COS 125-HMWK6 - Simulated Election System

This assignment requires you to build the core logic for a simplified election using Python lists and dictionaries, logic structures, repetition structures, and functions. The focus is on simulating voting processes and tallying results using complex data structures.

I. General Rules & Academic Integrity

- **Topics:** Practicing elements and concepts covered in class: Getting user input, data type conversion, manipulating lists/strings, implementing decision/control structures (if/else), loops, complex data types, and functions.
- **Concepts Used:** Only use concepts presented in the course to-date. Using advanced concepts or concepts from outside the course does not demonstrate understanding of the required material and may result in a zero for the assignment.
- **Restrictions:** Do not use breaks, global values (unless working with shallow copy), or modules other than random.
- **Resources:** You may reference the textbook, class materials, and tutoring (if you use a tutor, list their name in collaboration statement).
- **Academic Integrity:** ***Do not use AI***. Asking AI, online searches, or people not associated with the course (results in a grade of zero) is not allowed and many times leads you to incorrect solutions. Specify any collaboration you had within the comment header. If you went to Boardman 138, state that.
- **Planning:** Start early. Planning out the program helps. Create a written plan of the processes in the program, how values will be passed, what values will be modified, processes, data flow, indicate all logic and inputs/outputs required.

II. Submission Requirements

- **File Name:** yourName_election.py (replace "yourName" with your actual name, e.g., lauraGurney_election.py).
- **Comment Header:** Include the required comment header in your file and ensure data within the header is updated.
- **Folder Name:** Place the .py file into a folder named yourName_electionSim.
- **Submission:** Keep python files inside your assignment folder. Compress the folder (.zip) and submit the compressed file containing all project files.

III. Assignment Tasks & Logic

Begin by planning what you will need, how things are processed, and passed around. Read the entire assignment first. The program will simulate two main components: the voters and the election results.

- **Voter List:** A <u>**list of dictionaries**</u> where each dictionary represents a voter and stores their ranked candidate preferences and a voting weight.
- **Result Dictionary:** A **dictionary** tracking candidate names and their current vote count.

Tasks

1. **Submit Planning:** Submit a flowchart, outline, or other planning document showing how you planned out the program. Images of whiteboard drawings, etc. will be accepted. Place in the submission folder with the .py file.
2. Create a function to generate the voter's voting record that takes the **number of voters** (N) and a list of candidate names as input.
   - The function must create and return a **list of dictionaries** representing each voter's mocked up voting register.
   - To get the candidate names, ask the user for 5 names.
   - With these names each voter needs:
     - a voter (dictionary) record contain:
       1. 'voterID': A unique integer randomly generated and recorded. But check for uniqueness.
       2. 'voterPreference': A list containing the candidate's names (entered by the user above) for the voter to rank.
       3. 'voteWeight': A random integer between 1 and 5 (using random.randint()) representing the voter's weighted selection for each of the candidates.

     Example: if the voter has 5 candidates to vote for: Henry, Pat, Claire, Sally, George then the weights indicate the ranking of choice – 5, 3, 1, 2, 4 meaning Henry is 5th choice, Pat is 3rd Claire is 1st, Sally 4th, and George is 4th choice.

3. Create a function that takes the voting record as input. The input is a list of dictionaries, where each dictionary represents a voter and candidate names with the ranks (e.g., {"Henry": 1, "Claire": 3}).
   - Since the voters were given the candidate names in random order that they then were asked to rank by choice 1st to 5th (highest choice to lowest choice) you'll need to create a function that takes the voters information and determines each of the candidates sum of votes and count of voters ranking the candidate as 1st.
     - Rank is determined by the candidate with the lowest score that was ranked first by most voters.
     - The function returns a dictionary of the total scores for each candidate separate from the voter information.

