PIYUSH KUMAR MISHRA

230957212

ROLL N0:70

WEEK 5:

Q1:Exercise 1: Student Grades Management System

Create a simple student grades management system which perform the following functions

(Use a dictionary where the keys are student names, and the values are lists of grades.):

• Add a student: Add a student's name and their grades for multiple subjects.

• Update a Grade: Update a specific grade for a student in each subject.

• Remove a student: Remove a student from the system.

• Get Average Grade: Calculate and return the average grade for a student across all subjects.

• Get Subject Average: Calculate and return the average grade for a specific subject.

• List All Students: List all students with their average grades for each subject and overall.

• Get Highest Grade: Find the highest grade in a specific subject

```
class StudentGradesSystem:
    def __init__(self):
        self.students = {}
```

```
def add_student(self, name, grades):
    self.students[name] = grades
    print(f"Student {name} added with grades: {grades}")

def update_grade(self, name, subject_index, new_grade):
    if name in self.students:
        if 0 <= subject_index < len(self.students[name]):
            self.students[name][subject_index] = new_grade
            print(f"Grade updated for {name} in subject {subject_index + 1}: {new_grade}")
        else:
            print(f"Subject index {subject_index} is invalid.")
    else:
        print(f"Student {name} not found.")

def remove_student(self, name):
    if name in self.students:
        del self.students[name]
        print(f"Student {name} removed.")
    else:
        print(f"Student {name} not found.")

def get_average_grade(self, name):
    if name in self.students:
        grades = self.students[name]
        avg = sum(grades) / len(grades) if grades else 0
        return f"Average grade for {name}: {avg:.2f}"
    else:
```

```
        return f"Student {name} not found."

    def get_subject_average(self, subject_index):
        total, count = 0, 0
        for grades in self.students.values():
            if 0 <= subject_index < len(grades):
                total += grades[subject_index]
                count += 1
        if count > 0:
            return f"Average grade for subject {subject_index + 1}: {total / count:.2f}"
        else:
            return f"No data available for subject {subject_index + 1}."

    def list_all_students(self):
        if self.students:
            for name, grades in self.students.items():
                overall_avg = sum(grades) / len(grades) if grades else 0
                print(f"{name}: Grades: {grades} | Overall Average: {overall_avg:.2f}")
        else:
            print("No students available.")

    def get_highest_grade(self, subject_index):
        highest_grade = -1
        highest_student = None
        for name, grades in self.students.items():
            if 0 <= subject_index < len(grades):
                if grades[subject_index] > highest_grade:
```

```
                highest_grade = grades[subject_index]
                highest_student = name
        if highest_student:
            return f"Highest grade in subject {subject_index + 1}: {highest_grade} by {highest_student}"
        else:
            return f"No data available for subject {subject_index + 1}."

system = StudentGradesSystem()
system.add_student("Alice", [85, 90, 78])
system.add_student("Bob", [88, 76, 92])
system.add_student("Charlie", [95, 85, 80])

system.update_grade("Alice", 1, 95)
system.remove_student("Bob")
print(system.get_average_grade("Alice"))
print(system.get_subject_average(1))
system.list_all_students()
print(system.get_highest_grade(2))
```

OUTPUT:

```
Student Alice added with grades: [85, 90, 78]
Student Bob added with grades: [88, 76, 92]
Student Charlie added with grades: [95, 85, 80]
Grade updated for Alice in subject 2: 95
Student Bob removed.
Average grade for Alice: 86.00
Average grade for subject 2: 90.00
```

Alice: Grades: [85, 95, 78] | Overall Average: 86.00

Charlie: Grades: [95, 85, 80] | Overall Average: 86.67

Highest grade in subject 3: 80 by Charlie

Q2)Exercise 2: Employee Management System

Implement Employee Management System using nested dictionaries and lists and implement following functions to handle different operations.

- **add_employee(): Adds a new employee or updates an existing employee's details.**
- **update_salary(): Updates the salary of an existing employee.**
- **add_performance_score(): Adds a performance score to an employee's record.**
- **remove_employee(): Removes an employee from the records.**
- **get_average_salary_by_department(): Computes the average salary of employees in a specified department.**
- **get_employee_with_highest_performance(): Finds the employee with the highest average performance score.**
- **list_employees_by_department(): Lists all employees in a specified department.**

```
class EmployeeManagementSystem:
    def __init__(self):
        self.employees = {}

    def add_employee(self, employee_id, name, department, salary, performance_scores=None):
        if performance_scores is None:
            performance_scores = []
        self.employees[employee_id] = {
            "name": name,
            "department": department,
```

```
            "salary": salary,
            "performance_scores": performance_scores
        }
        print(f"Employee {name} added/updated.")

    def update_salary(self, employee_id, new_salary):
        if employee_id in self.employees:
            self.employees[employee_id]['salary'] = new_salary
            print(f"Salary updated for {self.employees[employee_id]['name']}: {new_salary}")
        else:
            print(f"Employee {employee_id} not found.")

    def add_performance_score(self, employee_id, score):
        if employee_id in self.employees:
            self.employees[employee_id]['performance_scores'].append(score)
            print(f"Performance score {score} added for {self.employees[employee_id]['name']}.")
        else:
            print(f"Employee {employee_id} not found.")

    def remove_employee(self, employee_id):
        if employee_id in self.employees:
            print(f"Employee {self.employees[employee_id]['name']} removed.")
            del self.employees[employee_id]
        else:
            print(f"Employee {employee_id} not found.")

    def get_average_salary_by_department(self, department):
```

```
    total_salary, count = 0, 0
    for employee in self.employees.values():
        if employee['department'] == department:
            total_salary += employee['salary']
            count += 1
    if count > 0:
        return f"Average salary in {department}: {total_salary / count:.2f}"
    else:
        return f"No employees found in {department}."

def get_employee_with_highest_performance(self):
    highest_avg = -1
    top_employee = None
    for employee_id, employee in self.employees.items():
        scores = employee['performance_scores']
        if scores:
            avg_score = sum(scores) / len(scores)
            if avg_score > highest_avg:
                highest_avg = avg_score
                top_employee = employee
    if top_employee:
        return f"Employee with highest performance: {top_employee['name']} with average score {highest_avg:.2f}"
    else:
        return "No performance scores available."

def list_employees_by_department(self, department):
    employees_in_dept = [emp['name'] for emp in self.employees.values() if emp['department'] == department]
```

```
        if employees_in_dept:
            print(f"Employees in {department}: {', '.join(employees_in_dept)}")
        else:
            print(f"No employees found in {department}.")

system = EmployeeManagementSystem()
system.add_employee(1, "Alice", "HR", 50000, [4.5, 4.7])
system.add_employee(2, "Bob", "IT", 60000, [4.0, 3.8])
system.add_employee(3, "Charlie", "HR", 55000, [4.8, 4.9])

system.update_salary(2, 65000)
system.add_performance_score(1, 4.6)
system.remove_employee(3)

print(system.get_average_salary_by_department("HR"))
print(system.get_employee_with_highest_performance())
system.list_employees_by_department("IT")
```

