# ASSIGNMENT 4

## 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.

```python
def check_delivery_status(order_status):
    if order_status == "Delivered":
        print("The order has been delivered.")
    elif order_status == "Processing":
        print("The order is still being processed.")
    elif order_status == "Cancelled":
        print("The order has been cancelled.")
    else:
        print("Invalid order status.")

1 usage
def main():
    order_status = input("Enter the order status: ")
    check_delivery_status(order_status)

if __name__=="__main__":
    main()
```

check_delivery_status() › elif order_status == "Cancelled"

Run ControlFlow_Task1

```
Enter the order status: Delivered
The order has been delivered.
```

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

```python
def categorize_parcel(weight):
    category = None

    if weight < 5:
        category = "Light"
    elif 5 <= weight < 10:
        category = "Medium"
    elif weight >= 10:
        category = "Heavy"
    else:
        category = "Invalid weight category."

    return f"The parcel is {category}."

1 usage
def main1():
    parcel_weight = float(input("Enter the parcel weight (in kg): "))
    print(categorize_parcel(parcel_weight))
```

main()

Run ControlFlow_Task1

```
Enter the parcel weight (in kg): 8
The parcel is Medium.
```

3. Implement User Authentication

Create a login system for employees and customers using control flow statements.

```python
def authenticate_user(username, password, user_type):
    employee_username = "employee123"
    employee_password = "pass123"
    customer_username = "customer123"
    customer_password = "pass123"

    if user_type == "employee":
        if username == employee_username and password == employee_password:
            return "Welcome, Employee!"
        else:
            return "Invalid credentials for employee. Please try again."
    elif user_type == "customer":
        if username == customer_username and password == customer_password:
            return "Welcome, Customer!"
        else:
            return "Invalid credentials for customer. Please try again."
    else:
        return "Invalid user type. Please enter 'employee' or 'customer'."
def main2():
    user_type = input("Enter user type (employee/customer): ")
    username = input("Enter username: ")
    password = input("Enter password: ")
    result = authenticate_user(username, password, user_type.lower())
    print(result)
```

Output-

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\ControlFlow_Task1.py
Enter user type (employee/customer): employee
Enter username: employee123
Enter password: pass123
Welcome, Employee!

Process finished with exit code 0
```

Implement Courier Assignment Logic 1. Develop a mechanism to assign couriers to shipments based on predefined criteria (e.g., proximity, load capacity) using loops.

```python
shipments = [
    {"destination": 7, "weight": 10},
    {"destination": 10, "weight": 5},
    {"destination": 5, "weight": 8},
]

couriers = [
    {"name": "Courier1", "proximity": 5, "max_capacity": 20, "current_load": 0},
    {"name": "Courier2", "proximity": 8, "max_capacity": 15, "current_load": 0},
]
for shipment in shipments:
    min_distance = float('inf')
    assigned_courier = None

    for courier in couriers:
        distance = abs(shipment["destination"] - courier["proximity"])

        if distance < min_distance and courier["current_load"] + shipment["weight"] <= courier["max_capacity"]:
            min_distance = distance
            assigned_courier = courier
    if assigned_courier:
        assigned_courier["current_load"] += shipment["weight"]
        print(f"Shipment to {shipment['destination']} kg assigned to courier {assigned_courier['name']}.")
    else:
        print(f"No available courier for shipment to {shipment['destination']} kg.")
```

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\ControlFlow_Task1.py
Shipment to 7 kg assigned to courier Courier2.
Shipment to 10 kg assigned to courier Courier2.
Shipment to 5 kg assigned to courier Courier1.

Process finished with exit code 0
```

## Task 2: Loops and Iteration

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

```python
customers = ["Customer1", "Customer2", "Customer3"]
orders_customer1 = ["Order1A", "Order1B", "Order1C"]
orders_customer2 = ["Order2A", "Order2B"]
orders_customer3 = ["Order3A", "Order3B", "Order3C", "Order3D"]

1 usage
def display_orders(customer_name):
    orders = globals().get(f"orders_{customer_name.lower()}")
    if orders:
        print(f"Orders for {customer_name}:")
        for order in orders:
            print(order)
    else:
        print("No orders available for the customer.")
1 usage
def main():
    customer_name = input("Enter customer name: ")

    if customer_name in customers:
        display_orders(customer_name)
    else:
        print("Customer not found.")

if __name__ == "__main__":
    main()
```

## Output

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\Loops&Itertaion_Task2.
Enter customer name: Customer1
Orders for Customer1:
Order1A
Order1B
Order1C

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.

```python
def track_courier(courier_name, current_location, destination):
    print(f"{courier_name} is on the way to {destination}.")

    while current_location != destination:
        current_location += 1
        time.sleep(random.uniform(a: 0.5, b: 2.0))
        print(f"{courier_name} is now at location {current_location}.")

    print(f"{courier_name} has reached the destination {destination}.")

def main():
    courier_name = input("Enter courier name: ")
    current_location = int(input("Enter current location: "))
    destination = int(input("Enter destination location: "))

    track_courier(courier_name, current_location, destination)

if __name__ == "__main__":
    main()
```

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\Loops&Itertaion_Task2.
Enter courier name: Delhivery
Enter current location: 1
Enter destination location: 5
Delhivery  is on the way to 5.
Delhivery  is now at location 2.
Delhivery  is now at location 3.
Delhivery  is now at location 4.
Delhivery  is now at location 5.
Delhivery  has reached the destination 5.