4. Create a function called to determine the winner takes the candidate score dictionary a passed value.
   - Uses the dictionary passed in to determine and return who wins
   - Include logic to handle ties and lists who tied
5. Controller in the `main()` function:
   - Ask the user for the number of voters to simulate (N) and the list of candidate names (minimum 3).
   - Use the `main()` to control the flow of data between functions and receive results from the functions.
   - Display the results of the election.
6. Allow the user to run the election again if they'd like to.

IV. Grading Criteria

Grading will be based on: Implementing course content correctly. Implementing logic, loops, and data types properly.

Criteria	**Exceeds (Max Pts)**	**Meets (Max Pts)**	**Partially Meets (Max Pts)**	**Does not meet (Max Pts)**	**Not evident (Max Pts)**
Interpretation	**10 points**: The code is concise, clear, and efficient. It uses insightful representations to model the problem. The student has a deep and thoughtful understanding of the project and problem requirements.	**8.5 points**: The student accurately converts the problem's requirements into code. All inputs are correctly interpreted, and the processes are successful and comprehensive.	**7.5 points**: The student attempts to model the problem but makes minor errors in their interpretation. For instance, they might correctly model the loop but make a small error.	**6.5 points**: The student's code fundamentally misunderstands the problem. They may attempt to use programmatic structures but fail to correctly represent the problem model.	**0 points**: Nothing evident.
Function Implementation & Abstraction	**12 points**: All required functions are defined, correctly manage	**10.2 points**: All required functions are defined	**9 points**: Most required functions are defined, but	**7.8 points**: Few required functions are defined or	**0 points**: The program fails to

Criteria	Exceeds (Max Pts)	Meets (Max Pts)	Partially Meets (Max Pts)	Does not meet (Max Pts)	Not evident (Max Pts)
	inputs/outputs, and demonstrate optimal abstraction. The program uses functions to strictly separate logic, resulting in highly modular and readable code.	and correctly manage inputs/outputs (parameters and return values). The program uses functions to organize logic into separate, distinct tasks, fulfilling the abstraction requirement.	there are minor errors in parameter passing and/or return values. The code attempts to use functions for organization but may include some logic that fails the abstraction.	used. Function definitions are present but contain significant logical and/or structural errors (e.g., incorrect scope, missing return statements, major parameter issues). Function usage is inconsistent.	define or use functions entirely, or the functions are so flawed that they do not contribute to solving the problem.
Logical Structures	**12 points**: The program correctly uses logical decision structures (e.g., if/elif/else) to handle all possible scenarios, including edge cases. The use of these structures is efficient and well-placed.	**10.2 points**: The program correctly uses if/elif/else to handle the core logic and ensure the program doesn't crash on invalid input, if applicable.	**9 points**: The program attempts to use conditional logic but makes minor errors that may cause the program to fail or produce incorrect logical outputs under certain conditions.	**7.8 points**: The program fails to use conditional logic.	**0 points**: Nothing evident.
Looping and Iterations	**12 points**: The student implements	**10.2 points**: The student correctly	**9 points**: Loop(s) not implemented	**7.8 points**: The student fails to	**0 points**: Nothing evident.

Criteria	Exceeds (Max Pts)	Meets (Max Pts)	Partially Meets (Max Pts)	Does not meet (Max Pts)	Not evident (Max Pts)
	looping structures correctly to complete required processes perfectly. Loop(s) terminate correctly and efficiently, and the variables are updated within and nest, if used is correctly implemented.	implements loop(s). Functions as intended, terminating when the initial cost is covered. Minor issues may be present.	or contain logical errors (e.g., infinite loop, incorrect termination condition) that prevent from functioning properly.	implement a while loop, or uses an incorrect loop type and or structure. Not Implemented correctly to solve specified challenge.	
Data Handling	**10 points**: All inputs are stored in self-documenting variables, and the program demonstrates robust data conversion and handling of program data and user inputs.	**8.5 points**: The program correctly prompts the user for all inputs, stores them in descriptive variables, and converts data types correctly for calculations. Handles program data properly with minimal errors.	**7.5 points**: The program prompts for some but not all inputs, or the variable names are not clear. There may be errors in data type conversion that led to incorrect output.	**6.5 points**: The program does not prompt for all inputs or fails to store them correctly. Data type conversion is either not attempted or is consistently incorrect.	**0 points**: Nothing evident.
Communication & Formatting	**8 points**: The program's output is highly effective	**6.8 points**: The program	**6 points**: The program's output does	**5.2 points**: The program fails to	**0 points**: Nothing evident.