OUTPUT:

```
Student Alice added with grades: [85, 90, 78]
Student Bob added with grades: [88, 76, 92]
Student Charlie added with grades: [95, 85, 80]
Grade updated for Alice in subject 2: 95
Student Bob removed.
Average grade for Alice: 86.00
Average grade for subject 2: 90.00
Alice: Grades: [85, 95, 78] | Overall Average: 86.00
```

Charlie: Grades: [95, 85, 80] | Overall Average: 86.67

Highest grade in subject 3: 80 by Charlie

PIYUSH KUMAR MISHRA

230957212

ROLL N0:70

WEEK 6:

Exercise 1:

You are tasked with designing a Library Management System for a local library. The library has both EBooks (digital format) and Printed Books (physical copies), and the system should allow members (both students and teachers) to borrow books. The library also has a librarian who manages the addition and removal of books.

Your system should include the following features:

1. Book Management:

o Books can be either EBooks or Printed Books.

o Each book should have a title, author, and ISBN.

o EBooks should have a file format, while Printed Books should have a page count.

2. Member Management:

o The library has members who can either be students or teachers.

o Each member has a name and member ID.

o Members should be able to borrow books.

3. Librarian Management:

o A librarian can add or remove books from the library.

o A librarian can be both a student and a teacher.

4. Library Operations:

o The system should allow the librarian to:

- **Add new books to the library.**
- **Remove books from the library using their ISBN.**
- **Search for books by title or author.**

Requirements:

Using Python and Object-Oriented Programming principles, implement the following:

1. Create a class hierarchy to represent books (including EBooks and Printed Books).

2. Create a class hierarchy to represent members (students and teachers).

3. Implement the functionalities for adding, removing, and searching for books.

4. Demonstrate the following types of inheritance:

o Single Inheritance for books.

o Multiple Inheritance for the librarian, who is both a student and a teacher.

o Hierarchical Inheritance for members (students and teachers).

Tasks:

1. Book Management:

o Define a base class Book with attributes for title, author, and ISBN.

o Define a subclass EBook that adds the attribute for file format.

o Define another subclass PrintedBook that adds the attribute for page count.

2. Member and Librarian Management:

o Define a base class Member with attributes for name and member ID.

o Define two subclasses: Student and Teacher, which inherit from Member.

o Create a Librarian class that inherits from both Student and Teacher (multiple inheritance).

3. Library Class:

o Implement a Library class to manage the collection of books.

o Add methods to the Library class to:

- **Add new books.**
- **Remove a book by its ISBN.**
- **Search for books by title or author.**

4. Demonstration:

o Instantiate a library and add books (both EBooks and Printed Books) to it.

o Demonstrate searching for books using keywords.

o Show how a librarian can add and remove books from the system.

```
class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn

class EBook(Book):
    def __init__(self, title, author, isbn, file_format):
        super().__init__(title, author, isbn)
        self.file_format = file_format

class PrintedBook(Book):
    def __init__(self, title, author, isbn, page_count):
        super().__init__(title, author, isbn)
        self.page_count = page_count
class Member:
    def __init__(self, name, member_id):
        self.name = name
```

```
        self.member_id = member_id


class Student(Member):
    pass


class Teacher(Member):
    pass


class Librarian(Student, Teacher):
    pass
class Library:
    def __init__(self):
        self.books = []


    def add_book(self, book):
        self.books.append(book)


    def remove_book(self, isbn):
        for book in self.books:
            if book.isbn == isbn:
                self.books.remove(book)
                print(f"Book with ISBN {isbn} removed successfully.")
                return
        print(f"Book with ISBN {isbn} not found.")


    def search_book(self, keyword):
        results = []
```

```
        for book in self.books:
            if keyword in book.title or keyword in book.author:
                results.append(book)
        return results
def main():
    library = Library()

    while True:
        print("Library Management System")
        print("1. Add Book")
        print("2. Remove Book")
        print("3. Search Book")
        print("4. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            title = input("Enter book title: ")
            author = input("Enter book author: ")
            isbn = input("Enter book ISBN: ")
            book_type = input("Enter book type (EBook or PrintedBook): ")

            if book_type == "EBook":
                file_format = input("Enter file format: ")
                book = EBook(title, author, isbn, file_format)
            elif book_type == "PrintedBook":
                page_count = int(input("Enter page count: "))
```

```
            book = PrintedBook(title, author, isbn, page_count)

        library.add_book(book)
        print("Book added successfully.")

    elif choice == "2":
        isbn = input("Enter ISBN of book to remove: ")
        library.remove_book(isbn)

    elif choice == "3":
        keyword = input("Enter keyword to search: ")
        results = library.search_book(keyword)

        if results:
            print("Search results:")
            for book in results:
                print(f"Title: {book.title}, Author: {book.author}, ISBN: {book.isbn}")
        else:
            print("No books found.")

    elif choice == "4":
        break

    else:
        print("Invalid choice. Please try again.")

if __name__ == "__main__":
```

main()

OUTPUT:

Library Management System

1. Add Book

2. Remove Book

3. Search Book

4. Exit

Enter your choice: 1

Enter book title: x

Enter book author: y

Enter book ISBN: z

Enter book type (EBook or PrintedBook): EBook

Enter file format: pdf

Book added successfully.

Library Management System

1. Add Book

2. Remove Book

3. Search Book

4. Exit

Enter your choice:

↑↓ for history. Search history with c-↑/c-↓

Exercise 2: Advanced E-Commerce System Utilizing Polymorphism

A rapidly growing online retail company is looking to upgrade its E-Commerce System.

