**Task1:**

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.

**Code:**

class flight\_booking\_system:

def \_\_init\_\_(self):

self.lst = [(1,'karachi','multan','10/dec/24','8:00 AM','12:00pm',2000,500),(1,'karachi','multan','10/dec/24','8:00 AM','12:00pm',2000,500),(1,'karachi','multan','10/dec/24','8:00 AM','12:00pm',2000,500),(1,'karachi','multan','10/dec/24','8:00 AM','12:00pm',2000,500)] # 1==flight n0 , 1=departure time , 2=arrival time, 3=price , 4=number of seats available

def search(self,departure,destination,date):

for i in self.lst:

if i[1]==departure and i[2]==destination and i[3]==date:

print("this seat is available ")

return 'yes'

else:

print("this seat is not avalaible")

return 'no'

def reserved\_seat(self):

print('this seat is reserved from your account')

def generating\_booking\_confirmation(self,con,departure,destination,date):

if con=='yes' and self.search(departure,destination,date)=='yes':

self.reserved\_seat()

else:

print('explore other options or try again later')

if \_\_name\_\_ == "\_\_main\_\_":

name = str(input("Enter Your Name : "))

name=flight\_booking\_system()

print('------welcome into the Flight Booking System-------\nchoice no 1 is search the train and avalaible seat \n choice no 2 generate booking system after search the seat')

choice=int(input("enter your choice from 1 to 2 : "))

if choice==1:

dep,desti,dat=str(input("Enter your deparure city : ")),str(input("Enter your destination city")),str(input("Enter the date of journey "))

name.search(dep,desti,dat)

elif choice==2:

dep,desti,dat=str(input("Enter your deparure city")),str(input("Enter your destination city")),str(input("Enter the date of journey "))

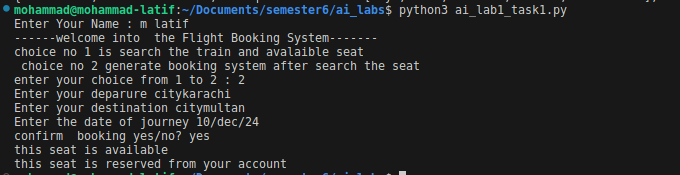
con=input('confirm booking yes/no? ')

name.generating\_booking\_confirmation(con,dep,desti,dat)

else:

print("enter correct number or contact us ")

**output**:



**task2:**

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.

**Code:**

import string

def analayze\_sentance(sentance):

words=0

charecter=0

word = sentance.split()

unique\_words = list(set(word))

alphabets = list(string.ascii\_letters)

for i in sentance:

if i==' ':

words+=1

elif (i in alphabets):

charecter+=1

else:

pass

dict = {"words":words,"charecter":charecter,"unique word list":unique\_words,"average word":words/charecter}

return dict

a=analayze\_sentance('my roll number is 21b-029-se')

print(a)

**output:**

****

**task3:**

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.

**Code:**

class employee:

def \_\_init\_\_(self,name,id,salary):

self.\_\_name = name

self.\_\_id = id

self.\_\_salary=salary

def set\_attribute(self,name,id,salary):

self.\_\_name = name

self.\_\_id = id

self.\_\_salary =salary

def get\_attribute(self):

print(f'name = {self.\_\_name} , id = {self.\_\_id}, salary = {self.\_\_salary}')

def get\_salary(self):

return self.\_\_salary

def \_\_str\_\_(self):

return f'name = {self.\_\_name} , id = {self.\_\_id}, salary = {self.\_\_salary}'

class manager(employee):

def \_\_init\_\_(self,name,id,salary,bonus):

employee.\_\_init\_\_(self,name,id,salary)

self.\_\_bonus=bonus

def calculate\_salary(self):

base=self.get\_salary()

print(f'salary = {base+self.\_\_bonus}')

print("before set the attribute ")

a=employee('m',1,200)

print(a)

print("after set the attribute through set attribute method ")

a.set\_attribute('l',2,40000)

print(a)

print("before set the attribute of manager class ")

m=manager('k',44,20000,400)

print(f"salary of manager ")

m.calculate\_salary()

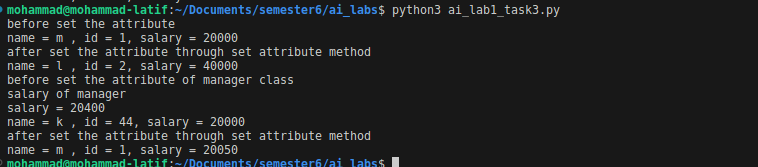
print(m)

print("after set the attribute through set attribute method ")

m.set\_attribute('m',1,200)

print(m)

**output:**

****

**task4:**

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.

**Code:**

class inventory:

def \_\_init\_\_(self):

self.inventry\_list = []

def add\_product(self,id,product,qty,price,type):

product\_detail=[id,product,qty,price,type]

self.inventry\_list.append(product\_detail)

print(f'product added successfully = {self.inventry\_list}')

def update\_qty(self,id,qty):

for i in range(len(self.inventry\_list)):

if self.inventry\_list[i][0]==id:

self.inventry\_list=[i][2]=qty

def total\_value(self):

total=0

for i in range(len(self.inventry\_list)):

a=self.inventry\_list[i][2] \* self.inventry\_list[i][2]

total+=a

print(f'total value of inventry is {total}')

class electronic\_inventory(inventory):

def \_\_init\_\_(self):

self.inventry\_list=[]

def \_\_str\_\_(self):

return(str(self.inventry\_list))

class clothing(inventory):

def \_\_init\_\_(self):

self.inventry\_list=[]

def \_\_str\_\_(self):

return(str(self.inventry\_list))

class groceries(inventory):

def \_\_init\_\_(self):

self.inventry\_list=[]

def \_\_str\_\_(self):

return(str(self.inventry\_list))

e=electronic\_inventory()

e.add\_product(1,'fan',3,2000,'electronic')

e.add\_product(2,'laptop',3,2500,'electronic')

print(e)

c=clothing()

c.add\_product(1,'cotton cloth',3,2000,'cloth')

c.add\_product(2,'cotton cloth',4,200,'cloth')

print(e)

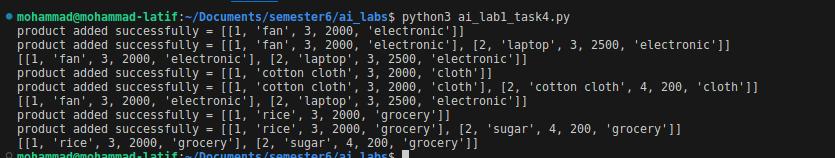
g=groceries()

g.add\_product(1,'rice',3,2000,'grocery')

g.add\_product(2,'sugar',4,200,'grocery')

print(g)

**Output:**

****