

# Assignment 4

## Courier Management System

### 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.

```
Assignment 4 > 1.py > ...
1  def check_order_status(order_status):
2      if order_status == 'Delivered':
3          print("The order has been delivered.")
4      elif order_status == 'Processing':
5          print("The order is still being processed.")
6      elif order_status == 'Cancelled':
7          print("The order has been cancelled.")
8      else:
9          print("Invalid order status.")
10
11 # Example usage:
12 order_status = 'Delivered'
13 check_order_status(order_status)
14
```

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

```
Assignment 4 > 2.py > categorize_parcel
1  def categorize_parcel(weight):
2      categories = {
3          'Light': weight <= 2.0,
4          'Medium': 2.0 < weight <= 5.0,
5          'Heavy': weight > 5.0
6      }
7
8      for category, condition in categories.items():
9          if condition:
10             return category
11
12 # Example usage:
13 parcel_weight = 4.8
14 parcel_category = categorize_parcel(parcel_weight)
15 print(f"The parcel is categorized as: {parcel_category}")
16
```

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

```
Assignment 4 > Python > ...
1 user_data = {
2     'employees': {'admin': 'admin123', 'john.doe': 'password123', 'jane.smith': 'password456'},
3     'customers': {'john.customer': 'customer123', 'jane.customer': 'customer456', 'bob.customer': 'customer789'}
4 }
5
6 def authenticate_user(user_type, username, password):
7     # Check if the user_type is valid ('employees' or 'customers')
8     if user_type not in user_data:
9         print("Invalid user type.")
10        return False
11
12    # Check if the username exists in the specified user_type
13    if username not in user_data[user_type]:
14        print("Invalid username.")
15        return False
16
17    # Check if the provided password matches the stored password for the given username
18    if user_data[user_type][username] == password:
19        print(f"Welcome, {user_type[:-1].capitalize()} {username}!")
20        return True
21    else:
22        print("Incorrect password.")
23        return False
24
25 # Example usage:
26 user_type_input = input("Enter user type (employees/customers): ").lower()
27 username_input = input("Enter username: ")
28 password_input = input("Enter password: ")
29
30 # Perform user authentication
31 authentication_result = authenticate_user(user_type_input, username_input, password_input)
32
```

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

```
Assignment 4 > Python > ...
1 couriers = ['Courier1', 'Courier2', 'Courier3', 'Courier4']
2 shipments = ['Shipment1', 'Shipment2', 'Shipment3', 'Shipment4']
3
4 for shipment in shipments:
5     assigned_courier = couriers.pop(0)
6     print(f"{shipment} assigned to {assigned_courier}")
7
8
```

