CA3 - Voting Rules

Due 28 Jan by 23:59 **Points** 100 **Submitting** an external tool
Available until 28 Jan at 23:59

This assignment was locked 28 Jan at 23:59.

COMP517 Continuous Assessment Task 3

Assignment Number: 3 of 3

Deadline for submission: **14th January 2022, 23.59**

Submission Mode: Please provide your solutions electronically via Canvas, via the integrated CodeGrade system. **You should submit a single .py file.**

Purpose of the assessment: This assignment will test the ability of the students to work with functions, basic and advanced Python operations, file handling and error handling.

Learning Outcomes Assessed:

LO1. Demonstrate knowledge of fundamental imperative programming concepts such as variables and assignment, conditional statements, loops and methods.

LO2. Be able to design and code applications in a suitable programming language.

LO4. Critical awareness of important principles of software design and development, including appropriate naming of variables and classes, code layout, testing and debugging, and documentation.

Marking Criteria: Based on the marking descriptors of the University's Code of Practice on Assessment

Late submissions: For late submissions the standard penalties apply, according to UoL policy.

Plagiarism: Please provide individual answers and do not coordinate with other students to complete the assignment. Check the official UoL guidelines for plagiarism and academic integrity.
The submitted files will be checked using tools for plagiarism and collusion.

You may use the "Discussions" to ask clarification questions about the assignment, but please ***do not ask any questions that would give away the solutions or even parts of the solutions.***

Continuous Assessment Task 3 - Voting Rules

Background:

In this assignment you will design and implement several voting rules. In a voting setting, we have a set of n agents and a set of m alternatives. Every agent has a *preference ordering* $\succ$ where $\alpha \succ \beta$ means that the agent prefers alternative α to alternative β. A *preference profile* is a set of n preference orderings, one for every agent.

For example, if we have a voting setting with 4 agents and 4 alternatives, one possible preference profile could be the following:

Agent 1: $\alpha \succ \gamma \succ \beta \succ \delta$
Agent 2: $\alpha \succ \beta \succ \delta \succ \gamma$
Agent 3: $\gamma \succ \beta \succ \alpha \succ \delta$
Agent 4: $\beta \succ \alpha \succ \delta \succ \gamma$

A ***voting rule*** is a function that takes as input the preferences of a set of agents and outputs a winning alternative.

Consider the following voting rules:

Voting Rules:

Dictatorship:
An agent is selected, and the winner is the alternative that this agent ranks first. For example, if the preference ordering of the selected agent is $\alpha \succ \gamma \succ \beta \succ \delta$, then the winner is alternative α.

Plurality:
The winner is the alternative that appears the most times in the first position of the agents' preference orderings. ***In the case of a tie, use a tie-breaking rule to select a single winner.***

Veto:
Every agent assigns 0 points to the alternative that they rank in the last place of their preference orderings, and 1 point to every other alternative. The winner is the alternative with the most number of points. ***In the case of a tie, use a tie-breaking rule to select a single winner.***

Borda:
Every agent assigns a score of 0 to the their least-preferred alternative (the one at the bottom of the preference ranking), a score of 1 to the second least-preferred alternative, ... , and a score of $m-1$ to their favourite alternative. In other words, the alternative ranked at position j receives a score of $m-j$. The winner is the alternative with the highest score. ***In the case of a tie, use a tie-breaking rule to select a single winner.***

Harmonic:
Every agent assigns a score of $\frac{1}{m}$ to the their least-preferred alternative (the one at the bottom of the preference ranking), a score of $\frac{1}{m-1}$ to the second least-preferred alternative, ... , and a score of 1 to their favourite alternative. In other words, the alternative ranked at position j receives a score of $\frac{1}{j}$. ***The winner***

is the alternative with the highest score. In the case of a tie, use a tie-breaking rule to select a single winner.

Single Transferable Vote (STV):
The voting rule works in rounds. In each round, the alternatives that appear the least frequently in the first position of agents' rankings are removed, and the process is repeated. When the final set of alternatives is removed (one or possibly more), then this last set is the set of possible winners. ***If there are more than one**, **a tie-breaking rule is used to select a single winner.***

Example:

Consider the preference profile of the example above. In the first round alternative (\delta is removed\) and we get the following new preference profile:

Agent 1: $\alpha \succ \gamma \succ \beta$
Agent 2: $\alpha \succ \beta \succ \gamma$
Agent 3: $\gamma \succ \beta \succ \alpha$
Agent 4: $\beta \succ \alpha \succ \gamma$

In the second round, both γ and β are removed. In the third round, α is removed, and α is the winner.

Tie-Breaking Rules:

We will consider the following three tie-breaking rules. Here, we assume that the alternatives are represented by integers.

- ***max***: Among the possible winning alternatives, select the one with the highest number.
- ***min***: Among the possible winning alternatives, select the one with the lowest number.
- ***agent i***: Among the possible winning alternatives, select the one that agent i ranks the highest in his/her preference ordering.

Objective:

Task 1: Implement a function called **generatePreferences(values)** which inputs a set of numerical values that the agents have for the different alternatives and outputs a preference profile.
In particular:

- The input **values** to the **generatePreferences** function is a worksheet corresponding to an xlsx file.

  Note: By worksheet, it is meant what we have seen as a worksheet in class, meaning an openpyxl worksheet. Note that the input to the function should not be an xlsx file or a workbook.

