VIGNAN'S
Foundation for Science, Technology & Research
(Deemed to be University)
-Estd. u/s 3 of UGC Act 1956

DEPARTMENT OF INFORMATION TECHNOLOGY & COMPUTER APPLICATIONS

22IT206-PYTHON PROGRAMMING

II IT-A&B Even Semester (2023-24)

Module-2 Question Bank

1. **Scenario:** You are tasked with developing a Python program that reads data from a CSV file containing information about students' grades. The program should process the data, calculate the average grade for each student, and create a new file with the student names and their corresponding average grades.

   Understanding (CSV File Processing):
   - Explain the role of a **for** loop in Python when iterating through rows of a CSV file. Discuss how this loop facilitates the processing of each row's data.

   Remembering (CSV File Reading and Looping):
   - Create a CSV file named "grades.csv" with columns for student names and their grades. Write a Python program that reads the data from the file and uses a **for** loop to iterate through each row, printing the student name and grade.

   Applying (Handling CSV File Errors):
   - Enhance the previous program to handle potential errors that might occur when reading the CSV file. Implement a try-except-else block to catch and handle file-related exceptions, providing a meaningful error message.

   TARGET-2:

   Analyzing/Evaluating (Calculating Average Grades):
   - Modify the program to calculate the average grade for each student based on the grades in the CSV file. Discuss the logic involved in computing the average and how it is integrated into the existing program.

   Group Programming Challenge (Custom Exception and Output File):
   - Introduce a custom exception class, e.g., **FileProcessingError**, to handle unexpected errors related to file processing. Implement the program to catch and handle this custom exception. Additionally, create a new CSV file named "average_grades.csv" and write the student names along with their calculated average grades to this file. Discuss the benefits of using a custom exception for error handling.

Test cases:

Test Case 1: Basic CSV Reading and Printing
Test Case 2: CSV File Error Handling
Test Case 3: Average Grade Calculation
Test Case 4: Custom Exception Handling
Test Case 5: Writing to Output File

2. **Scenario: Word Frequency Analysis for Content Optimization**

You are part of a content creation team responsible for managing a vast database of articles on your company's website. The marketing team has identified the need to optimize the content by focusing on the most relevant and frequently used words across various articles. This optimization aims to improve SEO rankings and enhance the overall user experience.

Understanding Word Frequency Analysis:

- Explain the significance of word frequency analysis in optimizing textual content for SEO and user experience.

Remembering: Basic File Handling and Tokenization:

- Write a Python code snippet demonstrating how to open an article file, read its contents, and close the file afterward.
- Provide an example of tokenizing the text from a sample article into words using basic file handling operations and save the tokenized words into a new text file for future analysis.

Applying: Word Frequency Analysis Function:

- Develop a Python function that takes an article file as input and counts the frequency of each word.
- Apply this function to a sample article and print the resulting word frequency dictionary and save the word frequency dictionary into a new text file for further analysis.

TARGET-2:

Analyzing/Evaluating: Sorting Word Frequencies and Analysis:

- Write a Python function to sort the word frequency dictionary based on the counts in descending order.
- Evaluate the impact of sorting on identifying the most frequent words for content optimization and save the sorted word frequency data into a new text file for detailed analysis.

Creating (Group Programming Challenge: Advanced Collaborative Analysis and Optimization:

- Propose and implement enhancements to the program that allow reading and analyzing multiple article files in a directory.
- Discuss the potential challenges and considerations in handling a large dataset of articles.
- Explore collaborative strategies for merging word frequency data from multiple files and generating a consolidated report.

Test cases:

1. Ensure file can be opened, read, and closed without errors.
2. Validate accurate tokenization of text into words, excluding punctuation.
3. Verify correctness of word frequency counting function.
4. Confirm correct sorting of word frequency dictionary by counts.
5. Test program's ability to handle large dataset of article files.

3. Scenario: File Handling with Exception Handling

You are developing a file processing application in Python that reads data from text files, performs operations such as data manipulation or analysis, and writes the results to output files. To ensure robustness and handle potential errors, you need to incorporate exception handling mechanisms throughout the application.

Understanding (File Handling and Exception Handling Overview):

- Provide an overview of file handling in Python, emphasizing the importance of handling file-related errors using exception handling techniques. Discuss common file operations such as reading, writing, and error types encountered during file processing.

Remembering (Basic File Operations with Try-Except Blocks):

- Write Python code snippets demonstrating basic file operations such as opening, reading, and writing to files within try-except blocks. Discuss how try-except blocks can be used to handle exceptions such as FileNotFoundError and PermissionError.

