This is CS50

CS50's Introduction to Computer Science

OpenCourseWare

Donate (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

Lab 6: World Cup

You are welcome to collaborate with one or two classmates on this lab, though it is expected that every student in any such group contribute equally to the lab.

Write a program to run simulations of the FIFA World Cup.

```
$ python tournament.py 2018m.csv
Belgium: 20.9% chance of winning
Brazil: 20.3% chance of winning
Portugal: 14.5% chance of winning
Spain: 13.6% chance of winning
Switzerland: 10.5% chance of winning
Argentina: 6.5% chance of winning
England: 3.7% chance of winning
France: 3.3% chance of winning
Denmark: 2.2% chance of winning
Croatia: 2.0% chance of winning
Colombia: 1.8% chance of winning
Sweden: 0.5% chance of winning
Uruguay: 0.1% chance of winning
Mexico: 0.1% chance of winning
```

Background

In soccer's World Cup, the knockout round consists of 16 teams. In each round, each team plays another team and the losing teams are eliminated. When only two teams remain, the winner of the final match is the champion.

In soccer, teams are given FIFA Ratings (https://en.wikipedia.org/wiki/FIFA_World_Rankings#Current_calculation_method), which are numerical values representing each team's relative skill level. Higher FIFA ratings indicate better previous game results, and given two teams' FIFA ratings, it's possible to estimate the probability that either team wins a game based on their current ratings. The FIFA Ratings from two previous World Cups are available as the May 2018 Men's FIFA Ratings (https://www.fifa.com/fifa-world-ranking/ranking-table/men/rank/id12189/) and March 2019 Women's FIFA Ratings (https://www.fifa.com/fifa-world-ranking/ranking-table/women/rank/ranking_20190329/).

Using this information, we can simulate the entire tournament by repeatedly simulating rounds until we're left with just one team. And if we want to estimate how likely it is that any given team wins the tournament, we might simulate the tournament many times (e.g. 1000 simulations) and count how many times each team wins a simulated tournament. 1000 simulations might seem like many, but with today's computing power we can accomplish those simulations in a matter of milliseconds. At the end of this lab, we'll experiment with how worthwhile it might be to increase the number of simulations we run, given the trade-off of runtime.

Your task in this lab is to do just that using Python!

Getting Started

Open VS Code (https://cs50.dev/).

Start by clicking inside your terminal window, then execute `cd` by itself. You should find that its "prompt" resembles the below.

```
$
```

Click inside of that terminal window and then execute

```
wget https://cdn.cs50.net/2022/fall/labs/6/world-cup.zip
```

followed by Enter in order to download a ZIP called `world-cup.zip` in your codespace. Take care not to overlook the space between `wget` and the following URL, or any other character for that matter!

Now execute

```
unzip world-cup.zip
```

to create a folder called `world-cup`. You no longer need the ZIP file, so you can execute

```
rm world-cup.zip
```

and respond with "y" followed by Enter at the prompt to remove the ZIP file you downloaded.

Now type

```
cd world-cup
```

followed by Enter to move yourself into (i.e., open) that directory. Your prompt should now resemble the below.

```
world-cup/ $
```

If all was successful, you should execute

```
ls
```

and you should see the following files:

```
answers.txt  2018m.csv  2019w.csv  tournament.py
```

If you run into any trouble, follow these same steps again and see if you can determine where you went wrong!

Understanding

Start by taking a look at the `2018m.csv` file. This file contains the 16 teams in the knockout round of the 2018 Men's World Cup and the ratings for each team. Notice that the CSV file has two columns, one called `team` (representing the team's country name) and one called `rating` (representing the team's rating).

The order in which the teams are listed determines which teams will play each other in each round (in the first round, for example, Uruguay will play Portugal and France will play Argentina; in the next round, the winner of the Uruguay-Portugal match will play the winner of the France-Argentina match). So be sure not to edit the order in which teams appear in this file!

Ultimately, in Python, we can represent each team as a dictionary that contains two values: the team name and the rating. Uruguay, for example, we would want to represent in Python as `{"team": "Uruguay", "rating": 976}`.

Next, take a look at `2019w.csv`, which contains data formatted the same way for the 2019 Women's World Cup.

Now, open `tournament.py` and see that we've already written some code for you. The variable `N` at the top represents how many World Cup simulations to run: in this case, 1000.

The `simulate_game` function accepts two teams as inputs (recall that each team is a dictionary containing the team name and the team's rating), and simulates a game between them. If the first team wins, the function returns `True`; otherwise, the function returns `False`.

The `simulate_round` function accepts a list of teams (in a variable called `teams`) as input, and simulates games between each pair of teams. The function then returns a list of all of the teams that won the round.

In the `main` function, notice that we first ensure that `len(sys.argv)` (the number of command-line arguments) is 2. We'll use command-line arguments to tell Python which team CSV file to use to run the tournament simulation. We've then defined a list called `teams` (which will eventually be a list of teams) and a dictionary called `counts` (which will associate team names with the number of times that team won a simulated tournament). Right now they're both empty, so populating them is left up to you!

