mt1_fa_2021

September 29, 2024

1 Midterm 1, Fall 2021: Chess Ratings

Version 1.0

Solution

Change Log: 1.0 - Initial Release

This problem builds on your knowledge of **Python data structures, string processing, and implementing mathematical functions**.

For other preliminaries and pointers, refer back to the Piazza post titled **"Midterm 1 Release Notes"**. - Total Exercises: **8**
- Total Points: **16** - Time Limit: **3 Hours**

Each exercise builds logically on the previous one, but you may **solve them in any order**. That is, if you can't solve an exercise, you can still move on and try the next one. **However, if you see a code cell introduced by the phrase, "Sample result for ...", please run it.** Some demo cells in the notebook may depend on these precomputed results.

The point values of individual exercises are as follows:

- Exercise 0: 3 points
- Exercise 1: 2 points
- Exercise 2: 1 points
- Exercise 3: 2 points
- Exercise 4: 1 points
- Exercise 5: 3 points
- Exercise 6: 2 points
- Exercise 7: 2 points

Good luck!

1.1 Elo Ratings

The Elo (rhymes with "Hello") rating system is a widely used method for quantifying relative skill levels of players in a game or sport. The method was originally used to rate chess players and is named for its creator, Arpad Elo. This system is very simple but is able to rate players much more effectively than a win/loss record.

On a high level, the winning player in a game takes rating points away from the losing player. How many points change hands is determined by the difference in the initial ratings of each player. For example, if a highly rated player records a victory over a lower rated player, then they would gain only a few points. This is reflective of the highly rated player being expected to win. However, if the lower rated player is able to pull off an upset, a larger quantity of points

would be exchanged. The idea is that over time the system will adjust players' ratings to their true relative skill levels. Additionally, the difference in Elo ratings between two players can be used to calculate the expectation for the number of wins each player would accrue, which is often expressed as "win probability".

Here we will extract data from a recent chess tournament that captures players' ratings at the start of the tournament and the outcome of all games played. We will then use that data to calculate expected wins based on the matchups and compare our expectation with the observed results. Finally we will determine the updated Elo ratings for the players. There are many variations on this system, but here we will use the original version. You can find more information about the Elo rating system here

Let's get started by taking a look at the data!

```
In [1]: ###
        ### AUTOGRADER TEST - DO NOT REMOVE
        ###

        import run_tests as test_utils
        raw_data = test_utils.read_raw_data('Bucharest2021.pgn')
        test_utils.get_mem_usage_str()
```

```
Out[1]: '48.3 MiB'
```

Take note of how the data is **split** into sections by **blank lines** (`'\n\n'`); this fact might be useful later on! *(hint! hint!)* Here are the first 4 sections.

```
In [2]: demo_raw_data = '\n\n'.join(raw_data.split('\n\n')[:4])
        print(demo_raw_data)
```

```
[Event "Superbet Classic 2021"]
[Site "Bucharest ROU"]
[Date "2021.06.05"]
[Round "1.5"]
[White "Deac,Bogdan-Daniel"]
[Black "Giri,A"]
[Result "1/2-1/2"]
[WhiteElo "2627"]
[BlackElo "2780"]
[ECO "D43"]

1.d4 d5 2.c4 c6 3.Nc3 Nf6 4.Nf3 e6 5.Bg5 h6 6.Bh4 dxc4 7.e4 g5 8.Bg3 b5 9.Be2 Bb7
10.Qc2 Nh5 11.Rd1 Nxg3 12.hxg3 Na6 13.a3 Bg7 14.e5 Qe7 15.Ne4 O-O-O 16.Nd6+ Rxd6
17.exd6 Qxd6 18.O-O g4 19.Ne5 Bxe5 20.dxe5 Qxe5 21.Bxg4 h5 22.Rfe1 Qf6 23.Bf3 h4
24.b3 cxb3 25.Qxb3 hxg3 26.fxg3 Qg7 27.Qd3 Nc7 28.Qd6 c5 29.Qd7+ Kb8 30.Bxb7 Kxb7
31.Rxe6 Qxg3 32.Qc6+ Kb8 33.Qd6 Qxd6 34.Rexd6 Kb7 35.Rf6 Rh7 36.Rd7 b4 37.axb4 cxb4
38.Kf2 a5 39.Ke2 Rg7 40.Rfxf7 Rxg2+ 41.Kd1 Rg1+ 42.Kc2 Rg2+ 43.Kb1 Rg1+ 44.Kb2 Rg2+
45.Kb1 Rg1+ 46.Kb2 Rg2+ 47.Kb1 Rg1+  1/2-1/2

[Event "Superbet Classic 2021"]
[Site "Bucharest ROU"]
```

```
[Date "2021.06.05"]
[Round "1.4"]
[White "Lupulescu,C"]
[Black "Aronian,L"]
[Result "1/2-1/2"]
[WhiteElo "2656"]
[BlackElo "2781"]
[ECO "E39"]

1.d4 Nf6 2.c4 e6 3.Nc3 Bb4 4.Qc2 c5 5.dxc5 O-O 6.Nf3 Na6 7.g3 Nxc5 8.Bg2 Nce4
9.O-O Nxc3 10.bxc3 Be7 11.e4 d6 12.e5 dxe5 13.Nxe5 Qc7 14.Qe2 Nd7 15.Bf4 Nxe5
16.Bxe5 Bd6 17.Bxd6 Qxd6 18.Qe3 Qc7 19.Rfb1 Qxc4 20.Bxb7 Bxb7 21.Rxb7 h6
22.Rxa7 Rxa7 23.Qxa7 Qxc3 24.Rb1 Qc2 25.Rb8 Qd1+ 26.Kg2 Qd5+ 27.Kg1 Qd1+
28.Kg2 Qd5+ 29.Kg1 Qd1+  1/2-1/2
```