Applying (Custom Exception Classes for File Processing):

- Define custom exception classes, such as FileProcessingError and FileFormatError, to handle specific types of errors encountered during file processing. Implement these custom exceptions in the file processing application to provide meaningful error messages and enhance code readability.

TARGET-2:

Analyzing (Error Handling Strategies for File Operations):

- Analyze error handling strategies for different file operations, including reading, writing, and closing files. Discuss best practices for error recovery, graceful error handling, and logging error information to facilitate debugging and troubleshooting.

Group Programming Challenge (Enhanced Error Handling Mechanisms):

- Collaborate with your team to enhance the file processing application with advanced error handling mechanisms, such as context managers (using the **with** statement) and error propagation strategies. Implement comprehensive error handling for complex file processing scenarios and discuss strategies for handling nested exceptions.

Test Cases:

1. Ensure files can be opened, read, and closed without errors using try-except blocks.
2. Validate handling of FileNotFoundError and PermissionError exceptions during file operations.
3. Handling exceptions like FileProcessingError and FileFormatError to handle specific file-related errors.
4. Analyze error handling strategies for reading, writing, and closing files to ensure robust error recovery.

4. **Scenario: Letter Frequency Analysis from Multilingual Texts**

You are tasked with developing a Python program that reads text files containing passages written in different languages and analyses the frequency of letters from A to Z. The program should convert all input to lowercase, exclude spaces, digits, punctuation, and any characters other than letters A to Z. The goal is to compare the frequency of letters across multiple languages and observe variations in letter frequency.

Understanding (Letter Frequency Analysis Overview):

- Provide an overview of letter frequency analysis and its significance in fields such as cryptography, linguistics, and data analysis. Discuss the importance of converting text to lowercase and excluding non-alphabetic characters for accurate analysis.

Remembering (Reading and Processing Text Files):

- Write Python code snippets demonstrating how to read text files and preprocess the content by converting it to lowercase and removing non-alphabetic characters. Implement functions to tokenize the text and count the frequency of each letter from A to Z.

Applying (Multilingual Text Analysis):

- Incorporate text samples from several different languages, including English, Spanish, French, and others, into the program. Analyze the frequency of letters across these languages and visualize the results using histograms or bar charts to compare the distribution of letters.

TARGET-2:

Analyzing (Comparison of Letter Frequency Variations):

- Analyze the results of the letter frequency analysis across different languages. Discuss observed variations in letter frequency distributions and potential factors influencing these variations, such as language structure, word usage, and cultural influences.

Group Programming Challenge (Enhanced Multilingual Analysis):

- Collaborate with your team to enhance the letter frequency analysis program by incorporating advanced text processing techniques, such as stemming, lemmatization, or language detection. Implement additional features to extract insights from the text data and generate comparative reports on letter frequency variations among different languages.

Test Cases:

1. English text analysis.
2. Multilingual comparison of letter frequencies.
3. Empty input file handling.
4. Non-alphabetic character exclusion test.

5. **Scenario: Online Shopping System Using OOP**

In this scenario, let's explore the concept of Object-Oriented Programming (OOP) in Python through the development of an online shopping system. The system will have different classes representing key entities, such as products, customers, and the shopping cart.

Understanding (Class Definition):

- **Product Class:**
  - **Attributes: product_id**, **name**, **price**, **stock_quantity**
  - **Methods: __init__** (constructor), **get_details** (to retrieve product information), **update_stock** (to manage stock quantity)
- **Customer Class:**
  - **Attributes: customer_id**, **name**, **email**, **address**
  - **Methods: __init__** (constructor), **get_details** (to retrieve customer information)

Remembering (Creating Instances):

- Create instances of the **Product** class to represent different items available for purchase.
- Create instances of the **Customer** class to represent individual customers.

Applying (Shopping Cart Class):

- **ShoppingCart Class:**
  - **Attributes: customer** (instance of Customer), **cart_items** (list to store selected products)
  - **Methods: add_to_cart** (to add products to the cart), **remove_from_cart** (to remove products), **view_cart** (to display items in the cart)

Analyzing (Inheritance):

- Introduce the concept of inheritance by creating a **PremiumCustomer** class that inherits from the **Customer** class. It can have additional attributes/methods, such as **membership_status**.

TARGET-2:

Analyzing (Polymorphism):

- Utilize polymorphism by ensuring that both **Customer** and **PremiumCustomer** classes have a **get_details** method, but with variations.

