Computational Biology I: Quantitative Data Analysis
Frühjahrsemester 2018

Exercise 1

Anne Krämer, Athos Fiori and Jérémie Breda

Getting started

1. Login to the ADAM https://adam.unibas.ch and go to the course workspace
2. Download the file "Exercise1.zip" and save it in the directory "Exercise1"
3. Unpack the file contents into your working directory.

If you need assistance in order to get you environment ready for work, please refer to the Preliminary exercises sheet in ADAM.

Three programming examples

Now we are ready for programming in Python! Lets start by writing a simple function that adds two numbers.

```
def addTwoNumbers(a,b):
    return(a+b)
```

Type in the function in the Spyder editor. Save it in the "Exercise1" directory, e.g. under the file name "my_program_exemple.py". Load the function into the Python shell by pressing the "Run" button in the menu. Now test whether the function works correctly within the Python shell.

```
In [1]: addTwoNumbers(3,-2)
Out[1]: 1

In [2]: addTwoNumbers(3.4,-2)
Out[2]: 1.4
```

The second function introduces you to an important data type called lists. It is designed to return the maximum in a list of numbers.

```
def maximumNumber(li):
    maximum = li[0]
    for n in li:
        if n>maximum:
            maximum = n
    return(maximum)
```

You can write this function within the previous file. Thus, type the function into the editor, save it, and run it. Now check its functionality.

```
In [1]: maximumNumber([-3,4,1,8])
Out[1]: 8
```

Lists allow you to store a set of objects like numbers in a sequential manner and access them by indexing the list. Note that the first element of a list is indexed by 0, the last element of list of length n is indexed by $n-1$.

```
In [1]: x = [3,4,5,6]

In [2]: x[0]
Out[2]: 3

In [3]: len(x)   # len() returns the length of a list
Out[3]: 4

In [4]: x[3]
Out[4]: 6

In [5]: x[4]     # WRONG INDEX! The last element of list of length 4 is x[3]
---------------------------------------------------------------------------
IndexError
```

The following function shows you how you can build up a list using the `append` method of list objects. It computes the growth in money (x) over a number of years (t) given a certain interest rate (r) using the formula:

$$x_{t+1} = x_t + rx_t$$

```
def moneyGrowth(initialAmount,years,interestRate):
    savings = [initialAmount]
    for n in range(years):
        newAmount = savings[n] * (1.0 + interestRate)
        savings.append(newAmount)
    return(savings)
```

The function works like this. We initialize the list that collects our savings over the years with the inital amount of money. We then iterate over the years using a for-loop. The command `range(n)` produces a list of n numbers ranging from 0 to $n-1$. Every round of the loop we calculate the new amount of money according to the formula above and append the result to the end of the list. You can check visually the result by plotting it.
First you should add the plotting module, by writing :

```
import matplotlib.pyplot as plt
```

at the beginning of your script. After running your code, you can then test.

```
In [1]: x = moneyGrowth(100,25,0.03)
In [2]: plt.plot(range(25+1),x)
```

Homework: Programming Part

Open the file `Exercise1.py` which is part of the “Exercise1.zip” archive and complete the mentioned functions, as demanded.

1. The function `sumList` takes a list of numbers $n_1, n_2, ..., n_K$ and calculates their sum.

$$s = \sum_{i=1}^{K} n_i$$

2. The function `counts2frequencies` takes a list of integers $n_1, n_2, ...n_K$ (counts) and normalizes each count by the total number of counts.

$$f_i = \frac{n_i}{\sum_i n_i}$$

The output is a list with the frequencies $f_1, f_2, ..., f_K$.

3. The function `factorial` calculates the factorial defined as

$$n! = 1 \cdot 2 \cdot 3 \cdot ... \cdot n = \prod_{i=1}^{n} i$$

4. The next exercises introduces some basics of combinatorics. Imagine that you want to create a new language from scratch. First, you created some characters (or letters) and you want to explore the possibilities of words that you have.

   (a) Imagine that you have a number N of different letters. How many different words of k letters can you form with those letters? For now, you can use each letters as many times as you want in your words.
   Once you came up with a formula, fill it in the `number_of_k_words` function. The function takes the number of letters N and the word length k as input and gives back the total different words you can create.
   Hint: Try to think of how many possibilities there is for the first letter of your word, then for the second, and so on until the k-th position. It might help to take small numbers (*e.g.* $N = 4,\ k = 3$) and count by hand. You might need the exponent operator in python, x^y is written `x**y`