## Task 2: Loops and Iteration

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

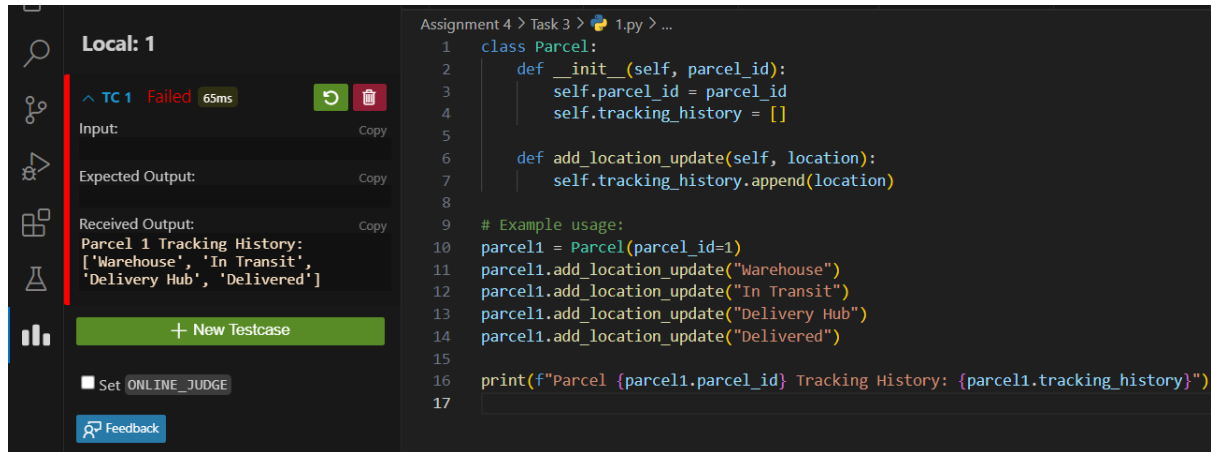
```
Assignment 4 > Task 2 > 147 > ...
1  orders = [
2      {'OrderID': 1, 'UserID': 1, 'Status': 'Delivered'},
3      {'OrderID': 2, 'UserID': 2, 'Status': 'Processing'},
4      {'OrderID': 3, 'UserID': 1, 'Status': 'In Transit'},
5      {'OrderID': 4, 'UserID': 3, 'Status': 'Pending'},
6      {'OrderID': 5, 'UserID': 2, 'Status': 'Delivered'},
7      {'OrderID': 6, 'UserID': 3, 'Status': 'Pickup'},
8      {'OrderID': 7, 'UserID': 1, 'Status': 'In Transit'},
9      {'OrderID': 8, 'UserID': 2, 'Status': 'Pending'},
10     {'OrderID': 9, 'UserID': 3, 'Status': 'Delivered'},
11     {'OrderID': 10, 'UserID': 1, 'Status': 'Pickup'}
12 ]
13
14 def display_orders_for_customer(customer_id):
15     print(f"Orders for Customer {customer_id}:")
16     for order in orders:
17         if order['UserID'] == customer_id:
18             print(f"Order ID: {order['OrderID']}, Status: {order['Status']}")
19
20 # Example usage:
21 customer_id_input = int(input("Enter customer ID: "))
22 display_orders_for_customer(customer_id_input)
```

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

```
1  import random
2  import time
3
4  def track_courier(courier_id):
5      print(f"Tracking Courier {courier_id}:")
6
7      while True:
8          current_location = random.choice(['LocationA', 'LocationB', 'LocationC', 'LocationD'])
9          print(f"Current Location: {current_location}")
10
11         if current_location == 'Destination':
12             print("Courier has reached the destination.")
13             break
14
15         time.sleep(2)
16
17 # Example usage:
18 courier_id_input = input("Enter courier ID: ")
19 track_courier(courier_id_input)
```

### 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.



```
Assignment 4 > Task 3 > 1.py > ...
1 class Parcel:
2     def __init__(self, parcel_id):
3         self.parcel_id = parcel_id
4         self.tracking_history = []
5
6     def add_location_update(self, location):
7         self.tracking_history.append(location)
8
9 # Example usage:
10 parcel1 = Parcel(parcel_id=1)
11 parcel1.add_location_update("Warehouse")
12 parcel1.add_location_update("In Transit")
13 parcel1.add_location_update("Delivery Hub")
14 parcel1.add_location_update("Delivered")
15
16 print(f"Parcel {parcel1.parcel_id} Tracking History: {parcel1.tracking_history}")
17
```

Local: 1

TC 1 Failed 65ms

Input: Copy

Expected Output: Copy

Received Output: Copy

Parcel 1 Tracking History:  
['Warehouse', 'In Transit', 'Delivery Hub', 'Delivered']

