Python Assignment-Student information system

Name:Soundarya V

Task 1 & Task 2:

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

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

Output:

Student:

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

Course:

```
class Course:
    def __init__(self, course_id, course_name, course_code, instructor_name):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
        self.enrollments=[]
```

Enrollment:

```
class Enrollment:
    def __init__(self, enrollment_id, student, course, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student = student
        self.course = course
        self.enrollment_date = enrollment_date
```

Teacher:

```
def display_teacher_info(self):
    print("Teacher ID:", self.teacher_id)
    print("Name:", self.first_name, self.last_name)
    print("Email:", self.email)
```

Payment:

```
class Payment:
    def __init__(self, payment_id, student, amount, payment_date):
        self.payment_id = payment_id
        self.student = student
        self.amount = amount
        self.payment_date = payment_date
```

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.

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

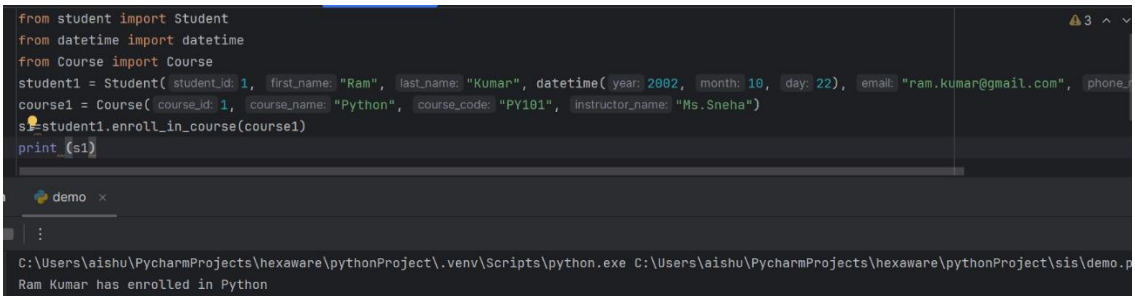
Task 3: Implement Methods Implement methods in your classes to perform various operations related to the Student Information System (SIS). These methods will allow you to interact with and manipulate data within your system. Below are detailed instructions on how to implement methods in each class:

Student Class:

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

```
def enroll_in_course(self, course):
    enrollment_id=len(self.enrollments)+1
    enrollment = Enrollment(enrollment_id,self.student_id,
course.course_id, datetime.now())
    self.enrollments.append(enrollment)
    print(f"{self.first_name} {self.last_name} has enrolled in
{course.course_name}")
```

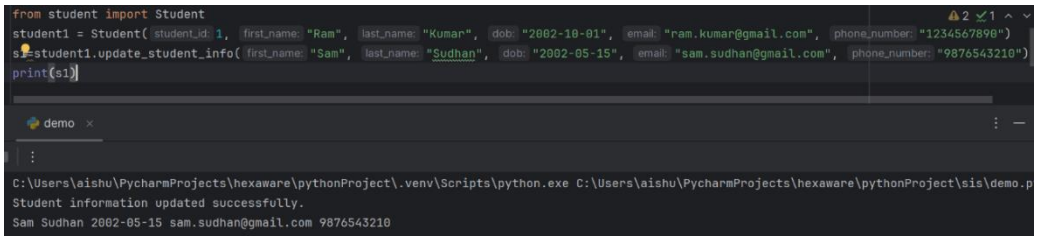
Implementation:



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

```
def update_student_info(self, first_name, last_name, dob, email,
phone_number):
    if not self.is_valid_email(email):
        raise InvalidStudentDataException("Invalid email format.")
    self.first_name = first_name
    self.last_name = last_name
    self.dob = dob
    self.email = email
    self.phone_number = phone_number
    print("Student information updated successfully.")
    print(f"{self.first_name} {self.last_name} {self.dob} {self.email}
{self.phone_number}")
```

Implementation:



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

```
def make_payment(self, amount, payment_date):
    payment_id = len(self.payments) + 1
    payment = Payment(payment_id, self.student_id, amount,
payment_date)
    self.payments.append(payment)
    print(f"Payment of Rs.{amount} recorded successfully on
{payment_date} by {self.first_name}.")
```

Implementation:

```
from student import Student
from datetime import datetime
student1 = Student( student_id: 1, first_name: "Ram", last_name: "Kumar", dob: "2002-10-01", email: "ram.kumar@gmail.com", phone_number: "1234567890")
payment_amount = 2000.0
payment_date = datetime.now()
student1.make_payment(payment_amount, payment_date)
```

```
demo
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\pythonProject\sis\dem
Payment of Rs.2000.0 recorded successfully on 2024-04-26 19:40:47.888910 by Ram.
```

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

```
def display_student_info(self):
    print("Student ID:", self.student_id)
    print("First Name:", self.first_name)
    print("Last Name:", self.last_name)
    print("Date of Birth:", self.dob)
    print("Email:", self.email)
    print("Phone Number:", self.phone_number)
```

Implementation:

```
from student import Student
from datetime import datetime
student1 = Student( student_id: 1, first_name: "Ram", last_name: "Kumar", dob: "2002-10-01", email: "ram.kumar@gmail.com", phone_number: "1234567890")
student1.display_student_info()
```

```
demo
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\pythonProject\sis\demo.
Student ID: 1
First Name: Ram
Last Name: Kumar
Date of Birth: 2002-10-01
Email: ram.kumar@gmail.com
Phone Number: 1234567890
```

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

```
def get_enrolled_courses(self, student_id):
    enrolled_courses = []
    for enrollment in self.enrollments:
        if enrollment.student_id == student_id:
            enrolled_courses.append(enrollment.course_id)
    return enrolled_courses
```

Implementation:

```
from student import Student
from Enrollment import Enrollment
from datetime import datetime
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-01",  email: "ram.kumar@gmail.com",  phone_number: "1234567890")
student2 = Student( student_id: 2,  first_name: "Prem",  last_name: "Kumar",  dob: "2002-11-01",  email: "prem.kumar@gmail.com",  phone_number: "1238767890
student1.enrollments.append(Enrollment( enrollment_id: 1, student_id=1, course_id=1, enrollment_date=datetime.now()))
student2.enrollments.append(Enrollment( enrollment_id: 2, student_id=2, course_id=2, enrollment_date=datetime.now()))
student1.enrollments.append(Enrollment( enrollment_id: 3, student_id=1, course_id=3, enrollment_date=datetime.now()))
student2.enrollments.append(Enrollment( enrollment_id: 4, student_id=2, course_id=4, enrollment_date=datetime.now()))
student2.enrollments.append(Enrollment( enrollment_id: 5, student_id=2, course_id=5, enrollment_date=datetime.now()))
enrolled_courses = student1.get_enrolled_courses(1)
print("Enrolled Courses for Student:")
for course_id in enrolled_courses:
    print(f" - Course ID: {course_id}")
```

```
demo
Enrolled Courses for Student:
 - Course ID: 1
 - Course ID: 3
```

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

```
def get_payment_history(self,student_id):
    payment_history = []
    for payment in self.payments:
        if payment.student_id == student_id:
            payment_history.append(payment)
    return payment_history
```

Implementation:

```
from student import Student
from Payment import Payment
from datetime import datetime
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-01",  email: "ram.kumar@gmail.com",  phone_number: "1234567890")
student2 = Student( student_id: 2,  first_name: "Prem",  last_name: "Kumar",  dob: "2002-11-01",  email: "prem.kumar@gmail.com",  phone_number: "1238767890
student1.payments.append(Payment( payment_id: 1, student_id=1, amount=50.0, payment_date=datetime.now()))
student2.payments.append(Payment( payment_id: 2, student_id=2, amount=100.0, payment_date=datetime.now()))
student1.payments.append(Payment( payment_id: 3, student_id=1, amount=75.0, payment_date=datetime.now()))

payment_history = student1.get_payment_history(1)
print("Payment History for Student:")
for payment in payment_history:
    print(f" - Payment ID: {payment.payment_id}, Amount: {payment.amount}, Payment Date: {payment.payment_date}")
```

```
demo
Payment History for Student:
- Payment ID: 1, Amount: 50.0, Payment Date: 2024-04-26 19:55:31.853631
- Payment ID: 3, Amount: 75.0, Payment Date: 2024-04-26 19:55:31.853631
```

Course Class:

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

```
def assign_teacher(self, teacher):
    self.instructor_name = f"{teacher.first_name} {teacher.last_name}"
    print(f"{self.course_name} has been assigned to {self.instructor_name}.")
```

