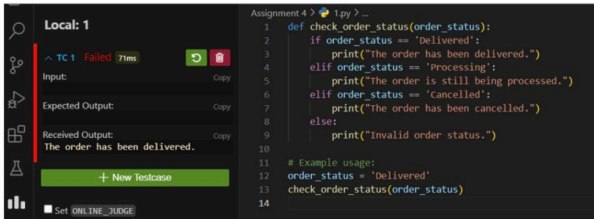


## Assignment 4

### 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.

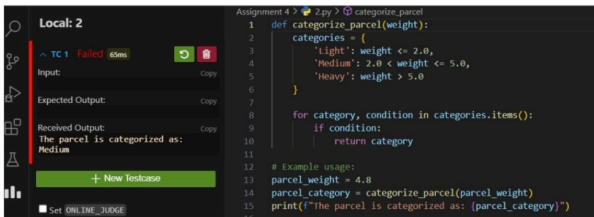


The screenshot shows a coding environment with a test case runner on the left and a code editor on the right. The test case runner shows a failed test case (TC 1) with a duration of 71ms. The input is empty, the expected output is empty, and the received output is "The order has been delivered." The code editor shows a Python function `check_order_status` that uses if-elif-else statements to print the status of an order. The function is called with `order_status = 'Delivered'`.

```
Local: 1
^ TC 1 Failed 71ms
Input:
Expected Output:
Received Output:
The order has been delivered.
+ New Testcase
Set ONLINE_JUDGE

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."



The screenshot shows a coding environment with a test case runner on the left and a code editor on the right. The test case runner shows a failed test case (TC 1) with a duration of 65ms. The input is empty, the expected output is empty, and the received output is "The parcel is categorized as: Medium". The code editor shows a Python function `categorize_parcel` that uses a dictionary to categorize parcels based on their weight. The function is called with `parcel_weight = 4.8`.

```
Local: 2
^ TC 1 Failed 65ms
Input:
Expected Output:
Received Output:
The parcel is categorized as:
Medium
+ New Testcase
Set ONLINE_JUDGE

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.

```

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[0].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)

```

Local: 3  
 TC1 Failed 7ms  
 Input: admin, john, password123  
 Expected Output: Enter user type (employees/customers): Enter username: Enter password: Invalid user type.  
 Received Output: Enter user type (employees/customers): Enter username: Enter password: Invalid user type.

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.

```

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

```

Local: 4  
 TC1 Failed 71ms  
 Input: (empty)  
 Expected Output: (empty)  
 Received Output: Shipment1 assigned to Courier1, Shipment2 assigned to Courier2, Shipment3 assigned to Courier3, Shipment4 assigned to Courier4

## Task 2: Loops and Iteration

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

```

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)
23

```

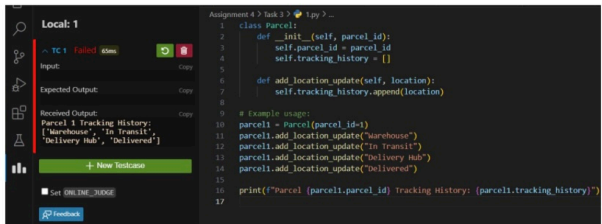
Local: 1  
 TC1 Failed 35ms  
 Input: 1  
 Expected Output: (empty)  
 Received Output: Enter customer ID: Orders for Customer 1, Order ID: 1, Status: Delivered, Order ID: 3, Status: In Transit, Order ID: 7, Status: In Transit, Order ID: 10, Status: Pickup

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.



The screenshot shows a code editor with a Python class named `Parcel`. The class has an `__init__` method that initializes `parcel_id` and `tracking_history` (an empty list). It also has an `add_location_update` method that appends a location to the `tracking_history` list. The example usage shows creating a `Parcel` object and adding four location updates: "Warehouse", "In Transit", "Delivery Hub", and "Delivered". The output shows the tracking history as a list of these locations.

```
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
```

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.

**Local: 1**

TC 1 Failed

Input: ABC123

Expected Output:

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

+ New Testcase

Set ONLINE\_JUDGE

Feedback

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

```

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

```

Local: 2
TC1 Failed
Input:
Enter the 1234567890
Expected Output:
Received Output:
Enter customer name: Name is valid: false
Enter customer address: Address is valid:
Enter customer phone number: 000-000-0000
Phone number is valid: True
+ New Instance
Set ONLINE_MODE
Assignment 4 Task 3 Zpp 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[0] == '0' and data[1].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 (000-000-0000): ")
19 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
TC1 Failed
Input:
Bhamburda Road
Mugner
Maharashtra
440035
Expected Output:
Received Output:
Enter street: Enter city: Enter state:
Enter zip code: Formatted Address:
Bhamburda Road, Mugner, Maharashtra 440035
Assignment 4 Task 4 Zpp
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

```

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
TC1 Failed
Input:
12345
Bhamburda
Mugner
2023-12-31
Expected Output:
Received Output:
Enter customer name: Enter order number:
Enter delivery address: Enter expected delivery date: Order Confirmation Email:
Dear Customer,
Thank you for your order!
Order Number: 12345
Delivery Address: Bhamburda Mugner
Expected Delivery Date: 2023-12-31
Best regards,
The E-commerce Team
Assignment 4 Task 5 Zpp
1 def generate_order_confirmation_email(customer_name, order_number, delivery_address, delivery_date):
2     email_body = f'''Thank you for your order! Order Number: {order_number} Delivery Address: {delivery_address} 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:", confirmation_email)
12

```

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
TC1 Failed
Input:
source
mugner
1
Expected Output:
Received Output:
Enter source address: Enter destination address: Enter parcel weight (in kg):
Shipping Cost: 0.5
Assignment 4 Task 6 Zpp
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

```

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 Jupyter Notebook interface. On the left, the 'Local: 6' tab is active, displaying the 'Input' field with the text 'Expected Output:'. Below it, the 'Received Output' field shows 'Generated Password: „Fh0uG&Bq“'. A green button labeled '+ New Notebook' is at the bottom. On the right, the code editor shows the following Python code:

```

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 Jupyter Notebook interface. On the left, the 'Local: 7' tab is active, displaying the 'Input' field with the text 'Expected Output:'. Below it, the 'Received Output' field shows 'Enter address to find: Similar Addresses: ["456 Park Avenue", "4567 Birch Drive"]'. A green button labeled '+ New Notebook' is at the bottom. On the right, the code editor shows the following Python code:

```

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", "5678 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

The screenshot shows a Python code editor with the following class definition:

```

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@123")
5 db_connector.open_connection()
6
7 user = User(db_connector)
8
9 user.create_user(1, "Krishna Patel", "krishna@gmail.com", "krishna123", "9334567842", "Shivan Nagar")
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

DatabaseConnector.py

main.py

user.py

task1

task2

task3

task4

task5

task6

task7

task8

task9

task10

task11

task12

task13

task14

task15

task16

task17

task18

task19

task20

task21

task22

task23

task24

task25

task26

task27

task28

task29

task30

task31

task32

task33

task34

task35

task36

task37

task38

task39

task40

task41

task42

task43

task44

task45

task46

task47

task48

task49

task50

task51

task52

task53

task54

task55

task56

task57

task58

task59

task60

task61

task62

task63

task64

task65

task66

task67

task68

task69

task70

task71

task72

task73

task74

task75

task76

task77

task78

task79

task80

task81

task82

task83

task84

task85

task86

task87

task88

task89

task90

task91

task92

task93

task94

task95

task96

task97

task98

task99

task100

task101

task102

task103

task104

task105

task106

task107

task108

task109

task110

task111

task112

task113

task114

task115

task116

task117

task118

task119

task120

task121

task122

task123

task124

task125

task126

task127

task128

task129

task130

task131

task132

task133

task134

task135

task136

task137

task138

task139

task140

task141

task142

task143

task144

task145

task146

task147

task148

task149

task150

task151

task152

task153

task154

task155

task156

task157

task158

task159

task160

task161

task162

task163

task164

task165

task166

task167

task168

task169

task170

task171

task172

task173

task174

task175

task176

task177

task178

task179

task180

task181

task182

task183

task184

task185

task186

task187

task188

task189

task190

task191

task192

task193

task194

task195

task196

task197

task198

task199

task200

task201

task202

task203

task204

task205

task206

task207

task208

task209

task210

task211

task212

task213

task214

task215

task216

task217

task218

task219

task220

task221

task222

task223

task224

task225

task226

task227

task228

task229

task230

task231

task232

task233

task234

task235

task236

task237

task238

task239

task240

task241

task242

task243

task244

task245

task246

task247

task248

task249

task250

task251

task252

task253

task254

task255

task256

task257

task258

task259

task260

task261

task262

task263

task264

task265

task266

task267

task268

task269

task270

task271

task272

task273

task274

task275

task276

task277

task278

task279

task280

task281

task282

task283

task284

task285

task286

task287

task288

task289

task290

task291

task292

task293

task294

task295

task296

task297

task298

task299

task300

task301

task302

task303

task304

task305

task306

task307

task308

task309

task310

task311

task312

task313

task314

task315

task316

task317

task318

task319

task320

task321

task322

task323

task324

task325

task326

task327

task328

task329

task330

task331

task332

task333

task334

task335

task336

task337

task338

task339

task340

task341

task342

task343

task344

task345

task346

task347

task348

task349

task350

task351

task352

task353

task354

task355

task356

task357

task358

task359

task360

task361

task362

task363

task364

task365

task366

task367

task368

task369

task370

task371

task372

task373

task374

task375

task376

task377

task378

task379

task380

task381

task382

task383

task384

task385

task386

task387

task388

task389

task390

task391

task392

task393

task394

task395

task396

task397

task398

task399

task400

task401

task402

task403

task404

task405

task406

task407

task408

task409

task410

task411

task412

task413

task414

task415

task416

task417

task418

task419

task420

task421

task422

task423

task424

task425

task426

task427

task428

task429

task430

task431

task432

task433

task434

task435

task436

task437

task438

task439

task440

task441

task442

task443

task444

task445

task446

task447

task448

task449

task450

task451

task452

task453

