COMP202 Assignment 3
Due: Nov 14 at 23:59
This is an individual assignment.

Voting Systems

We had a federal election earlier this month! You may have noticed that the party which won (the Liberal Party) was not the same as the party which won the popular vote (the Conservative Party).

This is an artifact of the voting system we use in Canada and is not uncommon. It is one of many reasons many Canadians want electoral reform.

A *voting system* is an algorithm to pick a winner from a set of votes. Different voting systems can produce different results. The study of voting systems is called *voting theory*, and is important not just for politics — in Artificial Intelligence, computational agents need to be able to make decisions between multiple options. A lot of voting theory research is done by computer scientists.

In this assignment, we implement six voting systems, and compare how the results of the recent election could be different depending on the system.

Instructions

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

Up to 30% can be removed for bad indentation of your code as well as omitting comments, or poor coding structure.

To get full marks, you must:

- Follow all directions below
  - In particular, make sure that all function and variable names are **spelled exactly** as described in this document. Else a 50% penalty will be applied.
- Make sure that your code runs.
  - Code with errors will receive a very low mark.
- Write your name and student ID as a comment in all .py files you hand in
- Name your variables and helper functions appropriately
  - The purpose of each variable should be obvious from the name
- Comment your work
  - A comment every line is not needed, but there should be enough comments to fully understand your program

Errata and Frequently Asked Questions

On MyCourses we have a discussion forum titled Assignment 3. **A thread will be pinned in the forum with any errata and frequently asked questions. If you are stuck on the assignment, start by checking that thread.**

We strongly encourage starting the assignment early.

What To Submit

Please put all your files in a folder called Assignment3. Zip the folder (DO NOT RAR it) and submit it in MyCourses. If you do not know how to zip files, please ask any search engine or friends. Google will be your best friend with this, and a lot of different little problems as well.

Inside your zipped folder, there must be the following files. **Do not submit any other files.** Any deviation from these requirements may lead to lost marks.

1. `a3_helpers.py`
2. `single_winner.py`
3. `instant_run_off.py`
4. `proportional_representation.py`
5. `HOC.png`
6. `README.txt` In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your program, it may lead the TA to give you more partial credit.

   This file is also where you should make note of anybody you talked to about the assignment. Remember this is an individual assignment, but you can talk to other students using the **Gilligan's Island Rule**: you can't take any notes/writing/code out of the discussion, and afterwards you must do something inane like watch television for at least 30 minutes.

   If you didn't talk to anybody nor have anything you want to tell the TA, just say "nothing to report" in the file.

Style

There are 70 marks for completing functions in this assignment.
There are 30 marks for the style of your code.

Some tips:

- Call helper functions rather than copy and paste (or reinvent) code
- Use descriptive variable names
- Lines of code should NOT require the TA to scroll horizontally to read the whole thing

Writing Your Own Docstrings

This is the first assignment this term where we do not give you docstrings for your functions.
It is important for you to practice writing tests on your own and learning how to identify important test cases on your own.

Randomness!

This is also the first assignment where randomness will affect the results. For many functions, you'll find your results could change each time you run your module. **Don't panic!**

One source of randomness comes from dictionaries. The order of the keys is not something that Python fixes. So if you have a doctest where the output is a dictionary, you could sometimes "fail" the doctest with correct code, because the keys were printed in a different order than you expected.

Some tricks for writing doctests that will consistently pass even though the output can be affected by randomness:

- We give you a function in `helpers.py` that will print a dictionary sorted by the keys.
- You can have doctest test properties of a dictionary, such as:

```
>>> dict = {'a': 4, 'b': 2, 'd': 1}
>>> len(dict)
3
>>> dict['a']
4
>>> dict['b']
2
>>> sum(dict.values())
7
>>> sorted(dict.values())
[1, 2, 4]
```

When we mark your code

We will be using a different system than doctest to test your code. It will not be affected by how a dictionary is printed out.

If your code is correct, but doesn't always pass your own doctests, it should pass our tests.

Background

I've put together a YouTube playlist of some videos about how the Canadian voting system works, and how all the other voting systems in this assignment work: `https://tinyurl.com/a3-youtube`

One unavoidable source of confusion is that each voting system goes by many different names. Some names are more common in different regions. This is also true of other voting terminology. For example, in Canada we usually say "riding" instead of saying "constituency" or "district". The YouTube videos sometimes use different names than what I typically use — for example "first past the post" instead of plurality.

