

class Enrollment(Student, Course):
    def __init__(self, Enrollment_ID=None, Student_ID=None, Course_ID=None,
Enrollment_Date=None):
        Student.__init__(self, Student_ID)
        Course.__init__(self, Course_ID)
        self.Enrollment_ID = Enrollment_ID
        self.Enrollment_Date = Enrollment_Date

    def GetStudent(self):
        return self.Student_ID

    def GetCourse(self):
        return self.Course_ID

class Teacher:
    def
__init__(self,Teacher_ID=None,First_Name=None,Last_Name=None,Email=None):
        self.Teacher_ID=Teacher_ID
        self.First_Name = First_Name
        self.Last_Name = Last_Name
        self.Email = Email
        self.assigned_courses=[]

```

```

def DisplayTeacherInfo(self):
    print('-' * 50)
    print("Teacher Info: ")
    print(f"Teacher ID: {self.Teacher_ID}")
    print(f"Name: {self.First_Name} {self.Last_Name}")
    print(f"Email: {self.Email}")

def UpdateTeacherInfo(self, First_Name, Last_Name, Email):
    self.First_Name = First_Name
    self.Last_Name = Last_Name
    self.Email = Email

def AssignToCourse(self, course):
    if course not in self.assigned_courses:
        self.assigned_courses.append(course)
        print('-' * 50)
        print(f"Teacher {self.First_Name} assigned to course {course.Course_Name}")
    else:
        print('-' * 50)
        print(f"Teacher {self.First_Name} is already assigned to course {course.Course_Name}")

def GetAssignedCourses(self):
    print('-' * 50)
    print(f"Courses assigned to Teacher {self.First_Name}:")
    for course in self.assigned_courses:
        print(f"{course.Course_ID}-{course.Course_Name}")

class Payment(Student):
    def __init__(self, Payment_ID=None, Student_ID=None, Amount=None, Payment_Date=None):
        self.Payment_ID = Payment_ID
        Student.__init__(self, Student_ID)
        self.Amount = Amount
        self.Payment_Date = Payment_Date

    def GetStudent(self):
        print(f"{self.Student_ID} has made the payment")

    def GetPaymentAmount(self):

```

```

        print(f"Payment of {self.Amount} has been made")

    def GetPaymentDate(self):
        print(f"The Payment was made on {self.Payment_Date} ")

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

    def EnrollStudentInCourse(self, student, course):
        if course not in self.courses:
            raise CourseNotFoundException(f"{course.Course_ID} not found")
        if not isinstance(student, Student):
            raise InvalidStudentDataException("Invalid student object
provided")
        if student in self.students:
            raise
DuplicateEnrollmentException(student.Student_ID, course.Course_ID)
        if not isinstance(course, Course):
            raise InvalidCourseDataException("Invalid course object provided")

        try:
            student.EnrollInCourse(course)
            self.students.append(student)
        except DuplicateEnrollmentException as e:
            print(f"Enrollment Error: {e}")
            raise e
        except Exception as e:
            print(f"Unexpected Error: {e}")
            raise e

    def AssignTeacherToCourse(self, teacher, course):
        if teacher not in self.teachers:
            raise TeacherNotFoundException(teacher.Teacher_ID)
        if type(teacher) is not Teacher:
            raise InvalidTeacherDataException("Invalid teacher object
provided")

```

```

        if type(course) is not Course:
            raise InvalidCourseDataException("Invalid course object provided")

    try:
        course.AssignTeacher(teacher)
    except TeacherNotFoundException as e:
        raise e # Re-raise the specific exception
    except Exception as e:
        raise e # Re-raise any unexpected exceptions

def RecordPayment(self, student, amount, payment_date):
    if student not in self.students:
        raise StudentNotFoundException(student.Student_ID)
    if type(student) is not Student:
        raise InvalidStudentDataException("Invalid student object
provided")
    if amount < 0:
        raise PaymentValidationException("Invalid Amount")
    try:
        payment = Payment(Student_ID=student.Student_ID, Amount=amount,
Payment_Date=payment_date)
        student.MakePayment(payment)
        self.payments.append(payment)
    except PaymentValidationException as e:
        raise e
    except Exception as e:
        raise e

def CalculateCourseStatistics(self, course):
    enrolled_students_count = len(course.enrolled_students)
    total_payments = sum(payment.Amount for student in
course.enrolled_students for payment in student.payment_made)
    print('-' * 50)
    print(f"Statistics for Course {course.Course_Name}:")
    print(f"Number of Enrollments: {enrolled_students_count}")
    print(f"Total Payments: {total_payments}")

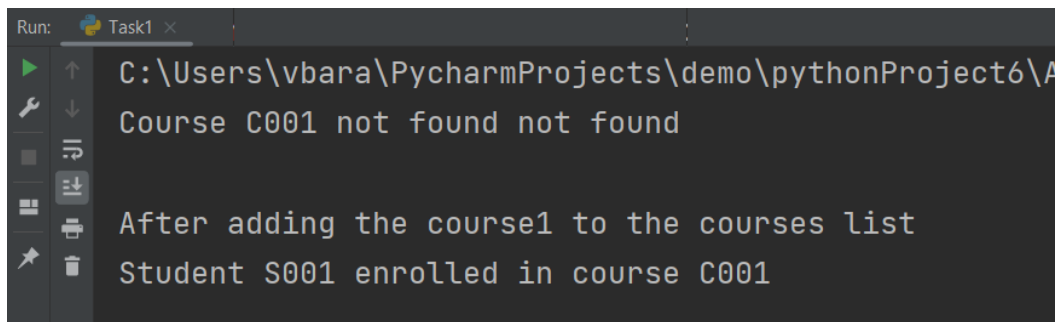
```

### Driver Code:

Trying to enrol student1 in course1. Since there is no particular course named course1 in the list courses[] of SIS() class it excepts an exception **CourseNotFoundException**. But after been appending the course1 to the courses list it is able to enrol student1.

```
sis = SIS()
student1 = Student(Student_ID="S001", First_Name="Alice", Last_Name="Smith",
DOB="1995-05-15", Email="alice.smith@example.com", Phone_NO="123-456-7890")
course1 = Course(Course_ID="C001", Course_Name="Mathematics",
Course_Code="MATH101", Instructor_Name="Instructor1")
try:
    sis.EnrollStudentInCourse(student1, course1)
except CourseNotFoundException as e:
    print(e)
print("\nAfter adding the course1 to the courses list")
sis.courses.append(course1)
sis.EnrollStudentInCourse(student1, course1)
```

### Output:



```
Run: Task1 x
C:\Users\vbara\PycharmProjects\demo\pythonProject6\A
Course C001 not found not found
After adding the course1 to the courses list
Student S001 enrolled in course C001
```

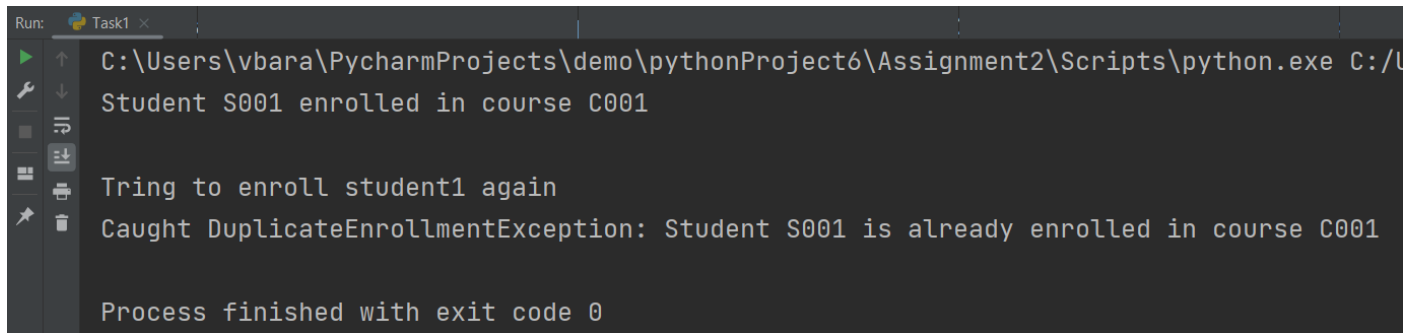
### Driver Code:

Trying to enroll the student in the course twice the try-except block catches this exception and prints a error message.