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

```python
Arrays&DataStructures.py

import time
import random

class Parcel:
    def __init__(self, parcel_id):
        self.parcel_id = parcel_id
        self.tracking_history = []

    def add_location_update(self, location):
        timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
        self.tracking_history.append({"timestamp": timestamp, "location": location})

def track_parcel(parcel, destination):
    print(f"Parcel {parcel.parcel_id} is on the way to {destination}.")

    while parcel.tracking_history[-1]["location"] != destination:
        new_location = parcel.tracking_history[-1]["location"] + 1
        parcel.add_location_update(new_location)
        time.sleep(random.uniform(0.5, 2.0))
        print(f"Parcel {parcel.parcel_id} is now at location {new_location}.")

    print(f"Parcel {parcel.parcel_id} has reached the destination {destination}.")

def main():
```

OUTPUT:-

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\Arrays&DataStructures.
Enter parcel ID: 101
Enter starting location: 1
Enter destination location: 4
Parcel 101 is on the way to 4.
Parcel 101 is now at location 2.
Parcel 101 is now at location 3.
Parcel 101 is now at location 4.
Parcel 101 has reached the destination 4.

Tracking History for Parcel 101:
2023-12-27 11:32:28 - Location: 1
2023-12-27 11:32:28 - Location: 2
2023-12-27 11:32:28 - Location: 3
2023-12-27 11:32:30 - Location: 4

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.

```python
class Courier:
    def __init__(self, name, proximity):
        self.name = name
        self.proximity = proximity

def find_nearest_courier(order_location, couriers):
    min_distance = sys.maxsize
    nearest_courier = None

    for courier in couriers:
        distance = abs(order_location - courier.proximity)
        if distance < min_distance :
            min_distance = distance
            nearest_courier = courier

    return nearest_courier

def main():
    couriers = [
        Courier( name: "Courier1",  proximity: 5 ),
        Courier( name: "Courier2",  proximity: 8 ),
        Courier( name: "Courier3",  proximity: 12)
    ]
```

**OUTPUT**

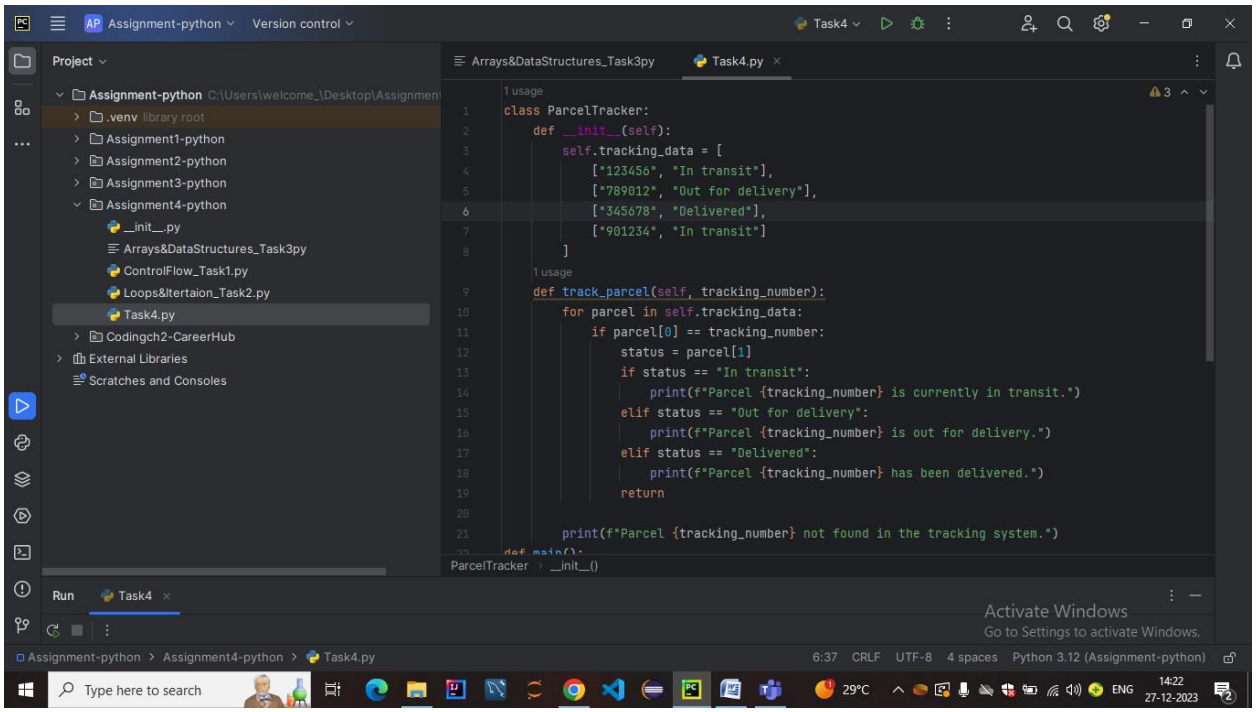
```python
def main():
    couriers = [
        Courier( name: "Courier1",  proximity: 5 ),
        Courier( name: "Courier2",  proximity: 8 ),
        Courier( name: "Courier3",  proximity: 12)
    ]

    order_location = int(input("Enter order location: "))

    nearest_courier = find_nearest_courier(order_location, couriers)

    if nearest_courier:
        print(f"The nearest available courier is {nearest_courier.name}.")
    else:
        print("No available courier for the order.")

if __name__ == "__main__":
    main()
```