They need the system to manage various types of products, allow users to add them to their

shopping carts, apply discounts, and handle different payment methods. To make the system robust, flexible, and scalable, you decide to use Object-Oriented Programming (OOP) principles.

Objectives:

1. Product Management:

o Products in the system belong to different categories such as Electronics and Clothing. Each product category has its own discount logic.

o Discounts should be applied based on product types, demonstrating method overriding.

2. Shopping Cart Management:

o Users should be able to add multiple items to their shopping carts and see the total cost after discounts.

o The system should allow merging two shopping carts using operator overloading.

3. Payment Processing:

o Customers should be able to process payments using various methods (e.g., credit card, PayPal). Even though Python does not natively support method overloading, it should be simulated to handle different payment methods efficiently.

Functional Requirements:

1. Products:

o Implement a base Product class that represents general products, containing attributes like name and price.

o Create derived classes such as Electronics and Clothing that override the base class method for calculating product discounts.

2. Shopping Cart:

o Implement a ShoppingCart class that can hold a collection of products.

o Overload the + operator to merge two shopping carts into one.

3. Payment Processing:

o Implement a PaymentProcessor class that simulates method overloading to handle different payment methods (e.g., credit card and PayPal) using variable arguments.

```
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def calculate_discount(self):
        return self.price * 0.1  # 10% discount as a placeholder

class Electronics(Product):
    def calculate_discount(self):
        return self.price * 0.15  # 15% discount for Electronics

class Clothing(Product):
    def calculate_discount(self):
        return self.price * 0.05  # 5% discount for Clothing
class ShoppingCart:
    def __init__(self, products=None):
        self.products = products or []

    def add_product(self, product):
        self.products.append(product)
```

```
    def calculate_total(self):
        total = sum(product.price for product in self.products)
        discount = sum(product.calculate_discount() for product in self.products)
        return total - discount

    def __add__(self, other):
        combined_cart = ShoppingCart()
        combined_cart.products = self.products + other.products
        return combined_cart
class PaymentProcessor:
    def process_payment(self, payment_method, *args, **kwargs):
        if payment_method == "credit_card":
            self.process_credit_card_payment(*args, **kwargs)
        elif payment_method == "paypal":
            self.process_paypal_payment(*args, **kwargs)
        else:
            raise ValueError(f"Unsupported payment method: {payment_method}")

    def process_credit_card_payment(self, amount, card_number, expiration_date, cvv):
        # Implement credit card payment processing
        pass

    def process_paypal_payment(self, amount, paypal_email, paypal_password):
        # Implement PayPal payment processing
        pass
def main():
    electronics = Electronics("Smartphone", 500)
```

```
    clothing = Clothing("T-Shirt", 20)

    cart1 = ShoppingCart()
    cart1.add_product(electronics)
    cart1.add_product(clothing)

    cart2 = ShoppingCart()
    cart2.add_product(electronics)

    print("Cart 1 total:", cart1.calculate_total())
    print("Cart 2 total:", cart2.calculate_total())

    merged_cart = cart1 + cart2
    print("Merged cart total:", merged_cart.calculate_total())

    payment_processor = PaymentProcessor()
    payment_processor.process_payment("credit_card", 500, "1234567890123456", "123", "2025")
    payment_processor.process_payment("paypal", 500, "john.doe@example.com", "secret_password")

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

OUTPUT:

Cart 1 total: 444.0

Cart 2 total: 425.0

Merged cart total: 869.0

PIYUSH KUMAR MISHRA

230957212

ROLL N0:70

WEEK 7:

Exercise 1:

Inventory Management System using Python Inheritance

Scenario: Designing an Inventory Management System

You are tasked with designing an Inventory Management System that handles various types of products. Each product shares common attributes but has specific attributes based on the category of the product. The system must support operations such as adding products, calculating inventory value, applying discounts, and checking stock levels.

Problem Definition:

We need to manage three types of products:

1. Electronics (e.g., phones, laptops).

2. Clothing (e.g., shirts, pants).

3. Groceries (e.g., fruits, vegetables).

All product types share basic attributes like name, price, quantity, and SKU (Stock Keeping Unit). However, they also have specific attributes:

- **Electronics may have warranty period and brand.**
- **Clothing has attributes like size and material.**
- **Groceries include expiration date and organic status.**

Additionally, the system needs to:

1. Add new products.

2. Update stock.

3. Calculate the total value of inventory.

4. Apply discounts based on product type.

5. Check if products are low in stock.

```
class Product:
    def __init__(self, name, price, quantity, sku):
        self.name = name
        self.price = price
        self.quantity = quantity
        self.sku = sku

    def calculate_value(self):
        return self.price * self.quantity

    def is_low_stock(self, threshold=5):
        return self.quantity < threshold


class Electronics(Product):
    def __init__(self, name, price, quantity, sku, warranty_period, brand):
        super().__init__(name, price, quantity, sku)
        self.warranty_period = warranty_period
        self.brand = brand
```

```
class Clothing(Product):
    def __init__(self, name, price, quantity, sku, size, material):
        super().__init__(name, price, quantity, sku)
        self.size = size
        self.material = material


class Groceries(Product):
    def __init__(self, name, price, quantity, sku, expiration_date, organic_status):
        super().__init__(name, price, quantity, sku)
        self.expiration_date = expiration_date
        self.organic_status = organic_status


class Inventory:
    def __init__(self):
        self.products = {}

    def add_product(self, product):
        if product.sku in self.products:
            print("Product with this SKU already exists.")
        else:
            self.products[product.sku] = product
            print(f"Added {product.name} to inventory.")

    def update_stock(self, sku, quantity):
        if sku in self.products:
```

```
            self.products[sku].quantity += quantity
            print(f"Updated stock for {sku}. New quantity: {self.products[sku].quantity}.")
        else:
            print("Product not found.")

    def calculate_total_value(self):
        total_value = sum(product.calculate_value() for product in self.products.values())
        return total_value

    def apply_discount(self, sku, discount_percentage):
        if sku in self.products:
            product = self.products[sku]
            discount_amount = product.price * (discount_percentage / 100)
            product.price -= discount_amount
            print(f"Applied discount to {product.name}. New price: {product.price:.2f}.")
        else:
            print("Product not found.")