```
sis = SIS()
student1 = Student(Student_ID="S001", First_Name="Alice", Last_Name="Smith",
DOB="1995-05-15", Email="alice.smith@example.com", Phone_NO="123-456-7890")
course1 = Course(Course_ID="C001", Course_Name="Mathematics",
Course_Code="MATH101", Instructor_Name="Instructor1")
sis.courses.append(course1)
sis.EnrollStudentInCourse(student1, course1)
print("\nTring to enroll student1 again")
try:
    sis.EnrollStudentInCourse(student1, course1)
except DuplicateEnrollmentException as e:
    print(f"Caught DuplicateEnrollmentException: {e}")
```



## Output:



```
Run: Task1 x
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\Scripts\python.exe C:/U
Student S001 enrolled in course C001

Tring to enroll student1 again
Caught DuplicateEnrollmentException: Student S001 is already enrolled in course C001

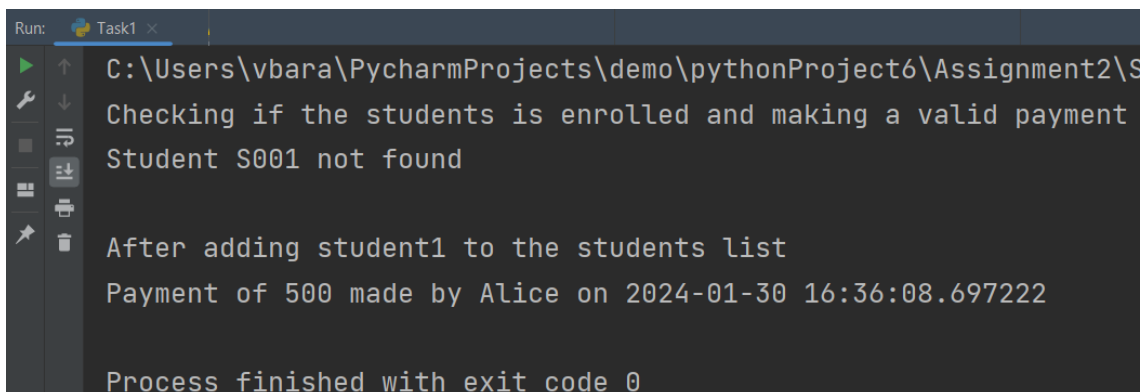
Process finished with exit code 0
```

## Driver Code:

This code attempts to record a payment of \$500 for the student using the RecordPayment function within a try-except block. The purpose of this block is to handle the scenario where a StudentNotFoundException occurs, indicating that the specified student is not present in the list of students maintained by the SIS. It the records the payment after been appending the student1 to the students list.

```
sis = SIS()
student1 = Student(Student_ID="S001", First_Name="Alice", Last_Name="Smith",
DOB="1995-05-15", Email="alice.smith@example.com", Phone_NO="123-456-7890")
course1 = Course(Course_ID="C001", Course_Name="Mathematics",
Course_Code="MATH101", Instructor_Name="Instructor1")
print("Checking if the students is enrolled and making a valid payment")
try:
    sis.RecordPayment(student1, amount=500, payment_date=datetime.now())
except StudentNotFoundException as e:
    print(e)
sis.students.append(student1)
print("\nAfter adding student1 to the students list")
sis.RecordPayment(student1, amount=500, payment_date=datetime.now())
```

## Output:



```
Run: Task1 x
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\S
Checking if the students is enrolled and making a valid payment
Student S001 not found

After adding student1 to the students list
Payment of 500 made by Alice on 2024-01-30 16:36:08.697222

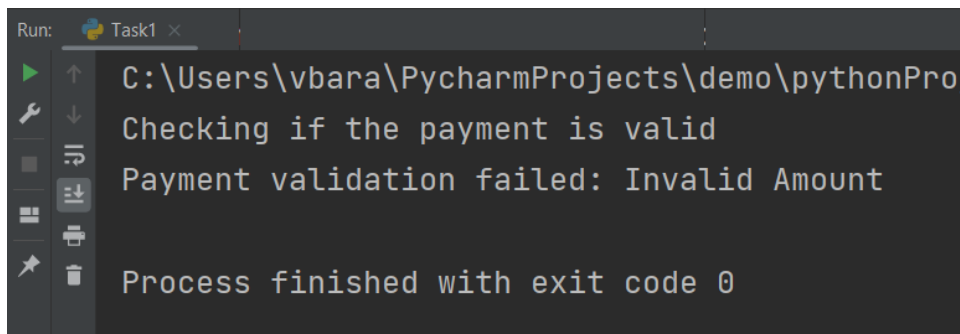
Process finished with exit code 0
```

### Driver Code:

student1 is manually added to the SIS. It then attempts to record a payment of -\$500 for the student1, catching and printing a PaymentValidationException as the the payment amount is invalid.

```
sis = SIS()
student1 = Student(Student_ID="S001", First_Name="Alice", Last_Name="Smith",
DOB="1995-05-15", Email="alice.smith@example.com", Phone_NO="123-456-7890")
course1 = Course(Course_ID="C001", Course_Name="Mathematics",
Course_Code="MATH101", Instructor_Name="Instructor1")
sis.students.append(student1)
print("Checking if the payment is valid")
try:
    sis.RecordPayment(student1, amount=-500, payment_date=datetime.now())
except PaymentValidationException as e:
    print(e)
```

### Output:



```
Run: Task1 x
C:\Users\vbara\PycharmProjects\demo\pythonPro
Checking if the payment is valid
Payment validation failed: Invalid Amount
Process finished with exit code 0
```

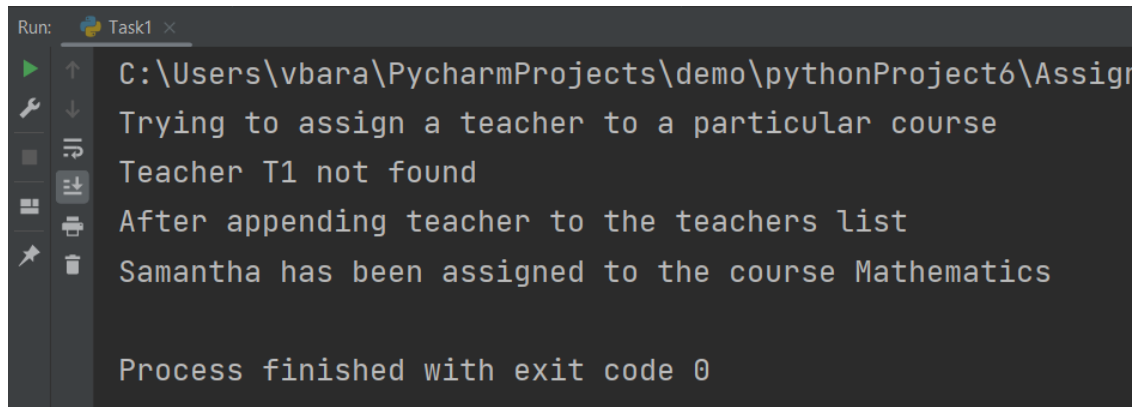
### Driver Code:

It attempts to assign the teacher to the course, catching and printing a TeacherNotFoundException as the teacher is not present in the system. After adding the teacher to the system, it successfully assigns the teacher to the course.

```
sis = SIS()
student1 = Student(Student_ID="S001", First_Name="Alice", Last_Name="Smith",
DOB="1995-05-15", Email="alice.smith@example.com", Phone_NO="123-456-7890")
course1 = Course(Course_ID="C001", Course_Name="Mathematics",
Course_Code="MATH101", Instructor_Name="Instructor1")
teacher1 = Teacher(Teacher_ID="T1", First_Name="Samantha", Last_Name="Akkineni",
Email="samantha@gmail.com")
print("Trying to assign a teacher to a particular course")
try:
    sis.AssignTeacherToCourse(teacher1, course1)
except TeacherNotFoundException as e:
    print(e)
sis.teachers.append(teacher1)
```

```
print("After appending teacher to the teachers list")
sis.AssignTeacherToCourse(teacher1,course1)
```

### Output:



```
Run: Task1 x
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignr
Trying to assign a teacher to a particular course
Teacher T1 not found
After appending teacher to the teachers list
Samantha has been assigned to the course Mathematics

Process finished with exit code 0
```

## Task 5: Collections

### Implement Collections:

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

These relationships are essential for the Student Information System (SIS) to track and manage student enrollments, teacher assignments, and payments accurately.

### Define Class-Level Data Structures

You will need class-level data structures within each class to maintain relationships. Here's how to define them for each class:

### Update Constructor(s)

In the constructors of your classes, initialize the list or collection properties to create empty collections when an object is instantiated.

Repeat this for the Course, Teacher, and Payment classes, where applicable.

### Student Class:

Create a list or collection property to store the student's enrollments. This property will hold references to Enrollment objects.

Example: `List<Enrollment> Enrollments { get; set; }`

