Home Work -5

Contribution

Rinish : 1,2,7

Roshita : 3,4 ,8

Sai kumar reddy Kaluvakolu: 5,6,9,10

Q1 and Q2 . Create a Python program to model a simple banking system. You will have a base class BankAccount with the following methods and attributes: and Continuation to question 1, Now, create a child class SavingsAccount that inherits from BankAccount with the following additional features:

In [25]:
```
class BankAccount:
    account_counter = 1

    def __init__(self, account_holder, initial_balance=0):
        self.account_number = BankAccount.account_counter
        self.account_holder = account_holder
        self.balance = initial_balance
        BankAccount.account_counter += 1

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited ${amount}. New balance: ${self.balance}")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds. Withdrawal canceled.")
        else:
            self.balance -= amount
            print(f"Withdrawn ${amount}. New balance: ${self.balance}")

    def get_balance(self):
        return self.balance

    def display_account_info(self):
        print(f"Account Number: {self.account_number}")
        print(f"Account Holder: {self.account_holder}")
        print(f"Current Balance: ${self.balance}")

account1 = BankAccount("Rinish", 1000)
account2 = BankAccount("Sai Kumar", 500)

account1.deposit(200)
account1.withdraw(150)

account1.display_account_info()

account2.deposit(300)
```

```
account2.withdraw(700)

account2.display_account_info()

class SavingsAccount(BankAccount):
    def __init__(self, account_holder, interest_rate, initial_balance=0):
        super().__init__(account_holder, initial_balance)
        self.interest_rate = interest_rate

    def add_interest(self):
        interest_amount = self.balance * (self.interest_rate / 100)
        self.balance += interest_amount
        print(f"Interest added: ${interest_amount}. New balance: ${self.balance}")

    def display_account_info(self):
        super().display_account_info()
        print(f"Interest Rate: {self.interest_rate}%")

account = BankAccount("Rinish", 1000)

account.deposit(200)
account.withdraw(150)

account.display_account_info()

savings_account = SavingsAccount("Sai Kumar", interest_rate=3, initial_balance=500)

savings_account.deposit(300)
savings_account.withdraw(200)

savings_account.add_interest()

savings_account.display_account_info()
```

```
Deposited $200. New balance: $1200
Withdrawn $150. New balance: $1050
Account Number: 1
Account Holder: Rinish
Current Balance: $1050
Deposited $300. New balance: $800
Withdrawn $700. New balance: $100
Account Number: 2
Account Holder: Sai Kumar
Current Balance: $100
Deposited $200. New balance: $1200
Withdrawn $150. New balance: $1050
Account Number: 3
Account Holder: Rinish
Current Balance: $1050
Deposited $300. New balance: $800
Withdrawn $200. New balance: $600
Interest added: $18.0. New balance: $618.0
Account Number: 4
Account Holder: Sai Kumar
Current Balance: $618.0
Interest Rate: 3%
```

Q3. Write a Shopping Cart class to implement a shopping cart that you often find on websites where you could purchase some goods. Think about what things you could store in a cart and

also what operations you could perform on the cart. To simplify matters, you could consider the website to be an electronics e-store that has goods like flat-panel TVs, boomboxes, iPods, camcorders, and so on.

In [31]:
```
class ShoppingCart:
    def __init__(self):
        self.items = {}

    def add_item(self, product_name, quantity, price):
        if product_name in self.items:
            self.items[product_name]['quantity'] += quantity
        else:
            self.items[product_name] = {'quantity': quantity, 'price': price}

    def remove_item(self, product_name, quantity):
        if product_name in self.items:
            if quantity >= self.items[product_name]['quantity']:
                del self.items[product_name]
            else:
                self.items[product_name]['quantity'] -= quantity

    def calculate_total(self):
        total = 0
        for product_name, product_info in self.items.items():
            total += product_info['quantity'] * product_info['price']
        return total

    def display_cart(self):
        print("Shopping Cart:")
        for product_name, product_info in self.items.items():
            print(f"{product_name}: {product_info['quantity']} x ${product_info['price


cart = ShoppingCart()

cart.add_item("Fridge", 2, 500)
cart.add_item("Boombox", 1, 100)
cart.add_item("Camcorder", 1, 300)

cart.display_cart()
print(f"Total: ${cart.calculate_total()}")

cart.remove_item("Fridge", 1)

cart.display_cart()
print(f"Total: ${cart.calculate_total()}")
```

```
Shopping Cart:
Fridge: 2 x $500 each
Boombox: 1 x $100 each
Camcorder: 1 x $300 each
Total: $1400
Shopping Cart:
Fridge: 1 x $500 each
Boombox: 1 x $100 each
Camcorder: 1 x $300 each
Total: $900
```