Implementation:

```
from Teacher import Teacher
from Course import Course
teacher1 = Teacher( teacher_id: 1,  first_name: "Swetha",  last_name: "Sri",  email: "Swetha.sri@gmail.com")
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Swetha")
course1.assign_teacher(teacher1)
```

```
demo ×
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmPr
Python has been assigned to Swetha Sri.
```

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

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

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

def display_course_info(self):
    print("Course ID:", self.course_id)
    print("Course Name:", self.course_name)
    print("Course Code:", self.course_code)
    print("Instructor_name:",self.instructor_name)
```

Implementation:

```
from Course import Course
course1=Course( course_id: 1, course_name: "Python", course_code: "PY101",  instructor_name: "Swetha")
course1.display_course_info()
print("-----")
course1.update_course_info( course_code: "PY101",  course_name: "Python",  instructor: "Priya")
course1.display_course_info()
```

```
demo ×
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\Pycha
Course ID: 1
Course Name: Python
Course Code: PY101
Instructor_name: Swetha
-----
Course information updated successfully.
Course ID: 1
Course Name: Python
Course Code: PY101
Instructor_name: Priya
```

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

```
def get_enrollments(self):
    course_enrollments = []
    for enrollment in self.enrollments:
        if enrollment.course_id == self.course_id:
            course_enrollments.append(enrollment)
    return course_enrollments
```

Implementation:

```
from Enrollment import Enrollment
from Course import Course
from datetime import datetime
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101", instructor_name: "Swetha")
course1.enrollments.append(Enrollment( enrollment_id: 1, student_id=1, course_id=1, enrollment_date=datetime.now()))
course1.enrollments.append(Enrollment( enrollment_id: 2, student_id=2, course_id=2, enrollment_date=datetime.now()))
course1.enrollments.append(Enrollment( enrollment_id: 3, student_id=3, course_id=1, enrollment_date=datetime.now()))

enrollments_for_course1 = course1.get_enrollments()
print("Enrollments for Course:")
for enrollment in enrollments_for_course1:
    print(f" - Student ID: {enrollment.student_id}")
```

```
demo ×
⋮
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware
Enrollments for Course:
 - Student ID: 1
 - Student ID: 3
```

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

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

Implementation:

```
from Enrollment import Enrollment
from Course import Course
from datetime import datetime
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101", instructor_name: "Swetha")
teacher = course1.get_teacher()
print(f"The teacher for {course1.course_name} is {teacher}.")
```

```
demo ×
⋮
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\Pycha
The teacher for Python is Swetha.
```

Enrollment Class:

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

• GetCourse(): Retrieves the course associated with the enrollment.

```python
class Enrollment:

    1 usage
    def get_student_id(self, enrollment_id):
        if self.enrollment_id == enrollment_id:
            return self.student.student_id
        else:
            return None
    1 usage
    def get_course_id(self, enrollment_id):
        if self.enrollment_id == enrollment_id:
            return self.course.course_id
        else:
            return None
```

Implementation:

```python
from datetime import datetime
from Enrollment import Enrollment
from Course import Course
from student import Student

course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Swetha")
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2000-01-01",  email: "ram.kumar@gmail.com",  phone_number: "1234567890")
enrollment1 = Enrollment( enrollment_id: 1, student1, course1, datetime.now())

student_id = enrollment1.get_student_id(1)
print(f"The student ID associated with this enrollment is: {student_id}")

course_id = enrollment1.get_course_id(1)
print(f"The course ID associated with this enrollment is: {course_id}")
```

```
demo
:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\pythonProject\sis\demo.p
he student ID associated with this enrollment is: 1
he course ID associated with this enrollment is: 1
```

Teacher Class:

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

• DisplayTeacherInfo(): Displays detailed information about the teacher

```python
class Teacher:
    def update_teacher_info(self, first_name,last_name, email):
        self.last_name = last_name
        self.email = email
        print("Teacher information updated successfully.")

    2 usages
    def display_teacher_info(self):
        print("Teacher ID:", self.teacher_id)
        print("Name:", self.first_name, self.last_name)
        print("Email:", self.email)
```

Implementation:

```
from Teacher import Teacher
from Course import Course
teacher1 = Teacher( teacher_id: 1,  first_name: "Swetha",  last_name: "Sri",  email: "Swetha.sri@gmail.com")
teacher1.display_teacher_info()
print("-----")
teacher1.update_teacher_info( first_name: "Priya", last_name: "Dharshini", email: "priya@gmail.com")
teacher1.display_teacher_info()
```

```
demo
Name: Swetha Sri
Email: Swetha.sri@gmail.com
-----
Teacher information updated successfully.
Teacher ID: 1
Name: Priya Dharshini
Email: priya@gmail.com
```

Payment Class:

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

```
def get_student(self, student_id):
    if self.student_id == student_id:
        return self
```

Implementation:

```
from Payment import Payment
from datetime import datetime
payment1 = Payment(payment_id=1, student_id=1, amount=100, payment_date=datetime.now())
payment2 = Payment(payment_id=2, student_id=2, amount=150, payment_date=datetime.now())
student_id_to_find = 1
found_payment = payment1.get_student(student_id_to_find)
if found_payment:
    print("Payment found:")
    print("Payment ID:", found_payment.payment_id)
    print("Amount:", found_payment.amount)
    print("Payment Date:", found_payment.payment_date)
```

```
demo
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\a
Payment found:
Payment ID: 1
Amount: 100
Payment Date: 2024-04-26 21:25:33.600702
```

• GetPaymentAmount(): Retrieves the payment amount.

```
def get_payment_amount(self):
    return self.amount
```

Implementation:

```
from Payment import Payment
from datetime import datetime
payment1 = Payment(payment_id=1, student_id=1, amount=100, payment_date=datetime.now())
payment2 = Payment(payment_id=2, student_id=2, amount=150, payment_date=datetime.now())
payment_amount = payment1.get_payment_amount()
print("Payment Amount:", payment_amount)
```

```
demo ×
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\Pyc
Payment Amount: 100
```

• GetPaymentDate(): Retrieves the payment date.

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

Implementation:

```
from Payment import Payment
from datetime import datetime
payment1 = Payment(payment_id=1, student_id=1, amount=100, payment_date=datetime.now())
payment2 = Payment(payment_id=2, student_id=2, amount=150, payment_date=datetime.now())
payment_date = payment1.get_payment_date()
print("Payment Date:", payment_date)
```

```
demo ×
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\a
Payment Date: 2024-04-26 21:28:46.349138
```

SIS Class :

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

```
class SIS:

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

Implementation:

```
from student import Student
from Course import Course
from datetime import datetime
from sis import SIS
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-22",  email: "ram.kumar@example.com",  phone_number: "1234567890"
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Ms.Sneha")
sis = SIS()
sis.enroll_student_in_course(student1,course1)
```

```
demo ×
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\pythonProject\sis\demo.p
Student Ram Kumar enrolled in course Python.
```

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

```
class SIS:

    1 usage (1 dynamic)
    def assign_teacher_to_course(self, teacher, course):
        course.instructor_name = f"{teacher.first_name} {teacher.last_name}"
        print(f"Teacher {teacher.first_name} {teacher.last_name} assigned to course {course.course_name}.")
```

Implementation:

```
from Teacher import Teacher
from Course import Course
from datetime import datetime
from sis import SIS
#student1 = Student(1, "Ram", "Kumar", "2002-10-22", "ram.kumar@example.com", "1234567890")
teacher1 = Teacher(teacher_id=1, first_name="Ms.", last_name="Sneha", email="sneha@gmail.com")
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Ms.Sneha")
sis = SIS()
sis.assign_teacher_to_course(teacher1,course1)
```

```
demo ×
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\Pychar
Teacher Ms. Sneha assigned to course Python.
```

• RecordPayment(student: Student, amount: decimal, paymentDate: DateTime): Records a payment made by a student.

```
def record_payment(self, student, amount, payment_date):
    if student not in self.students:
        raise StudentNotFoundException(student.student_id)
    if amount <= 0:
        raise PaymentValidationException("Payment amount must be greater than zero.")
    payment = Payment(len(self.payments) + 1, student.student_id, amount, payment_date)
    self.payments.append(payment)
    print(f"Payment of {amount} made by {student.first_name} {student.last_name} recorded.")
```

Implementation:

```
from Payment import Payment
from student import Student
from datetime import datetime
from sis import SIS
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-22",  email: "ram.kumar@example.com",  phone_number: "1234567890"
payment1=Payment( payment_id: 1,  student_id: 1,  amount: 100,  payment_date: "2022-04-21")
sis = SIS()
sis.record_payment(student1, payment1.amount, payment1.payment_date)
```

```
demo ×
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\pythonProject\sis\demo.p
Payment of 100 made by Ram Kumar recorded.
```

