Chapter11_Exercises

May 31, 2017

1 Chapter 11 Exercises

1.1 Exercise 11.1

Write a function that reads the words in words.txt and stores them as keys in a dictionary. It doesn't matter what the values are. Then you can use the in operator as a fast way to check whether a string is in the dictionary. If you did Exercise 10.10, you can compare the speed of this implementation with the list in operator and the bisection search.

```
In [73]: fin = open('words.txt')
         dic = dict()

         for line in fin:
             dic[line.strip()] = line
```

1.2 Exercise 11.2

Read the documentation of the dictionary method setdefault and use it to write a more concise version of invert_dict.

```
In [11]: def invert_dict(d):
             inverse = dict()
             for key in d:
                 val = d[key]
                 inverse.setdefault(val, []).append(key)
             return inverse

         def invert_dict_original(d):
             inverse = dict()
             for key in d:
                 val = d[key]
                 if val not in inverse:
                     inverse[val] = [key]
                 else:
                     inverse[val].append(key)
             return inverse

         eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

```
        print(invert_dict(eng2sp))
        print(invert_dict_original(eng2sp))
```

```
{'uno': ['one'], 'dos': ['two'], 'tres': ['three']}
{'uno': ['one'], 'dos': ['two'], 'tres': ['three']}
```

1.3 Exercise 11.3

Memoize the Ackermann function from Exercise 6.2 and see if memoization makes it possible to evaluate the function with bigger arguments.

```
In [16]: def ack_original(m,n):
             if not isinstance(m, int) or not isinstance(n,int):
                 print('Undefined for non-integer inputs')
                 return None
             elif m < 0 or n < 0:
                 print('Undefined for negative inputs')
                 return None
             elif m == 0:
                 return n+1
             elif n == 0:
                 return ack(m-1,1)
             else:
                 return ack(m-1, ack(m,n-1))

         known = dict()

         def ack(m,n):
             if not isinstance(m, int) or not isinstance(n,int):
                 print('Undefined for non-integer inputs')
                 return None
             elif m < 0 or n < 0:
                 print('Undefined for negative inputs')
                 return None
             elif str(str(m)+' '+str(n)) in known:
                 return known[str(str(m)+' '+str(n))]
             elif m == 0:
                 known[str(str(m)+' '+str(n))] = n+1
                 return n+1
             elif n == 0:
                 return ack(m-1,1)
             else:
                 return ack(m-1, ack(m,n-1))

         print(ack_original(3,4))
         print(ack(3,4))
```

```
125
125
```

1.4 Exercise 11.4

If you did Exercise 10.7, you already have a function named has_duplicates that takes a list as a parameter and returns True if there is any object that appears more than once in the list. Use a dictionary to write a faster, simpler version of has_duplicates.

```
In [20]: def has_duplicates(listo):
             tmp = dict()
             for element in listo:
                 if element in tmp:
                     return True
                 else:
                     tmp[element] = 0
             return False

         listey = [1,2,3]
         listey2the_return_of_listey = [1,1,1]

         print(has_duplicates(listey))
         print(has_duplicates(listey2the_return_of_listey))
```

```
False
True
```

1.5 Exercise 11.5

Two words are "rotate pairs" if you can rotate one of them and get the other (see rotate_word in Exercise 8.5). Write a program that reads a wordlist and finds all the rotate pairs.

```
In [42]: def rotate_word(s, rot):
             new = ''
             for c in s:
                 if c != ' ':
                     term = ord(c) + rot
                     if c.islower() and term < 97:
                         term = 122 - (96 - term)
                     if c.islower() and term > 122:
                         term = 96 + (term - 122)
                     if c.isupper() and term < 64:
                         term = 90 - (64 - term)
                     if c.isupper() and term > 90:
                         term = 64 + (term - 90)
                     new += chr(term)
                 else:
```

```
                new += ' '
        return new

    def find_rotate_pairs(wordlist):
        tmpdict = dict()
        for element in wordlist:
            tmplist = []
            for i in range(25):
                tmplist.append(rotate_word(element.lower().strip(), i+1))
                for key in tmpdict:
                    if tmplist[len(tmplist)-1] == key.lower():
                        print(tmplist[len(tmplist)-1] + ' ' + element)

            tmpdict[element] = tmplist

        #print(tmpdict)

    find_rotate_pairs(['IBM', 'cheer', 'HAL', 'jolly', 'lan'])
```

```
ibm HAL
cheer jolly
```

1.6 Exercise 11.6

Here's another Puzzler from Car Talk (http: // www. cartalk. com/ content/ puzzlers): This was sent in by a fellow named Dan O'Leary. He came upon a common one-syllable, five-letter word recently that has the following unique property. When you remove the first letter, the remaining letters form a homophone of the original word, that is a word that sounds exactly the same. Replace the first letter, that is, put it back and remove the second letter and the result is yet another homophone of the original word. And the question is, what's the word? Now I'm going to give you an example that doesn't work. Let's look at the five-letter word, 'wrack.' W-R-A-C-K, you know like to 'wrack with pain.' If I remove the first letter, I am left with a four-letter word, 'R-A-C-K.' As in, 'Holy cow, did you see the rack on that buck! It must have been a nine-pointer!' It's a perfect homophone. If you put the 'w' back, and remove the 'r,' instead, you're left with the word, 'wack,' which is a real word, it's just not a homophone of the other two words. But there is, however, at least one word that Dan and we know of, which will yield two homophones if you remove either of the first two letters to make two, new four-letter words. The question is, what's the word? You can use the dictionary from Exercise 11.1 to check whether a string is in the word list

```
In [78]: # From book
         def read_dictionary(filename='c06d.txt'):
             """Reads from a file and builds a dictionary that maps from
             each word to a string that describes its primary pronunciation.

             Secondary pronunciations are added to the dictionary with
             a number, in parentheses, at the end of the key, so the
```

```
    key for the second pronunciation of "abdominal" is "abdominal(2)".

    filename: string
    returns: map from string to pronunciation
    """
    d = dict()
    fin = open(filename)
    for line in fin:

        # skip over the comments
        if line[0] == '#': continue

        t = line.split()
        word = t[0].lower()
        pron = ' '.join(t[1:])
        d[word] = pron

    return d

pronounce = read_dictionary()
# dic is word dictionary

def homophones(word1, word2, pronounce):
    if word1 not in pronounce or word2 not in pronounce:
        return False

    return pronounce[word1] == pronounce[word2]

def check(word, dic, pronounce):
    word1 = word[1:]
    if word1 not in dic:
        return False
    if not homophones(word, word1, pronounce):
        return False

    word2 = word[0] + word[2:]
    if word2 not in dic:
        return False
    if not homophones(word, word2, pronounce):
        return False

    return True

for word in dic:
    if check(word, dic, pronounce):
        print(word, word[1:], word[0] + word[2:])
```

```
llama lama lama
llamas lamas lamas
scent cent sent
```