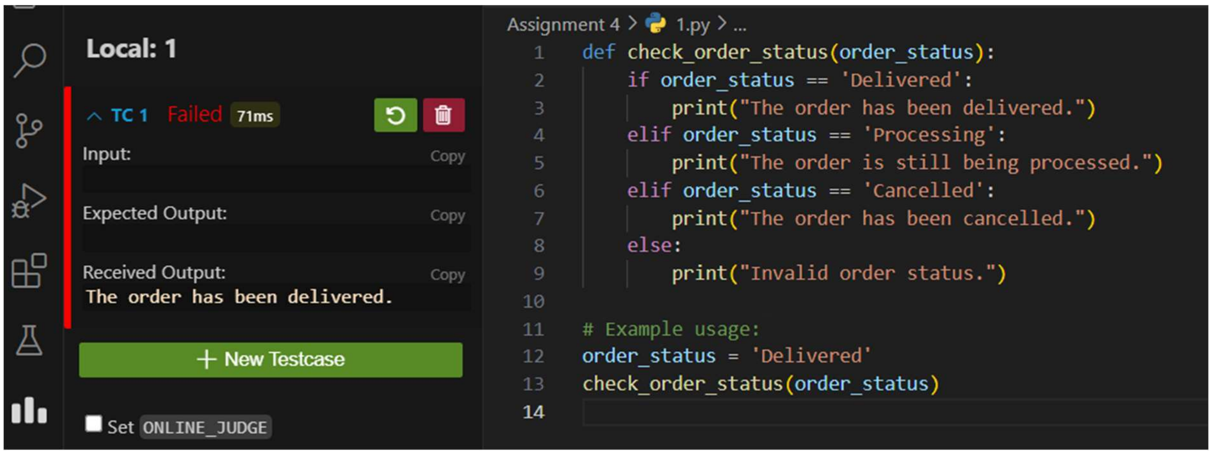
Assignment 4

Coding Task 1: Control Flow Statements

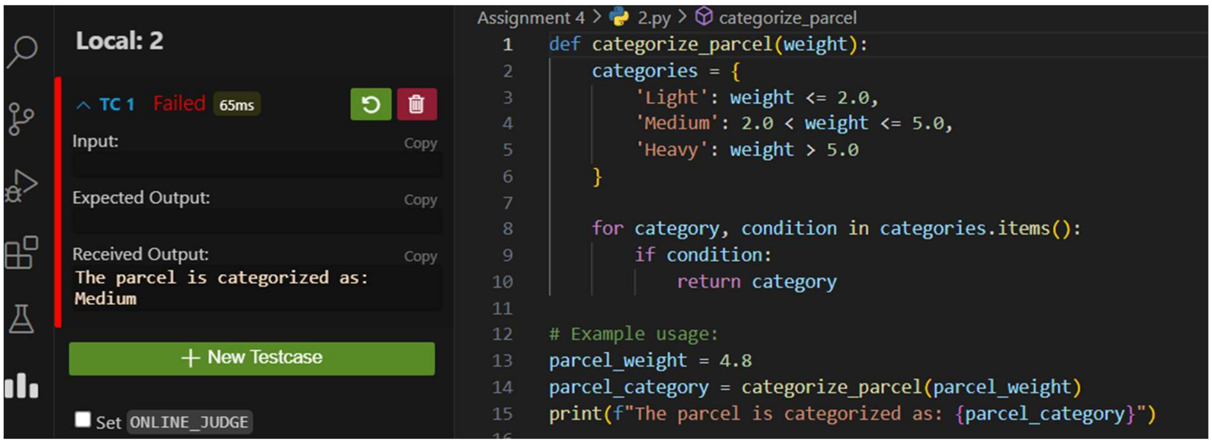
1. Write a program that checks whether a given order is delivered or not based on its status (e.g., "Processing," "Delivered," "Cancelled"). Use if-else statements for this.



```
The order has been delivered.
```

```python
def check_order_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.")

# Example usage:
order_status = 'Delivered'
check_order_status(order_status)
```

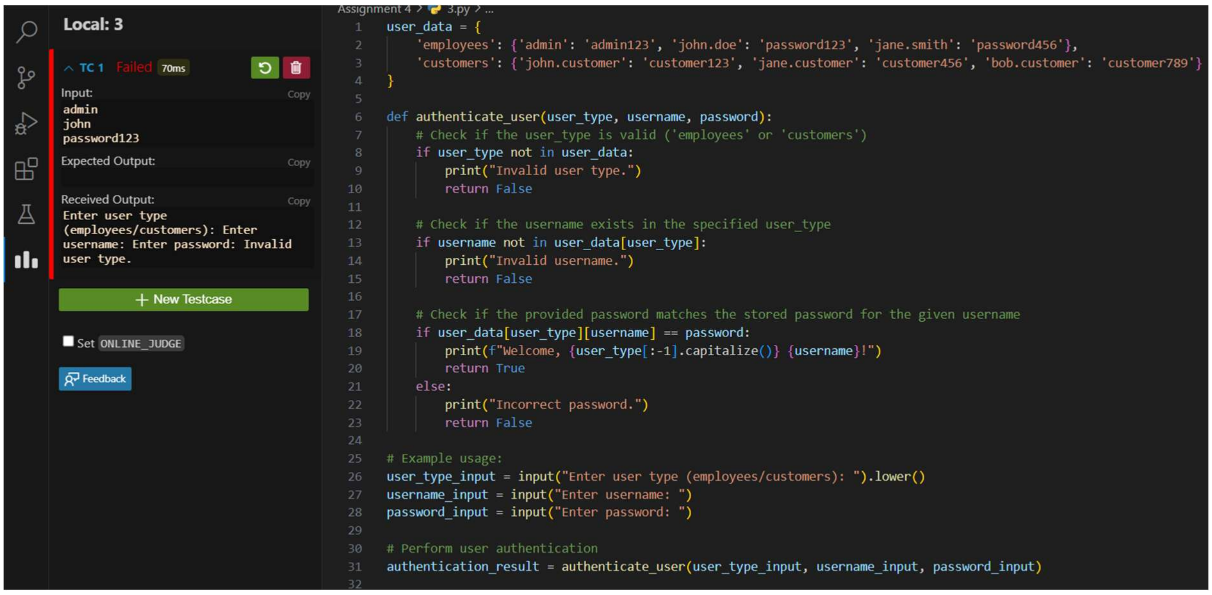
2. Implement a switch-case statement to categorize parcels based on their weight into "Light," "Medium," or "Heavy."



```
The parcel is categorized as:
Medium
```

```python
def categorize_parcel(weight):
    categories = {
        'Light': weight <= 2.0,
        'Medium': 2.0 < weight <= 5.0,
        'Heavy': weight > 5.0
    }

    for category, condition in categories.items():
        if condition:
            return category

# Example usage:
parcel_weight = 4.8
parcel_category = categorize_parcel(parcel_weight)
print(f"The parcel is categorized as: {parcel_category}")
```

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



Local: 3

TC 1 Failed 70ms

Input:
```
admin
john
password123
```

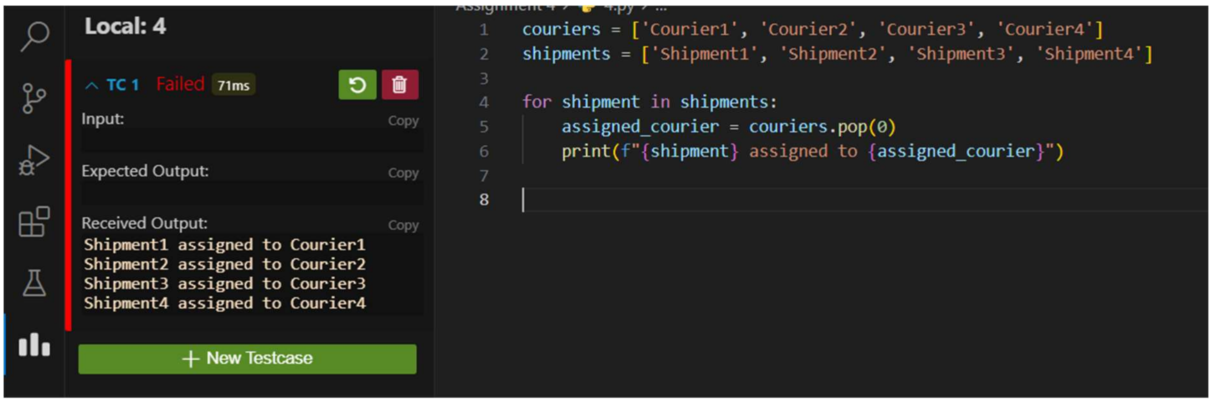
Expected Output:

Received Output:
```
Enter user type
(employees/customers): Enter
username: Enter password: Invalid
user type.
```

Assignment 4 > 3.py > ...

```python
user_data = {
    'employees': {'admin': 'admin123', 'john.doe': 'password123', 'jane.smith': 'password456'},
    'customers': {'john.customer': 'customer123', 'jane.customer': 'customer456', 'bob.customer': 'customer789'}
}

def authenticate_user(user_type, username, password):
    # Check if the user_type is valid ('employees' or 'customers')
    if user_type not in user_data:
        print("Invalid user type.")
        return False

    # Check if the username exists in the specified user_type
    if username not in user_data[user_type]:
        print("Invalid username.")
        return False

    # Check if the provided password matches the stored password for the given username
    if user_data[user_type][username] == password:
        print(f"Welcome, {user_type[:-1].capitalize()} {username}!")
        return True
    else:
        print("Incorrect password.")
        return False

# Example usage:
user_type_input = input("Enter user type (employees/customers): ").lower()
username_input = input("Enter username: ")
password_input = input("Enter password: ")

# Perform user authentication
authentication_result = authenticate_user(user_type_input, username_input, password_input)
```

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.