Criteria	**Exceeds (Max Pts)**	**Meets (Max Pts)**	**Partially Meets (Max Pts)**	**Does not meet (Max Pts)**	**Not evident (Max Pts)**
	and easy to read. It uses excellent formatting (\t, \n) to display results of program clearly and concisely.	successfully adheres to all output specification standards. The readability and user experience output could be more clearly formatted and effective.	not effectively meet the overall purpose of the program/problem. The output is presented without a clear explanation of what is meant, lacking formatting and clarity.	provide adequately formatted output. It may use vague, hard to read, or the output is completely absent or incomprehensible.	
File & Folder Organization	**8 points**: The file and folder naming and organization are professional, clear, and logical.	**6.8 points**: The file and folder naming and organization are correct.	**6 points**: The file or folder naming is slightly off, but the file is still recognizable.	**5.2 points**: The file and folder naming and organization are incorrect, making the file difficult to find or identify.	**0 points**: Nothing evident.
Header Comments	**8 points**: The student includes a comment header that is well-formatted and includes all of the required information along with a brief description of the program's functionality.	**6.8 points**: Comment header present without required information and is significantly lacking relevant information.	**6 points**: Comment header in present without required information and is significantly lacking relevant information.	**5.2 points**: Comment header in present without required information and is significantly lacking relevant information.	**0 points**: Nothing evident—Zero for assignment.

Criteria	**Exceeds (Max Pts)**	**Meets (Max Pts)**	**Partially Meets (Max Pts)**	**Does not meet (Max Pts)**	**Not evident (Max Pts)**
Planning Documentation	**10 points**: The documentation (flowchart/pseudocode) is fully comprehensive, accurately mapping all functions, data flow, variable modifications, and complex logic. It demonstrates an exceptional understanding of the problem structure before coding.	**8.5 points**: The documentation is complete, clearly outlining the main functions, the passing of primary data structures, and the general sequence of the program. It accurately addresses the required inputs and outputs for core functions.	**7.5 points**: The documentation is present but incomplete or lacks clarity. It might skip key functions or fail to clearly indicate handling of data and processes between steps.	**6.5 points**: The documentation is minimal or confusing. It fails to map the core dependencies or uses unclear symbols/formats, making it difficult to follow the intended program flow.	**0 points**: No planning documentation (flowchart, pseudocode, or design diagram) is submitted or provided.
Program Comprehensiveness & Adherence to Instructions/Problem Model	**10 points**: The submission is complete, meeting all functional, structural, and abstract requirements (e.g., all prompts answered, all code functions, all of the program parts model addressed). All submission rules are followed perfectly.	**8.5 points**: The submission meets all core functional requirements and follows all explicit submission and academic integrity rules (e.g.,	**7.5 points**: The submission achieves most core functional requirements but fails one or more critical components (e.g., a major section is missing, a required	**6.5 points**: The submission is missing multiple critical functional components and/or shows a clear violation of a major instruction(s). The submission is	**0 points**: The submission fails to open/run, or it violates a core academic rule. Most functionality is absent.

Criteria	Exceeds (Max Pts)	Meets (Max Pts)	Partially Meets (Max Pts)	Does not meet (Max Pts)	Not evident (Max Pts)
		correct file type, required components present, no plagiarism). Minor, non-critical aspects may be missing or slightly flawed.	function doesn't work, or the core logic/process is flawed).	incomplete or unusable.	