Q4. Consider a TableFan class. What would be the attributes of this class? Examples of attributes could be speed levels of fan, side-to-side movement (on/off, and degrees of movement), manufacturer name, cost, used/new, and so on. Think about it this way. You want to be able to get information about the fan; for example, if you want to buy or sell a fan from some website such as Craigslist, what features would you be interested in? Also, consider control (operations) you might want to have over the fan such as setting the fan speed or having it pan from side to side. Write python program demonstrating all these features.

In [30]:
```
class TableFan:
    def __init__(self, maker, model, price, status='new'):
        self.maker = maker
        self.model = model
        self.price = price
        self.status = status
        self.speed = 0
        self.rotation_enabled = False

    def set_speed(self, speed):
        if 0 <= speed <= 3:
            self.speed = speed
            print(f"Fan speed set to {speed}")
        else:
            print("Invalid speed level. Please choose a speed between 0 and 3.")

    def toggle_rotation(self):
        self.rotation_enabled = not self.rotation_enabled
        if self.rotation_enabled:
            print("Fan rotation is ON.")
        else:
            print("Fan rotation is OFF.")

    def get_info(self):
        return f"Maker: {self.maker}, Model: {self.model}, Price: ${self.price}, Statu


fan = TableFan(maker="ABC Fans", model="Model X", price=50, status='new')

print(fan.get_info())

fan.set_speed(2)
fan.toggle_rotation()

print(fan.get_info())
```

```
Maker: ABC Fans, Model: Model X, Price: $50, Status: new, Speed: 0, Rotation: OFF
Fan speed set to 2
Fan rotation is ON.
Maker: ABC Fans, Model: Model X, Price: $50, Status: new, Speed: 2, Rotation: ON
```

Q5. Design a Python program that represents an animal hierarchy. Create a base class Animal with the following attributes and methods: Animal Class: Attributes: name: Name of the animal. species: Species of the animal. Create a child class Bird that inherits from Animal. The Bird class should include: Bird Class (Inherits from Animal): Additional Attributes: wing_span: Wing span of the bird (in meters). beak_length: Length of the bird's beak (in centimeters). Additional Methods: method that simulates the bird flying. Create another child class Mammal that inherits from

Animal. The Mammal class should include: Mammal Class (Inherits from Animal): Additional Attributes: fur_color: Color of the mammal's fur. leg_count: Number of legs the mammal has. Additional Methods: run(): A method that simulates the mammal running. Demonstrate how you can create instances of Bird and Mammal objects, set their attributes, and call their methods. This example uses class inheritance, with Bird and Mammal inheriting attributes and methods from the base Animal class.

In [6]:
```
class Animal:
    def __init__(self, name, species):
        self.name = name
        self.species = species


class Bird(Animal):
    def __init__(self, name, species, wing_span, beak_length):
        Animal.__init__(self, name, species)
        self.wing_span = wing_span
        self.beak_length = beak_length

    def fly(self):
        print(f"{self.name}, is a {self.species}, it can fly.")
        print(f"Wing Span: {self.wing_span} meters")
        print(f"Beak Length: {self.beak_length} centimeters")


class Mammal(Animal):
    def __init__(self, name, species, fur_color, leg_count):
        Animal.__init__(self, name, species)
        self.fur_color = fur_color
        self.leg_count = leg_count

    def run(self):
        print(f"{self.name},is a  {self.species}, it can run.")
        print(f"Fur Color: {self.fur_color}")
        print(f"Leg Count: {self.leg_count}")


eagle = Bird(name="Sparrow", species="Bird", wing_span=0.25, beak_length=2)
eagle.fly()

cheetah = Mammal(name="Monkey", species="Mammal", fur_color="Black", leg_count=2)
cheetah.run()
```

```
Sparrow, is a Bird, it can fly.
Wing Span: 0.25 meters
Beak Length: 2 centimeters
Monkey,is a  Mammal, it can run.
Fur Color: Black
Leg Count: 2
```