Local: 4

TC 1 Failed 71ms

Input:

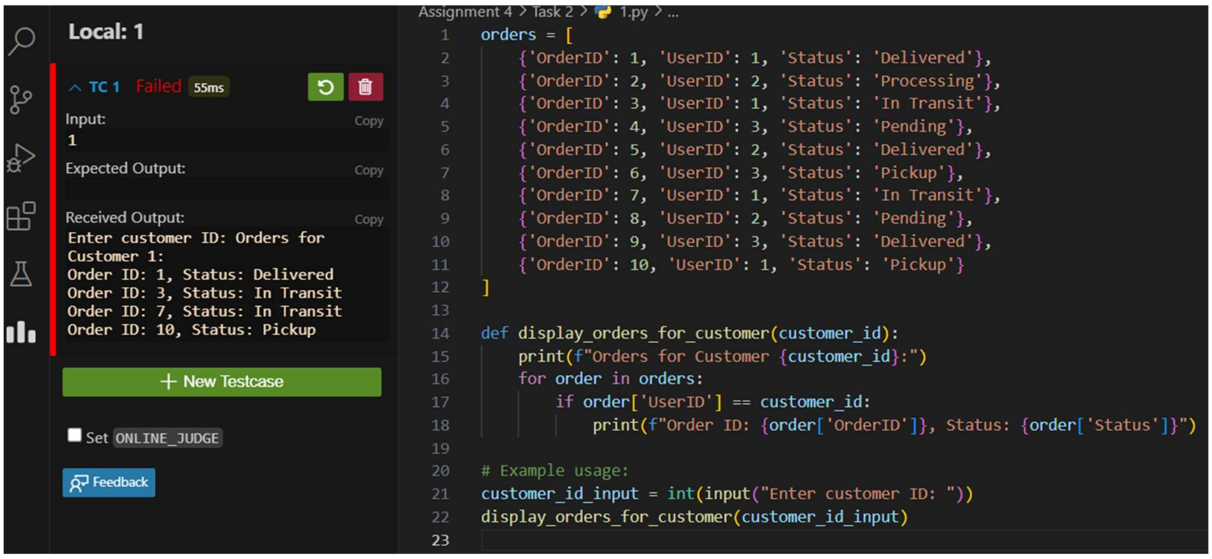
Expected Output:

Received Output:
```
Shipment1 assigned to Courier1
Shipment2 assigned to Courier2
Shipment3 assigned to Courier3
Shipment4 assigned to Courier4
```

```python
couriers = ['Courier1', 'Courier2', 'Courier3', 'Courier4']
shipments = ['Shipment1', 'Shipment2', 'Shipment3', 'Shipment4']

for shipment in shipments:
    assigned_courier = couriers.pop(0)
    print(f"{shipment} assigned to {assigned_courier}")

```

Task 2: Loops and Iteration

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



Local: 1

TC 1 Failed 55ms

Input:
```
1
```

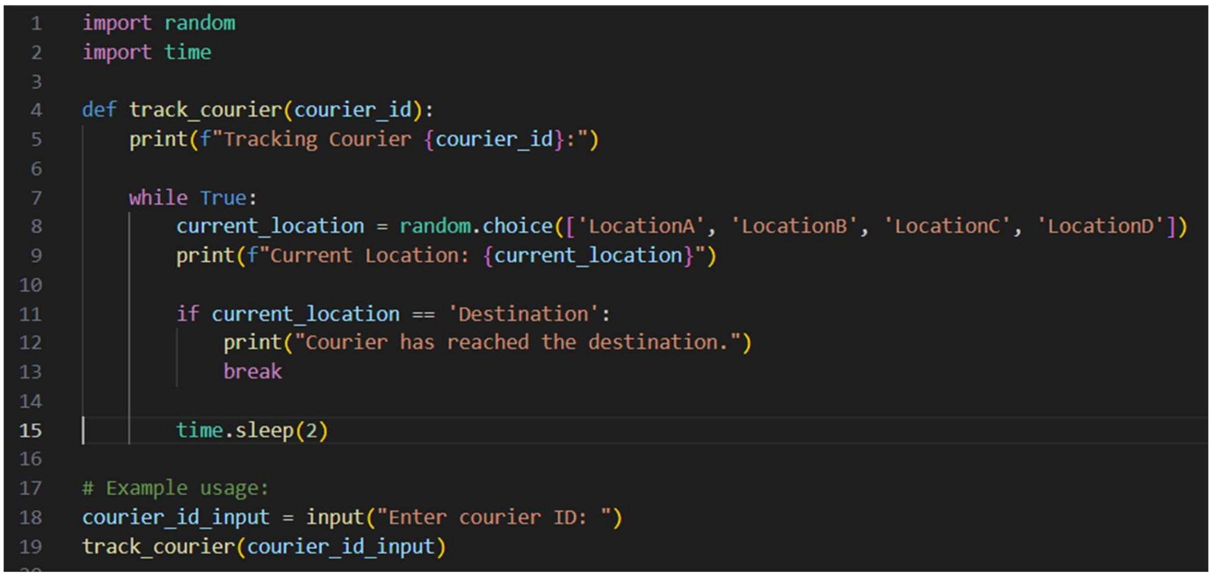
Expected Output:

Received Output:
```
Enter customer ID: Orders for
Customer 1:
Order ID: 1, Status: Delivered
Order ID: 3, Status: In Transit
Order ID: 7, Status: In Transit
Order ID: 10, Status: Pickup
```

Assignment 4 > Task 2 > 1.py > ...

```python
orders = [
    {'OrderID': 1, 'UserID': 1, 'Status': 'Delivered'},
    {'OrderID': 2, 'UserID': 2, 'Status': 'Processing'},
    {'OrderID': 3, 'UserID': 1, 'Status': 'In Transit'},
    {'OrderID': 4, 'UserID': 3, 'Status': 'Pending'},
    {'OrderID': 5, 'UserID': 2, 'Status': 'Delivered'},
    {'OrderID': 6, 'UserID': 3, 'Status': 'Pickup'},
    {'OrderID': 7, 'UserID': 1, 'Status': 'In Transit'},
    {'OrderID': 8, 'UserID': 2, 'Status': 'Pending'},
    {'OrderID': 9, 'UserID': 3, 'Status': 'Delivered'},
    {'OrderID': 10, 'UserID': 1, 'Status': 'Pickup'}
]

def display_orders_for_customer(customer_id):
    print(f"Orders for Customer {customer_id}:")
    for order in orders:
        if order['UserID'] == customer_id:
            print(f"Order ID: {order['OrderID']}, Status: {order['Status']}")

# Example usage:
customer_id_input = int(input("Enter customer ID: "))
display_orders_for_customer(customer_id_input)
```

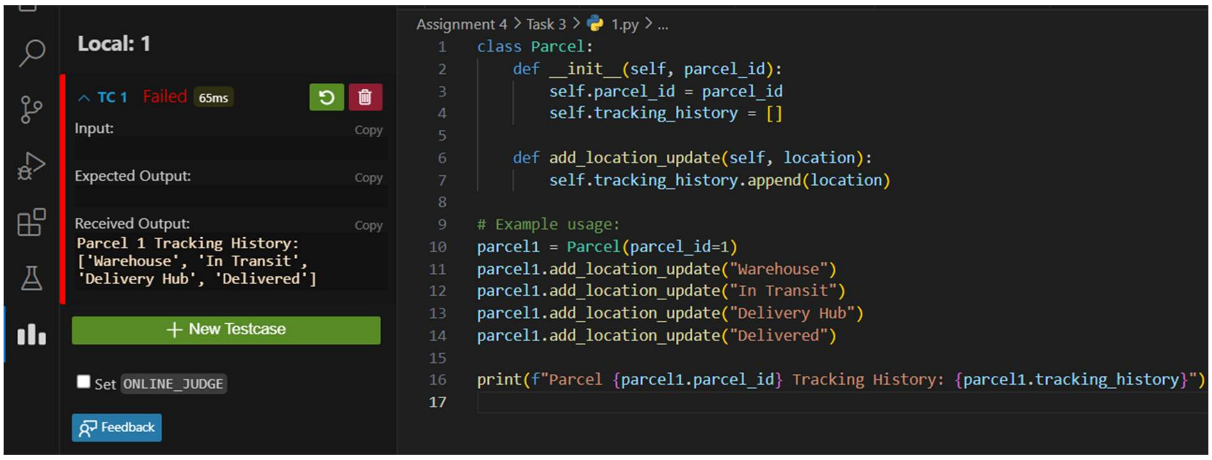
6. Implement a while loop to track the real-time location of a courier until it reaches its destination.



```python
import random
import time

def track_courier(courier_id):
    print(f"Tracking Courier {courier_id}:")

    while True:
        current_location = random.choice(['LocationA', 'LocationB', 'LocationC', 'LocationD'])
        print(f"Current Location: {current_location}")

        if current_location == 'Destination':
            print("Courier has reached the destination.")
            break

        time.sleep(2)

# Example usage:
courier_id_input = input("Enter courier ID: ")
track_courier(courier_id_input)
```

Task 3: Arrays and Data Structures

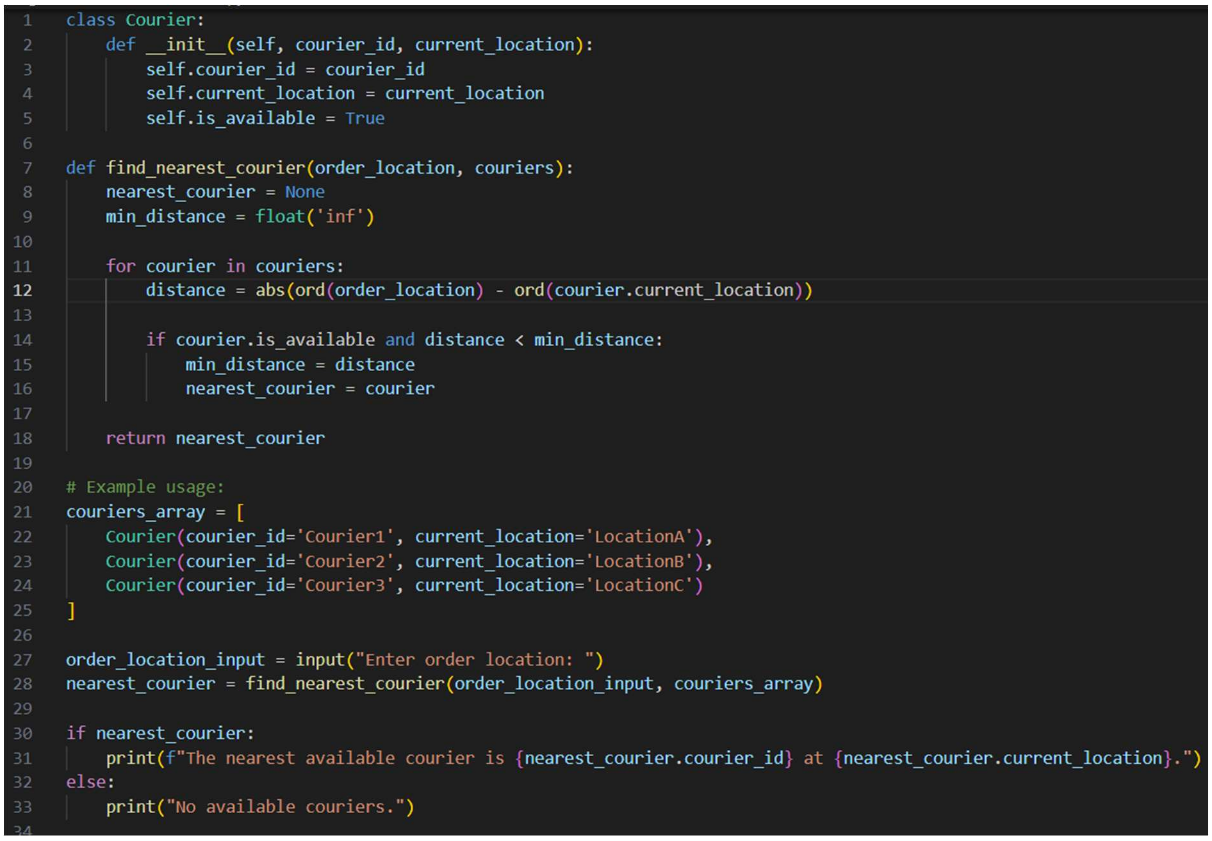
7. Create an array to store the tracking history of a parcel, where each entry represents a location update.