The sections in the raw data alternate between **metadata** and **moves data**. The metadata is information about the game, such as who is playing with what pieces, the ratings of each player, and the results of the game. The moves data contains a record of each chess move executed in the game. Since players' Elo ratings are only affected by the outcomes of the games, we are primarily concerned with the metadata.

1.2 Exercise 0 (3 points)

The first thing we need to do in our analysis is get the data in a more structured form.

Fill out the function `extract_games(raw_data)` in the code cell below with the following requirements:

Given a string read from a text file `raw_data`, extract the following information about each game and store in a **list of dictionaries** `games`. Below are details for what one of these dictionaries should look like: * `games[i]['white_player']` - String - Name of the player assigned the white pieces. * Example from `raw_data`: [White "Deac,Bogdan-Daniel"] * Example value: `'Deac,Bogdan-Daniel'`
* Value type: `str`

- `games[i]['black_player']` - String - Name of the player assigned the black pieces.
- Example from `raw_data`: [Black "Giri,A"]
- Example value: `'Giri,A'`
- Value type: `str`
- `games[i]['white_rating']` - Integer - Pre-tournament rating of the white player.
- Example from `raw_data`: [WhiteElo "2627"]
- Example value: `2627`
- Value type: `int`

- `games[i]['black_rating']` - Integer - Pre-tournament rating of the black player.
- Example from `raw_data`: [BlackElo "2780"]
- Example value: `2780`

- Value type: `int`
- `games[i]['result']` - String - Result of the game.
- Example from `raw_data`: [Result "1/2-1/2"]
- Example value: `'1/2-1/2'`
- Value type: `str`

You may assume that the required metadata is included, that sections are separated by blank lines, and that the sections alternate between metadata and moves data (starting with metadata). Additional metadata tags (beyond the 5 you are tasked with extracting) may be present, but they should be ignored. The ordering of the metadata **may be different** from the example above. Additionally, the moves data sections **may not be formatted** the same way as the example above.

A demo of your function run on the `demo_raw_data` defined above is included in the solution cell. The result should be:

```
[{  'white_player': 'Deac,Bogdan-Daniel',
    'black_player': 'Giri,A',
    'result': '1/2-1/2',
    'white_rating': 2627,
    'black_rating': 2780},
  { 'white_player': 'Lupulescu,C',
    'black_player': 'Aronian,L',
    'result': '1/2-1/2',
    'white_rating': 2656,
    'black_rating': 2781}]
```

