

ASSIGNMENT 2 PYTHON

NAME :- SUSHANT KUMAR SINGH

Project:- Student Information System (SIS)

Student Information System (SIS)

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 | 2: Define Classes | Methods

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

```
Version control
main.py DatabaseConnector.py student.py x course.py enrollment.py teacher.py payment.py

1 from enrollment import Enrollment
2 from datetime import datetime
3 from payment import Payment
4
5 class Student:
6     def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number):
7         self.student_id = student_id
8         self.first_name = first_name
9         self.last_name = last_name
10        self.date_of_birth = date_of_birth
11        self.email = email
12        self.phone_number = phone_number
13        self.enrollments = []
14        self.payments = []
15
16    def enroll_in_course(self, course):
17        enrollment_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
18        enrollment = Enrollment(len(self.enrollments) + 1, self, course, enrollment_date)
19        self.enrollments.append(enrollment)
20        course.enrollments.append(enrollment)
21        print(f"{self.first_name} {self.last_name} enrolled in {course.course_name}.")
22
23    def update_student_info(self, first_name, last_name, date_of_birth, email, phone_number):
24        self.first_name = first_name
25        self.last_name = last_name
26        self.date_of_birth = date_of_birth
27        self.email = email
28        self.phone_number = phone_number
29
30    def make_payment(self, amount, payment_date):
31        payment = Payment(len(self.payments) + 1, self, amount, payment_date)
32        self.payments.append(payment)
33        print(f"Payment of ${amount} made by {self.first_name} {self.last_name} on {payment_date}.")
34
35    def display_student_info(self):
36        print(f"Student ID: {self.student_id}")
37        print(f"Name: {self.first_name} {self.last_name}")
38        print(f>Date of Birth: {self.date_of_birth}")
39        print(f>Email: {self.email}")
40        print(f>Phone Number: {self.phone_number}")
41
42    def get_enrolled_courses(self):
43        return [enrollment.course for enrollment in self.enrollments]
44
45    def get_payment_history(self):
46        return self.payments
47
48    def __str__(self):
49        return f"Student {self.student_id}: {self.first_name} {self.last_name}, {self.date_of_birth}, {self.email}, {self.phone_number}"
50
51    def __repr__(self):
52        return f"<__main__.Student object at {hex(id(self))}>"
53
54    def __eq__(self, other):
55        return isinstance(other, Student) and self.student_id == other.student_id
56
57    def __lt__(self, other):
58        return self.student_id < other.student_id
59
60    def __gt__(self, other):
61        return self.student_id > other.student_id
62
63    def __le__(self, other):
64        return self.student_id <= other.student_id
65
66    def __ge__(self, other):
67        return self.student_id >= other.student_id
68
69    def __hash__(self):
70        return hash(self.student_id)
```

```
main.py DatabaseConnector.py student.py x course.py enrollment.py teacher.py payment.py

21 self.first_name = first_name
22 self.last_name = last_name
23 self.date_of_birth = date_of_birth
24 self.email = email
25 self.phone_number = phone_number
26 print(f"Student information updated for {self.first_name} {self.last_name}.")
27
28 def make_payment(self, amount, payment_date):
29     payment = Payment(len(self.payments) + 1, self, amount, payment_date)
30     self.payments.append(payment)
31     print(f"Payment of ${amount} made by {self.first_name} {self.last_name} on {payment_date}.")
32
33 def display_student_info(self):
34     print(f"Student ID: {self.student_id}")
35     print(f>Name: {self.first_name} {self.last_name}")
36     print(f>Date of Birth: {self.date_of_birth}")
37     print(f>Email: {self.email}")
38     print(f>Phone Number: {self.phone_number}")
39
40 def get_enrolled_courses(self):
41     return [enrollment.course for enrollment in self.enrollments]
42
43 def get_payment_history(self):
44     return self.payments
45
46 def __str__(self):
47     return f"Student {self.student_id}: {self.first_name} {self.last_name}, {self.date_of_birth}, {self.email}, {self.phone_number}"
48
49 def __repr__(self):
50     return f"<__main__.Student object at {hex(id(self))}>"
51
52 def __eq__(self, other):
53     return isinstance(other, Student) and self.student_id == other.student_id
54
55 def __lt__(self, other):
56     return self.student_id < other.student_id
57
58 def __gt__(self, other):
59     return self.student_id > other.student_id
60
61 def __le__(self, other):
62     return self.student_id <= other.student_id
63
64 def __ge__(self, other):
65     return self.student_id >= other.student_id
66
67 def __hash__(self):
68     return hash(self.student_id)
```

