Implement the following hierarchy . The Book function has name, n (number of authors), authors (list of authors), publisher, ISBN, and year as its data members and the derived class has course as its data member. The derived class method overrides (extends) the methods of the base class.

In [1]:
```
class Book:
    def __init__(self, name, n, authors, publisher, ISBN, year):
        self.name = name
        self.n = n
        self.authors = authors
        self.publisher = publisher
        self.ISBN = ISBN
        self.year = year

    def display(self):
        print(f"Name: {self.name}")
        print(f"Number of authors: {self.n}")
        print(f"Authors: {', '.join(self.authors)}")
        print(f"Publisher: {self.publisher}")
        print(f"ISBN: {self.ISBN}")
        print(f"Year: {self.year}")


class CourseBook(Book):
    def __init__(self, name, n, authors, publisher, ISBN, year, course):
        super().__init__(name, n, authors, publisher, ISBN, year)
        self.course = course

    def display(self):
        super().display()
        print(f"Course: {self.course}")


book = Book("The Great Gatsby", 1, ["F. Scott Fitzgerald"], "Scribner", "97807
book.display()

print()

course_book = CourseBook("Data Science from Scratch", 1, ["Joel Grus"], "O'Rei
course_book.display()
```

```
Name: The Great Gatsby
Number of authors: 1
Authors: F. Scott Fitzgerald
Publisher: Scribner
ISBN: 9780743273565
Year: 1925

Name: Data Science from Scratch
Number of authors: 1
Authors: Joel Grus
Publisher: O'Reilly
ISBN: 9781492041139
Year: 2019
Course: Data Science
```

Implement the following hierarchy . The Staff function has name and salary as its data members, the derived class Teaching has subject as its data member and the class NonTeaching has department as its data member. The derived class method overrides (extends) the methods of the base class.

In [2]:

```python
class Staff:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def display(self):
        print(f"Name: {self.name}")
        print(f"Salary: {self.salary}")

class Teaching(Staff):
    def __init__(self, name, salary, subject):
        super().__init__(name, salary)
        self.subject = subject

    def display(self):
        super().display()
        print(f"Subject: {self.subject}")

class NonTeaching(Staff):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

    def display(self):
        super().display()
        print(f"Department: {self.department}")
staff_member = Staff("John", 50000)
```

```
teaching_member = Teaching("Jane", 60000, "Math")
non_teaching_member = NonTeaching("Joe", 40000, "Finance")

staff_member.display()
print()
teaching_member.display()
print()
non_teaching_member.display()
```

```
Name: John
Salary: 50000

Name: Jane
Salary: 60000
Subject: Math

Name: Joe
Salary: 40000
Department: Finance
```

Create a class called Student, having name and email as its data members andm *init*(self, name, email) and putdata(self) as bound methods. The *init* function should assign the values passed as parameters to the requisite variables. The putdata function should display the data of the student. Create another class called PhDguide having name, email, and students as its data members. Here, the students variable is the list of students under the guide. The PhDguide class should have four bound methods: *init*, putdata, add, and remove. The *init* method should initialize the variables, the putdata should show the data of the guide, include the list of students, the add method should add a student to the list of students of the guide and the remove function should remove the student (if the student exists in the list of students of that guide) from the list of students.

In [5]:
```
class Person:
    def __init__(self, name, email):
        self.name = name
```

```
        self.email = email

    def putdata(self):
        print("Name:", self.name)
        print("Email:", self.email)

class Student(Person):
    def __init__(self, name, email):
        super().__init__(name, email)

class PhDguide(Person):
    def __init__(self, name, email):
        super().__init__(name, email)
        self.students = []

    def putdata(self):
        super().putdata()
        print("Students:", self.students)

    def add(self, student):
        self.students.append(student)

    def remove(self, student):
        if student in self.students:
            self.students.remove(student)
            print(f"{student.name} has been removed from the list of students.
        else:
            print(f"{student.name} is not in the list of students.")
# Creating a Student object
student1 = Student("John Doe", "johndoe@example.com")
student1.putdata()  # Output: Name: John Doe, Email: johndoe@example.com

# Creating a PhDguide object with an empty list of students
guide1 = PhDguide("Jane Smith", "janesmith@example.com")
guide1.putdata()  # Output: Name: Jane Smith, Email: janesmith@example.com, St

# Adding the student1 to guide1's list of students
guide1.add(student1)
guide1.putdata()  # Output: Name: Jane Smith, Email: janesmith@example.com, St

# Removing the student1 from guide1's list of students
guide1.remove(student1)
guide1.putdata()  # Output: Name: Jane Smith, Email: janesmith@example.com, St
```

```
Name: John Doe
Email: johndoe@example.com
Name: Jane Smith
Email: janesmith@example.com
Students: []
Name: Jane Smith
Email: janesmith@example.com
Students: [<__main__.Student object at 0x0000026FCFB731F0>]
John Doe has been removed from the list of students.
Name: Jane Smith
Email: janesmith@example.com
Students: []
```