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\Arrays&DataStructures.
Enter order location: 14
The nearest available courier is Courier3.

Process finished with exit code 0
```

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

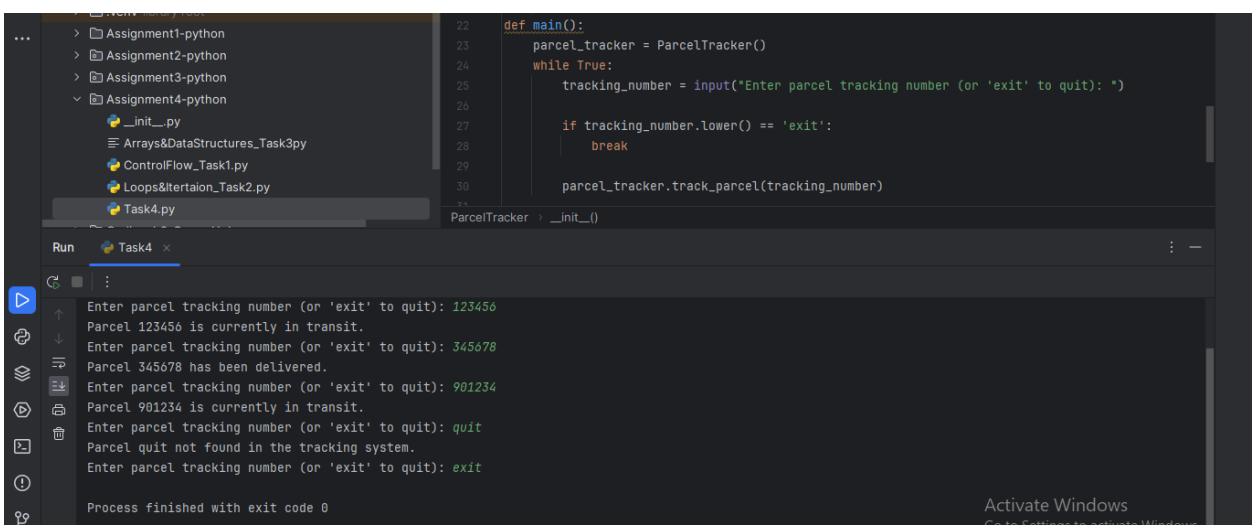
***9.Parcel Tracking:***

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.



```python
class ParcelTracker:
    def __init__(self):
        self.tracking_data = [
            ["123456", "In transit"],
            ["789012", "Out for delivery"],
            ["345678", "Delivered"],
            ["901234", "In transit"]
        ]
    def track_parcel(self, tracking_number):
        for parcel in self.tracking_data:
            if parcel[0] == tracking_number:
                status = parcel[1]
                if status == "In transit":
                    print(f"Parcel {tracking_number} is currently in transit.")
                elif status == "Out for delivery":
                    print(f"Parcel {tracking_number} is out for delivery.")
                elif status == "Delivered":
                    print(f"Parcel {tracking_number} has been delivered.")
                return

