1. Case Study: Online Shopping Cart Exception Handling

You are working as a Python developer for an e-commerce company, and your team is responsible for building and maintaining the shopping cart module of the website Customers can add items to their cart, view the cart contents, and proceed to checkout.

Recently, there have been reports of unexpected crashes and errors when customers interact with their shopping carts. Your task is to investigate these issues and improve the exception handling in the shopping cart code to make it more robust.

Requirements and Scenarios:

Scénario 1 - Adding Items to Cart:

When a customer adds an item to their cart, they provide the product ID and quantity. Handle exceptions that may occur during this process, such as:

i. Product ID not found in the product catalog.

i. Invalid quantity (e.g., negative quantity or non-integer input).

```
import sqlite3

def register_customer(name, email, password):
    try:
        conn = sqlite3.connect("bookstore.db")
        cursor = conn.cursor()

        cursor.execute("SELECT COUNT(*) FROM customers WHERE email = ?", (email,))
        count = cursor.fetchone()[0]
        if count > 0:
            raise ValueError("Email address already exists.")

        cursor.execute("INSERT INTO customers (name, email, password) VALUES (?, ?, ?)", (r
        conn.commit()

        print("Registration successful!")

    except sqlite3.Error as e:
        print(f"Database error: {e}")
    except ValueError as e:
        print(f"Error: {e}")
    except Exception as e:
```

```
    finally:
        conn.close()

try:
    register_customer("John Doe", "john@example.com", "password123")
except Exception as e:
    print(f"An error occurred: {e}")
```

```
Database error: no such table: customers
```

Scenario 2-Viewing Cart Contents:

When a customer views their cart, display the list of items and their quantities. Handle exceptions that may occur during this process, such as:

i. Empty cart (no items added).

ii. Unexpected errors (e.g., network issues when fetching cart data).

Scenario 3-Proceeding to Checkout:

```
import sqlite3

def add_book(title, author, quantity):
    try:
        conn = sqlite3.connect("bookstore.db")
        cursor = conn.cursor()

        if not title or not author or quantity <= 0:
            raise ValueError("Invalid book data.")

        cursor.execute("INSERT INTO inventory (title, author, quantity) VALUES (?, ?, ?)",
        conn.commit()

        print("Book added to inventory!")

    except sqlite3.Error as e:
        print(f"Database error: {e}")
    except ValueError as e:
        print(f"Error: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
    finally:
        conn.close()
```

```
try:
    add_book("Book Title", "Author Name", 10)
except Exception as e:
    print(f"An error occurred: {e}")
```

```
Database error: no such table: inventory
```

Scenario 3-Proceeding to Checkout:

When a customer proceeds to checkout, validate the cart and process the payment. Handle exceptions that may occur during this process, such as:

i. Insufficient stock for some items in the cart.

ii. Payment gateway errors.

iii. Customer payment method declined.

Your Tasks:

1. Review the existing shopping cart code to identify potential areas where exceptions may occur.
2. Enhance the exception handling in the code by adding appropriate try, except, and finally blocks to handle exceptions gracefully. Provide helpful error messages to the user where applicable.
3. Ensure that the program continues to run smoothly even when exceptions occur, and customers receive informative feedback.
4. Test the shopping cart thoroughly with different scenarios to ensure that it handles exceptions correctly.

```
import sqlite3

def place_order(customer_id, books):
    try:
        conn = sqlite3.connect("bookstore.db")
        cursor = conn.cursor()

        for book_id, quantity in books:
            cursor.execute("SELECT quantity FROM inventory WHERE book_id = ?", (book_id,))
            available_quantity = cursor.fetchone()[0]
            if quantity > available_quantity:
                raise ValueError(f"Insufficient stock for book ID {book_id}.")
```

```
        cursor.execute("INSERT INTO orders (customer_id) VALUES (?)", (customer_id,))
        order_id = cursor.lastrowid

        for book_id, quantity in books:
            cursor.execute("INSERT INTO order_details (order_id, book_id, quantity) VALUES
            conn.commit()

        print("Order placed successfully!")

    except sqlite3.Error as e:
        print(f"Database error: {e}")
    except ValueError as e:
        print(f"Error: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
    finally:
        conn.close()

try:
    place_order(1, [(1, 2), (2, 3)])
except Exception as e:
    print(f"An error occurred: {e}")
```

```
 Database error: no such table: inventory
```

Scenario 4-Order History:

Customers should be able to view their order history, which includes details of past orders.

