# P04 – Functions

## 1. Rewrite existing code

Your task is to rewrite some of your code of P01 so that it is encapsulate in a function. First study carefully slides 5, 6 and 9 of V07 where it is described how to write a function for the breaking example. Now write a function for each of the exercises:

- 4.3 Distance between to aircrafts
- 4.4 Liters to Kilograms – Another conversion task

At the end of the script you shall call the functions with predefined values as arguments. Do not use **`input()`** or **`raw_input()`** to ask a user for the values.

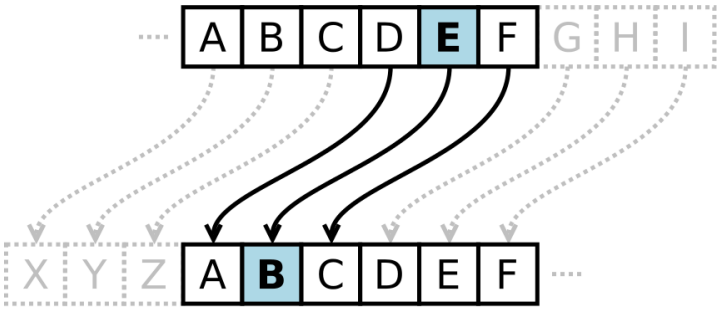
Pseudo code:

```
# Script

# define the function and its behaviour
def do_something(value):
    # something is done
    <Block>

some_value = 5

# call the function
function_response = do_something(some_value)

# do whatever you want with the function response
print(function_response)
```

## 2. Caesar cipher

Write a program that encrypts and decrypts a given message of lower-case text using a *Ceasar cipher*. Design it in a way that you have two separate functions **`encrypt()`** and **`decrypt()`**.

Background: In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, 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,

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

The replacement remains the same throughout the message, so the cipher is classed as a type of monoalphabetic substitution, as opposed to polyalphabetic substitution.[1]

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))
```

[1] Check out the full Wikipedia article on Ceasar ciphers: http://en.wikipedia.org/wiki/Caesar_cipher

- Implement the two functions

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

## 3. Credit Card Number Check (optional)

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()`**

## 4. 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 new 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.[2]

### 4.1 Count first-person singular pronouns

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

### 4.2 Document Length

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

### 4.3 Words per Sentence

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

### 4.4 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 "male" if less than *x* words are given in response to a prompt, and "female" if the input text is *considerably long* or has *many* first-person singular pronouns ("unknown" otherwise").

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.[3]

➔ *In any case, discuss all of your results and findings with your advisor before the end of the second week's lab session.*

[2] Check out the full article about Pennebaker and his text analysis studies:
http://www.yalescientific.org/2012/03/the-secret-life-of-pronouns/
[3] Check out a more advanced gender detection program:
http://www.hackerfactor.com/GenderGuesser.php