Evaluating/Creating (Group Challenge - Order Processing):

- Collaboratively design and implement a class named **Order** that represents a customer's order.
- The **Order** class should have attributes such as **order_id**, **order_date**, **ordered_items** (list of products), and methods like **calculate_total** (to calculate the order total).
- Discuss the advantages of encapsulation, inheritance, and polymorphism in the context of the online shopping system.

Test Cases:

1. Verify that the product details are correctly initialized.
2. Ensure that the shopping cart can accurately add and remove items.
3. Confirm that the premium customer class inherits from the customer class and has additional attributes.
4. Test the order processing functionality and total calculation for different scenarios.

6. **Scenario: Histogram Generator**

You are tasked with developing a Python program that analyzes the word lengths in a given text file and generates a histogram to visualize the distribution of word lengths. This tool is valuable for understanding the characteristics of the text content and identifying patterns in word lengths.

Understanding Word Length Analysis:

- Explain the significance of analyzing word lengths in a text file.
- Discuss how a word length histogram can provide insights into the structure and complexity of the text.

Remembering: Basic File Handling and Dictionary Usage:

- Provide a code snippet demonstrating how to open and read a text file in Python.
- Explain how to use a dictionary to store word lengths and their corresponding counts.

Applying: Word Length Histogram Program:

- Develop a Python program that reads an input text file and generates a histogram of word lengths. Utilize a dictionary to count the occurrences of each word length. Ensure the program excludes words with non-alphabetic characters.

TARGET-2:

Analyzing/Evaluating:

- Discuss the significance of sorting the word lengths in the histogram output.
- Evaluate the effectiveness of the program in providing a clear representation of word length distribution.

Group Programming Challenge: In a collaborative effort, the team will enhance the Python program to include advanced features and address potential challenges in handling diverse text data.

Advanced Features and Handling Challenges:

- Propose and implement enhancements to the program, such as handling case sensitivity or supporting multiple file formats.
- Discuss potential challenges in analyzing word lengths in languages with unique characters or symbols.
- Explore collaborative strategies for optimizing the program's efficiency with large text files.

Test Cases:

1. Test if the program correctly reads the input text file and handles different file formats such as .txt, .csv, or .docx.
2. Test the accuracy of word length calculation by providing input text files with known word lengths and comparing the program's output.
3. Validate the exclusion of non-alphabetic characters by inputting text files containing special characters and verifying that they are not counted in the word lengths.
4. Test the sorting functionality by providing input text files with randomized word lengths and ensuring that the histogram output is sorted in ascending or descending order.

7. **Scenario: Library Management System - Inheritance**

In the context of a Library Management System implemented in Python, let's explore the concept of inheritance. The system comprises the following classes:

Person Class:

- Represents a generic person in the library system.
- Attributes: **name**, **age**, **address**
- Methods: **__init__** (constructor), **display_info** (to display basic information)

Student Class (Inherits from Person):

- Represents a student in the library system.
- Additional Attributes: **student_id**, **grade**
- Additional Method: **display_student_info** (to display student-specific information)

Librarian Class (Inherits from Person):

- Represents a librarian in the library system.
- Additional Attributes: **employee_id**, **department**
- Additional Method: **display_librarian_info** (to display librarian-specific information)

Questions:

Understanding (Base Class):

- Explain the role of the **Person** class in the Library Management System.
- Describe the attributes (**name**, **age**, **address**) of the **Person** class and their significance.
- How does the **display_info** method in the **Person** class contribute to the overall functionality?

Remembering (Inherited Class - Student):

- List the attributes inherited by the **Student** class from the **Person** class.
- Explain the purpose of the **__init__** method in the **Student** class and how it relates to inheritance.
- Provide an example of creating an instance of the **Student** class and initializing its attributes.

Applying (Inherited Class - Librarian):

- Enumerate the attributes inherited by the **Librarian** class from the **Person** class.
- Discuss how the **display_librarian_info** method in the **Librarian** class extends the functionality of the base class.
- Write a code snippet demonstrating the creation of a **Librarian** object and setting its attributes.

TARGET-2:

Applying (Inherited Class - Librarian):

- Write a code snippet demonstrating the creation of a **Librarian** object and setting its attributes.

Analyzing (Inheritance Relationship):

- Compare and contrast the attributes of the **Student** and **Librarian** classes, highlighting similarities and differences.

- Discuss the benefits of using inheritance in the Library Management System design.
- Analyze how the **super()** function is utilized in the **__init__** method of both the **Student** and **Librarian** classes.

Evaluating/Creating (Group Challenge - System Enhancement):

- As a group task, propose an extension to the Library Management System that involves a new class inheriting from the **Person** class (e.g., **Author** or **Visitor**).
- Define relevant attributes and methods for the new class.
- Justify the choice of inheritance and how it enhances the overall system design.