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

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

    def add_location_update(self, location):
        self.tracking_history.append(location)

# Example usage:
parcel1 = Parcel(parcel_id=1)
parcel1.add_location_update("Warehouse")
parcel1.add_location_update("In Transit")
parcel1.add_location_update("Delivery Hub")
parcel1.add_location_update("Delivered")

print(f"Parcel {parcel1.parcel_id} Tracking History: {parcel1.tracking_history}")
```

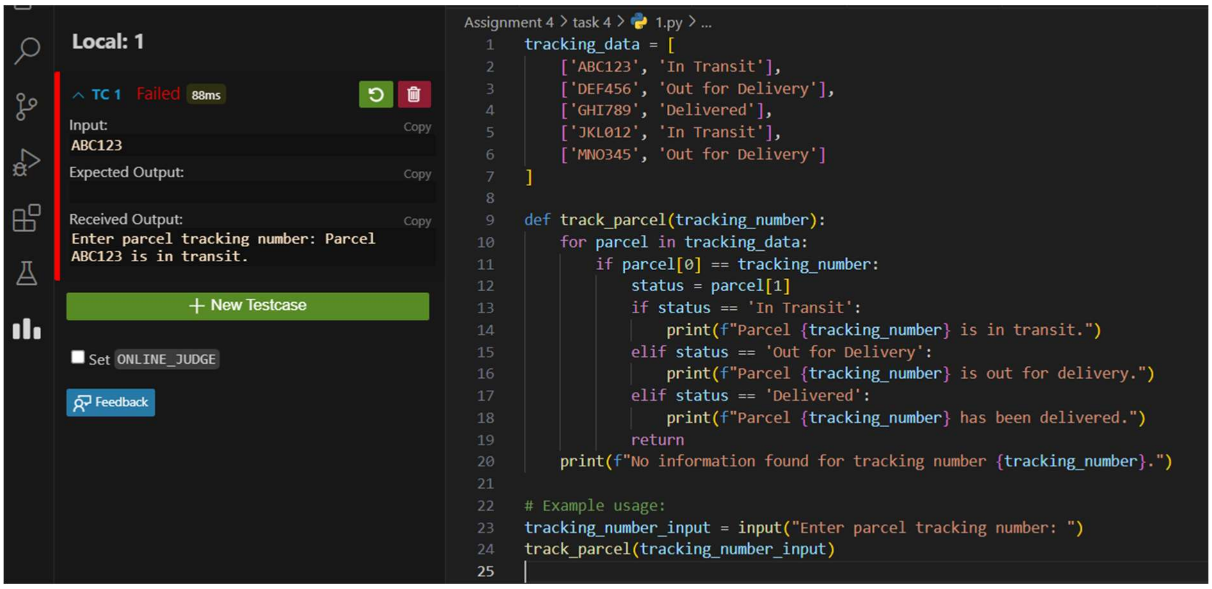
8. Implement a method to find the nearest available courier for a new order using an array of couriers.



```python
class Courier:
    def __init__(self, courier_id, current_location):
        self.courier_id = courier_id
        self.current_location = current_location
        self.is_available = True

def find_nearest_courier(order_location, couriers):
    nearest_courier = None
    min_distance = float('inf')

    for courier in couriers:
        distance = abs(ord(order_location) - ord(courier.current_location))

        if courier.is_available and distance < min_distance:
            min_distance = distance
            nearest_courier = courier

    return nearest_courier

# Example usage:
couriers_array = [
    Courier(courier_id='Courier1', current_location='LocationA'),
    Courier(courier_id='Courier2', current_location='LocationB'),
    Courier(courier_id='Courier3', current_location='LocationC')
]

order_location_input = input("Enter order location: ")
nearest_courier = find_nearest_courier(order_location_input, couriers_array)

if nearest_courier:
    print(f"The nearest available courier is {nearest_courier.courier_id} at {nearest_courier.current_location}.")
else:
    print("No available couriers.")
```

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

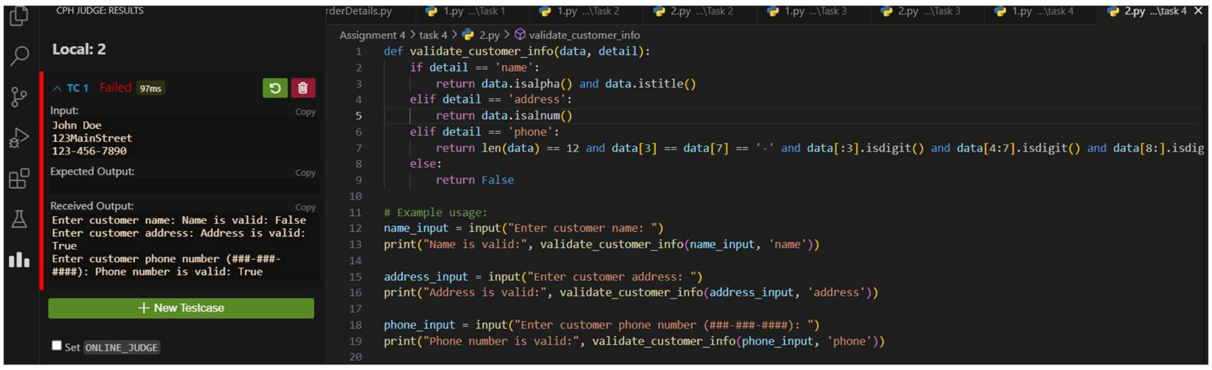
9. Parcel Tracking: Create a program that allows users to input a parcel tracking number.Store the tracking number and Status in 2d String Array. Initialize the array with values. Then, simulate the tracking process by displaying messages like "Parcel in transit," "Parcel out for delivery," or "Parcel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



```
Input:
ABC123
Expected Output:

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

```python
tracking_data = [
    ['ABC123', 'In Transit'],
    ['DEF456', 'Out for Delivery'],
    ['GHI789', 'Delivered'],
    ['JKL012', 'In Transit'],
    ['MNO345', 'Out for Delivery']
]

def track_parcel(tracking_number):
    for parcel in tracking_data:
        if parcel[0] == tracking_number:
            status = parcel[1]
            if status == 'In Transit':
                print(f"Parcel {tracking_number} is 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"No information found for tracking number {tracking_number}.")

# Example usage:
tracking_number_input = input("Enter parcel tracking number: ")
track_parcel(tracking_number_input)
```

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


CPH JUDGE: RESULTS

Local: 2

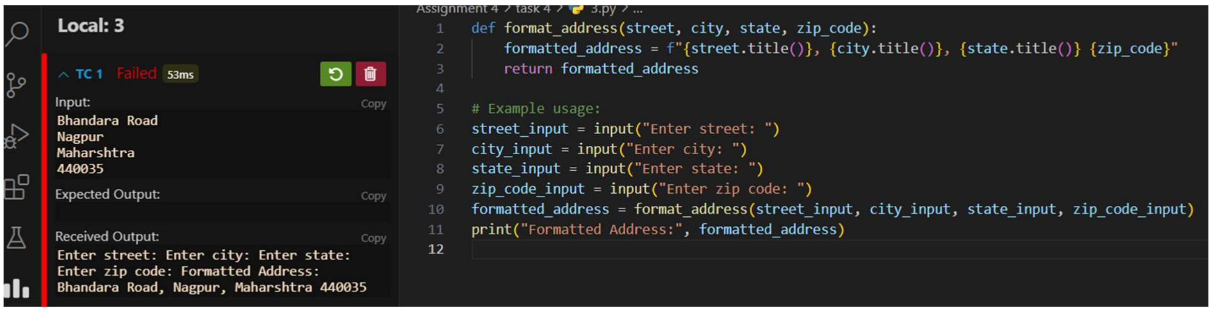
TC 1 Failed 97ms

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
```

Assignment 4 > task 4 > 2.py > validate_customer_info
```python
def validate_customer_info(data, detail):
    if detail == 'name':
        return data.isalpha() and data.istitle()
    elif detail == 'address':
        return data.isalnum()
    elif detail == 'phone':
        return len(data) == 12 and data[3] == data[7] == '-' and data[:3].isdigit() and data[4:7].isdigit() and data[8:].isdig
    else:
        return False

