Task - 1

Develop a flight booking system that allows users to search for flights based on departure and destination cities, departure date, etc. Use lists and tuples to store flight information such as flight number, departure time, and available seats. Implement decision statements and loops to handle user input, display available flights, and manage booking transactions. Utilize functions for different stages of the booking process, such as searching for flights, reserving seats, and generating booking confirmations.

In [ ]:
```python
# ------------------------- TASK - 1 ------------------------- #
"""Develop a flight booking system that allows users to search for flights based on
departure and destination cities, departure date, etc. Use lists and tuples to stor
information such as flight number, departure time, and available seats. Implement
decision statements and loops to handle user input, display available flights, and
manage booking transactions. Utilize functions for different stages of the booking
process, such as searching for flights, reserving seats, and generating booking
confirmations"""

import datetime

class FlightBookingSystem:
    def __init__(self,sys_name, sys_date,sys_time):
        self.sys_name = sys_name
        self.sys_date = sys_date
        self.sys_time = sys_time
        self.total_flights = 0
        self.flights_data = []

    def registerFlight(self):
        flight_num = int(input("Enter a flight number to register: "))
        dept_time = str(input("Enter a departure time: "))
        avail_seats = int(input("Enter available seats: "))
        self.flights_data.append([Flight(flight_num,dept_time,avail_seats)])
        self.total_flights += 1

    def displayFlights(self):
        numofflights = self.total_flights
        for i in range(numofflights):
            print(f"The available flights are:\nFlight number: {self.flights_data[i


class Flight(FlightBookingSystem):
    def __init__(self,flight_number,departure_time,available_seats):
        self.flight_number = flight_number
        self.departure_time = departure_time
        self.available_seats = available_seats
        self.passengers_count = 0
        self.passengers_data = []

class Passenger(FlightBookingSystem):
    def __init__(self,passenger_name,passenger_id,passenger_number,flightsys):
```

```
        self.passenger_name = passenger_name
        self.passenger_id = passenger_id
        self.passenger_number = passenger_number
        self.flightsys = flightsys

    def searchFlights(self):
        numofflights = self.flightsys.total_flights
        print(f"------------------- FLIGHTS AVAILABLE ------------------- ")
        for i in range(numofflights):
            print(f"\nFlight number: {self.flightsys.flights_data[i][0].flight_numb

    def reserveSeat(self):
        flight_num = eval(input("Choose flight number to book: "))
        for i in range(self.flightsys.total_flights):
            if self.flightsys.flights_data[i][0].flight_number == flight_num:
                self.flightsys.flights_data[i][0].passengers_data.append([self.pass
                self.flightsys.flights_data[i][0].passengers_count += 1
                print(f"FLIGHT {flight_num} BOOKED SUCCESSFULLY!")



system = FlightBookingSystem("PIA",datetime.date,datetime.time)
user_1 = Passenger("Ali",20,923000,system)
user_2 = Passenger("Zaid",15,9212000,system)

# Registering flights on the system.
system.registerFlight()
system.registerFlight()


# User_1 searching for flight and reserving it.
user_1.searchFlights()
user_1.reserveSeat()

# User_2 searching for flight and reserving it.
user_2.searchFlights()
user_2.reserveSeat()
```

Task - 2

Write a function named analyze_sentence that takes a sentence as input and returns a dictionary containing the following information: the total number of words, the total number of characters (excluding spaces), the average word length, and a list of unique words.

In [ ]:

```
def analyzeSentence(sentence):
    """Function to find the total num of words,total num of chars(no spaces),avg wo

    var splitted_list(List): splits the sentence by any empty space found. Creates
    var words(List): Stores all the words in the sentence, including duplicates.
    var unique_words(List): Stores all the words excluding the duplicates.
    var numchars(int): Stores the number of chars found in the whole sentence.
    var avg_words_len: Stores the length of each word, including duplicates.
    """
```

```
    splitted_list = sentence.split(" ")
    numwords = []
    words = []
    unique_words = []
    numchars = 0
    avg_words_len = []
    print("Splitted List:",splitted_list)
    for elem in splitted_list:
        curelem = elem.strip() # Strips the current element.
        avg_words_len.append(len(curelem)) # Adds the current element's length to t
        if curelem.isalpha(): # Checks if the current element is an alphabet or all
            words.append(curelem)
            if curelem not in unique_words: # If the current element is unique.
                unique_words.append(elem)

    # For counting the total num of characters.
    for word in words:
        for chars in word:
            numchars += 1

    print(f"Num of Words:{len(words)}\nNum of chars:{numchars}\nUnique Words:{uniqu



analyzeSentence("Coding Coding   with Syed Muhammad Zaid ")
```

```
Splitted List: ['Coding', 'Coding', '', '', 'with', 'Syed', 'Muhammad', 'Zaid', '']
Num of Words:6
Num of chars:32
Unique Words:['Coding', 'with', 'Syed', 'Muhammad', 'Zaid']
3.5555555555555554
```

Task - 3

Create a class called Employee with private attributes __name, __id, and __salary. Implement methods to set and get these attributes. Then, create another class Manager which inherits from Employee. Override the salary method to calculate the manager's salary with an additional bonus. Test these classes by creating instances of both Employee and Manager and demonstrating encapsulation and polymorphism principles.

