P06 – Functions

1. Defining Simple Functions

Recall the exercises 1–4 from P02 (Variables):

- Braking Distance
- Celsius to Fahrenheit
- Distance between to aircrafts
- Litres to Kilograms

Develop the respective small scripts (if you haven't yet), then *refactor* (i.e., change / adapt) your code so that

- all the computation is encapsulated in a function and
- the calling code communicates with the user (**`input`**/**`print`** etc.) and delegates all real functionality to the function

Make sure your calling code (your "test script") uses only the function's parameter list and return values to interchange values with the code inside the function.

2. Credit Card Number Check

Suppose you have been hired by *MeisterCard* to write a function which checks if a given credit card number is valid. Your function **`check(card_number)`** should take a string **`card_number`** as input.

- First, if the string does not follow the format **`"#### #### #### ####"`**, where each **`#`** is a digit, it should return **`False`**.
- Then, if the sum of the digits is divisible by 10 (a "checksum" method), then the procedure should return **`True`**, otherwise it should return **`False`**.

For example, if **`card_number`** is the string **`"9384 3495 3297 0123"`** then although the format is correct, the digit's sum is 72 so you should return **`False`**.

Hints:

- You can split a string at a specific character using the function **`split()`**.
  `parts = my_string.split('a')`
- You can test if a string contains only digits with the function **`isdigit()`**.
  `only_digits = my_string.isdigit()`

3. Gender Detection in Text

Write functions to automatically detect certain features of a text document (such as the gender of the author). For all functions, assume a text document as input in form of a single string, and create meaningful function- and parameter-names.

Background: The average native English speaker knows ca. one hundred thousand words. According to research by psychologist James W. Pennebaker, however, the most revealing words in this impressive vocabulary are the ones we barely notice at all.

Gender-based language variations were among the first topics that Pennebaker studied with LIWC. Not surprisingly, he found that there are striking differences between the language usage of men and women. After analyzing thousands of blogs, essays, and other writing samples, he found that women use first-person singular pronouns like “I,” “me,” and “my” more frequently than men; according to his research, the average woman will use about 85,000 more pronouns per year than will the average man. Pennebaker attributes these findings to the tendency of women to be more self-aware than men, as has been documented by numerous psychological studies. He also found that women use more verbs and hedge phrases (such as “I think” and “I believe”), whereas men tend to use more numbers, nouns, and words per sentence.[1]

3.1 Count first-person singular pronouns

Write a function that counts the number of first-person singular pronouns (*I, me, my*) in a given text.

3.2 Document Length

Write a function that computes the number of words in a given text.

3.3 Words per Sentence

Write a function that computes the average number of words in a sentence for a given text.

3.4 Gender Detection

Use the developed functions to write a program which does very basic gender detection.

E.g., you could implement a rule of thumb that decides “female” if the input text has *many* first-person singular pronouns and “male” if the average number of words per sentence is *considerably long* (“unknown” otherwise).

[1] Check out the full article about Pennebaker and his text analysis studies: http://www.yalescientific.org/2012/03/the-secret-life-of-pronouns/

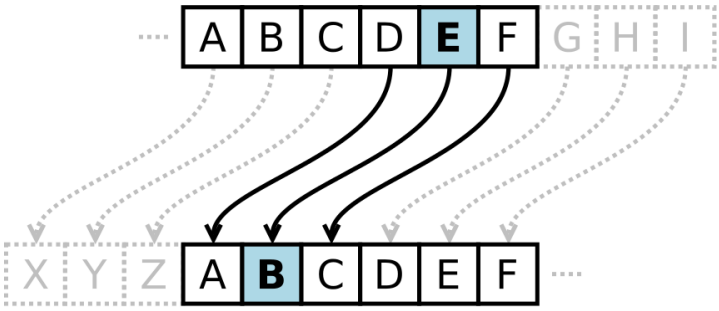
Come up with your own notions of what the above *quantities* may be according to tests with some texts of known authorship from the web or your peers, but don't use disproportional amounts of time for tuning.[2]

You can use the two provided eBooks, "Gone with the wind" by Margaret Mitchell and "The great Gatsby" by F. Scott Fitzgerald from the Project Gutenberg of Australia, as test cases. Write your script in a way then that it can read in the content of each file into a single big string and feed it to your gender detection routine.

4. Caesar cypher

Write a program that encrypts and decrypts a given message of lower-case text using a *Ceasar cypher*.

Background: In cryptography, a Caesar cypher, also known as Caesar's cypher, the shift cypher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence.



The encryption can also be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, **`a = 0, b = 1,..., z = 25`**. Encryption of a letter **`x`** by a shift **`n`** can be described mathematically as,

$$E_n(x) = (x + n)\, mod(26).$$

Decryption is performed similarly,

[2] Check out a more advanced gender detection program:
http://www.hackerfactor.com/GenderGuesser.php

$$D_n(x) = (x - n) \, mod(26).$$

The replacement remains the same throughout the message, so the cypher is classed as a type of monoalphabetic substitution, as opposed to polyalphabetic substitution.[3]

Hints:

- Think about what happens at the end of the alphabet.
- Only encrypt/decrypt the letters a-z

- Ensure all the characters in the message are in lower case. You can use the python function **`lower()`**: **`lower_message = message.lower()`**

- You can represent each character as a number (called ordinal) using the ASCII code of the character. It maps **`a`** to **`97`**, **`b`** to **`98`**, …, **`z`** to **`122`**. Check out the list of ASCII values:
  http://www.programiz.com/ascii-character-codes

```
# Program to find the value of the given character
c = raw_input("Enter a character: ")
print "The ASCII value of '" + c + "' is " + str(ord(c))
print "The original character was " + chr(ord(c))
```

- Implement the two functions

```
def encrypt(message, key):
    # TODO
    return encrypted_message
def decrypt(message, key):
    # TODO
    return decrypted_message
```

- …or write a single function **`caesar_cypher(message, key)`** that is able to de- and encrypt the **`message`** based on the value the **`key`** (positive or negative):

```
def encrypt(message, key):
    return caesar_cyper(message, key)
def decrypt(message, key):
    return Caesar_cypher(message, -key)
```

→ How/why is this possible?

[3] Check out the full Wikipedia article on Caesar cyphers: http://en.wikipedia.org/wiki/Caesar_cipher

5. Conway's Game of Life

"Game of Life" is a simple simulation of the "life" and "death" of cells on a grid that produces astonishing results. It has been developed by John Horton Conway in 1970 and is said to having consumed the majority of CPU cycles of all office computer of that era due to its vast distribution and viewership.

Take the minimal example in **`p06_game_of_life_simple.py`** (See the appendix of V03 for a detailed explanation of this code, if you wish) and first refactor it to that it uses several functions for isolated tasks. Then, extend it in several ways to appear more interesting to watch. For example, …

- You could indicate the "age" of alive cells, e.g. by color
  → How could some notion of age be computed? When is its display "valuable" for an observer?
- You could add some "artificial evolution" that randomly switches the state of cells, thereby skipping the rules.
  → How much randomness is helpful to make the simulation more interesting to watch without disturbing the order of that universe too much?
- You could read the Wikipedia article to get inspired for more ideas:
  http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
- You could port the script to your favorite iDevice using e.g. Pythonista:
  http://omz-software.com/pythonista/