Handle exceptions that may occur when retrieving order history, such as:

1. No orders found for the customer.
2. Database connection issues

```
import sqlite3

def get_order_history(customer_id):
    try:
        conn = sqlite3.connect("bookstore.db")
        cursor = conn.cursor()

        cursor.execute("SELECT * FROM orders WHERE customer_id = ?", (customer_id,))
        orders = cursor.fetchall()
```

```
        if not orders:
            raise ValueError("No orders found for the customer.")

        for order in orders:
            cursor.execute("SELECT * FROM order_details WHERE order_id = ?", (order[0],))
            order_details = cursor.fetchall()

        print("Order history retrieved successfully!")

    except sqlite3.Error as e:
        print(f"Database error: {e}")
    except ValueError as e:
        print(f"Error: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
    finally:
        conn.close()

try:
    get_order_history(1)  # Customer ID
except Exception as e:
    print(f"An error occurred: {e}")
```

```
Database error: no such table: orders
```

Your Tasks:

1. Review the existing database interaction code to identify potential areas where exceptions may occur.

```
import sqlite3

try:
    conn = sqlite3.connect("example.db")

    cursor = conn.cursor()

    cursor.execute("INSERT INTO users (name, email) VALUES (?, ?)", ("John Doe", "john@exar

    conn.commit()

except sqlite3.Error as e:
    print(f"Database error: {e}")
```

```
except Exception as e:
    print(f"An unexpected error occurred: {e}")

finally:
    conn.close()
```

```
 Database error: no such table: users
```

2. Enhance the exception handling in the code by adding appropriate try, except, and finally blocks to handle exceptions gracefully. Provide helpful error messages to the user where applicable.

```
import sqlite3

try:
    conn = sqlite3.connect("example.db")

    cursor = conn.cursor()


    cursor.execute("INSERT INTO users (name, email) VALUES (?, ?)", ("John Doe", "john@exar

    conn.commit()

    print("User 'John Doe' added successfully!")

except sqlite3.Error as e:
    print(f"Database error: {e}")

except ValueError as e:
    print(f"Value error: {e}")

except Exception as e:
    print(f"An unexpected error occurred: {e}")

finally:
    try:
        conn.close()
    except Exception as e:
        print(f"Error while closing the database connection: {e}")
```

```
Database error: no such table: users
```

3. Ensure that the program continues to run smoothly even when exceptions occur, and customers receive informative feedback.

```
import sqlite3

def add_user_to_database(name, email):
    try:
        conn = sqlite3.connect("example.db")

        cursor = conn.cursor()

        cursor.execute("INSERT INTO users (name, email) VALUES (?, ?)", (name, email))

        conn.commit()

        print("User added successfully!")

    except sqlite3.Error as e:
        print("Database error:", e)
        print("Please try again later or contact support.")

    except Exception as e:
        print("An unexpected error occurred:", e)
        print("Please try again later or contact support.")

    finally:
        try:
            conn.close()
        except Exception as e:
            print("Error while closing the database connection:", e)

try:
    add_user_to_database("John Doe", "john@example.com")
except Exception as e:
    print("An error occurred:", e)
```

```
Database error: no such table: users
Please try again later or contact support.
```

4. Implement database queries and transactions following

4. Implement database queries and transactions following best practices to maintain data integrity.

```
import sqlite3

def insert_user_to_database(name, email):
    try:
        conn = sqlite3.connect("example.db")

        cursor = conn.cursor()

        conn.execute("BEGIN TRANSACTION")

        cursor.execute("INSERT INTO users (name, email) VALUES (?, ?)", (name, email))

        conn.commit()

        print("User added successfully!")

    except sqlite3.Error as e:
        conn.rollback()
        print("Database error:", e)
        print("Please try again later or contact support.")

    except Exception as e:
        conn.rollback()
        print("An unexpected error occurred:", e)
        print("Please try again later or contact support.")

    finally:
        try:
            conn.close()
        except Exception as e:
            print("Error while closing the database connection:", e)

try:
    insert_user_to_database("John Doe", "john@example.com")
except Exception as e:
    print("An error occurred:", e)
```

```
Database error: no such table: users
Please try again later or contact support.
```

5. Test the website's database interactions thoroughly with different scenarios to ensure that it handles exceptions

different scenarios to ensure that it handles exceptions correctly.

```
Yes all are correct
```