In [ ]:

```
class Employee:
    def __init__(self, name, emp_id, salary):
        self.__name = name
        self.__id = emp_id
        self.__salary = salary

    def __get_name__(self):
        return self.__name

    def __get_id__(self):
        return self.__id

    def __get_salary__(self):
```

```
        return self.__salary

    def __set_name__(self, name):
        self.__name = name

    def __set_id__(self, emp_id):
        self.__id = emp_id

    def __set_salary__(self, salary):
        self.__salary = salary


class Manager(Employee):
    def __init__(self, name, emp_id, salary, bonus):
        super().__init__(name, emp_id, salary)
        self.__bonus = bonus

    # Overriding the salary method
    def __get_salary__(self):
        base_salary = super().__get_salary__()
        return base_salary + self.__bonus



employee_1 = Employee("Zaid", 1001, 50000)
employee_2 = Employee("Ali", 1001, 50000)
manager_1 = Manager("Zaid", 2001, 70000, 10000)

# encapsulation
print("Employee Name:", employee_1.__get_name__())
print("Employee ID:", employee_1.__get_id__())
print("Employee Salary:", employee_1.__get_salary__())

# encapsulation
print("Employee Name:", employee_2.__get_name__())
print("Employee ID:", employee_2.__get_id__())
print("Employee Salary:", employee_2.__get_salary__())

# polymorphism
print("\nManager Name:", manager_1.__get_name__())
print("Manager ID:", manager_1.__get_id__())
print("Manager Salary:", manager_1.__get_salary__())
```

```
Employee Name: Zaid
Employee ID: 1001
Employee Salary: 50000
Employee Name: Ali
Employee ID: 1001
Employee Salary: 50000

Manager Name: Zaid
Manager ID: 2001
Manager Salary: 80000
```

Task - 4

WAP for an inventory management system for a retail store. Utilize classes and inheritance to represent different types of products (e.g., electronics, clothing, groceries) with attributes such as name, price, quantity, etc. Implement methods to add products to the inventory, update quantities, and calculate the total value of the inventory. Ensure encapsulation by using private attributes where appropriate.

In [ ]:
```
class Product:
    def __init__(self, name, price, quantity):
        self.__name = name
        self.__price = price
        self.__quantity = quantity

    def get_name(self):
        return self.__name

    def get_price(self):
        return self.__price

    def get_quantity(self):
        return self.__quantity

    def set_price(self, price):
        self.__price = price

    def set_quantity(self, quantity):
        self.__quantity = quantity


class Electronics(Product):
    def __init__(self, name, price, quantity, brand):
        super().__init__(name, price, quantity)
        self.__brand = brand

    def get_brand(self):
        return self.__brand


class Clothing(Product):
    def __init__(self, name, price, quantity, size):
        self.__size = size
        super().__init__(name, price, quantity)

    def get_size(self):
        return self.__size


class Groceries(Product):
    def __init__(self, name, price, quantity, expiration_date):
        super().__init__(name, price, quantity)
        self.__expiration_date = expiration_date

    def get_expiration_date(self):
        return self.__expiration_date
```

```
class Inventory:
    def __init__(self):
        self.__products = []

    def add_product(self, product):
        self.__products.append(product)

    def update_quantity(self, product_name, new_quantity):
        for product in self.__products:
            if product.get_name() == product_name:
                product.set_quantity(new_quantity)
                break

    def calculate_total_value(self):
        total_value = 0
        for product in self.__products:
            total_value += product.get_price() * product.get_quantity()
        return total_value


inventory = Inventory()
inventory.add_product(Electronics("Laptop", 1000, 5, "Dell"))
inventory.add_product(Clothing("T-shirt", 20, 10, "M"))
inventory.add_product(Groceries("Milk", 2, 20, "2024-03-15"))
inventory.update_quantity("Milk", 25)
total_value = inventory.calculate_total_value()
print("Total inventory value:", total_value)
```

```
Total inventory value: 5250
```