To help you get started, consider the following snippet, which converts `demo_raw_data` into a nested list of lists. A similar strategy may be helpful in processing the `raw_data` parameter in the exercise.

```
In [3]: demo_metadata_list = [metadata.splitlines() for metadata in demo_raw_data.split('\n\n')
        print(f'type(demo_metadata_list[0]): {type(demo_metadata_list[0])}') # outer list item
        print(f'type(demo_metadata_list[0][0]): {type(demo_metadata_list[0][0])}') # inner lis
        demo_metadata_list

type(demo_metadata_list[0]): <class 'list'>
type(demo_metadata_list[0][0]): <class 'str'>
```

```
Out[3]: [['[Event "Superbet Classic 2021"]',
          '[Site "Bucharest ROU"]',
          '[Date "2021.06.05"]',
          '[Round "1.5"]',
          '[White "Deac,Bogdan-Daniel"]',
          '[Black "Giri,A"]',
          '[Result "1/2-1/2"]',
          '[WhiteElo "2627"]',
          '[BlackElo "2780"]',
          '[ECO "D43"]'],
         ['[Event "Superbet Classic 2021"]',
          '[Site "Bucharest ROU"]',
          '[Date "2021.06.05"]',
          '[Round "1.4"]',
          '[White "Lupulescu,C"]',
          '[Black "Aronian,L"]',
          '[Result "1/2-1/2"]',
          '[WhiteElo "2656"]',
          '[BlackElo "2781"]',
          '[ECO "E39"]']]
```

```
In [4]: def extract_games(raw_data):
            import re
            ###
            ### YOUR CODE HERE
            ###
            nested_data = [metadata.splitlines() for metadata in raw_data.split('\n\n')[::2]]
            key_map = {'White': 'white_player', 'Black': 'black_player', 'Result': 'result',
                       'BlackElo': 'black_rating', 'WhiteElo': 'white_rating'}
            games = []
            for md in nested_data:
                game = {}
                for row in md:
                    data_key, data_val = re.findall(r'([a-zA-Z]+) "([^"]+)', row)[0]
                    if data_key in key_map:
                        key = key_map[data_key]
                        if data_val.isdigit():
                            val = int(data_val)
                        else:
                            val = data_val
                        game[key] = val
                games.append(game)
            return games

        # Demo
        extract_games(demo_raw_data)
```

```
Out[4]: [{'white_player': 'Deac,Bogdan-Daniel',
          'black_player': 'Giri,A',
```

```
  'result': '1/2-1/2',
  'white_rating': 2627,
  'black_rating': 2780},
 {'white_player': 'Lupulescu,C',
  'black_player': 'Aronian,L',
  'result': '1/2-1/2',
  'white_rating': 2656,
  'black_rating': 2781}]
```

The test cell below runs your function **many times**. Remove or comment out any `print` statements to avoid generating excessive output.

```
In [5]: # `ex0_test`: Test cell
        from run_tests import ex0_test
        for _ in range(100):
            ex0_test(10, 4, extract_games)
        print('Passed!')

        ###
        ### AUTOGRADER TEST - DO NOT REMOVE
        ###
        test_utils.get_mem_usage_str()
```

```
Passed!
```

```
Out[5]: '48.6 MiB'
```

Run the following cell, even if you skipped Exercise 0.

We are loading a pre-computed solution that will be used in the following sections. The first two sections items in the list are displayed.

```
In [6]: # Sample result for ex0
        games_metadata = test_utils.read_pickle('games_metadata')
        print(games_metadata[:2])
        test_utils.get_mem_usage_str()
```

```
[{'white_player': 'Deac,Bogdan-Daniel', 'black_player': 'Giri,A', 'result': '1/2-1/2', 'white_
```

```
Out[6]: '48.6 MiB'
```

1.3 Exercise 1 (2 points)

The next bit of information we will need in our analysis is the outcome of each player's games paired with their opponent.

Fill out the function `extract_player_results(games)` in the code cell below with the following requirements:

Given `games`, a list of dictionaries containing the metadata for each game, create dictionary `player_results` mapping each player's name to a list of the outcomes of that player's games.

Each outcome should include the opponent's name (String) and the number of points that the player received (Float) as the outcome of the game as a Tuple.

The order of tuples in the list associated with each player should be the **same as the order of the matchups in `games`**.

You should interpret the value associated with `'result'` as `"<white player points>-<black player points>"` separated by a dash "-". There are three possible outcomes of a game of chess: White wins (`'1-0'`), black wins (`'0-1'`), or draw (`'1/2-1/2'`).

For example, if the input is:

`[{'white_player': 'Dwight Schrute', 'black_player: 'Jim Halpert', 'result': '1-0'}, {'white_player': 'Stanley Hudson', 'black_player': 'Dwight Schrute', 'result': '1/2-1/2'}]`

Then the output should be:

`{'Dwight Schrute': [('Jim Halpert', 1.0), ('Stanley Hudson', 0.5)], 'Jim Halpert': [('Dwight Schrute', 0.0)], 'Stanley Hudson': [('Dwight Schrute', 0.5)]}`

You can assume that each dictionary in `games` will have the keys `'white_player'`, `'black_player'`, and `'result'` and that the values associated with each of those keys are Strings. There may be duplicated matchups where the same two players are paired in the tournament more than once. These cases should be handled the same as any other game and do not require any special treatment.

```
In [7]: demo_games_metadata = [{'white_player': 'Dwight Schrute', 'black_player': 'Jim Halpert
```

```
In [8]: def extract_player_results(games):
            ### BEGIN SOLUTION
            results = {}
            for game in games:
                if game['white_player'] not in results.keys():
                    results[game['white_player']] = []
                if game['black_player'] not in results.keys():
                    results[game['black_player']] = []
                resultSplit = game['result'].split("-")
                if resultSplit[0] == "1/2":
                    whiteResult = .5
                    blackResult = .5
                else:
                    whiteResult = float(resultSplit[0])
                    blackResult = float(resultSplit[1])
                results[game['white_player']].append((game['black_player'], whiteResult))
                results[game['black_player']].append((game['white_player'], blackResult))
            return results
            ### END SOLUTION

        # Demo
        extract_player_results(demo_games_metadata)
```

```
Out[8]: {'Dwight Schrute': [('Jim Halpert', 1.0), ('Stanley Hudson', 0.5)],
         'Jim Halpert': [('Dwight Schrute', 0.0)],
         'Stanley Hudson': [('Dwight Schrute', 0.5)]}
```

The test cell below runs your function **many times**. Remove or comment out any `print` statements to avoid generating excessive output.

```
In [9]: # `ex1_test`: Test cell
        from run_tests import ex1_test
        for _ in range(100):
            ex1_test(10, 4, extract_player_results)
        print('Passed!')

        ###
        ### AUTOGRADER TEST - DO NOT REMOVE
        ###
        test_utils.get_mem_usage_str()
```

```
Passed!
```

```
Out[9]: '48.7 MiB'
```

Run the following cell, even if you skipped Exercise 1.

We are loading a pre-computed solution that will be used in the following sections. The first two entries are displayed.

```
In [10]: # Sample result for ex1
         player_results = test_utils.read_pickle('player_results')
         {k:v for k, v in list(player_results.items())[:2]}
```

```
Out[10]: {'Deac,Bogdan-Daniel': [('Giri,A', 0.5),
           ('Vachier', 1.0),
           ('Mamedyarov,S', 0.5),
           ('Grischuk,A', 0.0),
           ('So,W', 0.5),
           ('Radjabov,T', 0.5),
           ('Lupulescu,C', 0.5),
           ('Aronian,L', 0.0),
           ('Caruana,F', 0.5)],
          'Giri,A': [('Deac,Bogdan-Daniel', 0.5),
           ('Radjabov,T', 0.5),
           ('Lupulescu,C', 0.0),
           ('Aronian,L', 0.5),
           ('Caruana,F', 0.5),
           ('So,W', 0.5),
           ('Vachier', 1.0),
           ('Grischuk,A', 0.5)]}
```

1.4 Exercise 2 (1 point)

Our next task is to compute the total tournament score for each player.

Fill in the function `calculate_score(player_results)` satisfying the following requirements:

Given a dictionary `player_results` mapping player names to their tournament results (similar to the output of Excercise 1), create a **new** dictionary `player_scores` that maps each player (String) to their total score for the tournament (Float).

For example, given the following input:

`{'Angela Martin': [('Oscar Martinez', 1.0), ('Kevin Malone', 0.5), ('Andy Bernard', 0.0)], 'Michael Scott': [('Pam Halpert', 0.0), ('Toby Flenderson', 0.0), ('Todd Packer', 0.0)]}`

Your function should output:

`{'Angela Martin': 1.5, 'Michael Scott': 0.0}`

(Michael isn't exactly a chess prodigy...)

You can assume that the lists keyed to each String in the input will be of the form (String, Float). You do not need to worry about verifying that all of the games implied by the input are present. If you look closely at the example, you will see that this is **not** the case.

```
In [11]: demo_player_results = {'Angela Martin': [('Oscar Martinez', 1.0), ('Kevin Malone', 0.5
```

```
In [12]: def calculate_score(player_results):
             ###BEGIN SOLUTION
             output = {}
             for person in player_results:
                 opponents = player_results[person]
                 total = 0
                 for opponent in opponents:
                     total += opponent[1]
                     output[person] = total
             return output

         # Demo
         calculate_score(demo_player_results)
```

```
Out[12]: {'Angela Martin': 1.5, 'Michael Scott': 0.0}
```

The test cell below runs your function **many times**. Remove or comment out any `print` statements to avoid generating excessive output.

```
In [13]: # `ex2_test`: Test cell
         from run_tests import ex2_test
         for _ in range(200):
             ex2_test(10, 4, calculate_score)
         print('Passed!')

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###
         test_utils.get_mem_usage_str()
```

```
Passed!
```

```
Out[13]: '48.8 MiB'
```

Run the following cell, even if you skipped Exercise 2.

We are loading a pre-computed solution that will be used in the following sections. The first two entries are displayed.

```
In [14]: # Sample result for ex2
         player_scores = test_utils.read_pickle('player_scores')
         {k:v for k, v in list(player_scores.items())[:2]}
```

```
Out[14]: {'Deac,Bogdan-Daniel': 4.0, 'Giri,A': 4.0}
```

1.5 Exercise 3 (2 points)

Our next task is to extract the Elo rating of each player from the metadata.

Fill in the function `extract_ratings(games)` to satisfy the following requirements:

Given a list of dictionaries, `games`, create a dictionary `player_ratings` that maps each player to their Elo rating before the tournament. You can assume that each dictionary in `games` will have the following keys and value types: `'white_player'`: (String), `'black_player'`: (String), `'white_rating'`: (Integer), and `'black_rating'`: (Integer).

Additionally, if the same player has different ratings in the input, your function should raise a `ValueError`.

For example:

Input : `[{'white_player': 'Jim Halpert', 'black_player': 'Darryl Philbin', 'white_rating': 1600, 'black_rating': 1800}, {'white_player': 'Darryl Philbin', 'black_player': 'Phyllis Vance', 'white_rating': 1800, 'black_rating': 1700}]`

Output : `{'Darryl Philbin': 1800, 'Jim Halpert': 1600, 'Phyllis Vance': 1700}`

Input : `[{'white_player': 'Jim Halpert', 'black_player': 'Darryl Philbin', 'white_rating': 1600, 'black_rating': 1800}, {'white_player': 'Darryl Philbin', 'black_player': 'Phyllis Vance', 'white_rating': 1850, 'black_rating': 1700}]`

Here `'Darryl Philbin'` has two ratings: 1800 in his first game and 1850 in his second. Your function should raise a `ValueError`!

```
In [15]: demo_metadata_good = [{'white_player': 'Jim Halpert', 'black_player': 'Darryl Philbin
         demo_metadata_bad = [{'white_player': 'Jim Halpert', 'black_player': 'Darryl Philbin'
```

```
In [16]: def extract_ratings(games):

             from collections import defaultdict
             rating_sets = defaultdict(set)
             for game in games:
                 w, b = game['white_player'], game['black_player']
                 w_rating, b_rating = game['white_rating'], game['black_rating']
                 rating_sets[w].add(w_rating)
                 rating_sets[b].add(b_rating)
             ratings = {}
             for player, rs in rating_sets.items():
                 if len(rs) != 1:
                     raise ValueError
```

```
            ratings[player] = list(rs)[0]
        return ratings


    # Demo
    try:
        extract_ratings(demo_metadata_bad)
        print('This should raise a ValueError')
    except ValueError:
        print('Correctly raised ValueError')
    extract_ratings(demo_metadata_good)
```

```
Correctly raised ValueError
```

Out[16]:
```
{'Jim Halpert': 1600, 'Darryl Philbin': 1800, 'Phyllis Vance': 1700}
```

In [17]:
```
# `ex3_test`: Test cell
from run_tests import ex3_test
for _ in range(200):
    ex3_test(10, 4, extract_ratings)
print('Passed!')

###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