# Example usage:
name_input = input("Enter customer name: ")
print("Name is valid:", validate_customer_info(name_input, 'name'))

address_input = input("Enter customer address: ")
print("Address is valid:", validate_customer_info(address_input, 'address'))

phone_input = input("Enter customer phone number (###-###-####): ")
print("Phone number is valid:", validate_customer_info(phone_input, 'phone'))
```

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.


Local: 3

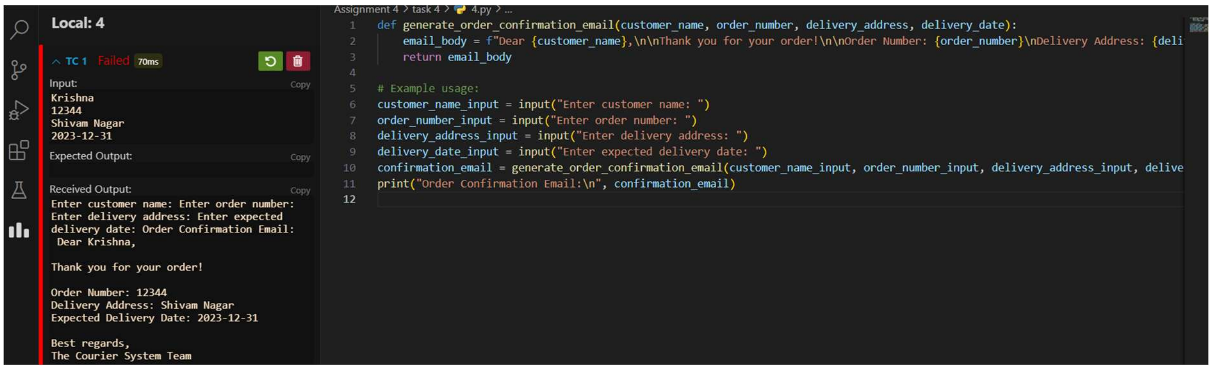
TC 1 Failed 53ms

Input:
```
Bhandara Road
Nagpur
Maharshtra
440035
```
Expected Output:

Received Output:
```
Enter street: Enter city: Enter state:
Enter zip code: Formatted Address:
Bhandara Road, Nagpur, Maharshtra 440035
```

Assignment 4 > task 4 > 3.py > ...
```python
def format_address(street, city, state, zip_code):
    formatted_address = f"{street.title()}, {city.title()}, {state.title()} {zip_code}"
    return formatted_address

# Example usage:
street_input = input("Enter street: ")
city_input = input("Enter city: ")
state_input = input("Enter state: ")
zip_code_input = input("Enter zip code: ")
formatted_address = format_address(street_input, city_input, state_input, zip_code_input)
print("Formatted Address:", formatted_address)

```

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.


Local: 4

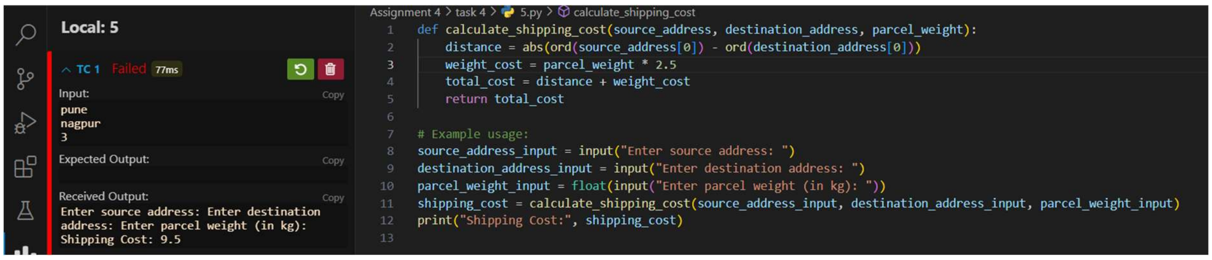
TC 1 Failed 70ms

Input:
```
Krishna
12344
Shivam Nagar
2023-12-31
```
Expected Output:

Received 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
```

Assignment 4 > task 4 > 4.py > ...
```python
def generate_order_confirmation_email(customer_name, order_number, delivery_address, delivery_date):
    email_body = f"Dear {customer_name},\n\nThank you for your order!\n\nOrder Number: {order_number}\nDelivery Address: {deli
    return email_body

# Example usage:
customer_name_input = input("Enter customer name: ")
order_number_input = input("Enter order number: ")
delivery_address_input = input("Enter delivery address: ")
delivery_date_input = input("Enter expected delivery date: ")
confirmation_email = generate_order_confirmation_email(customer_name_input, order_number_input, delivery_address_input, delive
print("Order Confirmation Email:\n", confirmation_email)

```

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.


Local: 5

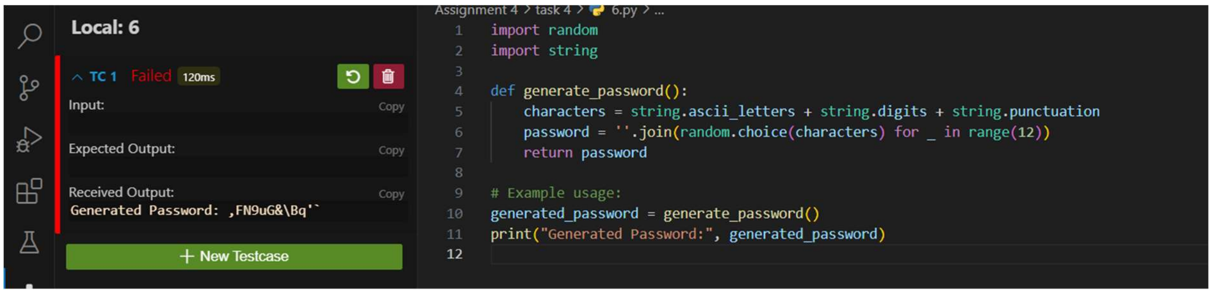
TC 1 Failed 77ms

Input:
```
pune
nagpur
3
```
Expected Output:

Received Output:
```
Enter source address: Enter destination
address: Enter parcel weight (in kg):
Shipping Cost: 9.5
```

Assignment 4 > task 4 > 5.py > calculate_shipping_cost
```python
def calculate_shipping_cost(source_address, destination_address, parcel_weight):
    distance = abs(ord(source_address[0]) - ord(destination_address[0]))
    weight_cost = parcel_weight * 2.5
    total_cost = distance + weight_cost
    return total_cost

# Example usage:
source_address_input = input("Enter source address: ")
destination_address_input = input("Enter destination address: ")
parcel_weight_input = float(input("Enter parcel weight (in kg): "))
shipping_cost = calculate_shipping_cost(source_address_input, destination_address_input, parcel_weight_input)
print("Shipping Cost:", shipping_cost)

```

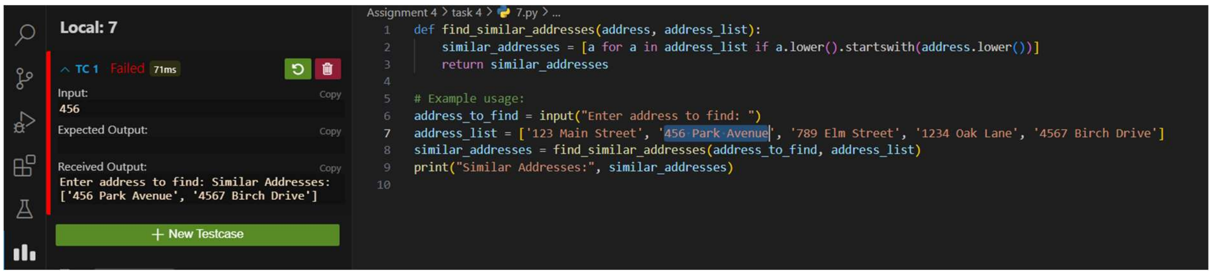
14. Password Generator: Create a function that generates secure passwords for courier system accounts. Ensure the passwords contain a mix of uppercase letters, lowercase letters, numbers, and special characters.



```
Generated Password: ,FN9uG&\Bq'`
```

```python
import random
import string

def generate_password():
    characters = string.ascii_letters + string.digits + string.punctuation
    password = ''.join(random.choice(characters) for _ in range(12))
    return password

# Example usage:
generated_password = generate_password()
print("Generated Password:", generated_password)
```

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.



```
456
```

```
Enter address to find: Similar Addresses:
['456 Park Avenue', '4567 Birch Drive']
```

```python
def find_similar_addresses(address, address_list):
    similar_addresses = [a for a in address_list if a.lower().startswith(address.lower())]
    return similar_addresses

# Example usage:
address_to_find = input("Enter address to find: ")
address_list = ['123 Main Street', '456 Park Avenue', '789 Elm Street', '1234 Oak Lane', '4567 Birch Drive']
similar_addresses = find_similar_addresses(address_to_find, address_list)
print("Similar Addresses:", similar_addresses)
```

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

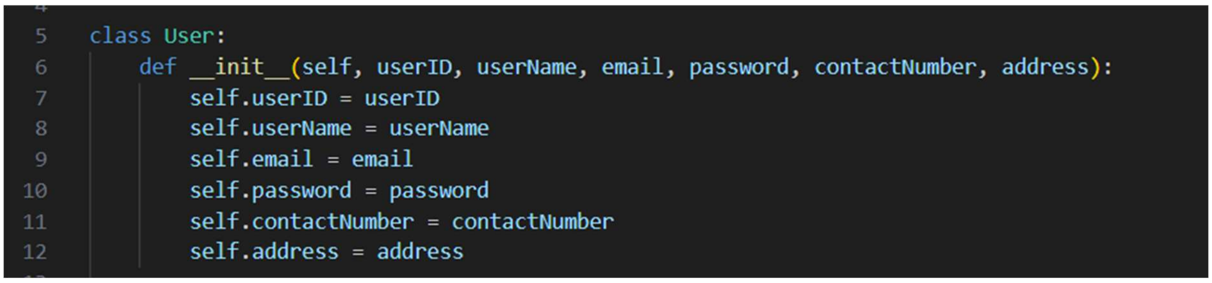
Create the following model/entity classes within package entities with variables declared

private, constructors(default and parametrized,getters,setters and toString())

