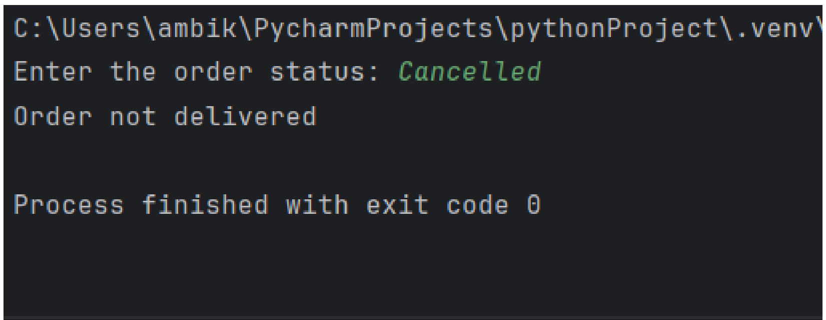
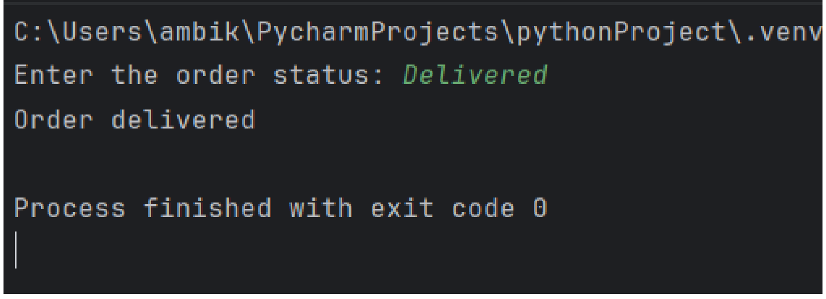
**Coding**
**Task 1: Control Flow Statements**

1. Write a program that checks whether a given order is delivered or not based on its status (e.g.**, "Processing," "Delivered," "Cancelled").** Use if-else statements for this.

```
def checkStatus(status):
    if status=='Delivered':
        print('Order delivered')
    else:
        print('Order not delivered')
status=input('Enter the order status: ')
checkStatus(status)
```



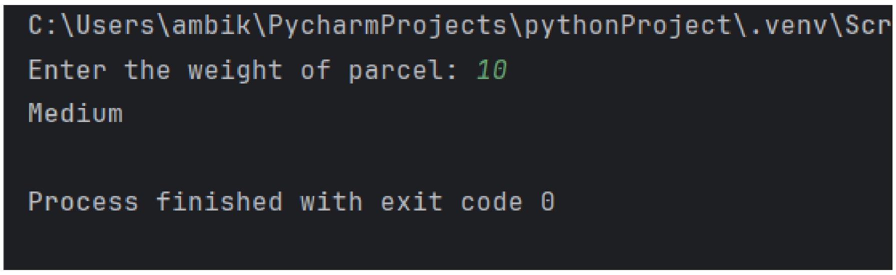
```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\
Enter the order status: Cancelled
Order not delivered

Process finished with exit code 0
```



```
C:\Users\ambik\PycharmProjects\pythonProject\.venv
Enter the order status: Delivered
Order delivered

Process finished with exit code 0
```

**2.** Implement a **switch-case statement** to categorize parcels based on their weight into **"Light," "Medium," or "Heavy."**

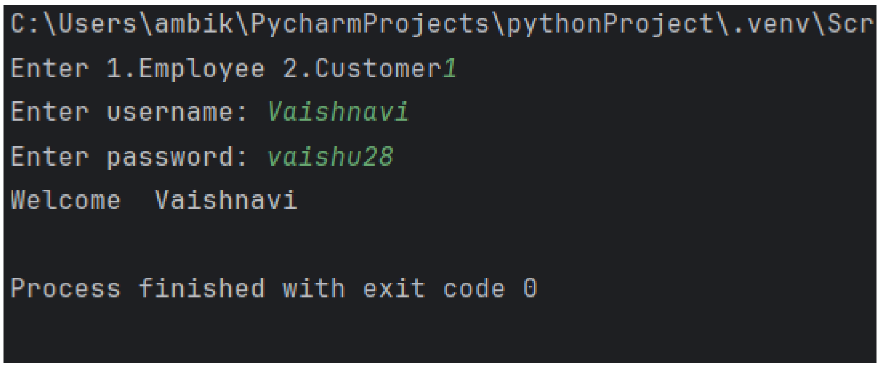
```
def detectWeiht(weight):
    match weight:
        case w if w<5:
            return 'Light'
        case w if w>=5 and w<20:
            return 'Medium'
        case _:
            return 'Heavy'
weight=int(input('Enter the weight of parcel: '))
print(detectWeiht(weight))
```



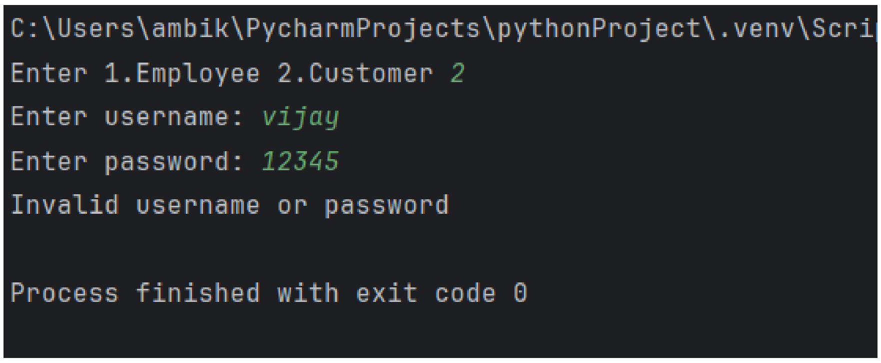
```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scr
Enter the weight of parcel: 10
Medium

Process finished with exit code 0
```

**3. Implement User Authentication 1. Create a login system for employees and customers using Java
control flow statements.**

```
employeeDetails = {
  "Sindhu":"sindhuknk",
  "Vaishnavi":"vaishu28"
}
customerDetails = {
  "Surya":"ford20",
  "Keerthana":"Voice04"
}

def login():
  loginType = int(input('Enter 1.Employee 2.Customer'))
  name = input('Enter username: ')
  password = input('Enter password: ')

  if loginType==1:
    if name in employeeDetails and password == employeeDetails[name]:
      print("Welcome ",name)
    else:
      print('Invalid username or password')
  elif loginType == 2:
    if name in customerDetails and password == customerDetails[name]:
      print("Welcome ", name)
    else:
      print('Invalid username or password')

login()
```



```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scr
Enter 1.Employee 2.Customer1
Enter username: Vaishnavi
Enter password: vaishu28
Welcome  Vaishnavi

Process finished with exit code 0
```

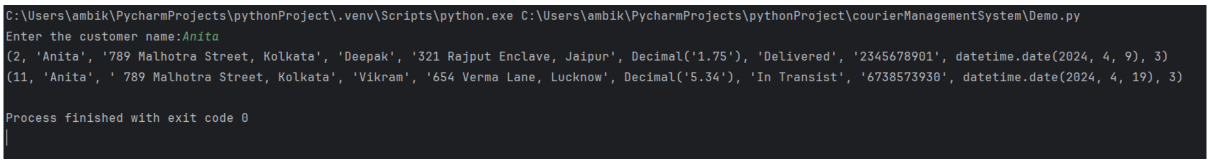


```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scri
Enter 1.Employee 2.Customer 2
Enter username: vijay
Enter password: 12345
Invalid username or password

Process finished with exit code 0
```