• GenerateEnrollmentReport(course: Course): Generates a report of students enrolled in a specific course.

```
class SIS:

    1 usage (1 dynamic)
    def generate_enrollment_report(self, course):

        enrolled_students = [student for enrollment in self.enrollments if enrollment.course_id == course.course_id
                             for student in self.students if student.student_id == enrollment.student_id]

        print(f"Enrollment Report for Course: {course.course_name}")

        for student in enrolled_students:
            print(f"Student ID: {student.student_id}")
            print(f"Name: {student.first_name} {student.last_name}")
            print(f"Email: {student.email}")
```

Implementation:

```
sis.enrollments.extend([
    Enrollment( enrollment_id: 1, student1.student_id, course1.course_id, datetime.now()),
    Enrollment( enrollment_id: 2, student2.student_id, course1.course_id, datetime.now()),
    Enrollment( enrollment_id: 3, student3.student_id, course2.course_id, datetime.now()),
    Enrollment( enrollment_id: 4, student4.student_id, course2.course_id, datetime.now())
])
sis.generate_enrollment_report(course1)
sis.generate_enrollment_report(course2)
```

```
Enrollment Report for Course: Python
Student ID: 1
Name: Ram Kumar
Email: ram.kumar@example.com
Student ID: 2
Name: Sita Sharma
Email: sita.sharma@example.com
Enrollment Report for Course: Java
Student ID: 3
Name: Amit Singh
Email: amit.singh@example.com
Student ID: 4
Name: Priya Patel
Email: priya.patel@example.com
```

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

```
class SIS:
    def generate_payment_report(self, student):

        student_payments = [payment for payment in self.payments if payment.student_id == student.student_id]
        print(f"Payment Report for Student: {student.first_name} {student.last_name}")

        for payment in student_payments:
            print(f"Payment ID: {payment.payment_id}")
            print(f"Amount: {payment.amount}")
            print(f"Payment Date: {payment.payment_date}")
```

Implementation:

```
from sis import SIS
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-22",  email: "ram.kumar@example.com",  phone_number: "123456789
student2 = Student( student_id: 2,  first_name: "Sita",  last_name: "Sharma",  dob: "2003-05-15",  email: "sita.sharma@example.com",  phone_number: "98765
payment1 = Payment( payment_id: 1,  student_id: 1,  amount: 100,  payment_date: "2022-04-21")
payment2 = Payment( payment_id: 2,  student_id: 2,  amount: 50,  payment_date: "2022-04-25")
payment3 = Payment( payment_id: 3,  student_id: 1,  amount: 75,  payment_date: "2022-04-22")
payment4 = Payment( payment_id: 4,  student_id: 2,  amount: 125,  payment_date: "2022-04-26")
sis = SIS()
sis.students.extend([student1, student2])
sis.payments.extend([payment1, payment2, payment3, payment4])
sis.generate_payment_report(student1)
sis.generate_payment_report(student2)
```

```
Payment Report for Student: Ram Kumar
Payment ID: 1
Amount: 100
Payment Date: 2022-04-21
Payment ID: 3
Amount: 75
Payment Date: 2022-04-22
Payment Report for Student: Sita Sharma
Payment ID: 2
Amount: 50
Payment Date: 2022-04-25
Payment ID: 4
Amount: 125
Payment Date: 2022-04-26
```

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

```
def calculate_course_statistics(self, course):
    num_enrollments = sum(1 for enrollment in self.enrollments if enrollment.course_id == course.course_id)
    total_payments = sum(payment.amount for payment in self.payments if payment.student_id in
                         [enrollment.student_id for enrollment in self.enrollments if
                          enrollment.course_id == course.course_id])
    print(f"Course Statistics for {course.course_name}:")
    print("-----------------------------------------------")
    print(f"Number of Enrollments: {num_enrollments}")
    print(f"Total Payments: {total_payments}")
    print("-----------------------------------------------")
```

Implementation:

```
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-22",  email: "ram.kumar@example.com",  phone_number: "1234567890
student2 = Student( student_id: 2,  first_name: "Sita",  last_name: "Sharma",  dob: "2003-05-15",  email: "sita.sharma@example.com",  phone_number: "987654
course = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Ms. Sneha")
payment1 = Payment( payment_id: 1,  student_id: 1,  amount: 100,  payment_date: "2022-04-21")
payment2 = Payment( payment_id: 2,  student_id: 1,  amount: 50,  payment_date: "2022-04-25")
payment3 = Payment( payment_id: 3,  student_id: 2,  amount: 75,  payment_date: "2022-04-22")
payment4 = Payment( payment_id: 4,  student_id: 2,  amount: 125,  payment_date: "2022-04-26")
sis = SIS()
sis.students.extend([student1, student2])
sis.courses.append(course)
sis.payments.extend([payment1, payment2, payment3, payment4])
sis.enrollments.extend([
    Enrollment( enrollment_id: 1, student1.student_id, course.course_id, datetime.now()),
    Enrollment( enrollment_id: 2, student2.student_id, course.course_id, datetime.now())
])
sis.calculate_course_statistics(course)
```

```
Course Statistics for Python:
-----------------------------------------------
Number of Enrollments: 2
Total Payments: 350
-----------------------------------------------
```

Task 4: Exceptions handling and Custom Exceptions:

• 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.

```
class DuplicateEnrollmentException(Exception):
    def __init__(self, student, course):
        self.student = student
        self.course = course
        super().__init__(
            f"Student {student.first_name} {student.last_name} is already enrolled in course {course.course_name}")
```

Implementation:

```
from Exception import DuplicateEnrollmentException
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-22",  email: "ram.kumar@example.com",  phone_numb
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Ms. Sneha")
sis = SIS()
sis.enrollments.append(Enrollment(enrollment_id=1, student_id=student1, course_id=course1, enrollment_date=datetime.now()))
sis.enrollments.append(Enrollment(enrollment_id=2, student_id=student1, course_id=course1, enrollment_date=datetime.now()))
sis.enroll_student_in_course(student1, course1)
try:
    sis.enroll_student_in_course(student1, course1)
except DuplicateEnrollmentException as e:
    print(f"Error: {str(e)}")
```

```
demo ×
⋮
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\pythonProj
Student Ram Kumar enrolled in course Python.
Error: Student Ram Kumar is already enrolled in course Python
```

• 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)

```
4 usages
class CourseNotFoundException(Exception):

    def __init__(self, course_code):
        self.course_code = course_code
        super().__init__(f"Course with code '{course_code}' not found in the system.")
```

Implementation:

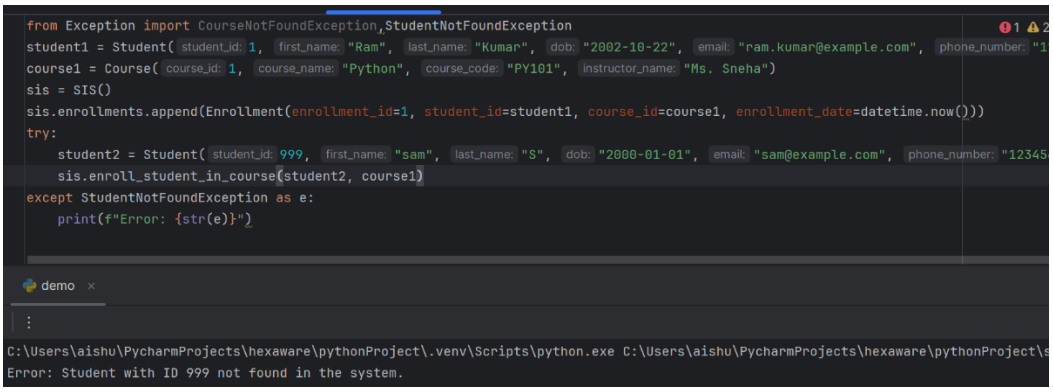
```
from sis import SIS
from Exception import CourseNotFoundException
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-22",  email: "ram.kumar@example.com",  phone_nu
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Ms. Sneha")
sis = SIS()
sis.enrollments.append(Enrollment(enrollment_id=1, student_id=student1, course_id=course1, enrollment_date=datetime.now()))
try:
    course2 = Course( course_id: 999,  course_name: "Java",  course_code: "JV101",  instructor_name: "Ms.Madhu")
    sis.enroll_student_in_course(student1, course2)
except CourseNotFoundException as e:
    print(f"Error: {str(e)}")
```

```
demo ×
⋮
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\pythonPr
Error: Course with code 'JV101' not found in the system.
```

