Exam BIO134: Programming in Biology HS2019

23.01.2020; 9:30 – 11:30

Question 1

```
s = 'TCGATAATTTCTGACATGGCGTCAATGGTACTCGCGGAG'
```

Write a program that creates a list of strings with 3-letter codons starting from the start codon *ATG*. It should then print the list:

```
['ATG', 'GCG', 'TCA', 'ATG', 'GTA', 'CTC', 'GCG', 'GAG']
```

The program should be general enough such that it would still work if *s* was replaced by another sequence with a different length. You may assume that the *ATG* start codon starts "in frame" (i.e. after a multiple of 3) and that the sequence length is a product of 3.

Question 2

```
import numpy as np
np.random.seed(0)
n = 9
```

Write a program that starts with the lines above and simulates rolling a dice (*würfeln*) *n* times by using *np.random.randint(1,7)*. This program should create a dictionary with the dice scores (*Augenzahlen*) as keys. The values should be lists of integers between 1 and *n* that indicate in which try (or tries) the corresponding dice score was rolled – or empty lists in case the score was never rolled in the *n* tries.

It should then print the dictionary:

```
{1: [3], 2: [7], 3: [], 4: [4, 5, 6, 8], 5: [1], 6: [2, 9]}
```

The program should be general enough such that the number of simulated tosses (*Würfe*) changes if *n* is changed.

Note that a dictionary does not have a specified order, so that the same dictionary can be printed in many different ways. For example, a correct program might also print the dictionary as follows:

```
{6: [2, 9], 3: [], 5: [1], 4: [4, 5, 6, 8], 1: [3], 2: [7]}
```

Question 3

```
lys1 = [35, 4, 12, 6, 4, 27, 4, 3]
lys2 = [1, 43, 2, 3, 85, 34]
```

Write a function *min_max()* that takes a list with integers between 1 and 99 as input and returns the minimum and maximum value from the list.

Call the function as follows:

```
print(min_max(lys1))
print(min_max(lys2))
```

The program should now print:

```
(3, 35)
(1, 85)
```

You are not allowed to use existing python functions like *min()* or *max()* for this.
The function should be general enough that it would work according to the same principle if the lists are replaced by other lists of different length with integers between 1 and 99.

Question 4

Here is a list of UZH rectors (presidents) in reverse chronological order:

```
rectors = [['Hengartner', 'Jarren', 'Fischer', 'Weder'], ['Michael',
            'Ottfried', 'Andreas', 'Hans'],
            [2014, 2013, 2008, 2000]]
```

Write a program that collects the information on each rector into a new list of lists. Each sublist of this new list should be in the format [first name, surname, start year as rector]. Sublists should be stored in proper chronological order.

The program should then print the new list:

```
[['Hans', 'Weder', 2000], ['Andreas', 'Fischer', 2008], ['Ottfried',
'Jarren', 2013], ['Michael', 'Hengartner', 2014]]
```

The program should still work if the list *rectors* contained information about a different number of rectors. You may assume that the rectors in the original lists are stored in reverse chronological order. You may also assume that there is not more than one rector for a given year.

Question 5

```
poem = 'Wanderer tritt still herein - \
        Schmerz versteinert nun die Schwelle - \
        da erglaenzt in reiner Helle - \
        auf dem Tische Brot und Wein.'
```

Write a program that, based on *poem,* prints the exact following lines:

```
Wanderer ********  1
Schmerz *********  5
Schwelle ********  9
Helle *********** 14
Tische ********** 17
Brot ************ 18
Wein ************ 20
```

Each printed line is a 20-character string corresponding to each word in the string *poem* that starts with a capital letter. The lines are ordered according to the appearance of the words in the string *poem*. A word is defined here as a group of letters of the alphabet without any special characters.

The format of each printed line is the following:

- Each line starts with a *word subunit* consisting of the word starting with a capital letter followed by a space.
- The line ends with a *number subunit* consisting of one or two spaces and the positional rank that the word has among all words in the poem; For example, *Wanderer* is the first word, *Schmerz* is the fifth word in the poem. The number is printed aligned such that its last digit is the last character in the line.
- The space in between the *word subunit* and the *number subunit* is filled with an appropriate number of stars.

The program should be general enough such that it would still work according to the same principle if the poem was replaced by a different string which may contain different special characters. You may assume that no word is shorter than 2 characters or longer than 15 characters and that the text contains no more than 99 words.

The points for this question are (largely) based on the printed output. If you don’t manage to print the required output, make sure to print your intermediate results.

Question 6

The file *lineage.txt* contains genomes with unique ID numbers and the taxonomic assignment that they were given. For example, the first and the 81st lines of the file are:

```
'genome_31 Bacteria;Firmicutes;Clostridia;Clostridiales\n'
'genome_22 Bacteria\n'
```

The taxonomic ranks are:

```
taxa =['domain','phylum','class','order','family','genus','species']
```

The first line describes the genome number 31. It has been identified to belong to the order *Clostridiales*, with all the higher taxonomic ranks listed before, separated by semicolons. The 81st line describes the genome number 22, belonging to the domain *Bacteria*. Note that no other taxonomic ranks were assigned.

Write a function *lineage_analysis()* that takes as input a list of taxonomic ranks and a text file. The text file may contain information about any number of genomes and their assigned lineages following the systematics in *lineage.txt*. The function should analyze how often each taxonomic rank in the input list was assigned to a genome in the text file, and output two dictionaries, *lineages* and *ranks*.
The dictionary *lineages* should contain genome numbers (integers) as keys and their assigned lineage as values of the data type *list*. Please note, that some species names consist of two words separated by a space (e.g. genome_3 is an *Eubacterium ramulus*).
The dictionary *ranks* should have all the taxonomic ranks as keys and the number of times each rank was assigned as integer values. For example, the most general rank *domain* is assigned to 100 genomes whereas the rank *species* is only assigned to 11 genomes.

When calling the function as follows:

```
lineages, ranks = lineage_analysis(taxa, 'lineage.txt')
print(lineages[31])
print(lineages[22])
print(lineages[3][-1])
print(ranks)
```

it should print:

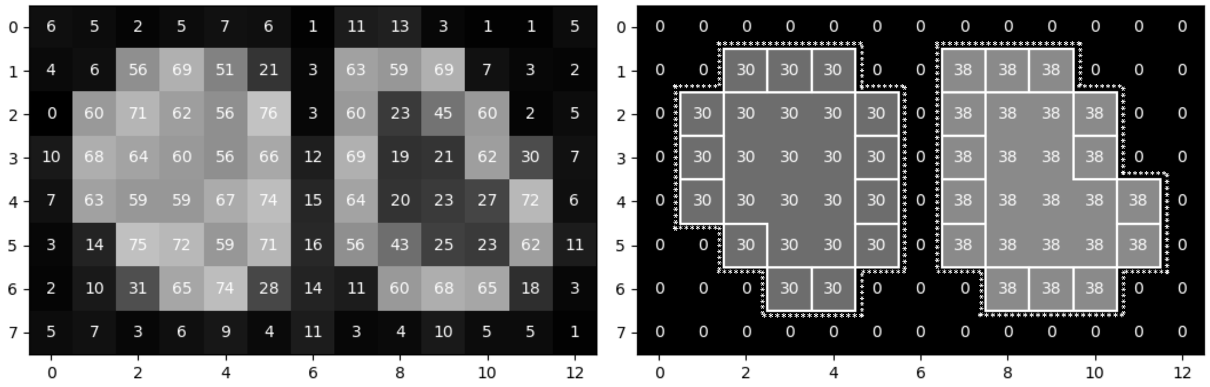
```
['Bacteria', 'Firmicutes', 'Clostridia', 'Clostridiales']
['Bacteria']
Eubacterium ramulus
{'domain': 100, 'phylum': 58, 'class': 58, 'order': 58, 'family':
27, 'genus': 14, 'species': 11}
```

(Please note that the order of elements in dictionaries is not important.)

You may assume that if a genome was mapped to a certain level, an assignment was found for all the higher ranks as well. In addition, you may assume that the text file contains taxonomic lineages according to the list *taxa* and that each genome was assigned at least to the most general taxonomic rank (*domain*).

Please note, that the points you get for this question are largely based on your printed output and you will be given 0 points if the program prints nothing. In case not all the printed values are correct, you may still get part of the points.

Question 7



	0		2		4		6		8		10		12
0	6	5	2	5	7	6	1	11	13	3	1	1	5
1	4	6	56	69	51	21	3	63	59	69	7	3	2
2	0	60	71	62	56	76	3	60	23	45	60	2	5
3	10	68	64	60	56	66	12	69	19	21	62	30	7
4	7	63	59	59	67	74	15	64	20	23	27	72	6
5	3	14	75	72	59	71	16	56	43	25	23	62	11
6	2	10	31	65	74	28	14	11	60	68	65	18	3
7	5	7	3	6	9	4	11	3	4	10	5	5	1

	0		2		4		6		8		10		12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	30	30	30	0	0	38	38	38	0	0	0
2	0	30	30	30	30	30	0	38	38	38	38	0	0
3	0	30	30	30	30	30	0	38	38	38	38	0	0
4	0	30	30	30	30	30	0	38	38	38	38	38	0
5	0	0	30	30	30	30	0	38	38	38	38	38	0
6	0	0	0	30	30	0	0	0	38	38	38	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 1: Left panel: grayscale image with signal intensities as pixel values. Right panel: segmented cells with positive integers as pixel values indicating cell numbers the pixels belong to and 0s for background pixels. Cell boundary pixels are indicated with white border lines. The cell's circumferences are indicated with dashed white lines.

Suppose we have imaged some fluorescently labelled cells, resulting in pixel arrays filled with signal intensity values (left image in figure 1; numpy array *raw* in the file *for_copying.py*). These cells were segmented into objects, such that we now know for each pixel in the image whether it belongs to a given cell or to the background. This segmentation information is stored in another array, of the same size as the fluorescent image, with positive integers for cell pixels and with zeros for background pixels (right image in figure 1; numpy array *segmented* in the file *for_copying.py*). The cell pixel values indicate the (meaningless but unique) cell number that was assigned when the cell objects were separated from each other and from the background. Our example shows two cells: cell number 30 and cell number 38.
The fluorescent dye that was used can penetrate dead cells but stays outside of living cells. This behavior allows to distinguish dead cells from living cells by comparing the fluorescent signal at cell boundaries to the signal inside the cells: our example cell 30 is apparently dead whereas cell 38 is alive.

Write a program that calculates for each cell the following features: the area in pixels, the mean pixel intensities of two cellular sub-compartments – the *cell inside* and *cell boundary*, the ratio of these two values (cell inside / boundary) and the cell circumference in dimensionless units (number of sides the *cell boundary* presents to the background). The cell boundary pixels are indicated with white border lines, the cell circumferences with dashed white lines in figure 1. You may assume that no cell is touching the image boundary nor a neighboring cell.
Print your results such that in the output there is one line per cell with values for each of the requested features in the following order: the cell number, the cell area, the mean intensity of the *cell boundary*, the mean intensity of the *cell inside*, the ratio of the two values and the cell circumference. Print 0's for features that you did not manage to determine.

Your program should still work on images of different sizes and different number of cells and/or cell identities. You may assume that both images have the same dimensions.

Please note, that the points you get for this question are largely based on your printed output and you will be given 0 points if the program prints nothing. In case not all the printed values are correct, you may still get part of the points.