task454

task455

task456

task457

task458

task459

task460

task461

task462

task463

task464

task465

task466

task467

task468

task469

task470

task471

task472

task473

task474

task475

task476

task477

task478

task479

task480

task481

task482

task483

task484

task485

task486

task487

task488

task489

task490

task491

task492

task493

task494

task495

task496

task497

task498

task499

task500

task501

task502

task503

task504

task505

task506

task507

task508

task509

task510

task511

task512

task513

task514

task515

task516

task517

task518

task519

task520

task521

task522

task523

task524

task525

task526

task527

task528

task529

task530

task531

task532

task533

task534

task535

task536

task537

task538

task539

task540

task541

task542

task543

task544

task545

task546

task547

task548

task549

task550

task551

task552

task553

task554

task555

task556

task557

task558

task559

task560

task561

task562

task563

task564

task565

task566

task567

task568

task569

task570

task571

task572

task573

task574

task575

task576

task577

task578

task579

task580

task581

task582

task583

task584

task585

task586

task587

task588

task589

task590

task591

task592

task593

task594

task595

task596

task597

task598

task599

task600

task601

task602

task603

task604

task605

task606

task607

task608

task609

task610

task611

task612

task613

task614

task615

task616

task617

task618

task619

task620

task621

task622

task623

task624

task625

task626

task627

task628

task629

task630

task631

task632

task633

task634

task635

task636

task637

task638

task639

task640

task641

task642

task643

task644

task645

task646

task647

task648

task649

task650

task651

task652

task653

task654

task655

task656

task657

task658

task659

task660

task661

task662

task663

task664

task665

task666

task667

task668

task669

task670

task671

task672

task673

task674

task675

task676

task677

task678

task679

task680

task681

task682

task683

task684

task685

task686

task687

task688

task689

task690

task691

task692

task693

task694

task695

task696

task697

task698

task699

task700

task701

task702

task703

task704

task705

task706

task707

task708

task709

task710

task711

task712

task713

task714

task715

task716

task717

task718

task719

task720

task721

task722

task723

task724

task725

task726

task727

task728

task729

task730

task731

task732

task733

task734

task735

task736

task737

task738

task739

task740

task741

task742

task743

task744

task745

task746

task747

task748

task749

task750

task751

task752

task753

task754

task755

task756

task757

task758

task759

task760

task761

task762

task763

task764

task765

task766

task767

task768

task769

task770

task771

task772

task773

task774

task775

task776

task777

task778

task779

task780

task781

task782

task783

task784

task785

task786

task787

task788

task789

task790

task791

task792

task793

task794

task795

task796

task797

task798

task799

task800

task801

task802

task803

task804

task805

task806

task807

task808

task809

task810

task811

task812

task813

task814

task815

task816

task817

task818

task819

task820

task821

task822

task823

task824

task825

task826

task827

task828

task829

task830

task831

task832

task833

task834

task835

task836

task837

task838

task839

task840

task841

task842

task843

task844

task845

task846

task847

task848

task849

task850

task851

task852

task853

task854

task855

task856

task857

task858

task859

task860

task861

task862

task863

task864

task865

task866

task867

task868

task869

task870

task871

task872

task873

task874

task875

task876

task877

task878

task879

task880

task881

task882

task883

task884

task885

task886

task887

task888

task889

task890

task891

task892

task893

task894

task895

task896

task897

task898

task899

task900

task901

task902

task903

task904

task905

task906

task907

task908

task909

task910

task911

task912

task913

task914

task915

task916

task917

task918

task919

task920

task921

task922

task923

task924

task925

task926

task927

task928

task929

task930

task931

task932

task933

task934

task935

task936

task937

task938

task939

task940

task941

task942

task943

task944

task945

task946

task947

task948

task949

task950

task951

task952

task953

task954

task955

task956

task957

task958

task959

task960

task961

task962

task963

task964

task965

task966

task967

task968

task969

task970

task971

task972

task973

task974

task975

task976

task977

task978

task979

task980

task981

task982

task983

task984

task985

task986

task987

task988

task989

task990

task991

task992

task993

task994

task995

task996

task997

task998

task999

task1000

## 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@123")
6 db_connector.open_connection()
7
8 # user=user(db_connector)
9
10 # user.create_user(11, "Krishna Patel", "krishna@gmail.com", "krishna123", "9216567812", "Shivan Nagar")
11
12 # user.get_user(11)
13
14 courier_detail=Couriers(db_connector)
15 courier_detail.get_couriers()
16

```

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: 456 Oak Lane
weight: 2.50
status: Picked
trackingNumber: TRF456
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@gmail.com
contactNumber: 111-111-1111
role: IT Administrator
salary: $5000.00
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()

```

```

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

```