Test Cases:

1. Test if the Person class correctly initializes with the provided name, age, and address attributes.
2. Validate the display_info method in the Person class by checking if it correctly displays the basic information of a person.
3. Verify the functionality of the display_student_info method in the Student class by checking if it properly displays student-specific information.
4. Test if the **init** method in the Student class properly initializes student-specific attributes when creating a Student object.
5. Test the overall system enhancement by creating objects of the new class (e.g., Author or Visitor) and ensuring that inheritance from the Person class works as intended.

8. **Scenario: E-commerce System - Polymorphism**

In the realm of an E-commerce System implemented in Python, let's explore the concept of polymorphism. The system consists of different product types, each represented by a class:

1. **Product Class:**
    - Represents a generic product in the E-commerce System.
    - Attributes: **name**, **price**
    - Methods: **__init__** (constructor), **calculate_discount** (to calculate the discount)
2. **Electronics Class (Inherits from Product):**
    - Represents electronic products in the E-commerce System.
    - Additional Attribute: **warranty_period**
    - Overridden Method: **calculate_discount** (specific implementation for electronic products)
3. **Clothing Class (Inherits from Product):**
    - Represents clothing items in the E-commerce System.
    - Additional Attribute: **size**
    - Overridden Method: **calculate_discount** (specific implementation for clothing items)

Questions:

Understanding (Base Class):

- Explain the role of the **Product** class in the E-commerce System.
- Describe the attributes (**name**, **price**) of the **Product** class and their significance.
- How does the **calculate_discount** method in the **Product** class exhibit polymorphism?

Remembering (Inherited Class - Electronics):

- List the attributes inherited by the **Electronics** class from the **Product** class.
- Explain the purpose of the **__init__** method in the **Electronics** class and how it relates to polymorphism.

Applying (Inherited Class - Clothing):

- Enumerate the attributes inherited by the **Clothing** class from the **Product** class.
- Discuss how the **calculate_discount** method in the **Clothing** class demonstrates polymorphism.

Analyzing (Polymorphism in Action):

- Compare and contrast the **calculate_discount** methods in the **Electronics** and **Clothing** classes, emphasizing their polymorphic behavior.
- Discuss the benefits of using polymorphism in the context of diverse product types in an E-commerce System.

TARGET-2:

Analyzing (Polymorphism in Action

- Analyze how the same method name (**calculate_discount**) can exhibit different behaviors based on the object's type.
- Provide an example of creating an instance of the **Electronics** class and initializing its attributes.
- Write a code snippet demonstrating the creation of a **Clothing** object and setting its attributes.

Evaluating/Creating (Group Challenge - Adding a New Product Type):

- As a group task, propose the introduction of a new product type (e.g., **Books** or **Furniture**) in the E-commerce System, represented by a new class inheriting from the **Product** class.
- Define relevant attributes and methods for the new class.
- Justify how polymorphism can be leveraged to seamlessly integrate the new product type into the existing system.

Test Cases:

1. Validate the calculate_discount method in the Product class by creating a generic product instance and verifying that the discount calculation behaves as expected.
2. Test the calculate_discount method in the Electronics class by creating an electronics product instance and ensuring that the specific discount calculation for electronics is applied correctly.
3. Verify the calculate_discount method in the Clothing class by creating a clothing product instance and confirming that the discount calculation specific to clothing items is executed properly.
4. Test the polymorphic behavior of the calculate_discount method by creating instances of both the Electronics and Clothing classes and observing how the method adapts to the respective product types.
5. Perform a stress test by creating a large number of instances of various product types (including potential new product types) and checking if polymorphism handles the diverse calculations efficiently and accurately.

9. **Scenario: Shopping Mall Inventory Management - OOP Concepts**

Imagine you are tasked with developing a Python program to manage inventory for a shopping mall. The program should utilize Object-Oriented Programming (OOP) concepts, including classes and inheritance, to represent different types of products and efficiently manage the inventory.

Product Class:

- Represents a generic product with attributes like **name**, **price**, and **quantity**.
- Methods include **__init__** (constructor), **calculate_total_price** (to calculate the total cost for a given quantity), and **display_product_info** (to print product details).

Inheritance - Electronics and Clothing Classes:

- **Electronics** and **Clothing** classes inherit from the **Product** class.
- **Electronics** has additional attributes like **brand** and **warranty_period**, while **Clothing** has attributes like **size** and **material**.