1 Helper Functions [9 points]

Download the file `a3_helpers.py`. Add your name and student ID to the top.

We provide:

- `pr_dict`: given a dictionary, sort it by key and then print it. This function will be useful for testing your code since it prints dictionaries in a consistent way.

Complete these functions:

1.1 Flatten lists [1 points]

Complete and test the function `flatten_lists`:

- Input: a list that can contain lists (assume the nested lists won't also contain lists)
- Replaces any lists inside this list with their values.
- Return the new version of this list, and do not modify the input list.
- Example:

```
>>> flatten_lists([[0], [1, 2], 3])
[0, 1, 2, 3]
```

1.2 Flatten dicts [1 points]

Complete and test the function `flatten_dict`:

- Input: a dictionary where all keys are non-negative integers
- Make and return a list from this dictionary.
- The list contains: the keys of the dictionary, repeated v many times, where v is the value associated with the key in the dictionary.
- The dictionary should not be modified.
- Example:

```
>>> flatten_dict({'LIBERAL': 5, 'NDP':2})
['LIBERAL', 'LIBERAL', 'LIBERAL', 'LIBERAL', 'LIBERAL', 'NDP', 'NDP']
```

1.3 Add Dictionaries [1 points]

Complete and test the function `add_dicts`:

- Input: two dictionaries where all the keys are numbers
- It merges the two dictionaries. If a key is in both dictionaries, add their values.
- Return the resulting dictionary. The two input dictionaries should not be modified.
- Example:

```
>>> add_dicts({'a':5, 'b':2, 'd':-1}, {'a':7, 'b':1, 'c':5})
{'a': 12, 'b': 3, 'd': -1, 'c': 5}
```

1.4 Get All Candidates [2 points]

Complete and test the function `get_all_candidates`:

- Input: a list
- This list can be a list of strings, a list of lists, or a list of dictionaries. Or a mix of all three.
- Return: all the unique strings in this list and its nested contents.
- Example:

```
>>> get_all_candidates([{'GREEN':3, 'NDP':5}, {'NDP':2, 'LIBERAL':4},
                        ['CPC', 'NDP'], 'BLOC'])
['GREEN', 'NDP', 'LIBERAL', 'CPC', 'BLOC']
```

1.5 Get Min/Max Candidate [2 points]

Complete and test the function `get_candidate_by_place`:

- Input: a dictionary with the format we have from the last four functions, and a function (assume either `min` or `max`)
- Evaluate the function on the dictionary's values. Return the key of the dictionary corresponding to that value.
- Put another way: if the function is `max`, we return the winner of the election. If the function is `min`, we return the last place candidate. Return the key of the dictionary
- Break ties randomly.
- Example:

```
>>> result = {'LIBERAL':4, 'NDP':6, 'CPC':6, 'GREEN':4}
>>> random.seed(0)
>>> get_candidate_by_place(result, min)
'GREEN'
>>> random.seed(1)
>>> get_candidate_by_place(result, min)
'LIBERAL'
```

1.6 Get Winner [1 point]

Complete and test the function `get_winner`:

- Input: a dictionary with same format as above
- Return the key with the greatest value. If there are ties, break them randomly.
- Example:

```
>>> get_winner({'NDP': 2, 'GREEN': 1, 'LIBERAL': 0, 'BLOC': 0})
'NDP'
```

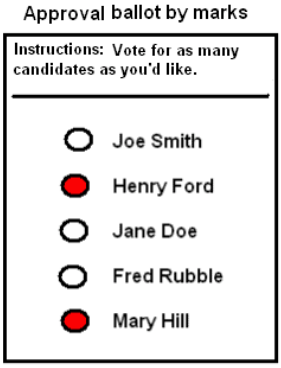
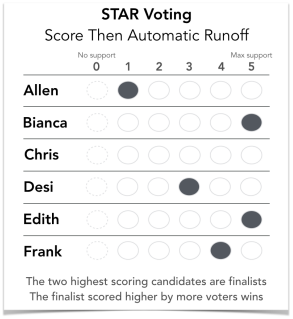
1.7 Get Last Place [1 point]

Complete and test the function `last_place`:

- Input: a dictionary with same format as above
- Return the key with the *lowest* value. If there are ties, break them randomly.