Finally, at the end of `main`, we sort the teams in descending order of how many times they won simulations (according to `counts`) and print the estimated probability that each team wins the World Cup.

Populating `teams` and `counts` and writing the `simulate_tournament` function are left up to you!

Implementation Details

Complete the implementation of `tournament.py`, such that it simulates a number of tournaments and outputs each team's probability of winning.

First, in `main`, read the team data from the CSV file into your program's memory, and add each team to the list `teams`.

- The file to use will be provided as a command-line argument. You can access the name of the file, then, with `sys.argv[1]`.
- Recall that you can open a file with `open(filename)`, where `filename` is a variable storing the name of the file.
- Once you have a file `f`, you can use `csv.DictReader(f)` to give you a "reader": an object in Python that you can loop over to read the file one row at a time, treating each row as a dictionary.
- By default, all values read from the file will be strings. So be sure to first convert the team's `rating` to an `int` (you can use the `int` function in Python to do this).

- Ultimately, append each team's dictionary to `teams`. The function call `teams.append(x)` will append `x` to the list `teams`.
- Recall that each team should be a dictionary with a `team` name and a `rating`.

Next, implement the `simulate_tournament` function. This function should accept as input a list of teams and should repeatedly simulate rounds until you're left with one team. The function should then return the name of that team.

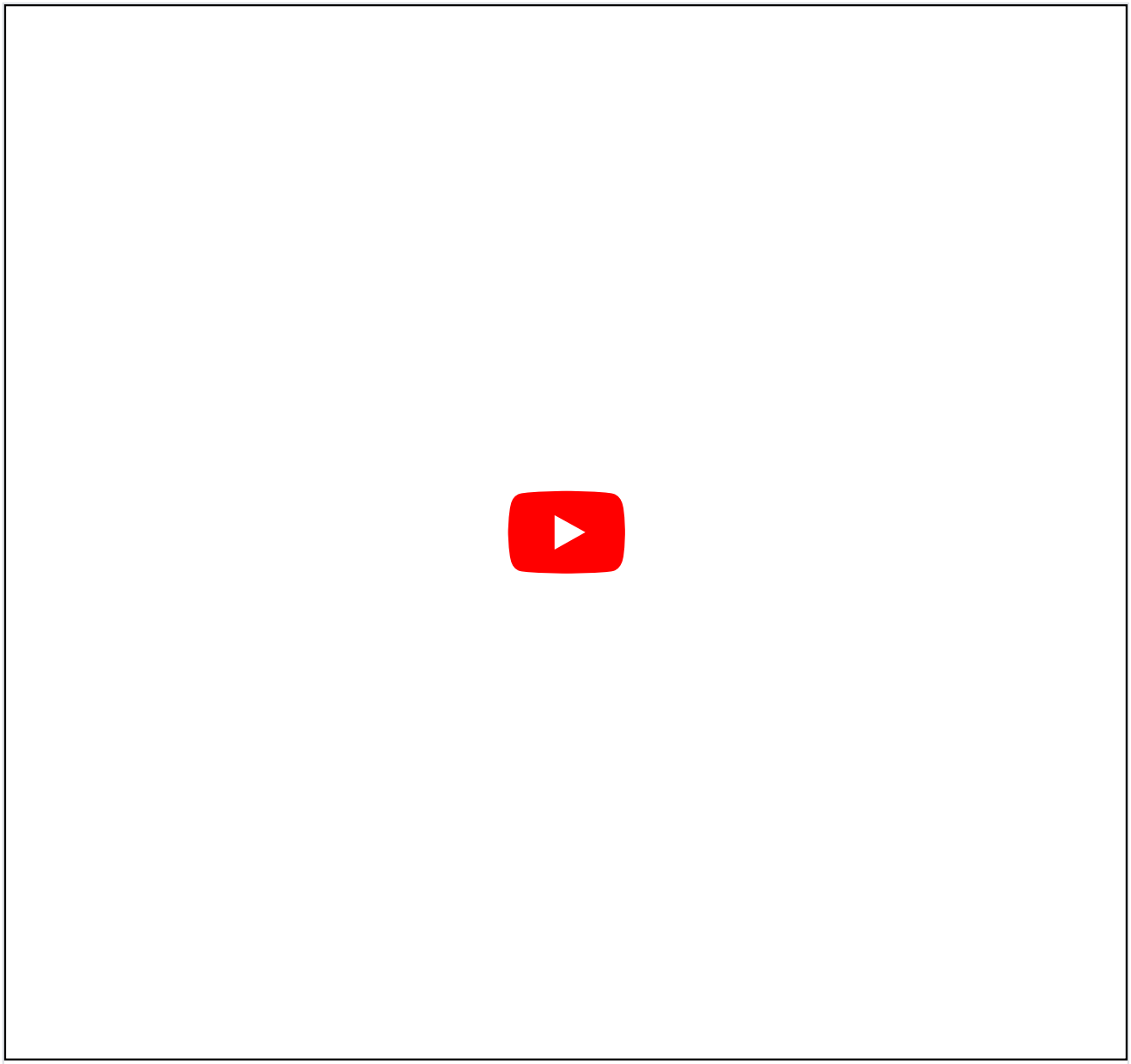
- You can call the `simulate_round` function, which simulates a single round, accepting a list of teams as input and returning a list of all of the winners.
- Recall that if `x` is a list, you can use `len(x)` to determine the length of the list.
- You should not assume the number of teams in the tournament, but you may assume it will be a power of 2.

Finally, back in the `main` function, run `N` tournament simulations, and keep track of how many times each team wins in the `counts` dictionary.

- For example, if Uruguay won 2 tournaments and Portugal won 3 tournaments, then your `counts` dictionary should be `{"Uruguay": 2, "Portugal": 3}`.
- You should use your `simulate_tournament` to simulate each tournament and determine the winner.
- Recall that if `counts` is a dictionary, then syntax like `counts[team_name] = x` will associate the key stored in `team_name` with the value stored in `x`.
- You can use the `in` keyword in Python to check if a dictionary has a particular key already. For example, `if "Portugal" in counts:` will check to see if `"Portugal"` already has an existing value in the `counts` dictionary.

Walkthrough

> This video was recorded when the course was still using CS50 IDE for writing code. Though the interface may look different from your codespace, the behavior of the two environments should be largely similar!



Hints

- When reading in the file, you may find this syntax helpful, with `filename` as the name of your file and `file` as a variable.

```
with open(filename) as file:
    reader = csv.DictReader(file)
```

- In Python, to append to the end of a list, use the `.append()` function.

▶ **Not sure how to solve?**

Testing

Your program should behave per the examples below. Since simulations have randomness within each, your output will likely not perfectly match the examples below.

```
$ python tournament.py 2018m.csv
Belgium: 20.9% chance of winning
Brazil: 20.3% chance of winning
Portugal: 14.5% chance of winning
Spain: 13.6% chance of winning
Switzerland: 10.5% chance of winning
Argentina: 6.5% chance of winning
England: 3.7% chance of winning
France: 3.3% chance of winning
Denmark: 2.2% chance of winning
Croatia: 2.0% chance of winning
Colombia: 1.8% chance of winning
Sweden: 0.5% chance of winning
Uruguay: 0.1% chance of winning
Mexico: 0.1% chance of winning
```

```
$ python tournament.py 2019w.csv
Germany: 17.1% chance of winning
United States: 14.8% chance of winning
England: 14.0% chance of winning
France: 9.2% chance of winning
Canada: 8.5% chance of winning
Japan: 7.1% chance of winning
Australia: 6.8% chance of winning
Netherlands: 5.4% chance of winning
Sweden: 3.9% chance of winning
Italy: 3.0% chance of winning
Norway: 2.9% chance of winning
Brazil: 2.9% chance of winning
Spain: 2.2% chance of winning
China PR: 2.1% chance of winning
Nigeria: 0.1% chance of winning
```

- You might be wondering what actually happened at the 2018 and 2019 World Cups! For Men's, France won, defeating Croatia in the final. Belgium defeated England for the third place position. For Women's, the United States won, defeating the Netherlands in the final. England defeated Sweden for the third place position.

Number of Simulations

Once you're sure your code is correct, let's tinker with the value of `N`, the constant at the top of our file, to adjust the number of times we simulate the tournament. More tournament simulations will give us more accurate predictions (why?), at the cost of time.

We can time programs by prepending their execution at the command-line with `time`. For example, with `N` set to 1000 (the default) execute

```
time python tournament.py 2018m.csv
```

or

```
time python tournament.py 2019w.csv
```

which should output something like

```
real    0m0.037s
user    0m0.028s
sys     0m0.008s
```

though your own times might vary.

Pay attention to the **real** metric, which is the time, in total, it took `tournament.py` to run. And notice that you're given time in minutes and seconds, with accuracy to thousandths of a second.

In `answers.txt`, keep track of how long it takes `tournament.py` to simulate...

- 10 (ten) tournaments
- 100 (one hundred) tournaments
- 1000 (one thousand) tournaments
- 10000 (ten thousand) tournaments
- 100000 (one hundred thousand) tournaments
- 1000000 (one million) tournaments

Each time you adjust `N`, record the **real** time in the appropriate TODO in `answers.txt` by using the same `0m0.000s` format. After timing each scenario, answer the two follow-up questions by overwriting the given TODO:

- Which predictions, if any, proved incorrect as you increased the number of simulations?
- Suppose you're charged a fee for each second of compute time your program uses. After how many simulations would you call the predictions "good enough"?

▶ **See a correctly formatted `answers.txt`**

How to Test Your Code

Execute the below to evaluate the correctness of your code using `check50`. But be sure to compile and test it yourself as well!

```
check50 cs50/labs/2023/x/worldcup
```

Execute the below to evaluate the style of your code using `style50`.

```
style50 tournament.py
```

How to Submit

In your terminal, execute the below to submit your work.

```
submit50 cs50/labs/2023/x/worldcup
```