Input/Output given by User

```
C:\Users\ssush\PycharmProjects\SIS5\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\SIS5\main.py

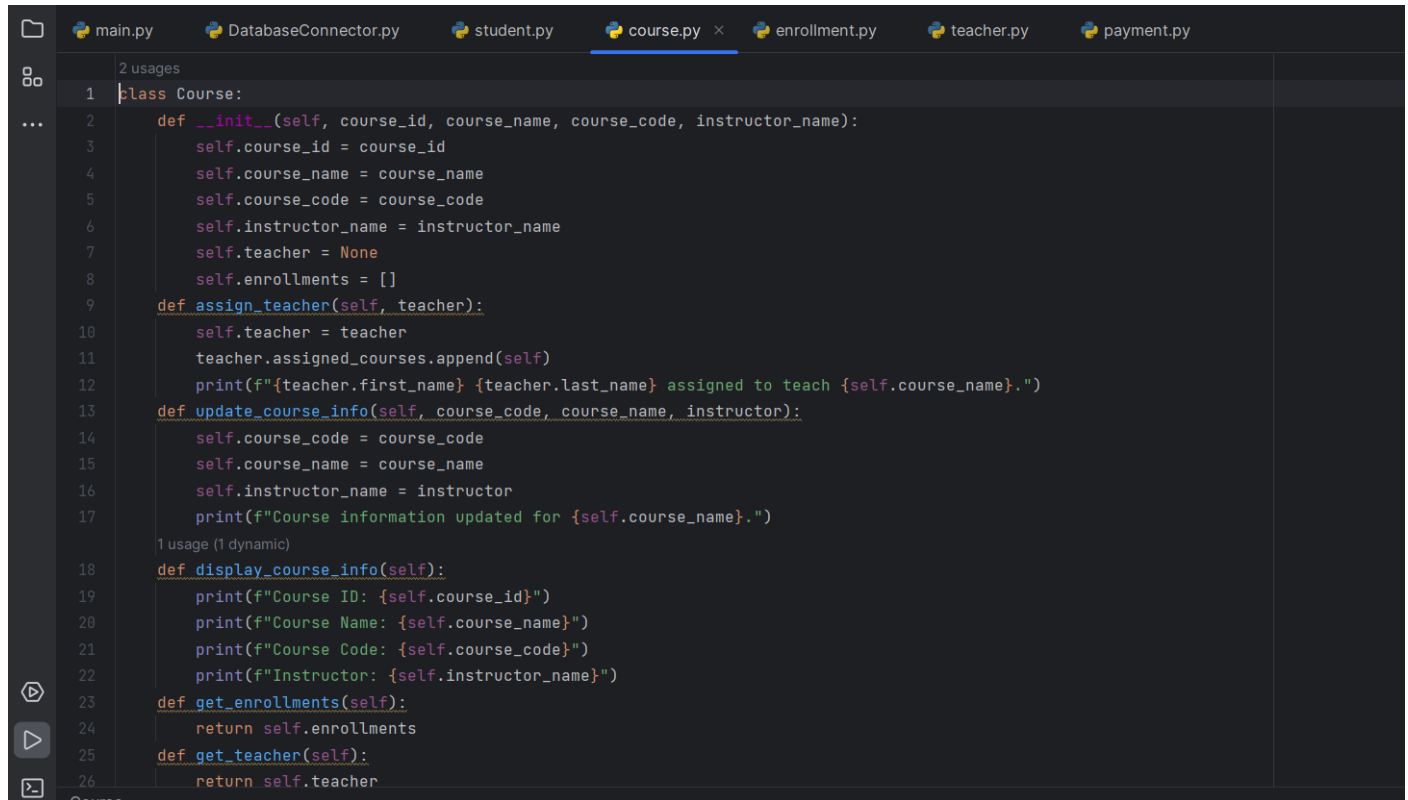
1. Add a New Student
2. Enroll Student in a Course
3. Assign Teacher to a Course
4. Make Payment
5. Display Student Information
6. Display Enrolled Courses
7. Add New Teacher
8. Display Payment History
9. Exit
Enter your choice (1-10): 1
Enter student's first name: Sushant
Enter student's last name: Kumar
Enter of birth date (YYYY-MM-DD): 2001-10-31
Enter student's email: ssushant776@gmail.com
Enter student's phone number: 96542535425
Student added successfully.
Student added successfully.
```

```
mysql> SELECT * FROM students;
+----+-----+-----+-----+-----+-----+
| id | first_name | last_name | birth_date | email | phone |
+----+-----+-----+-----+-----+-----+
| 1 | RAMA | KUMAR | 0500-01-22 | rama@gmail.com | 9652458358 |
| 2 | aman | singh | 2024-09-09 | aman@gmail.com | 9652321459 |
| 3 | jane | jhonson | 2024-01-30 | jane@yahoo.com | 965243585 |
| 4 | Jhon | Doe | 1995-08-15 | jhon.doe@example.com | 123-456-7890 |
| 5 | RAMA | KUMAR | 0500-01-22 | rama@gmail.com | 9652458358 |
| 6 | Sushant | Kumar | 2001-10-31 | ssushant776@gmail.com | 96542535425 |
+----+-----+-----+-----+-----+-----+
6 rows in set (0.01 sec)
```

Data successfully saved into the database

Course class with the following attributes:

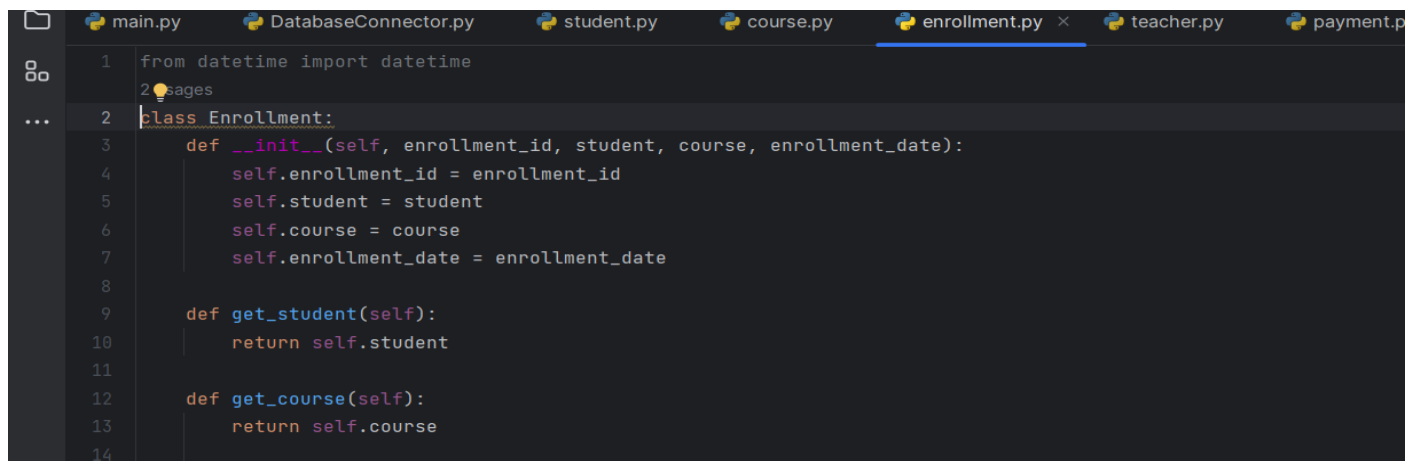
- Course ID
- Course Name
- Course Code
- Instructor Name



```
1 2 usages
2 class Course:
3     def __init__(self, course_id, course_name, course_code, instructor_name):
4         self.course_id = course_id
5         self.course_name = course_name
6         self.course_code = course_code
7         self.instructor_name = instructor_name
8         self.teacher = None
9         self.enrollments = []
10    def assign_teacher(self, teacher):
11        self.teacher = teacher
12        teacher.assigned_courses.append(self)
13        print(f"{teacher.first_name} {teacher.last_name} assigned to teach {self.course_name}.")
14    def update_course_info(self, course_code, course_name, instructor):
15        self.course_code = course_code
16        self.course_name = course_name
17        self.instructor_name = instructor
18        print(f"Course information updated for {self.course_name}.")
19    1 usage (1 dynamic)
20    def display_course_info(self):
21        print(f"Course ID: {self.course_id}")
22        print(f"Course Name: {self.course_name}")
23        print(f"Course Code: {self.course_code}")
24        print(f"Instructor: {self.instructor_name}")
25    def get_enrollments(self):
26        return self.enrollments
27    def get_teacher(self):
28        return self.teacher
```