  The rows of the file correspond to agents and the columns correspond to alternatives. The value of a cell i, j is a numerical value that signifies how happy the agent would be if that alternative were to be selected. We refer to that value as the *valuation* of the agent. An example of such a worksheet is the

following:

0.296178	0.434362	0.033033	0.758968
0.559323	0.455792	0.770423	0.770423
0.59096	0.51958	0.731606	0.767473
0.555108	0.344285	0.543484	0.396021
0.836279	0.950928	0.871996	0.852851
0.793427	1.509148	0.700621	0.659306

Here for example, the value of agent 5 for alternative 2 is 0.950928. **You can also access the same table as an .xlsx file here. (https://canvas.liverpool.ac.uk/courses/46744/files/6626076/download?download_frd=1)**

- The output (the return) of the **generatePreferences** function is a dictionary where the keys $\{1, 2, \ldots, n\}$ are the agents and the values are ***lists*** that correspond to the preference orderings of those agents. In particular, a value is a list of numbers [x, y , z, ...] denoting that the agent prefers x to y, y to z and so on. These lists have to be ***consistent*** with the valuations that are given as input in the following sense:
  - If the valuation of an agent for alternative j is larger than the valuation for alternative k, then j should appear before k in the list.
  - If the valuation of an agent for alternative j is the same as the valuation for alternative k, then j should appear before k in the list if $j > k$, and otherwise k should appear before j in the list. In other words, in case of ties in the numerical values, alternatives with larger indices are considered to be more preferred by the agent.
  - For the table above, the corresponding dictionary would then be:
    {
    1: [4, 2, 1, 3],
    2: [4, 3, 1, 2],
    3: [4, 3, 1, 2],
    4: [1, 3, 4, 2],
    5: [2, 3, 4, 1],
    6: [2, 1, 3, 4]
    }

Task 2: Implement the following functions:

dictatorship(preferenceProfile, agent) -> int

The function should input a preference profile represented by a dictionary as described above and an integer corresponding to an agent. The return of the function should be the winner according to the

Dictatorship rule described above. The function should include error handling in case the inputted integer does not correspond to an agent (but not necessarily if it is not an integer).

scoringRule(preferences, scoreVector, tieBreak): -> int

The function should input

- a preference profile represented by a dictionary as described above.
- a score vector of length m, i.e., equal to the number of alternatives, i.e., a list of length m containing positive floating numbers.
- an option for the tie-breaking among possible winners (see section "Tie-Breaking" below).

For every agent, the function assigns the highest score in the scoring vector to the most preferred alternative of the agent, the second highest score to the second most preferred alternative of the agent and so on, and the lowest score to the least preferred alternative of the agent. In the end, it returns the alternative with the highest total score, using the tie-breaking option to distinguish between alternatives with the same score.

The function should contain error-handling code for the case when the length of the scoring vector is not m. In that case, the message "Incorrect input" should be printed and the function should return False.

plurality (preferences, tieBreak) -> int

The function should input

- a preference profile represented by a dictionary as described above.
- an option for the tie-breaking among possible winners (see section "Tie-Breaking" below).

The function should return the winner of the Plurality rule as described above, using the tie-breaking option to distinguish between possible winners.

veto (preferences, tieBreak) -> int

The function should input

- a preference profile represented by a dictionary as described above.
- an option for the tie-breaking among possible winners (see section "Tie-Breaking" below).

The function should return the winner of the Veto rule as described above, using the tie-breaking option to distinguish between possible winners.

borda (preferences, tieBreak) -> int

The function should input

- a preference profile represented by a dictionary as described above.
- an option for the tie-breaking among possible winners (see section "Tie-Breaking" below).

The function should return the winner of the Borda rule as described above, using the tie-breaking option to distinguish between possible winners.

harmonic (preferences, tieBreak) -> int

The function should input

- a preference profile represented by a dictionary as described above.

- an option for the tie-breaking among possible winners (see section "Tie-Breaking" below).

The function should return the winner of the Harmonic rule as described above, using the tie-breaking option to distinguish between possible winners.

STV (preferences, tieBreak) -> int

The function should input

- a preference profile represented by a dictionary as described above.
- an option for the tie-breaking among possible winners (see section "Tie-Breaking" below).

The function should return the winner of the Single Transferable Vote rule as described above, using the tie-breaking option to distinguish between possible winners.

rangeVoting (values, tieBreak) -> int

The function should input

- a worksheet corresponding to an xlsx file, see **Task 1** for the details.
- an option for the tie-breaking among possible winners (see section "Tie-Breaking" below).

The function should return the alternative that has the maximum sum of valuations, i.e., the maximum sum of numerical values in the xlsx file, using the tie-breaking option to distinguish between possible winners.

Tie-breaking: For the ***tieBreak*** input of these functions, the possible options are the following:

- ***tieBreak = "max".*** In that case the max tie-breaking rule should be applied (see above).
- ***tieBreak = "min".*** In that case the min tie-breaking rule should be applied (see above).
- ***tieBreak* is an integer between 1 and n:** In that case the *agent* i rule should be applied, with i being the inputted integer (see above). The function should include error handling in case the inputted integer does not correspond to an agent.

Your mark will be determined based on the following:

Correctness against Brief and Robustness: 50% - This will be evaluated by the auto-tests that will be running on the CodeGrade system. Please make sure to interact with the system, use the feedback and update your code if it does not pass the visible tests. Note also that some tests will be invisible upon submission, but will become visible after the marking has taken place.

Design Choices: 40%

Readability and layout: 10%