Q6. Design a Python program to simulate the Olympic Games. Create classes to represent the following entities:

In [10]:
```
class Athlete:
    def __init__(self, name, country, age, sports=None):
        self.name = name
        self.country = country
```

```
        self.age = age
        self.sports = sports or []

    def __str__(self):
        return f"Athlete: {self.name}, Country: {self.country}, Age: {self.age}, Sport


class Sport:
    def __init__(self, name, rules):
        self.name = name
        self.rules = rules

    def __str__(self):
        return f"Sport: {self.name}, Rules: {self.rules}"


class Event:
    def __init__(self, name, date, time, location, sport):
        self.name = name
        self.date = date
        self.time = time
        self.location = location
        self.sport = sport

    def __str__(self):
        return f"Event: {self.name}, Date: {self.date}, Time: {self.time}, Location: {


class Medal:
    def __init__(self, medal_type, event, athlete):
        self.medal_type = medal_type
        self.event = event
        self.athlete = athlete

    def __str__(self):
        return f"Medal: {self.medal_type} in {self.event.name} for {self.athlete.name}


class Olympics:
    def __init__(self):
        self.athletes = []
        self.sports = []
        self.events = []
        self.medals = []

    def register_athlete(self, athlete):
        self.athletes.append(athlete)

    def add_sport(self, sport):
        self.sports.append(sport)

    def schedule_event(self, event):
        self.events.append(event)

    def record_medal_winner(self, medal_type, event_name, athlete_name):
        event = next((e for e in self.events if e.name == event_name), None)
        athlete = next((a for a in self.athletes if a.name == athlete_name), None)

        if event and athlete:
            medal = Medal(medal_type, event, athlete)
```

```
            self.medals.append(medal)
            print(f"{athlete.name} won a {medal.medal_type} medal in {event.name}!")

    def display_athletes(self):
        for athlete in self.athletes:
            print(athlete)

    def display_sports(self):
        for sport in self.sports:
            print(sport)

    def display_events(self):
        for event in self.events:
            print(event)

    def display_medals(self):
        for medal in self.medals:
            print(medal)


olympics = Olympics()

athlete1 = Athlete(name="Rinish", country="India", age=21, sports=["Swimming", "Athlet
athlete2 = Athlete(name="Mike lebron", country="USA", age=28, sports=["Gymnastics", "D
olympics.register_athlete(athlete1)
olympics.register_athlete(athlete2)

swimming = Sport(name="Swimming", rules="Compete by swimming in a pool.")
athletics = Sport(name="Athletics", rules="Track and field events.")
gymnastics = Sport(name="Gymnastics", rules="Artistic and rhythmic gymnastics.")
diving = Sport(name="Diving", rules="Perform acrobatic dives into water.")
olympics.add_sport(swimming)
olympics.add_sport(athletics)
olympics.add_sport(gymnastics)
olympics.add_sport(diving)


event1 = Event(name="100m Freestyle", date="2023-08-01", time="10:00 AM", location="Ac
event2 = Event(name="High Jump", date="2023-08-02", time="3:00 PM", location="Athletic
event3 = Event(name="Artistic Gymnastics Final", date="2023-08-03", time="6:00 PM", lo
event4 = Event(name="Platform Diving", date="2023-08-04", time="1:00 PM", location="Di
olympics.schedule_event(event1)
olympics.schedule_event(event2)
olympics.schedule_event(event3)
olympics.schedule_event(event4)

olympics.record_medal_winner("Gold", "100m Freestyle", "Rinish")
olympics.record_medal_winner("Silver", "High Jump", "Mike lebron")
olympics.record_medal_winner("Bronze", "Artistic Gymnastics Final", "Rinish")
olympics.record_medal_winner("Gold", "Platform Diving", "Mike lebron")


print("\nAthletes:")
olympics.display_athletes()

print("\nSports:")
olympics.display_sports()

print("\nEvents:")
olympics.display_events()
```

```
print("\nMedals:")
olympics.display_medals()
```

```
Rinish won a Gold medal in 100m Freestyle!
Mike lebron won a Silver medal in High Jump!
Rinish won a Bronze medal in Artistic Gymnastics Final!
Mike lebron won a Gold medal in Platform Diving!

Athletes:
Athlete: Rinish, Country: India, Age: 21, Sports: Swimming, Athletics
Athlete: Mike lebron, Country: USA, Age: 28, Sports: Gymnastics, Diving

Sports:
Sport: Swimming, Rules: Compete by swimming in a pool.
Sport: Athletics, Rules: Track and field events.
Sport: Gymnastics, Rules: Artistic and rhythmic gymnastics.
Sport: Diving, Rules: Perform acrobatic dives into water.

Events:
Event: 100m Freestyle, Date: 2023-08-01, Time: 10:00 AM, Location: Aquatics Center, S
port: Swimming
Event: High Jump, Date: 2023-08-02, Time: 3:00 PM, Location: Athletics Stadium, Spor
t: Athletics
Event: Artistic Gymnastics Final, Date: 2023-08-03, Time: 6:00 PM, Location: Gymnasti
cs Hall, Sport: Gymnastics
Event: Platform Diving, Date: 2023-08-04, Time: 1:00 PM, Location: Diving Pool, Spor
t: Diving

Medals:
Medal: Gold in 100m Freestyle for Rinish from India
Medal: Silver in High Jump for Mike lebron from USA
Medal: Bronze in Artistic Gymnastics Final for Rinish from India
Medal: Gold in Platform Diving for Mike lebron from USA
```

