**Exercise 1: Compress a String (Run-Length Encoding)**

Write a program that compresses a string by replacing consecutive repeated characters with the character followed by the count.

If the compressed string is not shorter, return the original string.

Compress the string, but **uppercase the character** if its count > 1,  
and **lowercase** it otherwise.

**Input**: aabcccdee

**Output**: A2b1C3d1E2

**Exercise 2: Object-Oriented Programming**

Problem: *Bank Account Management System*

Create a Python class called **BankAccount** that simulates a simple bank account.

**Requirements**

1. Class should have:
   * account\_holder (string)
   * balance (float, default = 0)
2. Methods:
   * deposit(amount) → adds to balance
   * withdraw(amount) → deducts if balance is sufficient
   * display() → shows account details
3. In the main program:
   * Create an account for the user.
   * Ask them to deposit and withdraw money.
   * Display the final balance.

**Exercise 1: Full-Stack Blog Project (Django + DRF + React)**

***With User Authentication, Likes, and Comments***

**Project Overview**

In this project, students will build a full-stack blog application using Django REST Framework (backend) and React (frontend).  
The app will allow users to register, log in, create posts, like/unlike posts, and comment on posts.

**Main Features**

**User Authentication**

* Users can register and log in using JWT (JSON Web Tokens).
* Only authenticated users can create, edit, or delete their own posts.
* Logged-in users can like/unlike posts and write comments.

**Blog Posts**

* **Each post should have:**
  + Title
  + Content
  + Author (linked to the logged-in user)
  + Created and updated timestamps
* **Users can:**
  + Create new posts
  + Edit or delete their own posts
  + View all posts (everyone can view)

**Likes**

* Users can like or unlike any post (once per user).
* The post should display a like count.

**Comments**

* Users can add comments to any post.
* Comments show the author name and text.
* Everyone can view comments.

**Technology Stack**

| **Layer** | **Technology** |
| --- | --- |
| **Backend** | **Django, Django REST Framework (DRF), SimpleJWT** |
| **Frontend** | **React (Vite), Axios, React Router DOM** |
| **Database** | **SQLite (default)** |
| **Authentication** | **JWT (JSON Web Tokens)** |

**Test Flow**

1. Register → /api/auth/register/
2. Login → /api/auth/login/
3. Copy JWT token → localStorage (handled by React)
4. Create Post → /api/posts/
5. View All Posts → /api/posts/
6. Like/Comment on Post → /api/posts/:id/like/ & /api/comments/

**Learning Outcomes**

By completing this project:

* Understand **frontend-backend integration**.
* Implement **JWT authentication** with DRF and React.
* Work with **REST APIs** and **Axios**.
* Build **CRUD functionality** in a full-stack environment.
* Manage **state, navigation, and user sessions** in React.