```
class Student:
    def
__init__(self, Student_ID=None, First_Name=None, Last_Name=None, DOB=None, Email=None, Phone_NO=None):
    self.Student_ID=Student_ID
    self.First_Name = First_Name
    self.Last_Name = Last_Name
    self.DOB = DOB
```

```

self.Email = Email
self.Phone_NO = Phone_NO
self.enrolled_courses=[]
self.payment_made=[]
self.enrollments = []

```

### Course Class:

Create a list or collection property to store the course's enrollments. This property will hold references to Enrollment objects.

Example: List<Enrollment> Enrollments { get; set; }

```

class Course:

    def
__init__(self, Course_ID=None, Course_Name=None, Course_Code=None, Instructor_Name=
None):

    self.Course_ID=Course_ID
    self.Course_Name = Course_Name
    self.Course_Code = Course_Code
    self.Instructor_Name = Instructor_Name
    self.course_assigned = []
    self.enrolled_students = []
    self.enrollments = []

```

### Enrollment Class:

Include properties to hold references to both the Student and Course objects.

Example: Student Student { get; set; } and Course Course { get; set; }

```

class Enrollment(Student, Course):

    def __init__(self, Enrollment_ID=None, Student_ID=None, Course_ID=None,
Enrollment_Date=None):

        Student.__init__(self, Student_ID)
        Course.__init__(self, Course_ID)
        self.Enrollment_ID = Enrollment_ID
        self.Enrollment_Date = Enrollment_Date

```

### 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<Course> AssignedCourses { get; set; }

```
class Teacher:
    def
__init__(self, Teacher_ID=None, First_Name=None, Last_Name=None, Email=None):
    self.Teacher_ID=Teacher_ID
    self.First_Name = First_Name
    self.Last_Name = Last_Name
    self.Email = Email
    self.assigned_courses=[]
```

### Payment Class:

Include a property to hold a reference to the Student object.

Example: Student Student { get; set; }

```
class Payment(Student):
    def __init__(self, Payment_ID=None, Student_ID=None, Amount=None,
Payment_Date=None):
    self.Payment_ID = Payment_ID
    Student.__init__(self, Student_ID)
    self.Amount = Amount
    self.Payment_Date = Payment_Date
```

## Task 6: Create Methods for Managing Relationships

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

- `AddEnrollment(student, course, enrollmentDate)`: In the SIS class, create a method that adds an enrollment to both the Student's and Course's enrollment lists. Ensure the Enrollment object references the correct Student and Course.
- `AssignCourseToTeacher(course, teacher)`: In the SIS class, create a method to assign a course to a teacher. Add the course to the teacher's AssignedCourses list.
- `AddPayment(student, amount, paymentDate)`: In the SIS class, create a method that adds a payment to the Student's payment history. Ensure the Payment object references the correct Student.
- `GetEnrollmentsForStudent(student)`: In the SIS class, create a method to retrieve all enrollments for a specific student.
- `GetCoursesForTeacher(teacher)`: In the SIS class, create a method to retrieve all courses assigned to a specific teacher.

Source Code:

```
from Student import Student
from Course import Course
from Enrollment import Enrollment
from Teacher import Teacher
from Payment import Payment
from Exceptions import *
from datetime import datetime

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

    def EnrollStudentInCourse(self, student, course):
        self.students.append(student)
        self.courses.append(course)
        course.enrolled_students.append(student)
        if course not in self.courses:
            raise CourseNotFoundException(f"{course.Course_ID} not found")
        if not isinstance(student, Student):
            raise InvalidStudentDataException("Invalid student object
provided")
        if student in self.students:
            raise
DuplicateEnrollmentException(student.Student_ID, course.Course_ID)
        if not isinstance(course, Course):
            raise InvalidCourseDataException("Invalid course object provided")
```

```
try:
    student.EnrollInCourse(course)
    self.students.append(student)
except DuplicateEnrollmentException as e:
    print(f"Enrollment Error: {e}")
    raise e
except Exception as e:
    print(f"Unexpected Error: {e}")
    raise e

def AssignTeacherToCourse(self, teacher, course):
    self.courses.append(course)
    if teacher not in self.teachers:
        raise TeacherNotFoundException(teacher.Teacher_ID)
    if type(teacher) is not Teacher:
        raise InvalidTeacherDataException("Invalid teacher object
provided")

    if type(course) is not Course:
        raise InvalidCourseDataException("Invalid course object provided")

    try:
        course.AssignTeacher(teacher)
    except TeacherNotFoundException as e:
        raise e # Re-raise the specific exception
    except Exception as e:
        raise e # Re-raise any unexpected exceptions

def RecordPayment(self, student, amount, payment_date):
    if student not in self.students:
        raise StudentNotFoundException(student.Student_ID)
    if type(student) is not Student:
        raise InvalidStudentDataException("Invalid student object
provided")
    if amount < 0:
        raise PaymentValidationException("Invalid Amount")
    try:
        payment = Payment(Student_ID=student.Student_ID, Amount=amount,
Payment_Date=payment_date)
        student.MakePayment(payment)
```

```

        self.payments.append(payment)
    except PaymentValidationException as e:
        raise e
    except Exception as e:
        raise e

    def GeneratePaymentReport(self, student):
        print(f"Payment Report for Student {student.Student_ID} - {student.First_Name} {student.Last_Name}:")
        for payment in student.payment_made:
            print(f"Payment Amount: {payment.Amount}, Date: {payment.Payment_Date}")

    def GenerateEnrollmentReport(self, course):
        print(f"Enrollment Report for Course {course.Course_Name}:")
        for student in course.enrolled_students:
            print(f"Student ID: {student.Student_ID}, Name: {student.First_Name} {student.Last_Name}")

    def CalculateCourseStatistics(self, course):
        enrolled_students_count = len(course.enrolled_students)
        total_payments = sum(payment.Amount for student in course.enrolled_students for payment in student.payment_made)
        print('-' * 50)
        print(f"Statistics for Course {course.Course_Name}:")
        print(f"Number of Enrollments: {enrolled_students_count}")
        print(f"Total Payments: {total_payments}")

    def AddEnrollment(self, student, course):
        self.students.append(student)
        student.enrollments.append(student)
        course.enrollments.append(course)

    def AssignCourseToTeacher(self, course, teacher):
        self.teachers.append(teacher)
        course.AssignTeacher(teacher)

    def AddPayment(self, payment, student):
        self.payments.append(payment)
        student.MakePayment(payment)

```



```

def GetEnrollmentsForStudent(self, student):
    if student not in self.students:
        raise StudentNotFoundException(student.Student_ID)
    print(f"Enrollments for Student {student.First_Name}
{student.Last_Name}:")

    enrolled_courses=[course for course in student.enrolled_courses]
    for course in enrolled_courses:
        print(f"{course.Course_ID}-{course.Course_Name}-
{course.Course_Code}")

def GetCoursesForTeacher(self, teacher):
    if teacher not in self.teachers:
        raise TeacherNotFoundException(teacher.Teacher_ID)

    teacher_courses = [course for course in teacher.assigned_courses]
    print('-' * 50)
    print(f"Courses assigned to Teacher {teacher.First_Name}
{teacher.Last_Name}:")
    for course in teacher_courses:
        print(f"{course.Course_ID}-{course.Course_Name}")

```

### 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.

```

def Main():
    student1 = Student(Student_ID="S1", First_Name="Vijay", Last_Name="Joseph",
DOB="1974-06-22",
                        Email="vijay@gmail.com", Phone_NO="123-456-7890")
    course1 = Course(Course_ID="C1", Course_Name="Machine Learning",
Course_Code="AB101", Instructor_Name="Rajinikanth")
    course2 = Course(Course_ID="C2", Course_Name="Artificial Intelligence",
Course_Code="DC102",
                    Instructor_Name="Kamal Haasan")
    enrollment1 = Enrollment(Enrollment_ID="E1",
Student_ID=student1.Student_ID, Course_ID=course1.Course_ID,

```

```

        Enrollment_Date="2024-01-29")

teacher1 = Teacher(Teacher_ID="T1", First_Name="Samantha",
Last_Name="Akkineni", Email="samantha@gmail.com")