6. Read a text file containing a list of names or numbers, sort the data, and write the sorted data back to a new file.

```
def read_and_sort(input_file, output_file):
    try:
        with open(input_file, 'r') as file:
            data = file.readlines()
        data = [line.strip() for line in data]

        data.sort()

        with open(output_file, 'w') as file:
            file.write('\n'.join(data))

        print("Data sorted and written to", output_file)

    except FileNotFoundError:
        print("Input file not found.")
    except Exception as e:
        print("An error occurred:", e)

input_file = "unsorted.txt"
output_file = "sorted.txt"
read_and_sort(input_file, output_file)
```

```
 Input file not found.
```

7. Write a Python script that compares two text files and identifies the differences between them, including added, modified, and deleted lines

```
import difflib
```

```
  '

def compare_files(file1, file2):
    try:
        with open(file1, 'r') as f1:
            file1_lines = f1.readlines()

        with open(file2, 'r') as f2:
            file2_lines = f2.readlines()

        differ = difflib.Differ()
        diff = list(differ.compare(file1_lines, file2_lines))

        added_lines = [line[2:] for line in diff if line.startswith('+ ')]
        modified_lines = [line[2:] for line in diff if line.startswith('? ')]
        deleted_lines = [line[2:] for line in diff if line.startswith('- ')]

        return added_lines, modified_lines, deleted_lines

    except FileNotFoundError:
        print("One or both files not found.")
        return [], [], []

    except Exception as e:
        print("An error occurred:", e)
        return [], [], []


file1_path = "/content/file1.txt"
file2_path = "/content/file2.txt"
added, modified, deleted = compare_files(file1_path, file2_path)

print("Added Lines:")
print("\n".join(added))
print("\nModified Lines:")
print("\n".join(modified))
print("\nDeleted Lines:")
print("\n".join(deleted))
```

```
Added Lines:
Speaking at the 2019 Conscious Companies Awards in Johannesburg, Iskander explained t

Modified Lines:


Deleted Lines:
Speaking at the 2019 Conscious Companies Awards in Johannesburg, Iskander explained t
```

8. Develop a Python program that compresses a large text

file using a compression algorithm (eg, gzip) and then decompresses it back to its original form.

```
import gzip

def compress_file(input_file, compressed_file):
    try:
        with open(input_file, 'rb') as f_in:
            with gzip.open(compressed_file, 'wb') as f_out:
                f_out.writelines(f_in)
        print(f"File '{input_file}' compressed to '{compressed_file}'")

    except FileNotFoundError:
        print("Input file not found.")
    except Exception as e:
        print("An error occurred:", e)

def decompress_file(compressed_file, output_file):
    try:
        with gzip.open(compressed_file, 'rb') as f_in:
            with open(output_file, 'wb') as f_out:
                f_out.writelines(f_in)
        print(f"File '{compressed_file}' decompressed to '{output_file}'")

    except FileNotFoundError:
        print("Compressed file not found.")
    except Exception as e:
        print("An error occurred:", e)


input_file = "large_text_file.txt"
compressed_file = "compressed_file.gz"
output_file = "decompressed_text_file.txt"


compress_file(input_file, compressed_file)


decompress_file(compressed_file, output_file)
```

```
Input file not found.
Compressed file not found.
```

9. Read a binary file (e.g., an image or audio file) in Python

and perform an operation, such as resizing an image or modifying audio data.

```
from PIL import Image


with Image.open('/content/hd.jpg') as img:
    new_size = (300, 200)

    resized_img = img.resize(new_size)

    resized_img.save('output_image.jpg')

print('Image resized and saved as output_image.jpg')
```

```
 Image resized and saved as output_image.jpg
```

10. Write a python program to Combine the contents of multiple text files into a single file using Python. Each file should be appended to the end of the resulting file.

```
def combine_text_files(input_files, output_file):
    try:
        with open(output_file, 'w') as output:
            for input_file in input_files:
                with open(input_file, 'r') as file:
                    content = file.read()
                    output.write(content)

                    output.write("\n==== End of File ====\n")

        print(f"Combined files into {output_file}")

    except FileNotFoundError:
        print("One or more input files not found.")
    except Exception as e:
        print("An error occurred:", e)

input_files = ["file1.txt", "file2.txt", "file3.txt"]
output_file = "combined_file.txt"
```

```
combine_text_files(input_files, output_file)
```

```
One or more input files not found.
```