Chapter10_Exercises

May 30, 2017

1 Chapter 10 Exercises

1.1 Exercise 10.1

Write a function called nested_sum that takes a list of lists of integers and adds up the elements from all of the nested lists

```
In [1]: def nested_sum(t):
            summation = 0
            for element in t:
                summation += sum(element)
            return summation

        print(nested_sum( [[1, 2], [3], [4, 5, 6]]))
```

```
21
```

1.2 Exercise 10.2

Write a function called csum that takes a list of numbers and returns the cumulative sum; that is, a new list where the ith element is the sum of the first i + 1 elements from the original list

```
In [2]: def csum(t):
            new = []
            for i in range(len(t)):
                new.append(sum(t[:i+1]))
            return new

        print(csum([1, 2, 3]))
```

```
[1, 3, 6]
```

1.3 Exercise 10.3

Write a function called middle that takes a list and returns a new list that contains all but the first and last elements

```
In [3]: def middle(t):
            new = t[1:len(t)-1]
            return new

        print(middle( [1, 2, 3, 4]))
```

```
[2, 3]
```

1.4 Exercise 10.4

Write a function called chop that takes a list, modifies it by removing the first and last elements, and returns None.

```
In [4]: def chop(t):
            del t[0]
            del t[len(t) - 1]

        t = [1, 2, 3, 4]

        chop(t)

        print(t)
```

```
[2, 3]
```

1.5 Exercise 10.5

Write a function called is_sorted that takes a list as a parameter and returns True if the list is sorted in ascending order and False otherwise. For example:

```
In [5]: def is_sorted(t):
            t2 = t[:]
            t2.sort()
            print(t)
            print(t2)
            if t == t2:
                return True
            else:
                return False

        print(is_sorted([1,2,3,4]))
        print(is_sorted([2,1,3,4]))
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
True
[2, 1, 3, 4]
```

```
[1, 2, 3, 4]
False
```

1.6 Exercise 10.6

Two words are anagrams if you can rearrange the letters from one to spell the other. Write a function called is_anagram that takes two strings and returns True if they are anagrams.

```
In [6]: def is_anagram(s1,s2):
            t1 = list(s1.lower())
            t2 = list(s2.lower())
            t1.sort()
            t2.sort()

            if t1 == t2:
                return True
            else:
                return False

        print(is_anagram("billy", "llyib"))
        print(is_anagram("parliament ", "partial men"))
        print(is_anagram("Clint Eastwood ", "Old West Action"))
        print(is_anagram("Clint Eastwood ", "Old West aaAction"))
```

```
True
True
True
False
```

1.7 Exercise 10.7

Write a function called has_duplicates that takes a list and returns True if there is any element that appears more than once. It should not modify the original list.

```
In [7]: def has_duplicates(t):
            """Does not work for nested lists"""
            tmp = t[:]
            tmp.sort()
            for i in range(len(tmp)):
                for j in range(i+1, len(tmp)):
                    if tmp[i] == tmp[j]:
                        return True
            return False

        print(has_duplicates([1,1,2]))
        print(has_duplicates([1,2,3]))
```

```
True
False
```

1.8 Exercise 10.8

This exercise pertains to the so-called Birthday Paradox, which you can read about at http: // en. wikipedia. org/ wiki/ Birthday_ paradox . If there are 23 students in your class, what are the chances that two of you have the same birthday? You can estimate this probability by generating random samples of 23 birthdays and checking for matches. Hint: you can generate random birthdays with the randint function in the random module

```
In [8]: import random as rand

        def tester(n):
            bigcount = 0
            for i in range(n):
                birthdays = []

                students = 23

                for i in range(students):
                    birthdays.append(rand.randint(0, 365))

                tmp = birthdays[:]

                tmp.sort()
                count = 0
                #print(tmp)

                for i in range(len(tmp)):
                        for j in range(i+1, len(tmp)):
                            if tmp[i] == tmp[j]:
                                count = 1
                                break
                if count == 1:
                    bigcount += 1
            print('{}% there is at least one match'.format(bigcount/n*100))

        tester(5000)
```

```
51.22% there is at least one match
```

1.9 Exercise 10.9

Write a function that reads the file words.txt and builds a list with one element per word. Write two versions of this function, one using the append method and the other using the idiom t = t + [x]. Which one takes longer to run? Why?

```
In [7]: import time

        start_time = time.time()

        def maker1():
            fin = open('words.txt')
            new = []

            for line in fin:
                new.append(line.strip())

            #return new

        maker1()

        print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 0.0451202392578125 seconds ---
```

```
In [8]: def maker2():
            fin = open('words.txt')
            new = []

            for line in fin:
                new = new + [line.strip()]

            #return new

        start_time2 = time.time()

        maker2()

        print("--- %s seconds ---" % (time.time() - start_time2))
```

```
--- 37.15094614028931 seconds ---
```

1.10 Exercise 10.10

To check whether a word is in the word list, you could use the in operator, but it would be slow because it searches through the words in order. Because the words are in alphabetical order, we can speed things up with a bisection search (also known as binary search), which is similar to what you do when you look a word up in the dictionary. You start in the middle and check to see whether the word you are looking for comes before the word in the middle of the list. If so, you search the first half of the list the same way. Otherwise you search the second half. Either way, you cut the remaining search space in half. If the word list has 113,809 words, it will take about 17 steps to find the word or conclude that it's not there. Write a function called in_bisect that takes a

sorted list and a target value and returns the index of the value in the list if it's there, or None if it's not. Or you could read the documentation of the bisect module and use that!

```
In [22]: import bisect

         def in_bisect(t, target):
             """t is a sorted list"""
             index = bisect.bisect(t, target)
             if index > 0:
                 if t[index-1] == target:
                     return True
                 else:
                     return False
             else:
                 if t[index] == target:
                     return True
                 else:
                     return False

         print(in_bisect([1,3,4,5],3))
         print(in_bisect([1,8,25,90],5))
         print(in_bisect([1,8,25,90],0))
         print(in_bisect([1,8,25,90],90))
         print(in_bisect([1,8,25,90],25))
         print(in_bisect([1,8,25,90],1))
         print(in_bisect([1,8,25,90],250))
```

```
True
False
False
True
True
True
False
```

1.11 Exercise 10.11

Two words are a "reverse pair" if each is the reverse of the other. Write a program that finds all the reverse pairs in the word list.

```
In [5]: def reverse_pair(word_list, word):
            reverse = word[::-1]
            return in_bisect(word_list, reverse)

        word_list = ['hi', 'bye', 'dog', 'cat', 'tac', 'racecar', 'racecar']

        for word in word_list:
```

```
            if reverse_pair(word_list, word):
                print(word)
```

```
tac
racecar
racecar
```

1.12 Exercise 10.12

Two words "interlock" if taking alternating letters from each forms a new word. For example, "shoe" and "cold" interlock to form "schooled".

```
In [12]: def interlock(s1,s2):
             new = []
             l1 = list(s1)
             l2 = list(s2)

             if len(l1) > len(l2):
                 tmp1 = l1[len(l2):]
                 final1 = l1[:(len(l1)-len(l2))+1]
                 print(final1)

                 for i in range(len(final1)*2):
                     if i % 2 == 0:
                         new.append(final1[0])
                         del final1[0]
                     else:
                         new.append(l2[0])
                         del l2[0]

                 delimiter = ''
                 tmp2 = delimiter.join(tmp1)
                 new.append(tmp2)

             if len(l1) < len(l2):
                 tmp3 = l2[len(l1):]
                 final2 = l2[:(len(l2)-len(l1))+1]
                 print(final2)

                 for i in range(len(final2)*2):
                     if i % 2 == 0:
                         new.append(final2[0])
                         del final2[0]
                     else:
                         new.append(l1[0])
                         del l1[0]
```

```
                delimiter = ''
                tmp4 = delimiter.join(tmp3)
                new.append(tmp4)

            print(new)
            delimiter = ''
            s = delimiter.join(new)

            return s

        print(interlock('hi', 'bye'))
        print(interlock('bye', 'hi'))
```

```
['b', 'y']
['b', 'h', 'y', 'i', 'e']
bhyie
['b', 'y']
['b', 'h', 'y', 'i', 'e']
bhyie
```