+ New Testcase

Set ONLINE JUDGE

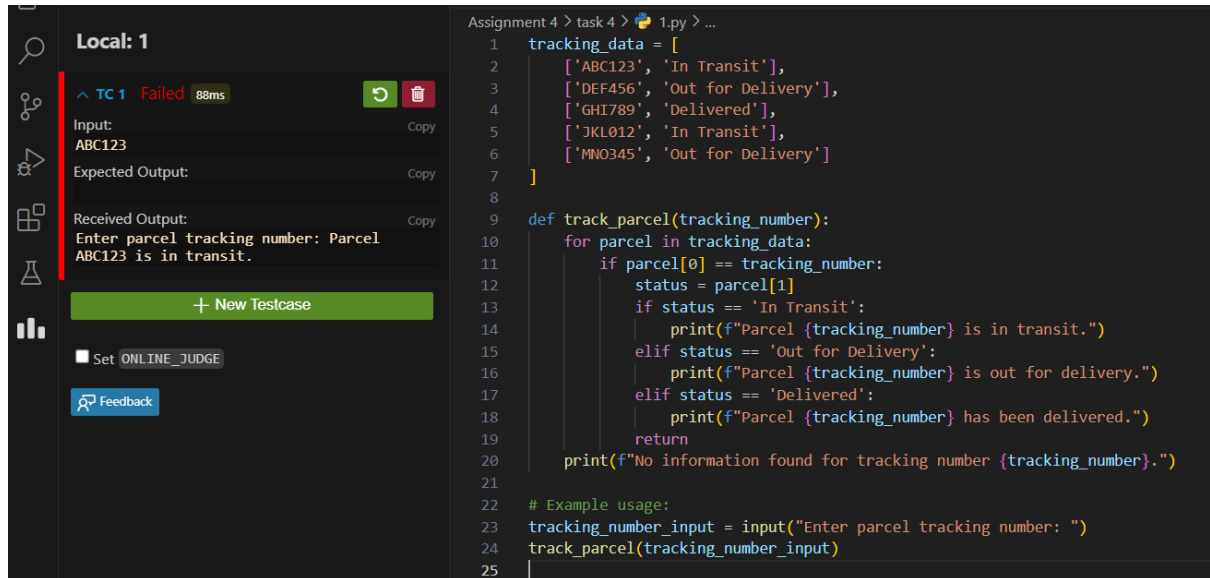
Feedback

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

```
1 class Courier:
2     def __init__(self, courier_id, current_location):
3         self.courier_id = courier_id
4         self.current_location = current_location
5         self.is_available = True
6
7 def find_nearest_courier(order_location, couriers):
8     nearest_courier = None
9     min_distance = float('inf')
10
11     for courier in couriers:
12         distance = abs(ord(order_location) - ord(courier.current_location))
13
14         if courier.is_available and distance < min_distance:
15             min_distance = distance
16             nearest_courier = courier
17
18     return nearest_courier
19
20 # Example usage:
21 couriers_array = [
22     Courier(courier_id='Courier1', current_location='LocationA'),
23     Courier(courier_id='Courier2', current_location='LocationB'),
24     Courier(courier_id='Courier3', current_location='LocationC')
25 ]
26
27 order_location_input = input("Enter order location: ")
28 nearest_courier = find_nearest_courier(order_location_input, couriers_array)
29
30 if nearest_courier:
31     print(f"The nearest available courier is {nearest_courier.courier_id} at {nearest_courier.current_location}.")
32 else:
33     print("No available couriers.")
34
```

## 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.



```
Assignment 4 > task 4 > 1.py > ...
1 tracking_data = [
2     ['ABC123', 'In Transit'],
3     ['DEF456', 'Out for Delivery'],
4     ['GHI789', 'Delivered'],
5     ['JKL012', 'In Transit'],
6     ['MNO345', 'Out for Delivery']
7 ]
8
9 def track_parcel(tracking_number):
10     for parcel in tracking_data:
11         if parcel[0] == tracking_number:
12             status = parcel[1]
13             if status == 'In Transit':
14                 print(f"Parcel {tracking_number} is in transit.")
15             elif status == 'Out for Delivery':
16                 print(f"Parcel {tracking_number} is out for delivery.")
17             elif status == 'Delivered':
18                 print(f"Parcel {tracking_number} has been delivered.")
19             return
20     print(f"No information found for tracking number {tracking_number}.")
21
22 # Example usage:
23 tracking_number_input = input("Enter parcel tracking number: ")
24 track_parcel(tracking_number_input)
25
```

Local: 1

TC 1 Failed 88ms

Input: ABC123

Expected Output:

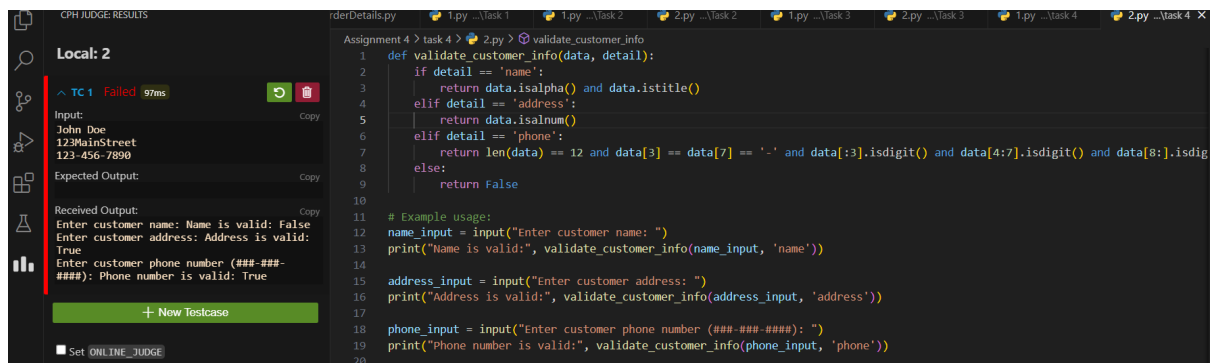
Received Output: Enter parcel tracking number: Parcel ABC123 is in transit.

+ New Testcase

Set ONLINE\_JUDGE

Feedback

10. Customer Data Validation: Write a function which takes 2 parameters, data-denotes the data and detail-denotes if it is name address or phone number. Validate customer information based on following criteria. 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., ###-###-####).



```
CPH JUDGE: RESULTS
Assignment 4 > task 4 > 2.py > validate_customer_info
1 def validate_customer_info(data, detail):
2     if detail == 'name':
3         return data.isalpha() and data.istitle()
4     elif detail == 'address':
5         return data.isalnum()
6     elif detail == 'phone':
7         return len(data) == 12 and data[3] == '-' and data[7] == '-' and data[3:].isdigit() and data[4:7].isdigit() and data[8:].isdigit()
8     else:
9         return False
10
11 # Example usage:
12 name_input = input("Enter customer name: ")
13 print("Name is valid:", validate_customer_info(name_input, 'name'))
14
15 address_input = input("Enter customer address: ")
16 print("Address is valid:", validate_customer_info(address_input, 'address'))
17
18 phone_input = input("Enter customer phone number (###-###-####): ")
19 print("Phone number is valid:", validate_customer_info(phone_input, 'phone'))
20
```

Local: 2

TC 1 Failed 97ms

Input: John Doe  
123MainStreet  
123-456-7890

Expected Output:

Received Output: Enter customer name: Name is valid: False  
Enter customer address: Address is valid: True  
Enter customer phone number (###-###-####): Phone number is valid: True

+ New Testcase

Set ONLINE\_JUDGE

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.

```

1 def format_address(street, city, state, zip_code):
2     formatted_address = f"{street.title()}, {city.title()}, {state.title()} {zip_code}"
3     return formatted_address
4
5 # Example usage:
6 street_input = input("Enter street: ")
7 city_input = input("Enter city: ")
8 state_input = input("Enter state: ")
9 zip_code_input = input("Enter zip code: ")
10 formatted_address = format_address(street_input, city_input, state_input, zip_code_input)
11 print("Formatted Address:", formatted_address)
12

```

Local: 3  
 TC 1 Failed 53ms  
 Input: Bhandara Road, Nagpur, Maharashtra, 440035  
 Expected Output: Bhandara Road, Nagpur, Maharashtra 440035  
 Received Output: Enter street: Enter city: Enter state: Enter zip code: Formatted Address: Bhandara Road, Nagpur, Maharashtra 440035

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.

```

1 def generate_order_confirmation_email(customer_name, order_number, delivery_address, delivery_date):
2     email_body = f"Dear {customer_name},\n\nThank you for your order!\n\nOrder Number: {order_number}\nDelivery Address: {delivery_address}\nExpected Delivery Date: {delivery_date}"
3     return email_body
4
5 # Example usage:
6 customer_name_input = input("Enter customer name: ")
7 order_number_input = input("Enter order number: ")
8 delivery_address_input = input("Enter delivery address: ")
9 delivery_date_input = input("Enter expected delivery date: ")
10 confirmation_email = generate_order_confirmation_email(customer_name_input, order_number_input, delivery_address_input, delivery_date_input)
11 print("Order Confirmation Email:\n", confirmation_email)
12

