Assignment 3: Python Programming Concepts

Instructions:

- Submit your solution as a **Jupyter Notebook** (`.ipynb`).
- Each task should be solved in a separate code cell.
- Include comments explaining your code.

Task 1: E-commerce Data Processing

You are tasked with building a system to handle order and customer data for an online store. The system needs to use **lambda functions**, Python's **built-in functions** (e.g., `map()`, `filter()`, `reduce()`), and proper **exception handling**.

Part A: Data Validation

You are given a list of dictionaries where each dictionary represents an order with customer details.

```
orders = [
    {"customer": "Alice", "total": 250.5},
    {"customer": "Bob", "total": "invalid_data"},
    {"customer": "Charlie", "total": 450},
    {"customer": "Daisy", "total": 100.0},
    {"customer": "Eve", "total": -30},  # Invalid total
]
```

Write a function that:

- Uses a **lambda function** with the `filter()` built-in function to filter out invalid orders where the `total` is either non-numeric or less than zero.
- Uses **exception handling** to handle any type conversion issues.
- Return the filtered valid orders as a list of dictionaries.

Part B: Discount Application

After validating the orders, the store is offering a 10% discount on all orders above $300.

Write a function that:

- Uses the **`map()`** function with a **lambda** to apply the discount to qualifying orders.
- Returns a new list with the updated totals for each customer.

Part C: Total Sales Calculation

Use the **`reduce()`** function with a **lambda** to:

- Calculate the total sales from the list of valid orders (after applying discounts).

Task 2: Iterator and Generator

Part A: Custom Iterator

Create a custom iterator class `SquareIterator` that:

- Takes an integer `n` and iterates over the first `n` natural numbers, yielding their squares.

Part B: Fibonacci Generator

Write a **generator function** `fibonacci_generator()` that:

- Yields the Fibonacci sequence up to the number `n`.

Task 3: Exception Handling and Function Decorator

You need to implement robust exception handling in the system.

Part A: Chained Exceptions

Write a function that:

- Takes a list of numbers and tries to divide each number by a divisor.
- If the divisor is zero, raise a custom exception.
- If any other error occurs (e.g., non-numeric input), raise an appropriate exception and **chain** it to the custom exception to provide context.

Part B: Exception Logging Decorator

Create a decorator that:

- Logs exceptions raised during the execution of a function.
- It should print the exception type, message, and the function where the exception occurred.