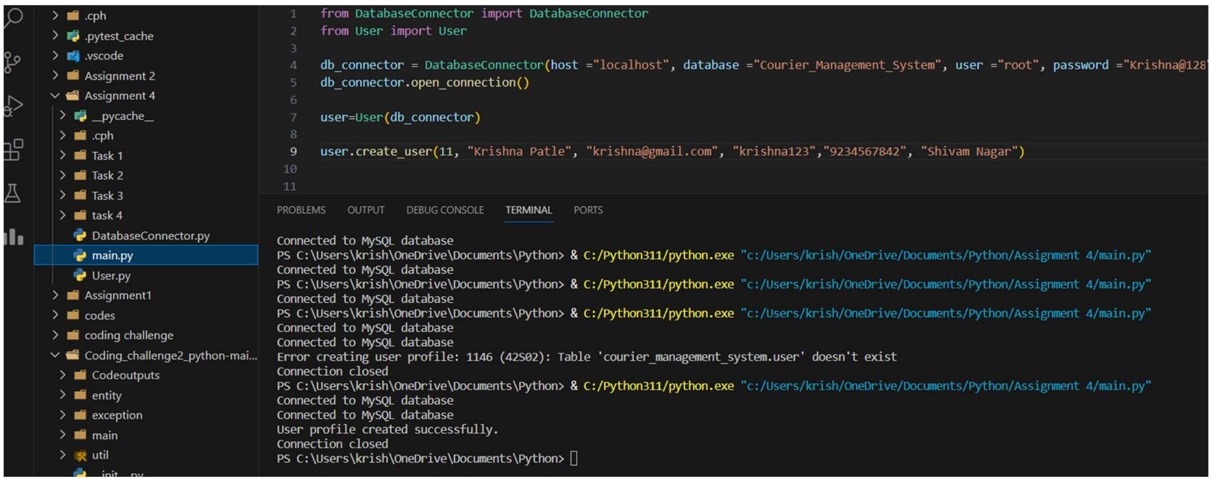
1. User Class:

Variables:

userID , userName , email , password , contactNumber , address

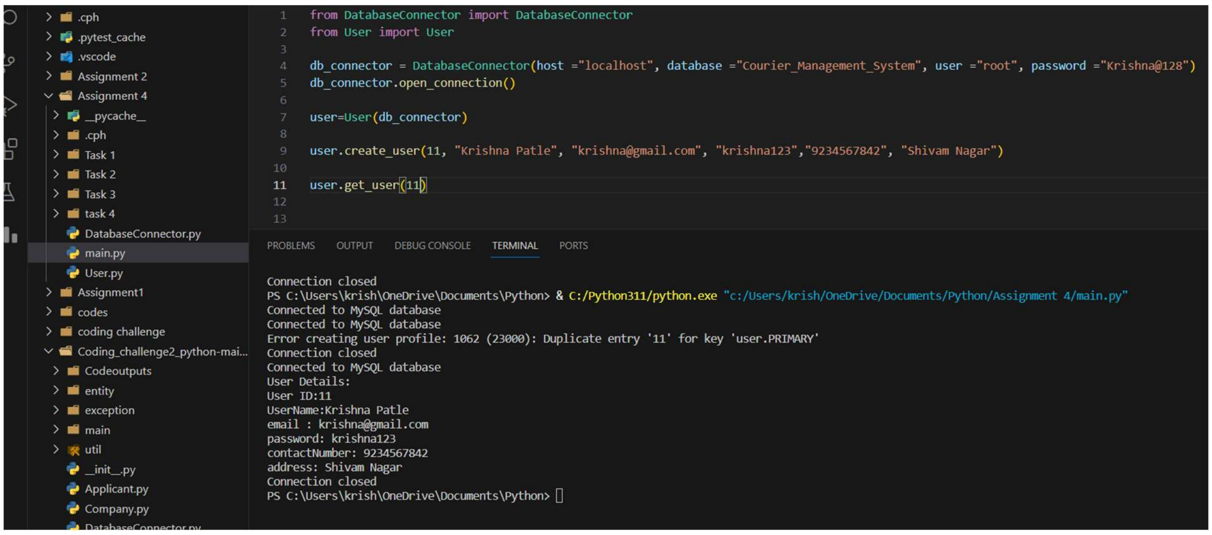


```python
class User:
    def __init__(self, userID, userName, email, password, contactNumber, address):
        self.userID = userID
        self.userName = userName
        self.email = email
        self.password = password
        self.contactNumber = contactNumber
        self.address = address
```



```python
from DatabaseConnector import DatabaseConnector
from User import User

db_connector = DatabaseConnector(host ="localhost", database ="Courier_Management_System", user ="root", password ="Krishna@128
db_connector.open_connection()

user=User(db_connector)

user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123","9234567842", "Shivam Nagar")
```

```
Connected to MySQL database
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
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
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 profile: 1146 (42S02): Table 'courier_management_system.user' doesn't exist
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
User profile created successfully.
Connection closed
PS C:\Users\krish\OneDrive\Documents\Python> 
```



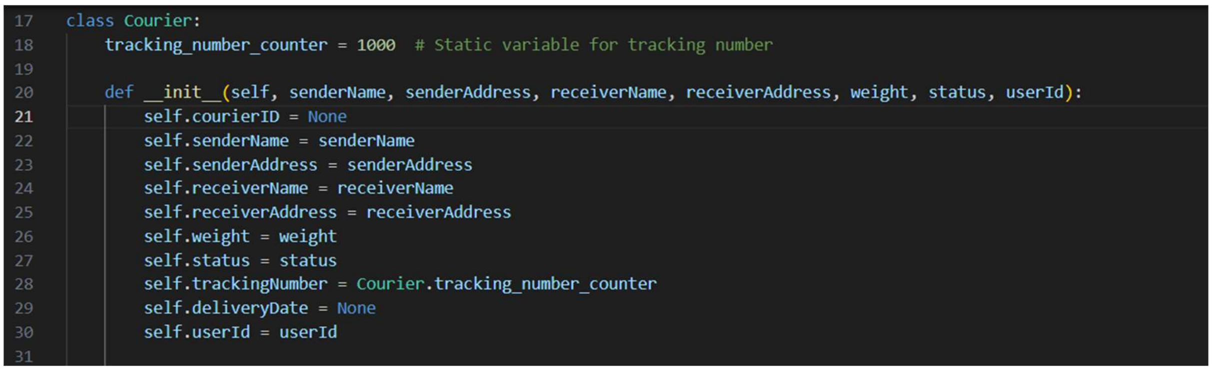
```python
from DatabaseConnector import DatabaseConnector
from User import User

db_connector = DatabaseConnector(host ="localhost", database ="Courier_Management_System", user ="root", password ="Krishna@128")
db_connector.open_connection()

user=User(db_connector)

user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123","9234567842", "Shivam Nagar")

user.get_user(11)
```

```
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 profile: 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> 
```

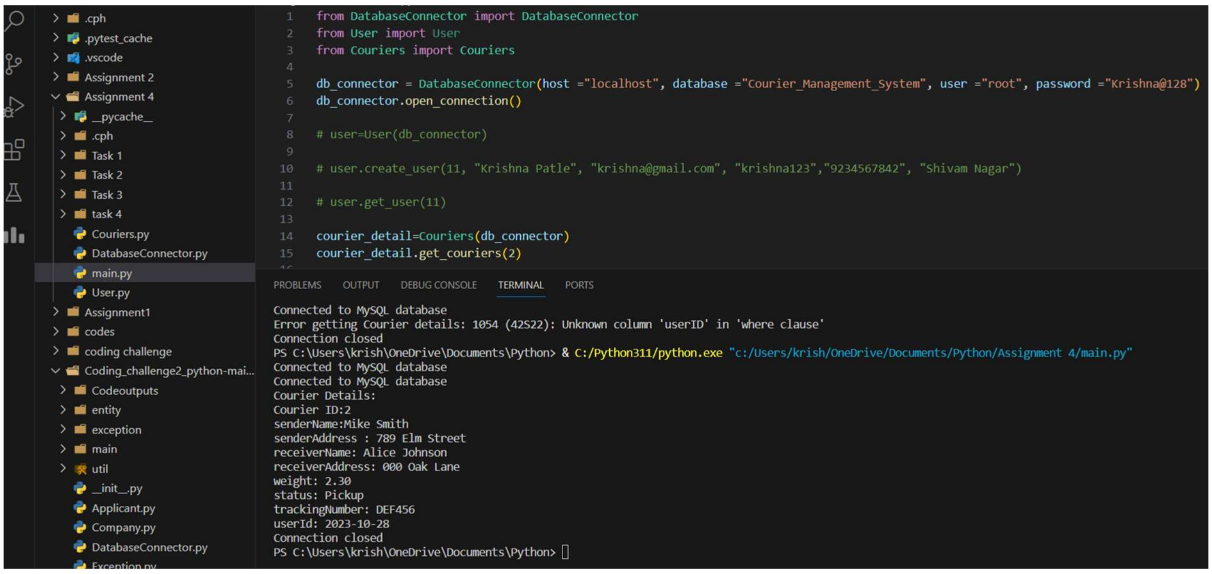
2. Courier Class

Variables: courierID , senderName , senderAddress , receiverName , receiverAddress , weight ,

status, trackingNumber , deliveryDate ,userId



```python
class Courier:
    tracking_number_counter = 1000  # Static variable for tracking number

    def __init__(self, senderName, senderAddress, receiverName, receiverAddress, weight, status, userId):
        self.courierID = None
        self.senderName = senderName
        self.senderAddress = senderAddress
        self.receiverName = receiverName
        self.receiverAddress = receiverAddress
        self.weight = weight
        self.status = status
        self.trackingNumber = Courier.tracking_number_counter
        self.deliveryDate = None
        self.userId = userId

```

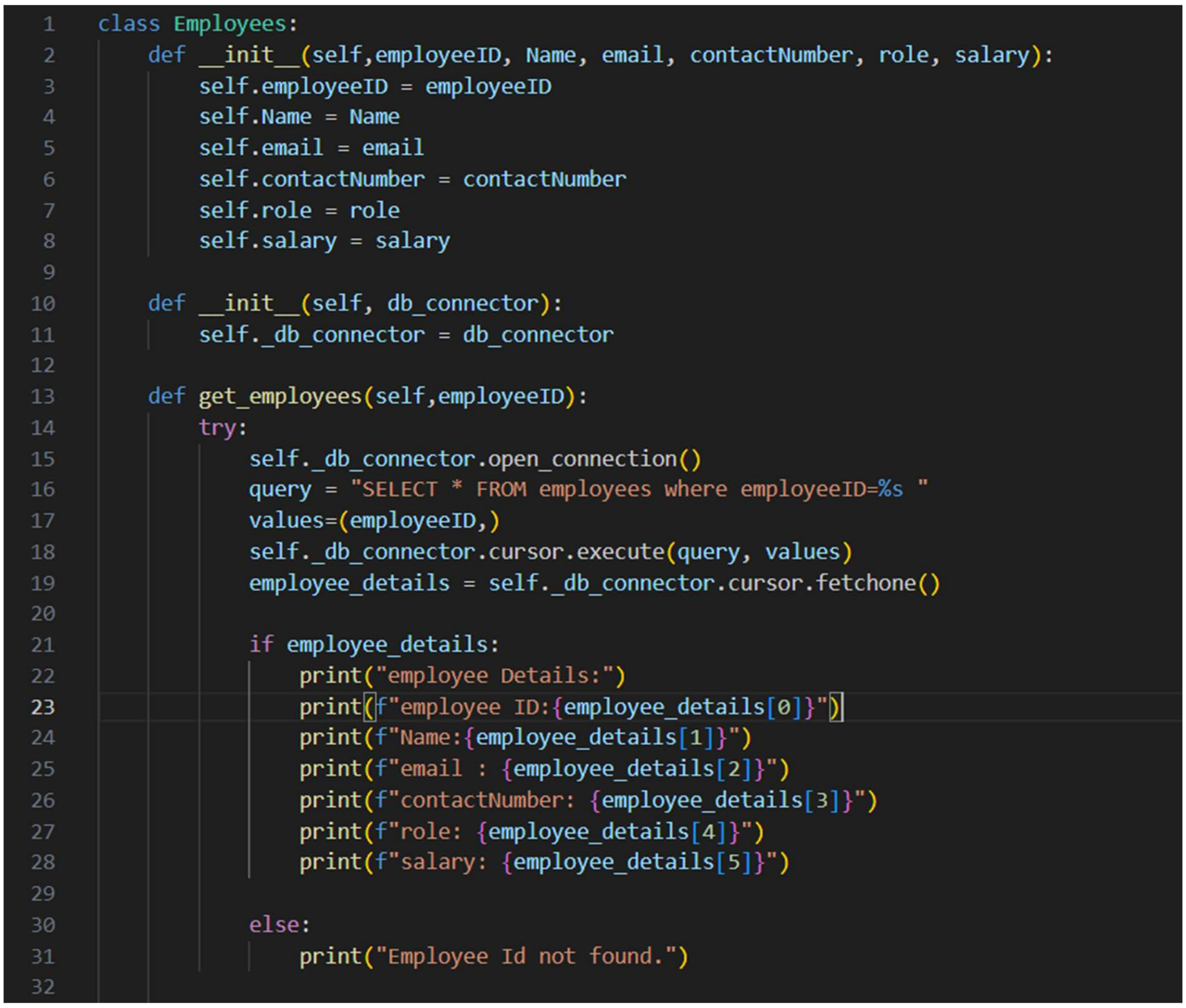


```python
from DatabaseConnector import DatabaseConnector
from User import User
from Couriers import Couriers

db_connector = DatabaseConnector(host ="localhost", database ="Courier_Management_System", user ="root", password ="Krishna@128")
db_connector.open_connection()

# user=User(db_connector)

# user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123","9234567842", "Shivam Nagar")

# user.get_user(11)

courier_detail=Couriers(db_connector)
courier_detail.get_couriers(2)
```

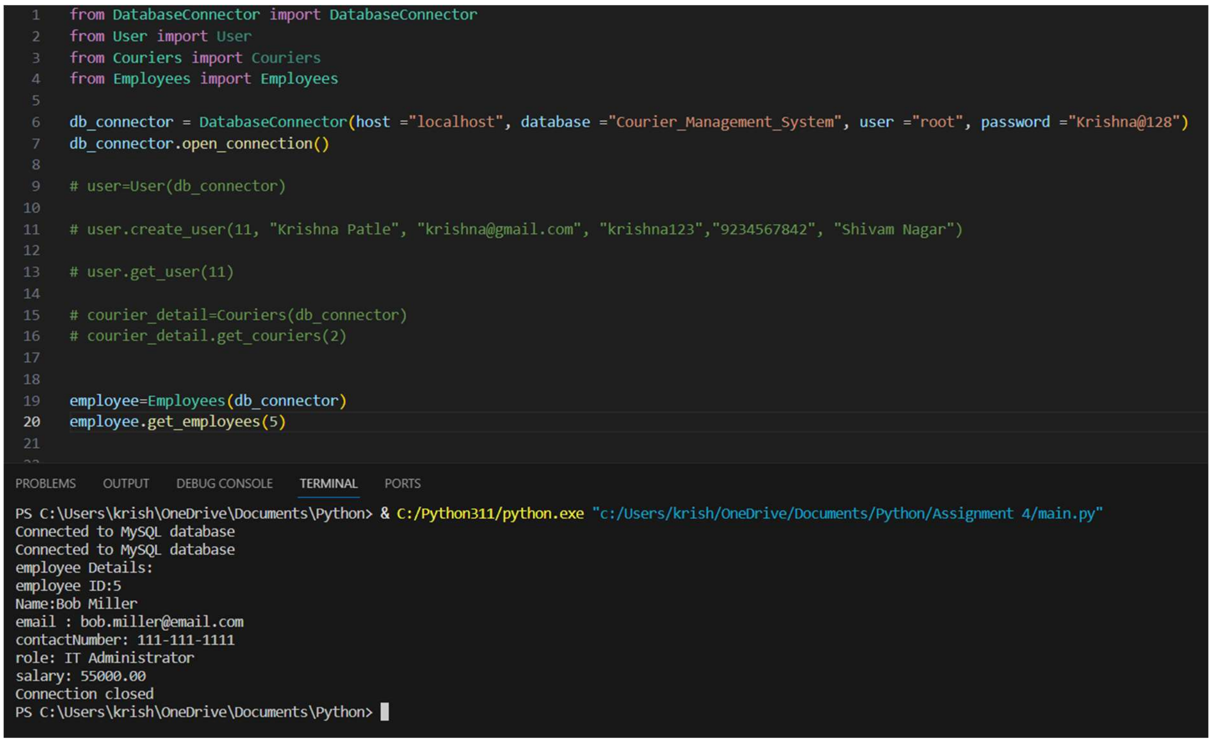
```
Connected to MySQL database
Error getting Courier details: 1054 (42S22): Unknown column 'userID' in 'where clause'
Connection closed
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents/Python/Assignment 4/main.py"
Connected to MySQL database
Connected 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



```python
class Employees:
    def __init__(self,employeeID, Name, email, contactNumber, role, salary):
        self.employeeID = employeeID
        self.Name = Name
        self.email = email
        self.contactNumber = contactNumber
        self.role = role
        self.salary = salary

    def __init__(self, db_connector):
        self._db_connector = db_connector

    def get_employees(self,employeeID):
        try:
            self._db_connector.open_connection()
            query = "SELECT * FROM employees where employeeID=%s "
            values=(employeeID,)
            self._db_connector.cursor.execute(query, values)
            employee_details = self._db_connector.cursor.fetchone()

            if employee_details:
                print("employee Details:")
                print(f"employee ID:{employee_details[0]}")
                print(f"Name:{employee_details[1]}")
                print(f"email : {employee_details[2]}")
                print(f"contactNumber: {employee_details[3]}")
                print(f"role: {employee_details[4]}")
                print(f"salary: {employee_details[5]}")

            else:
                print("Employee Id not found.")

```

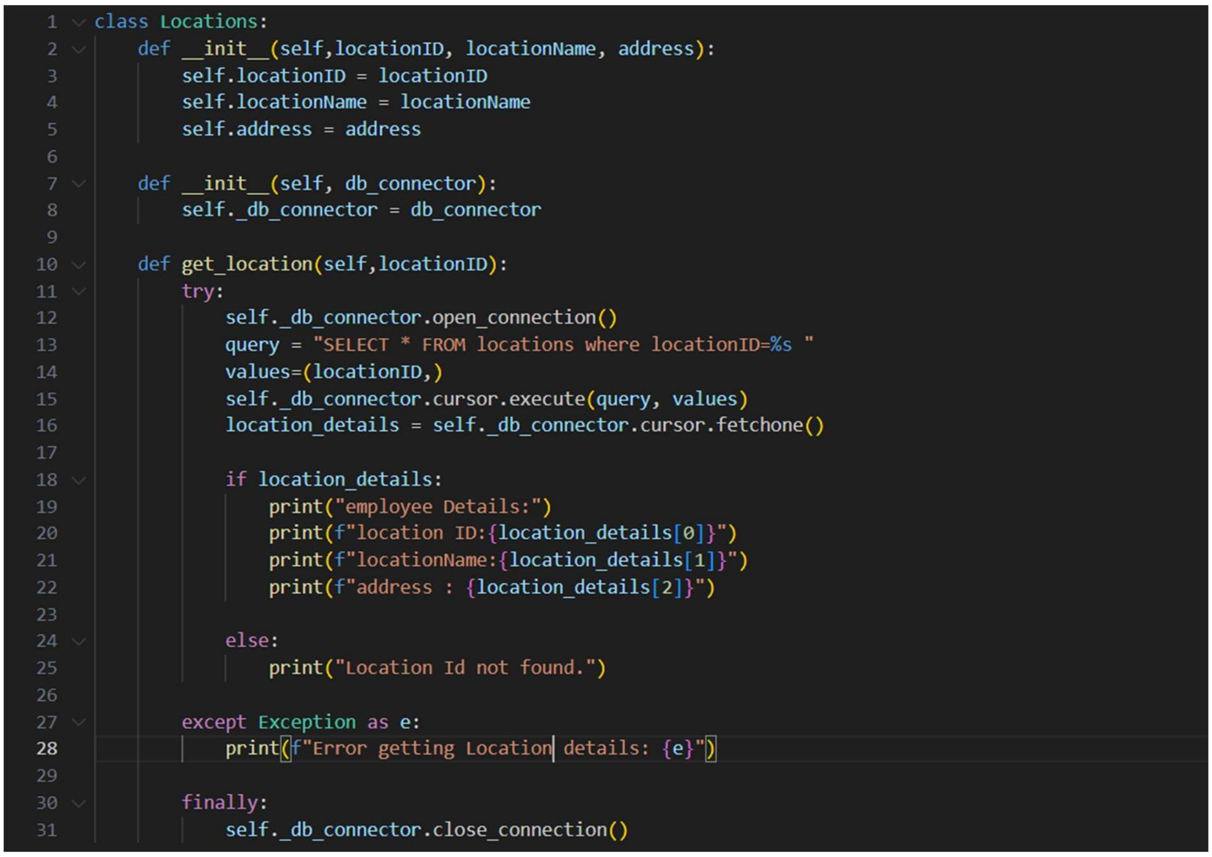


```python
from DatabaseConnector import DatabaseConnector
from User import User
from Couriers import Couriers
from Employees import Employees

db_connector = DatabaseConnector(host ="localhost", database ="Courier_Management_System", user ="root", password ="Krishna@128")
db_connector.open_connection()

# user=User(db_connector)

# user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123","9234567842", "Shivam Nagar")

# user.get_user(11)

# courier_detail=Couriers(db_connector)
# courier_detail.get_couriers(2)


employee=Employees(db_connector)
employee.get_employees(5)
```

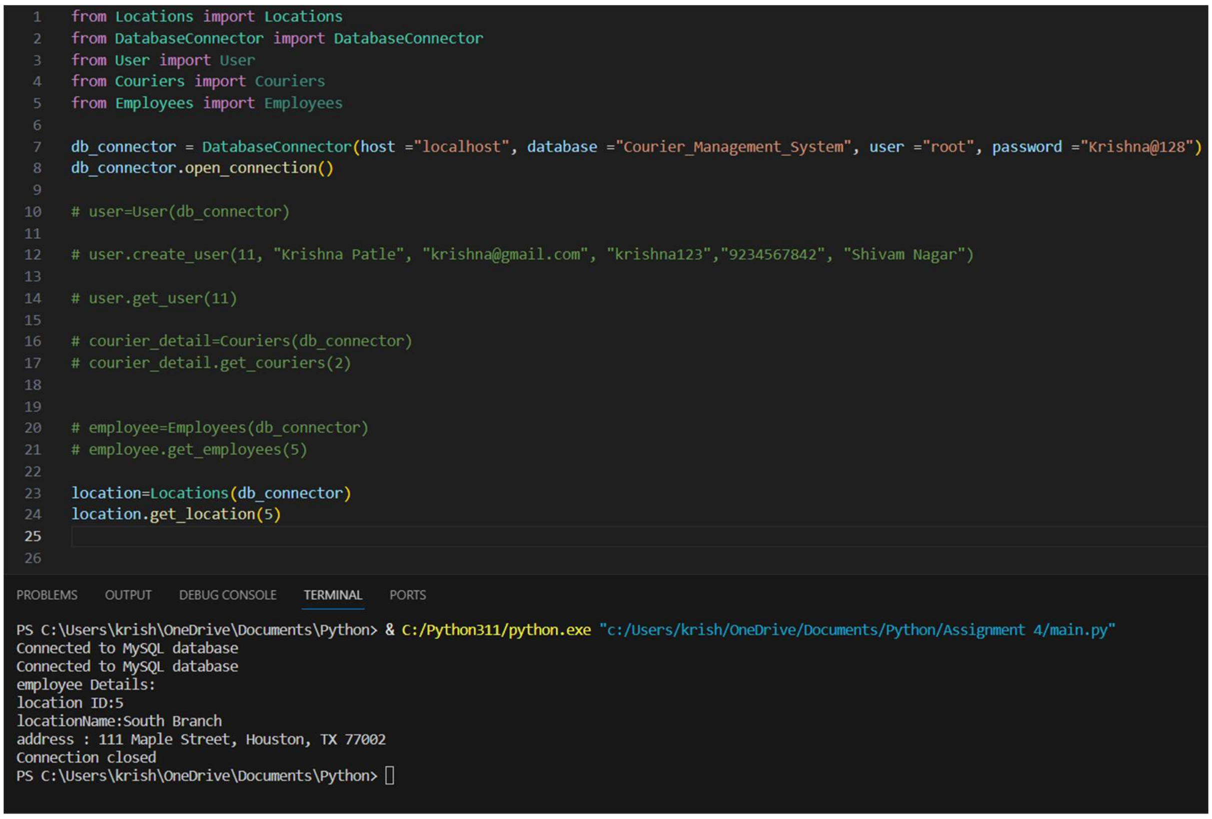
```
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents/Python/Assignment 4/main.py"
Connected to MySQL database
Connected to MySQL database
employee Details:
employee ID:5
Name:Bob Miller
email : bob.miller@email.com
contactNumber: 111-111-1111
role: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

4. Location Class

Variables LocationID , LocationName , Address



```python
class Locations:
    def __init__(self,locationID, locationName, address):
        self.locationID = locationID
        self.locationName = locationName
        self.address = address

    def __init__(self, db_connector):
        self._db_connector = db_connector

    def get_location(self,locationID):
        try:
            self._db_connector.open_connection()
            query = "SELECT * FROM locations where locationID=%s "
            values=(locationID,)
            self._db_connector.cursor.execute(query, values)
            location_details = self._db_connector.cursor.fetchone()

            if location_details:
                print("employee Details:")
                print(f"location ID:{location_details[0]}")
                print(f"locationName:{location_details[1]}")
                print(f"address : {location_details[2]}")

            else:
                print("Location Id not found.")

        except Exception as e:
            print(f"Error getting Location details: {e}")

        finally:
            self._db_connector.close_connection()
```

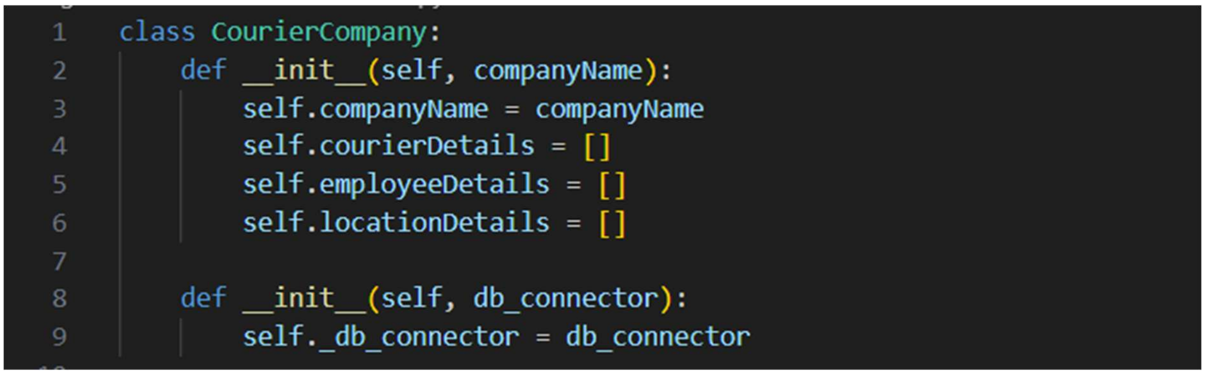


```python
from Locations import Locations
from DatabaseConnector import DatabaseConnector
from User import User
from Couriers import Couriers
from Employees import Employees

db_connector = DatabaseConnector(host ="localhost", database ="Courier_Management_System", user ="root", password ="Krishna@128")
db_connector.open_connection()

# user=User(db_connector)

# user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123","9234567842", "Shivam Nagar")

# user.get_user(11)

# courier_detail=Couriers(db_connector)
# courier_detail.get_couriers(2)


# employee=Employees(db_connector)
# employee.get_employees(5)

location=Locations(db_connector)
location.get_location(5)