```

Local: 4  
 TC 1 Failed 70ms  
 Input: Krishna, 12344, Shivam Nagar, 2023-12-31  
 Expected Output: Enter customer name: Enter order number: Enter delivery address: Enter expected delivery date: Order Confirmation Email: Dear Krishna, Thank you for your order! Order Number: 12344 Delivery Address: Shivam Nagar Expected Delivery Date: 2023-12-31 Best regards, The Courier System Team

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.

```

1 def calculate_shipping_cost(source_address, destination_address, parcel_weight):
2     distance = abs(ord(source_address[0]) - ord(destination_address[0]))
3     weight_cost = parcel_weight * 2.5
4     total_cost = distance + weight_cost
5     return total_cost
6
7 # Example usage:
8 source_address_input = input("Enter source address: ")
9 destination_address_input = input("Enter destination address: ")
10 parcel_weight_input = float(input("Enter parcel weight (in kg): "))
11 shipping_cost = calculate_shipping_cost(source_address_input, destination_address_input, parcel_weight_input)
12 print("Shipping Cost:", shipping_cost)
13

```

Local: 5  
 TC 1 Failed 77ms  
 Input: Pune, Nagpur, 3  
 Expected Output: Enter source address: Enter destination address: Enter parcel weight (in kg): Shipping Cost: 9.5

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.

The screenshot shows a code editor with a Python file named '6.py'. The code defines a function `generate_password()` that generates a 12-character password using a mix of uppercase letters, lowercase letters, numbers, and punctuation. The function is tested with a test case (TC 1) that failed. The input is empty, and the expected output is an empty string. The received output is a generated password: `,FN9uG&\Bq''`. The test case failed with a message: `TC 1 Failed 120ms`.

```

1 import random
2 import string
3
4 def generate_password():
5     characters = string.ascii_letters + string.digits + string.punctuation
6     password = ''.join(random.choice(characters) for _ in range(12))
7     return password
8
9 # Example usage:
10 generated_password = generate_password()
11 print("Generated Password:", generated_password)
12

```

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.

The screenshot shows a code editor with a Python file named '7.py'. The code defines a function `find_similar_addresses(address, address_list)` that finds similar addresses in a list. The function is tested with a test case (TC 1) that failed. The input is the address '456', and the expected output is an empty list. The received output is a list of similar addresses: `['456 Park Avenue', '4567 Birch Drive']`. The test case failed with a message: `TC 1 Failed 71ms`.

```

1 def find_similar_addresses(address, address_list):
2     similar_addresses = [a for a in address_list if a.lower().startswith(address.lower())]
3     return similar_addresses
4
5 # Example usage:
6 address_to_find = input("Enter address to find: ")
7 address_list = ['123 Main Street', '456 Park Avenue', '789 Elm Street', '1234 Oak Lane', '4567 Birch Drive']
8 similar_addresses = find_similar_addresses(address_to_find, address_list)
9 print("Similar Addresses:", similar_addresses)
10

```

Following tasks are incremental stages to build an application and should be done in a single project

