Data Handling for Business Analytics
Exercise number 3, Due Lecture #5

Test your code with different inputs and see how it behaves.

Please submit all the solutions in a **single .py** file (separated by #comments). Make sure that your **'.py'** file is **ready to run**, that is, loading it via Spyder and running it must work.

Do not send WORD files. Do not send print screens. Do not add your output.

Please name your .py file based on the following pattern: Ex(*number*)_ **ID**.py.
For example, if my ID is 1234 and this is exercise number 3, the name of the .py file should be: Ex3_1234.py.

Once done, upload your .py file to **Moodle.**

Feel free to ask questions regarding the HW/course material (via Moodle). Good luck!

1. Write a function **print_pairs(text)** that receives a string *text* and prints every pair of subsequent characters in it in a single line separated by single space(s). For example:

```
>>> print_pairs('ACGTACGT')
AC CG GT TA AC CG GT
```

2. Write a function **is_prime(n)** that receives a number $n>2$ and returns *True* if n is a prime number, otherwise returns *False* (Hint: go over all the numbers that are lower than n and check whether they divide it). Notice that the returned value should be of type *bool* (not *str*).

3. Implement a function **log10_list_comprehension(l)** that receives a list. Using the special *list comprehension* syntax, you should generate and return a **new** independent list where each number in the list is the logarithm (in base 10) of the corresponding number in the input list. You may assume that the numbers (if any) are positive. If the list is empty you need to return a new empty list. Guidance: you need to use *log10* function from *Math* Module (i.e., using an 'import' command).

```
>>> l = [1 ,10, 20]
>>> result = log10_list_comprehension(l)
>>> print(result)
[0.0, 1.0, 1.3010299956639813]
>>> print(l)
[1, 10, 20]
```

4. Implement a function **log10_inplace(l)** that receives a list and **replaces** each number in the list by its logarithm (in base 10). The list should be manipulated

in-place, that is, the function should return **None**. You may assume that the numbers (if any) are positive. If the list is empty you need to keep it as is. Guidance: you need to use *log10* function in *Math* Module.

```
>>> l = [1 ,10, 20]
>>> result = log10_inplace(l)
>>> print(result)
None
>>> print(l)
[0.0, 1.0, 1.3010299956639813]
```

5. Write a function **most_common_value_in_dict(d)** that receives a dictionary *d* and returns the most common value in it. You may assume that *d* is not empty and that the dictionary values are immutable (hint…).
   For example:

```
>>> most_common_value_in_dict({'1':2, 'a':3, 'x':3, 4:3, 2:2, 'y':1})
3
```

6. In class we've represented a matrix as a nested list such that each list in the nested list represents a line of the matrix, put differently, the element in the i^{th} row and in the j^{th} column is the $j\text{-}1^{th}$ element in the $i\text{-}1^{th}$ list of the nested list. Write a function **get_highest_average_column(mat)** that receives a matrix *mat* (as a nested list) and prints the index of the column (in this case, index is starting from 1) with the highest average of values in it. If there is a "tie" you should print the last one. You may assume the matrix is legal and contains at least one number. For example:

   The matrix

$$\begin{bmatrix} -4 & 5 & 3 & 3.2 \\ 5 & 6.0 & -2.4 & 6 \\ 7 & 3 & 2 & -1 \end{bmatrix}$$

   which is represented by

```
[[-4, 5, 3, 3.2], [5, 6.0, -2.4, 6], [7, 3, 2, -1]]
```

   the output will be:

```
2
```

   Since the average of the values in the second column (~4.666) is the highest.

   The matrix

$$[-42]$$

   which is represented by

```
[[-42]]
```

the output will be:
1

Since the average of the values in the first column (42.0) is the highest (it's the only one…).

Challenge (optional):
An *increasing subsequence* in the matrix is a sequence of numbers in the **same row** in which each value is equal or greater than the value preceded it. Write a function **get_longest_increasing_subsequence_length(mat)** that receives a matrix *mat* (as a nested list) and returns the length of the longest *increasing subsequence* in it. If there is a "tie" you should return the last one You may assume the matrix is legal and contains at least one number. For example:

For the matrix

$$\begin{bmatrix} -4 & 5 & 3 & 3 \\ 5 & 6 & -2 & 6 \\ 7 & 3 & 2 & -1 \end{bmatrix}$$

The output will be:
2

For the matrix

$$[-42]$$

The output will be:
1