2 Single-Winner Counting-Only Elections [20 points]

In elections, voters cast their votes on *ballots*. There are four major types of ballots (see table). We'll start with voting systems which simply give points for each candidate for each ballot, and then we declare the winner to be the one with the most points.

Plurality	**Approval**	**Rated**	**Ranked**
			
Vote for only one candidate	Vote for as many candidates as you like	Rate each candidate out of a common scale (e.g. out of 5)	Rank each candidate (1st choice, 2nd choice, etc)
Each ballot is represented as a string, for example: `'John Citizen'`	Each ballot is a list where order doesn't matter, e.g. `['Henry Ford', 'Mary Hill']`	Each ballot is a dictionary, e.g. `{'Allen':1, 'Bianca':5, 'Chris':3, 'Edith':5, 'Frank':5}`	Each ballot is a list where position indicates rank, e.g. `['John Citizen', 'Mary Hill', 'Jane Doe']`

Now, download `single_winner.py` and put your name and student ID at the top.

2.1 Count Plurality Ballots [5 points]

Complete and test the function `count_plurality`:

- Input: a list of plurality ballots
- Return: a dictionary of how many votes each candidate got
- Example:

```
>>> count_plurality(['LIBERAL', 'LIBERAL', 'NDP', 'LIBERAL'])
{'LIBERAL': 3, 'NDP': 1}
```

This is the voting system used in Canada to elect our members of parliament. The country is divided into 338 ridings (also known as districts, or constituencies). Each riding elects a member of parliament using plurality.

Nearly all candidates in Canadian elections belong to a political party. Traditionally, the party with the most members of parliament ("seats") is the party which forms the government. The leader of that party becomes prime minister.

2.2 Count Approval Ballots [5 points]

Complete and test the function `count_approval`:

- Input: a list of approval ballots
- Return: a dictionary of how many votes each candidate got
- Example:

```
>>> count_approval([ ['LIBERAL', 'NDP'], ['NDP'], ['NDP', 'GREEN', 'BLOC']] )
{'LIBERAL': 1, 'NDP': 3, 'GREEN': 1, 'BLOC': 1}
```

Electing a winner this way is called *Approval Voting*. It used in many open source communities for making communal decisions about writing software.

If we ask voters to vote two ways — one with plurality, one with approval — we can get different winners. Plurality rewards candidates who have a lot of supporters who strongly favour them. Over time, this can lead to political polarization. Approval voting instead rewards consensus, so a candidate that many people are fine with is more likely to win because of their broad appeal.

2.3 Count Rated Ballots [5 points]

Complete and test the function `count_rated`:

- Input: a list of rated ballots
- Here, on each ballot, the voter gave each candidate a score out of 5. Sum up the scores for each candidate.
- Return: a dictionary of how many points each candidate got
- Example:

```
>>> count_rated([{'LIBERAL': 5, 'NDP':2}, {'NDP':4, 'GREEN':5}])
{'LIBERAL': 5, 'NDP': 6, 'GREEN': 5}
```

Electing a winner this way is called *Score voting* or *Range voting*. Ratings like these are seen on many websites like IMDb and Amazon. It considers more of the voters' preferences than Approval Voting does.

2.4 Count First Choices of Ranked Ballots [5 points]

Complete and test the function `count_first_choices`:

- Input: a list of ranked ballots
- Return: a dictionary storing, for every party represented in all the ballots, how many ballots for which that party was the first choice
- Example:

```
>>> count_first_choices([['NDP', 'LIBERAL'], ['GREEN', 'NDP'], ['NDP', 'BLOC']])
{'NDP': 2, 'GREEN': 1, 'LIBERAL': 0, 'BLOC': 0}
```

This is basically Plurality but with different ballots. On its own, it is not a distinct voting system. But some voting systems use this as a helper function, such as Instant Run-Off Voting. We'll be implementing IRV in subsection 3.4.

3 Instant Run-Off Vote [20 points]

Download the file `instant_run_off.py` and put your name and student ID at the top.

3.1 Votes Needed to Win [5 points]

Many voting systems use the Droop Quota to determine if a candidate has enough votes to win:

$$\text{RoundDown}\left(\frac{\text{total votes}}{\text{winners}+1}\right)+1$$

So far in this assignment we have been dealing with voting systems where there is only one winner in each riding. (This will change in section 4.)

Note: When *winners* is 1, this means that each candidate needs a 50% + 1 majority to win.

Complete and test the function `votes_needed_to_win`:

- Input: a list of ballots, an integer number of winners
- Return: the integer number of votes a candidate would need to win using the Droop Quota.
- Examples:

```
>>> votes_needed_to_win([{'CPC':3, 'NDP':5}, {'NDP':2, 'CPC':4},
                         {'CPC':3, 'NDP':5}], 1)
2
>>> votes_needed_to_win(['g']*20, 2)
7
```

3.2 Has Votes Needed [5 points]

Complete and test the function `has_votes_needed`:

- Input: a dict of election results (like what `count_approval` returns), and integer votes needed
- Return: a boolean representing whether the candidate with the most votes in this election has at least votes_needed votes. (If there is a tie for which candidate it should not matter which of those candidates you look at.)
- Example:

```
>>> has_votes_needed({'NDP': 4, 'LIBERAL': 3}, 4)
True
```

3.3 Eliminate Candidate [5 points]

Complete and test the function `eliminate_candidate`:

- Input: a list of ranked ballots, a list of candidates to eliminate
- Return: a new list of ranked ballots where all the candidates in `to_eliminate` have been removed
- The input list should not be modified.
- If all candidates on a ballot have been eliminated, it should become an empty list
- Example:

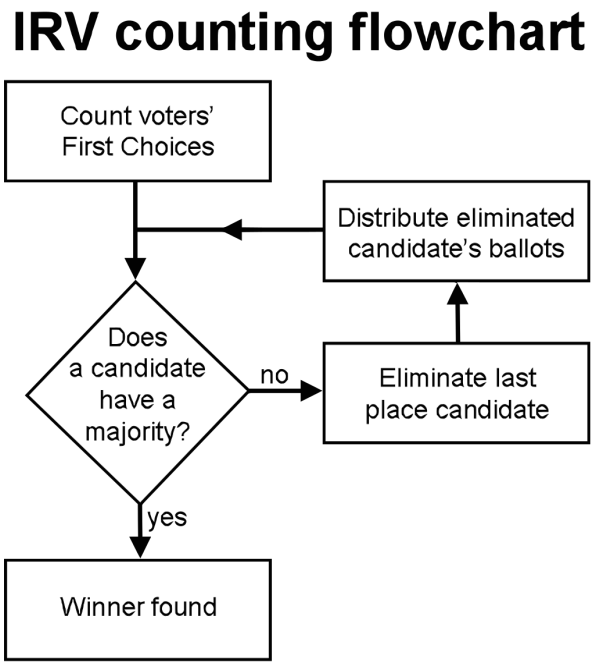
```
>>> eliminate_candidate([['NDP', 'LIBERAL'], ['GREEN', 'NDP'],
                         ['NDP', 'BLOC']], ['NDP', 'LIBERAL'])
[[], ['GREEN'], ['BLOC']]
```

3.4 Count IRV [5 points]

One of the most popular voting systems to use ranked ballots is the *Instant Run-off Vote* (IRV), also known as the *Alternative vote* or *Preferential voting*. This is a single-winner system.

In IRV, we begin by counting up all the voters' first choices. If no party/candidate has a majority, what we do is then identify which candidate came in last. We then go find all the ballots with that candidate as the top choice, and then recount these ballots with the second choice instead.

If we have a majority now, we stop. But if we still don't have a majority, we again eliminate the last place candidate and recount their ballots. The cycle looks like this:



If you would like a visual example, see this YouTube video: `https://www.youtube.com/watch?v=Rgo-eJ-D__s`

People who like IRV often like it because it reduces the pressure on voters to vote strategically. If their first choice doesn't make it, their vote can still have meaning. All of the major political parties in Canada use IRV to select their leaders (or a variant of IRV).

Complete and test the function `count_irv`:

- Input: a list of ranked ballots
- Return: a dictionary of how many votes each candidate ends with after counting with IRV. Every candidate in the input list should appear in the returned dictionary.
- Example:

```
>>> count_irv([['NDP'], ['GREEN', 'NDP', 'BLOC'], ['LIBERAL','NDP'],
               ['LIBERAL'], ['NDP', 'GREEN'], ['BLOC', 'GREEN', 'NDP'],
               ['BLOC', 'CPC'], ['LIBERAL', 'GREEN'], ['NDP']])
{'BLOC': 0, 'CPC': 0, 'GREEN': 0, 'LIBERAL': 3, 'NDP': 5}
```

4 Proportional Representation [17 points]

Download the file `proportional_representation.py`. In this module we explore two common forms of proportional representation: the Single Transferable Vote (STV), and the Sainte-Laguë method. These systems are *multiwinner systems*: they elect multiple winners instead of one.

4.1 Single Transferable Vote

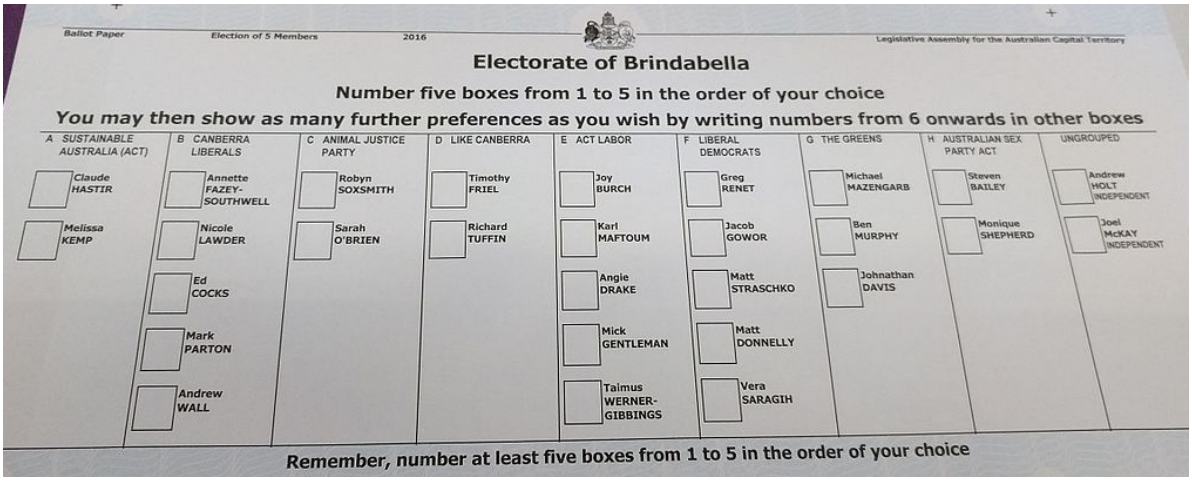
We will start with STV, because it is closely related to IRV. STV is used in many parts of the world such as Australia, India, Ireland, Northern Ireland, and the local elections in Scotland. STV has been proposed for use in Canada, and nearly passed a referendum in British Columbia.

Here is a video explaining STV: `https://www.youtube.com/watch?v=P38Y4VG1Ibo`.
The quota should be familiar to you — we implemented it already!

4.1.1 IRV to STV Ballot [5 points]

Because there are mulitple winners in an STV election, each party will typically run multiple candidates. The ridings are also much bigger. For example, if we have five traditional ridings in one city, and changed to STV, the five ridings would be merged together and together elect five candidates.

Here's what a real-world STV ballot looks like. Each column represents a different party; each box is a different candidate for that party.



Complete and test the function `irv_to_stv_ballot`:

- Input: a list of ranked ballots, and a positive integer `num_winners`
- For every ranked ballot: replace each party with `num_winners` many candidates from that party, numbering them starting from 0 ('GREEN0', 'GREEN1', etc).
- Example:

```
>>> irv_to_stv_ballot([['NDP', 'CPC'], ['GREEN']], 3)
[['NDP0', 'NDP1', 'NDP2', 'CPC0', 'CPC1', 'CPC2'], ['GREEN0', 'GREEN1', 'GREEN2']]
```

- Note: we are assuming that every party would run the maximum number of candidates! This isn't necessarily the case in the real world.

4.1.2 Count STV [5 points]

We have written a function for you that implements STV when given a list of STV ballots: `stv_vote_results`. This function returns how many votes wound up for each candidate after all transfers.