## 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

```
4
5 class User:
6     def __init__(self, userID, userName, email, password, contactNumber, address):
7         self.userID = userID
8         self.userName = userName
9         self.email = email
10        self.password = password
11        self.contactNumber = contactNumber
12        self.address = address
```

```
1 from DatabaseConnector import DatabaseConnector
2 from User import User
3
4 db_connector = DatabaseConnector(host="localhost", database="Courier_Management_System", user="root", password="Krishna@128")
5 db_connector.open_connection()
6
7 user=User(db_connector)
8
9 user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123", "9234567842", "Shivam Nagar")
10
11
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
1 from DatabaseConnector import DatabaseConnector
2 from User import User
3
4 db_connector = DatabaseConnector(host="localhost", database="Courier_Management_System", user="root", password="Krishna@128")
5 db_connector.open_connection()
6
7 user=User(db_connector)
8
9 user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123", "9234567842", "Shivam Nagar")
10
11 user.get_user(11)
12
13
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

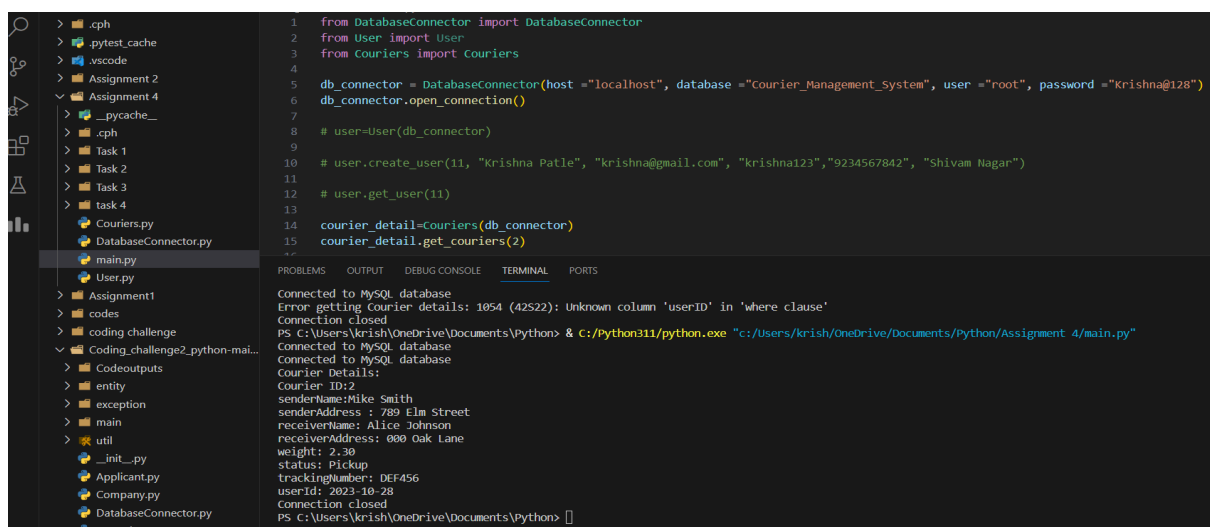
Connection closed  
PS C:\Users\krish\OneDrive\Documents\Python> & C:\Python311\python.exe "C:\Users\krish\OneDrive\Documents\Python\Assignment 4/main.py"  
Connected to MySQL database  
Connected to MySQL database  
Error creating user profiles: 1062 (23000): Duplicate entry '11' for key 'user.PRIMARY'  
Connection closed  
Connected to MySQL database  
User Details:  
User ID:11  
Username:Krishna Patle  
email : krishna@gmail.com  
password: krishna123  
contactNumber: 9234567842  
address: Shivam Nagar  
Connection closed  
PS C:\Users\krish\OneDrive\Documents\Python> []



## 2. Courier Class

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

```
17 class Courier:
18     tracking_number_counter = 1000 # Static variable for tracking number
19
20     def __init__(self, senderName, senderAddress, receiverName, receiverAddress, weight, status, userId):
21         self.courierID = None
22         self.senderName = senderName
23         self.senderAddress = senderAddress
24         self.receiverName = receiverName
25         self.receiverAddress = receiverAddress
26         self.weight = weight
27         self.status = status
28         self.trackingNumber = Courier.tracking_number_counter
29         self.deliveryDate = None
30         self.userId = userId
31
```



```
1 from DatabaseConnector import DatabaseConnector
2 from User import User
3 from Couriers import Couriers
4
5 db_connector = DatabaseConnector(host="localhost", database="Courier_Management_System", user="root", password="Krishna@128")
6 db_connector.open_connection()
7
8 # user=User(db_connector)
9
10 # user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123", "9234567842", "Shivam Nagar")
11
12 # user.get_user(11)
13
14 courier_detail=Couriers(db_connector)
15 courier_detail.get_couriers(2)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Connected to MySQL database  
Error getting Courier details: 1054 (42S22): Unknown column 'userId' in 'where clause'  
Connection closed  
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "C:/Users/krish/OneDrive/Documents/Python/Assignment 4/main.py"  
Connected to MySQL database  
Connected to MySQL database  
Courier Details:  
Courier ID:2  
senderName:Mike Smith  
senderAddress : 789 Elm Street  
receiverName: Alice Johnson  
receiverAddress: 000 Oak Lane  
weight: 2.30  
status: Pickup  
trackingNumber: DEF456  
userId: 2023-10-28  
Connection closed  
PS C:\Users\krish\OneDrive\Documents\Python>

