CptS355 - Assignment 3 - Fall 2020

Python Warm-up

Assigned: Monday, October 19, 2020

Due: Tuesday, October 27, 2020

Weight: This assignment will count for 6% 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, rename the **`HW3SampleTests.py`** file as **`HW3Tests.py`** and add your own test cases in this file. You are expected to add at least 2 more test cases for each problem. Choose test inputs different than those provided in the assignment prompt. When you are done and certain that everything is working correctly, turn in your by uploading on the Assignment-3(Python) DROPBOX on Blackboard. The files that you upload must be named **`HW3.py`** and **`HW3Tests.py`**. 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 (appropriate algorithms, good indentation and appropriate comments -- refer to the Python style guide) -- as well as thoroughness of testing and clean and correct execution. You will lose points if you don't (1) provide test functions / additional test cases, (2) explain your code with appropriate comments, and (3) follow a good programming style. A**round 5% of the points will be reserved for the test functions and the programming style.**

- Turning in "final" code that produces debugging output is bad form, and points may be deducted if you have extensive debugging output. 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`.

(How it works: 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. (Dictionaries)

a) **`getNumCases(data,counties,months)`** **– 10%**

Assume that you work for a "Healthcare Data Analytics" company and you write scripts to process various dataset. In your analysis, you use the CDC's COVID-19 dataset. For example, the following dataset reports the monthly new COVID cases for some counties in WA.

```
CDCdata =
{'King':{'Mar':2706,'Apr':3620,'May':1860,'Jun':2157,'July':5014,'Aug':4327,'Sep':2843},
'Pierce':{'Mar':460,'Apr':965,'May':522,'Jun':647,'July':2470,'Aug':1776,'Sep':1266},
'Snohomish':{'Mar':1301,'Apr':1145,'May':532,'Jun':568,'July':1540,'Aug':1134,'Sep':811},
'Spokane':{'Mar':147,'Apr':222,'May':233,'Jun':794,'July':2412,'Aug':1530,'Sep':1751},
'Whitman' : {'Apr':7,'May':5,'Jun':19,'July':51,'Aug':514,'Sep':732, 'Oct':278} }
```

The keys of the dictionary are the county names, and the values are the dictionaries which include the monthly new COVID cases. Note that some counties may not have any new cases in some months.

Define a function, **`getNumCases`**, which calculates the total number of new cases for a given list of counties during a given list of months. For example:

`getNumCases(CDCdata,['Whitman'],['Apr','May','Jun'])` returns 31, and

`getNumCases(CDCdata,['King','Pierce'],['July','Aug'])` returns 13587.

(*Important note*: Your function should not hardcode the county names and the month abbreviations. It should simply iterate over the keys that appear in the given dictionary. You will be deducted points if you hardcode any keys.)

You can start with the following code:

```
def getNumCases(data,counties,months):
   #write your code here
```

b) **`getMonthlyCases(data)`** **– 15%**

Assume, your supervisor asks you to reformat the data and create a dictionary that includes the number of cases for each county, organized by months. For example, when you reformat the above dictionary you will get the following.

```
{'Mar':{'King':2706,'Pierce':460,'Snohomish':1301,'Spokane':147},
'Apr':{'King':3620,'Pierce':965,'Snohomish':1145,'Spokane':222,'Whitman':7},
'May':{'King':1860,'Pierce':522,'Snohomish':532,'Spokane':233,'Whitman':5},
'Jun':{'King':2157,'Pierce':647,'Snohomish':568,'Spokane':794,'Whitman':19},
'July':{'King':5014,'Pierce':2470,'Snohomish':1540,'Spokane':2412,'Whitman':51},
'Aug':{'King':4327,'Pierce':1776,'Snohomish':1134,'Spokane':1530,'Whitman':514},
'Sep':{'King':2843,'Pierce':1266,'Snohomish':811,'Spokane':1751,'Whitman':732},
'Oct':{'Whitman':278}}
```

Define a function `getMonthlyCases` that reformats the CDC data as described above. Your function should not hardcode the county names and the month abbreviations.
(The items in the output dictionary can have arbitrary order.)

c) **`mostCases(data)` – 15%**
Assume, you would like to find the month that had the maximum total number of new cases in all counties. For example:

```
mostCases(CDCdata) returns ('July', 11487)
#i.e., July has the max number of total new cases, which is 11487.
```

Your function definition should not use loops or recursion but use the Python map and reduce functions. You should also use the `getMonthlyCases` function you defined in part(b). You may define and call helper (or anonymous) functions, however your helper functions should not use loops or recursion.

2. (Dictionaries and Lists)

a) `searchDicts(L,k)` – 5%
Write a function `searchDicts` that takes a list of dictionaries `L` and a key `k` as input and checks each dictionary in `L` starting from the end of the list. If `k` appears in a dictionary, `searchDicts` returns the value for key `k`. If `k` appears in more than one dictionary, it will return the one that it finds first (closer to the end of the list).
For example:

```
L1 = [{"x":1,"y":True,"z":"found"},{"x":2},{"y":False}]

searchDicts(L1,"x") returns 2
searchDicts(L1,"y") returns False
searchDicts(L1,"z")  returns  "found"
searchDicts(L1,"t") returns None
```

You can start with the following code:

```
def searchDicts(L,k):
   #write your code here
```

b) `searchDicts2(tL,k)` – 10%
Write a function `searchDicts2` that takes a list of tuples (`tL`) and a key `k` as input. Each tuple in the input list includes an integer index value and a dictionary. The index in each tuple represent a link to another tuple in the list (e.g. index 3 refers to the 4th tuple, i.e., the tuple at index 3 in the list) `searchDicts2` checks the dictionary in each tuple in `tL` starting from the end of the list and following the indexes specified in the tuples.
For example, assume the following:

```
[(0,d0),(0,d1),(0,d2),(1,d3),(2,d4),(3,d5),(5,d6)]
    0         1        2        3        4        5        6
```

The `searchDicts2` function will check the dictionaries `d6,d5,d3,d1,d0` in order (it will skip over `d4` and `d2`) The tuple in the beginning of the list will always have index 0.
It will return the first value found for key `k`. If `k` is couldn't be found in any dictionary, then it will return `None`.

For example:

```
L2 = [(0,{"x":0,"y":True,"z":"zero"}),
      (0,{"x":1}),
      (1,{"y":False}),
      (1,{"x":3, "z":"three"}),
      (2,{})]
```

```
searchDicts2 (L2,"x") returns 1
searchDicts2 (L2,"y") returns False
searchDicts2 (L2,"z")  returns "zero"
searchDicts2 (L2,"t") returns None
```

(*Note*: I suggest you to provide a recursive solution to this problem.
Hint: Define a helper function with an additional parameter that hold the list index which will be searched in the next recursive call.)
You can start with the following code:

```
def searchDicts2(L,k):
   #write your code here
```

3. **adduptoZero(L,n) – 10%**

Write a function, `adduptoZero`, which takes a integer list (L) and a number (n) as argument, and it returns the L's sublists of length "n", where the elements in each sublist adds up to "0".
For example:

```
adduptoZero([1,-2,3,-4,-5,6,-7,8,9,-10], 3) returns
[[-4, -5, 9], [-2, -7, 9], [-2, -4, 6], [1, 3, -4], [1, 6, -7], [1, 9, -10]]
adduptoZero(list(range (-3,3), 4) returns [[-3, 0, 1, 2], [-2, -1, 1, 2]]
adduptoZero(list(range (1,10)),2)  returns []
```

- You can assume that the input list (L) does not have any duplicates.
- You can use the "`combinations`" function from `itertools` to get the possible combinations of length "n". "`combinations`" returns a collection of tuples and you need convert this output to a list of tuples.
- Make sure to import `itertools` in your HW3.py file, i.e.,
  `from itertools import combinations`

You can start with the following code:

```
def adduptoZero(L,n):
      #write your code here
```

4. **getLongest(L) – 10%**

Write a function, `getLongest`, which takes an arbitrarily nested list of strings (L) and it returns the longest string in L. Note that the longest string can be found at any nesting level, so your function should recursively check all sublists. You should not assume a max depth for the nesting. If there are more than one string that have the max length, you should return the one that appears earlier in the list.

For example:

```
getLongest(['1',['22',['333',['4444','55555',['666666']],'7777777'],'4444'],'22'])
```
returns `'7777777'`

```
getLongest([['cat',['dog','horse'],['bird',['bunny','fish']]]])
```
returns `'horse'`

You can start with the following code:

```
def getLongest (L):
      #write your code here
```

5. Iterators

`apply2nextN()` – 20%

Create an iterator that represents the aggregated sequence of values from an input iterator. The iterator will be initialized with a combining function (op), an integer value (n) , and an input iterator (`input`). When the iterator's `__next__()` method is called, it will combine the next "n" values in the "`input`" by applying the "op" function and it will return the combined value. The iterator should stop when it reaches the end of the input sequence. If the input sequence is infinite, the `apply2nextN` will return an infinite sequence as well.

For example:

```
iSequence = apply2nextN(lambda a,b:a+b, 3, iter(range(1,32)))
# iSequence represents the sequence [6, 15, 24, 33, 42, 51, 60, 69, 78, 87, 31]

iSequence.__next__()   # returns 6
iSequence.__next__()   # returns 15
iSequence.__next__()   # returns 24
rest = []
for item in iSequence:
   rest.append(item)
# rest is [33, 42, 51, 60, 69, 78, 87, 31]

strIter =iter('aaaabbbbccccddddeeeeffffgggghhhhjjjjkkkkllllmmmm')
iSequence = apply2nextN(lambda a,b:a+b, 4, strIter)
iSequence.__next__()   # returns 'aaaa'
iSequence.__next__()   # returns 'bbbb'
iSequence.__next__()   # returns 'cccc'
rest = []
for item in iSequence:
    rest.append(item)
# rest is ['dddd','eeee','ffff','gggg','hhhh','jjjj','kkkk','llll','mmmm']
```

You can start with the following code:

```
class apply2nextN ():
    #write your code here
```

Testing your functions (5%)

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.py` provides some sample test cases comparing the actual output with the expected (correct) output for some problems. This file imports the `HW3` module (`HW3.py` file) which will include your implementations of the given problems.

Rename the `HW3SampleTests.py` file as `HW3Tests.py` and add your own test cases in this file. You are expected to add **at least 2 more test cases** for each problem. Make sure that your test inputs cover all boundary cases. Choose test input different than those provided in the assignment prompt.

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 HW3SampleTests
```

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

```
python -m unittest -v HW3SampleTests
```

If you don't add new test cases you will be deduced at least 5% in this homework.

Main Program

In this assignment, we simply write some unit tests to verify and validate the functions. If you would like to execute the code, you need to write the code for the "main" program. Unlike in C or Java, this is not done by writing a function with a special name. Instead the following idiom is used. This code is to be written at the left margin of your input file (or at the same level as the `def` lines if you've indented those.

```
if __name__ == '__main__':
    ...code to do whatever you want done...
```