        print(f"Parcel {tracking_number} not found in the tracking system.")
```

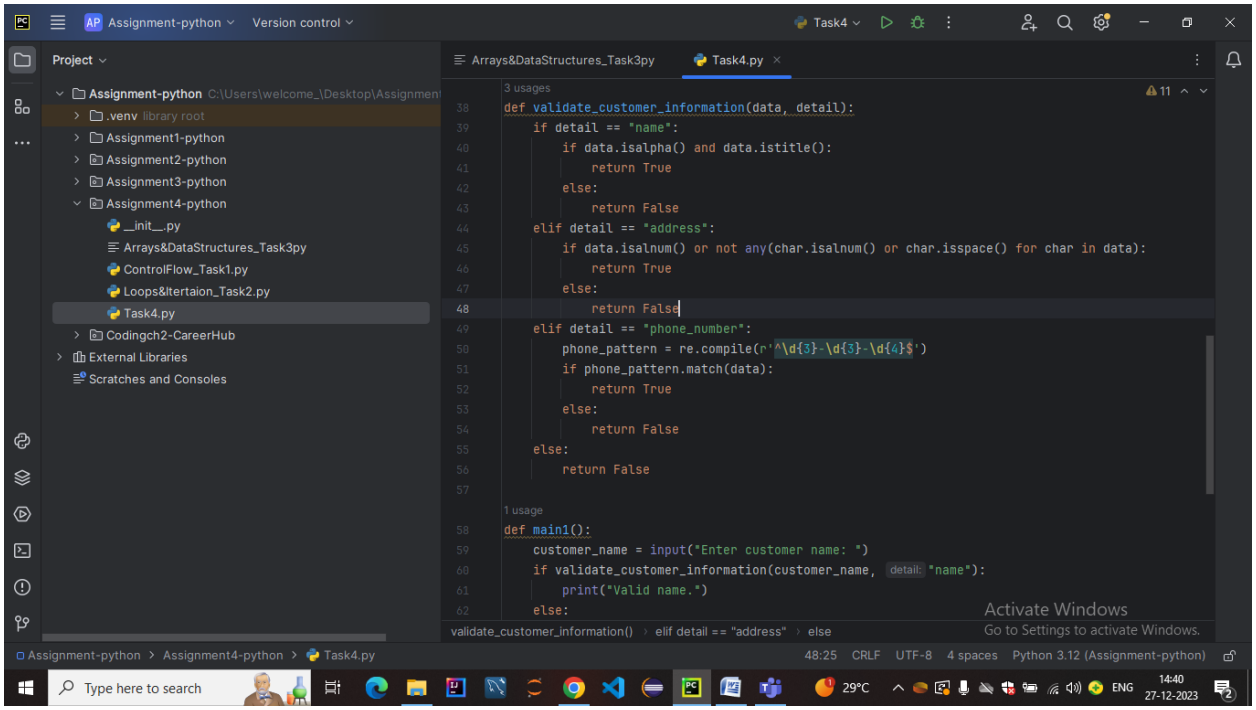
**OUTPUT:-**



```python
def main():
    parcel_tracker = ParcelTracker()
    while True:
        tracking_number = input("Enter parcel tracking number (or 'exit' to quit): ")

        if tracking_number.lower() == 'exit':
            break

        parcel_tracker.track_parcel(tracking_number)
```

```
Enter parcel tracking number (or 'exit' to quit): 123456
Parcel 123456 is currently in transit.
Enter parcel tracking number (or 'exit' to quit): 345678
Parcel 345678 has been delivered.
Enter parcel tracking number (or 'exit' to quit): 901234
Parcel 901234 is currently in transit.
Enter parcel tracking number (or 'exit' to quit): quit
Parcel quit not found in the tracking system.
Enter parcel tracking number (or 'exit' to quit): exit

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., ###-###-####).*



```python
def validate_customer_information(data, detail):
    if detail == "name":
        if data.isalpha() and data.istitle():
            return True
        else:
            return False
    elif detail == "address":
        if data.isalnum() or not any(char.isalnum() or char.isspace() for char in data):
            return True
        else:
            return False
    elif detail == "phone_number":
        phone_pattern = re.compile(r'^\d{3}-\d{3}-\d{4}$')
        if phone_pattern.match(data):
            return True
        else:
            return False
    else:
        return False

def main1():
    customer_name = input("Enter customer name: ")
    if validate_customer_information(customer_name, detail: "name"):
        print("Valid name.")
    else:
```

***OUTPUT***

```python
def main1():
    customer_name = input("Enter customer name: ")
    if validate_customer_information(customer_name, detail: "name"):
        print("Valid name.")
    else:
        print("Invalid name.")

    customer_address = input("Enter customer address: ")
    if validate_customer_information(customer_address, detail: "address"):
        print("Valid address.")
    else:
        print("Invalid address.")

    customer_phone_number = input("Enter customer phone number (format: ###-###-####): ")
    if validate_customer_information(customer_phone_number, detail: "phone_number"):
```

```
Enter customer name: Aniket
Valid name.
Enter customer address: Pune
Valid address.
Enter customer phone number (format: ###-###-####): 702090539
Invalid phone number.
```