    def check_low_stock(self):
        low_stock_products = [product for product in self.products.values() if product.is_low_stock()]
        return low_stock_products


# Example Usage with user-defined input
if __name__ == "__main__":
    inventory = Inventory()
```

```
while True:
    print("\n1. Add Product")
    print("2. Update Stock")
    print("3. Calculate Total Inventory Value")
    print("4. Apply Discount")
    print("5. Check Low Stock Items")
    print("6. Exit")
    choice = input("Enter your choice: ")

    if choice == "1":
        print("\nSelect Product Type:")
        print("1. Electronics")
        print("2. Clothing")
        print("3. Groceries")
        product_type = input("Enter choice (1/2/3): ")

        name = input("Enter product name: ")
        price = float(input("Enter product price: "))
        quantity = int(input("Enter product quantity: "))
        sku = input("Enter SKU: ")

        if product_type == "1":
            warranty_period = input("Enter warranty period: ")
            brand = input("Enter brand: ")
            product = Electronics(name, price, quantity, sku, warranty_period, brand)
            inventory.add_product(product)
        elif product_type == "2":
```

```
            size = input("Enter size: ")
            material = input("Enter material: ")
            product = Clothing(name, price, quantity, sku, size, material)
            inventory.add_product(product)
        elif product_type == "3":
            expiration_date = input("Enter expiration date (YYYY-MM-DD): ")
            organic_status = input("Is it organic? (yes/no): ").lower() == "yes"
            product = Groceries(name, price, quantity, sku, expiration_date, organic_status)
            inventory.add_product(product)
        else:
            print("Invalid product type.")

    elif choice == "2":
        sku = input("Enter SKU of the product to update stock: ")
        quantity = int(input("Enter quantity to add: "))
        inventory.update_stock(sku, quantity)

    elif choice == "3":
        total_value = inventory.calculate_total_value()
        print(f"Total inventory value: ${total_value:.2f}")

    elif choice == "4":
        sku = input("Enter SKU of the product to apply discount: ")
        discount_percentage = float(input("Enter discount percentage: "))
        inventory.apply_discount(sku, discount_percentage)

    elif choice == "5":
```

```
            low_stock_items = inventory.check_low_stock()
            if low_stock_items:
                print("Low stock items:")
                for item in low_stock_items:
                    print(f"{item.name} (Quantity: {item.quantity})")
            else:
                print("No low stock items.")

        elif choice == "6":
            break

        else:
            print("Invalid choice, please try again.")
```

OUTPUT:

```
1. Add Product
2. Update Stock
3. Calculate Total Inventory Value
4. Apply Discount
5. Check Low Stock Items
6. Exit
Enter your choice:  1

Select Product Type:
1. Electronics
2. Clothing
3. Groceries
Enter choice (1/2/3):  1
Enter product name:  fRUITS
Enter product price:  500
Enter product quantity:  5
Enter SKU:  9
Enter warranty period:  1
Enter brand:  NIOI
Added fRUITS to inventory.

1. Add Product
2. Update Stock
3. Calculate Total Inventory Value
4. Apply Discount
```

5. Check Low Stock Items
6. Exit

[ ]:

Exercise 2

Building a Payment Processing System

You are tasked with designing a Payment Processing System that handles multiple payment methods (Credit Card, PayPal, Bank Transfer). Each payment method has unique steps involved in processing payments, but they all share the common interface of processing a payment and issuing a refund.

Problem Definition:

The system must support the following payment methods:

1. Credit Card Payment: Requires card number, expiry date, and CVV to process payments.

2. PayPal Payment: Uses a PayPal account email and password.

3. Bank Transfer Payment: Processes payments using a bank account number and a sort code.

Each payment method has:

- **A method to process payments.**
- **A method to issue refunds.**
- **Error handling for failed payments.**

```
class PaymentMethod:
    def process_payment(self, amount):
        raise NotImplementedError("This method should be overridden in subclasses")

    def issue_refund(self, amount):
```

```
        raise NotImplementedError("This method should be overridden in subclasses")


class CreditCardPayment(PaymentMethod):
    def __init__(self, card_number, expiry_date, cvv):
        self.card_number = card_number
        self.expiry_date = expiry_date
        self.cvv = cvv

    def process_payment(self, amount):
        if self.validate_card():
            print(f"Processing credit card payment of ${amount}")
        else:
            print("Credit card validation failed.")

    def issue_refund(self, amount):
        print(f"Issuing credit card refund of ${amount}")

    def validate_card(self):
        return True

class PayPalPayment(PaymentMethod):
    def __init__(self, email, password):
        self.email = email
        self.password = password
```

```
    def process_payment(self, amount):
        if self.authenticate():
            print(f"Processing PayPal payment of ${amount}")
        else:
            print("PayPal authentication failed.")

    def issue_refund(self, amount):
        print(f"Issuing PayPal refund of ${amount}")

    def authenticate(self):
        return True

class BankTransferPayment(PaymentMethod):
    def __init__(self, account_number, sort_code):
        self.account_number = account_number
        self.sort_code = sort_code

    def process_payment(self, amount):
        print(f"Processing bank transfer of ${amount}")

    def issue_refund(self, amount):
        print(f"Issuing bank transfer refund of ${amount}")

# User-defined input usage
def main():
```

```
print("Select Payment Method:")
print("1. Credit Card")
print("2. PayPal")
print("3. Bank Transfer")
choice = input("Enter choice (1/2/3): ")

if choice == "1":
    card_number = input("Enter Credit Card Number: ")
    expiry_date = input("Enter Expiry Date (MM/YY): ")
    cvv = input("Enter CVV: ")
    amount = float(input("Enter Payment Amount: "))
    credit_card_payment = CreditCardPayment(card_number, expiry_date, cvv)
    credit_card_payment.process_payment(amount)
    refund_choice = input("Would you like to issue a refund? (y/n): ")
    if refund_choice.lower() == 'y':
        refund_amount = float(input("Enter Refund Amount: "))
        credit_card_payment.issue_refund(refund_amount)

elif choice == "2":
    email = input("Enter PayPal Email: ")
    password = input("Enter PayPal Password: ")
    amount = float(input("Enter Payment Amount: "))
    paypal_payment = PayPalPayment(email, password)
    paypal_payment.process_payment(amount)
    refund_choice = input("Would you like to issue a refund? (y/n): ")
```

```
        if refund_choice.lower() == 'y':
            refund_amount = float(input("Enter Refund Amount: "))
            paypal_payment.issue_refund(refund_amount)

    elif choice == "3":
        account_number = input("Enter Bank Account Number: ")
        sort_code = input("Enter Sort Code: ")
        amount = float(input("Enter Payment Amount: "))
        bank_transfer_payment = BankTransferPayment(account_number, sort_code)
        bank_transfer_payment.process_payment(amount)
        refund_choice = input("Would you like to issue a refund? (y/n): ")
        if refund_choice.lower() == 'y':
            refund_amount = float(input("Enter Refund Amount: "))
            bank_transfer_payment.issue_refund(refund_amount)

    else:
        print("Invalid choice, please try again.")