**Task 2: Loops and Iteration**

5. Write a Java program that uses a for loop to display all the orders for a specific customer.

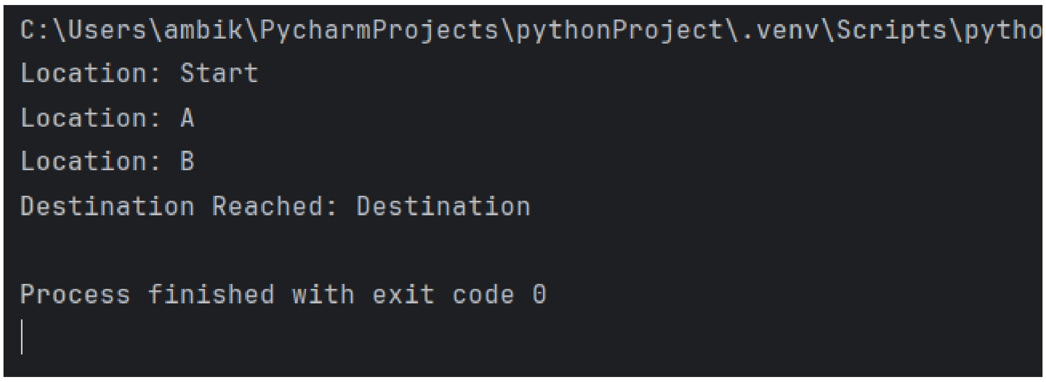
```
def customerOrder(cursor,name):
    query="select * from courier where senderName=%s"
    cursor.execute(query,(name,))
    result=cursor.fetchall()
    for row in result:
        print(row)
    return
```



```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\ambik\PycharmProjects\pythonProject\courierManagementSystem\Demo.py
Enter the customer name:Anita
(2, 'Anita', '789 Malhotra Street, Kolkata', 'Deepak', '321 Rajput Enclave, Jaipur', Decimal('1.75'), 'Delivered', '2345678901', datetime.date(2024, 4, 9), 3)
(11, 'Anita', ' 789 Malhotra Street, Kolkata', 'Vikram', '654 Verma Lane, Lucknow', Decimal('5.34'), 'In Transist', '6738573930', datetime.date(2024, 4, 19), 3)

Process finished with exit code 0
```

**6.** Implement a **while loop** to track the real-time location of a courier until it reaches its destination.

```
import time
def track_courier(current, destination):
    while current != destination:
        print("Location:", current)

        if current == "Start":
            current = "A"
        elif current == "A":
            current = "B"
        elif current == "B":
            current = destination

        time.sleep(2)

    print("Destination Reached:", destination)


starting = "Start"
final = "Destination"
track_courier(starting, final)
```

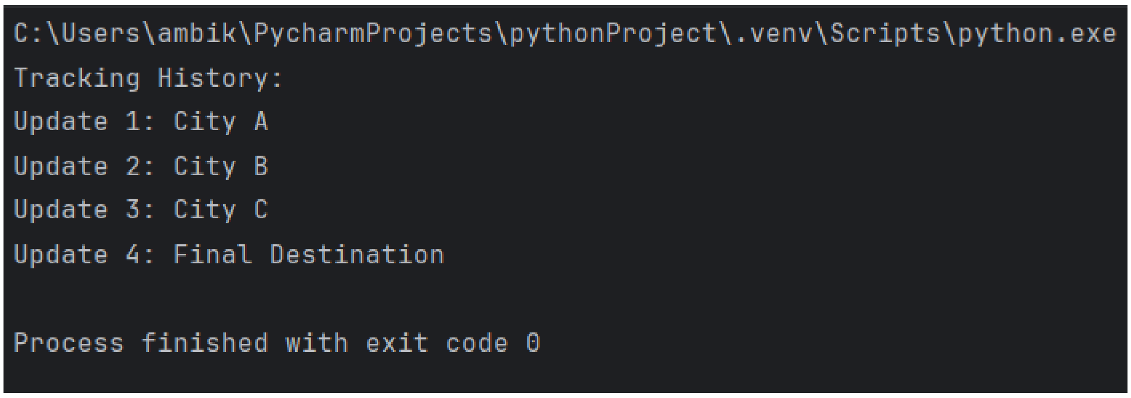


```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\pytho
Location: Start
Location: A
Location: B
Destination Reached: Destination

Process finished with exit code 0
```

**Task 3: Arrays and Data Structures**

7. Create an array to store the tracking history of a parcel, where each entry represents a location update.

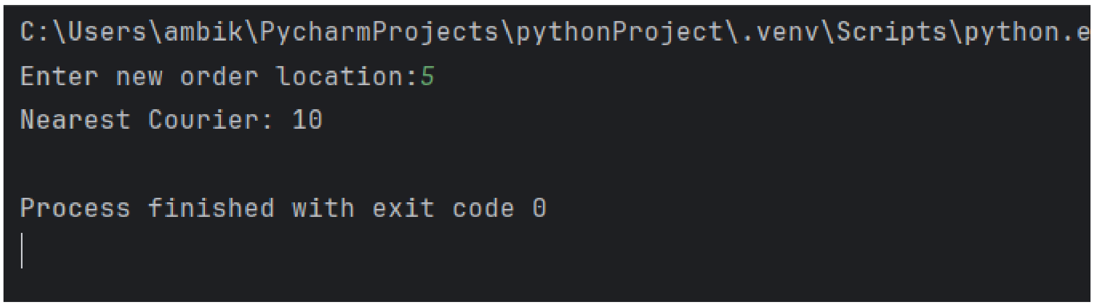
```
tracking_history = []

updates = ["City A", "City B", "City C", "Final Destination"]
for i in range(len(updates)):
    tracking_history.append(updates[i])

print("Tracking History:")
for i in range(len(tracking_history)):
    print(f"Update {i+1}: {tracking_history[i]}")
```



```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.exe
Tracking History:
Update 1: City A
Update 2: City B
Update 3: City C
Update 4: Final Destination

Process finished with exit code 0
```

8. Implement a method to find the nearest available courier for a new order using an **array of couriers.**

```
def nearest_courier(pickup, couriers):
    nearest = None
    min_dist = float('inf')

    for courier in couriers:
        dist = abs(pickup - courier)
        if dist < min_dist:
            min_dist = dist
            nearest = courier

    return nearest

couriers = [10, 15, 20, 25]
new_order = int(input("Enter new order location:"))

ans = nearest_courier(new_order, couriers)
print("Nearest Courier:", ans)
```

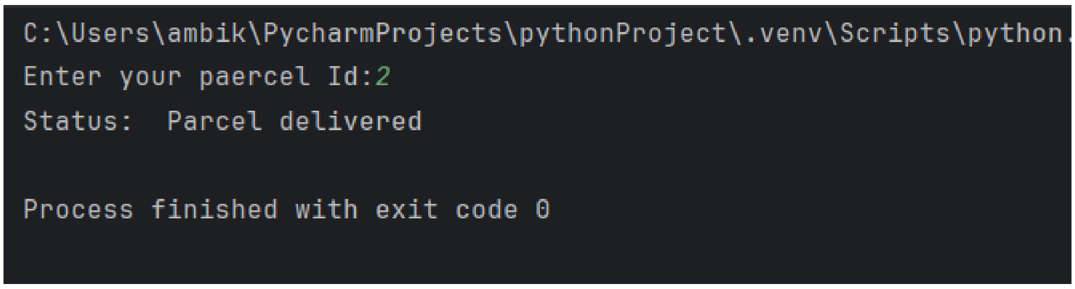


```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.e
Enter new order location:5
Nearest Courier: 10

Process finished with exit code 0
```

**Task 4: Strings,2d Arrays, user defined functions,Hashmap**

9. **Parcel Tracking**: Create a program that allows users to input a parcel tracking number.Store the tracking number and Status in 2d String Array. Initialize the array with values. Then, simulate the tracking process by displaying messages like "Parcel in transit," "Parcel out for delivery," or "Parcel delivered" based on the tracking number's status.

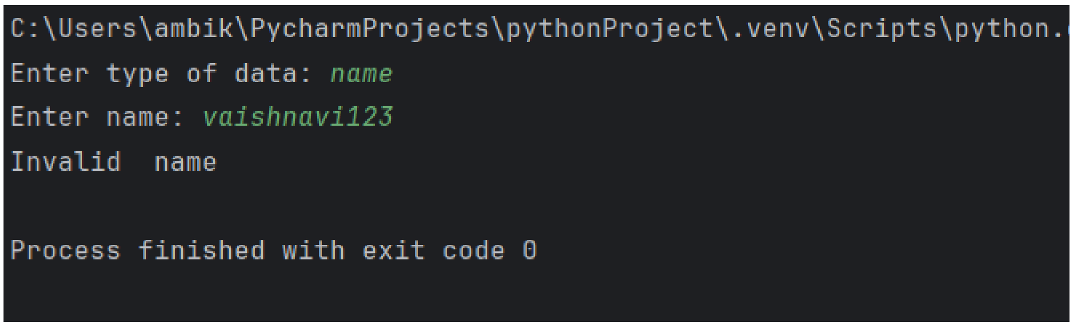
```
ar=[[1,"Parcel in transit"],[2,"Parcel delivered"],[3,"Parcel out for delivery"]]
Id=int(input("Enter your paercel Id:"))

for i in range(len(ar)):
  if ar[i][0]==Id:
    print("Status: ",ar[i][1])
    break
```



```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python
Enter your paercel Id:2
Status:  Parcel delivered

Process finished with exit code 0
```

10. **Customer Data Validation**: Write a function which takes 2 parameters, data-denotes the data and detail-denotes if it is name addtress or phone number. Validate customer information based on following critirea. Ensure that names contain only letters and are properly capitalized, addresses do not contain special characters, and phone numbers follow a specific format (e.g., ###-###-####).

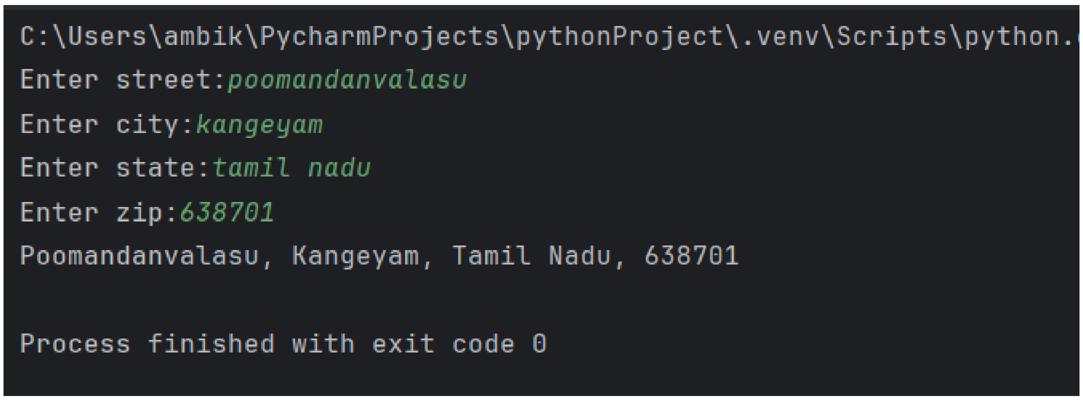
```
import re;
def dataValidation(data,type):
  if(type=="name"):
    if(data.isalpha() and data.istitle()):
      print(data," is valid name")
    else:
      print("Invalid ",type)
  elif(type=="address"):
    if(data.isalnum()):
      print(data, " is valid address")
    else:
      print("Invalid ", type)
  else:
    pattern = r'\d{3}-\d{3}-\d{4}'
    if re.match(pattern, data):
      print(data, " is valid number")
    else:
      print("Invalid ", type)

type = input("Enter type of data: ")
data = input("Enter "+type+": ")

dataValidation(data,type)
```



```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.
Enter type of data: name
Enter name: vaishnavi123
Invalid  name

Process finished with exit code 0
```

11. **Address Formatting:** Develop a function that takes an address as input (street, city, state, zip code) and formats it correctly, including capitalizing the first letter of each word and properly formatting the zip code.

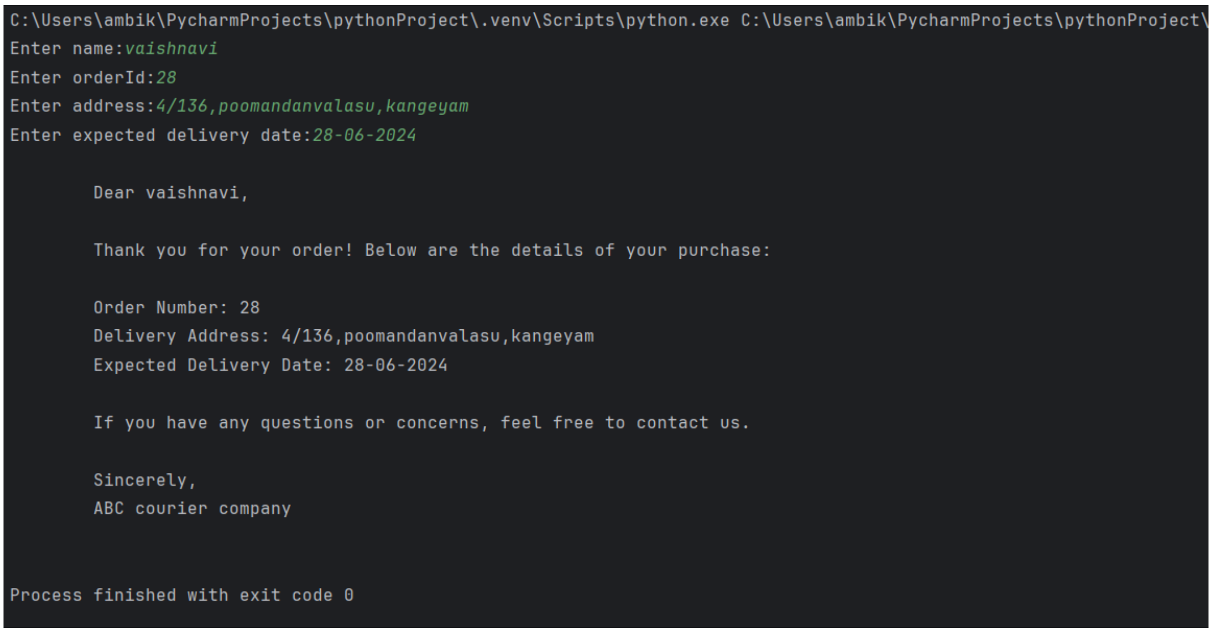
```
def formatAddress(street,city,state,zip):
    ans = ""
    ans += street+", "+city+", "+state+", "+str(zip)
    ans = ans.title()
    return ans

street=input("Enter street:")
city=input("Enter city:")
state=input("Enter state:")
zip=int(input("Enter zip:"))

print(formatAddress(street,city,state,zip))
```



```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.
Enter street:poomandanvalasu
Enter city:kangeyam
Enter state:tamil nadu
Enter zip:638701
Poomandanvalasu, Kangeyam, Tamil Nadu, 638701

Process finished with exit code 0
```

12. **Order Confirmation Email**: Create a program that generates an order confirmation email. The email should include details such as the customer's name, order number, delivery address, and expected delivery date.

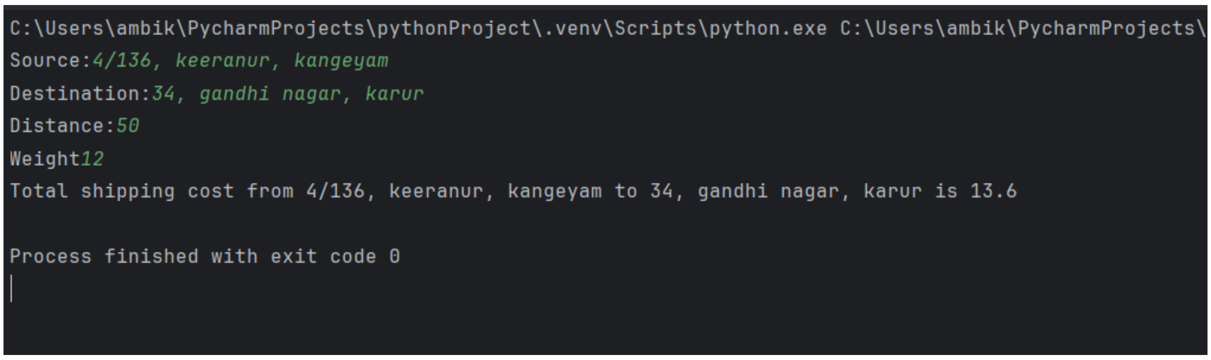
```
def orderConfirmation(name,orderDate,address,expectedDate):
    emailContent = f"""
        Dear {name},

        Thank you for your order! Below are the details of your purchase:

        Order Number: {orderDate}
        Delivery Address: {address}
        Expected Delivery Date: {expectedDate}

        If you have any questions or concerns, feel free to contact us.
    Sincerely,
    ABC courier company
    """
  return emailContent

name=input("Enter name:")
orderId=int(input("Enter orderId:"))
address=input("Enter address:")
expectedDate=input("Enter expected delivery date:")

print(orderConfirmation(name,orderId,address,expectedDate))
```



```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\ambik\PycharmProjects\pythonProject\
Enter name:vaishnavi
Enter orderId:28
Enter address:4/136,poomandanvalasu,kangeyam
Enter expected delivery date:28-06-2024

        Dear vaishnavi,

        Thank you for your order! Below are the details of your purchase:

        Order Number: 28
        Delivery Address: 4/136,poomandanvalasu,kangeyam
        Expected Delivery Date: 28-06-2024

        If you have any questions or concerns, feel free to contact us.

        Sincerely,
        ABC courier company


Process finished with exit code 0
```

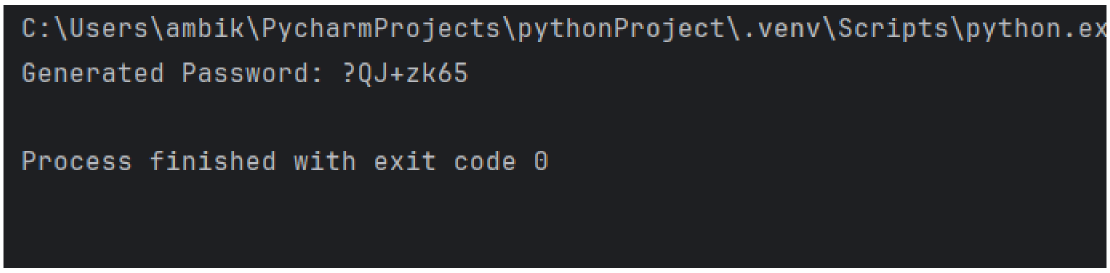
13. **Calculate Shipping Costs**: Develop a function that calculates the shipping cost based on the distance between two locations and the weight of the parcel. You can use string inputs for the source and destination addresses.

```
def shippingCost(source,destination,dist,weight):
  totalCost=dist*0.2
  totalCost+=weight*0.3

  print(f"Total shipping cost from {source} to {destination} is {totalCost}")

source=input("Source:")
destination=input("Destination:")
dist=int(input("Distance:"))
weight=int(input("Weight"))

shippingCost(source,destination,dist,weight)
```


```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\ambik\PycharmProjects\
Source:4/136, keeranur, kangeyam
Destination:34, gandhi nagar, karur
Distance:50
Weight12
Total shipping cost from 4/136, keeranur, kangeyam to 34, gandhi nagar, karur is 13.6

Process finished with exit code 0
```

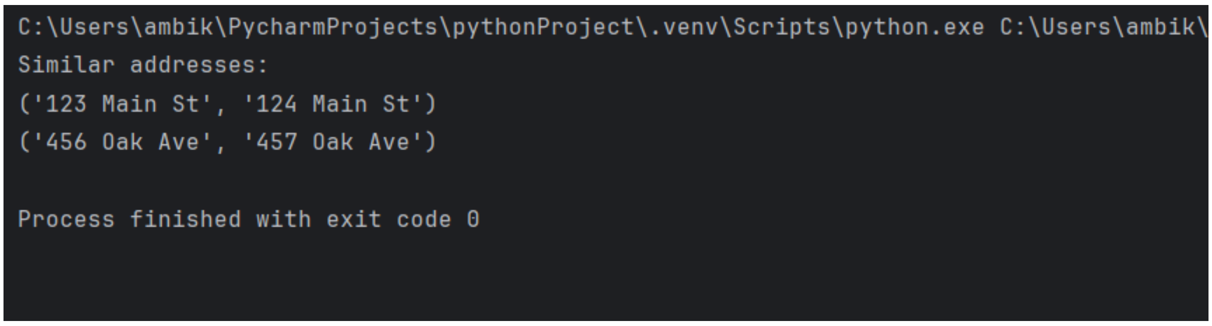
14. **Password Generator**: Create a function that generates secure passwords for courier system accounts. Ensure the passwords contain a mix of uppercase letters, lowercase letters, numbers, and special characters.

```
import string
import random

def generate_password(length=8):
    uppercase = string.ascii_uppercase
    lowercase = string.ascii_lowercase
    digits = string.digits
    special = string.punctuation

    all_chars = uppercase + lowercase + digits + special

    password = ''.join(random.choices(all_chars, k=length))
    return password

password = generate_password()
print("Generated Password:", password)
```


```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.ex
Generated Password: ?QJ+zk65

Process finished with exit code 0
```

15. **Find Similar Addresses:** Implement a function that finds similar addresses in the system. This can be useful for identifying duplicate customer entries or optimizing delivery routes. Use string functions to implement this.

```
def similarAddresses(addresses):
    similarPairs = []

    for i, address1 in enumerate(addresses):
        words1 = address1.split()
        for address2 in addresses[i+1:]:
            words2 = address2.split()
            common_words = set(words1).intersection(set(words2))
            if(len(common_words)>=(max(len(words1),len(words2)))/2):
                similarPairs.append((address1,address2))
    return similarPairs

addresses = [
    "123 Main St",
    "124 Main St",
    "1234 Elm St",
    "456 Oak Ave",
    "457 Oak Ave"
]

similarPairs = similarAddresses(addresses)
print("Similar addresses:")
for pair in similarPairs:
    print(pair)
```



```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\ambik\
Similar addresses:
('123 Main St', '124 Main St')
('456 Oak Ave', '457 Oak Ave')

Process finished with exit code 0
```

**Task 5: Object Oriented Programming**
**Scope : Entity classes/Models/POJO, Abstraction/Encapsulation**
Create the following **model/entity classes** within package **entities** with variables declared private, constructors(default and parametrized,getters,setters and toString())
**1. User Class:**
**Variables**:
userID , userName , email , password , contactNumber , address

```
class User:
    def __init__(self, userID="", userName="", email="", password="",
contactNumber="", address=""):
        self.__userID = userID
        self.__userName = userName
        self.__email = email
        self.__password = password
        self.__contactNumber = contactNumber
        self.__address = address

    # Getters
    def get_userID(self):
        return self.__userID

    def get_userName(self):
        return self.__userName

    def get_email(self):
        return self.__email

    def get_password(self):
        return self.__password

    def get_contactNumber(self):
        return self.__contactNumber

    def get_address(self):
        return self.__address

    # Setters
    def set_userID(self, userID):
        self.__userID = userID

    def set_userName(self, userName):
        self.__userName = userName

    def set_email(self, email):
        self.__email = email

    def set_password(self, password):
        self.__password = password

    def set_contactNumber(self, contactNumber):
        self.__contactNumber = contactNumber

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

    def __str__(self):
        return (f"User ID: {self.__userID}, Username: {self.__userName}, 
Email: {self.__email}, "
               f"Password: {self.__password}, Contact Number: 
{self.__contactNumber}, Address: {self.__address}")
```

**2. Courier Class**

**Variables:** courierID , senderName , senderAddress , receiverName , receiverAddress , weight , status, trackingNumber , deliveryDate ,userId

```
class Courier:
    def __init__(self, courierID="", senderName="", senderAddress="", 
receiverName="", receiverAddress="",
                 weight=0.0, status="", trackingNumber="", deliveryDate="", 
userId=""):
        self.__courierID = courierID
        self.__senderName = senderName
        self.__senderAddress = senderAddress
        self.__receiverName = receiverName
        self.__receiverAddress = receiverAddress
        self.__weight = weight
        self.__status = status
        self.__trackingNumber = trackingNumber
        self.__deliveryDate = deliveryDate
        self.__userId = userId

    # Getters
    def get_courierID(self):
        return self.__courierID

    def get_senderName(self):
        return self.__senderName

    def get_senderAddress(self):
        return self.__senderAddress
    def get_receiverName(self):
        return self.__receiverName

    def get_receiverAddress(self):
        return self.__receiverAddress

    def get_weight(self):
        return self.__weight

    def get_status(self):
        return self.__status

    def get_trackingNumber(self):
        return self.__trackingNumber

    def get_deliveryDate(self):
        return self.__deliveryDate

    def get_userId(self):
        return self.__userId

    # Setters
    def set_courierID(self, courierID):
        self.__courierID = courierID

    def set_senderName(self, senderName):
        self.__senderName = senderName

    def set_senderAddress(self, senderAddress):
        self.__senderAddress = senderAddress

    def set_receiverName(self, receiverName):
        self.__receiverName = receiverName

    def set_receiverAddress(self, receiverAddress):
        self.__receiverAddress = receiverAddress

    def set_weight(self, weight):
        self.__weight = weight

    def set_status(self, status):
        self.__status = status

    def set_trackingNumber(self, trackingNumber):
        self.__trackingNumber = trackingNumber

    def set_deliveryDate(self, deliveryDate):
        self.__deliveryDate = deliveryDate

    def set_userId(self, userId):
        self.__userId = userId

    def __str__(self):
        return f"Courier ID: {self.__courierID}, Sender Name:
{self.__senderName}, " \
               f"Sender Address: {self.__senderAddress}, Receiver Name:
{self.__receiverName}, " \
               f"Receiver Address: {self.__receiverAddress}, Weight:
{self.__weight}, " \
               f"Status: {self.__status}, Tracking Number:
```

```
{self.__trackingNumber}, " \
               f"Delivery Date: {self.__deliveryDate}, User ID:
{self.__userId}"
```

**3. Employee Class**:
**Variables** employeeID , employeeName , email , contactNumber , role String, salary

```
class Employee:
    def __init__(self, employeeID="", employeeName="", email="",
contactNumber="", role="", salary=0.0):
        self.__employeeID = employeeID
        self.__employeeName = employeeName
        self.__email = email
        self.__contactNumber = contactNumber
        self.__role = role
        self.__salary = salary

    # Getters
    def get_employeeID(self):
        return self.__employeeID

    def get_employeeName(self):
        return self.__employeeName

    def get_email(self):
        return self.__email

    def get_contactNumber(self):
        return self.__contactNumber

    def get_role(self):
        return self.__role

    def get_salary(self):
        return self.__salary

    # Setters
    def set_employeeID(self, employeeID):
        self.__employeeID = employeeID

    def set_employeeName(self, employeeName):
        self.__employeeName = employeeName

    def set_email(self, email):
        self.__email = email

    def set_contactNumber(self, contactNumber):
        self.__contactNumber = contactNumber

    def set_role(self, role):
        self.__role = role

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

    def __str__(self):
        return f"Employee ID: {self.__employeeID}, Employee Name:
{self.__employeeName}, " \
               f"Email: {self.__email}, Contact Number:
```

```
{self.__contactNumber}, " \
               f"Role: {self.__role}, Salary: {self.__salary}"
```

**4. Location Class**

**Variables** LocationID , LocationName , Address

```
class Location:
    def __init__(self, locationID="", locationName="", address=""):
        self.__locationID = locationID
        self.__locationName = locationName
        self.__address = address

    # Getters
    def get_locationID(self):
        return self.__locationID

    def get_locationName(self):
        return self.__locationName

    def get_address(self):
        return self.__address

    # Setters
    def set_locationID(self, locationID):
        self.__locationID = locationID

    def set_locationName(self, locationName):
        self.__locationName = locationName

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

    def __str__(self):
        return f"Location ID: {self.__locationID}, Location Name:
{self.__locationName}, " \
               f"Address: {self.__address}"
```

**5. CourierCompany Class**

**Variables** companyName , courierDetails -collection of Courier Objects, employeeDetails collection of Employee Objects, locationDetails - collection of Location Objects.

```
from courier import Courier
from employee import Employee
from location import Location
class CourierCompany:
    def __init__(self, companyName="", courierDetails=None,
employeeDetails=None, locationDetails=None):
        if courierDetails is None:
            courierDetails = []
        if employeeDetails is None:
            employeeDetails = []
        if locationDetails is None:
            locationDetails = []
        self.__companyName = companyName
        self.__courierDetails = courierDetails
        self.__employeeDetails = employeeDetails
        self.__locationDetails = locationDetails
    # Getters
    def get_companyName(self):
        return self.__companyName

    def get_courierDetails(self):
        return self.__courierDetails

    def get_employeeDetails(self):
        return self.__employeeDetails

    def get_locationDetails(self):
        return self.__locationDetails

    # Setters
    def set_companyName(self, companyName):
        self.__companyName = companyName

    def set_courierDetails(self, courierDetails):
        self.__courierDetails = courierDetails

    def set_employeeDetails(self, employeeDetails):
        self.__employeeDetails = employeeDetails

    def set_locationDetails(self, locationDetails):
        self.__locationDetails = locationDetails

    def __str__(self):
        return f"Company Name: {self.__companyName}, Courier Details:
{self.__courierDetails}, " \
               f"Employee Details: {self.__employeeDetails}, Location
Details: {self.__locationDetails}"
```

**6. Payment Class:**
**Variables** PaymentID long, CourierID long, Amount double, PaymentDate Date

```
class Payment:
    def __init__(self, paymentID=0, courierID=0, amount=0.0,
paymentDate=None):
        self.__paymentID = paymentID
        self.__courierID = courierID
        self.__amount = amount
        self.__paymentDate = paymentDate

    # Getters
    def get_paymentID(self):
        return self.__paymentID

    def get_courierID(self):
        return self.__courierID

    def get_amount(self):
        return self.__amount

    def get_paymentDate(self):
        return self.__paymentDate

    # Setters
    def set_paymentID(self, paymentID):
        self.__paymentID = paymentID
    def set_courierID(self, courierID):
        self.__courierID = courierID

    def set_amount(self, amount):
        self.__amount = amount

    def set_paymentDate(self, paymentDate):
        self.__paymentDate = paymentDate

    def __str__(self):
        return f"Payment ID: {self.__paymentID}, Courier ID: 
{self.__courierID}, " \
               f"Amount: {self.__amount}, Payment Date: 
{self.__paymentDate}"
```

**Task 6: Service Provider Interface /Abstract class**
Create 2 **Interface /Abstract class ICourierUserService** and **ICourierAdminService interface ICourierUserService {**
**// Customer-related functions**
**placeOrder()**
/** Place a new courier order.
* @param courierObj Courier object created using values entered by users
* @return The unique tracking number for the courier order .
**getOrderStatus();**
/**Get the status of a courier order.
*@param trackingNumber The tracking number of the courier order.
* @return The status of the courier order (e.g., **yetToTransit, In Transit**, **Delivered**).
*/
**cancelOrder**()
/** Cancel a courier order.
* @param trackingNumber The tracking number of the courier order to be canceled.
* @return True if the order was successfully canceled, false otherwise.*/ © Hexaware Technologies Limited. All rights
www.hexaware.com
**getAssignedOrder();**
/** Get a list of orders assigned to a specific courier staff member
* @param courierStaffId The ID of the courier staff member.
* @return A list of courier orders assigned to the staff member.*/

```
from abc import ABC,abstractmethod

class ICourierUserService(ABC):
    @abstractmethod
    def placeOrder(self,courierObj):
        pass
    @abstractmethod
    def getOrderStatus(self,trackingNumber):
        pass
    @abstractmethod
    def cancelOrder(self,trackingNumber):
        pass
    @abstractmethod
    def getAssignedOrder(self,courierStaffId):
        pass
```

**// Admin functions**
**ICourierAdminService**
**int addCourierStaff(Employee obj);**
/** Add a new courier staff member to the system.
* @param name The name of the courier staff member.
* @param contactNumber The contact number of the courier staff member.
* @return The ID of the newly added courier staff member.
*/

```
class ICourierAdminService:
    @abstractmethod
    def addCourierStaff(self,employeeObj):
        pass
```

**Task 7: Exception Handling**
Define the following custom exceptions and throw them in methods whenever needed . Handle all the
excpetionsin main method,
1. **TrackingNumberNotFoundException** :throw this exception when user try to withdraw amount or transfer amount to another acco
2. **InvalidEmployeeIdException** throw this exception when id entered for the employee not existing in
the system

**Exception.py:**

```
class TrackingNumberNotFoundException(Exception):
    def __init__(self):
        self.message = "Invalid tracking number"
class  InvalidEmployeeIdException(Exception):
    def __init__(self):
        self.message = "Invalid EmployeeId"
```

**main.py**

```
import mysql.connector
from Exception.exceptions import TrackingNumberNotFoundException,
InvalidEmployeeIdException

class exceptionHandling:
    def isTracknumExists(self,trackingNum):
        raise(TrackingNumberNotFoundException())

    def isValidEmployee(self,employeeId):
        raise(InvalidEmployeeIdException())


conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Vaishu@28",
    database="courier_management_system"
)

cursor = conn.cursor()

try:
    obj = exceptionHandling()
    # 1
    trackingNum = int(input())
    obj.isTracknumExists(trackingNum)
    #2
    employeeId=int(input())
    obj.isValidEmployee(employeeId)

except TrackingNumberNotFoundException as e:
    print("Error:"+e.message)
except InvalidEmployeeIdException as e:
    print("Error:" + e.message)
finally:
    cursor.close()
    conn.close()
```

**Task 8: Service implementation**

1.Create **CourierUserServiceImpl** class which implements **ICourierUserService** interface which holds a variable named **companyObj** of type **CourierCompany**.

This variable can be used to access the Object Arrays to access data relevant in method implementations.

```
from entities.courier import Courier
from entities.employee import Employee
from entities.location import Location
from entities.courierCompany import CourierCompany
from serviceProvider import ICourierUserService


class CourierUserServiceImpl(ICourierUserService):
    def __init__(self, companyObj: CourierCompany):
        self.companyObj = companyObj

    def placeOrder(self, courierObj: Courier):
        self.companyObj.get_courierDetails().append(courierObj)

    def getOrderStatus(self, trackingNumber):
        for courier in self.companyObj.get_courierDetails():
            if courier.get_trackingNumber() == trackingNumber:
                return courier.get_status()
        return "Order not found."

    def cancelOrder(self, trackingNumber):
        for courier in self.companyObj.get_courierDetails():
            if courier.get_trackingNumber() == trackingNumber:
                courier.set_status("Cancelled")
                return "Order cancelled successfully."
        return "Order not found."

    def getAssignedOrder(self, courierStaffId):
        assigned_orders = []
        for courier in self.companyObj.get_courierDetails():
            if courier.get_assignedStaffId() == courierStaffId:
                assigned_orders.append(courier)
        return assigned_orders
```

2. Create **CourierAdminService Impl** class which inherits from **CourierUserServiceImpl** and implements **ICourierAdminService** interface.

```
from dao.courierUserServiceImpl import CourierUserServiceImpl
from dao.serviceProvider import ICourierAdminService
from entities.courierCompany import CourierCompany
from entities.employee import Employee

class CourierAdminServiceImpl(CourierUserServiceImpl,
ICourierAdminService):
    def __init__(self, companyObj: CourierCompany):
        super().__init__(companyObj)

    def addCourierStaff(self, employeeObj: Employee):
        self.companyObj.get_employeeDetails().append(employeeObj)
        return employeeObj.get_employeeID()
```

**Task 9: Database Interaction**
Connect your application to the SQL database for the Courier Management System
1. Write code to establish a connection to your SQL database.
Create a class **DBConnection** in a package **connectionutil** with a static variable **connection** of Type **Connection** and a static method **getConnection()** which returns connection.
Connection properties supplied in the connection string should be read from a property file.

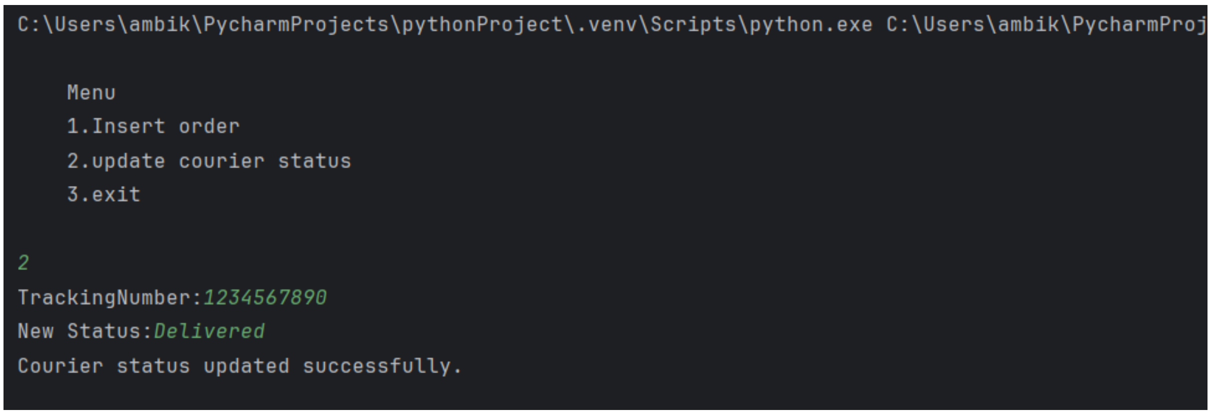
```
import mysql.connector

class DBConnection:
    connection=None
    @staticmethod
    def getConnection():
        f=open("propertyFile","r")
        lines=f.readlines()
        host=lines[0].strip()
        username=lines[1].strip()
        password=lines[2].strip()
        database=lines[3].strip()

        DBConnection.connection=mysql.connector.connect(
            host=host,
            user=username,
            password=password,
            database=database
        )

        return DBConnection.connection
```

2. Create a Service class **CourierServiceDb** in **dao** with a static variable named connection of type **Connection** which can be assigned in the constructor by invoking the method in **DBConnection** Class.

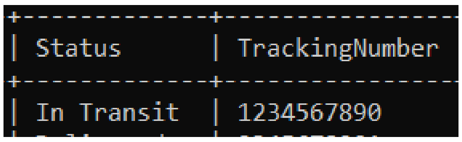
```
from connectionutil.DBConnection import DBConnection

class CourierServiceDb:
    connection = None

    @staticmethod
    def initialize_connection():
        if CourierServiceDb.connection is None:
            CourierServiceDb.connection = DBConnection.getConnection()
```

3. Include methods to **insert, update, and retrieve data** from the database (e.g., **inserting a new order, updating courier status**).

```
def updateCourierStatus(self,trackingNumber, new_status):

    cursor = CourierServiceDb.connection.cursor()

    update_query = """
        UPDATE courier
        SET Status = %s
        WHERE TrackingNumber = %s
        """

    data = (new_status, trackingNumber)

    cursor.execute(update_query, data)
    CourierServiceDb.connection.commit()
    print("Courier status updated successfully.")
```

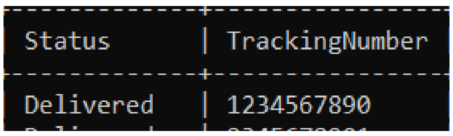


```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\ambik\PycharmProj

    Menu
    1.Insert order
    2.update courier status
    3.exit

2
TrackingNumber:1234567890
New Status:Delivered
Courier status updated successfully.
```



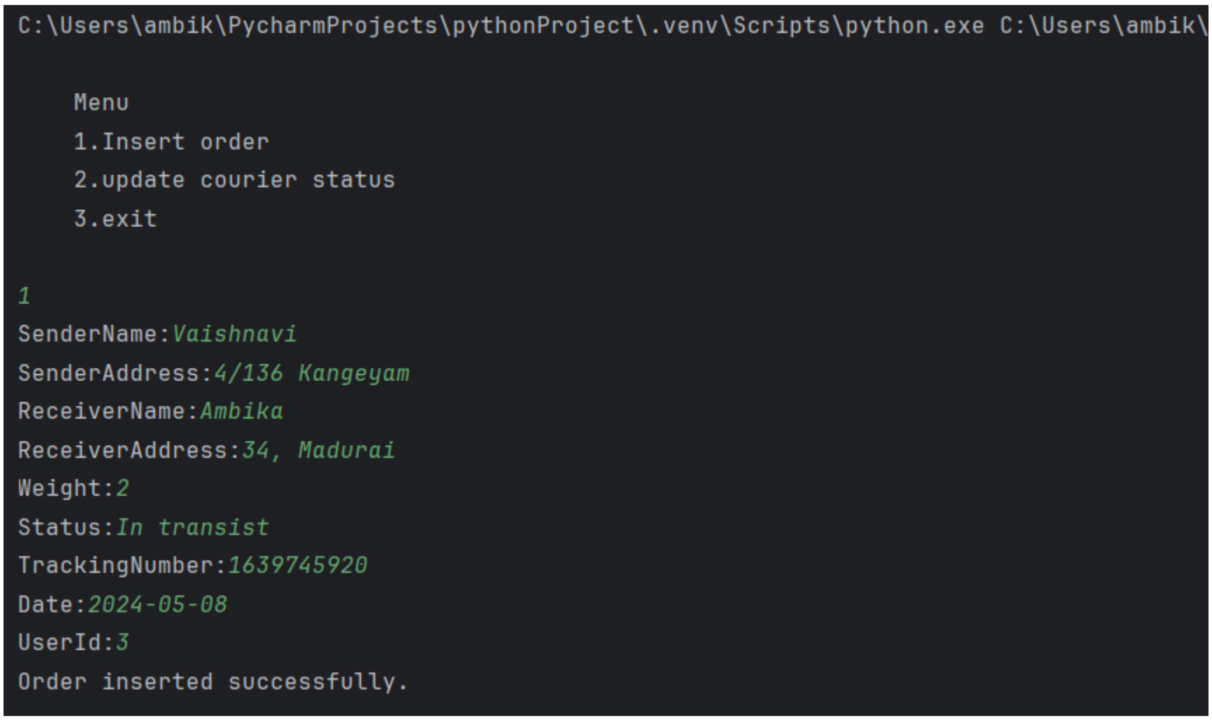
```
+-------------+----------------
| Status      | TrackingNumber
+-------------+----------------
| In Transit  | 1234567890
```

→→→



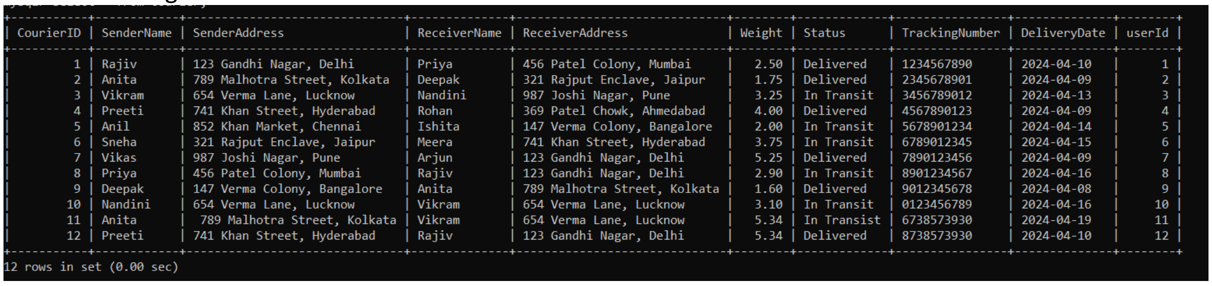
```
-------------+----------------
 Status      | TrackingNumber
-------------+----------------
 Delivered   | 1234567890
```

```
def insertOrder(self,courier_data):

    cursor = CourierServiceDb.connection.cursor()

    insert_query = """
        INSERT INTO courier (SenderName, SenderAddress, ReceiverName,
ReceiverAddress, Weight, Status, TrackingNumber, DeliveryDate, userId)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
        """

    data = (
        courier_data.get_senderName(),
        courier_data.get_senderAddress(),
        courier_data.get_receiverName(),
        courier_data.get_receiverAddress(),
        courier_data.get_weight(),
        courier_data.get_status(),
        courier_data.get_trackingNumber(),
        courier_data.get_deliveryDate(),
        courier_data.get_userId()
    )
    cursor.execute(insert_query, data)
    CourierServiceDb.connection.commit()
    print("Order inserted successfully.")
```



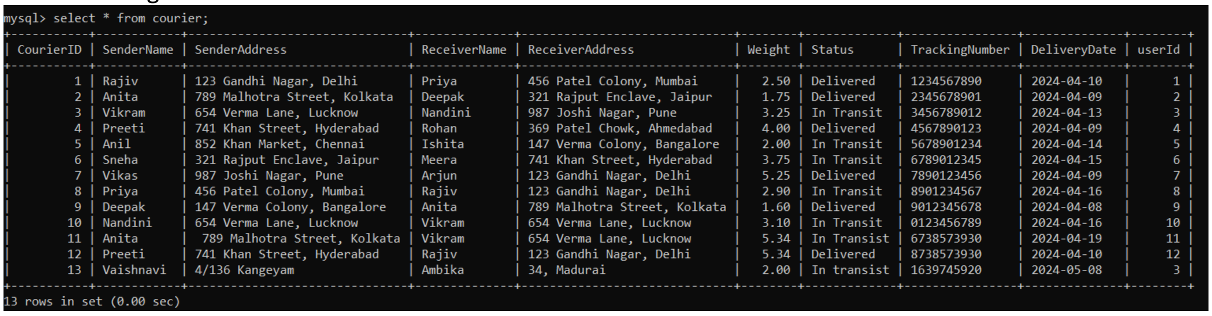
```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\ambik\

    Menu
    1.Insert order
    2.update courier status
    3.exit

1
SenderName:Vaishnavi
SenderAddress:4/136 Kangeyam
ReceiverName:Ambika
ReceiverAddress:34, Madurai
Weight:2
Status:In transist
TrackingNumber:1639745920
Date:2024-05-08
UserId:3
Order inserted successfully.
```

Before inserting order



```
+-----------+------------+-----------------------------+--------------+-------------------------------+--------+-------------+----------------+--------------+--------+
| CourierID | SenderName | SenderAddress               | ReceiverName | ReceiverAddress               | Weight | Status      | TrackingNumber | DeliveryDate | userId |
+-----------+------------+-----------------------------+--------------+-------------------------------+--------+-------------+----------------+--------------+--------+
|         1 | Rajiv      | 123 Gandhi Nagar, Delhi     | Priya        | 456 Patel Colony, Mumbai      |   2.50 | Delivered   | 1234567890     | 2024-04-10   |      1 |
|         2 | Anita      | 789 Malhotra Street, Kolkata | Deepak      | 321 Rajput Enclave, Jaipur    |   1.75 | Delivered   | 2345678901     | 2024-04-09   |      2 |
|         3 | Vikram     | 654 Verma Lane, Lucknow     | Nandini      | 987 Joshi Nagar, Pune         |   3.25 | In Transit  | 3456789012     | 2024-04-13   |      3 |
|         4 | Preeti     | 741 Khan Street, Hyderabad  | Rohan        | 369 Patel Chowk, Ahmedabad    |   4.00 | Delivered   | 4567890123     | 2024-04-09   |      4 |
|         5 | Anil       | 852 Khan Market, Chennai    | Ishita       | 147 Verma Colony, Bangalore   |   2.00 | In Transit  | 5678901234     | 2024-04-14   |      5 |
|         6 | Sneha      | 321 Rajput Enclave, Jaipur  | Meera        | 741 Khan Street, Hyderabad    |   3.75 | In Transit  | 6789012345     | 2024-04-15   |      6 |
|         7 | Vikas      | 987 Joshi Nagar, Pune       | Arjun        | 123 Gandhi Nagar, Delhi       |   5.25 | Delivered   | 7890123456     | 2024-04-09   |      7 |
|         8 | Priya      | 456 Patel Colony, Mumbai    | Rajiv        | 123 Gandhi Nagar, Delhi       |   2.90 | In Transit  | 8901234567     | 2024-04-16   |      8 |
|         9 | Deepak     | 147 Verma Colony, Bangalore | Anita        | 789 Malhotra Street, Kolkata  |   1.60 | Delivered   | 9012345678     | 2024-04-08   |      9 |
|        10 | Nandini    | 654 Verma Lane, Lucknow     | Vikram       | 654 Verma Lane, Lucknow       |   3.10 | In Transit  | 0123456789     | 2024-04-16   |     10 |
|        11 | Anita      |  789 Malhotra Street, Kolkata | Vikram     | 654 Verma Lane, Lucknow       |   5.34 | In Transist | 6738573930     | 2024-04-19   |     11 |
|        12 | Preeti     | 741 Khan Street, Hyderabad  | Rajiv        | 123 Gandhi Nagar, Delhi       |   5.34 | Delivered   | 8738573930     | 2024-04-10   |     12 |
+-----------+------------+-----------------------------+--------------+-------------------------------+--------+-------------+----------------+--------------+--------+
12 rows in set (0.00 sec)
```

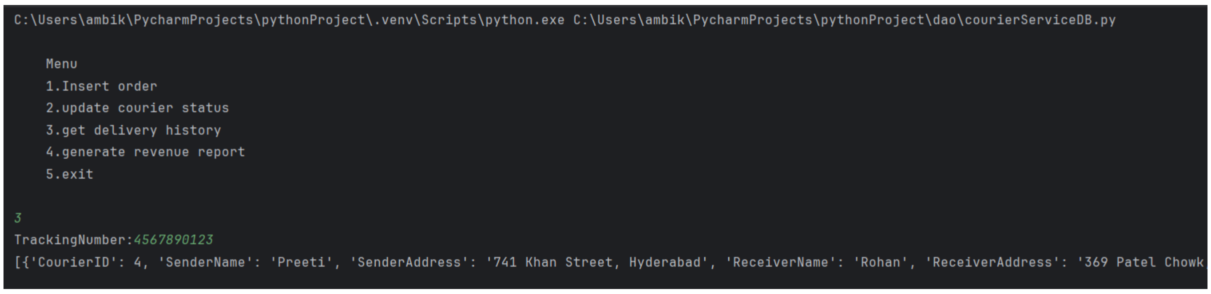
After inserting



```
mysql> select * from courier;
+-----------+------------+-----------------------------+--------------+-------------------------------+--------+-------------+----------------+--------------+--------+
| CourierID | SenderName | SenderAddress               | ReceiverName | ReceiverAddress               | Weight | Status      | TrackingNumber | DeliveryDate | userId |
+-----------+------------+-----------------------------+--------------+-------------------------------+--------+-------------+----------------+--------------+--------+
|         1 | Rajiv      | 123 Gandhi Nagar, Delhi     | Priya        | 456 Patel Colony, Mumbai      |   2.50 | Delivered   | 1234567890     | 2024-04-10   |      1 |
|         2 | Anita      | 789 Malhotra Street, Kolkata | Deepak      | 321 Rajput Enclave, Jaipur    |   1.75 | Delivered   | 2345678901     | 2024-04-09   |      2 |
|         3 | Vikram     | 654 Verma Lane, Lucknow     | Nandini      | 987 Joshi Nagar, Pune         |   3.25 | In Transit  | 3456789012     | 2024-04-13   |      3 |
|         4 | Preeti     | 741 Khan Street, Hyderabad  | Rohan        | 369 Patel Chowk, Ahmedabad    |   4.00 | Delivered   | 4567890123     | 2024-04-09   |      4 |
|         5 | Anil       | 852 Khan Market, Chennai    | Ishita       | 147 Verma Colony, Bangalore   |   2.00 | In Transit  | 5678901234     | 2024-04-14   |      5 |
|         6 | Sneha      | 321 Rajput Enclave, Jaipur  | Meera        | 741 Khan Street, Hyderabad    |   3.75 | In Transit  | 6789012345     | 2024-04-15   |      6 |
|         7 | Vikas      | 987 Joshi Nagar, Pune       | Arjun        | 123 Gandhi Nagar, Delhi       |   5.25 | Delivered   | 7890123456     | 2024-04-09   |      7 |
|         8 | Priya      | 456 Patel Colony, Mumbai    | Rajiv        | 123 Gandhi Nagar, Delhi       |   2.90 | In Transit  | 8901234567     | 2024-04-16   |      8 |
|         9 | Deepak     | 147 Verma Colony, Bangalore | Anita        | 789 Malhotra Street, Kolkata  |   1.60 | Delivered   | 9012345678     | 2024-04-08   |      9 |
|        10 | Nandini    | 654 Verma Lane, Lucknow     | Vikram       | 654 Verma Lane, Lucknow       |   3.10 | In Transit  | 0123456789     | 2024-04-16   |     10 |
|        11 | Anita      |  789 Malhotra Street, Kolkata | Vikram     | 654 Verma Lane, Lucknow       |   5.34 | In Transist | 6738573930     | 2024-04-19   |     11 |
|        12 | Preeti     | 741 Khan Street, Hyderabad  | Rajiv        | 123 Gandhi Nagar, Delhi       |   5.34 | Delivered   | 8738573930     | 2024-04-10   |     12 |
|        13 | Vaishnavi  | 4/136 Kangeyam              | Ambika       | 34, Madurai                   |   2.00 | In transist | 1639745920     | 2024-05-08   |      3 |
+-----------+------------+-----------------------------+--------------+-------------------------------+--------+-------------+----------------+--------------+--------+
13 rows in set (0.00 sec)
```

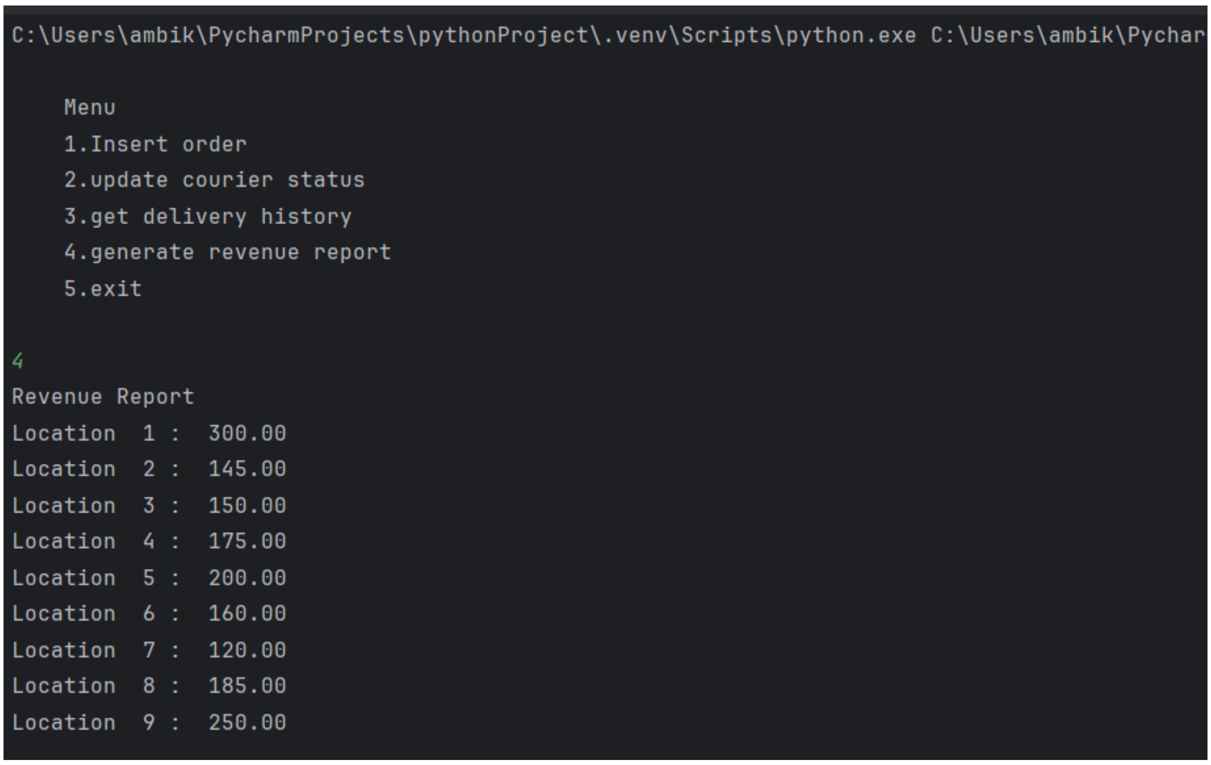
4. Implement a feature to retrieve and display the delivery history of a specific parcel by querying the database. 1. Generate and display reports using data retrieved from the database (e.g., **shipment status report, revenue report**).

```
def getDeliveryHistory(self, trackingNumber):
    cursor = CourierServiceDb.connection.cursor()
    delivery_history=[]

    select_query = """
       SELECT *
       FROM courier
       WHERE TrackingNumber = %s
       """

    cursor.execute(select_query, (trackingNumber,))
    rows = cursor.fetchall()

    for row in rows:
        delivery_history.append({
            "CourierID": row[0],
            "SenderName": row[1],
            "SenderAddress": row[2],
            "ReceiverName": row[3],
            "ReceiverAddress": row[4],
            "Weight": row[5],
            "Status": row[6],
            "TrackingNumber": row[7],
            "DeliveryDate": row[8],
            "userId": row[9]
        })

    print(delivery_history)
```



```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\ambik\PycharmProjects\pythonProject\dao\courierServiceDB.py

    Menu
    1.Insert order
    2.update courier status
    3.get delivery history
    4.generate revenue report
    5.exit

3
TrackingNumber:4567890123
[{'CourierID': 4, 'SenderName': 'Preeti', 'SenderAddress': '741 Khan Street, Hyderabad', 'ReceiverName': 'Rohan', 'ReceiverAddress': '369 Patel Chowk
```

```
def generateRevenueReport(self):

    cursor = CourierServiceDb.connection.cursor()

    select_query = """
    SELECT LocationId, SUM(Amount) AS TotalRevenue
    FROM payment
    GROUP BY LocationId
    """

    cursor.execute(select_query)
    rows = cursor.fetchall()

    print("Revenue Report")
    for row in rows:
        print("Location ",row[0],": ",row[1])
```



```
C:\Users\ambik\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\ambik\Pychar

    Menu
    1.Insert order
    2.update courier status
    3.get delivery history
    4.generate revenue report
    5.exit

4
Revenue Report
Location  1 :  300.00
Location  2 :  145.00
Location  3 :  150.00
Location  4 :  175.00
Location  5 :  200.00
Location  6 :  160.00
Location  7 :  120.00
Location  8 :  185.00
Location  9 :  250.00
```