*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*

```python
    pass
def format_address(street, city, state, zip_code):

    formatted_street = ' '.join(word.capitalize() for word in street.split())
    formatted_city = city.title()
    formatted_state = state.capitalize()
    formatted_zip_code = str(zip_code).zfill(5)
    formatted_address = f"{formatted_street}, {formatted_city}, {formatted_state} {formatted_zip_code}"

    return formatted_address
def main2():
    print("Enter the address")
    street_input = input("Enter street: ")
    city_input = input("Enter city: ")
```

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\Task4.py
Enter the address
Enter street: main road
Enter city: Laturrr city
Enter state: maharashtra
Enter zip code: 413522
Formatted Address: Main Road, Laturrr City, Maharashtra 413522
```

*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*

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\Task4.py
Enter the Customer name Aniket
Enter the order number 123
Enter the address Pune
Enter the Expected Delievery date 2023-12-31
Generated Order Confirmation Email:

Dear  Aniket,

Thank you for your order! Your order number is 123.
We will deliver your items to the following address:
 Pune
Expected delivery date:  2023-12-31

Thank you for choosing our services. If you have any questions, feel free to contact us.

Best regards,
The Delhivery Company
```

*OUTPUT*

```python
#TASK #4.12
def generate_order_confirmation(customer_name, order_number, delivery_address, expected_delivery_date):
    email_content = f"Dear {customer_name},\n\n"
    email_content += f"Thank you for your order! Your order number is {order_number}.\n"
    email_content += f"We will deliver your items to the following address:\n{delivery_address}\n"
    email_content += f"Expected delivery date: {expected_delivery_date}\n\n"
    email_content += "Thank you for choosing our services. If you have any questions, feel free to contact us.\n\n"
    email_content += "Best regards,\nThe Delhivery Company"

    return email_content
def main3():
    customer_name = input("Enter the Customer name")
    order_number = int(input("Enter the order number"))
    delivery_address = input("Enter the address")
    expected_delivery_date = input("Enter the Expected Delievery date")

    confirmation_email = generate_order_confirmation(customer_name, order_number, delivery_address, expected_delivery_date)

    print("Generated Order Confirmation Email:\n")
    print(confirmation_email)
if __name__ == "__main__":
    main3()
```

*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.*

```python
#TASK 4.13
def calculate_shipping_cost(source_address,destination_address,parcel_weight):
    rate_per_km=5

    if(parcel_weight<=5):
        rate_Of_weight=10
    else:
        rate_Of_weight=20
    distance=destination_address-source_address
    rate=distance*rate_per_km+rate_Of_weight
    return rate


def main4():
    source_address =float(input("Enter the source address location point in number: "))
    destination_address = float(input("Enter the destination location Point: "))
    parcel_weight = float(input("Enter the parcel weight in pounds: "))
    shipping_cost = calculate_shipping_cost(source_address,destination_address,parcel_weight)
    print(f"Shipping cost: Rs{shipping_cost:.2f}")

if __name__ == "__main__":
    main4()
```

*OUTPUT:-*

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\Task4.py
Enter the source address location point in number: 5
Enter the destination location Point: 19
Enter the parcel weight in pounds: 8
Shipping cost: Rs90.00

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.*

```python
#TASK 4.14
import random
import string

1 usage
def generate_secure_password(length=12):
    uppercase_letters = string.ascii_uppercase
    lowercase_letters = string.ascii_lowercase
    digits = string.digits
    special_characters = string.punctuation

    all_characters = uppercase_letters + lowercase_letters + digits + special_characters

    password = (
        random.choice(uppercase_letters)
        + random.choice(lowercase_letters)
        + random.choice(digits)
        + random.choice(special_characters)
    )
    for _ in range(length - 4):
        password += random.choice(all_characters)
    password_list = list(password)
    random.shuffle(password_list)
    password = ''.join(password_list)

    return password
```

*OUTPUT.*

```python
def main5():
    secure_password = generate_secure_password()
    print("Generated Password:", secure_password)
if __name__ == "__main__":
    main5()
```

```
generate_secure_password()
Run    Task4
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\Task4.py
Generated Password: TX{vLeOT#>1@
```

*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.*

```python
1 usage
def find_similar_addresses(target_address, addresses_list):
    similar_addresses = []
    target_address_lower = target_address.lower()
    for address in addresses_list:
        address_lower = address.lower()

        if target_address_lower in address_lower or address_lower in target_address_lower:
            similar_addresses.append(address)

    return similar_addresses
1 usage
def main5():
    target_address = "123 Main St, Cityville, USA"
    addresses_list = [
        "123 Main St, Cityville, USA",
        "124 Main St, Cityville, USA",
        "456 Broad St, Townsville, USA",
        "789 Oak St, Villagetown, USA",
    ]

    similar_addresses = find_similar_addresses(target_address, addresses_list)
    print(f"Target Address: {target_address}")
    print("Similar Addresses:")
    for similar_address in similar_addresses:
        print(similar_address)
```

***OUTPUT:-***

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\Task4.py
Target Address: 123 Main St, Cityville, USA
Similar Addresses:
123 Main St, Cityville, USA

