```
#Q1. Write a Python function that takes a list of temperatures in
Fahrenheit as input and outputs a new list with the temperatures
converted to Celsius?

def fahrenheit_to_celsius(fahrenheit_temps):
    celsius_temps = []
    for temp in fahrenheit_temps:
        celsius = (temp - 32) * 5/9
        celsius_temps.append(celsius)
    return celsius_temps


fahrenheit_temperatures = [32, 68, 86, 104]
celsius_temperatures = fahrenheit_to_celsius(fahrenheit_temperatures)
print(celsius_temperatures)
```

```
[0.0, 20.0, 30.0, 40.0]
```

```
#Q2. Write a Python function that takes a list of book titles and
their respective authors as input and outputs a new dictionary with
the authorsas keys and the book titles as values?

def create_author_book_dictionary(book_titles_and_authors):
    author_book_dict = {}
    for title, author in book_titles_and_authors:
        if author in author_book_dict:
            author_book_dict[author].append(title)
        else:
            author_book_dict[author] = [title]
    return author_book_dict


books_and_authors = [("Book1", "Author1"), ("Book2", "Author2"),
("Book3", "Author1"), ("Book4", "Author3")]
author_book_dictionary =
create_author_book_dictionary(books_and_authors)
print(author_book_dictionary)
```

```
{'Author1': ['Book1', 'Book3'], 'Author2': ['Book2'], 'Author3':
['Book4']}
```

```
#Q3. Write a Python function that takes a list of grocery items and
their respective prices as input and outputs the total cost of the
grocery bill?


def calculate_grocery_bill(grocery_items_and_prices):
    total_cost = 0
    for item, price in grocery_items_and_prices:
        total_cost += price
    return total_cost
```

```
grocery_items_and_prices = [("Apple", 1.25), ("Banana", 0.75),
("Milk", 2.50), ("Bread", 1.80)]
total_cost = calculate_grocery_bill(grocery_items_and_prices)
print("Total grocery bill: $", total_cost)
```

```
Total grocery bill: $ 6.3
```

```
#Q4. Write a Python function that takes a list of movie titles and
thei respective ratings as input and outputs a new list with the
movies sorted indescending order based on their ratings?

def sort_movies_by_rating(movies_and_ratings):
    sorted_movies = sorted(movies_and_ratings, key=lambda x: x[1],
reverse=True)
    return sorted_movies


movies_and_ratings = [("Movie1", 8.5), ("Movie2", 7.9), ("Movie3",
9.2), ("Movie4", 6.8)]
sorted_movies = sort_movies_by_rating(movies_and_ratings)
print(sorted_movies)
```

```
[('Movie3', 9.2), ('Movie1', 8.5), ('Movie2', 7.9), ('Movie4', 6.8)]
```

```
#Q5. Write a Python function that takes a list of student names and
their respective grades as input and outputs a new dictionary with the
studentnames as keys and their average grade as values?

def calculate_average_grades(student_names_and_grades):
    average_grades = {}
    for name, grades in student_names_and_grades.items():
        average_grade = sum(grades) / len(grades)
        average_grades[name] = average_grade
    return average_grades


student_names_and_grades = {
    "Alice": [90, 85, 88],
    "Bob": [75, 80, 82],
    "Charlie": [95, 92, 98]
}
average_grades = calculate_average_grades(student_names_and_grades)
print(average_grades)
```

```
{'Alice': 87.66666666666667, 'Bob': 79.0, 'Charlie': 95.0}
```

```
# Q6. Inventory Management: Create a class called "Inventory" that
hasattributes such as item name, quantity, price, etc. You can then
createobjects of this class for each item in your inventory and use
```

```
its methods toupdate the inventory as items are sold or restocked?


class Inventory:
    def __init__(self, item_name, quantity, price):
        self.item_name = item_name
        self.quantity = quantity
        self.price = price

    def sell_item(self, quantity_sold):
        if quantity_sold <= self.quantity:
            self.quantity -= quantity_sold
            print(f"{quantity_sold} {self.item_name}(s) sold.")
        else:
            print("Not enough stock available.")

    def restock_item(self, quantity_restocked):
        self.quantity += quantity_restocked
        print(f"{quantity_restocked} {self.item_name}(s) restocked.")

    def update_price(self, new_price):
        self.price = new_price
        print(f"Price updated to ${new_price}.")

    def display_inventory(self):
        print(f"Item: {self.item_name}")
        print(f"Quantity: {self.quantity}")
        print(f"Price: ${self.price}")


# Example usage:
item1 = Inventory("Apple", 10, 1.50)
item2 = Inventory("Banana", 20, 0.75)

item1.display_inventory()
print("---")

item1.sell_item(5)
item1.display_inventory()
print("---")

item1.restock_item(7)
item1.display_inventory()
print("---")

item1.update_price(1.75)
item1.display_inventory()

Item: Apple
Quantity: 10
Price: $1.5
```

```
---
5 Apple(s) sold.
Item: Apple
Quantity: 5
Price: $1.5
---
7 Apple(s) restocked.
Item: Apple
Quantity: 12
Price: $1.5
---
Price updated to $1.75.
Item: Apple
Quantity: 12
Price: $1.75
```

```python
#Q7. Employee Management: Create a class called "Employee" that hasattributes such as name, age, salary, etc. You can then create objects ofthis class for each employee in your organization and use its methods tmanage employee data, such as updating salaries or tracking employee


class Employee:
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
        self.attendance = 0

    def update_salary(self, new_salary):
        self.salary = new_salary
        print(f"Salary updated to ${new_salary} for {self.name}.")