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

OUTPUT:

```
Select Payment Method:
1. Credit Card
2. PayPal
3. Bank Transfer
Enter choice (1/2/3):  1
Enter Credit Card Number:  122
Enter Expiry Date (MM/YY):  10/27
Enter CVV:  805
Enter Payment Amount:  5000
```

Processing credit card payment of $5000.0
Would you like to issue a refund? (y/n): n

PIYUSH KUMAR MISHRA

230957212

ROLL N0:70

WEEK 8:

Exercise 1:

File Handling in Python with Error and Exception Handling

You are tasked with building a simple data processing system for an e-commerce company that handles product inventories and customer reviews. The company uses two types of files to store data:

1. CSV Files: The product inventory is stored in a CSV file with the following columns: Product IL), Product Name, Category, Price, and Stock Quantity.

2, JSON Files: Customer reviews for each product are stored in separate JSON files. Each review file contains a list of reviews for a product, where each review has the following fields: Review IL), User ID, Rating, Comment, and Date.

Your system needs to process these files to provide insights and updates on product availability, pricing trends, and customer feedback. Additionally, the system should allow adding new products and updating product information in the CSV file, while handling errors and exceptions gracefully.

System Requirements:

1. Product Management (CSV File):

• Add New Products: Write a function to add a new product to the inventory CSV file.

• Update Stock and Price: Implement a function to update the stock quantity and price of an existing product in the CSV file.

Check Product Availability: Write a function that reads the CSV file to check if a product is in stock and, if so, how many units are available.

2. Customer Review Management (JSON Files):

Add Customer Review: Implement a function that adds a new customer review to the corresponding JSON file for a product.

Average Rating Calculation: Write a function that reads a product's JSON review file and calculates its average rating.

Review Search: Implement a search function that reads the JSON file and allows users to search for reviews containing specific keywords in the comment section.

3. Data Analysis:

• Top Rated Products: Create a function that reads all product review JSON files, calculates the average rating for each product, and lists the top 5 highest-rated products.

• Out of Stock Products: Write a function that reads the CSV inventory file and lists all products that are out of stock.

• Price Trends: Implement a function to read the CSV file and identify products whose prices have increased or decreased in the past month.

4. Error and Exception Handling:

• Implement error handling for situations such as:

File Not Found: When the CSV or JSON file is missing.

Invalid Data Format: If the data in the files does not match the expected structure.

File Corruption: When a file contains corrupted data.

