The Simple Online Voting System project allows users to vote for candidates, tracks votes, and provides an admin interface to view results and reset the voting process, all implemented in Python.

Simple Online Voting System - Python Project

**key**

* Basic Python programming
* Working with lists and dictionaries
* Handling user input
* Using control structures (if-else, loops)

# Steps to Develop

# Step 1: Understand the Problem and Define Requirements

* **Objective**: Understand the basic features required in the voting system.
  + Users should be able to vote for one of the available candidates.
  + The system should prevent users from voting more than once.
  + Admin should be able to view voting results and reset the voting system.
  + The system should display confirmation messages to users after voting.

### Step 2: Define Data Structures

* **Objective**: Define how to store data in the system.
  + Use a **dictionary** or **list** to store the candidates and their respective vote counts.
    - Example: {'Candidate 1': 0, 'Candidate 2': 0, 'Candidate 3': 0}
  + Use a **set** or **list** to track users who have voted (so they cannot vote again).
    - Example: {'user1', 'user2'}

### Step 3: Set Up Basic Program Structure

* **Objective**: Create the basic structure of the Python program with functions to handle the different tasks.
  + Create functions like:
    - **Display candidates**: Show the list of candidates to the user.
    - **Vote**: Allow the user to vote for a candidate.
    - **View results**: Show the current vote count for each candidate.
    - **Reset voting**: Allow the admin to reset votes.

### Step 4: Create Function to Display Candidates

* **Objective**: Write a function to display the available candidates to the user.
  + This function should list all candidates and their vote counts.

### Step 5: Create Function to Allow Users to Vote

* **Objective**: Create a function where users can cast their vote.
  + The function should ask the user to choose a candidate.
  + Check if the user has already voted (using the set of users who voted).
  + If the user hasn’t voted, allow them to vote, and add their ID to the set of voters.
  + Display a confirmation message after voting.

### Step 6: Create Admin Function to View Results

* **Objective**: Create an admin function to view the voting results.
  + The function should show the current vote counts for all candidates.

### Step 7: Create Admin Function to Reset Votes

* **Objective**: Allow the admin to reset the votes if needed.
  + This function should reset the vote counts for all candidates to zero and clear the list of voters.

### Step 8: Set Up the Main Program Loop

* **Objective**: Create a main loop where users can choose to vote, view results, or reset the voting (admin only).
  + Use a while loop to continuously ask users for their input until they decide to exit.
  + Provide options like:
    - Vote
    - View results (admin only)
    - Reset votes (admin only)
    - Exit the program

### Step 9: Implement User Input and Error Handling

* **Objective**: Ensure the program handles invalid inputs gracefully.
  + Use try-except blocks to catch invalid inputs (e.g., a user entering a non-existent candidate).
  + Ensure the program asks the user for valid choices if they enter something wrong.

### Step 10: Testing

* **Objective**: Test the system with different scenarios.
  + Test as a regular user to ensure they can vote and view results.
  + Test as an admin to ensure the reset and result-viewing functions work correctly.
  + Test edge cases like trying to vote twice or viewing results before anyone votes.

**Code Structure Overview**

## Data Structures**:**

* + Dictionary to store candidates and their vote counts.
  + Set or list to store users who have voted.

## Functions**:**

* + display\_candidates(candidates): Displays all candidates and vote counts.
  + vote(candidates, voters): Allows users to cast their vote.
  + view\_results(candidates): Allows the admin to see voting results.
  + reset\_votes(candidates, voters): Allows the admin to reset the voting.

## Main Program Loop**:**

* + Continuously prompts the user (or admin) for actions like voting, viewing results, or resetting votes.

**Testing Scenarios:**

## User Testing**:**

* + Test that a user can vote for a candidate.
  + Test that a user can’t vote twice.

## Admin Testing**:**

* + Test that the admin can view the results.
  + Test that the admin can reset votes.

## Edge Cases**:**

* + Handle invalid inputs, like entering an incorrect candidate name.
  + Ensure the program handles invalid admin passwords.

**Submission Guidelines**

## Code Structure**:**

* + Organize the code logically with separate functions for different tasks (voting, viewing results, resetting votes).
  + Include comments in the code to explain what each function does and how it works.

## README File**:**

* + Provide a **README.md** file with the following sections:
    - **Project Overview**: A brief description of the voting system.
    - **Features**: Describe key features like voting, viewing results, and resetting votes.
    - **Setup Instructions**: Instructions on how to run the project.
    - **Usage**: How users and admins can interact with the system.

## Code Documentation**:**

* + Ensure the code is well-documented with comments, especially around complex logic.
  + Include clear explanations in the **README** for users to understand how to run and use the system.

## Testing**:**

* + Document the testing process in the **README**, explaining how you tested the program and any issues or bugs encountered.

**Key Learning Outcomes**

* **Random Number Generation**: Using Python’s random module.
* **Loops**: Writing loops to allow multiple attempts.
* **Conditionals**: Using if-else statements to compare guesses.
* **User Input and Data Types**: Converting inputs and handling basic user interaction.

This project is perfect for beginners to get comfortable with foundational Python programming concepts while building an interactive game.

Create simple voting system where main program loop continuously reminds the individuals for actions like voting, viewing results, resetting votes, with vote(candidates, voters) function and these functions must have admin access only using while loop, allow Individuals to vote for any one candidate only such that it must ask every individual which candidate they want to choose continuously until they decide to exit, and also check if that individual has already voted or not, if not voted then let them vote with ID, then display confirmation message after voting, for any one of the invalid input display invalid input for that use try-except blocks for input and error handling, also confirm that every individual get right voting choices to vote without any error, define storing data in the dictionary and use set or list to track vote of every individual voters in the system, use function display\_candidates(candidates) to display candidates and their vote counts list to every individual, allow individual to vote for a candidate, view results showing current vote count for each candidate, use view\_results(candidates) function to view results as admin interface, use reset\_votes(candidates, voters) function to allow admin to reset votes if essential in voting system in setting option, then try edge cases and test entire system made

**METHOD 1:**

Here candidates details will be provided in the list just like voting system. As an admin we must know the mandatory admin password as only admin knows the password and who can view the result. Users can only vote to anyone candidate by giving vote.