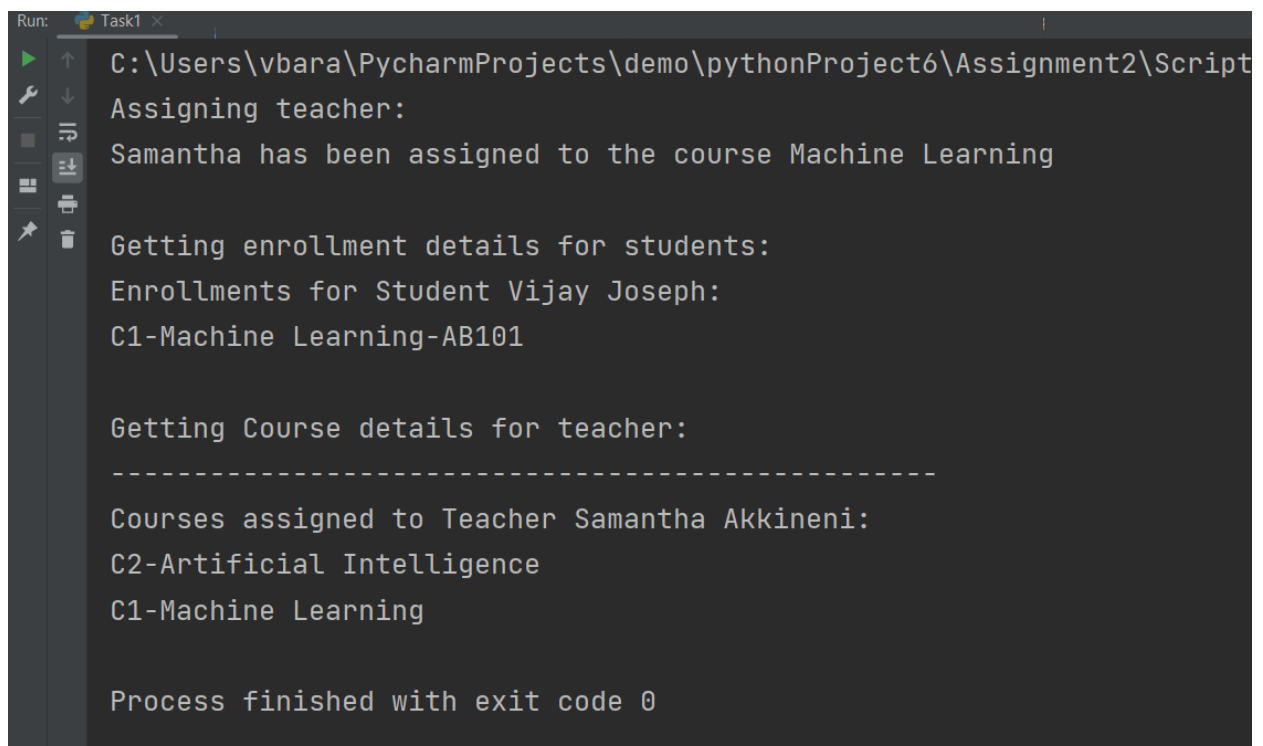
payment1 = Payment(Payment_ID="P1", Student_ID=student1.Student_ID,
Amount=500, Payment_Date=datetime.now())


sis = SIS()
sis.students.append(student1)
sis.teachers.append(teacher1)
student1.enrolled_courses.append(course1)
teacher1.assigned_courses.append(course2)
teacher1.assigned_courses.append(course1)
print("Assigning teacher: ")
sis.AssignCourseToTeacher(course1, teacher1)
sis.AddEnrollment(student1, course1)
print("\nGetting enrollment details for students: ")
sis.GetEnrollmentsForStudent(student1)
print("\nGetting Course details for teacher: ")
sis.GetCoursesForTeacher(teacher1)


if __name__ == "__main__":
    Main()

```

### Output:



```

Run: Task1 x
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\Script
Assigning teacher:
Samantha has been assigned to the course Machine Learning

Getting enrollment details for students:
Enrollments for Student Vijay Joseph:
C1-Machine Learning-AB101

Getting Course details for teacher:
-----
Courses assigned to Teacher Samantha Akkineni:
C2-Artificial Intelligence
C1-Machine Learning

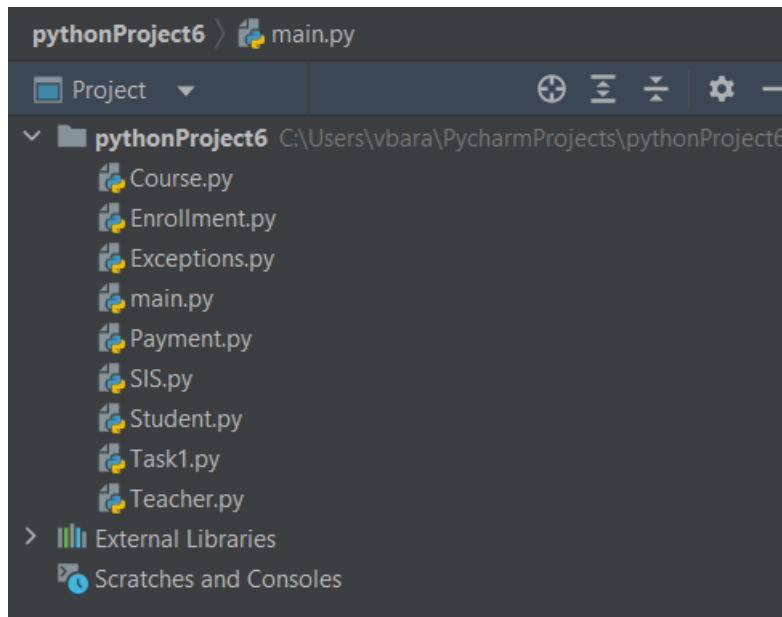
Process finished with exit code 0

```

## 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:



I have organized my code into separate files, following a modular approach for better maintainability. Each class (Student, Course, Enrollment, Teacher, Payment) is defined in its own file, ensuring a clear and organized project structure.

## 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.

### Source Code:

```
from Student import Student
from Course import Course
from Teacher import Teacher
from Payment import Payment
from SIS import SIS

def main():
    sis = SIS()

    student4 = Student(Student_ID="TN001", First_Name="Arvind",
Last_Name="Kumar", DOB="1993-05-12",
                        Email="arvind.kumar@example.com", Phone_NO="9876543210")
    student5 = Student(Student_ID="TN002", First_Name="Kavya", Last_Name="Raj",
DOB="1990-08-25",
                        Email="kavya.raj@example.com", Phone_NO="8765432109")

    course5 = Course(Course_ID="TN101", Course_Name="Tamil Literature",
Course_Code="TAML101",
                    Instructor_Name="Ramachandran")
    course6 = Course(Course_ID="TN102", Course_Name="History",
Course_Code="HISTTN101",
                    Instructor_Name="Prof. Senthil Kumar")

    teacher5 = Teacher(Teacher_ID="TN01", First_Name="Dr.Sanjay",
Last_Name="Ramachandran", Email="dr.ram@gmail.com")
    teacher6 = Teacher(Teacher_ID="TN02", First_Name="Prof.Senthil ",
Last_Name="Kumar",
                    Email="prof.senthil@gmail.com")

    payment4 = Payment(Payment_ID="TNP001", Student_ID="TN001", Amount=800,
Payment_Date="2024-02-05")
    payment5 = Payment(Payment_ID="TNP002", Student_ID="TN002", Amount=700,
Payment_Date="2024-02-08")

    course5.enrolled_students.append(student5)
    sis.students.append(student4)
    sis.students.append(student5)
```

```
sis.teachers.append(teacher5)
sis.teachers.append(teacher6)
student4.enrolled_courses.append(course5)
teacher5.assigned_courses.append(course5)
teacher5.assigned_courses.append(course6)

print("\nAssigning teachers to courses: ")
sis.AssignTeacherToCourse(teacher5, course5)
sis.AssignTeacherToCourse(teacher6, course6)
print("\nGetting Enrollment Details for students: ")
sis.GetEnrollmentsForStudent(student4)
print("\nGetting the courses assigned to a particular teacher: ")
sis.GetCoursesForTeacher(teacher5)
print("\nMaking payments: ")
sis.RecordPayment(student5, 500, "2024-02-01")
sis.RecordPayment(student4, 600, "2024-02-02")

print("\nCalculated course statistics: ")
sis.CalculateCourseStatistics(course5)
sis.AddEnrollment(student5, course6)
print("\nAssigning courses to a particular teacher")
sis.AssignCourseToTeacher(course5, teacher6)
print("\nGenerating Payment report for students:")
sis.GeneratePaymentReport(student4)
print("\nGenerating Enrollment report for Courses:")
sis.GenerateEnrollmentReport(course5)

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

## Output:

```
Run: main x
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\Sc
Assigning teachers to courses:
Dr.Sanjay has been assigned to the course Tamil Literature
Prof.Senthil has been assigned to the course History

Getting Enrollment Details for students:
Enrollments for Student Arvind Kumar:
TN101-Tamil Literature-TAML101

Getting the courses assigned to a particular teacher:
-----
Courses assigned to Teacher Dr.Sanjay Ramachandran:
TN101-Tamil Literature
TN102-History

Making payments:
Payment of 500 made by Kavya on 2024-02-01
Payment of 600 made by Arvind on 2024-02-02

Calculated course statistics:
-----
Statistics for Course Tamil Literature:
Number of Enrollments: 1
Total Payments: 500

Assigning courses to a particular teacher
Prof.Senthil has been assigned to the course Tamil Literature

Generating Payment report for students:
Payment Report for Student TN001 - Arvind Kumar:
Payment Amount: 600, Date: 2024-02-02

Generating Enrollment report for Courses:
Enrollment Report for Course Tamil Literature:
Student ID: TN002, Name: Kavya Raj

Process finished with exit code 0
```

## Task 7: Database Connectivity

### Database Initialization:

Implement a method that initializes a database connection and creates tables for storing student, course, enrollment, teacher, and payment information. Create SQL scripts or use code-first migration to create tables with appropriate schemas for your SIS.