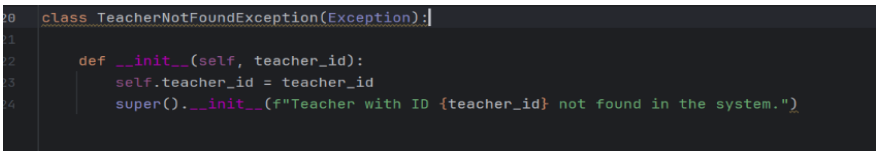
• StudentNotFoundException: Thrown when a student does not exist in the system, and you attempt to perform operations on the student (e.g., enrolling in a course, making a payment).

```
class StudentNotFoundException(Exception):

    def __init__(self, student_id):
        self.student_id = student_id
        super().__init__(f"Student with ID {student_id} not found in the system.")
```

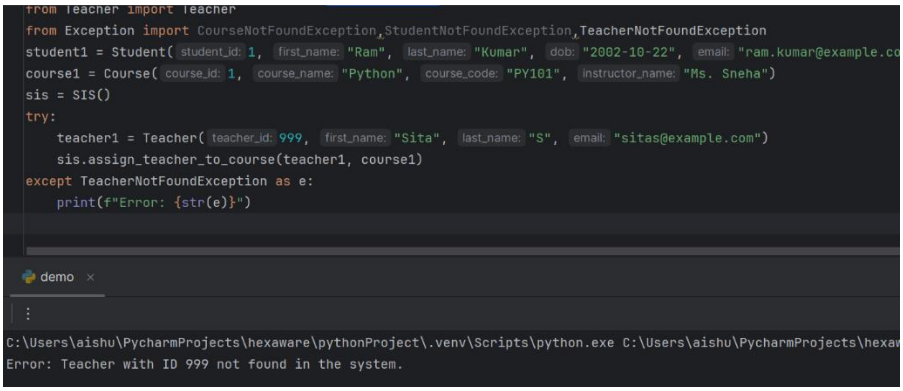
Implementation:



• TeacherNotFoundException: Thrown when a teacher does not exist in the system, and you attempt to assign them to a course.



Implementation:



• PaymentValidationException: Thrown when there is an issue with payment validation, such as an invalid payment amount or payment date.

Implementation:

```
from Exception import CourseNotFoundException,StudentNotFoundException,TeacherNotFoundException,PaymentValidationException
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-22",  email: "ram.kumar@example.com",  phone_numb
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Ms. Sneha")
sis = SIS()
try:
    sis.record_payment(student1, -100, "2022-04-21")
except PaymentValidationException as e:
    print(f"Error: {str(e)}")
```

```
demo ×
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\pythonProj
Error: Payment amount must be greater than zero.
```

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

```
class InvalidStudentDataException(Exception):

    def __init__(self, message):
        super().__init__(message)
```

Implementation:

```
    def update_student_info(self, first_name, last_name, dob, email, phone_number):
        if not self.is_valid_email(email):
            raise InvalidStudentDataException("Invalid email format.")
        self.first_name = first_name
        self.last_name = last_name
        self.dob = dob
        self.email = email
        self.phone_number = phone_number
        print("Student information updated successfully.")
        print(f"{self.first_name} {self.last_name} {self.dob} {self.email} {self.phone_number}")
```

```
demo ×
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProject
Error: Invalid email format.
```

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

```
class InvalidCourseDataException(Exception):

    def __init__(self, message):
        super().__init__(message)
```

Implementation:

```
from sis import SIS
from Teacher import Teacher
from Exception import CourseNotFoundException,StudentNotFoundException,TeacherNotFoundException,PaymentValidationExcepti
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-22",  email: "ram.kumar@example.com",  pho
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Ms. Sneha")
sis = SIS()
try:
    sis.create_course(1, "Python Programming", 40 , "Ms. Teacher")
except InvalidCourseDataException as e:
    print(f"Error: {str(e)}")
```

```
demo ×
⋮
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\pyth
Error: Invalid course code format.
```

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

```
class SIS:
    def enroll_student_in_course(self, student, course):
        if not student or not course:
            raise InvalidEnrollmentDataException("Student or course references are missing.")
        enrollment_id = len(self.enrollments) + 1
        enrollment = Enrollment(enrollment_id, student.student_id, course.course_id, datetime.now())
        self.enrollments.append(enrollment)
        print(f"Student {student.first_name} {student.last_name} enrolled in course {course.course_name}.")
```

Implementation:

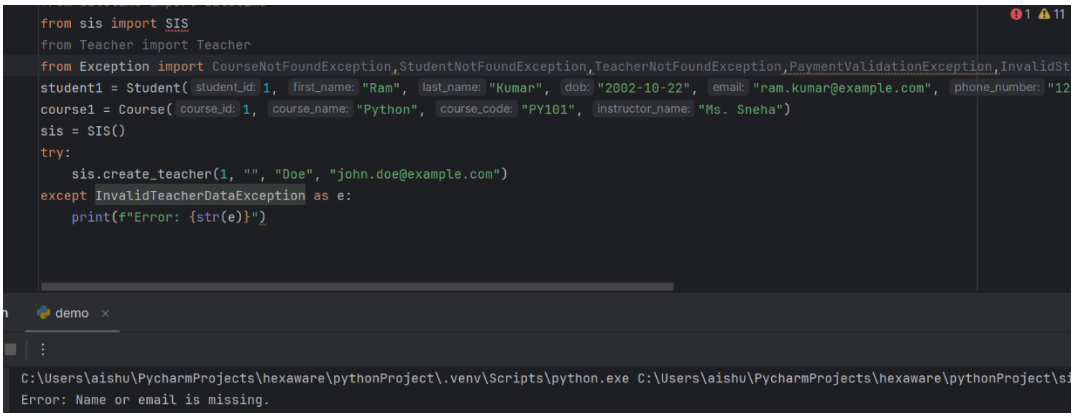
```
from sis import SIS
from Teacher import Teacher
from Exception import CourseNotFoundException,StudentNotFoundException,TeacherNotFoundException,PaymentValidationExcept
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-22",  email: "ram.kumar@example.com",  ph
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Ms. Sneha")
sis = SIS()
try:
    sis.enroll_student_in_course(student1, None)
except InvalidEnrollmentDataException as e:
    print(f"Error: {str(e)}")
```

```
demo ×
⋮
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\py
Error: Student or course references are missing.
```

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

```
class SIS:
    def create_teacher(self, teacher_id, first_name, last_name, email):
        if not first_name or not last_name or not email:
            raise InvalidTeacherDataException("Name or email is missing.")

        teacher = Teacher(teacher_id, first_name, last_name, email)
        self.teachers.append(teacher)
        print("Teacher created successfully.")
```

Implementation:



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

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

```
class Student:
    def __init__(self, student_id, first_name, last_name, dob, email, phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.dob = dob
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []
        self.payments=[]
```

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

```
class Course:
    def __init__(self, course_id, course_name, course_code, instructor_name):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
        self.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:
    def __init__(self, enrollment_id, student, course, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student = student
        self.course = course
        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 object Example: List AssignedCourses { get; set; }

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

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

```
class Payment:
    def __init__(self, payment_id, student, amount, payment_date):
        self.payment_id = payment_id
        self.student = student
        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.

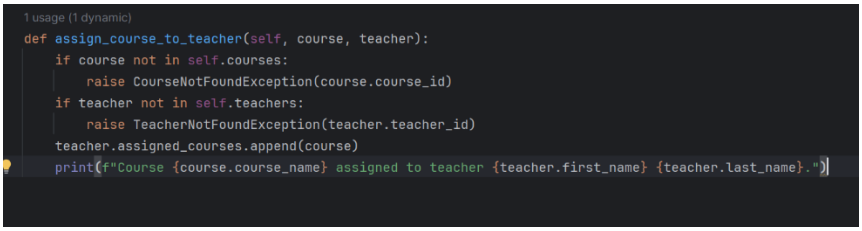
```
class SIS:

    2 usages (2 dynamic)
    def add_enrollment(self, student, course, enrollment_date):
        enrollments = Enrollment(len(self.enrollments) + 1, student, course, enrollment_date)
        student.enrollments.append(enrollments)
        course.enrollments.append(enrollments)
        self.enrollments.append(enrollments)
```