Read/Write Permissions: When the system does not have permission to read or write to a file.

```
import csv
import json
import os
```

```
def add_product(file_path):
    product_id = input("Enter Product ID: ")
    product_name = input("Enter Product Name: ")
    category = input("Enter Category: ")
    price = float(input("Enter Price: "))
    stock_quantity = int(input("Enter Stock Quantity: "))

    try:
        with open(file_path, mode='a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow([product_id, product_name, category, price, stock_quantity])
            print("Product added successfully.")
    except PermissionError:
        print("Error: Permission denied when trying to write to the file.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

def update_product(file_path):
    product_id = input("Enter Product ID to update: ")
    new_price = float(input("Enter New Price: "))
    new_stock_quantity = int(input("Enter New Stock Quantity: "))

    try:
        updated_rows = []
        with open(file_path, mode='r') as file:
            reader = csv.reader(file)
```

```
            for row in reader:
                if row[0] == product_id:
                    row[3] = str(new_price)
                    row[4] = str(new_stock_quantity)
                updated_rows.append(row)

        with open(file_path, mode='w', newline='') as file:
            writer = csv.writer(file)
            writer.writerows(updated_rows)
            print("Product updated successfully.")
    except FileNotFoundError:
        print("Error: The CSV file was not found.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

def check_availability(file_path):
    product_id = input("Enter Product ID to check availability: ")

    try:
        with open(file_path, mode='r') as file:
            reader = csv.reader(file)
            for row in reader:
                if row[0] == product_id:
                    return f"Available stock: {row[4]}"
        return "Product not found."
    except FileNotFoundError:
        print("Error: The CSV file was not found.")
```

```
def add_review():
    product_id = input("Enter Product ID for the review: ")
    review_id = input("Enter Review ID: ")
    user_id = input("Enter User ID: ")
    rating = int(input("Enter Rating (1-5): "))
    comment = input("Enter Comment: ")
    date = input("Enter Date (YYYY-MM-DD): ")

    review = {
        "Review ID": review_id,
        "User ID": user_id,
        "Rating": rating,
        "Comment": comment,
        "Date": date
    }

    file_name = f"{product_id}_reviews.json"

    try:
        if not os.path.exists(file_name):
            with open(file_name, 'w') as file:
                json.dump([], file)

        with open(file_name, 'r+') as file:
            reviews = json.load(file)
            reviews.append(review)
```

```
            file.seek(0)
            json.dump(reviews, file, indent=4)
            print("Review added successfully.")
    except Exception as e:
        print(f"An error occurred while adding the review: {e}")

def average_rating(product_id):
    file_name = f"{product_id}_reviews.json"

    try:
        with open(file_name) as file:
            reviews = json.load(file)
            total_rating = sum(review['Rating'] for review in reviews)
            average = total_rating / len(reviews) if reviews else 0
            return f"Average Rating: {average:.2f}"
    except FileNotFoundError:
        return "Review file not found."
    except json.JSONDecodeError:
        return "Error: Invalid JSON format."

def search_reviews():
    product_id = input("Enter Product ID to search reviews: ")
    keyword = input("Enter keyword to search in comments: ")

    file_name = f"{product_id}_reviews.json"

    try:
```

```
        with open(file_name) as file:
            reviews = json.load(file)
            matching_reviews = [review for review in reviews if keyword.lower() in
review['Comment'].lower()]
            return matching_reviews if matching_reviews else "No matching reviews found."
    except FileNotFoundError:
        return "Review file not found."

def top_rated_products(product_ids):
    ratings = {}

    for product_id in product_ids:
        avg_rating_result = average_rating(product_id)

        if isinstance(avg_rating_result, str) and "Average Rating" in avg_rating_result:
            ratings[product_id] = float(avg_rating_result.split(": ")[1])

    top_products = sorted(ratings.items(), key=lambda x: x[1], reverse=True)[:5]

    return top_products

def out_of_stock_products(file_path):
    out_of_stock = []

    try:
        with open(file_path) as file:
            reader = csv.reader(file)
            for row in reader:
```

```
            if int(row[4]) == 0:
                out_of_stock.append(row[1])

        return out_of_stock if out_of_stock else ["No products are out of stock."]

    except FileNotFoundError:
        return "Inventory file not found."

def price_trends(current_file_path, previous_file_path):
    current_prices = {}
    previous_prices = {}

    try:
        with open(current_file_path) as current_file:
            reader = csv.reader(current_file)
            for row in reader:
                current_prices[row[0]] = float(row[3])

        with open(previous_file_path) as previous_file:
            reader = csv.reader(previous_file)
            for row in reader:
                previous_prices[row[0]] = float(row[3])

        trends = {}
        for product_id in current_prices.keys():
            if product_id in previous_prices:
                change = current_prices[product_id] - previous_prices[product_id]
```

```
                trends[product_id] = change

        return trends
    except FileNotFoundError as e:
        return f"File not found: {e.filename}"

def main():
    inventory_file_path = 'inventory.csv'

    while True:
        print("\nE-commerce Data Processing System")
        print("1. Add Product")
        print("2. Update Product")
        print("3. Check Product Availability")
        print("4. Add Customer Review")
        print("5. Calculate Average Rating")
        print("6. Search Reviews")
        print("7. List Out of Stock Products")
        print("8. Show Top Rated Products")

        choice = input("\nSelect an option (or 'q' to quit): ")

        if choice == '1':
            add_product(inventory_file_path)
        elif choice == '2':
            update_product(inventory_file_path)
        elif choice == '3':
```

```
    print(check_availability(inventory_file_path))
elif choice == '4':
    add_review()
elif choice == '5':
    product_id = input("Enter Product ID to calculate average rating: ")
    print(average_rating(product_id))
elif choice == '6':
    results = search_reviews()
    if isinstance(results, list):
        for review in results:
            print(review)
    else:
        print(results)
elif choice == '7':
    out_of_stock_list = out_of_stock_products(inventory_file_path)
    for item in out_of_stock_list:
        print(item)
elif choice == '8':
    example_product_ids = ['P001', 'P002', 'P003', 'P004', 'P005']
    top_products_list = top_rated_products(example_product_ids)
    for product in top_products_list:
        print(product)
elif choice.lower() == 'q':
    break
else:
    print("Invalid option. Please try again.")
```

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

OUPUT:

```
E-commerce Data Processing System
1. Add Product
2. Update Product
3. Check Product Availability
4. Add Customer Review
5. Calculate Average Rating
6. Search Reviews
7. List Out of Stock Products
8. Show Top Rated Products

Select an option (or 'q' to quit):  1
Enter Product ID:  123
Enter Product Name:  XYZ
Enter Category:  GROCERY
Enter Price:  1000
Enter Stock Quantity:  9
Product added successfully.

E-commerce Data Processing System
1. Add Product
2. Update Product
3. Check Product Availability
4. Add Customer Review
5. Calculate Average Rating
6. Search Reviews
7. List Out of Stock Products
8. Show Top Rated Products

Select an option (or 'q' to quit):  q
```

PIYUSH KUMAR MISHRA

230957212

ROLL NO 70

WEEK 9

In [2]:
```python
import pandas as pd
import numpy as np

# Create a sample dataset
data = {
    'Date': pd.date_range(start='2024-01-01', periods=20, freq='W'),
    'Product Name': np.random.choice(['Widget A', 'Widget B', 'Widget C'], 20),
    'Units Sold': np.random.randint(1, 20, size=20),
    'Revenue': np.random.uniform(100, 1000, size=20).round(2),
    'Region': np.random.choice(['North', 'South', 'East', 'West'], 20),
    'Discount Offered (%)': np.random.uniform(0, 30, size=20).round(2),
    'Salesperson': np.random.choice(['Alice', 'Bob', 'Charlie', 'David'], 20)
}

df = pd.DataFrame(data)

# Calculate Revenue After Discounts
df['Revenue After Discount'] = df['Revenue'] * (1 - df['Discount Offered (%)'] / 100)
print(df)
```

```
         Date Product Name  Units Sold  Revenue Region  Discount Offered (%)  \
0  2024-01-07     Widget B          14   809.91  North                  7.00
1  2024-01-14     Widget A          19   200.50   West                 22.22
2  2024-01-21     Widget B          17   684.44  South                 21.18
3  2024-01-28     Widget A           3   325.69  South                 26.80
4  2024-02-04     Widget B           4   729.73  South                 28.41
5  2024-02-11     Widget B          14   603.22   West                  6.90
6  2024-02-18     Widget B           6   742.65  South                 11.88
7  2024-02-25     Widget B          13   610.45   West                  8.87
8  2024-03-03     Widget C           8   346.44  North                  8.32
9  2024-03-10     Widget C           6   974.03  South                 25.85
10 2024-03-17     Widget C           2   925.73   East                  4.99
11 2024-03-24     Widget B          14   847.99   East                 24.52
12 2024-03-31     Widget B          17   937.64   West                  7.30
13 2024-04-07     Widget B          19   514.51  North                 17.28
14 2024-04-14     Widget C          15   355.25  North                 10.64
15 2024-04-21     Widget B          19   349.53   West                 11.51
16 2024-04-28     Widget B          14   804.63  South                  2.94
17 2024-05-05     Widget B          17   202.10  South                  5.49
18 2024-05-12     Widget B           8   478.21   West                  0.35
19 2024-05-19     Widget C           6   237.60  South                 23.35

   Salesperson  Revenue After Discount
0        David              753.216300
1        Alice              155.948900
2        David              539.475608
3          Bob              238.405080
4          Bob              522.413707
5        Alice              561.597820
6      Charlie              654.423180
7      Charlie              556.303085
8        David              317.616192
9          Bob              722.243245
10         Bob              879.536073
11       Alice              640.062852
12         Bob              869.192280
13     Charlie              425.602672
14     Charlie              317.451400
15       David              309.299097
16     Charlie              780.973878
17       Alice              191.004710
18       Alice              476.536265
19         Bob              182.120400
```