Write program that has a class point. Define another class location which has two objects (Location and Destination) of class point. Also define function in Location that prints reflection of Destination on the x axis.

In [8]:
```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Location(Point):
    def __init__(self, x1, y1, x2, y2):
        super().__init__(x1, y1)
        self.destination = Destination(x2, y2)

    def reflect_destination_on_x_axis(self):
        self.destination.y = -self.destination.y
        print("Reflected Destination point on X axis: ({}, {})".format(self.de
class Destination(Point):
    pass

# Example usage
l = Location(1, 2, 3, 4)
l.reflect_destination_on_x_axis() # prints (3, -4)
```

```
Reflected Destination point on X axis: (3, -4)
```

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

In [9]:
```python
class Department:
    def __init__(self, name):
        self.name = name
```

```
class Course(Department):
    def __init__(self, name, department):
        super().__init__(department)
        self.course_name = name

class Student:
    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no

class Enroll(Student, Course):
    def __init__(self, name, roll_no, course_name, department):
        Student.__init__(self, name, roll_no)
        Course.__init__(self, course_name, department)

    def get_enrolled(self):
        print(f"{self.name} with roll no. {self.roll_no} has enrolled for {sel
enrollment = Enroll("Alice", 101, "Calculus", "Math")
enrollment.get_enrolled()
```

```
Math with roll no. 101 has enrolled for Calculus in Math department.
```

Create a class student with following member attributes: roll no, name, age and total marks. Create suitable methods for reading and printing member variables. Write a python program to overload '==' operator to print the details of students having same marks.

In [10]:

```
class Student:
    def __init__(self, roll_no, name, age, total_marks):
        self.roll_no = roll_no
        self.name = name
        self.age = age
        self.total_marks = total_marks

    def display_student_info(self):
        print(f"Roll No: {self.roll_no}")
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Total Marks: {self.total_marks}")

    def __eq__(self, other):
        if isinstance(other, Student):
            return self.total_marks == other.total_marks
        return False
# create two student objects
s1 = Student(1, "John", 20, 90)
s2 = Student(2, "Mary", 21, 80)
```

```
# compare the students based on their total marks
if s1 == s2:
    print(f"{s1.name} and {s2.name} have the same marks!")
else:
    print(f"{s1.name} and {s2.name} do not have the same marks.")
```

```
John and Mary do not have the same marks.
```

Write a program to create a class called Data having “value” as its data member. Overload the (>) and the (<) operator for the class. Instantiate the class and compare the objects using *lt* and *gt*.

In [11]:
```
class Data:
    def __init__(self, value):
        self.value = value

    def __lt__(self, other):
        return self.value < other.value

    def __gt__(self, other):
        return self.value > other.value

# Testing the Data class
d1 = Data(10)
d2 = Data(20)
d3 = Data(15)

print(d1 > d2) # False
print(d2 > d3) # True
print(d3 < d1) # False
print(d3 < d2) # True
```

```
False
True
False
True
```

The following illustration creates a class called data. If no argument is passed while instantiating the class a false is returned, otherwise a true is returned.

In [15]:
```
class Data:
    def __init__(self, value=None):
        if value:
            self.value = value
```

```
            self.status = True
        else:
            self.status = False

    def __str__(self):
        return f"status: {self.status}"
data_obj1 = Data()
print(data_obj1)
# Output: status: False

data_obj2 = Data(10)
print(data_obj2)
# Output: status: True
```

```
status: False
status: True
```