```
Passed!
```

The test cell below runs your function **many times**. Remove or comment out any `print` statements to avoid generating excessive output.

Run the following cell, even if you skipped Exercise 3.

We are loading a pre-computed solution that will be used in the following sections. The first two entries are displayed.

In [18]:
```
# Sample result for ex3
player_ratings = test_utils.read_pickle('player_ratings')
{k:v for k, v in list(player_ratings.items())[:2]}
```

Out[18]:
```
{'Deac,Bogdan-Daniel': 2627, 'Giri,A': 2780}
```

1.6 Exercise 4 (1 point)

The last task before we begin analysis is to implement some functionality to calculate the expected result of a match based on the Elo ratings of each player.

Fill out the function `expected_match_score(r_player, r_opponent)` to satisfy the following requirements:

Given a player's rating (Integer) and their opponent's rating (Integer), compute the player's expected score in a game against that opponent. The formula for the expected score is:

$$\text{Expected Score} = \frac{1}{1 + 10^d}$$

where

$$d = \frac{r_{\text{opponent}} - r_{\text{player}}}{400}$$

Output the expected score as a Float. **Do not round**.

For example:

`expected_match_score(1900, 1500)` should return about `0.909`
`expected_match_score(1500, 1500)` should return about `0.5`
`expected_match_score(1900, 1700)` should return about `0.76`

```
In [19]: demo_ratings = [(1900, 1500), (1500, 1500), (1900, 1700)]
```

```
In [20]: def expected_match_score(r_player, r_opponent):
             ###
             ### YOUR CODE HERE
             ###
             d = (r_opponent - r_player) / 400 #opponent rating - player rating
             return 1 / (1 + 10 ** d) #formula from above

         # Demo
         for rp, ro in demo_ratings:
             print(f'expected_match_score({rp}, {ro}) = {expected_match_score(rp, ro)}')
```

```
expected_match_score(1900, 1500) = 0.9090909090909091
expected_match_score(1500, 1500) = 0.5
expected_match_score(1900, 1700) = 0.7597469266479578
```

The test cell below runs your function **many times**. Remove or comment out any `print` statements to avoid generating excessive output.

```
In [21]: # `ex4_test`: Test cell
         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###
         from run_tests import ex4_test
         for _ in range(200):
             ex4_test(expected_match_score)
         print('Passed!')

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###
         test_utils.get_mem_usage_str()
```

```
Passed!
```

```
Out[21]: '48.8 MiB'
```

1.7 Aside - Functional Programming

It is often useful to write functions which take other functions as arguments. Inside of your function, the functional argument is called in a consistent way. This allows the caller of your function to customize it's behavior.

Here is an over-engineered arithmetic calculator as an example. These functions define mathematical operations.

```
In [22]: # add
         def a(a, b):
             return a+b
         # subtract
         def s(a, b):
             return a-b
         # multiply
         def m(a, b):
             return a*b
         # divide
         def d(a,b):
             return a/b
```

This function, `calc`, takes the two numbers as an argument and a third argument which determines how they are combined.

```
In [23]: def calc(a, b, opp):
             return opp(a,b)
```

Now we can use any function that takes two arguments, like the 4 defined above to determine the behavior of `calc`.

```
In [24]: calc(3,5,a)

Out[24]: 8

In [25]: calc(3,5,d)

Out[25]: 0.6
```

1.8 Exercise 5 (3 points)

Our next task is to write some functionality to determine each player's expected tournament score.

Fill in the function `expected_tournament_score(player_results, player_ratings, es_func)` to satisfy the following requirements:

Given a dictionary, `player_results`, mapping players to their tournament results as a list of tuples (similar to the output from Exercise 1) and a dictionary, `player_ratings`, mapping players to their Elo ratings, compute the **total** expected score for each player (you only need to compute total expected score for players that are keys in `player_results`). The total expected score is simply the sum of the expected scores for each of that players games. Output the results as a dictionary mapping players (String) to their expected tournament score (Float).

The third argument `es_func` is a function that takes two arguments (the player's rating and opponent's rating respectively) and returns an "expected score". You should use it to compute the expected scores for this exercise. **It might not be the same as the solution to Exercise 4!**

A call to `es_func(1450, 1575)` inside of your function would compute the "expected score" for the 1450-rated player against a 1575-rated player.

For example given:

`player_results = {'Angela Martin': [('Dwight Schrute', 1.0), ('Stanley Hudson', 0.5)], 'Dwight Schrute': [('Angela Martin', 0.0), ('Jim Halpert', 0.5)]}`

`player_ratings = {'Angela Martin': 1600, 'Dwight Schrute': 1750, 'Stanley Hudson': 1800, 'Jim Halpert': 1700}`

`es_func = lambda r_player, r_opponent: float(r_player - r_opponent)`

The output would be:

`{'Angela Martin': -350.0, 'Dwight Schrute': 200.0}`

```
In [26]: demo_player_results = {'Angela Martin': [('Dwight Schrute', 1.0), ('Stanley Hudson',
         demo_player_ratings = {'Angela Martin': 1600, 'Dwight Schrute': 1750, 'Stanley Hudson
         demo_es_func = lambda r_player, r_opponent: float(r_player - r_opponent)
```

```
In [27]: def expected_tournament_score(player_results, player_ratings, es_func):
             ###
             ### YOUR CODE HERE
             ###
             expected = {} #dictionary
             #iterating over the player results
             for player, results in player_results.items():
                 total = 0.0 #total expected points for each player
                 for opponent, pts in results:
                     total += es_func(player_ratings[player], player_ratings[opponent])
                 expected[player] = total
             return expected

         # Demo
         expected_tournament_score(demo_player_results, demo_player_ratings, demo_es_func)
```

```
Out[27]: {'Angela Martin': -350.0, 'Dwight Schrute': 200.0}
```

The test cell below runs your function **many times**. Remove or comment out any `print` statements to avoid generating excessive output.

```
In [28]: # `ex5_test`: Test cell
         from run_tests import ex5_test
         for _ in range(200):
             ex5_test(10, 4, expected_tournament_score)
         print('Passed!')

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###
         test_utils.get_mem_usage_str()
```

```
Passed!
```

```
Out[28]: '49.0 MiB'
```

Run the following cell, even if you skipped Exercise 5.

We are loading a pre-computed solution that will be used in the following sections. The first two entries are displayed.

```
In [29]: # Sample result for ex5
         player_expected_score = test_utils.read_pickle('player_expected_score')
         {k:v for k, v in list(player_expected_score.items())[:2]}
```

```
Out[29]: {'Deac,Bogdan-Daniel': 2.827559638896802, 'Giri,A': 4.389932419673484}
```

1.9 Exercise 6 (2 points)

Fill in the function `compute_final_ratings(player_scores, expected_player_scores, player_ratings)` to meet the following requirements:

Given three dictionaries:

- `player_scores`: mapping players (String) to their observed tournament scores (Float)
- `expected_player_scores`: mapping players (String) to their expected tournament scores (Float)
- `player_ratings`: mapping players (String) to their pre-tournament Elo ratings (Float)

calculate each player's post-tournament Elo ratings using this formula:

$$\text{Rating}_{\text{post}} = \text{Rating}_{\text{pre}} + 10(\text{Score}_{\text{observed}} - \text{Score}_{\text{expected}})$$

Return a dictionary mapping each player (String) to their post-tournament rating **rounded to the nearest integer**.

You can assume that all keys are common between the three input dictionaries.

For example:

`player_scores = {'Jim Halpert': 3.0, 'Dwight Schrute': 4.0, 'Stanley Hudson': 3.0}`

`expected_player_scores = {'Jim Halpert': 2.736, 'Dwight Schrute': 4.67, 'Stanley Hudson': 2.85}`

`player_ratings = {'Jim Halpert': 1500, 'Dwight Schrute': 1575, 'Stanley Hudson': 1452}`

Results: `{'Jim Halpert': 1503, 'Dwight Schrute': 1568, 'Stanley Hudson': 1454}`

```
In [30]: demo_player_scores = {'Jim Halpert': 3.0, 'Dwight Schrute': 4.0, 'Stanley Hudson': 3.0
         demo_expected_player_scores = {'Jim Halpert': 2.736, 'Dwight Schrute': 4.67, 'Stanley
         demo_player_ratings = {'Jim Halpert': 1500, 'Dwight Schrute': 1575, 'Stanley Hudson':
```

```
In [31]: def compute_final_ratings(player_scores, expected_player_scores, player_ratings):
             ###
             ### YOUR CODE HERE
             ###
             final = {}
             for player in player_scores:
                 rating_new = player_ratings[player] + 10 * (player_scores[player] - expected_
                 final[player] = round(rating_new)
             return final


         # Demo
         compute_final_ratings(demo_player_scores, demo_expected_player_scores, demo_player_ra
```

```
Out[31]: {'Jim Halpert': 1503, 'Dwight Schrute': 1568, 'Stanley Hudson': 1454}
```

The test cell below runs your function **many times**. Remove or comment out any `print` statements to avoid generating excessive output.

```
In [32]: # `ex6_test`: Test cell
         from run_tests import ex6_test
         for _ in range(200):
             ex6_test(10, compute_final_ratings)
         print('Passed!')

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###
         test_utils.get_mem_usage_str()
```

```
Passed!
```

```
Out[32]: '49.0 MiB'
```

Run the following cell, even if you skipped Exercise 6.

We are loading a pre-computed solution that will be used in the following sections. The first two entries are displayed.

```
In [33]: # Sample result for ex6
         player_final_ratings = test_utils.read_pickle('player_final_ratings')
         {k:v for k, v in list(player_final_ratings.items())[:2]}
```

```
Out[33]: {'Deac,Bogdan-Daniel': 2639, 'Giri,A': 2776}
```

1.10 Exercise 7 (2 points)

The last task we have is to compute the change in rating. This isn't just an intermediate step in Exercise 6, because we have to handle some special cases as well.

Fill in the function `compute_deltas(old_ratings, new_ratings)` to meet the following requirements:

Given dictionaries `old_ratings` mapping players (String) to their pre-tournament Elo ratings (Integer) and `new_ratings` mapping players (String) to their post-tournament Elo ratings, determine the change in each player's rating. Return your result as a dictionary mapping players (String) to their delta (Integer).

Compute the delta as

$$\Delta = \text{Rating}_{\text{new}} - \text{Rating}_{\text{old}}$$

If a player is not present as a key in the `old_ratings` input but is present as a key in the `new_ratings` input, then assume this is a new player with a starting rating of `1200`. Likewise, if a player is present as a key in `old_ratings` but is not present in `new_ratings`, assume that player did not play in the tournament and their rating is unchanged.

For example:

`old_ratings = {'Ryan Howard': 1755, 'Dwight Schrute': 1675}`
`new_ratings = {'Michael Scott': 1250, 'Ryan Howard': 1750}`

Should return:

`{'Michael Scott': 50, 'Ryan Howard': -5, 'Dwight Schrute': 0}`

```
In [34]: demo_old_ratings = {'Ryan Howard': 1755, 'Dwight Schrute': 1675}
         demo_new_ratings = {'Michael Scott': 1250, 'Ryan Howard': 1750}
```

```
In [35]: def compute_deltas(old_ratings, new_ratings):
             ###
             ### YOUR CODE HERE
             ###
             both = set(old_ratings) & set(new_ratings)
             old = set(old_ratings) - set(new_ratings)
             new = set(new_ratings) - set(old_ratings)
             deltas = {}
             for player in both:
                 deltas[player] = new_ratings[player] - old_ratings[player]
             for player in new:
                 deltas[player] = new_ratings[player] - 1200
             for player in old:
                 deltas[player] = 0

             return deltas


         # Demo
         compute_deltas(demo_old_ratings, demo_new_ratings)
```

```
Out[35]: {'Ryan Howard': -5, 'Michael Scott': 50, 'Dwight Schrute': 0}
```

The test cell below runs your function **many times**. Remove or comment out any `print` statements to avoid generating excessive output.

```
In [36]: # `ex7_test`: Test cell
         from run_tests import ex7_test
```

```
        for _ in range(200):
            ex7_test(10, compute_deltas)
        print('Passed!')

        ###
        ### AUTOGRADER TEST - DO NOT REMOVE
        ###
        test_utils.get_mem_usage_str()
```

```
Passed!
```

```
Out[36]: '49.0 MiB'
```

1.11 Wrapping up

After parsing all of the information from the text file, we can display a summary of the tournament results.

```
In [37]: import pandas as pd
         df = pd.DataFrame(index=player_scores.keys())
         df['Initial Rating'] = pd.Series(player_ratings)
         df['Score'] = pd.Series(player_scores)
         df['Expected Score'] = pd.Series(player_expected_score)
         df['Final Rating'] = pd.Series(player_final_ratings)
         df['Delta'] = pd.Series(test_utils.read_pickle('player_deltas'))
         display(df)
```

	Initial Rating	Score	Expected Score	Final Rating	Delta
Deac,Bogdan-Daniel	2627	4.0	2.827560	2639	12
Giri,A	2780	4.0	4.389932	2776	-4
Lupulescu,C	2656	3.5	3.197736	2659	3
Aronian,L	2781	4.5	4.395254	2782	1
Grischuk,A	2776	5.0	4.848488	2778	2
Vachier	2760	3.0	4.131705	2749	-11
Mamedyarov,S	2770	5.5	4.278985	2782	12
So,W	2770	5.0	4.764597	2772	2
Caruana,F	2820	3.5	4.876846	2806	-14
Radjabov,T	2765	3.0	3.288897	2762	-3

Fin! You've reached the end of this part. Don't forget to restart and run all cells again to make sure it's all working when run in sequence; and make sure your work passes the submission process. Good luck!