Python Assignment

Student Information System

Task 1 Defining 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*

```
class Student:
   student_id = ''
   first_name = ''
   last_name = ''
   date_of_birth = ''
   email = ''
   phone_number = ''
```

Course class *with the following attributes:*
- *Course ID*
- *Course Name*
- *Course Code*

```
class Course:
   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*

```
class Enrollments:
   enrollment_id = ''
   student_id = Student.student_id
   course_id = Course.course_id
   enrollment_date = ''
```

Teacher class *with the following attributes:*
• Teacher ID
• First Name
• Last Name
• Email

```
class Teacher:
   teacher_id = ''
   first_name = ''
   last_name = ''
   email = ''
```

Payment class *with the following attributes:*
• Payment ID
• Student ID (reference to a Student)
• Amount
• Payment Date

```
class Payment:
   payment_id = ''
   student_id = Student.student_id
   amount = ''
   payment_date = ''
```

TASK 2: Implement Constructors

Student Class Constructor

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

Course Class Constructor

```
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
```

Enrollments Class Constructor

```
class Enrollments:

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

```
        self.student_id = student.student_id
        self.course_id = course.course_id
        self.enrollment_date = enrollment_date
```

Teacher Class Constructor

```
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
```

Payment Class Constructor

```
class Payment:

   def __int__(self, payment_id, student, amount, payment_date):
        self.payment_id = payment_id
        self.student_id = student.student_id
        self.amount = amount
        self.payment_date = payment_date
```

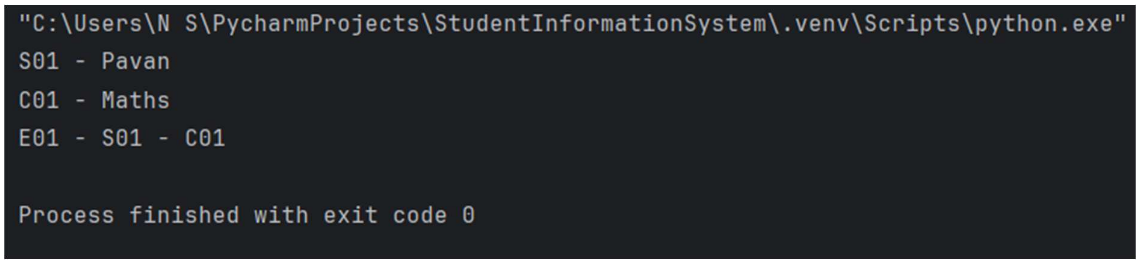
SIS Class Constructor

```
class SIS:

    def __init__(self, name, address, max_students):
        self.name = name
        self.address = address
        self.max_students = max_students
        self.courses = []
```

To invoke these constructors, I have just created some sample objects, with the str functions in the classes Student, Course, Enrollments respectively

```
S1 = Student("S01", "Pavan", "Namana", "06/11/2002", "abc@gamil.com",
"1234567899")
C1 = Course("C01", "Maths", "1001", "Keerthi")
E1 = Enrollments("E01", S1, C1, "10/04/2024")
print(S1)
print(C1)
print(E1)
```

Output:



```
"C:\Users\N S\PycharmProjects\StudentInformationSystem\.venv\Scripts\python.exe"
S01 - Pavan
C01 - Maths
E01 - S01 - C01

Process finished with exit code 0
```