   (b) Still with the same N letters, you want to count the number of words that you can create, not just with k letters, but with any number of letters from 0 to N. You can use each letters as many times as you want in your words.
   Once you came up with a formula, fill it in the `number_of_words` function. The function takes the number of letters N as input and gives back the total different words you can create.
   Hint: You might want to use the result of the previous exercise (4a). Indeed, you want to count the number of possible k-words for all the different values of k between 0 and N. Take notice that we do consider $k = 0$, meaning that the word with no letter is a word too. You might want to use a for loop or the exponential sum formula:

$$\sum_{i=0}^{N} x^i = \frac{1 - x^{N+1}}{1 - x}$$

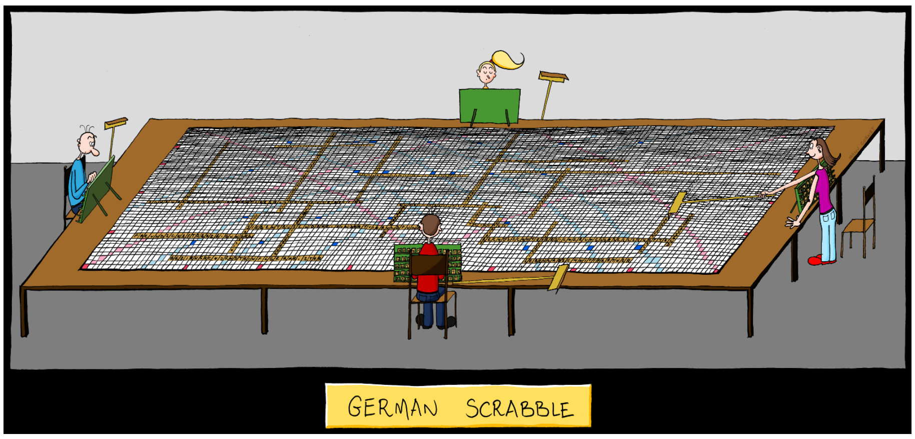
   (c) As much as you like your words, you would like to get rid of the words in which the same letter is repeated multiple times. How many words of exactly k letters can you form with the additional constraint that you can use each letter only **once** in each word.
   Once you came up with a formula, fill it in the `number_of_k_words_no_repetition` function. The function takes the number of letters N and the word length k as input and gives back the total

different words you can create.
Hint: Try to think in the same way as you did for point (4a). So, how many possibilities are there for the first letter? Then how many for the second position, given that you used one letter at the first position? And so on until the last k-th position where you already used $N-1$ of your letters. Again, it might help you to take small numbers (*e.g.* $N = 4,\ k = 3$) and count by hand. Again you can use a for loop to code that function, but a more elegant formula might use the basic fraction property

$$\frac{a \cdot b \cdot c \cdot d \cdot e}{c \cdot d \cdot e} = a \cdot b$$

(d) You're pretty happy with your different words and you want to play a bit of *Scrabble* with your new language. You're wondering how many different words you could form with k different letters.
Once you came up with a formula, fill it in the `number_of_permutations` function. The function takes the number of letters k as input and give back the total different words you can create with those letters.
Hint: This exercise is very similar to (4c). You want to know how many words you can create by rearranging k different letters. You need to use all the letters meaning that each word is of length k.

(e) You're still playing *Scrabble* with your new language. You're wondering how many different combinations of k letters you can draw out of the letters bag. Consider that each of the N letters can be drawn only once, or similarly that there is only one of each N different letters in the bag.
Once you came up with a formula, fill it in the `number_of_k_draws_no_repetition` function. The function takes the number of different letters in the bag N, and the number of drawn letters k as input and gives back the number of different combinations of letters that are possible.
Hint: This last exercise is a bit more tricky, but if you think in terms of the result of exercises (4c) and (4d), it will bring your reflection to the solution.
As in (4c), you want to count the number of different words of length k you can create from N different letters, with no repetition. Except, now you don't care about the position of each letters in a word. In other words you don't want to count all the rearrangements that you can do within each k letters word (you computed this in exercise (4d)). So what you really want to count is just a fraction of the number of words that are counted in `number_of_k_draws_no_repetition`. So what fraction is it?



Note, that each function is documented with a doc string. If you import the functions into IPython with

```
In [1]: run Exercise1.py
```

you can have a look at the documentation by using the help command. For example,

```
In [2]: help(sumList)
```

The documentation describes the form of the function, i.e. what variables it takes as inputs and produces as outputs, and an example of the behavior of the function on an example input.
Once you completed a function, check wether it is returning the correct result. You can easily test that by loading your completed function in the `exerise1.py` file into IPython,

```
In [1]: run Exercise1.py
```

and then test a particular function by calling it with suitable input arguments and checking for the correct output, e.g.,

```
In [2]: counts2frequencies([2,2])
Out[2]: [0.5, 0.5]
```

How to submit the exercises

Submit your completed `Exercise1.py` file by email to biocomp1-bioz@unibas.ch. The submission deadline is on Thursday, March 15th, midnight.

1. Check that your functions are working properly before submitting them!
2. Don't forget to attach your exercise file!
3. Don't change the name of the file or the name of any function within the file!
4. Please send your solutions from the same email address that you used to registering at the course.