Implementation:

```
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-22",  email: "ram.kumar@example.com",  phone_number: "1234567890"
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Ms. Sneha")
student2 = Student( student_id: 2,  first_name: "Prem",  last_name: "Kumar",  dob: "2002-10-22",  email: "prem.kumar@example.com",  phone_number: "987654321
course2 = Course( course_id: 2,  course_name: "Java",  course_code: "JV101",  instructor_name: "Ms. Madhu")
sis = SIS()
sis.students.append(student1)
sis.students.append(student2)
sis.courses.append(course1)
sis.courses.append(course2)
sis.add_enrollment(student1, course1, datetime.now())
sis.add_enrollment(student2, course2, datetime.now())
sis.print_enrollments()
```

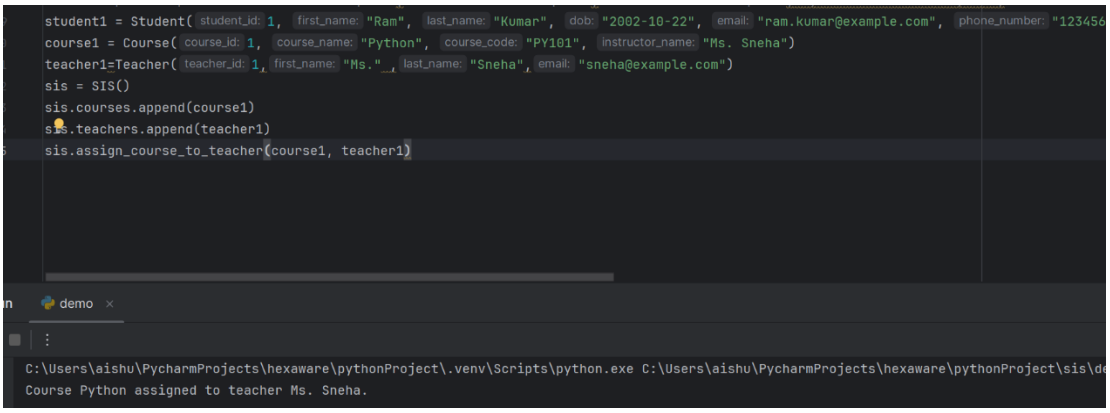
```
Enrollments:
Enrollment ID: 1
Student: Ram Kumar
Course: Python
Enrollment Date: 2024-04-27 16:41:57.836625
----------------------------------------------
Enrollment ID: 2
Student: Prem Kumar
Course: Java
Enrollment Date: 2024-04-27 16:41:57.836625
----------------------------------------------
```

• 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.



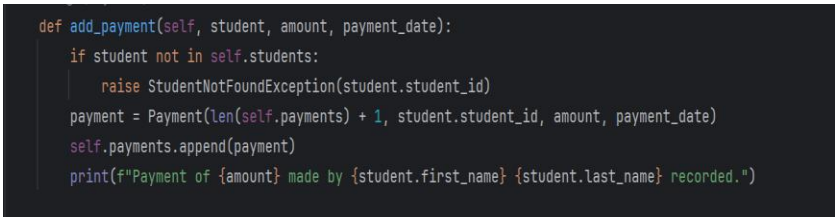
```
1 usage (1 dynamic)
def assign_course_to_teacher(self, course, teacher):
    if course not in self.courses:
        raise CourseNotFoundException(course.course_id)
    if teacher not in self.teachers:
        raise TeacherNotFoundException(teacher.teacher_id)
    teacher.assigned_courses.append(course)
    print(f"Course {course.course_name} assigned to teacher {teacher.first_name} {teacher.last_name}.")
```

Implementation:



```
student1 = Student( student_id: 1,  first_name: "Ram",  last_name: "Kumar",  dob: "2002-10-22",  email: "ram.kumar@example.com",  phone_number: "123456
course1 = Course( course_id: 1,  course_name: "Python",  course_code: "PY101",  instructor_name: "Ms. Sneha")
teacher1=Teacher( teacher_id: 1,  first_name: "Ms.",  last_name: "Sneha",  email: "sneha@example.com")
sis = SIS()
sis.courses.append(course1)
sis.teachers.append(teacher1)
sis.assign_course_to_teacher(course1, teacher1)

demo
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\pythonProject\sis\de
Course Python assigned to teacher Ms. Sneha.
```

• 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.



```
def add_payment(self, student, amount, payment_date):
    if student not in self.students:
        raise StudentNotFoundException(student.student_id)
    payment = Payment(len(self.payments) + 1, student.student_id, amount, payment_date)
    self.payments.append(payment)
    print(f"Payment of {amount} made by {student.first_name} {student.last_name} recorded.")
```

Implementation:

```
from datetime import datetime
from sis import SIS
from Teacher import Teacher
from Exception import CourseNotFoundException,StudentNotFoundException,TeacherNotFoundException,Payme
student1 = Student( student_id: 1, first_name: "Ram", last_name: "Kumar", dob: "2002-10-22", email: "ram.kumar
course1 = Course( course_id: 1, course_name: "Python", course_code: "PY101", instructor_name: "Ms. Sneha")
teacher1=Teacher( teacher_id: 1, first_name: "Ms.", last_name: "Sneha", email: "sneha@example.com")
sis = SIS()
sis.students.append(student1)
sis.add_payment(student1, 100, datetime.now())
sis.print_payment_list()
```

```
demo ×
⋮
Payment of 100 made by Ram Kumar recorded.
Payment List:
Payment ID: 1, Student ID: 1, Amount: 100, Date: 2024-04-27 16:55:35.419214
```

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

```
def get_enrollments_for_student(self, student):
    student_enrollments = []
    for enrollment in self.enrollments:
        if enrollment.student== student:
            student_enrollments.append(enrollment)
    return student_enrollments
```

Implementation:

```
student1 = Student( student_id: 1, first_name: "Ram", last_name: "Kumar", dob: "2002-10-22", email: "ram.kumar@example.com", phone_numb
course1 = Course( course_id: 1, course_name: "Python", course_code: "PY101", instructor_name: "Ms. Sneha")
teacher1=Teacher( teacher_id: 1, first_name: "Ms.", last_name: "Sneha", email: "sneha@example.com")
course2=Course( course_id: 2, course_name: "Java", course_code: "JV101", instructor_name: "Ms.Madhu")
sis = SIS()
enrollment1 = Enrollment( enrollment_id: 1, student1, course1, datetime.now())
enrollment2 = Enrollment( enrollment_id: 2, student1, course2, datetime.now())
sis.enrollments.extend([enrollment1, enrollment2])
student1_enrollments = sis.get_enrollments_for_student(student1)
for enrollment in student1_enrollments:
    print(f"Enrollment ID: {enrollment.enrollment_id}, Enrollment Date: {enrollment.enrollment_date}")
```

```
demo ×
⋮
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexaware\pythonPro
Enrollment ID: 1, Enrollment Date: 2024-04-27 17:11:39.412271
Enrollment ID: 2, Enrollment Date: 2024-04-27 17:11:39.412271
```

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

```
def get_courses_for_teacher(self, teacher):
    assigned_courses = []
    for course in self.courses:
        if course.instructor_name == f"{teacher.first_name}{teacher.last_name}":
            assigned_courses.append(course)
    return assigned_courses
```

Implementation:

```
from sis import SIS
from Teacher import Teacher
from Exception import CourseNotFoundException,StudentNotFoundException,TeacherNotFoundException,PaymentValidationE
student1 = Student( student_id: 1, first_name: "Ram", last_name: "Kumar", dob: "2002-10-22", email: "ram.kumar@example.com"
course1 = Course( course_id: 1, course_name: "Python", course_code: "PY101", instructor_name: "Ms.Sneha")
teacher1=Teacher( teacher_id: 1, first_name: "Ms.", last_name: "Sneha", email: "sneha@example.com")
course2=Course( course_id: 2, course_name: "Java", course_code: "JV101", instructor_name: "Ms.Madhu")
sis=SIS()
sis.assign_course_to_teacher(course1, teacher1)
sis.assign_course_to_teacher(course2, teacher1)
assigned_courses = sis.get_courses_for_teacher(teacher1)
for course in assigned_courses:
    print(f"Course ID: {course.course_id}, Name: {course.course_name}")
```

```
demo ×
⋮
C:\Users\aishu\PycharmProjects\hexaware\pythonProject\.venv\Scripts\python.exe C:\Users\aishu\PycharmProjects\hexawar
Course Python assigned to teacher Ms. Sneha.
Course Java assigned to teacher Ms. Sneha.
```

