Question 1: Bank Transactions with Exception Handling

Steps:

1. Define a custom exception InsufficientFundsError to handle withdrawal errors.
2. Create a class BankAccount with:
    - A balance attribute initialized to 0.
    - A method deposit(amount):
        - Raise ValueError if amount <= 0.
        - Add the amount to balance.
    - A method withdraw(amount):
        - Raise InsufficientFundsError if amount > balance.
        - Deduct the amount from balance.
3. Implement exception handling while performing transactions.

Example:

```
account = BankAccount()
account.deposit(100)
account.withdraw(150)  # Should raise InsufficientFundsError
```

Question 2: Student Marks Processing and File Handling

Steps:

1. Define calculate_average(marks):
    - Check if the list is empty and raise ValueError.
    - Validate that all elements are numbers; if not, raise TypeError.
    - Compute the average.
2. Define save_marks_to_file(filename, marks):
    - Open the file in write mode.
    - Write marks to the file.
    - Handle potential IOError.

3. Define read_marks_from_file(filename):
    - Open the file in read mode.
    - Read marks and convert them to integers.
    - Handle FileNotFoundError and ValueError.

Example:

```
student_marks = [85, 90, 78]
avg = calculate_average(student_marks)
print("Average Marks:", avg)
save_marks_to_file("marks.txt", student_marks)
read_marks = read_marks_from_file("marks.txt")
print("Read Marks:", read_marks)
```

Output:

Average Marks: 84.33333333333333

Read Marks: [85, 90, 78]

Question 3: User Age Verification and Log File Management

Steps:

1. Define a custom exception UnderageError for age verification failures.
2. Implement verify_age(age):
    - If age < 18, raise UnderageError.
    - Otherwise, print a success message.
3. Implement log_error(error_message):
    - Open error.log in append mode.
    - Write the error message.
    - Handle IOError.

Example:

verify_age(16) # Should raise UnderageError

Question 4: Library Management System with Exception Handling

Scenario:

A library maintains a catalog of books, allowing users to borrow and return books. Each book has a title, author, and availability status. If a user tries to borrow a book that is already borrowed, the system should raise an exception.

Task:

Create a Book class with attributes:

- title (str), author (str), and available (bool, default True).

Create a Library class with:

- Attributes: A list of books.
- Methods:
    - add_book(book): Adds a book to the catalog.
    - borrow_book(title): Changes the book's availability to False if available; otherwise, raises a BookNotAvailableException.
    - return_book(title): Marks the book as available if found in the catalog.

Define a custom exception BookNotAvailableException for handling book unavailability.

Steps to Solve:

1. Create a Book class with the required attributes.
2. Define a Library class that manages books.
3. Implement add_book(book), borrow_book(title), and return_book(title).
4. Define a custom exception BookNotAvailableException.
5. Write a test case where a user tries to borrow an already borrowed book, and handle the exception.

Example:

```
library.add_book("Python Programming")
library.add_book("Data Science Handbook")
library.borrow_book("Python Programming") # Borrow an available book
library.borrow_book("Python Programming") # Attempt to borrow the same book again
library.display_books() # Display available books after borrowing
```

Output:

```
Book 'Python Programming' has been borrowed.
Error: Book 'Python Programming' is not available in the library.
Available books:
- Data Science Handbook
```

[Bonus Question]: Anagrams

A student is taking a cryptography class and has found anagrams to be very useful. Two strings are anagrams of each other if the first string's letters can be rearranged to form the second string. In other words, both strings must contain the same exact letters in the same exact frequency. For example, bacdc and dcbac are anagrams, but bacdc and dcbad are not.

The student decides on an encryption scheme that involves two large strings. The encryption is dependent on the minimum number of character deletions required to make the two strings anagrams. You need to determine this number.

Given two strings, a and b, that may or may not be of the same length, determine the minimum number of character deletions required to make a and b anagrams. Any characters can be deleted from either of the strings. The strings a and b consist of lowercase English alphabets.

Example:

```
a = 'cde'
b = 'dcf'
```

Delete ‘e’ from a and ‘f’ from b so that the remaining strings are ‘cd’ and ‘dc’ which are anagrams. This takes 2 character deletions.

Function Description:

Create a `makeAnagram` function below.

Inputs:

string a: a string
string b: another string

Output:

int: the minimum total characters that must be deleted