In [3]:
```python
# 1. Top 3 sales transactions with the highest revenue
top_3_revenue = df.nlargest(3, 'Revenue')
print("Top 3 Sales Transactions with Highest Revenue:\n", top_3_revenue)
```

```
Top 3 Sales Transactions with Highest Revenue:
          Date Product Name  Units Sold  Revenue Region  Discount Offered (%)  \
9   2024-03-10     Widget C           6   974.03  South                 25.85
12  2024-03-31     Widget B          17   937.64   West                  7.30
10  2024-03-17     Widget C           2   925.73   East                  4.99

   Salesperson  Revenue After Discount
9          Bob              722.243245
12         Bob              869.192280
10         Bob              879.536073
```

In [4]:
```python
# 2. Units sold for each product
units_sold_per_product = df.groupby('Product Name')['Units Sold'].sum()
print("\nUnits Sold for Each Product:\n", units_sold_per_product)
```

```
Units Sold for Each Product:
 Product Name
Widget A     22
Widget B    176
Widget C     37
Name: Units Sold, dtype: int32
```

In [5]:
```python
# 3. Total revenue after applying discounts
total_revenue_after_discount = df['Revenue After Discount'].sum()
print("\nTotal Revenue After Discounts:\n", total_revenue_after_discount)
```

```
Total Revenue After Discounts:
 10093.422744
```

In [6]:
```python
# 4. Transaction with the highest discount offered
highest_discount = df.loc[df['Discount Offered (%)'].idxmax()]
highest_discount_revenue_after_discount = highest_discount['Revenue After Discount']
print("\nHighest Discount Transaction:\n", highest_discount)
```

```
Highest Discount Transaction:
 Date                      2024-02-04 00:00:00
Product Name                         Widget B
Units Sold                                  4
Revenue                                729.73
Region                                  South
Discount Offered (%)                    28.41
Salesperson                               Bob
Revenue After Discount             522.413707
Name: 4, dtype: object
```

In [7]:
```python
# 5. Salesperson generating the highest total revenue
highest_revenue_salesperson = df.groupby('Salesperson')['Revenue'].sum().idxmax()
print("\nSalesperson with Highest Total Revenue:\n", highest_revenue_salesperson)
```

```
Salesperson with Highest Total Revenue:
 Bob
```

In [8]:
```python
# 6. Average discount offered by each salesperson
avg_discount_per_salesperson = df.groupby('Salesperson')['Discount Offered (%)'].mean()
```

```
print("\nAverage Discount Offered by Each Salesperson:\n", avg_discount_per_salesperson)
```

```
Average Discount Offered by Each Salesperson:
 Salesperson
Alice      11.8960
Bob        19.4500
Charlie    10.3220
David      12.0025
Name: Discount Offered (%), dtype: float64
```

In [9]:
```
# 7. Revenue generated in each region
revenue_per_region = df.groupby('Region')['Revenue'].sum()
print("\nRevenue Generated in Each Region:\n", revenue_per_region)
```

```
Revenue Generated in Each Region:
 Region
East     1773.72
North    2026.11
South    4700.87
West     3179.55
Name: Revenue, dtype: float64
```

In [10]:
```
# 8. Region where Alice generated the highest sales
alice_sales = df[df['Salesperson'] == 'Alice']
alice_highest_sales_region = alice_sales.groupby('Region')['Revenue'].sum().idxmax()
print("\nAlice's Highest Sales Region:\n", alice_highest_sales_region)
```

```
Alice's Highest Sales Region:
 West
```

In [11]:
```
# 9. Product generating the highest revenue per unit sold
df['Revenue Per Unit'] = df['Revenue'] / df['Units Sold']
highest_revenue_per_unit_product = df.loc[df['Revenue Per Unit'].idxmax()]
print("\nHighest Revenue per Unit Sold Product:\n", highest_revenue_per_unit_product)
```

```
Highest Revenue per Unit Sold Product:
 Date                       2024-03-17 00:00:00
Product Name                           Widget C
Units Sold                                    2
Revenue                                  925.73
Region                                     East
Discount Offered (%)                       4.99
Salesperson                                 Bob
Revenue After Discount              879.536073
Revenue Per Unit                       462.865
Name: 10, dtype: object
```

In [13]:
```
# 10. Transactions rated as "High" performance (arbitrarily define as Revenue > 800)
high_performance_transactions = df[df['Revenue'] > 800]
print("\nHigh Performance Transactions Count:\n", len(high_performance_transactions))
```

```
High Performance Transactions Count:
 6
```

In [16]:
```python
# 11. Salesperson sold the most units in North region without offering any discount
north_sales_no_discount = df[(df['Region'] == 'North') & (df['Discount Offered (%)'] == 0)]

if not north_sales_no_discount.empty:
    most_units_north = north_sales_no_discount.groupby('Salesperson')['Units Sold'].sum().idxmax()
    print("\nMost Units Sold in North Region Without Discount:\n", most_units_north)
else:
    print("\nNo sales in North region without discount.")
```

```
No sales in North region without discount.
```

In [17]:
```python
# 12. Average revenue per unit sold in each region for each product
avg_revenue_per_unit_region_product = df.groupby(['Region', 'Product Name']).apply(lambda x: (x['Revenue'] / x['Units Sold']).mean())
print("\nAverage Revenue per Unit Sold in Each Region for Each Product:\n", avg_revenue_per_unit_region_product)
```

```
Average Revenue per Unit Sold in Each Region for Each Product:
 Region  Product Name
East    Widget B         60.570714
        Widget C        462.865000
North   Widget B         42.465094
        Widget C         33.494167
South   Widget A        108.563333
        Widget B         83.166097
        Widget C        100.969167
West    Widget A         10.552632
        Widget B         44.674539
dtype: float64
```