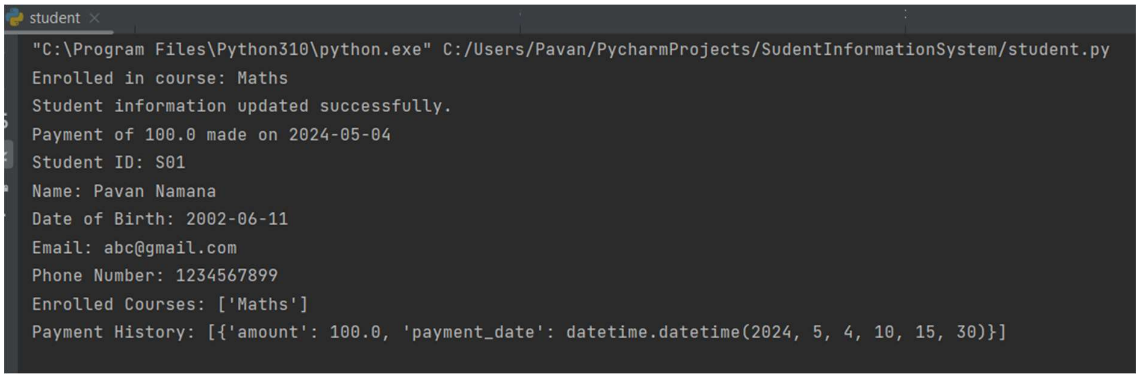
Task 3: Implement Methods

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.*

```python
def enroll_in_course(self, course: Course):

   self.enrolled_courses.append(course)
   print(f"Enrolled in course: {course.course_name}")

def update_student_info(self, first_name: str, last_name: str,
date_of_birth: datetime, email: str,
                       phone_number: str):
   self.first_name = first_name
   self.last_name = last_name
   self.date_of_birth = date_of_birth
   self.email = email
   self.phone_number = phone_number
   print("Student information updated successfully.")

def make_payment(self, amount: float, payment_date: datetime):
   self.payment_history.append({"amount": amount, "payment_date":
payment_date})
   print(f"Payment of {amount} made on {payment_date}")

def display_student_info(self):
   print(f"Student ID: {self.student_id}")
   print(f"Name: {self.first_name} {self.last_name}")
   print(f"Date of Birth: {self.date_of_birth}")
   print(f"Email: {self.email}")
   print(f"Phone Number: {self.phone_number}")

def get_enrolled_courses(self) -> List[Course]:
   return self.enrolled_courses

def get_payment_history(self):
   return self.payment_history
```

Sample Output for the above methods in the class:



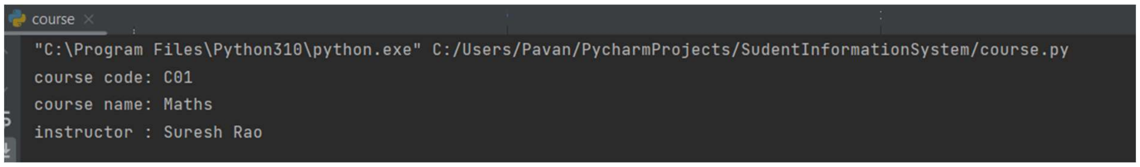
```
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/student.py
Enrolled in course: Maths
Student information updated successfully.
Payment of 100.0 made on 2024-05-04
Student ID: S01
Name: Pavan Namana
Date of Birth: 2002-06-11
Email: abc@gmail.com
Phone Number: 1234567899
Enrolled Courses: ['Maths']
Payment History: [{'amount': 100.0, 'payment_date': datetime.datetime(2024, 5, 4, 10, 15, 30)}]
```

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.*

```
def assignTeacher(self, teacher):
    self.instructor_name = teacher
    self.instructor_list.append(teacher)

def updateCourseInfo(self, courseCode, courseName, instructor):
    self.course_code = courseCode
    self.course_name = courseName
    self.instructor = instructor
    enrolled_students.append(self.student_id)

def displayCourseInfo(self):
    print(f"course code: {self.course_code}")
    print(f"course name: {self.course_name}")
    print(f"instructor : {self.Instructor_name}")

def getEnrollments(self):
    print(self.enrolled_students)

def getTeacher(self):
    print(self.instructor_list)
```

Sample Output:



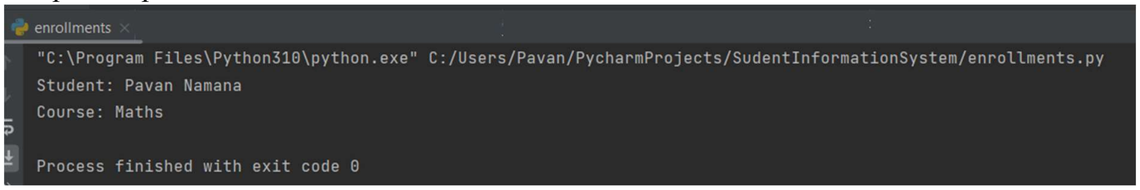
```
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/course.py
course code: C01
course name: Maths
instructor : Suresh Rao
```