    def mark_attendance(self):
        self.attendance += 1
        print(f"{self.name} marked attendance.")

    def display_employee_details(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Salary: ${self.salary}")
        print(f"Attendance: {self.attendance} days")



employee1 = Employee("Alice", 30, 50000)
employee2 = Employee("Bob", 25, 45000)

employee1.display_employee_details()
```

```
print("---")

employee1.update_salary(52000)
employee1.mark_attendance()
employee1.mark_attendance()
employee1.display_employee_details()

Name: Alice
Age: 30
Salary: $50000
Attendance: 0 days
---
Salary updated to $52000 for Alice.
Alice marked attendance.
Alice marked attendance.
Name: Alice
Age: 30
Salary: $52000
Attendance: 2 days

# Q8. Banking: Create a class called "Account" that has attributes
such asaccount number, balance, and interest rate. You can then create
objects othis class for each customer's account and use its methods to
handle transactions, such as deposits, withdrawals, and interest
calculations?


class Account:
    def __init__(self, account_number, balance, interest_rate):
        self.account_number = account_number
        self.balance = balance
        self.interest_rate = interest_rate

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited ${amount} to account {self.account_number}.
Current balance: ${self.balance}")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew ${amount} from account
{self.account_number}. Current balance: ${self.balance}")
        else:
            print("Insufficient funds.")

    def calculate_interest(self):
        interest_amount = (self.balance * self.interest_rate) / 100
        self.balance += interest_amount
        print(f"Interest calculated and added to account
```

```
{self.account_number}. Current balance: ${self.balance}")

    def display_account_details(self):
        print(f"Account Number: {self.account_number}")
        print(f"Balance: ${self.balance}")
        print(f"Interest Rate: {self.interest_rate}%")



account1 = Account("123456", 1000, 5)
account2 = Account("789012", 2000, 4.5)

account1.display_account_details()
print("---")

account1.deposit(500)
account1.withdraw(200)
account1.calculate_interest()
account1.display_account_details()
```

```
Account Number: 123456
Balance: $1000
Interest Rate: 5%
---
Deposited $500 to account 123456. Current balance: $1500
Withdrew $200 from account 123456. Current balance: $1300
Interest calculated and added to account 123456. Current balance:
$1365.0
Account Number: 123456
Balance: $1365.0
Interest Rate: 5%
```

```
#Q9. Medical Records Management: Create a class called "Patient" that
hasattributes such as name, age, medical history, etc. You can then
create of this class for each patient and use its methods to manage
patientdata, such as scheduling appointments or updating medical
records?


class Patient:
    def __init__(self, name, age, medical_history=None):
        self.name = name
        self.age = age
        if medical_history is None:
            medical_history = []
        self.medical_history = medical_history

    def schedule_appointment(self, appointment_date):
        print(f"Appointment scheduled for {self.name} on
{appointment_date}.")
```

```
    def update_medical_history(self, new_entry):
        self.medical_history.append(new_entry)
        print("Medical history updated.")

    def display_patient_details(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print("Medical History:")
        for entry in self.medical_history:
            print("- ", entry)



patient1 = Patient("Alice", 35, ["Allergy to penicillin"])
patient2 = Patient("Bob", 45)

patient1.display_patient_details()
print("---")

patient1.schedule_appointment("2024-06-15")
patient1.update_medical_history("High blood pressure")
patient1.display_patient_details()
```

```
Name: Alice
Age: 35
Medical History:
-  Allergy to penicillin
---
Appointment scheduled for Alice on 2024-06-15.
Medical history updated.
Name: Alice
Age: 35
Medical History:
-  Allergy to penicillin
-  High blood pressure
```

```
#Q10. Online Ordering: Create a class called "Order" that has
attributessuch as customer name, order details, total amount, etc. You
can then objects of this class for each order placed on your online
store anuse its methods to process the order, such as calculating the
total amount,generating a receipt, and updating inventory levels?

class Order:
    def __init__(self, customer_name, order_details):
        self.customer_name = customer_name
        self.order_details = order_details
        self.total_amount = self.calculate_total_amount()

    def calculate_total_amount(self):
        total = 0
```

```
        for item, price in self.order_details.items():
            total += price
        return total

    def generate_receipt(self):
        print("Receipt:")
        print("Customer Name:", self.customer_name)
        print("Order Details:")
        for item, price in self.order_details.items():
            print(f"- {item}: ${price}")
        print("Total Amount:", self.total_amount)

    def update_inventory(self, inventory):
        for item, quantity in self.order_details.items():
            if item in inventory:
                inventory[item] -= quantity
        print("Inventory updated.")

# Example usage:
order_details = {"Apple": 1.50, "Banana": 0.75, "Milk": 2.50, "Bread":
1.80}
order1 = Order("Alice", order_details)

order1.generate_receipt()
print("---")

inventory = {"Apple": 20, "Banana": 30, "Milk": 15, "Bread": 25}
order1.update_inventory(inventory)
print(inventory)
```

```
Receipt:
Customer Name: Alice
Order Details:
- Apple: $1.5
- Banana: $0.75
- Milk: $2.5
- Bread: $1.8
Total Amount: 6.55
---
Inventory updated.
{'Apple': 18.5, 'Banana': 29.25, 'Milk': 12.5, 'Bread': 23.2}
```