### Source Code:

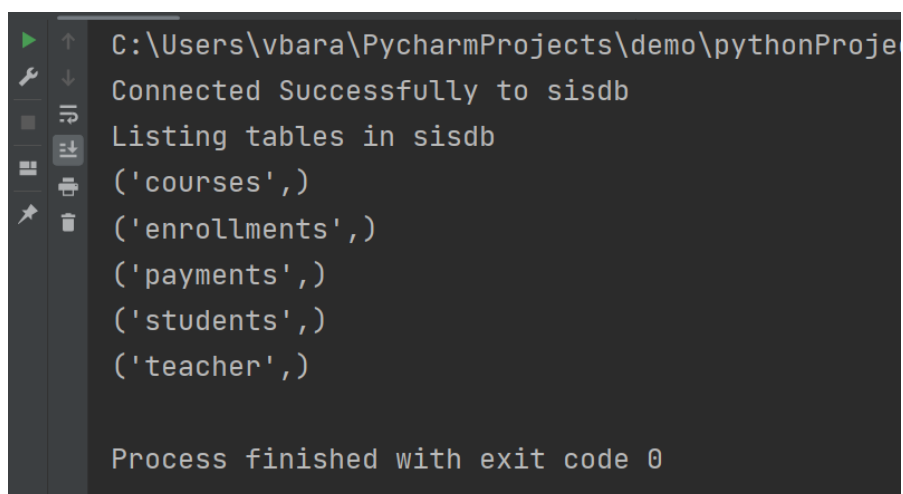
```
import mysql.connector
try:

conn=mysql.connector.connect(host='localhost',user='root',passwd='root',database='sisdb',port='3306')

    if conn:

        print(f"Connected Successfully to {conn.database}")
except Exception as e:
    print(e)
cur=conn.cursor()
cur.execute("show tables")
tables=cur.fetchall()
print(f"Listing tables in {conn.database}")
for i in tables:
    print(i)
```

### Output:



```
C:\Users\vbara\PycharmProjects\demo\pythonProje
Connected Successfully to sisdb
Listing tables in sisdb
('courses',)
('enrollments',)
('payments',)
('students',)
('teacher',)

Process finished with exit code 0
```

### Data Retrieval:

Implement methods to retrieve data from the database. Users should be able to request information about students, courses, enrollments, teachers, or payments. Ensure that the data retrieval methods handle exceptions and edge cases gracefully.

### Source Code:

```
import mysql.connector
try:

conn=mysql.connector.connect(host='localhost',user='root',passwd='root',database='sisdb',port='3306')
    if conn:
        print(f"Connected Successfully to {conn.database}")
except Exception as e:
    print(e)
cur=conn.cursor()
cur.execute("show tables")
tables=cur.fetchall()
print(f"Listing tables: ")
for i in tables:
    print(i)
#Data Retrieval
curl=conn.cursor()
curl.execute("select * from courses")
print(f"Listing courses: ")
courses=curl.fetchall()
for j in courses:
    print(j)
cur2=conn.cursor()
cur2.execute("select first_name,course_id from students s join enrollments e on e.student_id=s.student_id")
enrollments=cur2.fetchall()
print("Listing The students who have been enrolled: ")
for stud in enrollments:
    print(stud)
print("\nTrying to execute query with wrong table name:")
try:
    cur3=conn.cursor()
    cur3.execute("select s.first_name,p.amount,p.payment_date from payment p join students s on p.student_id=s.student_id")
    payment_details=cur3.fetchall()
```



```

print("Payment details for students: ")
for pay in payment_details:
    print(pay)
except Exception as e:
    print(e)

```

## Output:

```

Run: Task1
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignr
Connected Successfully to sisdb
Listing tables:
('courses',)
('enrollments',)
('payments',)
('students',)
('teacher',)
Listing courses:
(101, 'Mathematics', 3, 1)
(102, 'Physics', 4, 2)
(103, 'Chemistry', 3, 3)
(104, 'Biology', 4, 5)
(105, 'Computer Science', 3, 6)
(106, 'History', 3, 7)
(107, 'Literature', 3, 8)
(108, 'Economics', 4, 2)
(109, 'Psychology', 3, 9)
(110, 'Music', 2, 9)
(111, 'Physical Education', 0, None)

Listing The students who have been enrolled:
('John', 101)
('Jane', 102)
('Michael', 103)
('Emily', 104)
('Christopher', 105)
('Anna', 106)
('Olivia', 108)
('William', 109)
('Sophia', 110)
('Christopher', 101)
('John', None)
('Emily', None)
('Emily', 106)
Trying to execute query with wrong table name:
1146 (42S02): Table 'sisdb.payment' doesn't exist

Process finished with exit code 0

```

### Data Insertion and Updating:

Implement methods to insert new data (e.g., enrollments, payments) into the database and update existing data (e.g., student information). Use methods to perform data insertion and updating.

Implement validation checks to ensure data integrity and handle any errors during these operations.

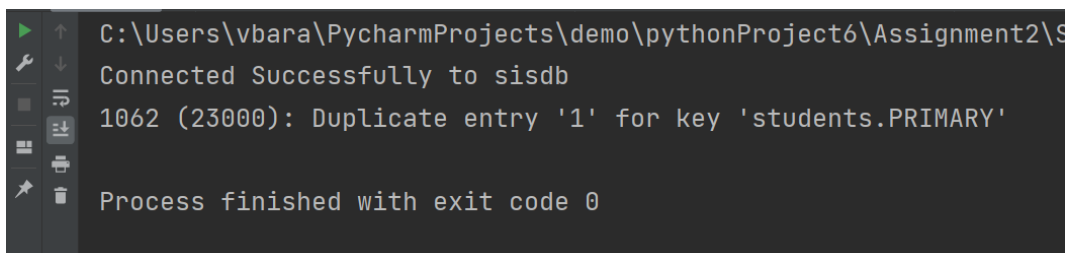
#### Insertion:

##### Case 1:

Trying to insert into students table with a duplicate entry to except the exception.

```
#insertion and Updation
try:
    ins=conn.cursor()
    ins.execute("insert into students values (1,'Mogesh','Kumar','2003-12-01','mogispensor@gmail.com','82736728') ")
    print("Inserted Sucessfully")
except Exception as e:
    print(e)
```

#### Output:



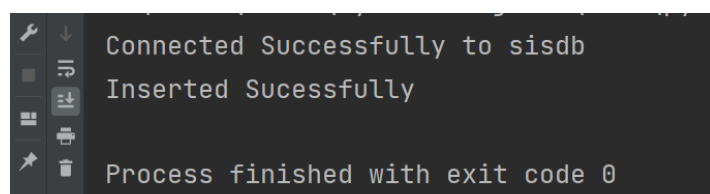
```
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\S
Connected Successfully to sisdb
1062 (23000): Duplicate entry '1' for key 'students.PRIMARY'
Process finished with exit code 0
```

##### Case 2:

As there is no duplicate entry or no error in the query it successful inserted the record into the students table.

```
try:
    ins=conn.cursor()
    ins.execute("insert into students values (16,'Koki','Kumar','2003-01-01','kokikumar@gmail.com','82736728') ")
    conn.commit()
    print("Inserted Sucessfully")
except Exception as e:
    print(e)
```

#### Output:



```
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\S
Connected Successfully to sisdb
Inserted Sucessfully
Process finished with exit code 0
```