Enrollment Class:

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

```
def get_student(self):
    return self.student

def get_course(self):
    return self.course
```

Sample Output:



enrollments

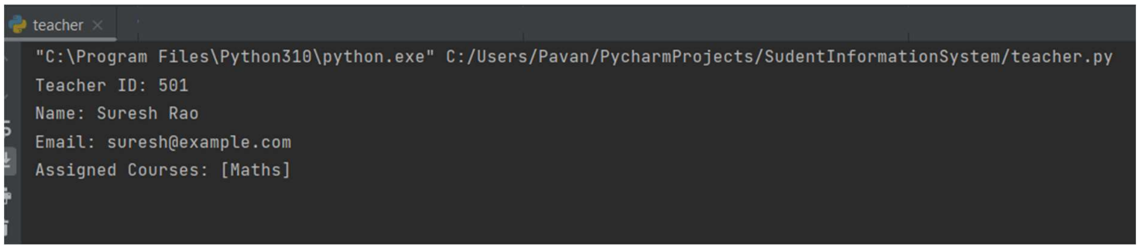
```
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/enrollments.py
Student: Pavan Namana
Course: Maths

Process finished with exit code 0
```

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.*

```
def update_teacher_info(self, name, email, expertise):
    self.name = name
    self.email = email
    self.expertise = expertise


def display_teacher_info(self):
    print("Teacher Information:")
    print("Name:", self.name)
    print("Email:", self.email)
    print("Expertise:", self.expertise)


def get_assigned_courses(self):
    return self.assigned_courses
```

Sample Output:



teacher

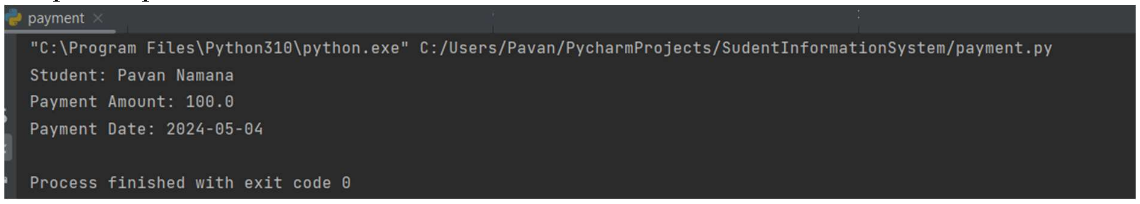
```
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/teacher.py
Teacher ID: 501
Name: Suresh Rao
Email: suresh@example.com
Assigned Courses: [Maths]
```

Payment Class:

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

```
def get_student(self):
    return self.student

def get_payment_amount(self):
    return self.amount

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

Sample Output:



```
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/payment.py
Student: Pavan Namana
Payment Amount: 100.0
Payment Date: 2024-05-04

Process finished with exit code 0
```

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

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

```
def enroll_student_in_course(self, student, course):
    enrollment = (student, course)
    self.enrollments.append(enrollment)
    return enrollment


def assign_teacher_to_course(self, teacher, course):
    assignment = (teacher, course)
    self.course_assignments.append(assignment)
    return assignment


def record_payment(self, student, amount, payment_date):
    payment = (student, amount, payment_date)
    self.payments.append(payment)
    return payment
```

```
def generate_enrollment_report(self, course):
    enrolled_students = [enrollment[0] for enrollment in self.enrollments
if enrollment[1] == course]
    return enrolled_students


def generate_payment_report(self, student):
    student_payments = [payment for payment in self.payments if payment[0]
== student]
    return student_payments


def calculate_course_statistics(self, course):
    enrollments_count = len([enrollment for enrollment in self.enrollments
if enrollment[1] == course])
    total_payments = sum(payment[1] for payment in self.payments if
                         payment[0] in [enrollment[0] for enrollment in
self.enrollments if enrollment[1] == course])
    return enrollments_count, total_payments
```

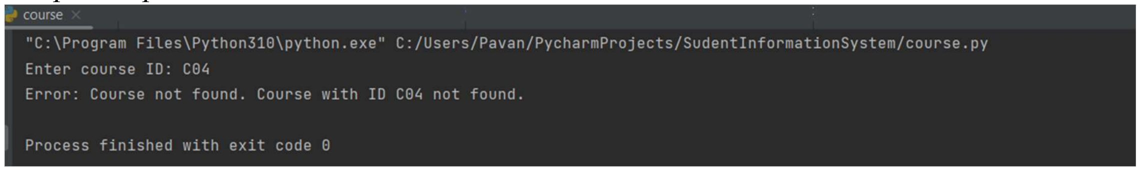
Task 4: Exceptions handling and Custom Exceptions

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

```
class CustomExceptions:

    def duplicate_enrollment(message="Student is already enrolled in the
course."):
        raise Exception(message)

    def course_not_found(message="Course not found."):
        raise Exception(message)
    def student_not_found(message="Student not found."):
        raise Exception(message)

    def teacher_not_found(message="Teacher not found."):
        raise Exception(message)

    def payment_validation_failed(message="Payment validation failed."):
        raise Exception(message)

    def invalid_student_data(message="Invalid student data."):
        raise Exception(message)

    def invalid_course_data(message="Invalid course data."):
        raise Exception(message)

    def invalid_enrollment_data(message="Invalid enrollment data."):
        raise Exception(message)

    def invalid_teacher_data(message="Invalid teacher data."):
        raise Exception(message)

    def insufficient_funds(message="Insufficient funds."):
        raise Exception(message)
```

Sample Output:



```
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/course.py
Enter course ID: C04
Error: Course not found. Course with ID C04 not found.

Process finished with exit code 0
```

Task 5: Collections

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, date_of_birth,
email, phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []
```

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

```
class Course:
    def __init__(self, course_code, course_name, instructor_id,
instructor_name):
        self.course_code = course_code
        self.course_name = course_name
        self.instructor_id = instructor_id
        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, student, course, enrollment_date):
        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 objects.
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, student, amount, payment_date):
        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.*
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.*

```
def add_enrollment(self, student, course, enrollment_date):
    enrollment = Enrollment(student, course, enrollment_date)
    student.enrollments.append(enrollment)
    course.enrollments.append(enrollment)

def assign_course_to_teacher(self, course, teacher):
    teacher.assigned_courses.append(course)

def add_payment(self, student, amount, payment_date):
    payment = Payment(student, amount, payment_date)
    student.payment_history.append(payment)

def get_enrollments_for_student(self, student):
    return [enrollment for enrollment in student.enrollments]

def get_courses_for_teacher(self, teacher):
    return [course for course in teacher.assigned_courses]
```

Create a Driver Program

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

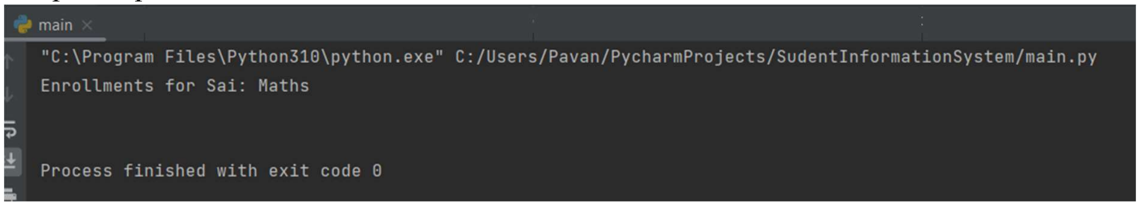
Add References to Your SIS Classes

Ensure that your SIS classes (Student, Course, Enrollment, Teacher, Payment) and the SIS class (if you have one to manage interactions) are defined in separate files within your project or are referenced properly.
If you have defined these classes in separate files, make sure to include using statements in your driver program to access them:

Implement the Main Method
In the console application, the Main method serves as the entry point for your program. This is where you will create instances of your classes, call methods, and interact with your Student Information System.
In the Main method, you create instances of your classes (e.g., Student, Course, and SIS) and then interact with your Student Information System by calling methods and handling exceptions.

```
if __name__ == "__main__":
    sis = SIS()
    sai = Student("S01", "Sai", "Kumar", "2000-01-01", "sai@example.com",
"1234567890")
    maths = Course("C01", "Maths", "T01", "Jane Doe")
    physics = Course("C02", "Physics", "T02", "John Smith")
   anji = Teacher("T02", "Anji", "Reddy", "anji@example.com")
    sis.students.extend(sai), sis.courses.extend([maths, physics]),
sis.teachers.extend(anji)
    sis.add_enrollment(sai, maths, "2024-05-01"),
sis.assign_course_to_teacher(m, anji) for m in [maths, physics]
    sis.add_payment(sai, 100.0, "2024-05-03"), sis.add_payment(krishna,
150.0, "2024-05-04")
    print("Enrollments for Sai:", *[e.course.course_name for e in
sis.get_enrollments_for_student(sai)])
```

Sample output:



```
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/main.py
Enrollments for Sai: Maths


Process finished with exit code 0
```

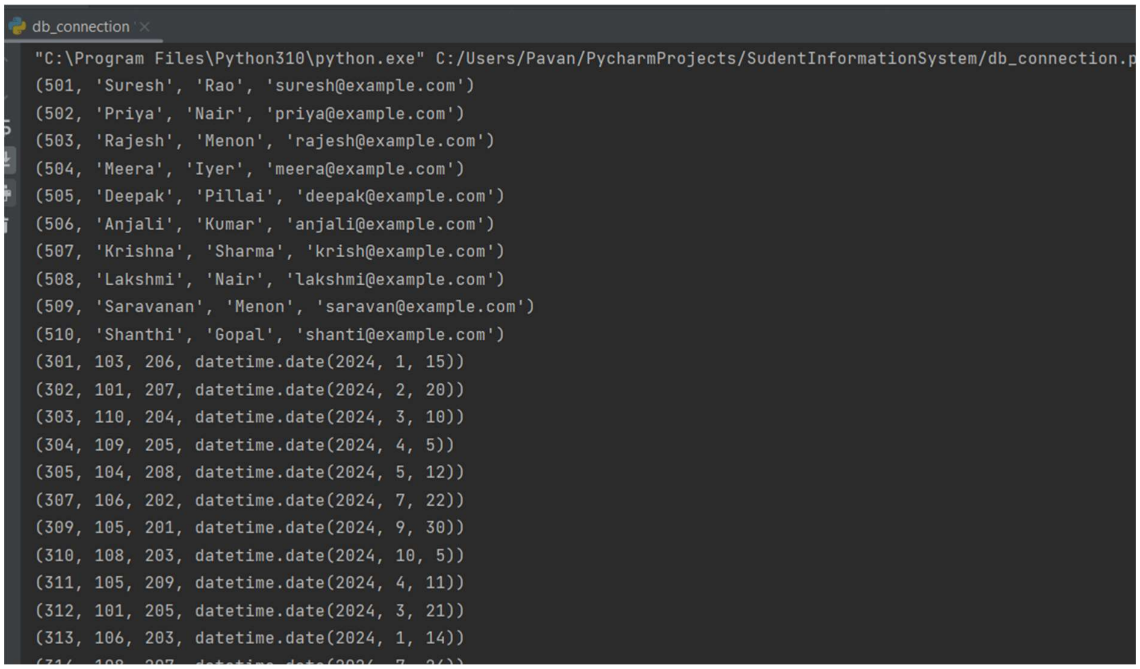
Task 7: Database Connectivity

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

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
import mysql.connector

conn = mysql.connector.connect(
    host= 'localhost',
    user = 'root',
    password = 'root',
    database = 'SISDB'
)

cur = conn.cursor()

cur.execute('Select * from teacher')
for i in cur:
    print(i)

cur.execute('select * from enrollments')
for i in cur:
    print(i)
```

Sample Output:



```
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/db_connection.p
(501, 'Suresh', 'Rao', 'suresh@example.com')
(502, 'Priya', 'Nair', 'priya@example.com')
(503, 'Rajesh', 'Menon', 'rajesh@example.com')
(504, 'Meera', 'Iyer', 'meera@example.com')
(505, 'Deepak', 'Pillai', 'deepak@example.com')
(506, 'Anjali', 'Kumar', 'anjali@example.com')
(507, 'Krishna', 'Sharma', 'krish@example.com')
(508, 'Lakshmi', 'Nair', 'lakshmi@example.com')
(509, 'Saravanan', 'Menon', 'saravan@example.com')
(510, 'Shanthi', 'Gopal', 'shanti@example.com')
(301, 103, 206, datetime.date(2024, 1, 15))
(302, 101, 207, datetime.date(2024, 2, 20))
(303, 110, 204, datetime.date(2024, 3, 10))
(304, 109, 205, datetime.date(2024, 4, 5))
(305, 104, 208, datetime.date(2024, 5, 12))
(307, 106, 202, datetime.date(2024, 7, 22))
(309, 105, 201, datetime.date(2024, 9, 30))
(310, 108, 203, datetime.date(2024, 10, 5))
(311, 105, 209, datetime.date(2024, 4, 11))
(312, 101, 205, datetime.date(2024, 3, 21))
(313, 106, 203, datetime.date(2024, 1, 14))
```

Task 8: Student Enrollment

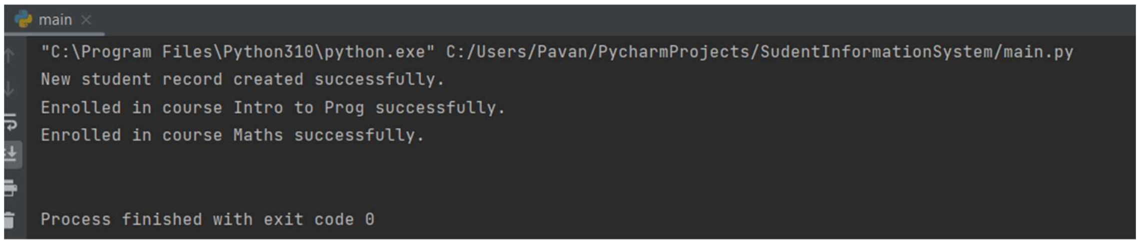
In this task, a new student, John Doe, is enrolling in the SIS. The system needs to record John's information, including his personal details, and enroll him in a few courses. Database connectivity is required to store this information.

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

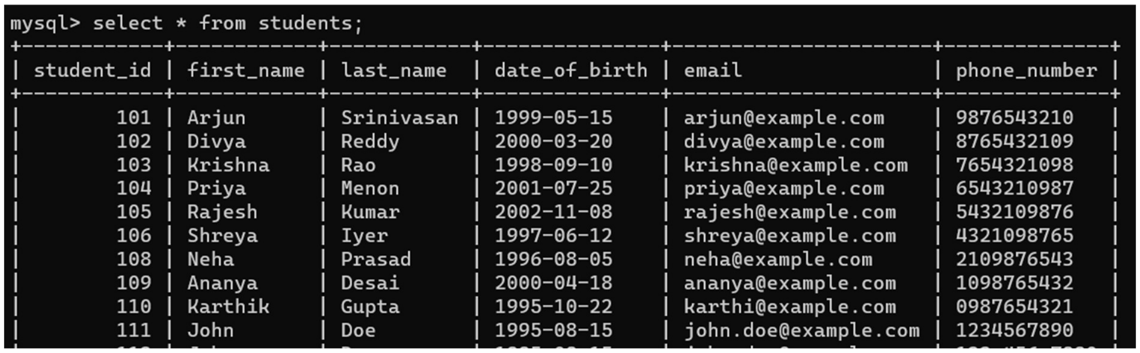
The system should perform the following tasks:
Create a new student record in the database.
Enroll John in the specified courses by creating enrollment records in the database.

Output:



```
main
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/main.py
New student record created successfully.
Enrolled in course Intro to Prog successfully.
Enrolled in course Maths successfully.


Process finished with exit code 0
```

Results in Database:



```
mysql> select * from students;
+------------+------------+------------+---------------+----------------------+--------------+
| student_id | first_name | last_name  | date_of_birth | email                | phone_number |
+------------+------------+------------+---------------+----------------------+--------------+
|        101 | Arjun      | Srinivasan | 1999-05-15    | arjun@example.com    | 9876543210   |
|        102 | Divya      | Reddy      | 2000-03-20    | divya@example.com    | 8765432109   |
|        103 | Krishna    | Rao        | 1998-09-10    | krishna@example.com  | 7654321098   |
|        104 | Priya      | Menon      | 2001-07-25    | priya@example.com    | 6543210987   |
|        105 | Rajesh     | Kumar      | 2002-11-08    | rajesh@example.com   | 5432109876   |
|        106 | Shreya     | Iyer       | 1997-06-12    | shreya@example.com   | 4321098765   |
|        108 | Neha       | Prasad     | 1996-08-05    | neha@example.com     | 2109876543   |
|        109 | Ananya     | Desai      | 2000-04-18    | ananya@example.com   | 1098765432   |
|        110 | Karthik    | Gupta      | 1995-10-22    | karthi@example.com   | 0987654321   |
|        111 | John       | Doe        | 1995-08-15    | john.doe@example.com | 1234567890   |
```

Task 9: Teacher Assignment

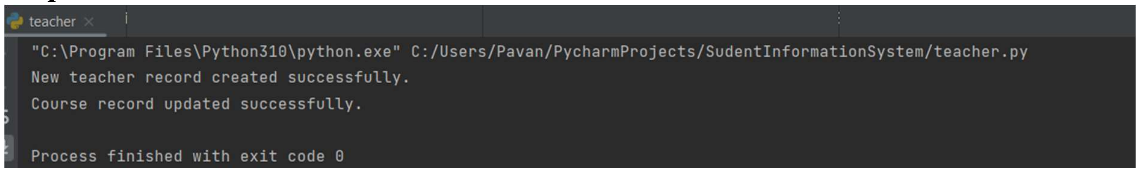
In this task, a new teacher, Sarah Smith, is assigned to teach a course. The system needs to update the course record to reflect the teacher assignment.

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

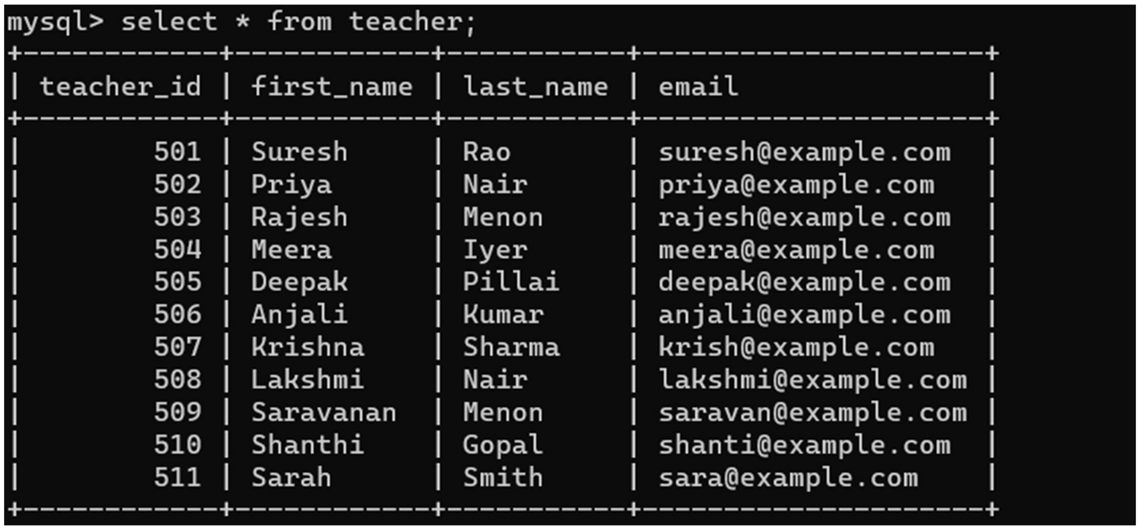
The system should perform the following tasks:
Retrieve the course record from the database based on the course code.
Assign Sarah Smith as the instructor for the course.