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 from datetime import datetime
2 2 usages
3 class Enrollment:
4     def __init__(self, enrollment_id, student, course, enrollment_date):
5         self.enrollment_id = enrollment_id
6         self.student = student
7         self.course = course
8         self.enrollment_date = enrollment_date
9
10    def get_student(self):
11        return self.student
12
13    def get_course(self):
14        return self.course
```

Teacher class with the following attributes:

- Teacher ID
- First Name
- Last Name
- Email

```
main.py DatabaseConnector.py student.py course.py enrollment.py teacher.py payment.py
1 from datetime import datetime
2 class Teacher:
3     def __init__(self, teacher_id, first_name, last_name, email):
4         self.teacher_id = teacher_id
5         self.first_name = first_name
6         self.last_name = last_name
7         self.email = email
8         self.assigned_courses = []
9
10    1 usage (1 dynamic)
11    def update_teacher_info(self, name, email, expertise):
12        self.first_name = name
13        self.email = email
14        print(f"Teacher information updated for {self.first_name} {self.last_name}.")
15
16    def display_teacher_info(self):
17        print(f"Teacher ID: {self.teacher_id}")
18        print(f"Name: {self.first_name} {self.last_name}")
19        print(f"Email: {self.email}")
20
21    def get_assigned_courses(self):
22        self.assigned_courses
23        print(f"assigned_courses{self.assigned_courses}.")
```

Payment class with the following attributes:

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

```
objects\SIS5
1 from datetime import datetime
2 4 usages
3 class Payment:
4     def __init__(self, payment_id, student, amount, payment_date):
5         self.payment_id = payment_id
6         self.student = student
7         self.amount = amount
8         self.payment_date = payment_date
9
10    def get_student(self):
11        return self.student
12
13    1 usage (1 dynamic)
14    def get_payment_amount(self):
15        return self.amount
16
17    1 usage (1 dynamic)
18    def get_payment_date(self):
19        return self.payment_date
```

Task 3: Implement Methods

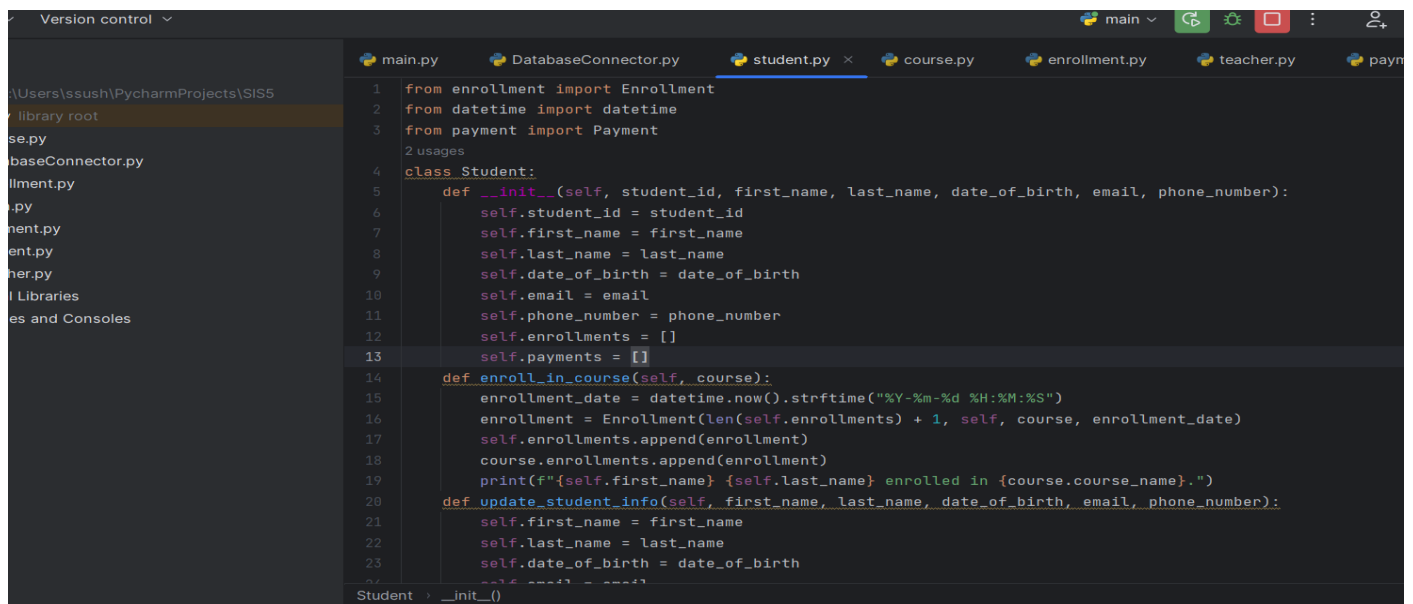
Implement methods in your classes to perform various operations related to the Student Information System (SIS). These methods will allow you to interact with and manipulate data within your system.

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.



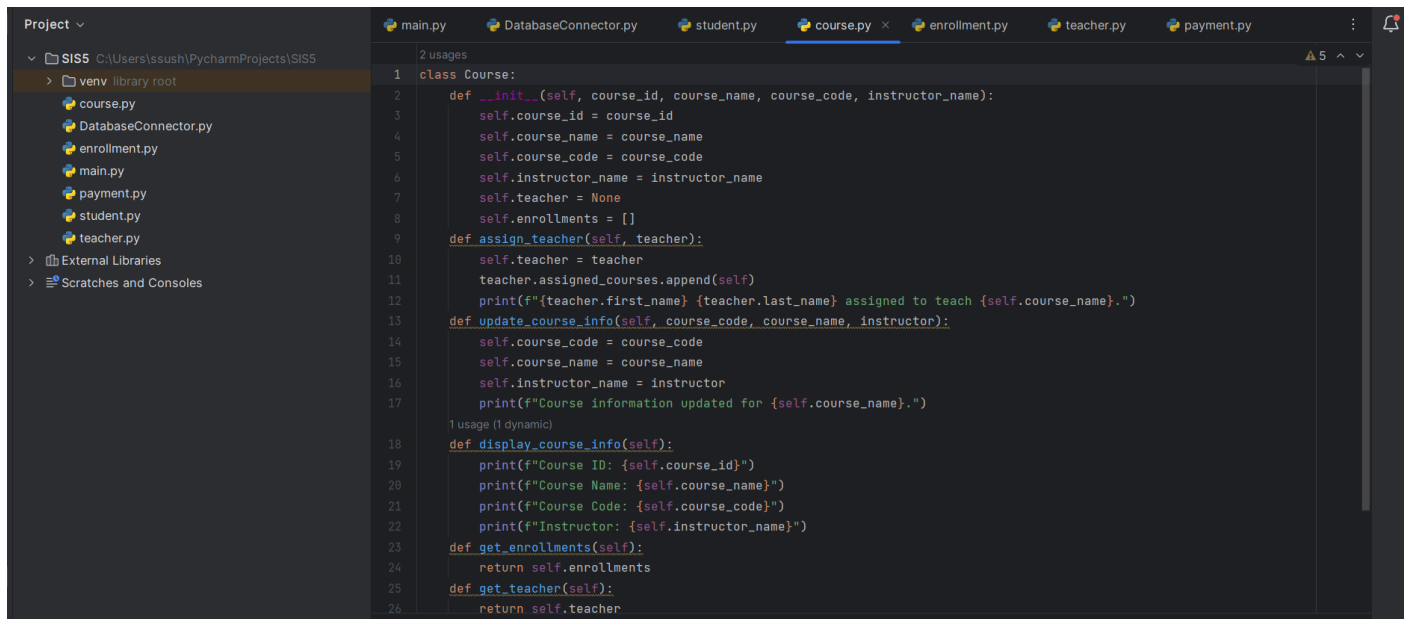
```
Version control
main.py DatabaseConnector.py student.py course.py enrollment.py teacher.py paym
1 from enrollment import Enrollment
2 from datetime import datetime
3 from payment import Payment
4
5 class Student:
6     def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number):
7         self.student_id = student_id
8         self.first_name = first_name
9         self.last_name = last_name
10        self.date_of_birth = date_of_birth
11        self.email = email
12        self.phone_number = phone_number
13        self.enrollments = []
14        self.payments = []
15
16    def enroll_in_course(self, course):
17        enrollment_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
18        enrollment = Enrollment(len(self.enrollments) + 1, self, course, enrollment_date)
19        self.enrollments.append(enrollment)
20        course.enrollments.append(enrollment)
21        print(f"{self.first_name} {self.last_name} enrolled in {course.course_name}.")
22
23    def update_student_info(self, first_name, last_name, date_of_birth, email, phone_number):
24        self.first_name = first_name
25        self.last_name = last_name
26        self.date_of_birth = date_of_birth
27        self.email = email
28        self.phone_number = phone_number
29        print(f"Student information updated for {self.first_name} {self.last_name}.")
30
31    def make_payment(self, amount, payment_date):
32        payment = Payment(len(self.payments) + 1, self, amount, payment_date)
33        self.payments.append(payment)
34        print(f"Payment of ${amount} made by {self.first_name} {self.last_name} on {payment_date}.")
35
36    def display_student_info(self):
37        print(f"Student ID: {self.student_id}")
38        print(f>Name: {self.first_name} {self.last_name}")
39        print(f>Date of Birth: {self.date_of_birth}")
40        print(f>Email: {self.email}")
41        print(f>Phone Number: {self.phone_number}")
42
43    def get_enrolled_courses(self):
44        return [enrollment.course for enrollment in self.enrollments]
45
46    def get_payment_history(self):
47        return self.payments
```



