Exercise 7

due 14.12.2017

This exercise covers Chapters 0-6 of the tutorial. Please submit your solution until 14.12., 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. Please explicitly state **type and value** of the expressions in lines 4 to 14.

```
1  s = ['live','long','and','prosper']
2  t = ['may','the','force','be','with','you']
3  u = 'They call it a Royale with cheese'.split()
4  s[1]
5  t[2:4]
6  len(s+t) == len(s) + len(t)
7  len(u)
8  s+t[3:8]
9  (s+u)[3:8]
10 len(' '.join(s))
11 u[4][3]
12 'the force' in ' '.join(t)
13 'the force' in '  '.join(t)
14 ' '.join(s)[:7] + ' '.join(s)[17:]
```

2. Before starting any kind of text processing, we typically need to tokenize text. Technically, tokenization means that we split a long string in shorter strings, and store them in a list. We will now write a function that tokenizes, following increasingly complex rules. All functions take a string as an argument and return a list of strings.

   (a) Write a function `tokenize1()` that splits at every white space. Examples:

```
tokenize1('hello world') # returns ['hello', 'world']
```

```python
tokenize1('hello! world!') # returns ['hello!', 'world!']
tokenize1('Mr. Anderson.') # returns ['Mr.', 'Anderson.']
```

(b) The second example illustrates the problem of such a simple approach: Punctuation characters are (in many languages) attached to the previous word. For processing purposes, it would be better to treat them as separate tokens. Write a function `tokenize2()` that correctly recognizes full stops, commas, exclamation and question marks as individual tokens.

```python
tokenize2('hello world') # returns ['hello', 'world']
tokenize2('hello! world!') # returns ['hello', '!', 'world', '!']
tokenize2('Mr. Anderson.') # returns ['Mr', '.', 'Anderson', '.']
```

(c) **Bonus Exercise** The third example shows yet another problem: Some dots are part of fixed expressions (like 'Mr.') and should *not* be treated as separate tokens. Write a function `tokenize3()`. This function contains a list of fixed expression tokens, e.g. `['Mr.', 'Mrs.', 'Ms.', 'Dr.', 'etc.']`.

```python
tokenize3('hello world') # returns ['hello', 'world']
tokenize3('hello! world!') # returns ['hello', '!', 'world', '!']
tokenize3('Mr. Anderson.') # returns ['Mr.', 'Anderson', '.']
```

3. In the previous exercise, we (again...) worked on the ATM. The new thing we did in Exercise 6 was the possibility for the user to enter `status` and then see the amount of bills that had been handed out before. The 'correct' solution available in ilias checks this in line 18. The solution works well if the users behave nicely and only enter numbers or 'status'. If, however, a rebellious user would enter something else, e.g., 'Python is great!', the program crashes. How would you prevent that? (description is ok, doesn't have to be running code).