Q7. Design a set of Python classes to model a social media platform, which includes classes for User, Post, and Comment. Each class should have specific attributes and methods to facilitate interactions and content creation

In [39]:

```
class User:
    def __init__(self, username, email):
        self.username = username
        self.email = email

    def create_post(self, content):
        new_post = Post(content, self)
        return new_post

    def create_comment(self, post, text):
        new_comment = Comment(text, self, post)
        return new_comment


class Post:
    def __init__(self, content, user):
        self.content = content
        self.user = user
        self.comments = []
```

```
    def edit_post(self, new_content):
        self.content = new_content

    def delete_post(self):
        print("Post deleted.")


class Comment:
    def __init__(self, text, user, post):
        self.text = text
        self.user = user
        self.post = post

    def edit_comment(self, new_text):
        self.text = new_text

    def delete_comment(self):
        print("Comment deleted.")

user1 = User(username="Rinish", email="rinish@example.com")
user2 = User(username="Roshita", email="roshita@example.com")

post1 = user1.create_post("This is my first post.")
comment1 = user2.create_comment(post1, "Great post!")

print(f"User: {user1.username}, Email: {user1.email}")
print(f"Post Content: {post1.content}")
print(f"Comment by {comment1.user.username}: {comment1.text}")
```

```
User: Rinish, Email: rinish@example.com
Post Content: This is my first post.
Comment by Roshita: Great post!
```

Q8. Create a Python program for a school management system with the following classes: Person: This class should have attributes like name, age, and gender. It should also have a method display_info to print the person's information. Address: This class should have attributes like street, city, and zip code. It should have a method to print the address details. Student: This class should inherit from Person and Address. It should have additional attributes like student ID and grade. Implement a method to print the student's information and address. Teacher: This class should also inherit from Person and Address. It should have attributes like employee ID and subject taught. Implement a method to print the teacher's information and address. Design these classes with multiple inheritance, and create instances of the Student and Teacher classes to demonstrate their functionalities.