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

```
import mysql.connector

def initialize_database():
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            port="3306"
        )

        cursor = con.cursor()

        cursor.execute("CREATE DATABASE IF NOT EXISTS SIS")
        cursor.execute("USE SIS")

        tables = [
            """
            CREATE TABLE IF NOT EXISTS students (
                student_id INT AUTO_INCREMENT PRIMARY KEY,
                first_name VARCHAR(100),
```

```
            last_name VARCHAR(100),
            dob DATE,
            email VARCHAR(100),
            phone_number VARCHAR(100)
        )
        """,
        """
        CREATE TABLE IF NOT EXISTS courses (
            course_id INT AUTO_INCREMENT PRIMARY KEY,
            course_name VARCHAR(100),
            course_code VARCHAR(20),
            instructor_name VARCHAR(100)
        )
        """,
        """
        CREATE TABLE IF NOT EXISTS teacher (
            teacher_id INT AUTO_INCREMENT PRIMARY KEY,
            first_name VARCHAR(100),
            last_name VARCHAR(20),
            email VARCHAR(100)
        )
        """,
        """
        CREATE TABLE IF NOT EXISTS enrollments (
            enrollment_id INT AUTO_INCREMENT PRIMARY KEY,
            student_id INT,
            course_id INT,
            enrollment_date DATE,
            FOREIGN KEY (student_id) REFERENCES students(student_id),
            FOREIGN KEY (course_id) REFERENCES courses(course_id)
        )
        """,
        """
        CREATE TABLE IF NOT EXISTS payments (
            payment_id INT AUTO_INCREMENT PRIMARY KEY,
            student_id INT,
            amount INT,
            payment_date DATE,
            FOREIGN KEY (student_id) REFERENCES students(student_id)
        )
        """
    ]

    for table in tables:
        cursor.execute(table)

    con.commit()
    cursor.close()
    con.close()

    print("Database initialization completed successfully")

except mysql.connector.Error as error:
    print("Error while connecting to MySQL", error)
```

```
mysql> use sis;
Database changed
mysql> show tables;
+----------------+
| Tables_in_sis |
+---------------+
| courses       |
| enrollments   |
| payments      |
| students      |
| teacher       |
+---------------+
5 rows in set (0.03 sec)
```

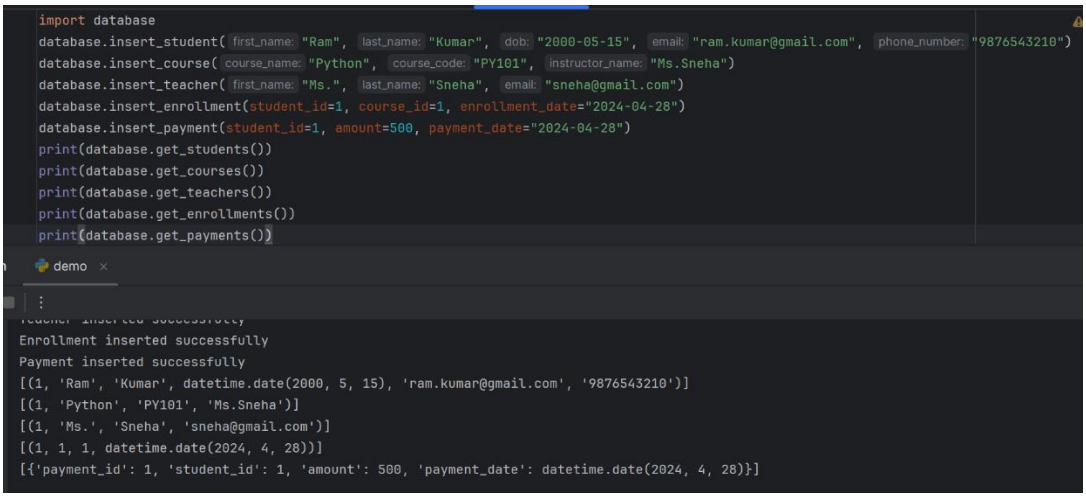
Data Retrieval: Implement methods to retrieve data from the database. Users should be able to request information about students, courses, enrollments, teachers, or payments. Ensure that the data retrieval methods handle exceptions and edge cases gracefully.

```python
def get_students():
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        cursor.execute("SELECT * FROM students")
        students = cursor.fetchall()

        cursor.close()
        con.close()

        return students

    except mysql.connector.Error as error:
        print("Error while connecting to MySQL", error)
        return []

def get_courses():
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        cursor.execute("SELECT * FROM courses")
        courses = cursor.fetchall()

        cursor.close()
        con.close()

        return courses

    except mysql.connector.Error as error:
        print("Error while connecting to MySQL", error)
        return []
```

```
def get_teachers():
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        cursor.execute("SELECT * FROM teacher")
        teacher = cursor.fetchall()

        cursor.close()
        con.close()

        return teacher

    except mysql.connector.Error as error:
        print("Error while connecting to MySQL", error)
        return []

def get_enrollments():
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        cursor.execute("SELECT * FROM enrollments")
        enrollments= cursor.fetchall()

        cursor.close()
        con.close()
        for enrollment in enrollments:
              print(enrollment)
        return enrollments

    except mysql.connector.Error as error:
        print("Error while connecting to MySQL", error)
        return []

def get_payments():
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor(dictionary=True)
        cursor.execute("SELECT * FROM payments")
```

```
        payments = cursor.fetchall()

        cursor.close()
        con.close()

        return payments

    except mysql.connector.Error as error:
        print("Error while connecting to MySQL", error)
        return []
```

Implementation:



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

```
def insert_student(first_name,last_name,dob,email,phone_number):
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        query = "INSERT INTO students
(first_name,last_name,dob,email,phone_number) VALUES (%s, %s, %s,%s,%s)"
        data = (first_name,last_name,dob,email,phone_number)
        cursor.execute(query, data)

        con.commit()
        cursor.close()
        con.close()
```

```
        print("Student inserted successfully")

    except mysql.connector.Error as error:
        print("Error while inserting student", error)

def insert_course(course_name,course_code,instructor_name):
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        query = "INSERT INTO courses(course_name,course_code,instructor_name)
VALUES (%s, %s, %s)"
        data = (course_name,course_code,instructor_name)
        cursor.execute(query, data)

        con.commit()
        cursor.close()
        con.close()

        print("Courses inserted successfully")

    except mysql.connector.Error as error:
        print("Error while inserting course:", error)

def insert_enrollment(student_id, course_id, enrollment_date):
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        query = "INSERT INTO enrollments (student_id, course_id,
enrollment_date) VALUES (%s, %s, %s)"
        data = (student_id, course_id, enrollment_date)
        cursor.execute(query, data)

        con.commit()
        cursor.close()
        con.close()

        print("Enrollment inserted successfully")

    except mysql.connector.Error as error:
        print("Error while inserting enrollment:", error)

def insert_teacher(first_name,last_name, email):
    try:
        con = mysql.connector.connect(
            host="localhost",
```

```
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        query = "INSERT INTO teacher (first_name,last_name, email) VALUES
(%s, %s, %s)"
        data = (first_name,last_name, email)
        cursor.execute(query, data)

        con.commit()
        cursor.close()
        con.close()

        print("Teacher inserted successfully")

    except mysql.connector.Error as error:
        print("Error while inserting teacher:", error)


def insert_payment(student_id, amount, payment_date):
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        query = "INSERT INTO payments (student_id, amount, payment_date)
VALUES (%s, %s, %s)"
        data = (student_id, amount, payment_date)
        cursor.execute(query, data)

        con.commit()
        cursor.close()
        con.close()

        print("Payment inserted successfully")

    except mysql.connector.Error as error:
        print("Error while inserting payment:", error)


def update_student_info(student_id=None, first_name=None, last_name=None,
dob=None, email=None, phone_number=None):
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
```

```
        query = "UPDATE students SET "
        update_values = []
        data = {}
        if first_name is not None:
            update_values.append("first_name = %(first_name)s")
            data["first_name"] = first_name
        if last_name is not None:
            update_values.append("last_name = %(last_name)s")
            data["last_name"] = last_name
        if dob is not None:
            update_values.append("dob = %(dob)s")
            data["dob"] = dob
        if email is not None:
            update_values.append("email = %(email)s")
            data["email"] = email
        if phone_number is not None:
            update_values.append("phone_number = %(phone_number)s")
            data["phone_number"] = phone_number

        if not update_values:
            print("No values provided for update")
            return

        query += ", ".join(update_values)
        query += " WHERE student_id = %(student_id)s"
        data["student_id"] = student_id

        cursor.execute(query, data)

        con.commit()
        cursor.close()
        con.close()

        print("Student information updated successfully")

    except mysql.connector.Error as error:
        print("Error while updating student information:", error)


def update_course_info(course_id=None, course_name=None, course_code=None,
instructor_name=None):
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        query = "UPDATE courses SET "
        update_values = []
        data = {}
        if course_name is not None:
            update_values.append("course_name = %(course_name)s")
            data["course_name"] = course_name
        if course_code is not None:
```

```
            update_values.append("course_code = %(course_code)s")
            data["course_code"] = course_code
        if instructor_name is not None:
            update_values.append("instructor_name = %(instructor_name)s")
            data["instructor_name"] = instructor_name

        if not update_values:
            print("No values provided for update")
            return

        query += ", ".join(update_values)
        query += " WHERE course_id = %(course_id)s"
        data["course_id"] = course_id

        cursor.execute(query, data)

        con.commit()
        cursor.close()
        con.close()

        print("Course information updated successfully")

    except mysql.connector.Error as error:
        print("Error while updating course information:", error)


def update_enrollment_info(enrollment_id=None, student_id=None,
course_id=None, enrollment_date=None):
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        query = "UPDATE enrollment SET "
        update_values = []
        data = {}
        if student_id is not None:
            update_values.append("student_id = %(student_id)s")
            data["student_id"] = student_id
        if course_id is not None:
            update_values.append("course_id = %(course_id)s")
            data["course_id"] = course_id
        if enrollment_date is not None:
            update_values.append("enrollment_date = %(enrollment_date)s")
            data["enrollment_date"] = enrollment_date

        if not update_values:
            print("No values provided for update")
            return

        query += ", ".join(update_values)
        query += " WHERE enrollment_id = %(enrollment_id)s"
        data["enrollment_id"] = enrollment_id
```

```
        cursor.execute(query, data)

        con.commit()
        cursor.close()
        con.close()

        print("Enrollment information updated successfully")

    except mysql.connector.Error as error:
        print("Error while updating enrollment information:", error)


def update_teacher_info(teacher_id=None, first_name=None, last_name=None,
email=None):
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()
        query = "UPDATE teacher SET "
        update_values = []
        data = {}
        if first_name is not None:
            update_values.append("first_name = %(first_name)s")
            data["first_name"] = first_name
        if last_name is not None:
            update_values.append("last_name = %(last_name)s")
            data["last_name"] = last_name
        if email is not None:
            update_values.append("email = %(email)s")
            data["email"] = email

        if not update_values:
            print("No values provided for update")
            return

        query += ", ".join(update_values)
        query += " WHERE teacher_id = %(teacher_id)s"
        data["teacher_id"] = teacher_id

        cursor.execute(query, data)

        con.commit()
        cursor.close()
        con.close()

        print("Teacher information updated successfully")

    except mysql.connector.Error as error:
        print("Error while updating teacher information:", error)
```

```
def update_payment_info(payment_id=None, student_id=None, amount=None,
payment_date=None):
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()

        # Constructing the query string
        query = "UPDATE payments SET "
        update_values = []
        data = {}
        if student_id is not None:
            update_values.append("student_id = %(student_id)s")
            data["student_id"] = student_id
        if amount is not None:
            update_values.append("amount = %(amount)s")
            data["amount"] = amount
        if payment_date is not None:
            update_values.append("payment_date = %(payment_date)s")
            data["payment_date"] = payment_date

        # Check if any update values were provided
        if not update_values:
            print("No values provided for update")
            return

        query += ", ".join(update_values)
        query += " WHERE payment_id = %(payment_id)s"
        data["payment_id"] = payment_id

        cursor.execute(query, data)

        con.commit()
        cursor.close()
        con.close()

        print("Payment information updated successfully")

    except mysql.connector.Error as error:
        print("Error while updating payment information:", error)
```

Implementation:

```
import database
database.insert_student( first_name: "Ram", last_name: "Kumar", dob: "2000-05-15", email: "ram.kumar@gmail.com", phone_number: "9876543210")
database.insert_course( course_name: "Python", course_code: "PY101", instructor_name: "Ms.Sneha")
database.insert_teacher( first_name: "Ms.", last_name: "Sneha", email: "sneha@gmail.com")
database.insert_enrollment(student_id=1, course_id=1, enrollment_date="2024-04-28")
database.insert_payment(student_id=1, amount=500, payment_date="2024-04-28")
print(database.get_students())
print(database.get_courses())
print(database.get_teachers())
print(database.get_enrollments())
print(database.get_payments())
```

```
demo
Enrollment inserted successfully
Payment inserted successfully
[(1, 'Ram', 'Kumar', datetime.date(2000, 5, 15), 'ram.kumar@gmail.com', '9876543210')]
[(1, 'Python', 'PY101', 'Ms.Sneha')]
[(1, 'Ms.', 'Sneha', 'sneha@gmail.com')]
[(1, 1, 1, datetime.date(2024, 4, 28))]
[{'payment_id': 1, 'student_id': 1, 'amount': 500, 'payment_date': datetime.date(2024, 4, 28)}]
```

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.

```
def execute_transaction(operations):
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        con.start_transaction()
        cursor = con.cursor()

        for operation in operations:
            cursor.execute(operation)

        con.commit()

        print("Transaction committed successfully")

    except mysql.connector.Error as error:
        con.rollback()
        print("Transaction rolled back due to error:", error)
```

Implementation:

```
operations = [
database.insert_student( first_name: "Prem", last_name: "Kumar", dob: "2000-05-15", email: "prem.kumar@gmail.com", phone_number: "9876543210"),
database.insert_course( course_name: "Java", course_code: "JV101", instructor_name: "Ms.Sudha")
]
database.execute_transaction(operations)
print(database.get_students())
print(database.get_courses())
```

```
Transaction committed successfully
[(1, 'Ram', 'Kumar', datetime.date(2000, 5, 15), 'ram.kumar@gmail.com', '9876543210'), (2, 'Prem', 'Kumar', datetime.date(2000, 5, 15), 'prem.ku
[(1, 'Python', 'PY101', 'Ms.Sneha'), (2, 'Java', 'JV101', 'Ms.Sudha')]
```

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.

```python
def execute_dynamic_query(columns, table, conditions=None, order_by=None):
    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="SIS"
        )

        cursor = con.cursor()

        query = f"SELECT {', '.join(columns)} FROM {table}"

        if conditions:
            query += f" WHERE {' AND '.join(conditions)}"

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

        cursor.execute(query)
        results = cursor.fetchall()
        for row in results:
            print(row)

    except mysql.connector.Error as error:
        print("Error while executing dynamic query:", error)

    finally:
        if cursor:
            cursor.close()
        if con:
            con.close()
```

Implementation:

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.

```
import database
database.insert_student( first_name: "John",  last_name: "Doe",  dob: "1995-08-15",  email: "john.doe@example.com",  phone_number: "123456789")
database.insert_course( course_name: "Introduction to Programming",  course_code: "IP101",  instructor_name: "Ms.Sneha")
database.insert_course( course_name: "Mathematics",  course_code: "MT101",  instructor_name: "Ms.Mala")
database.insert_enrollment(student_id=1, course_id=1, enrollment_date="2024-04-28")
database.insert_enrollment(student_id=1, course_id=2, enrollment_date="2024-04-28")
print(database.get_students())
print(database.get_courses())
print(database.get_enrollments())
```

```
mysql> select * from students;
+------------+------------+-----------+------------+----------------------+--------------+
| student_id | first_name | last_name | dob        | email                | phone_number |
+------------+------------+-----------+------------+----------------------+--------------+
|          1 | John       | Doe       | 1995-08-15 | john.doe@example.com | 123456789    |
+------------+------------+-----------+------------+----------------------+--------------+
1 row in set (0.00 sec)

mysql> select * from courses;
+-----------+-----------------------------+-------------+-----------------+
| course_id | course_name                 | course_code | instructor_name |
+-----------+-----------------------------+-------------+-----------------+
|         1 | Introduction to Programming | IP101       | Ms.Sneha        |
|         2 | Mathematics                 | MT101       | Ms.Mala         |
+-----------+-----------------------------+-------------+-----------------+
2 rows in set (0.00 sec)
```

Task 9:

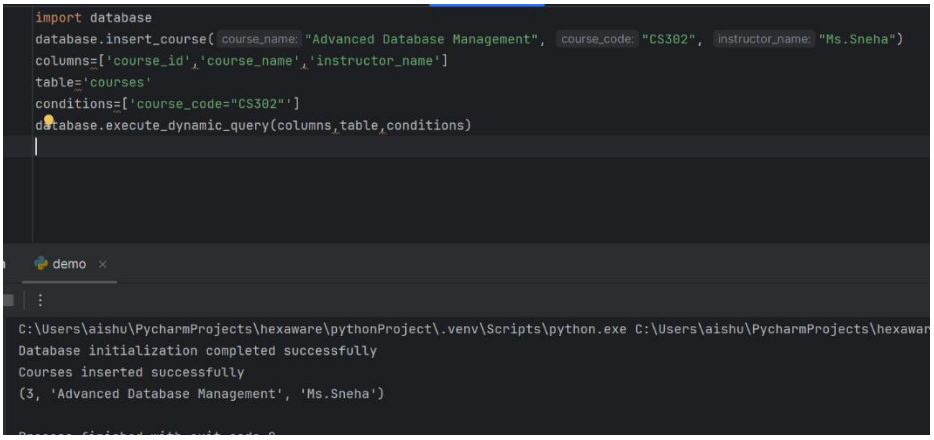
Teacher Assignment In this task, a new teacher, Sarah Smith, is assigned to teach a course. The system needs to update the course record to reflect the teacher assignment.