### 3. Employee Class:

**Variables** employeeID , employeeName , email , contactNumber , role String, salary

```
1 class Employees:
2     def __init__(self, employeeID, Name, email, contactNumber, role, salary):
3         self.employeeID = employeeID
4         self.Name = Name
5         self.email = email
6         self.contactNumber = contactNumber
7         self.role = role
8         self.salary = salary
9
10    def __init__(self, db_connector):
11        self._db_connector = db_connector
12
13    def get_employees(self, employeeID):
14        try:
15            self._db_connector.open_connection()
16            query = "SELECT * FROM employees where employeeID=%s "
17            values=(employeeID,)
18            self._db_connector.cursor.execute(query, values)
19            employee_details = self._db_connector.cursor.fetchone()
20
21            if employee_details:
22                print("employee Details:")
23                print(f"employee ID:{employee_details[0]}")
24                print(f"Name:{employee_details[1]}")
25                print(f"email : {employee_details[2]}")
26                print(f"contactNumber: {employee_details[3]}")
27                print(f"role: {employee_details[4]}")
28                print(f"salary: {employee_details[5]}")
29
30            else:
31                print("Employee Id not found.")
32
```

```
1 from DatabaseConnector import DatabaseConnector
2 from User import User
3 from couriers import Couriers
4 from Employees import Employees
5
6 db_connector = DatabaseConnector(host="localhost", database="Courier_Management_System", user="root", password="Krishna@128")
7 db_connector.open_connection()
8
9 # user=User(db_connector)
10
11 # user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123", "9234567842", "Shivam Nagar")
12
13 # user.get_user(11)
14
15 # courier_detail=Couriers(db_connector)
16 # courier_detail.get_couriers(2)
17
18
19 employee=Employees(db_connector)
20 employee.get_employees(5)
21
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents/Python/Assignment 4/main.py"
Connected to MySQL database
Connected to MySQL database
employee Details:
employee ID:5
Name:Bob Miller
email : bob.miller@email.com
contactNumber: 111-111-1111
role: IT Administrator
salary: 55000.00
Connection closed
PS C:\Users\krish\OneDrive\Documents\Python> |
```

## 4. Location Class

Variables LocationID , LocationName , Address

```
1 class Locations:
2     def __init__(self,locationID, locationName, address):
3         self.locationID = locationID
4         self.locationName = locationName
5         self.address = address
6
7     def __init__(self, db_connector):
8         self._db_connector = db_connector
9
10    def get_location(self,locationID):
11        try:
12            self._db_connector.open_connection()
13            query = "SELECT * FROM locations where locationID=%s "
14            values=(locationID,)
15            self._db_connector.cursor.execute(query, values)
16            location_details = self._db_connector.cursor.fetchone()
17
18            if location_details:
19                print("employee Details:")
20                print(f"location ID:{location_details[0]}")
21                print(f"locationName:{location_details[1]}")
22                print(f"address : {location_details[2]}")
23
24            else:
25                print("Location Id not found.")
26
27        except Exception as e:
28            print(f"Error getting Location details: {e}")
29
30        finally:
31            self._db_connector.close_connection()
```

```
1 from Locations import Locations
2 from DatabaseConnector import DatabaseConnector
3 from User import User
4 from Couriers import Couriers
5 from Employees import Employees
6
7 db_connector = DatabaseConnector(host="localhost", database="Courier_Management_System", user="root", password="Krishna@128")
8 db_connector.open_connection()
9
10 # user=User(db_connector)
11
12 # user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123","9234567842", "Shivam Nagar")
13
14 # user.get_user(11)
15
16 # courier_detail=Couriers(db_connector)
17 # courier_detail.get_couriers(2)
18
19
20 # employee=Employees(db_connector)
21 # employee.get_employees(5)
22
23 location=Locations(db_connector)
24 location.get_location(5)
25
26
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents/Python/Assignment 4/main.py"
Connected to MySQL database
Connected to MySQL database
employee Details:
location ID:5
locationName:South Branch
address : 111 Maple Street, Houston, TX 77002
Connection closed
PS C:\Users\krish\OneDrive\Documents\Python> 
```