In [18]:
```python
# 13. Salesperson with the highest average revenue after discounts
avg_revenue_after_discount_per_salesperson = df.groupby('Salesperson')['Revenue After Discount'].mean().idxmax()
print("\nSalesperson with Highest Average Revenue After Discounts:\n", avg_revenue_after_discount_per_salesperson)
```

```
Salesperson with Highest Average Revenue After Discounts:
 Bob
```

In [19]:
```python
# 14. Cumulative total revenue over time for each salesperson
cumulative_revenue = df.groupby(['Salesperson', 'Date'])['Revenue'].sum().groupby(level=0).cumsum()
print("\nCumulative Total Revenue Over Time for Each Salesperson:\n", cumulative_revenue)
```

```
Cumulative Total Revenue Over Time for Each Salesperson:
 Salesperson  Date
Alice        2024-01-14     200.50
             2024-02-11     803.72
             2024-03-24    1651.71
             2024-05-05    1853.81
             2024-05-12    2332.02
Bob          2024-01-28     325.69
             2024-02-04    1055.42
             2024-03-10    2029.45
             2024-03-17    2955.18
             2024-03-31    3892.82
             2024-05-19    4130.42
Charlie      2024-02-18     742.65
             2024-02-25    1353.10
             2024-04-07    1867.61
             2024-04-14    2222.86
             2024-04-28    3027.49
David        2024-01-07     809.91
             2024-01-21    1494.35
             2024-03-03    1840.79
             2024-04-21    2190.32
Name: Revenue, dtype: float64
```

In [20]:

```python
# 15. Rank transactions by revenue for each salesperson and find top 2
top_2_transactions_per_salesperson = df.groupby('Salesperson').apply(lambda x: x.nlargest(2, 'Revenue'))
print("\nTop 2 Transactions per Salesperson:\n", top_2_transactions_per_salesperson)
```

```
Top 2 Transactions per Salesperson:
                     Date Product Name  Units Sold  Revenue Region  \
Salesperson
Alice       11 2024-03-24     Widget B          14   847.99   East
            5  2024-02-11     Widget B          14   603.22   West
Bob         9  2024-03-10     Widget C           6   974.03  South
            12 2024-03-31     Widget B          17   937.64   West
Charlie     16 2024-04-28     Widget B          14   804.63  South
            6  2024-02-18     Widget B           6   742.65  South
David       0  2024-01-07     Widget B          14   809.91  North
            2  2024-01-21     Widget B          17   684.44  South

                Discount Offered (%) Salesperson  Revenue After Discount  \
Salesperson
Alice       11                 24.52       Alice              640.062852
            5                   6.90       Alice              561.597820
Bob         9                  25.85         Bob              722.243245
            12                  7.30         Bob              869.192280
Charlie     16                  2.94     Charlie              780.973878
            6                  11.88     Charlie              654.423180
David       0                   7.00       David              753.216300
            2                  21.18       David              539.475608

                Revenue Per Unit
Salesperson
Alice       11         60.570714
            5          43.087143
Bob         9         162.338333
            12         55.155294
Charlie     16         57.473571
            6         123.775000
David       0          57.850714
            2          40.261176
```

In [21]:

```python
# 16. Cumulative revenue for each product per day
cumulative_revenue_per_product = df.groupby(['Date', 'Product Name'])['Revenue'].sum().groupby(level=1).cumsum()
print("\nCumulative Revenue per Product per Day:\n", cumulative_revenue_per_product)
```

```
Cumulative Revenue per Product per Day:
 Date        Product Name
2024-01-07  Widget B         809.91
2024-01-14  Widget A         200.50
2024-01-21  Widget B        1494.35
2024-01-28  Widget A         526.19
2024-02-04  Widget B        2224.08
2024-02-11  Widget B        2827.30
2024-02-18  Widget B        3569.95
2024-02-25  Widget B        4180.40
2024-03-03  Widget C         346.44
2024-03-10  Widget C        1320.47
2024-03-17  Widget C        2246.20
2024-03-24  Widget B        5028.39
2024-03-31  Widget B        5966.03
2024-04-07  Widget B        6480.54
2024-04-14  Widget C        2601.45
2024-04-21  Widget B        6830.07
2024-04-28  Widget B        7634.70
2024-05-05  Widget B        7836.80
2024-05-12  Widget B        8315.01
2024-05-19  Widget C        2839.05
Name: Revenue, dtype: float64
```

In [22]:
```
# 17. Average revenue generated with discount vs without
avg_revenue_discount_vs_no_discount = df.groupby('Product Name').agg({
    'Revenue': ['mean'],
    'Discount Offered (%)': ['count']
}).rename(columns={('Revenue', 'mean'): 'Avg Revenue', ('Discount Offered (%)', 'count'): 'Transaction Count'})
print("\nAverage Revenue with vs without Discounts:\n", avg_revenue_discount_vs_no_discount)
```

```
Average Revenue with vs without Discounts:
                 Revenue Discount Offered (%)
                   mean                count
Product Name
Widget A      263.095000                    2
Widget B      639.616154                   13
Widget C      567.810000                    5
```

In [23]:
```
# 18. Weighted average discount offered by each salesperson
weighted_avg_discount = df.groupby('Salesperson').apply(
    lambda x: np.average(x['Discount Offered (%)'], weights=x['Revenue'])
)
print("\nWeighted Average Discount Offered by Each Salesperson:\n", weighted_avg_discount)
```

```
Weighted Average Discount Offered by Each Salesperson:
 Salesperson
Alice      13.158955
Bob        17.347127
Charlie     9.669257
David      12.359509
dtype: float64
```

In [24]:
```
# 19. Percentage of total revenue each region contributes
total_revenue = df['Revenue'].sum()
percentage_revenue_per_region = (revenue_per_region / total_revenue) * 100
print("\nPercentage of Total Revenue per Region:\n", percentage_revenue_per_region)
```

```
Percentage of Total Revenue per Region:
 Region
East     15.185634
North    17.346461
South    40.246313
West     27.221592
Name: Revenue, dtype: float64
```

In [ ]: