Exercise 6

due 07.12.2017

This exercise covers Chapters 0-5 of the tutorial. Please submit your solution until 07.12., 23:59, via e-mail to `programming-l1-ws1718@ims.uni-stuttgart.de` as a **plain text** and/or Python file (which should end on `.txt` or `.py`). Please also submit in groups of **at least 3 students**, and clearly indicate the **names and immatriculation** numbers of all involved students. Submissions that do not fulfill these requirements are not accepted. Please include in your submission how much **time** it took you (roughly, in hours) to complete the exercise. Thanks!

Questions

1. We are now writing a simple part of speech tagger. The tagger assumes each word form will be assigned a single possible part of speech tag (e.g., 'flies' will always be a noun). The assignment of word forms to part of speech tags is stored in a dictionary.

   (a) Add entries to the following dictionary for the words 'dog', 'cow', and 'barks'.

   ```
   myDictionary = {'the': 'ART',
       'cat': 'NN',
       'meaows':'VERB'}
   ```

   (b) Write a function `tag()`, that takes a list of words as an argument, and returns a list of part of speech tags (based on your dictionary.). The call `tag(['the','cat','meows'])` should return the list `['ART','NN','VERB']`. Make sure the function also works for the words you added to the dictionary above.

   (c) If you have not done so, make sure the function `tag()` uses `'OTHER'`, if a word can't be found in the dictionary. Calling `tag(['the','cow','mooooohs'])`, for instance, should return `['DET','NN','OTHER']`.

   (d) In addition to returning a list of tags, we also want some administrative information printed on the screen. The function should also *print* the number of unknown words on the screen. Calling `tag(['the','cow','mooooohs','loudly'])`, for instance, should return `['DET','NN','OTHER','OTHER']`, *and* print something like `'Number of unknown words: 2'` on the screen.[1]

[1] This is typical situation. We want the result of the function as return values, and more or less important information printed on the screen.

(e) We now want to change the return value of our tagger. For this reason, we create a new function `tag2()`. This function takes the same argument as `tag()`, but it returns a list of dictionaries. In this list, each word is represented by a dictionary. This dictionary should contain two keys: `'word'` and `'pos'` to store the input word and the pos tag. Calling `tag(['the','cat','meows'])` should then return `[{'word': 'the', 'pos': 'ART'},{'word': 'cat', 'pos': 'NN'},{'word': 'meows', 'pos': 'VERB'}]`.

(f) **Bonus Exercise**. The file `tueba.lex` (available via ilias) contains a list of word forms and part of speech tags that has been extracted from a manually annotated, large German newspaper corpus. 152,063 entries are present in this list. With the following function, the file can be loaded into your program. The function takes the filename as an argument (if you have not renamed the file, it should be the string `tueba.lex`), and returns a dictionary that is structured like the one we have used before.

```
def loadDictionary(filename):
    d = {}
    with open(filename,'r',encoding='UTF-8') as fin:
        lines = fin.read().splitlines()
    for line in lines:
        a = line.split(' ')
        d[a[0]] = a[1]
    return(d)
```

Change your function `tag2()` in such a way, that you can use the dictionary loaded from the file.

2. Let's revisit the ATM program from previous exercises, but focus only on Euro bills. We would like to extend our previous program such that it tracks how many banknotes have been withdrawn in total from the ATM. Whenever a banknote is handed out, the ATM records that the denomination of the banknote. These records should be stored in a dictionary that uses the banknote denomination as keys and the number of withdrawn bills as value. If the user does not enter a number, but the string `status`, the ATM should print the number of withdrawn bills.

   This transcript shows a possible interaction with the ATM

```
Enter the amount you want to withdraw: 45
5: 1
20: 2
Enter the amount you want to withdraw: status
5: 1
20: 2
Enter the amount you want to withdraw: 45
5: 1
20: 2
```

```
Enter the amount you want to withdraw: status
5: 2
20: 4
Enter the amount you want to withdraw: 5
5: 1
Enter the amount you want to withdraw: status
5: 3
20: 4
```

Please use the following code as a starting point (it's also available in ilias as file 'e6q2.py').

```
# this is the list of possible banknotes
l = [5,10,20,50,100,200,500]

# to make our lives easer, we sort them in DESCENDING order,
# i.e., the highest value is now first
l.sort(reverse=True)

# This program runs forever
while(True):
    # store number of returned bills
    ret = []

    # Get the users input and convert it directly into an int
    amount = int(input("Enter amount you want to withdraw: "))

    # We go over the list of possible banknotes
    # and do our stuff for each note. The number of notes is
    # appended to the list called ret
    for d in l:
        ret.append(amount // d)
        amount = amount % d
    # If there is something left we show
    # an error message.
    if amount > 0:
        print("The entered amount can not be withdrawn.")
    # If not, we print results, starting with the lowest
    # banknote type
    else:
        # len(l) gives us the number of bills
        # range(len(l)) goes over each list index (from '0' to 'len(l)')
        # reversed(range(len(l))) goes over each list index (starting
        # from the back)
        for d in reversed(range(len(l))):
            # d is now a the list index, i.e., an int
            # we use this index, because we want to extract the elements
```

```
36              # from two lists.
37              if (ret[d] > 0):
38                  print(str(l[d])+": "+str(ret[d]))
```

3. **Bonus Exercise.** This is an extension to the ATM. Real ATMs do not print banknotes on demand, but have a storage compartment that needs to be refilled so and so often. If we assume that our ATM program tracks the number of bills in the storage compartment, a specific configuration might not be withdrawable, because a certain denomination is not in storage anymore. E.g., if a user wants to withdraw 25, we usually would give them one 20-euro and one 5-euro bill. If the compartment is out of 20-euro bills, we cannot hand out this combination. In this case, we actually would like to fall back to handing out two 10-euro bills and one 5-euro bill. The task of this exercise is to handle this.

   Write a function called `withdraw(amount, storage)`, that takes two arguments. The first argument is the amount a user has entered (always an int). The second argument is a dictionary that contains the current status of the storage compartment. The function should return a list containing two elements, both dictionaries. The first dictionary represents the bills to be handed out, the second argument represents the storage after handing out.

   A few example calls:

```
1  withdraw(5,
2          {5: 10})
3          # returns [{5:1},{5:9}]
4          # same behavior as before
5  withdraw(15,
6          {5:10,10:10})
7          # returns [{5:1,10:1},{5:9,10:9}]
8          # same behavior as before
9  withdraw(15,
10         {5:10})
11         # returns [{5:3},{5:7}]
12         # no 10 euro bills, therefore we get everything in 5s
13 withdraw(45,
14         {50:10})          # we only have 50s
15         # returns [{},{50:10}]
16         # the amount can't be withdrawn
```