## 5. CourierCompany Class

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

```
1 class CourierCompany:
2     def __init__(self, companyName):
3         self.companyName = companyName
4         self.courierDetails = []
5         self.employeeDetails = []
6         self.locationDetails = []
7
8     def __init__(self, db_connector):
9         self._db_connector = db_connector
```

## 6. Payment Class:

**Variables** PaymentID long, CourierID long, Amount double, PaymentDate Date

```
1 from datetime import datetime
2 class Payments:
3     def __init__(self, paymentID, courierID, amount, paymentDate):
4         self.paymentID = paymentID
5         self.courierID = courierID
6         self.amount = amount
7         self.paymentDate = paymentDate
8
9     def __init__(self, db_connector):
10         self._db_connector = db_connector
11
12     def get_payments(self, paymentID):
13         try:
14             self._db_connector.open_connection()
15             query = "SELECT * FROM payments where paymentID=%s "
16             values=(paymentID,)
17             self._db_connector.cursor.execute(query, values)
18             payment_details = self._db_connector.cursor.fetchone()
19
20             if payment_details:
21                 print("employee Details:")
22                 print(f"payment ID:{payment_details[0]}")
23                 print(f"courierID:{payment_details[1]}")
24                 print(f"amount : {payment_details[2]}")
25                 print(f"paymentDate : {payment_details[3]}")
26             else:
27                 print("Payment Id not found.")
28
29         except Exception as e:
30             print(f"Error getting Payment details: {e}")
31
32         finally:
33             self._db_connector.close_connection()
```

```

4 from Couriers import Couriers
5 from Employees import Employees
6 from Payments import Payments
7
8 db_connector = DatabaseConnector(host="localhost", database="Courier_Management_System", user="root", password="Krishna@128")
9 db_connector.open_connection()
10
11 # user=User(db_connector)
12
13 # user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123", "9234567842", "Shivam Nagar")
14
15 # user.get_user(11)
16
17 # courier_detail=Couriers(db_connector)
18 # courier_detail.get_couriers(2)
19
20
21 # employee=Employees(db_connector)
22 # employee.get_employees(5)
23
24 # location=Locations(db_connector)
25 # location.get_location(5)
26
27 payment=Payments(db_connector)
28 payment.get_payments(3)
29

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents/Python/Assignment 4/main.py"
Connected to MySQL database
Connected to MySQL database
employee Details:
payment ID:3
courierID:3
amount : 3
paymentDate : 150.00
Connection closed
PS C:\Users\krish\OneDrive\Documents\Python> 

```

```

20 class CourierCompanyCollection:
21     def __init__(self):
22         self.courierDetails = []
23
24
25     def placeOrder(self, courierObj):
26         self.companyObj.courierDetails.append(courierObj)
27         return courierObj.trackingNumber
28
29     def getOrderStatus(self, trackingNumber):
30         for courier in self.companyObj.courierDetails:
31             if courier.trackingNumber == trackingNumber:
32                 return courier.status
33         raise TrackingNumberNotFoundException("Tracking number not found.")
34
35     def cancelOrder(self, trackingNumber):
36         for courier in self.companyObj.courierDetails:
37             if courier.trackingNumber == trackingNumber:
38                 self.companyObj.courierDetails.remove(courier)
39                 return True
40         raise TrackingNumberNotFoundException("Tracking number not found.")
41
42     def getAssignedOrder(self, courierStaffId):
43         assigned_orders = []
44         for courier in self.companyObj.courierDetails:
45             if courier.userId == courierStaffId:
46                 assigned_orders.append(courier)
47         return assigned_orders

```

## Task 7: Exception Handling

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

Define the following custom exceptions and throw them in methods whenever needed . Handle all the exceptions in 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

```
2  class TrackingNumberNotFoundException(Exception):
3      |   pass
4
5  class InvalidEmployeeIdException(Exception):
6      |   pass
7
8  def main():
```