```
21 self.first_name = first_name
22 self.last_name = last_name
23 self.date_of_birth = date_of_birth
24 self.email = email
25 self.phone_number = phone_number
26 print(f"Student information updated for {self.first_name} {self.last_name}.")
27
28 def make_payment(self, amount, payment_date):
29     payment = Payment(len(self.payments) + 1, self, amount, payment_date)
30     self.payments.append(payment)
31     print(f"Payment of ${amount} made by {self.first_name} {self.last_name} on {payment_date}.")
32
33 def display_student_info(self):
34     print(f"Student ID: {self.student_id}")
35     print(f>Name: {self.first_name} {self.last_name}")
36     print(f>Date of Birth: {self.date_of_birth}")
37     print(f>Email: {self.email}")
38     print(f>Phone Number: {self.phone_number}")
39
40 def get_enrolled_courses(self):
41     return [enrollment.course for enrollment in self.enrollments]
42
43 def get_payment_history(self):
44     return self.payments
```

Course Class:

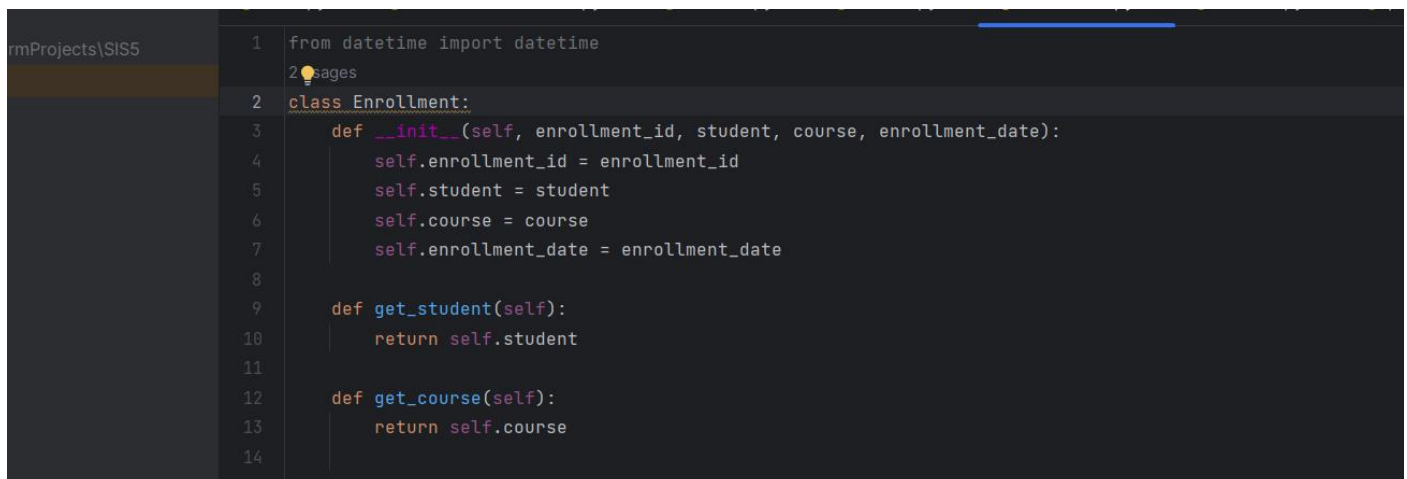
- AssignTeacher(teacher: Teacher): Assigns a teacher to the course.
- UpdateCourseInfo(courseCode: string, courseName: string, instructor: string): Updates course information.
- DisplayCourseInfo(): Displays detailed information about the course.
- GetEnrollments(): Retrieves a list of student enrollments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



```
1 class Course:
2     def __init__(self, course_id, course_name, course_code, instructor_name):
3         self.course_id = course_id
4         self.course_name = course_name
5         self.course_code = course_code
6         self.instructor_name = instructor_name
7         self.teacher = None
8         self.enrollments = []
9     def assign_teacher(self, teacher):
10        self.teacher = teacher
11        teacher.assigned_courses.append(self)
12        print(f"{teacher.first_name} {teacher.last_name} assigned to teach {self.course_name}.")
13    def update_course_info(self, course_code, course_name, instructor):
14        self.course_code = course_code
15        self.course_name = course_name
16        self.instructor_name = instructor
17        print(f"Course information updated for {self.course_name}.")
18    def display_course_info(self):
19        print(f"Course ID: {self.course_id}")
20        print(f"Course Name: {self.course_name}")
21        print(f"Course Code: {self.course_code}")
22        print(f"Instructor: {self.instructor_name}")
23    def get_enrollments(self):
24        return self.enrollments
25    def get_teacher(self):
26        return self.teacher
```

Enrollment Class:

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



```
1 from datetime import datetime
2 class Enrollment:
3     def __init__(self, enrollment_id, student, course, enrollment_date):
4         self.enrollment_id = enrollment_id
5         self.student = student
6         self.course = course
7         self.enrollment_date = enrollment_date
8
9     def get_student(self):
10        return self.student
11
12    def get_course(self):
13        return self.course
```

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.

```
1 from datetime import datetime
2 class Teacher:
3     def __init__(self, teacher_id, first_name, last_name, email):
4         self.teacher_id = teacher_id
5         self.first_name = first_name
6         self.last_name = last_name
7         self.email = email
8         self.assigned_courses = []
9
10    1 usage (1 dynamic)
11    def update_teacher_info(self, name, email, expertise):
12        self.first_name = name
13        self.email = email
14        print(f"Teacher information updated for {self.first_name} {self.last_name}.")
15
16    def display_teacher_info(self):
17        print(f"Teacher ID: {self.teacher_id}")
18        print(f"Name: {self.first_name} {self.last_name}")
19        print(f"Email: {self.email}")
20
21    def get_assigned_courses(self):
22        self.assigned_courses
23        print(f"assigned_courses{self.assigned_courses}.")
24
```

Payment Class:

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

```
objects\SIS5
main.py DatabaseConnector.py student.py course.py enrollment.py teacher.py payment.py
1 from datetime import datetime
2 4 usages
3 class Payment:
4     def __init__(self, payment_id, student, amount, payment_date):
5         self.payment_id = payment_id
6         self.student = student
7         self.amount = amount
8         self.payment_date = payment_date
9
10    def get_student(self):
11        return self.student
12
13    1 usage (1 dynamic)
14    def get_payment_amount(self):
15        return self.amount
16
17    1 usage (1 dynamic)
18    def get_payment_date(self):
19        return self.payment_date
20
```

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.

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.

- **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.

```
exception.py  InvalidTeacherDataException.py  InsufficientFundsException.py  student.py  DuplicateEnrollmentException.py x
2 usages
1 class DuplicateEnrollmentException(Exception):
2     def __init__(self, student_id, course_id):
3         super().__init__(f"Student with ID {student_id} is already enrolled in course with ID {course_id}.")
4
```

```
1 from datetime import datetime
2 from enrollment import Enrollment
3 from payment import Payment
4 from DuplicateEnrollmentException import DuplicateEnrollmentException
5
6
7 2 usages
8 class Student:
9     def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number):
10         self.student_id = student_id
11         self.first_name = first_name
12         self.last_name = last_name
13         self.date_of_birth = date_of_birth
14         self.email = email
15         self.phone_number = phone_number
16         self.enrollments = []
17         self.payments = []
18
19     def enroll_in_course(self, course):
20         if self.is_student_already_enrolled(course):
21             raise DuplicateEnrollmentException(self.student_id, course.course_id)
22
23         enrollment_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
24         enrollment = Enrollment(len(self.enrollments) + 1, self, course, enrollment_date)
25         self.enrollments.append(enrollment)
26         course.enrollments.append(enrollment)
27         print(f"{self.first_name} {self.last_name} enrolled in {course.course_name}.")
28
```

StudentNotFoundException:

```
Exception.py  InvalidTeacherDataException.py  InsufficientFundsException.py  student.py  StudentNotFoundException.py x
1 class StudentNotFoundException(Exception):
2     def __init__(self, student_id):
3         super().__init__(f"Student with ID {student_id} not found.")
4
```

TeacherNotFoundException:

```
Exception.py  InsufficientFundsException.py  student.py  StudentNotFoundException.py  TeacherNotFoundException.py x
1 class TeacherNotFoundException(Exception):
2     def __init__(self, teacher_id):
3         super().__init__(f"Teacher with ID {teacher_id} not found.")
```


PaymentValidationException:

```
Exception.py  student.py  StudentNotFoundException.py  TeacherNotFoundException.py  PaymentValidationException.py x  ⌵  ⋮  
1 class PaymentValidationException(Exception):  
2     def __init__(self, message):  
3         super().__init__(message)
```

InvalidStudentDataException:

```
Exception.py  student.py  StudentNotFoundException.py  TeacherNotFoundException.py  PaymentValidationException.py x  
1 class PaymentValidationException(Exception):  
2     def __init__(self, message):  
3         super().__init__(message)
```

InvalidCourseDataException:

```
StudentNotFoundException.py  TeacherNotFoundException.py  PaymentValidationException.py  InvalidCourseDataException.py x  
1 class InvalidCourseDataException(Exception):  
2     def __init__(self, message):  
3         super().__init__(message)
```

InvalidEnrollmentDataException:

```
TeacherNotFoundException.py  PaymentValidationException.py  InvalidCourseDataException.py  InvalidEnrollmentDataException.py x  ⌵  ⋮  
1 class InvalidEnrollmentDataException(Exception):  
2     def __init__(self, message):  
3         super().__init__(message)
```

InvalidTeacherDataException:

```
PaymentValidationException.py  InvalidCourseDataException.py  InvalidEnrollmentDataException.py  InvalidTeacherDataException.py x  ⌵  ⋮  
1 class InvalidTeacherDataException(Exception):  
2     def __init__(self, message):  
3         super().__init__(message)  
4
```

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.

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

```
1 usage
2 class Student:
3     def __init__(self, student_id, first_name, last_name, birth_date, email, phone):
4         self._student_id = student_id
5         self._first_name = first_name
6         self._last_name = last_name
7         self._birth_date = birth_date
8         self._email = email
9         self._phone = phone
10        self._enrollments = [] # List to store Enrollment objects
11
12 1 usage (1 dynamic)
13 @property
14 def student_id(self):
15     return self._student_id
16
17 2 usages (2 dynamic)
18 @property
19 def first_name(self):
20     return self._first_name
21
22 2 usages (2 dynamic)
23 @property
24 def last_name(self):
```

```
25     return self._last_name
26
27 @property
28 def birth_date(self):
29     return self._birth_date
30
31 @property
32 def email(self):
33     return self._email
34
35 @property
36 def phone(self):
37     return self._phone
38
39 3 usages (1 dynamic)
40 @property
41 def enrollments(self):
42     return self._enrollments
43
44 2 usages (1 dynamic)
45 @enrollments.setter
46 def enrollments(self, enrollment):
47     self._enrollments.append(enrollment)
```

Course Class:

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

```
1 usage
1 class Course:
2     def __init__(self, course_code, course_name):
3         self._course_code = course_code
4         self._course_name = course_name
5         self._enrollments = []
6
7     4 usages (1 dynamic)
8     @property
9     def enrollments(self):
10         return self._enrollments
11
12     2 usages (1 dynamic)
13     @enrollments.setter
14     def enrollments(self, enrollment):
15         self._enrollments.append(enrollment)
16
17     1 usage (1 dynamic)
18     @enrollments.deleter
19     def enrollments(self):
20         self._enrollments.clear()
```

Enrollment Class:

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

```
course.py  enrollment.py  ENROLLMENTT.PY  StudentNotFoundException.py  TeacherNotFoundException.py
1 usage
1 class Enrollment:
2     def __init__(self, student, course):
3         self._student = student
4         self._course = course
5
6     2 usages
7     @property
8     def student(self):
9         return self._student
10
11     @student.setter
12     def student(self, student):
13         self._student = student
14
15     3 usages (1 dynamic)
16     @property
17     def course(self):
18         return self._course
19
20     1 usage (1 dynamic)
21     @course.setter
22     def course(self, course):
23         self._course = course
```

Payment Class: Include a property to hold a reference to the Student object. Example: Student Student { get; set; }

```
1 usage
1 class Payment:
2     def __init__(self, payment_id, amount, payment_date, student):
3         self._payment_id = payment_id
4         self._amount = amount
5         self._payment_date = payment_date
6         self._student = student
7
8     @property
9     def student(self):
10         return self._student
11
12     @student.setter
13     def student(self, new_student):
14         self._student = new_student
15
16     @student.deleter
17     def student(self):
18         del self._student
```

Teacher Class:

Create a list or collection property to store the teacher's assigned courses. This property will hold references to Course objects.

```
1 usage
1 class Teacher:
2     def __init__(self, teacher_id, name, email, assigned_courses=None):
3         self._teacher_id = teacher_id
4         self._name = name
5         self._email = email
6         self._assigned_courses = assigned_courses or []
7
8     4 usages (1 dynamic)
9     @property
10    def assigned_courses(self):
11        return self._assigned_courses
12
13    1 usage (1 dynamic)
14    @assigned_courses.setter
15    def assigned_courses(self, new_courses):
16        self._assigned_courses = new_courses
17
18    1 usage (1 dynamic)
19    @assigned_courses.deleter
20    def assigned_courses(self):
21        del self._assigned_courses
```

Task 6: Create Methods for Managing Relationships

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.

```
main.py  SIS.py  Student.py  Course.py  Enrollment.py  Teacher.py  Payment.py
1 from datetime import datetime
2 from Enrollment import Enrollment
3 from Student import Student
4 from Teacher import Teacher
5 from Payment import Payment
6
7 5 usages
8 class SIS:
9     def __init__(self):
10         self.students = []
11         self.courses = []
12         self.teachers = []
13
14     3 usages
15     def enroll_student_in_course(self, student, course):
16         student.enroll_in_course(course)
17         enrollment = Enrollment(student.student_id, course.course_id, datetime.now())
18         course.enrollments.append(enrollment)
19
20     3 usages
21     def assign_teacher_to_course(self, teacher, course):
22         course.assign_teacher(teacher)
23         teacher.assigned_courses.append(course)
24
25     3 usages
26     def record_payment(self, student, amount, payment_date):
27         payment = Payment(student.student_id, amount, payment_date)
28         student.payments.append(payment)
29
30     2 usages
31     def generate_enrollment_report(self, course):
32         print(f"Enrollment Report for Course {course.course_name}")
33         for enrollment in course.enrollments:
34             print(f"Student ID: {enrollment.student_id}, Enrollment Date: {enrollment.enrollment_date}")
35
36     2 usages
37     def generate_payment_report(self, student):
38         print(f"Payment Report for Student {student.first_name} {student.last_name}")
39         for payment in student.payments:
40             print(f"Payment ID: {payment.payment_id}, Amount: {payment.amount}, Payment Date: {payment.payment_date}")
```

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.

```
main.py DatabaseConnector.py x student.py course.py enrollment.py teacher.py payn
1
2 import mysql.connector
2 usages
3 class DatabaseConnector:
4     def __init__(self):
5         self.connection = None
6     def open_connection(self):
7         self.connection = mysql.connector.connect(
8             host="localhost",
9             user="root",
10            password="Sushant@9546",
11            database="SIS5"
12        )
13        self.create_database()
14        self.create_tables()
1 usage (1 dynamic)
15 def close_connection(self):
16     if self.connection:
17         self.connection.close()
18
```

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.