It has been successfully inserted into the students table of sis database:



	student_id	first_name	last_name	date_of_birth	email	phone_number
▶	1	John	Doe	1990-01-15	john.doe@gmail.com	1234567890
	2	Jane	Smith	1992-05-22	jane.smith@gmail.com	9876543210
	3	Michael	Johnson	1991-08-10	michael.j@gmail.com	5555555555
	4	Emily	Davis	1993-03-30	emily.d@gmail.com	1111111111
	5	Christopher	Lee	1994-11-18	chris.lee@gmail.com	9999999999
	6	Anna	Wang	1995-07-05	anna.wang@gmail.com	chris.lee@gmail.com
	8	Olivia	Brown	1993-12-02	olivia.b@gmail.com	6666666666
	9	William	Wilson	1990-04-25	william.w@gmail.com	4444444444
	10	Sophia	Garcia	1995-01-08	sophia.g@gmail.com	2222222222
	11	John	Doe	1995-08-15	john.doe@example.com	1234567890
	16	Koki	Kumar	2003-01-01	kokikumar@gmail.com	82736728
*		NULL	NULL	NULL	NULL	NULL

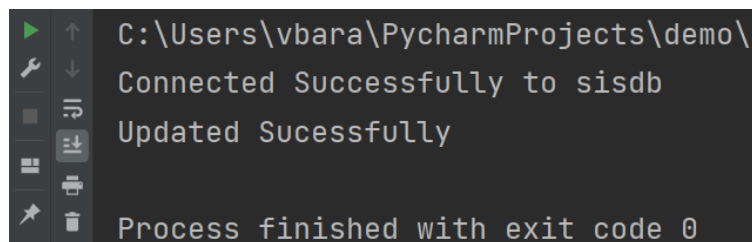
**Updating:**

**Source Code:**

```
import mysql.connector
try:

conn=mysql.connector.connect(host='localhost',user='root',passwd='root',database='sisdb',port='3306')
    if conn:
        print(f"Connected Successfully to {conn.database}")
except Exception as e:
    print(e)
# Updation
try:
    ins=conn.cursor()
    ins.execute("update teacher set email='manjojmaddy@gmail.com' where teacher_id=5")
    conn.commit()
    print("Updated Sucessfully")
except Exception as e:
    print(e)
```

**Output:**



```
C:\Users\vbara\PycharmProjects\demo\p
Connected Successfully to sisdb
Updated Sucessfully
Process finished with exit code 0
```

- Before updating the email of teacher with teacher\_id=5

	teacher_id	first_name	last_name	email
▶	1	Arvind	Kumar	arvind12gmail.com
	2	Deepa	Devi	deepa@gmail.com
	3	Ganesh	Subramanian	ganesh@gmail.com
	4	Kavitha	Raj	kavitha@gmail.com
	5	Manoj	Chellappan	manoj@gmail.com
	6	Nithya	Ve Venkataraman	nithya@gmail.com
	7	Prakash	Balasubramanian	prakash@gmail.com
	8	Rekha	Shankar	itzmerehks@hexa.com
	9	Suresh	Ramalingam	su@gmail.com
	10	Thirumalai	Muthusamy	thiru@gmail.com
*	NULL	NULL	NULL	NULL

- After updating the email of teacher with teacher\_id=5

Result Grid		Filter Rows:		Edit:				Exp
	teacher_id	first_name	last_name	email				
▶	1	Arvind	Kumar	arvind12gmail.com				
	2	Deepa	Devi	deepa@gmail.com				
	3	Ganesh	Subramanian	ganesh@gmail.com				
	4	Kavitha	Raj	kavitha@gmail.com				
	5	Manoj	Chellappan	manojmaddy@gmail.com				
	6	Nithya	Venkataraman	nithya@gmail.com				
	7	Prakash	Balasubramanian	prakash@gmail.com				
	8	Rekha	Shankar	itzmerehks@hexa.com				
	9	Suresh	Ramalingam	su@gmail.com				
	10	Thirumalai	Muthusamy	thiru@gmail.com				
*	NULL	NULL	NULL	NULL				

### Transaction Management:

Implement methods for handling database transactions when enrolling students, assigning teachers, or recording payments. Transactions should be atomic and maintain data integrity. Use database transactions to ensure that multiple related operations either all succeed or all fail. Implement error handling and rollback mechanisms in case of transaction failures.

#### Case 1:

Trying to record a payment history with student who have not enrolled in any courses:

#### Source Code:

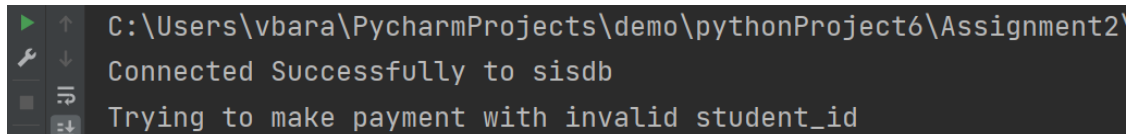
```
import mysql.connector
try:
    conn = mysql.connector.connect(host='localhost', user='root',
passwd='root', database='sisdb', port='3306')
    if conn:
        print(f"Connected Successfully to {conn.database}")
except Exception as e:
    print(e)
# Updation
```

```

try:
    ins = conn.cursor()
    ins.execute("insert into payments values(312,19,3000,'2024-01-27')")
    conn.commit()
    print("Updated Successfully")
except Exception :
    print(f"Trying to make payment with invalid student_id")
    conn.rollback()

```

**Output:**



```

C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\Script1.py
Connected Successfully to sisdb
Trying to make payment with invalid student_id

```

**Case 2:**

Trying to enrol a student in the enrolments table where the course does not exist :

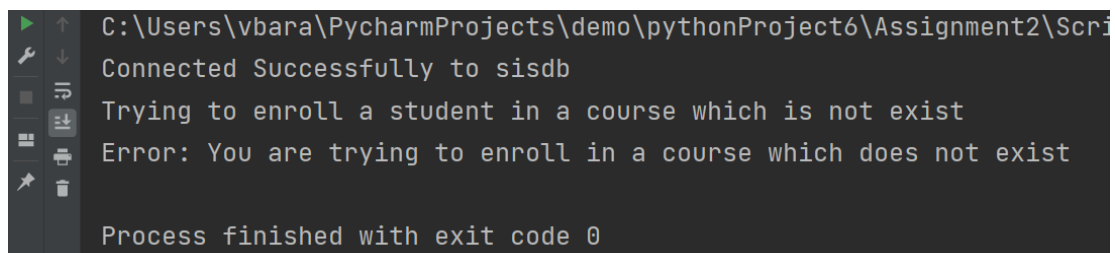
**Source Code:**

```

import mysql.connector
try:
    conn = mysql.connector.connect(host='localhost', user='root',
passwd='root', database='sisdb', port='3306')
    if conn:
        print(f"Connected Successfully to {conn.database}")
except Exception as e:
    print(e)
# Updation
print("Trying to enroll a student in a course which is not exist")
try:
    ins = conn.cursor()
    ins.execute("insert into enrollments values(215,5,199,'2024-01-27')")
    conn.commit()
    print("Updated Successfully")
except Exception :
    print(f"Error: You are trying to enroll in a course which does not exist")
    conn.rollback()

```

**Output:**



```

C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\Script2.py
Connected Successfully to sisdb
Trying to enroll a student in a course which is not exist
Error: You are trying to enroll in a course which does not exist
Process finished with exit code 0

```

### Dynamic Query Builder:

Implement a dynamic query builder that allows users to construct and execute custom SQL queries to retrieve specific data from the database. Users should be able to specify columns, conditions, and sorting criteria. Create a query builder method that dynamically generates SQL queries based on user input. Implement parameterization and sanitation of user inputs to prevent SQL injection.

