Q1. Write a program that has classes such as Student, Course and Department. Enroll a student in a course of a particular department.

In [1]:

```
class Department:
    def __init__(self, name):
        self.name = name
        self.courses = []

    def add_course(self, course):
        self.courses.append(course)

    def show_courses(self):
        print(f"Courses in {self.name} Department:")
        for course in self.courses:
            print(f" - {course.name} ({course.code})")

class Course:
    def __init__(self, name, code, department):
        self.name = name
        self.code = code
        self.department = department
        department.add_course(self)

    def __str__(self):
        return f"{self.name} [{self.code}]"

class Student:
    def __init__(self, name, student_id):
        self.name = name
        self.student_id = student_id
        self.enrolled_courses = []

    def enroll(self, course):
        self.enrolled_courses.append(course)
        print(f"Success: {self.name} has been enrolled in {course.name}.")

    def show_enrollments(self):
        print(f"\nStudent: {self.name} (ID: {self.student_id})")
        print("Enrolled Courses:")
        if not self.enrolled_courses:
```

```
            print(" - No courses enrolled yet.")
        else:
            for course in self.enrolled_courses:
                print(f" - {course.name} (Dept: {course.department.name})")


#Create Departments
cs_dept = Department("Computer Science")
math_dept = Department("Mathematics")

#Create Courses
c1 = Course("Intro to Python", "CS101", cs_dept)
c2 = Course("Data Structures", "CS102", cs_dept)
c3 = Course("Calculus I", "MATH101", math_dept)

#Create a Student
student1 = Student("Alice Smith", "S12345")

#Display available courses
cs_dept.show_courses()
math_dept.show_courses()
print("-" * 30)

#Enroll the student in courses
student1.enroll(c1)
student1.enroll(c3)

#student's final enrollment status
print("-" * 30)
student1.show_enrollments()
```

```
Courses in Computer Science Department:
 - Intro to Python (CS101)
 - Data Structures (CS102)
Courses in Mathematics Department:
 - Calculus I (MATH101)
------------------------------
Success: Alice Smith has been enrolled in Intro to Python.
Success: Alice Smith has been enrolled in Calculus I.
------------------------------

Student: Alice Smith (ID: S12345)
Enrolled Courses:
 - Intro to Python (Dept: Computer Science)
 - Calculus I (Dept: Mathematics)
```

Q2. Write a program with class Bill. The users have the option to pay the bill either by cheque or by cash. Use the inheritance to model this situation.

In [2]:
```python
class Bill:
    def __init__(self, amount):
        self.amount = amount

    def print_receipt(self):
        print(f"Bill Amount: ${self.amount:.2f}")

class CashBill(Bill):
    def __init__(self, amount, cash_tendered):
        super().__init__(amount)
        self.cash_tendered = cash_tendered

    def pay(self):
        print("\n--- Payment Mode: CASH ---")
        self.print_receipt()
        if self.cash_tendered >= self.amount:
            change = self.cash_tendered - self.amount
            print(f"Cash Tendered: ${self.cash_tendered:.2f}")
            print(f"Change to return: ${change:.2f}")
            print("Payment Successful!")
        else:
            shortfall = self.amount - self.cash_tendered
```

```
            print(f"Cash Tendered: ${self.cash_tendered:.2f}")
            print(f"Insufficient Cash. You need ${shortfall:.2f} more.")

class ChequeBill(Bill):
    def __init__(self, amount, cheque_number, bank_name):
        super().__init__(amount)
        self.cheque_number = cheque_number
        self.bank_name = bank_name

    def pay(self):
        print("\n--- Payment Mode: CHEQUE ---")
        self.print_receipt()
        print(f"Bank Name: {self.bank_name}")
        print(f"Cheque Number: {self.cheque_number}")
        print("Payment Pending Clearance.")


#Paying by Cash
print("User 1 is paying for groceries...")
grocery_bill = CashBill(50.00, 60.00)
grocery_bill.pay()

print("-" * 30)

#Paying by Cheque
print("User 2 is paying for electricity...")
electricity_bill = ChequeBill(120.50, "CHQ102938", "National Bank")
electricity_bill.pay()
```

```
User 1 is paying for groceries...

--- Payment Mode: CASH ---
Bill Amount: $50.00
Cash Tendered: $60.00
Change to return: $10.00
Payment Successful!
------------------------------
User 2 is paying for electricity...

--- Payment Mode: CHEQUE ---
Bill Amount: $120.50
Bank Name: National Bank
Cheque Number: CHQ102938
Payment Pending Clearance.
```

Q3.Write a program to overload the - =operator to subtract two Distance Objects.

In [3]:
```python
class Distance:
    def __init__(self, feet, inches):
        self.feet = feet
        self.inches = inches

    def __str__(self):
        return f"{self.feet} ft {self.inches} in"

    #Overloading the -= operator
    def __isub__(self, other):
        # Subtract the components
        self.feet -= other.feet
        self.inches -= other.inches

        #Handle negative inches by borrowing from feet
        while self.inches < 0:
            self.inches += 12
            self.feet -= 1

        #Handle case where total distance becomes negative
        if self.feet < 0:
            print("Warning: Resulting distance is negative!")
```

```
        return self

d1 = Distance(10, 2)
d2 = Distance(3, 8)

print(f"Initial Distance 1: {d1}")
print(f"Initial Distance 2: {d2}")

print("\nPerforming: d1 -= d2")
d1 -= d2

print(f"Resulting Distance 1: {d1}")
```

```
Initial Distance 1: 10 ft 2 in
Initial Distance 2: 3 ft 8 in

Performing: d1 -= d2
Resulting Distance 1: 6 ft 6 in
```

Q4. Write a program with two classes for calculating the distance. On has distance specified in meters and other has distance in kilometres. These functions take argument of class Distance that converts the distance into kilometres and meters.

In [4]:
```
class DistanceMeters:
    def __init__(self, meters):
        self.meters = meters

    def __str__(self):
        return f"{self.meters} meters"

class DistanceKilometers:
    def __init__(self, kms):
        self.kms = kms

    def __str__(self):
        return f"{self.kms} kilometers"

def convert_to_kilometers(meter_object):

    if isinstance(meter_object, DistanceMeters):
```

```
        km_value = meter_object.meters / 1000.0
        return DistanceKilometers(km_value)
    else:
        print("Error: Argument must be of type DistanceMeters")
        return None

def convert_to_meters(km_object):

    if isinstance(km_object, DistanceKilometers):
        meter_value = km_object.kms * 1000.0
        return DistanceMeters(meter_value)
    else:
        print("Error: Argument must be of type DistanceKilometers")
        return None


#Create a distance in Meters
d_meters = DistanceMeters(1500)  # 1500 meters
print(f"Original Distance: {d_meters}")

#Convert it to Kilometers using the function
d_kms_converted = convert_to_kilometers(d_meters)
print(f"Converted Distance: {d_kms_converted}")

print("-" * 30)

#Create a distance in Kilometers
d_kms = DistanceKilometers(2.5)  # 2.5 km
print(f"Original Distance: {d_kms}")

#Convert it to Meters using the function
d_meters_converted = convert_to_meters(d_kms)
print(f"Converted Distance: {d_meters_converted}")
```

```
Original Distance: 1500 meters
Converted Distance: 1.5 kilometers
------------------------------
Original Distance: 2.5 kilometers
Converted Distance: 2500.0 meters
```