Shopping Cart Class:

- Represents a shopping cart that can hold various products.
- Methods include **add_to_cart** (adds a product to the cart), **calculate_cart_total** (calculates the total cost of items in the cart), and **generate_invoice** (prints an invoice with details of items purchased).

Questions:

Understanding (Product Class):

- Explain the purpose of the **Product** class in the context of managing shopping mall inventory.
- How does the **calculate_total_price** method contribute to the overall functionality of the **Product** class?

Remembering (Inheritance - Electronics and Clothing):

- Provide the code snippet for the **Electronics** class, highlighting its additional attributes and methods.Similarly, provide the code snippet for the **Clothing** class, emphasizing its unique attributes and methods.
- How does the concept of inheritance enhance code reusability and organization in this scenario?

Applying (Shopping Cart Operations):

- Discuss how the shopping cart efficiently handles different types of products, including electronics and clothing.
- How does the program ensure that the quantity of products in the cart does not exceed the available quantity in the inventory?

TARGET-2:

Applying (Shopping Cart Operations):

- Implement the **ShoppingCart** class with appropriate methods for adding products, calculating the cart total, and generating an invoice.

Analyzing (OOP Design Evaluation):

- Analyze the design choices related to using classes and inheritance in this inventory management system.
- Discuss the advantages and potential drawbacks of representing products as objects in terms of code structure and maintainability.

- Evaluate the role of the **ShoppingCart** class in managing diverse products and facilitating seamless shopping experiences.

Evaluating/Creating (Group Challenge - Discount System):

- As a group task, propose and implement a discount system for the shopping mall.
- Introduce a discount mechanism based on factors like total purchase amount or specific product categories.
- Discuss how the discount system enhances the shopping experience and encourages customer loyalty.

Test Cases:

1. Verify that the Product class initializes correctly by creating an instance and confirming that the attributes are set as expected.
2. Test the calculate_total_price method in the Product class by providing a quantity and ensuring that the total price is calculated accurately.
3. Create instances of the Electronics and Clothing classes to validate that the additional attributes are properly initialized and accessible.
4. Add products to the shopping cart and verify that they are stored correctly by checking the contents of the cart.
5. Calculate the total cost of the items in the shopping cart using the calculate_cart_total method and compare it with the expected total based on the added products.

10. **Scenario: Library Catalog System - Files with Exception Handling**

Imagine you are developing a Python program to manage a library catalog system. The program should read information about books from a file, allow users to borrow and return books, and handle potential errors using exception handling.

Book Class:

- Represents a book with attributes such as **title**, **author**, and **availability**.
- Methods include **__init__** (constructor), **borrow_book** (updates book availability), and **return_book** (resets book availability).

File Handling Functions:

- Implement functions to read book information from a file and update book information in the file.
- Include exception handling to manage issues like a missing file, incorrect file format, or unexpected errors during file operations.

Questions:

Understanding (Book Class):

- Explain the role of the **Book** class in the context of managing books in a library catalog.
- How does the **return_book** method contribute to the functionality of the **Book** class?

Remembering (File Reading Function):

- Provide the code for the function responsible for reading book information from a file.
- How does the program handle the scenario where the specified file is not found during the reading process?

- What types of exceptions might occur during file reading, and how are they handled?

Applying (File Updating Function):

- Implement the function responsible for updating book information in a file, reflecting borrow and return operations.
- Discuss the measures taken to ensure the correct format and structure of the file during the updating process.
- How are exceptions related to file updating managed in the program?

TARGET-2:

Analyzing (Exception Handling Strategies):

- Analyze the exception handling strategies employed in both file reading and updating functions.
- How does the program differentiate between various file-related exceptions, and what specific actions are taken for each?
- Evaluate the effectiveness of the exception handling approach in ensuring robust file operations.

Evaluating/Creating (Group Challenge - Transaction Logging):

- As a group task, propose and implement a transaction logging system to record borrowing and returning transactions.
- Create a log file that captures details such as the timestamp, book ID, transaction type (borrow or return), and user information.
- Discuss the benefits of having a transaction log for library management and how it enhances system reliability.

Test Cases:

1. Verify that the Book class initializes correctly by creating an instance and checking that the attributes are set as expected.
2. Test the borrow_book method in the Book class by attempting to borrow a book and confirming that the availability status changes accordingly.
3. Use a mock file to simulate a missing file scenario and ensure that the program handles it gracefully by raising the appropriate exception.
4. Validate the file reading function by providing a test file with correct format and content, then checking that the book information is read accurately.
5. Perform borrow and return operations on multiple books, and then check the updated file to ensure that the changes are reflected correctly without corrupting the file structure.