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

```python
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

    1 usage
    @property
    def UserID(self):
        return self._UserID

    @UserID.setter
    def UserID(self, new_UserID):
        if not isinstance(new_UserID, int) or new_UserID <= 0:
            raise ValueError("User ID must be a positive integer.")
        self._UserID = new_UserID

    1 usage
    @property
    def UserName(self):
        return self._UserName

    @UserName.setter
    def user_name(self, new_UserName):
```

2. Courier Class Variables: courierID , senderName , senderAddress , receiverName , receiverAddress , weight , status, trackingNumber , deliveryDate ,userId

```python
class Courier:
    def __init__(self, CourierID, SenderName, SenderAddress, ReceiverName, ReceiverAddress,
                 Weight, 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

    1 usage
    @property
    def CourierID(self):
        return self._CourierID

    @CourierID.setter
    def CourierID(self, new_CourierID):
        if not isinstance(new_CourierID, int) or new_CourierID <= 0:
            raise ValueError("Courier ID must be a positive integer.")
        self._CourierID = new_CourierID

    1 usage
    @property
```

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

```python
class Employee:
    def __init__(self, EmployeeID, EmployeeName, Email, ContactNumber, Role, Salary):
        self._EmployeeID = EmployeeID
        self._EmployeeName = EmployeeName
        self._Email = Email
        self._ContactNumber = ContactNumber
        self._Role = Role
        self._Salary = Salary

    1 usage
    @property
    def EmployeeID(self):
        return self._EmployeeID

    @EmployeeID.setter
    def EmployeeID(self, new_EmployeeID):
        if not isinstance(new_EmployeeID, int) or new_EmployeeID <= 0:
            raise ValueError("Employee ID must be a positive integer.")
        self._EmployeeID = new_EmployeeID

    1 usage
    @property
    def EmployeeName(self):
        return self._EmployeeName

    @EmployeeName.setter
    def EmployeeName(self, new_EmployeeName):
```

Argument name should be lowercase
Ignore errors like this Alt+Shift+Enter More actions... Alt+Enter
Parameter new_EmployeeName of Employee.EmployeeName
new_EmployeeName: Any

4. Location Class Variables LocationID , LocationName , Address

```python
class Location:
    def __init__(self, LocationID, LocationName, Address):
        self._LocationID = LocationID
        self._LocationName = LocationName
        self._Address = Address

    1 usage
    @property
    def LocationID(self):
        return self._LocationID

    @LocationID.setter
    def LocationID(self, new_LocationID):
        if not isinstance(new_LocationID, int) or new_LocationID <= 0:
            raise ValueError("Location ID must be a positive integer.")
        self._LocationID = new_LocationID

    1 usage
    @property
    def LocationName(self):
        return self._LocationName

    @LocationName.setter
    def LocationName(self, new_LocationName):
        if not new_LocationName or not isinstance(new_LocationName, str):
            raise ValueError("Location name must be a non-empty string.")
        self._LocationName = new_LocationName
```

5. CourierCompany Class Variables companyName , courierDetails -collection of Courier Objects, employeeDetailscollection of Employee Objects, locationDetails - collection of Location Objects.

```python
class CourierCompany:
    def __init__(self, CompanyName, CourierDetails=None, EmployeeDetails=None, LocationDetails=None):
        self._CompanyName = CompanyName
        self._CourierDetails = CourierDetails or []
        self._EmployeeDetails = EmployeeDetails or []
        self._LocationDetails = LocationDetails or []

    1 usage
    @property
    def CompanyName(self):
        return self._CompanyName

    @CompanyName.setter
    def CompanyName(self, new_CompanyName):
        if not new_CompanyName or not isinstance(new_CompanyName, str):
            raise ValueError("Company name must be a non-empty string.")
        self._CompanyName = new_CompanyName

    1 usage
    @property
    def CourierDetails(self):
        return self._CourierDetails

    @CourierDetails.setter
    def CourierDetails(self, new_CourierDetails):
        # Add additional validation for CourierDetails if needed
        self._CourierDetails = new_CourierDetails
```

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

```python
class Payment:
    def __init__(self, PaymentID, CourierID, Amount, PaymentDate):
        self._PaymentID = PaymentID
        self._CourierID = CourierID
        self._Amount = Amount
        self._PaymentDate = PaymentDate

    1 usage
    @property
    def PaymentID(self):
        return self._PaymentID

    @PaymentID.setter
    def PaymentID(self, new_PaymentID):
        if not isinstance(new_PaymentID, int) or new_PaymentID <= 0:
            raise ValueError("Payment ID must be a positive integer.")
        self._PaymentID = new_PaymentID

    1 usage
    @property
    def CourierID(self):
        return self._CourierID

    @CourierID.setter
    def CourierID(self, new_CourierID):
        if not isinstance(new_CourierID, int) or new_CourierID <= 0:
            raise ValueError("Courier ID must be a positive integer.")
```

# *Task 6: Service Provider Interface /Abstract class*

## ICourierUserService

```python
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
```

***ICourierAdminService***

```python
from abc import ABC, abstractmethod

class ICourierAdminService(ABC):
    @abstractmethod
    def assignOrder(self, courierOrder, courierStaffId):
        pass

    @abstractmethod
    def updateOrderStatus(self, trackingNumber, newStatus):
        pass

    @abstractmethod
    def viewOrders(self):
        pass
```

## *Task 7: Exception Handling*

*(*Scope: User Defined Exception/Checked /Unchecked Exception/Exception handling using try..catch finally,thow & throws keyword usage) 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 account

2. InvalidEmployeeIdException throw this exception when id entered for the employee not existing in the system

```python
class TrackingNumberNotFoundException(Exception):
    def __init__(self, message="TrackingNumber Not Found."):
        self.message = message
        super().__init__(self.message)

class InvalidEmployeeIdException(Exception):
    def __init__(self, message="Invalid EmployeeId."):
        self.message = message
        super().__init__(self.message)
```

**Task 8: Collections Scope: ArrayList/**Hashmap Task:

1. Create a new model named CourierCompanyCollection in entity package replacing the Array of Objects with List to accommodate dynamic updates in the CourierCompany class

```python
class CourierCompanyCollection:
    def __init__(self, companyName):
        self.companyName = companyName
        self.courierDetails = []
        self.employeeDetails = []
        self.locationDetails = []


    3 usages (3 dynamic)
    def get_courier_details(self):
        return self.courierDetails

    def get_employee_details(self):
        return self.employeeDetails

    def get_location_details(self):
        return self.locationDetails
```

2. Create a new implementation class CourierUserServiceCollectionImpl class in package dao which implements ICourierUserService interface which holds a variable named companyObj of type CourierCompanyCollection

```python
class CourierUserServiceCollectionImpl:
    def __init__(self, company_obj):
        self.company_obj = company_obj

    def place_order(self, courier_obj):
        self.company_obj.get_courier_details().append(courier_obj)


    def get_order_status(self, tracking_number):
        for courier in self.company_obj.get_courier_details():
            if courier.get_tracking_number() == tracking_number:
                return courier.get_status()
        return "Tracking number not found"

    def cancel_order(self, tracking_number):
        courier_details = self.company_obj.get_courier_details()
        for courier in courier_details:
            if courier.get_tracking_number() == tracking_number:
                courier_details.remove(courier)
                return True
        return False

    def get_assigned_order(self, courier_staff_id):
        return None
```



## *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.

```python
class CourierUserServiceImpl(CourierUserServiceCollectionImpl):
    def __init__(self, company_obj):
        super().__init__(company_obj)

    def place_order(self, courier_obj):
        super().place_order(courier_obj)
        print(f"Order placed successfully. Tracking Number: {courier_obj.get_tracking_number()}")

    2 usages
    def get_order_status(self, tracking_number):
        status = super().get_order_status(tracking_number)
        if status != "Tracking number not found":
            print(f"Order status for tracking number {tracking_number}: {status}")
        else:
            print(f"Order with tracking number {tracking_number} not found.")
        return status

    def cancel_order(self, tracking_number):
        result = super().cancel_order(tracking_number)
        if result:
            print(f"Order with tracking number {tracking_number} canceled successfully.")
        else:
            print(f"Order with tracking number {tracking_number} not found.")
        return result

    1 usage
    def get_assigned_order(self, courier_staff_id):
```

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

```python
    2 usages
class CourierAdminServiceImpl(CourierUserServiceImpl):
    def __init__(self, company_obj):
        super().__init__(company_obj)

    1 usage
    def assign_order(self, courier_order, courier_staff_id):

        assigned_orders = super().get_assigned_order(courier_staff_id)
        if not assigned_orders:
            assigned_orders = []
        assigned_orders.append(courier_order)
        print(f"Order assigned to courier staff ID {courier_staff_id}: {courier_order}")

        return assigned_orders

    2 usages (1 dynamic)
    def update_order_status(self, tracking_number, new_status):
        status = super().get_order_status(tracking_number)
        if status != "Tracking number not found":
            print(f"Updating order status for tracking number {tracking_number} to {new_status}")
        else:
            print(f"Order with tracking number {tracking_number} not found.")

    def view_orders(self):
        orders = super().get_all_orders()
        if orders:
```

**4.** Create CourierAdminServiceCollectionImpl class which inherits from CourierUserServiceColectionImpl and implements ICourierAdminService interface.

```python
class CourierAdminServiceCollectionImpl(CourierAdminServiceImpl):
    def __init__(self, company_obj):
        super().__init__(company_obj)

    def assign_order(self, courier_order, courier_staff_id):
        assigned_orders = super().assign_order(courier_order, courier_staff_id)
        if assigned_orders:
            print(f"Order assigned to courier staff ID {courier_staff_id}: {courier_order}")
        else:
            print(f"Failed to assign order to courier staff ID {courier_staff_id}.")

        return assigned_orders

    1 usage (1 dynamic)
    def update_order_status(self, tracking_number, new_status):
        super().update_order_status(tracking_number, new_status)
        status = super().get_order_status(tracking_number)
        if status == new_status:
            print(f"Order status updated successfully. New status: {new_status}")
        else:
            print(f"Failed to update order status for tracking number {tracking_number}.")

    def view_orders(self):
        orders = super().get_all_orders()
        if orders:
            print(f"All courier orders: {orders}")
        else:
```

## *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

```
[DatabaseConfig]
db_url=localhost
db_port=3306
db_name=couriermanagementsystem
db_username=root
db_password=Aniket@123
```

```python
class DBConnection:
    connection = None

    @staticmethod
    def get_connection():
        if DBConnection.connection is None:
            config = ConfigParser()
            config.read('config.properties')

            db_url = config.get( section: 'DatabaseConfig',  option: 'db_url')
            db_username = config.get( section: 'DatabaseConfig',  option: 'db_username')
            db_password = config.get( section: 'DatabaseConfig',  option: 'db_password')

            try:
                DBConnection.connection = mysql.connector.connect(
                    host=db_url,
                    user=db_username,
                    password=db_password
                )
                if DBConnection.connection.is_connected():
                    print("Connected to the database.")
            except Error as e:
                print(f"Error: {e}")
        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.

```python
from DBConnection import DBConnection

2 usages
class CourierServiceDb:
    connection = None

    def __init__(self):
        if CourierServiceDb.connection is None:
            CourierServiceDb.connection = DBConnection.get_connection()
```

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

```python
    def insert_order(self, courier_order):
        try:
            cursor = CourierServiceDb.connection.cursor()
            query = "INSERT INTO courier (tracking_number, status) VALUES (%s, %s)"
            data = (courier_order.tracking_number, courier_order.status)
            cursor.execute(query, data)
            CourierServiceDb.connection.commit()
            print("Order inserted successfully.")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    def update_courier_status(self, tracking_number, new_status):
        try:
            cursor = CourierServiceDb.connection.cursor()
            query = "UPDATE courier_orders SET status = %s WHERE tracking_number = %s"
            data = (new_status, tracking_number)
            cursor.execute(query, data)
            CourierServiceDb.connection.commit()
            print("Courier status updated successfully.")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
```

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).

```python
    def retrieve_orders(self):
        try:
            cursor = CourierServiceDb.connection.cursor()
            query = "SELECT * FROM courier_orders"
            cursor.execute(query)
            orders = cursor.fetchall()
            return orders
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()

    1 usage
    def retrieve_delivery_history(self, tracking_number):
        try:
            cursor = CourierServiceDb.connection.cursor()
            query = "SELECT * FROM courier WHERE trackingnumber = %s"
            data = (tracking_number,)
            cursor.execute(query, data)
            delivery_history = cursor.fetchall()
            return delivery_history
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
```

*OUTPUT*

```python
tracking_number = "TN111111"


delivery_history = CourierServiceDb().retrieve_delivery_history(tracking_number)
print(f"Delivery History for Tracking Number {tracking_number}: {delivery_history}")

'''
shipment_status_report = CourierServiceDb().generate_shipment_status_report()
```

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\main.py
Connected to the database.
Delivery History for Tracking Number TN111111: [(101, 'Amit Patel', 'Sender Address 11', 'Riya Gupta', 'Receiver Address 11', Decimal('2.30'), 'In Transit

Process finished with exit code 0
```

```python
shipment_status_report = CourierServiceDb().generate_shipment_status_report()
print("Shipment Status Report:")
for order in shipment_status_report:
    print(f"Tracking Number: {order['trackingnumber']}, Status: {order['status']}")
```

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\main.py
Connected to the database.
Shipment Status Report:
Tracking Number: TN111111, Status: In Transit
Tracking Number: TN222222, Status: Delivered
Tracking Number: TN333333, Status: Pending
Tracking Number: TN444444, Status: Delivered
Tracking Number: TN555555, Status: Pending
Tracking Number: TN666666, Status: In Transit
Tracking Number: TN777777, Status: Delivered
Tracking Number: TN888888, Status: Pending
Tracking Number: TN999999, Status: In Transit
Tracking Number: TN101010, Status: Delivered

Process finished with exit code 0
```

```python
total_revenue = CourierServiceDb().generate_revenue_report()
print(f"Total Revenue: {total_revenue}")
```

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\Assignment4-python\main.py
Connected to the database.
Total Revenue: 1065.00

Process finished with exit code 0
```