Teacher's Details: • Name: Sarah Smith • Email: sarah.smith@example.com • Expertise: Computer Science

Course to be assigned: • Course Name: Advanced Database Management • Course Code: CS302

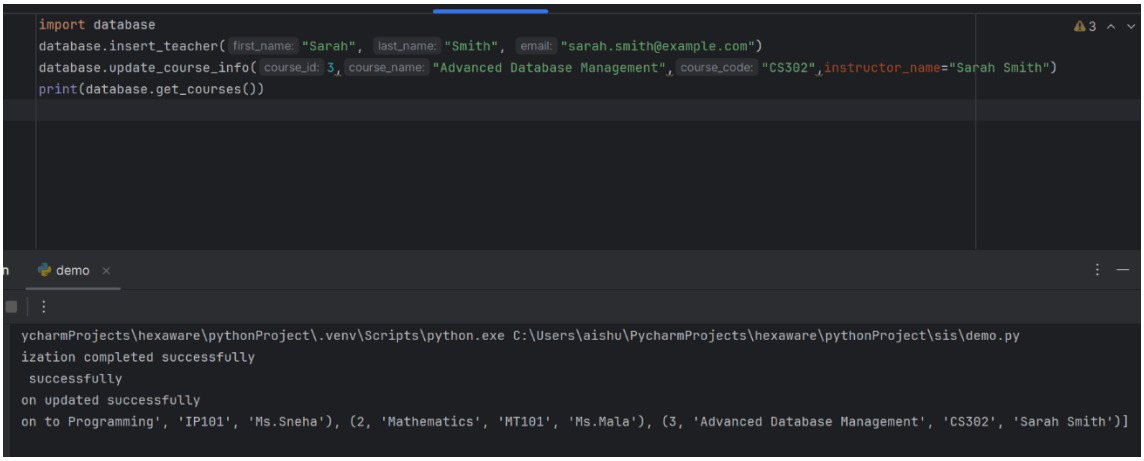
The system should perform the following tasks:

• 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.



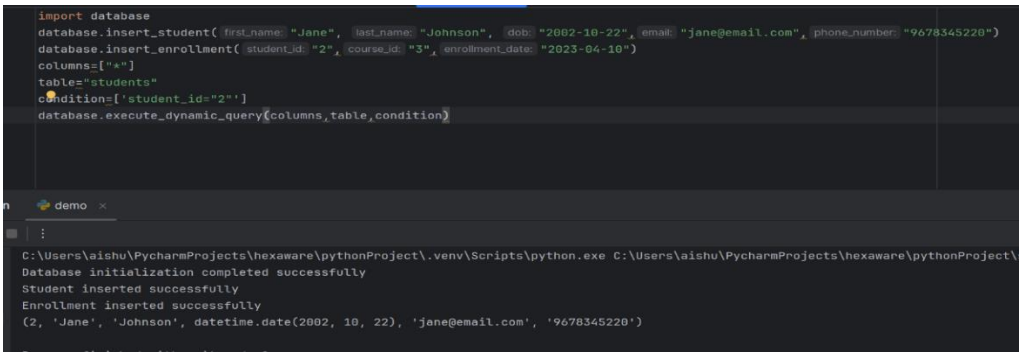
Task 10:

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.

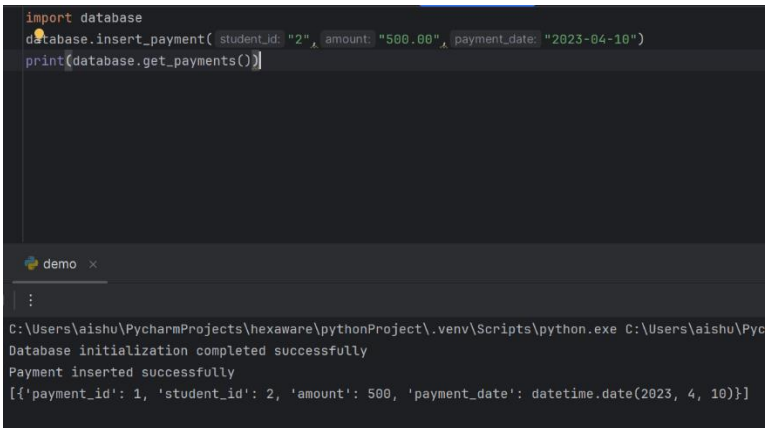
Jane Johnson's details: • Student ID: 101 • Payment Amount: $500.00 • Payment Date: 2023-04-10

The system should perform the following tasks:

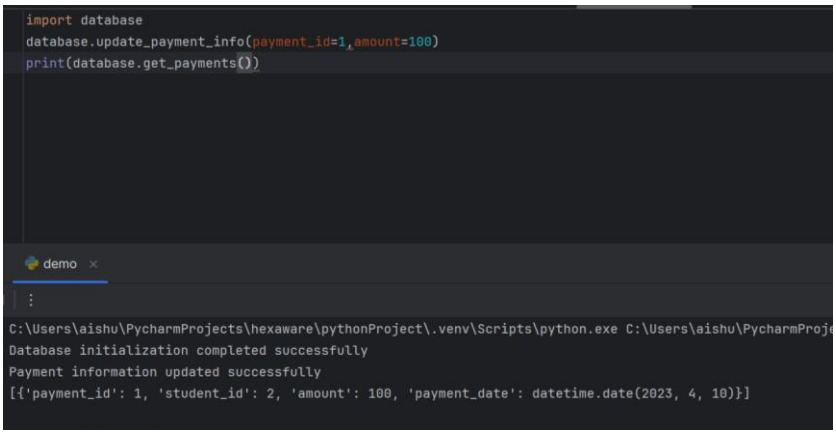
• Retrieve Jane Johnson's student record from the database based on her student ID.



• 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



Task 11:

Enrollment Report Generation

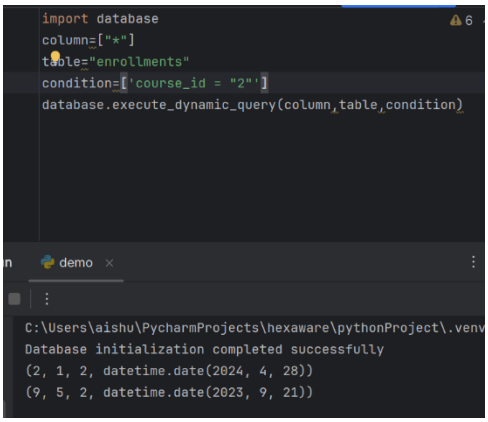
In this task, an administrator requests an enrollment report for a specific course, "Computer Science 101."

The system needs to retrieve enrollment information from the database and generate a report. Course to generate the report for:

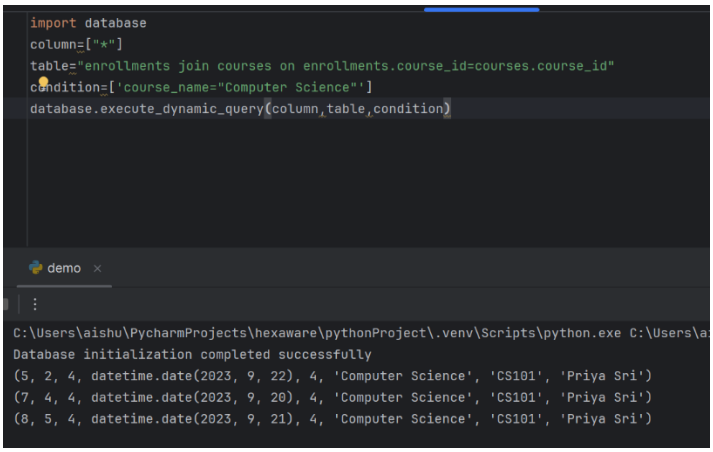
• Course Name: Computer Science 101

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