### Source Code:

```
import mysql.connector

def build_query(table=None, columns=None, conditions=None, order_by=None):
    query=f"SELECT {' , '.join(columns) if columns else '*'} FROM {table}"
    if conditions:
        query += f" WHERE {' AND '.join(conditions)}"

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

def execute_query(conn, query):
    cur=conn.cursor()
    cur.execute(query)
    res=cur.fetchall()
    return res

try:
    con = mysql.connector.connect(host='localhost', user='root', passwd='root',
    database='sisdb', port='3306')

    query1=build_query(table='students')
    print("Trying to list all the students details in the student table")
    print()
    res=execute_query(con, query1)
    for i in res:
        print(i)

    query2=build_query(
        table='payments',
        columns=['student_id', 'payment_id'],
        conditions=['amount>500']
    )

    print('=' * 90)
    print("Trying to retrieve payment details for the students id who have
    payed above 500")
    print()
    res1=execute_query(con, query2)

    for i in res1:
        print(i)
```

```
except Exception as e:
    print("Error executing: ",e)
finally:
    con.close()
```

### Output:

```
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\Scripts\python.exe C:/Users/vb
Trying to list all the students details in the student table:

(1, 'John', 'Doe', datetime.date(1990, 1, 15), 'john.doe@gmail.com', 1234567890)
(2, 'Jane', 'Smith', datetime.date(1992, 5, 22), 'jane.smith@gmail.com', 9876543210)
(3, 'Michael', 'Johnson', datetime.date(1991, 8, 10), 'michael.j@gmail.com', 5555555555)
(4, 'Emily', 'Davis', datetime.date(1993, 3, 30), 'emily.d@gmail.com', 1111111111)
(5, 'Christopher', 'Lee', datetime.date(1994, 11, 18), 'chris.lee@gmail.com', 9999999999)
(6, 'Anna', 'Wang', datetime.date(1995, 7, 5), 'anna.wang@gmail.com', 7777777777)
(8, 'Olivia', 'Brown', datetime.date(1993, 12, 2), 'olivia.b@gmail.com', 6666666666)
(9, 'William', 'Wilson', datetime.date(1990, 4, 25), 'william.w@gmail.com', 4444444444)
(10, 'Sophia', 'Garcia', datetime.date(1995, 1, 8), 'sophia.g@gmail.com', 2222222222)
(11, 'John', 'Doe', datetime.date(1995, 8, 15), 'john.doe@example.com', 1234567890)
(16, 'Koki', 'Kumar', datetime.date(2003, 1, 1), 'kokikumar@gmail.com', 82736728)
=====

=====
Trying to retrieve payment details for the students id who have payed above 500
(2, 302)
(3, 303)
(4, 304)
(5, 305)
(6, 306)
(8, 308)
(10, 310)
(5, 311)
```

### Task 8: Student Enrollment

In this task, a new student, John Doe, is enrolling in the SIS. The system needs to record John's information, including his personal details, and enroll him in a few courses. Database connectivity is required to store this information.

John Doe's details:

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

John is enrolling in the following courses:

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

The system should perform the following tasks:

- Create a new student record in the database.
- Enroll John in the specified courses by creating enrollment records in the database.

**Source Code:**

```
import mysql.connector
from datetime import datetime, date

def add_student(conn, id, first_name, last_name, birth_date, email, phone):
    try:
        cursor = conn.cursor()
        cursor.execute("INSERT INTO students VALUES (%s, %s, %s, %s, %s, %s)",
            (id, first_name, last_name, birth_date, email, phone))
        print("Data inserted successfully...")
        conn.commit()
    except Exception as e:
        print(f"Error during student enrollment: {e}")
        conn.rollback()

def enrol_student(conn, student_id, courses):
    try:
        cursor = conn.cursor()
        for course in courses:
            # Enroll the student in each specified course
            cursor.execute("INSERT INTO enrollments (student_id, course_id)
VALUES (%s, %s)",
                (student_id, course))
        print(f"Student enrolled in courses successfully.")
        conn.commit()
    except Exception as e:
        print(f"Error during course enrollment: {e}")
        conn.rollback()

try:
    conn = mysql.connector.connect(host='localhost', user='root',
passwd='root', database='sisdb', port='3306')

    id=20
    first_name = 'John'
    last_name = 'Doe'
    birth_date = date(1995, 8, 15)
    email = 'john.doe@example.com'
    phone = '123-456-7890'
    add_student(conn, id, first_name, last_name, birth_date, email, phone)
    course_id=[101,112]
```



```

enrol_student(conn,id,course_id)
except Exception as e:
    print(f"Error: {e}")

```

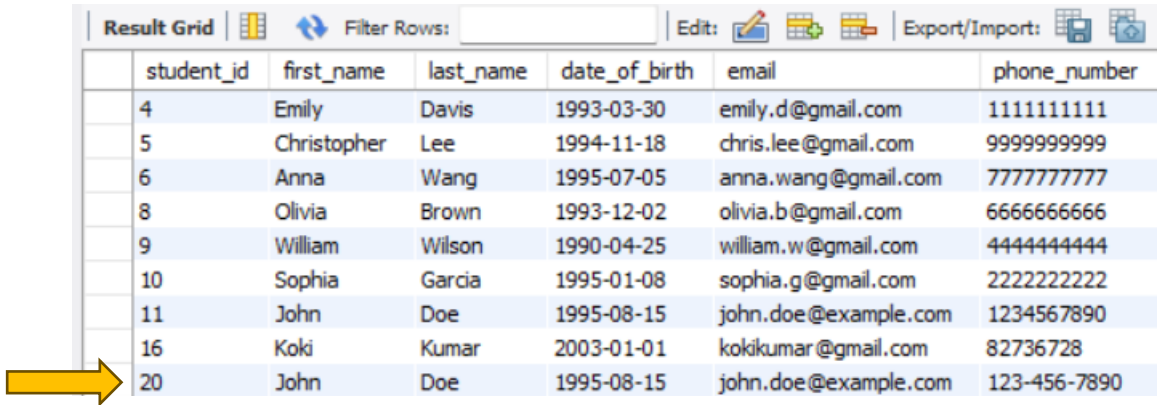
Output:

```

C:\Users\vbara\PycharmProjects\demo\pythonProject6\A
Data inserted successfully...
Student enrolled in courses successfully.

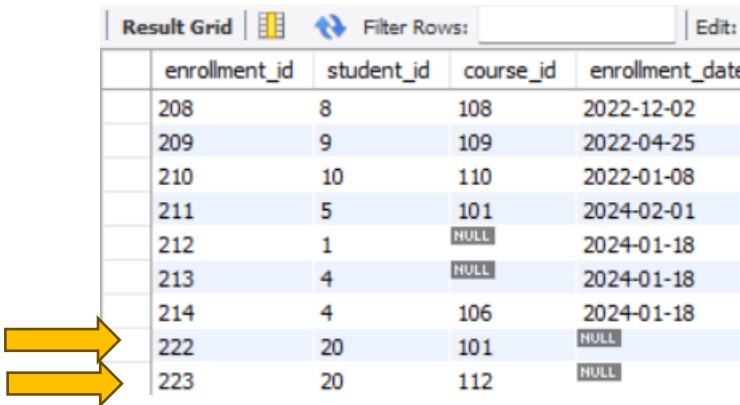
```

Student record has been successfully inserted into the students table:



	student_id	first_name	last_name	date_of_birth	email	phone_number
	4	Emily	Davis	1993-03-30	emily.d@gmail.com	1111111111
	5	Christopher	Lee	1994-11-18	chris.lee@gmail.com	9999999999
	6	Anna	Wang	1995-07-05	anna.wang@gmail.com	7777777777
	8	Olivia	Brown	1993-12-02	olivia.b@gmail.com	6666666666
	9	William	Wilson	1990-04-25	william.w@gmail.com	4444444444
	10	Sophia	Garcia	1995-01-08	sophia.g@gmail.com	2222222222
	11	John	Doe	1995-08-15	john.doe@example.com	1234567890
	16	Koki	Kumar	2003-01-01	kokikumar@gmail.com	82736728
	20	John	Doe	1995-08-15	john.doe@example.com	123-456-7890

The same student have enrolled in in the specified courses by creating enrolment records in the database



	enrollment_id	student_id	course_id	enrollment_date
	208	8	108	2022-12-02
	209	9	109	2022-04-25
	210	10	110	2022-01-08
	211	5	101	2024-02-01
	212	1	NULL	2024-01-18
	213	4	NULL	2024-01-18
	214	4	106	2024-01-18
	222	20	101	NULL
	223	20	112	NULL

### Task 9: Teacher Assignment

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

Teacher's Details:

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

Course to be assigned:

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

The system should perform the following tasks:

- Retrieve the course record from the database based on the course code.
- Assign Sarah Smith as the instructor for the course.
- Update the course record in the database with the new instructor information.