In [35]:
```
class Person:
    def __init__(self, full_name, age, gender):
        self.full_name = full_name
        self.age = age
        self.gender = gender

    def display_info(self):
        print(f"Full Name: {self.full_name}, Age: {self.age}, Gender: {self.gender}")


class Address:
    def __init__(self, street, city, zip_code):
```

```
        self.street = street
        self.city = city
        self.zip_code = zip_code

    def print_address(self):
        print(f"Street: {self.street}, City: {self.city}, Zip Code: {self.zip_code}")


class Student(Person, Address):
    def __init__(self, full_name, age, gender, street, city, zip_code, student_id, gra
        Person.__init__(self, full_name, age, gender)
        Address.__init__(self, street, city, zip_code)
        self.student_id = student_id
        self.grade = grade

    def print_student_info(self):
        self.display_info()
        self.print_address()
        print(f"Student ID: {self.student_id}, Grade: {self.grade}")


class Teacher(Person, Address):
    def __init__(self, full_name, age, gender, street, city, zip_code, employee_id, su
        Person.__init__(self, full_name, age, gender)
        Address.__init__(self, street, city, zip_code)
        self.employee_id = employee_id
        self.subject_taught = subject_taught

    def print_teacher_info(self):
        self.display_info()
        self.print_address()
        print(f"Employee ID: {self.employee_id}, Subject Taught: {self.subject_taught}


student1 = Student("Roshitha", 21, "Female", "123 Main St", "Saint Louis", "12345", "S
teacher1 = Teacher("Max", 35, "Male", "456 Oak St", "Townsville", "67890", "T567", "Ma

print("Student Information:")
student1.print_student_info()

print("\nTeacher Information:")
teacher1.print_teacher_info()
```

```
Student Information:
Full Name: Roshitha, Age: 21, Gender: Female
Street: 123 Main St, City: Saint Louis, Zip Code: 12345
Student ID: S123, Grade: 10th

Teacher Information:
Full Name: Max, Age: 35, Gender: Male
Street: 456 Oak St, City: Townsville, Zip Code: 67890
Employee ID: T567, Subject Taught: Mathematics
```

Q9. Design a Python program to simulate a library. In this library, you should create two classes: Book and Shelf. Each Shelf can contain multiple Book objects. Here are the details for each class:

In [14]:

```
class Book:
    def __init__(self, title, author, publication_year):
        self.title = title
```

```
        self.author = author
        self.publication_year = publication_year

    def display_info(self):
        return f"Book: {self.title} by {self.author} ({self.publication_year})"


class Shelf:
    def __init__(self, shelf_name):
        self.shelf_name = shelf_name
        self.books = []

    def add_book(self, book):
        self.books.append(book)
        print(f"{book.title} added to the shelf {self.shelf_name}.")

    def remove_book(self, book_title):
        for book in self.books:
            if book.title == book_title:
                self.books.remove(book)
                print(f"{book.title} removed from the shelf {self.shelf_name}.")
                break
        else:
            print(f"Book with title {book_title} not found on the shelf {self.shelf_na

    def display_info(self):
        return f"Shelf: {self.shelf_name}, Books: {len(self.books)}"


shelf1 = Shelf(shelf_name="Fiction")
shelf2 = Shelf(shelf_name="Study")

book1 = Book(title="Harry potter", author="jk rowling", publication_year=2000)
book2 = Book(title="Maxwell", author="cricket", publication_year=2015)
book3 = Book(title="bradpit", author="james", publication_year=2045)

shelf1.add_book(book1)
shelf1.add_book(book2)
shelf2.add_book(book3)

print(shelf1.display_info())
print(shelf2.display_info())

shelf1.remove_book("Maxwell")

print(shelf1.display_info())
```

```
Harry potter added to the shelf Fiction.
Maxwell added to the shelf Fiction.
bradpit added to the shelf Study.
Shelf: Fiction, Books: 2
Shelf: Study, Books: 1
Maxwell removed from the shelf Fiction.
Shelf: Fiction, Books: 1
```

Q10. Consider a hospital scenario. Design classes for:

In [37]:

```
class Patient:
    def __init__(self, patient_id, name, gender, age, address, phone_number, dob, heig
        self.patient_id = patient_id
```

```
        self.name = name
        self.gender = gender
        self.age = age
        self.address = address
        self.phone_number = phone_number
        self.dob = dob
        self.height = height
        self.weight = weight

    def get_patient_info(self):
        return {
            "Patient ID": self.patient_id,
            "Name": self.name,
            "Gender": self.gender,
            "Age": self.age,
            "Address": self.address,
            "Phone Number": self.phone_number,
            "Date of Birth": self.dob,
            "Height": self.height,
            "Weight": self.weight
        }

    def set_age(self, age):
        self.age = age

    def set_address(self, address):
        self.address = address

    def set_phone_number(self, phone_number):
        self.phone_number = phone_number


class Doctor:
    def __init__(self, registration_number, name, qualification, specialization, phone
        self.registration_number = registration_number
        self.name = name
        self.qualification = qualification
        self.specialization = specialization
        self.phone_number = phone_number
        self.office_hours = office_hours
        self.office_location = office_location

    def get_doctor_info(self):
        return {
            "Registration Number": self.registration_number,
            "Name": self.name,
            "Qualification": self.qualification,
            "Specialization": self.specialization,
            "Phone Number": self.phone_number,
            "Office Hours": self.office_hours,
            "Office Location": self.office_location
        }

    def set_office_hours(self, office_hours):
        self.office_hours = office_hours

    def set_office_location(self, office_location):
        self.office_location = office_location

    def set_phone_number(self, phone_number):
```

```
        self.phone_number = phone_number


class PatientRecord:
    def __init__(self, last_checkup_date, doctor_id, patient_id, health_problems, pres
        self.last_checkup_date = last_checkup_date
        self.doctor_id = doctor_id
        self.patient_id = patient_id
        self.health_problems = health_problems
        self.prescribed_medicines = prescribed_medicines
        self.checkup_cost = checkup_cost
        self.final_report = final_report

    def get_record_info(self):
        return {
            "Last Checkup Date": self.last_checkup_date,
            "Doctor ID": self.doctor_id,
            "Patient ID": self.patient_id,
            "Health Problems": self.health_problems,
            "Prescribed Medicines": self.prescribed_medicines,
            "Checkup Cost": self.checkup_cost,
            "Final Report": self.final_report
        }

    def set_health_problems(self, health_problems):
        self.health_problems = health_problems

    def set_prescribed_medicines(self, prescribed_medicines):
        self.prescribed_medicines = prescribed_medicines

    def set_checkup_cost(self, checkup_cost):
        self.checkup_cost = checkup_cost



patient1 = Patient(patient_id=1, name="Rinish", gender="Male", age=21, address="afdfgv
doctor1 = Doctor(registration_number="MLN", name="Roshita", qualification="MD", specia
record1 = PatientRecord(last_checkup_date="2023-10-01", doctor_id="MLN", patient_id=1,

print(patient1.get_patient_info())
print(doctor1.get_doctor_info())
print(record1.get_record_info())

patient1.set_age(31)
doctor1.set_office_hours("10 AM - 6 PM")
record1.set_checkup_cost(200)

print(patient1.get_patient_info())
print(doctor1.get_doctor_info())
print(record1.get_record_info())
```

```
{'Patient ID': 1, 'Name': 'Rinish', 'Gender': 'Male', 'Age': 21, 'Address': 'afdfgvya
ugsdyug', 'Phone Number': '1234567', 'Date of Birth': '1992-01-01', 'Height': 175, 'W
eight': 70}
{'Registration Number': 'MLN', 'Name': 'Roshita', 'Qualification': 'MD', 'Specializat
ion': 'Dentist', 'Phone Number': '1234567', 'Office Hours': '9 AM - 5 PM', 'Office Lo
cation': 'Dentist, Hospital XYZ'}
{'Last Checkup Date': '2023-10-01', 'Doctor ID': 'MLN', 'Patient ID': 1, 'Health Prob
lems': ['High Blood Pressure'], 'Prescribed Medicines': ['Aspirin'], 'Checkup Cost':
150, 'Final Report': 'Patient is advised to monitor blood pressure regularly.'}
{'Patient ID': 1, 'Name': 'Rinish', 'Gender': 'Male', 'Age': 31, 'Address': 'afdfgvya
ugsdyug', 'Phone Number': '1234567', 'Date of Birth': '1992-01-01', 'Height': 175, 'W
eight': 70}
{'Registration Number': 'MLN', 'Name': 'Roshita', 'Qualification': 'MD', 'Specializat
ion': 'Dentist', 'Phone Number': '1234567', 'Office Hours': '10 AM - 6 PM', 'Office L
ocation': 'Dentist, Hospital XYZ'}
{'Last Checkup Date': '2023-10-01', 'Doctor ID': 'MLN', 'Patient ID': 1, 'Health Prob
lems': ['High Blood Pressure'], 'Prescribed Medicines': ['Aspirin'], 'Checkup Cost':
200, 'Final Report': 'Patient is advised to monitor blood pressure regularly.'}
```

In [ ]:

In [ ]: