CptS355 - Assignment 3 - Spring 2020

Python Warm-up

Assigned: Friday, March 6, 2020

Due: Friday, March 13, 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. **For each problem below, around 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) **`sumSales(d)` – 15%**

   Assume that you have an online sales business and you sell products on Amazon, Ebay, Etsy, etc. and you keep track of your daily sales (in $) for each online store. You maintain the log of your sales in a Python dictionary as follows:

```
{'Amazon':{'Mon':30,'Wed':100,'Sat':200},'Etsy':{'Mon':50,'Tue':20,'Wed':25,'F
ri':30},'Ebay':{'Tue':60,'Wed':100,'Thu':30},'Shopify':{'Tue':100,'Thu':50,'Sa
t':20}}
```

   The keys of the dictionary are the online stores and the values are the dictionaries which include the total sales on different days of the week. Note that if there are no sales in a particular store during the week, that store will not appear in the dictionary.

   Define a function, **`sumSales(d)`** which adds up the amount of sales you made on each day of the week and returns the summed values as a dictionary. Note that the keys in the resulting dictionary should be the days of the week and the values should be the total amount of sales (in $) you made on that day. **`sumSales`** will return the following for the above dictionary:

```
{'Fri': 30, 'Mon': 80, 'Sat': 220, 'Thu': 80, 'Tue': 180, 'Wed': 225}
```

   (*Important note*: Your function should not hardcode the store names and the days of the week. It should simply iterate over the keys that appear in the given dictionary and should work on any dictionary with arbitrary store names and days of the week)

   You can start with the following code:

```
def sumSales(d):
   #write your code here
```

   b) **`sumSalesN(L)` – 15%**

   Now assume that you kept the log of sales for several weeks and stored the sales data as a list of dictionaries. This list includes a dictionary for each week you recorded your log. Assuming you kept the log for N weeks, your list will include N dictionaries.

   Define a function `sumSalesN` which takes a list of log dictionaries and returns a dictionary which includes the total amount of sales on each day of the week. Your function definition should use the Python map and reduce functions as well as the `sumSales` function you defined in part(a). You will need to define an additional helper function to combine dictionaries.

   Example:

   Assume you have recorded your log for 3 weeks only.

```
[{'Amazon':{'Mon':30,'Wed':100,'Sat':200},'Etsy':{'Mon':50,'Tue':20,'Wed':25,'
Fri':30},'Ebay':{'Tue':60,'Wed':100,'Thu':30},'Shopify':{'Tue':100,'Thu':50,'S
at':20}},{'Shopify':{'Mon':25},'Etsy':{'Thu':40, 'Fri':50},
'Ebay':{'Mon':100,'Sat':30}},
{'Amazon':{'Sun':88},'Etsy':{'Fri':55},'Ebay':{'Mon':40},'Shopify':{'Sat':35}}
]
```

   For the above dictionary `sumSalesN` will return:

```
{'Fri': 135,'Mon':245,'Sat':285,'Sun': 88,'Thu': 120,'Tue':180,'Wed':225}
```

   (The items in the dictionary can have arbitrary order.)

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. (List Comprehension)
`busStops(buses,stop)` – 10%
Pullman Transit offers many bus routes in Pullman. Assume that they maintain the bus stops for their routes as a dictionary. The keys in the dictionary are the bus route names and the values are the stops for the bus routes (see below for an example).

```
routes = {
"Lentil": ["Chinook", "Orchard", "Valley", "Emerald","Providence", "Stadium", "Main",
"Arbor", "Sunnyside", "Fountain", "Crestview", "Wheatland", "Walmart", "Bishop",
"Derby", "Dilke"],
"Wheat": ["Chinook", "Orchard", "Valley", "Maple","Aspen", "TerreView", "Clay",
"Dismores", "Martin", "Bishop", "Walmart", "PorchLight", "Campus"],
"Silver": ["TransferStation", "PorchLight", "Stadium", "Bishop","Walmart", "Shopco",
"RockeyWay"],
"Blue": ["TransferStation", "State", "Larry", "TerreView","Grand", "TacoBell",
"Chinook", "Library"],
"Gray": ["TransferStation", "Wawawai", "Main", "Sunnyside","Crestview", "CityHall",
"Stadium", "Colorado"]
}
```

Write a function `busStops` that takes a dictionary (`buses`) similar to the above dictionary and a stop name (`stop`) as input. The function returns the list of the buses which stop at the given `stop`. For example:

```
busStops(routes,"Stadium") returns  ['Lentil', 'Silver', 'Gray']
```

You should implement your function using "list comprehension" in 2 lines. A solution that doesn't use list comprehension will be worth half of the points.

You should not hardcode route names and the bus stop names in your function.
You can start with the following code:

```
def busStops (buses,stop):
   #write your code here
```

4. (Lists)
`palindromes(s)` – 15%
Write a function, `palindromes`, which takes a string as input and returns a list of the unique palindromes that appear in the input string. You may assume that the input string doesn't have any 'space' characters and all characters are lowercase. The strings in the output should be sorted alphabetically.
(Palindrome is a string that reads the same backward as forward, e.g., madam or kayak).

```
palindromes ('cabbbaccab') returns
['abbba', 'acca', 'baccab', 'bb', 'bbb', 'cabbbac', 'cc']

palindromes (' bacdcabdbacdc') returns
```

```
['abdba', 'acdca', 'bacdcab', 'bdb', 'cabdbac', 'cdc', 'cdcabdbacdc',
'dcabdbacd']

palindromes (' myracecars') returns
['aceca', 'cec', 'racecar']
```

You can start with the following code:

```
def palindromes(S):
    #write your code here
```

5. Iterators

a) **`interlaceIter()`** – 20%

Create an iterator that represents the interlaced sequence of values from two input iterators. The iterator will iterate over the values in the input sequences and return the next element from those interchangeably. The iterator should stop when it reaches the end of either of the input sequences. It should truncate the rest of the elements in the longer sequence. If both input sequences are infinite, the `interlaceIter` will return an infinite sequence as well.

For example:

```
iSequence = interlaceIter(iter([1,2,3,4,5,6,7,8,9]),iter("abcdefg"))
iSequence.__next__()   # returns 1
iSequence.__next__()   # returns 'a'
iSequence.__next__()   # returns 2
for item in iSequence:
    print(item)
# prints the rest of the items, i.e., 'b',3,'c',4,'d',5,'e',6,'f',7,'g'
```

You can start with the following code:

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

b) **`typeHistogram(it,n)`** – 10%

Define a function `typeHistogram` that takes an iterator "`it`" (representing a sequence of values of different types) and builds a histogram showing how many values of each type appears in the next `n` elements in the sequence. **`typeHistogram`** should return a list of tuples, where each tuple includes the type names and the number of times that type appears in the next n elements. For example:

```
typeHistogram(iSequence,5)  # returns [('int', 3), ('str', 2)]
typeHistogram(iSequence,5)  # returns [('str', 3), ('int', 2)]
typeHistogram(iSequence,5)  # returns [('int', 2), ('str', 2)]
typeHistogram(iSequence,5)  # returns []
```

(Note: You can use the Python "type" function to get the type name of a value. For example:

```
type("CptS355").__name__   returns 'str'
type(10).__name__   returns 'int'
```

You can start with the following code:

```
def typeHistogram (it,n):
    #write your code here
```

Testing your functions

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...
```