```
main.py DatabaseConnector.py x student.py course.py enrollment.py teacher.py payment.py
33 self.connection.close()
34
1 usage
35 def create_tables(self):
36     def create_tables(self):
37         student_table_query = """
38         CREATE TABLE IF NOT EXISTS students (
39             id INT AUTO_INCREMENT PRIMARY KEY,
40             first_name VARCHAR(255) NOT NULL,
41             last_name VARCHAR(255) NOT NULL,
42             birth_date DATE NOT NULL,
43             email VARCHAR(255) NOT NULL,
44             phone VARCHAR(20) NOT NULL
45         );
46         """
47         course_table_query = """
48         CREATE TABLE IF NOT EXISTS courses (
49             id INT AUTO_INCREMENT PRIMARY KEY,
50             course_name VARCHAR(255) NOT NULL,
51             course_code VARCHAR(20) NOT NULL,
52             instructor VARCHAR(255) NOT NULL
53         );
54
```

```
main.py DatabaseConnector.py x student.py course.py enrollment.py teacher.py payment.py
54
55     teacher_table_query = """
56     CREATE TABLE IF NOT EXISTS teachers (
57         id INT AUTO_INCREMENT PRIMARY KEY,
58         first_name VARCHAR(255) NOT NULL,
59         last_name VARCHAR(255) NOT NULL,
60         email VARCHAR(255) NOT NULL,
61         assigned_courses Varchar(255) Not null
62     );
63     """
64     payment_table_query = """
65     CREATE TABLE IF NOT EXISTS payments (
66         id INT AUTO_INCREMENT PRIMARY KEY,
67         student_id INT NOT NULL,
68         amount FLOAT NOT NULL,
69         payment_date DATE NOT NULL,
70         FOREIGN KEY (student_id) REFERENCES students(id)
71     );
72     """
73     enrollment_table_query = """
74     CREATE TABLE IF NOT EXISTS enrollments (
75         id INT AUTO_INCREMENT PRIMARY KEY,
76         student_id INT NOT NULL,
77         course_id INT NOT NULL,
78         enrollment_date DATE NOT NULL,
79         FOREIGN KEY (student_id) REFERENCES students(id),
80         FOREIGN KEY (course_id) REFERENCES courses(id)
81     );
```

```
main.py DatabaseConnector.py x student.py course.py enrollment.py teacher.py payment.py
84     def insert_enrollment(self):
85         student_id = input("Enter student ID for enrollment: ")
86
87         check_student_query = "SELECT id FROM students WHERE id = %s"
88         self.cursor.execute(check_student_query, (student_id,))
89         result = self.cursor.fetchone()
90
91         if result is None:
92             print(f"Error: Student with ID {student_id} does not exist.")
93             return
94
95         course_id = input("Enter course ID for enrollment: ")
96
97         check_course_query = "SELECT id FROM courses WHERE id = %s"
98         self.cursor.execute(check_course_query, (course_id,))
99         result = self.cursor.fetchone()
100
101         if result is None:
102             print(f"Error: Course with ID {course_id} does not exist.")
103             return
104
105         enrollment_date = get_date_input()
106
107         query = """
108         INSERT INTO enrollments (student_id, course_id, enrollment_date)
```

```
main.py DatabaseConnector.py x student.py course.py enrollment.py teacher.py payment.py
105     enrollment_date = get_date_input()
106
107     query = """
108     INSERT INTO enrollments (student_id, course_id, enrollment_date)
109     VALUES (%s, %s, %s);
110     """
111     values = (student_id, course_id, enrollment_date)
112     self.execute_query(query, values)
113     print("Enrollment successful.")
114
115     1 usage (1 dynamic)
116     def assign_teacher_to_course(self):
117         course_id = input("Enter course ID: ")
118         teacher_id = input("Enter teacher ID: ")
119
120         check_course_query = "SELECT id FROM courses WHERE id = %s"
121         self.cursor.execute(check_course_query, (course_id,))
122         result_course = self.cursor.fetchone()
123
124         check_teacher_query = "SELECT id FROM teachers WHERE id = %s"
125         self.cursor.execute(check_teacher_query, (teacher_id,))
126         result_teacher = self.cursor.fetchone()
127
128         if result_course is None:
129             print(f"Error: Course with ID {course_id} does not exist.")
130             return
131         if result_teacher is None:
132             print(f"Error: Teacher with ID {teacher_id} does not exist.")
```

```
main.py DatabaseConnector.py x student.py course.py enrollment.py teacher.py payment.py
132         return
133     update_course_query = "UPDATE courses SET instructor = %s WHERE id = %s"
134     values = (teacher_id, course_id)
135     self.execute_query(update_course_query, values)
136     print("Teacher assigned to the course.")
137
138     1 usage (1 dynamic)
139     def insert_payment(self):
140         student_id = input("Enter student ID for payment: ")
141         amount = float(input("Enter payment amount: "))
142         payment_date = get_date_input()
143         check_student_query = "SELECT id FROM students WHERE id = %s"
144         self.cursor.execute(check_student_query, (student_id,))
145         result = self.cursor.fetchone()
146
147         if result is None:
148             print(f"Error: Student with ID {student_id} does not exist.")
149             return
150         query = """
151         INSERT INTO payments (student_id, amount, payment_date)
152         VALUES (%s, %s, %s);
153         """
154         values = (student_id, amount, payment_date)
155         self.execute_query(query, values)
156         print("Payment recorded successfully.")
157
158     1 usage (1 dynamic)
159     def get_student_info(self, student_id):
```

```
main.py DatabaseConnector.py x student.py course.py enrollment.py teacher.py payment.py
157     def get_student_info(self, student_id):
158
159         check_student_query = "SELECT * FROM students WHERE id = %s"
160         self.cursor.execute(check_student_query, (student_id,))
161         result = self.cursor.fetchone()
162
163         if result is None:
164             print(f"Error: Student with ID {student_id} does not exist.")
165             return None
166
167         student_info = Student(result[0], result[1], result[2], result[3], result[4], result[5])
168         return student_info
169
170     1 usage (1 dynamic)
171     def get_payment_history(self, student_id):
172
173         query = """
174         SELECT payments.id, payments.amount, payments.payment_date
175         FROM payments
176         WHERE payments.student_id = %s;
177         """
178         self.cursor.execute(query, (student_id,))
179         payment_history_data = self.cursor.fetchall()
180
181         payment_history = []
182         for payment_data in payment_history_data:
183             payment = Payment(*payment_data)
184             payment_history.append(payment)
```