```

```
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

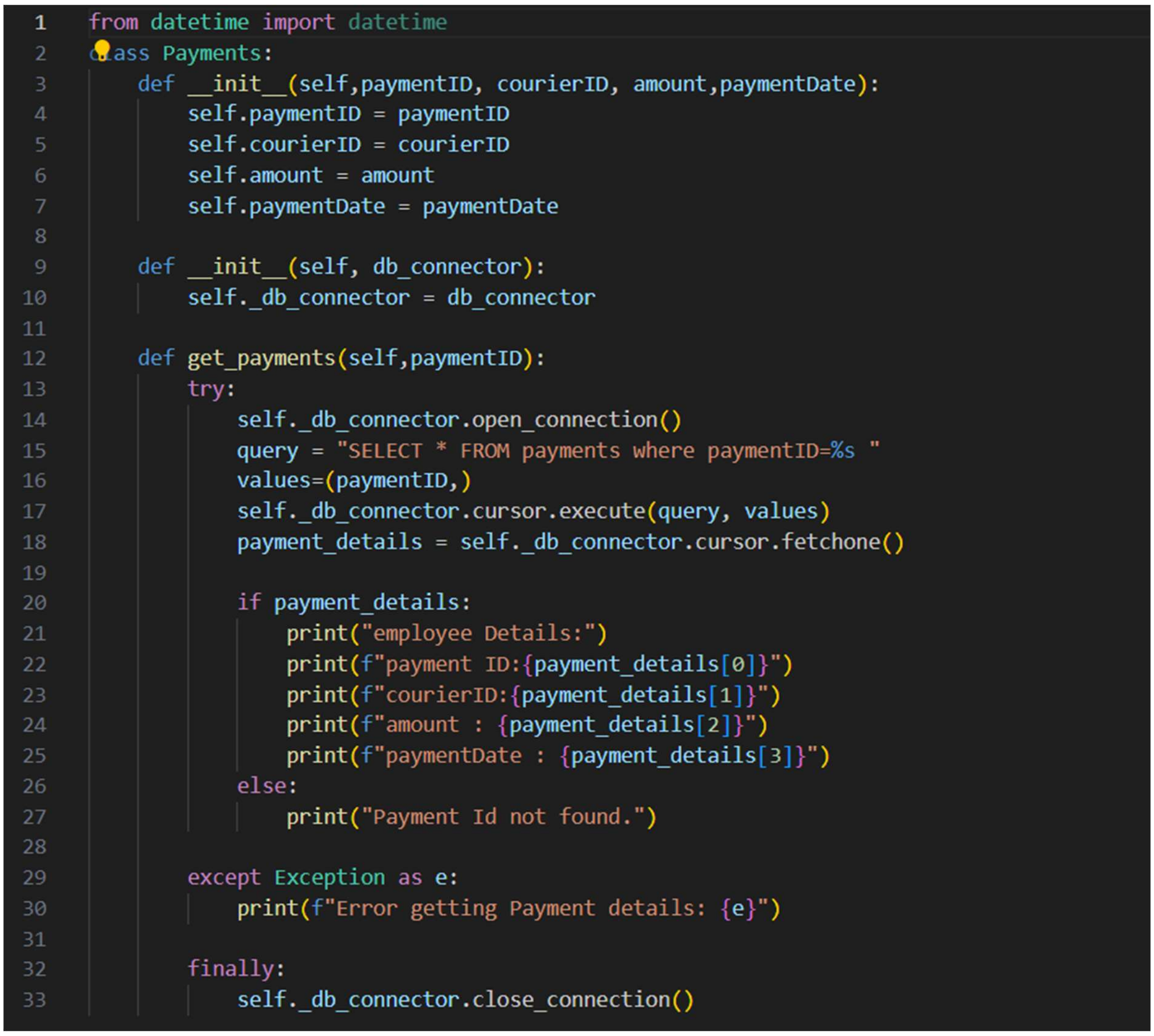
Variables companyName , courierDetails -collection of Courier Objects, employeeDetailscollection of Employee Objects, locationDetails - collection of Location Objects.



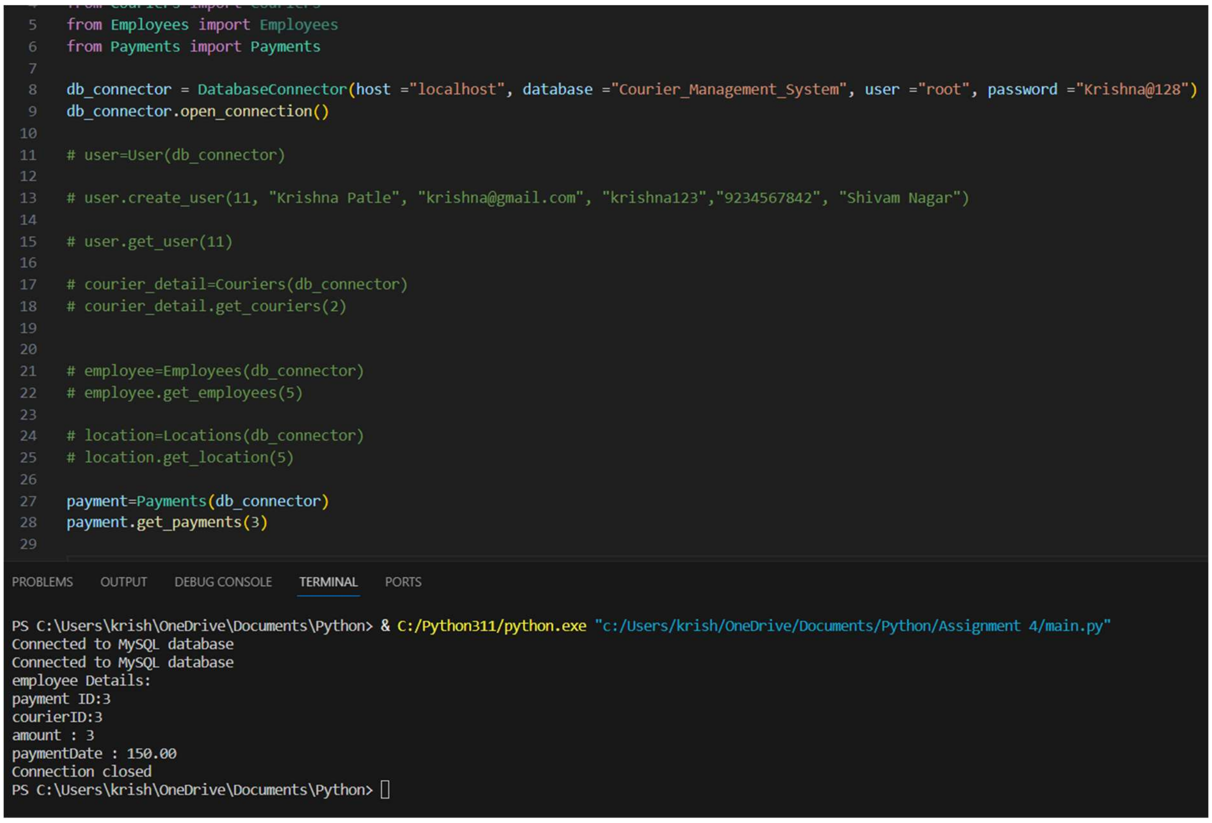
```python
class CourierCompany:
    def __init__(self, companyName):
        self.companyName = companyName
        self.courierDetails = []
        self.employeeDetails = []
        self.locationDetails = []

    def __init__(self, db_connector):
        self._db_connector = db_connector
```

6. Payment Class:

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

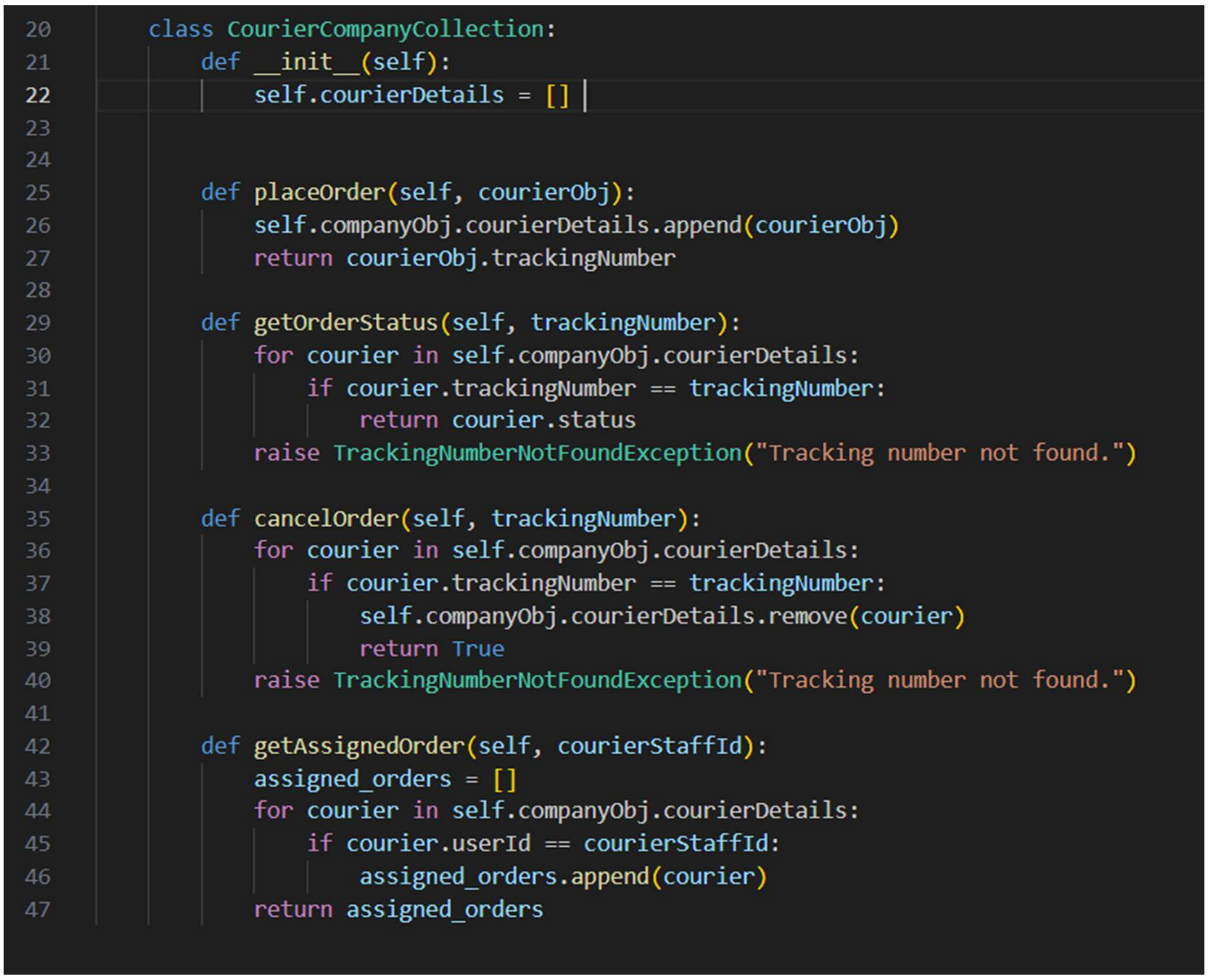


```python
from datetime import datetime
class Payments:
    def __init__(self,paymentID, courierID, amount,paymentDate):
        self.paymentID = paymentID
        self.courierID = courierID
        self.amount = amount
        self.paymentDate = paymentDate

    def __init__(self, db_connector):
        self._db_connector = db_connector

    def get_payments(self,paymentID):
        try:
            self._db_connector.open_connection()
            query = "SELECT * FROM payments where paymentID=%s "
            values=(paymentID,)
            self._db_connector.cursor.execute(query, values)
            payment_details = self._db_connector.cursor.fetchone()

            if payment_details:
                print("employee Details:")
                print(f"payment ID:{payment_details[0]}")
                print(f"courierID:{payment_details[1]}")
                print(f"amount : {payment_details[2]}")
                print(f"paymentDate : {payment_details[3]}")
            else:
                print("Payment Id not found.")

        except Exception as e:
            print(f"Error getting Payment details: {e}")

        finally:
            self._db_connector.close_connection()
```



```python
from Employees import Employees
from Payments import Payments

db_connector = DatabaseConnector(host ="localhost", database ="Courier_Management_System", user ="root", password ="Krishna@128")
db_connector.open_connection()

# user=User(db_connector)

# user.create_user(11, "Krishna Patle", "krishna@gmail.com", "krishna123","9234567842", "Shivam Nagar")

# user.get_user(11)

# courier_detail=Couriers(db_connector)
# courier_detail.get_couriers(2)


# employee=Employees(db_connector)
# employee.get_employees(5)

# location=Locations(db_connector)
# location.get_location(5)

payment=Payments(db_connector)
payment.get_payments(3)

```

```
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> 
```


```python
class CourierCompanyCollection:
    def __init__(self):
        self.courierDetails = []


    def placeOrder(self, courierObj):
        self.companyObj.courierDetails.append(courierObj)
        return courierObj.trackingNumber

    def getOrderStatus(self, trackingNumber):
        for courier in self.companyObj.courierDetails:
            if courier.trackingNumber == trackingNumber:
                return courier.status
        raise TrackingNumberNotFoundException("Tracking number not found.")

    def cancelOrder(self, trackingNumber):
        for courier in self.companyObj.courierDetails:
            if courier.trackingNumber == trackingNumber:
                self.companyObj.courierDetails.remove(courier)
                return True
        raise TrackingNumberNotFoundException("Tracking number not found.")

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