For an election with a quota of 26 and 100 votes and 3 winners, `stv_vote_results` could return something like[1]:

```
{'BLOC0': 26, 'BLOC1': 26, 'GREEN0': 0, 'GREEN1': 0, 'NDP0': 22, 'NDP1': 26}
```

Now: complete and test the function `count_stv`:

- Input: a list of ranked ballots, and an integer `num_winners` (how many winners)
- For the IRV ballots, convert them to STV and then count them using `stv_vote_results`.
- Return: how many candidates from each party won this election.
- This function should call `stv_vote_results`. We do not expect you to implement STV!
- Note: a winning candidate from `stv_vote_results` will have `quota` many votes (and not any more than that)
- If we get the above result when we call `stv_vote_results`, we would expect `count_stv` to return:

```
>>> random.seed(3) # make the random tie-break consistent
>>> g = ['GREEN', 'NDP', 'BLOC']
>>> n = ['NDP', 'GREEN', 'BLOC']
>>> pr_dict(count_stv([g]*5 + [n]*3, 4))
{'BLOC': 0, 'GREEN': 3, 'NDP': 1}
```

[1] It's possible that the result of `stv_vote_results` has a lower number of votes in it than the total votes cast. Those "lost" ballots where ones where all the candidates on the ballot were eliminated.

4.2 Sainte-Laguë Method [10 points]

When people say a country uses "proportional representation", usually they mean either the Sainte-Laguë Method, or the very similar D'Hondt Method[2] . We will implement the former. Sainte-Laguë is also known as Webster's Method.

In the Sainte-Laguë Method, there are no longer any local representatives. Instead, everybody casts a plurality-style ballot for the party they like the most. Then each party gets a number of seats in the parliament that is proportional to the popular vote.

The Sainte-Laguë Method is used in Norway, Sweden, Germany, Denmark, and many other countries. A variant of the method, called Mixed-Member Proportional (MMP) has been proposed for use in Canada. In MMP, people still get to vote for a local representative. Then extra seats are added to the House of Commons so that each party has the proportion of votes they would get under Sainte-Laguë. This is how New Zealand elects its House of Representatives.

How it works (quoting Wikipedia) "After all the votes have been tallied, successive quotients are calculated for each party. The formula for the quotient is:

$$\text{quotient} = \frac{V}{2s+1}$$

where:

- V is the total number of votes that party received, and
- s is the number of seats that have been allocated so far to that party, initially 0 for all parties.

Whichever party has the highest quotient gets the next seat allocated, and their quotient is recalculated. The process is repeated until all seats have been allocated."

The Wikipedia article also has a useful example here: `https://tinyurl.com/sainte-lague`

Now: complete and test the function `count_SL`:

- Input: a list of plurality vote results, and an integer `num_seats` (how many winners)
- Return: how many seats each party won using the Sainte-Laguë Method.
- For the example on the Wikipedia page, we would expect `count_SL` to return:

```
{'A': 3, 'B': 3, 'C': 1, 'D': 1}
```

[2]You may notice the YouTube playlist for the assignment features the D'Hondt Method rather than Sainte-Laguë. The only difference between the two is the divisor used. The algorithm of calculating successive quotients is the same. I found that the video I chose was much clearer than any of the English-language videos I found on Sainte-Laguë specifically.

5 Simulate Elections [1 point]

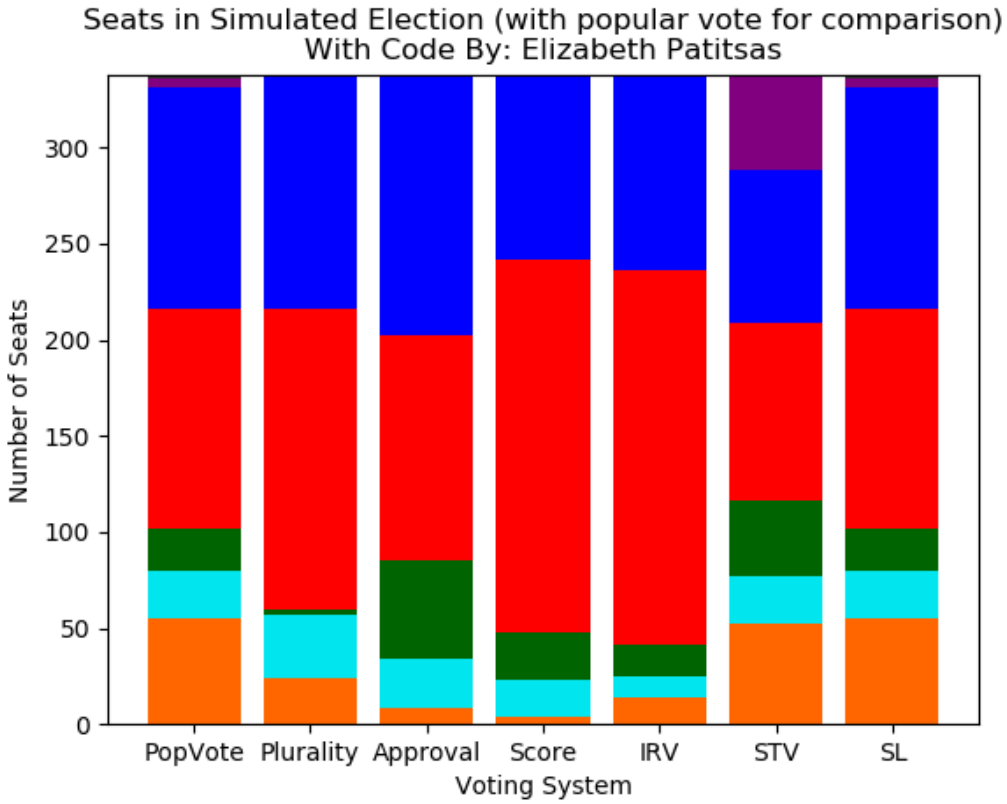
Download the files `simulate_elections.py`, `read_votes.py`, and `votes.txt`.

Open `simulate_elections.py`. At the top, change the value of `NAME` to your name. Do not make any other changes to the code.

You do not need to edit any of the code in the other two files. Together the three files simulate the recent Canadian election under the six voting systems we have seen in this assignment.

When are you done the other parts of the assignment, run the module `simulate_elections.py`. It will count the votes under the six different voting systems and the popular vote, and then make a plot of it saved as `HOC.png`. This could take a few minutes to run!

Have a look at your plot! It will look something like this:



Closing Notes

How the Simulation Works

The file `votes.txt` contains the results of a simulation of how voters in the recent Canadian election would vote under the different voting systems we have explored in this assignment. Each line represents one voter. The second number on each line is their riding's number (Elections Canada has a system for numbering ridings in Canada).

The module `read_votes.py` reads a file like `votes.txt` and stores its contents as a very big dictionary. Then the module `simulate_elections`, for each voting system, runs an election for each riding and keeps track of the results.

Wondering how that `votes.txt` file got created? Several years ago, I (Elizabeth) took a course on artificial intelligence and decision theory. Decision theory is a field which includes voting theory and game theory. My final project in the course was to make a model of electoral preferences in Canada.

To make this assignment I updated my model to reflect the latest election. One thing that made it tricky for me was adding in a new party! The PPC had not been created when I made the original model. I'm still a little dubious about the results we get for STV — it seems like the PPC does a bit too well. No simulation is perfect!

I am pleased by how closely the model parallels the actual results of the 2019 election under Plurality. In coding the solutions for this assignment, I found consistently that each party is at most 1 seat off from the actual totals (yay!) I'm proud of this also because writing an AI model to simulate strategic voting is hard.

Want to make a model of your own? COMP 202 starts you off on the path of CS courses you'll need! You'll also want to learn about probability and linear algebra. McGill offers a similar course to the one I took as a grad student. It's COMP 553: Algorithmic Game Theory. Many other universities offer similar courses.

Another Simulation

I'm not the only one who was curious how the recent election would have resulted under different systems! You can see a different simuation on the CBC website, which only looks at proporiotal representation: `https://ici.radio-canada.ca/info/2019/elections-federales/mode-scrutin-proportionnelle-mixte-compensatoire/index-en.html`. (Their model does not take into account strategic voting, whereas mine did.)

Learning More

If you'd like to learn more about voting theory, a great place to start is the book *Gaming the Vote: Why Elections Aren't Fair (and What We Can Do About It).*