Output:



```
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/teacher.py
New teacher record created successfully.
Course record updated successfully.

Process finished with exit code 0
```

Results in database:



```
mysql> select * from teacher;
+------------+------------+-----------+---------------------+
| teacher_id | first_name | last_name | email               |
+------------+------------+-----------+---------------------+
|        501 | Suresh     | Rao       | suresh@example.com  |
|        502 | Priya      | Nair      | priya@example.com   |
|        503 | Rajesh     | Menon     | rajesh@example.com  |
|        504 | Meera      | Iyer      | meera@example.com   |
|        505 | Deepak     | Pillai    | deepak@example.com  |
|        506 | Anjali     | Kumar     | anjali@example.com  |
|        507 | Krishna    | Sharma    | krish@example.com   |
|        508 | Lakshmi    | Nair      | lakshmi@example.com |
|        509 | Saravanan  | Menon     | saravan@example.com |
|        510 | Shanthi    | Gopal     | shanti@example.com  |
|        511 | Sarah      | Smith     | sara@example.com    |
+------------+------------+-----------+---------------------+
```

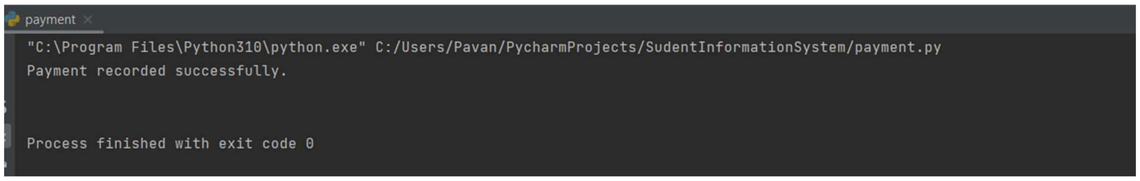
Task 10: Payment Record

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

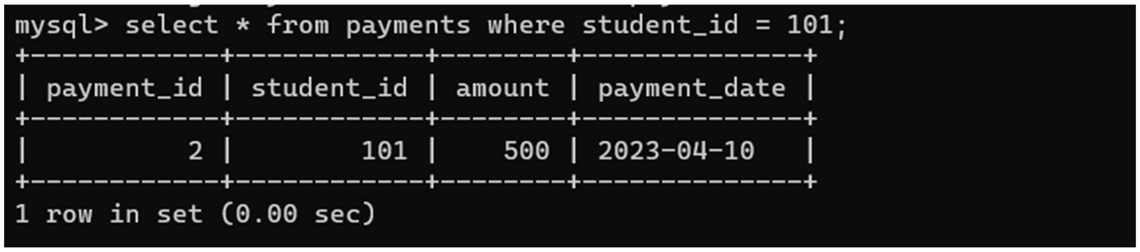
The system should perform the following tasks:
Retrieve Jane Johnson's student record from the database based on her student ID.
Record the payment information in the database, associating it with Jane's student record.
Update Jane's outstanding balance in the database based on the payment amount.

Output:



```
payment
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/payment.py
Payment recorded successfully.


Process finished with exit code 0
```

Results in database:



```
mysql> select * from payments where student_id = 101;
+------------+------------+--------+--------------+
| payment_id | student_id | amount | payment_date |
+------------+------------+--------+--------------+
|          2 |        101 |    500 | 2023-04-10   |
+------------+------------+--------+--------------+
1 row in set (0.00 sec)
```

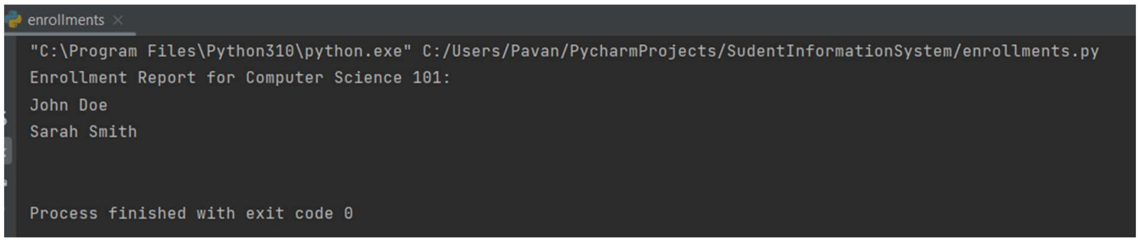
Task 11: Enrollment Report Generation

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.

Output:



```
enrollments
"C:\Program Files\Python310\python.exe" C:/Users/Pavan/PycharmProjects/SudentInformationSystem/enrollments.py
Enrollment Report for Computer Science 101:
John Doe
Sarah Smith


Process finished with exit code 0
```

****