CptS355 - Assignment 3 - Spring 2022

Python Warm-up

Assigned: Monday, March 7, 2022

Weight: This assignment will count for 7% of your final grade.

This assignment is to be your own work. Refer to the course academic integrity statement in the syllabus.

Turning in your assignment

All the problem solutions should be placed in a single file named **`HW3.py`**. At the top of the file in a comment, please include your name and the **names of the students with whom you discussed any of the problems in this homework**. This is an individual assignment and the final writing in the submitted file should be *solely yours*. You may NOT copy another student's code or work together on writing code. You may not copy code from the web, or anything else that lets you avoid solving the problems for yourself.

In addition, you will write unit tests using `unittest` testing framework. You will write your tests in the file **`HW3tests.py`** – the template of this file is available on the HW3 assignment page. You will edit this file and provide additional tests (add at least 2 tests per problem).

To submit your assignment, please upload both files (`HW3.py` and `HW3tests.py`) on the Assignment3 (Python) DROPBOX on Canvas (under Assignments).

Please don't zip your code; directly attach the .py files to the dropbox. You may turn in your assignment up to 3 times. Only the last one submitted will be graded. Implement your code for Python3.

Grading

The assignment will be marked for good programming style as well as thoroughness of testing and clean and correct execution. **6% of the points will be reserved for the test functions.** Turning in "final" code that produces debugging output is bad form, and points will be deducted if you kept the debugging output in your code. We suggest you the following:

- Near the top of your program write a debug function that can be turned on and off by changing a single variable. For example,

```
debugging = True
def debug(*s):
     if debugging:
        print(*s)
```

- Where you want to produce debugging output use:

```
debug("This is my debugging output",x,y)
```

  instead of `print`.

(<u>How it works</u>: Using * in front of the parameter of a function means that a variable number of arguments can be passed to that parameter. Then using *s as print's argument passes along those arguments to print.)

Problems:

1. `all_games(wsu_games)` – 8%

The following dictionary stores WSU's college football game scores for the past 4 years. In 2020, WSU played only 4 games due to pandemic.

```
wsu_games = {
    2018: { "WYO":(41,19), "SJSU":(31,0),  "EWU":(59,24),  "USC":(36,39), "UTAH":(28,24),
           "ORST":(56,37),  "ORE":(34,20), "STAN":(41,38), "CAL":(19,13), "COLO":(31,7),
           "ARIZ":(69,28), "WASH":(15,28), "ISU":(28,26)},
    2019: {"NMSU":(58,7),  "UNCO":(59,17), "HOU":(31,24), "UCLA":(63,67), "UTAH":(13,38),
            "ASU":(34,38), "COLO":(41,10), "ORE":(35,37),  "CAL":(20,33), "STAN":(49,22),
           "ORST":(54,53), "WASH":(13,31), "AFA":(21,31) },
    2020: {"ORST":(38,28),  "ORE":(29,43), "USC":(13,38), "UTAH":(28,45)},
    2021: { "USU":(23,26), "PORT ST.":(44,24), "USC":(14,45), "UTAH":(13,24), "CAL":(21,6),
           "ORST":(31,24), "STAN":(34,31), "BYU":(19,21),  "ASU":(34,21), "ORE":(24,38),
           "ARIZ":(44,18), "WASH":(40,13), "CMU":(21,24)} }
```

Assume, you would like to rearrange this data and create a dictionary where the keys are the opponent teams and the values are dictionaries of games WSU played against those teams, e.g.,

```
{      'WYO': {2018: (41, 19)},
      'SJSU': {2018: (31, 0)},
       'EWU': {2018: (59, 24)},
       'USC': {2018: (36, 39), 2020: (13, 38), 2021: (14, 45)},
      'UTAH': {2018: (28, 24), 2019: (13, 38), 2020: (28, 45), 2021: (13, 24)},
      'ORST': {2018: (56, 37), 2019: (54, 53), 2020: (38, 28), 2021: (31, 24)},
       'ORE': {2018: (34, 20), 2019: (35, 37), 2020: (29, 43), 2021: (24, 38)},
      'STAN': {2018: (41, 38), 2019: (49, 22), 2021: (34, 31)},
       'CAL': {2018: (19, 13), 2019: (20, 33), 2021: (21, 6)},
      'COLO': {2018: (31, 7), 2019: (41, 10)},
      'ARIZ': {2018: (69, 28), 2021: (44, 18)},
      'WASH': {2018: (15, 28), 2019: (13, 31), 2021: (40, 13)},
       'ISU': {2018: (28, 26)},
      'NMSU': {2019: (58, 7)},
      'UNCO': {2019: (59, 17)},
       'HOU': {2019: (31, 24)},
      'UCLA': {2019: (63, 67)},
       'ASU': {2019: (34, 38), 2021: (34, 21)},
       'AFA': {2019: (21, 31)},
       'USU': {2021: (23, 26)},
  'PORT ST.': {2021: (44, 24)},
       'BYU': {2021: (19, 21)},
       'CMU': {2021: (21, 24)} }
```

Write a function `all_games` that takes the WSU's game data as input and rearranges keys as described above. You may use loops in your solution. The items in the output dictionary can have arbitrary order.

`all_games(wsu_games)` returns the above dictionary.

Important Notes:

1. Your function should not change the input dictionary value.
2. You should not hardcode the keys (years and opponent team names) in your solution.

2. common_teams(wsu_games) – 15%

Now consider that you would like to find the teams WSU played with every year - for the years included in wsu_games data. Write a function common_teams that parses through the WSU game data and finds the teams that appear in every year's games. The function should return a dictionary where the keys are the team names and the values are the list of scores against those teams.
For example:
common_teams(wsu_games) returns

```
{'UTAH': [(28, 24), (13, 38), (28, 45), (13, 24)],
 'ORST': [(56, 37), (54, 53), (38, 28), (31, 24)],
  'ORE': [(34, 20), (35, 37), (29, 43), (24, 38)]}
```

WSU played with 'UTAH','ORST' and,'ORE' in all 4 years.

Important Note:

- You are not allowed to use Python libraries we haven't covered in class in your solution.
- Your function should not change the input dictionary value.
- You should not hardcode the keys (years and opponent team names) in your solution.

3. get_wins(wsu_games, team) – 16%

Assume you would like to find the scores for the games WSU won against a given team. Write a function "get_wins" that takes the WSU game data and a team name as input, and it returns a list of tuples that includes the years and scores of each game WSU played and won against that team. For example,

```
get_wins(wsu_games,'UTAH') returns  [(2018, (28, 24))]
#WSU played 4 games with 'UTAH' but won only the 2018 game

get_wins(wsu_games,'STAN') returns
[(2018, (41, 38)), (2019, (49, 22)), (2021, (34, 31))]
#WSU played 3 games with 'STAN' and won all 3 games
```

Your function definition should not use loops or recursion but use the Python map, reduce, and/or filter functions. You may define and call helper (or anonymous) functions, however your helper functions should not use loops or recursion. You cannot use all_games function you defined in problem 1. You will not get any points if your solution (or helper functions) uses a loop. If you are using reduce, make sure to import it from functools.

4. wins_by_year(wsu_games) – 16%

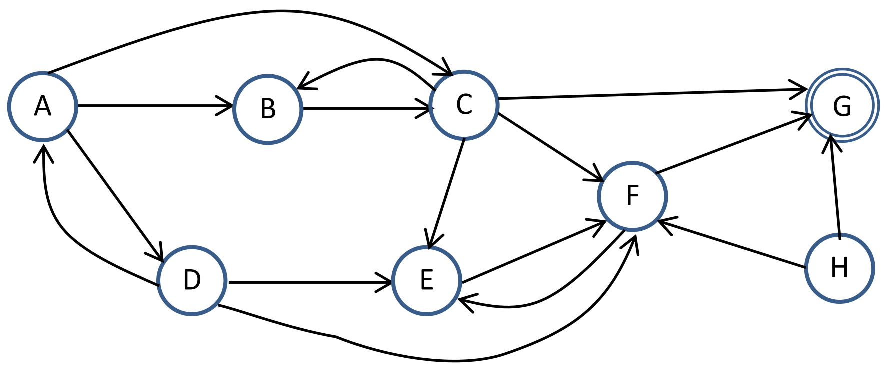
Assume you would like to find the number of games WSU won each year. Write a function "wins_by_year" that takes the WSU game data as input, and it returns a list of tuples where each tuple includes the year and the number wins during that year. For example,

```
wins_by_year(wsu_games) returns  [(2018, 11), (2019, 6), (2020, 1), (2021, 7)]
# WSU won 11, 6, 1, and 7 games in 2018, 2019, 2020, and 2021, respectively.
```

Your function definition should not use loops or recursion but use the Python map, reduce, and/or filter functions. You may define and call helper (or anonymous) functions, however your helper functions should not use loops or recursion. You <u>cannot</u> use `all_games` function you defined in problem 1. You will not get any points if your solution (or helper functions) uses a loop. If you are using `reduce`, make sure to import it from `functools`.

5. max_path(graph,node) – 16%

Consider the following directed graph where each node has zero or more outgoing edges. Assume the graph nodes are assigned unique labels. This graph can be represented as a Python dictionary where the keys are the starting nodes of the edges and the values are the set of the ending nodes (represented as Python sets). Note that some nodes in the graph are halting nodes, i.e., they don't have any outgoing edges. Those nodes are marked with double lines in the graph.



```
{'A':{'B','C','D'},'B':{'C'},'C':{'B','E','F','G'},'D':{'A','E','F'},'E':{'F'},
'F':{'E', 'G'},'G':{}, 'H':{'F','G'}}
```

Write a function, `longest_path,` which takes a graph dictionary (similar to the above) and a node label as input and returns the length of the longest path (without any cycles) from the given node to a halting node in the graph. For example, `longest_path(graph,'A')` will return 6, since the longest path from `'A'` to a halting node `'G'` – without cycles – goes through 6 nodes, including the node `'A'` and `'G'`, i.e., `['A', 'B', 'C', 'E', 'F','G']`.

You may use recursion and/or loops in your solution.

```
graph = {'A':{'B','C','D'}, 'B':{'C'}, 'C':{'B','E','F','G'}, 'D':{'A','E','F'},
'E':{'F'}, 'F':{'E', 'G'}, 'G':{}, 'H':{'F','G'}}
```

```
longest_path(graph,'A') returns 6 , i.e., ['A', 'B', 'C', 'E', 'F','G']
longest_path(graph,'D') returns 7 , i.e., ['D', 'A', 'B', 'C', 'E', 'F','G']
longest_path(graph,'C') returns 4 , i.e., ['C', 'E', 'F','G']
longest_path(graph,'F') returns 2 , i.e., ['F','G']
```

6. Iterators

`counter()` – 20%

Create an iterator that represents the sequence of words and from a string input. The iterator is initialized with the input string. At each call to `__next__()`, the iterator will return the next word and the number of times we have seen that word **so far** in the form of a tuple. The first occurrence of each word will have count 1.
The iterator should ignore all extra spaces, empty lines, and end of line characters, i.e., '\n' .

Important Note: Your `counter` implementation should extract the words from the input text as needed. An implementation that splits the complete string and dumps all words to a list all at once will be worth only 5 points.

For example:

```
numbers = """
    one
    one two
    one two three
    one two three four
    one two three four five
    one two three four five six
   """
rest = []
mywords = counter(numbers)
mywords.__next__() # skip over first 2 words
mywords.__next__()
for word in mywords:
    rest.append(word)
```

rest will be :

```
[('two', 1), ('one', 3), ('two', 2), ('three', 1), ('one', 4), ('two', 3),
('three', 2), ('four', 1), ('one', 5), ('two', 4), ('three', 3), ('four', 2),
('five', 1), ('one', 6), ('two', 5), ('three', 4), ('four', 3), ('five', 2),
('six', 1)]
```

Assignment rules – 3%

Make sure that your assignment submission complies with the following. :

- Make sure that all your debugging print statements are removed or disabled. When we run the tests, only the unittest output should be displayed.
- Make sure to include your own tests in `HW2tests.py` file.

Testing your functions (6%)

We will be using the `unittest` Python testing framework in this assignment. See https://docs.python.org/3/library/unittest.html for additional documentation.

The file `HW3SampleTests.zip` file includes 6 .py files where each one includes the unittest tests for a different HW problem. These files import the `HW3` module (`HW3.py` file) which will include your implementations of the given problems.

You should add your own tests in the `HW3tests.py` file – a template of this file is provided. You are expected to add **at least one more test case** for each problem. Make sure to create your own input dictionaries (or change the given dictionaries extensively) for problems 1,2,3,4, and 5.

In Python `unittest` framework, each test function has a "`test_`" prefix. To run all tests, execute the following command on the command line.

```
python -m unittest P1_HW3tests.py
```

You can run tests with more detail (higher verbosity) by passing in the -v flag:

```
python -m unittest -v P1_HW3tests.py
```

If you don't add new test cases you will be deduced at least 6% in this homework.