```
main.py DatabaseConnector.py x student.py course.py enrollment.py teacher.py payment.py
183         return payment_history
184
185     1 usage (1 dynamic)
186     def get_enrolled_courses(self, student_id):
187
188         query = """
189         SELECT courses.course_id, courses.course_name, courses.course_code, courses.instructor_name
190         FROM enrollments
191         JOIN courses ON enrollments.course_id = courses.course_id
192         WHERE enrollments.student_id = %s;
193         """
194
195         self.cursor.execute(query, (student_id,))
196         enrolled_courses_data = self.cursor.fetchall()
197
198         enrolled_courses = []
199         for course_data in enrolled_courses_data:
200             course = Course(*course_data)
201             enrolled_courses.append(course)
202
203         return enrolled_courses
204
205     1 usage (1 dynamic)
206     def insert_student(self, first_name, last_name, birth_date, email, phone):
207
208         query = """
209         INSERT INTO students (first_name, last_name, birth_date, email, phone)
210         VALUES (%s, %s, %s, %s, %s);
211         """
```

202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222

```
    return enrolled_courses

1 usage (1 dynamic)

def insert_student(self, first_name, last_name, birth_date, email, phone):
    query = """
    INSERT INTO students (first_name, last_name, birth_date, email, phone)
    VALUES (%s, %s, %s, %s, %s);
    """
    values = (first_name, last_name, birth_date, email, phone)
    self.execute_query(query, values)
    print("Student added successfully.")

1 usage (1 dynamic)

def insert_teacher(self, first_name, last_name, email, assigned_courses):
    try:
        query = "INSERT INTO teachers (first_name, last_name, email, assigned_courses) VALUES (%s, %s, %s, %s)"
        values = (first_name, last_name, email, assigned_courses)
        self.cursor.execute(query, values)
        self.connection.commit()
    except Exception as e:
        print(f"Error inserting teacher: {e}")
```


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

```
C:\Users\ssush\PycharmProjects\SIS5\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\SIS5\main.py

1. Add a New Student
2. Enroll Student in a Course
3. Assign Teacher to a Course
4. Make Payment
5. Display Student Information
6. Display Enrolled Courses
7. Add New Teacher
8. Display Payment History
9. Exit
Enter your choice (1-10): 1
Enter student's first name: Jhon
Enter student's last name: Doe
Enter of birth date (YYYY-MM-DD): 1995-08-15
Enter student's email: jhon.doe@example.com
Enter student's phone number: 123-456-7890
Student added successfully.
Student added successfully.
```

Expected Data saved in Database Successfully

```
mysql> select * from students;
+----+-----+-----+-----+-----+-----+
id | first_name | last_name | birth_date | email | phone |
+----+-----+-----+-----+-----+-----+
1 | RAMA | KUMAR | 0500-01-22 | rama@gmail.com | 9652458358 |
2 | aman | singh | 2024-09-09 | aman@gmail.com | 9652321459 |
3 | Ramu | raj | 2024-01-30 | ramuraj@yahoo.com | 965243585 |
4 | Jhon | Doe | 1995-08-15 | jhon.doe@example.com | 123-456-7890 |
+----+-----+-----+-----+-----+-----+
rows in set (0.01 sec)
```

John is enrolling in the following courses:

- 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.

```
C:\Users\ssush\PycharmProjects\SIS5\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\SIS5\main.py

1. Add a New Student
2. Enroll Student in a Course
3. Assign Teacher to a Course
4. Make Payment
5. Display Student Information
6. Display Enrolled Courses
7. Add New Teacher
8. Display Payment History
9. Exit
Enter your choice (1-10): 2
Enter student ID for enrollment: 4
Enter course ID for enrollment: 1
Enter date (YYYY-MM-DD): 2024-02-02
Enrollment successful.
```

Expected Data saved in Database Successfully

```
mysql> select * from ENROLLMENTS;
+-----+-----+-----+-----+
| id | student_id | course_id | enrollment_date |
+-----+-----+-----+-----+
| 2 | 1 | 1 | 2024-01-30 |
| 3 | 1 | 1 | 2024-01-30 |
| 4 | 4 | 1 | 2024-02-02 |
+-----+-----+-----+-----+
3 rows in set (0.01 sec)

mysql> select * from COURSES;
+-----+-----+-----+-----+
| id | course_name | course_code | instructor |
+-----+-----+-----+-----+
| 1 | Mathematics | MATH101 | Dr. RAMANUJAN |
+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

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 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.

```
1. Add a New Student
2. Enroll Student in a Course
3. Assign Teacher to a Course
4. Make Payment
5. Display Student Information
6. Display Enrolled Courses
7. Add New Teacher
8. Display Payment History
9. Exit
Enter your choice (1-10): 7
Enter teacher's first name: SARHA
Enter teacher's last name: SMITH
Enter teacher's email: sarh.smith@example.com
Enter teacher's assigned_courses: Computer Science
New teacher added successfully.
```

Expected Data saved in Database Successfully

```
mysql> select * from teachers;
+----+-----+-----+-----+-----+
| id | first_name | last_name | email | assigned_courses |
+----+-----+-----+-----+-----+
| 1 | DR | RAMANUJAN | dr.raama@yahoo.com | Mathematics |
| 2 | SARAH | SMITHS | sarah.smiths@example.com | Computer Science |
| 3 | anu | singh | anu@gmail.com | english |
+----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Task 10: Payment Record

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

Jane Johnson's details:

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

The system should perform the following tasks:

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

```
Run main x
C:\Users\ssush\PycharmProjects\SIS5\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\SIS5\main.py
1. Add a New Student
2. Enroll Student in a Course
3. Assign Teacher to a Course
4. Make Payment
5. Display Student Information
6. Display Enrolled Courses
7. Add New Teacher
8. Display Payment History
9. Exit
Enter your choice (1-10): 4
Enter student ID for payment: 3
Enter payment amount: 500.00
Enter date (YYYY-MM-DD): 2023-04-10
Payment recorded successfully.
```

Expected Data saved in Database Successfully

```
mysql> select * from payments;
+----+-----+-----+-----+
| id | student_id | amount | payment_date |
+----+-----+-----+-----+
| 1 | 1 | 5000 | 2024-02-03 |
| 2 | 3 | 95240 | 2024-03-10 |
| 3 | 1 | 5000 | 2024-02-03 |
| 4 | 3 | 500 | 2023-04-10 |
+----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```