**Source Code:**

```
import mysql.connector

def get_course(con, course_id):
    cur = con.cursor()
    cur.execute("SELECT * FROM Courses WHERE course_id=%s", (course_id,))
    res = cur.fetchone()
    print("Retrieving the course record from the database based on the course code")
    print(res)

def add_teacher(con, teacher_id, first_name, last_name, email):
    try:
        cur = con.cursor()
        cur.execute("INSERT INTO Teacher VALUES (%s, %s, %s, %s)", (teacher_id, first_name, last_name, email))
        print("Data inserted successfully...")
        con.commit()
    except Exception as e:
        print(e)

def assign_teacher(con, teacher_id, course_id):
    cur2 = con.cursor()
    cur2.execute("UPDATE Courses SET teacher_id=%s WHERE course_id=%s", (teacher_id, course_id))
    print(f"Teacher {teacher_id} has been assigned to the Course {course_id}")
    con.commit()

try:
    con = mysql.connector.connect(host='localhost', user='root', passwd='root', database='sisdb', port='3306')
    get_course(con, course_id=302)
    add_teacher(con, teacher_id=19,
                first_name='Sarah',
                last_name='Smith',
                email='sarah.smith@example.com')
    assign_teacher(con, teacher_id=19, course_id=302)
```

```
except Exception as e:
    print(e)
```

#### Output:

```
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\Scripts\pyt
Retrieving the course record from the database based on the course code
(302, 'Advanced Database Management', 7, None)
```

```
Data inserted Successfully...
Teacher 19 has been assigned to the Course 302
Process finished with exit code 0
```

Inserted Sarah Smith as the instructor into the teacher table.

teacher_id	first_name	last_name	email
3	Ganesh	Subramanian	ganesh@gmail.com
4	Kavitha	Raj	kavitha@gmail.com
5	Manoj	Chellappan	manojmaddy@gmail.com
6	Nithya	Venkataraman	nithya@gmail.com
7	Prakash	Balasubramanian	prakash@gmail.com
8	Rekha	Shankar	itzmerehks@hexa.com
9	Suresh	Ramalingam	su@gmail.com
10	Thirumalai	Muthusamy	thiru@gmail.com
19	Sarah	Smith	sarah.smith@example.com

Assigned Sarah Smith as the instructor for the course and updated the course record in the database with the new instructor information.

course_id	course_name	credits	teacher_id
105	Computer Science	3	6
106	History	3	7
107	Literature	3	8
108	Economics	4	2
109	Psychology	3	9
110	Music	2	9
111	Physical Education	0	NULL
112	Introduction to P...	5	5
302	Advanced Datab...	7	19

#### Task 10: Payment Record

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

Jane Johnson's details:

- Student ID: 101
- Payment Amount: \$500.00

- Payment Date: 2023-04-10

The system should perform the following tasks:

- Retrieve Jane Johnson's student record from the database based on her student ID.
- Record the payment information in the database, associating it with Jane's student record.

#### Source Code:

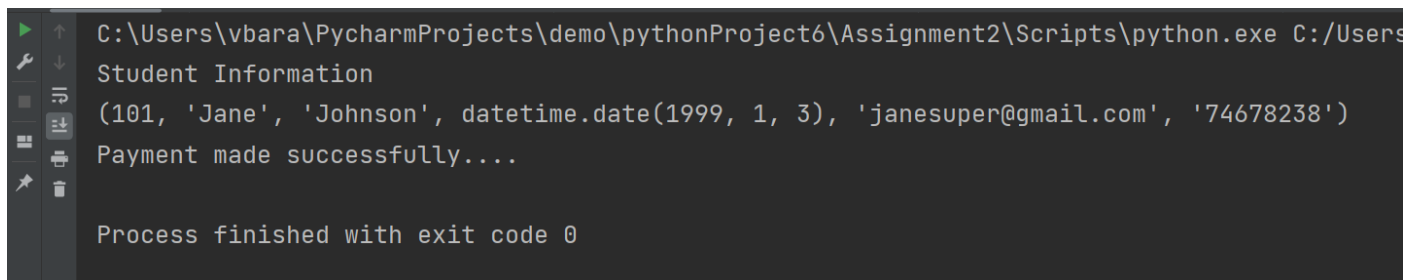
```
import mysql.connector

def retrieve_info(con, student_id):
    cur = con.cursor()
    cur.execute("SELECT * FROM Students WHERE student_id=%s", (student_id,))
    res = cur.fetchall()
    print("Student Information")
    for i in res:
        print(i)

def make_payment(con, student_id, amount, date):
    curr=con.cursor()
    curr.execute("INSERT INTO Payments (student_id,amount,payment_date) values (%s, %s, %s)", (student_id,amount,date))
    print("Payment made successfully....")
    con.commit()

try:
    con = mysql.connector.connect(host='localhost', user='root', passwd='root',
    database='sisdb', port='3306')
    retrieve_info(con, student_id=101)
    make_payment(con, student_id=101, amount=500, date='2023-04-10')
except Exception as e:
    print(e)
```

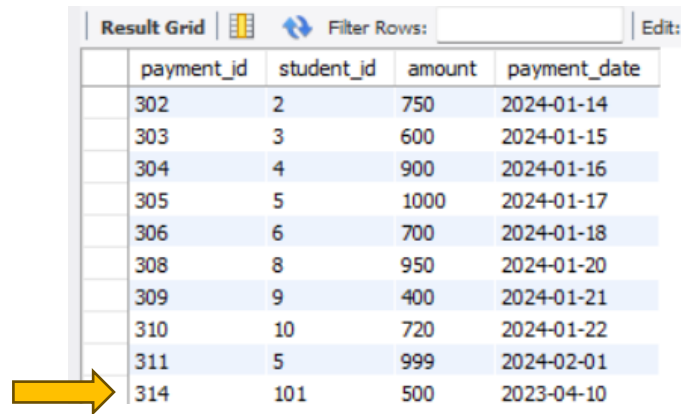
#### Output:



```
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\Scripts\python.exe C:/Users
Student Information
(101, 'Jane', 'Johnson', datetime.date(1999, 1, 3), 'janesuper@gmail.com', '74678238')
Payment made successfully....

Process finished with exit code 0
```

Recorded the payment information with the payment of 500 on 2023-04-10 in the database, associating it with Jane's student record (student\_id):



	payment_id	student_id	amount	payment_date
	302	2	750	2024-01-14
	303	3	600	2024-01-15
	304	4	900	2024-01-16
	305	5	1000	2024-01-17
	306	6	700	2024-01-18
	308	8	950	2024-01-20
	309	9	400	2024-01-21
	310	10	720	2024-01-22
	311	5	999	2024-02-01
	314	101	500	2023-04-10

### Task 11: Enrollment Report Generation

In this task, an administrator requests an enrollment report for a specific course, "Computer Science 101." The system needs to retrieve enrollment information from the database and generate a report.

Course to generate the report for:

- Course Name: Computer Science 101

The system should perform the following tasks:

- Retrieve enrollment records from the database for the specified course.
- Generate an enrollment report listing all students enrolled in Computer Science 101.
- Display or save the report for the administrator

#### Source Code:

```
import mysql.connector

def generate_enrollment_report(conn, course_name):
    try:
        cursor = conn.cursor(dictionary=True)

        cursor.execute("SELECT students.student_id, students.first_name,
students.last_name, students.email "
                        "FROM students "
                        "JOIN enrollments ON students.student_id =
enrollments.student_id "
                        "JOIN courses ON enrollments.course_id =
courses.course_id "
                        "WHERE courses.course_name = %s", (course_name,))

        enrollment_records = cursor.fetchall()

        print(f"Enrollment Report for Course: {course_name}")
        print("-----")
```

```

        print("Student ID | First Name | Last Name | Email")
        print("-----")
        for record in enrollment_records:
            print(f"{record['student_id']} | {record['first_name']} | {record['last_name']} | {record['email']}")
            report_content=''

        for record in enrollment_records:
            report_content += f"{record['student_id']} | {record['first_name']} | {record['last_name']} | {record['email']}\n"

        with open('records', 'w') as file:
            file.write(report_content)

    except Exception as e:
        print(f"Error during report generation: {e}")

def retrieve_enrollment(con, course_id):
    cur=con.cursor(dictionary=True)
    cur.execute("SELECT * FROM Enrollments WHERE course_id=%s", (course_id,))
    res=cur.fetchall()
    print(f"Retrieving enrollment records for the specified course {course_id}")
    print("-----")
    print("Enroll ID | Stud ID | Course ID| Enrollment Date")
    print("-----")
    for i in res:
        print(f"{i['enrollment_id']} | {i['student_id']} | {i['course_id']} | {i['enrollment_date']}")

try:
    conn = mysql.connector.connect(host='localhost', user='root', passwd='root', database='sisdb', port='3306')
    course_name_to_report = "Computer Science"
    generate_enrollment_report(conn, course_name_to_report)
    retrieve_enrollment(conn,101)
except Exception as e:
    print(f"Error: {e}")

```

## Output:

```
C:\Users\vbara\PycharmProjects\demo\pythonProject6\Assignment2\
Enrollment Report for Course: Computer Science
-----
Student ID | First Name | Last Name | Email
-----
5 | Christopher | Lee | chris.lee@gmail.com

Retrieving enrollment records for the specified course 101
-----
Enroll ID | Stud ID | Course ID| Enrollment Date
-----
201 | 1 | 101 | 2022-01-15
211 | 5 | 101 | 2024-02-01
222 | 20 | 101 | None

Process finished with exit code 0
```

Saving the report for the administrator in the file records. In the records file all the reports are stored.

```
pythonProject6 > records
- records x
1 5 | Christopher | Lee | chris.lee@gmail.com
```