

Exam BIO134: Programming in Biology HS2023

12.01.2024; 10:30-13:00

Question 1

```
codons = ['ATG', 'ATT', 'AGG', 'TAC', 'AAG', 'GGA', 'TAG', 'GTG']
```

Write a program that creates a string where the codons (three-letter strings) from the list *codons* are concatenated together in the order they occur in the list until, but not including, the stop codon 'TAG'.

Your program should print the newly created sequence string as follows:

```
ATGATTAGGTACAAGGGA
```

The program should be general enough such that it would still work according to the same principle if *codons* was replaced by another list of codons of different length and content. You may assume that there is always one 'TAG' stop codon.

Question 2

```
tens = [7, 4, 6, 0, 3, 5, 1, 4, 8, 2]
ones = [3, 1, 5, 2, 6, 2, 9, 8, 4, 1]
```

Write a program that creates a new list, containing integers that are computed as combinations of the single digit numbers of the above two lists. The “tens” digit should come from the list *tens* and the “ones” digit should come from the list *ones*.

For example, the first number in the new list is created by taking ten times the first number from *tens* plus the first number from *ones*: $73 = 10 \cdot 7 + 3$

The program should print the new list:

```
[73, 41, 65, 2, 36, 52, 19, 48, 84, 21]
```

Your program should be general enough such that it would still work according to the same logic if *tens* and *ones* contained different numbers and were of another length. You may assume that the length of *tens* is always the same as the length of *ones* and that both contain integers between 0 and 9.

Question 3

Write a function *composition()* that takes a protein sequence as input and finds its amino acid composition. For this, the function should create and return a dictionary where the keys are the amino acids found in the input sequence and the values are numbers indicating how often the amino acids occur in the sequence.

Call the function as follows:

```
sequence = 'MNREGAPGKSPEEMYIQQKVRVLLMLRKMGSNLTASEEEQGAEDVVMAFSRRRQ'
print(composition(sequence))
```

The program should now print:

```
{'M': 5, 'N': 2, 'R': 6, 'E': 7, 'G': 4, 'A': 4, 'P': 2, 'K': 3, 'S': 4, 'Y': 1, 'I': 1, 'Q': 4, 'V': 4, 'L': 4, 'T': 1, 'D': 1, 'F': 1}
```

Note that the order of the keys is irrelevant, it is only important that every key has the correct associated value. The function should be general enough such that it would work according to the same principle when called with a different input sequence of another length.

Question 4

```
pi_poem = 'How I wish I could recollect, of circle round, '
pi_poem += 'the exact relation Arkimedes unwound.'
```

The string *pi_poem* contains a poem where the length of each word represents a digit of the mathematical constant pi (π). The first word (*How*) corresponds to the digit before the decimal point and the subsequent words represent the decimal places (13 digits in this example). Note that only the alphabetic letters count to a word's length, not the special characters comma (,) and full stop (.).

Write a program that extracts the value of pi from *pi_poem*. Print the process of constructing pi, formatted exactly as follows, with 10 spaces reserved for the word column:

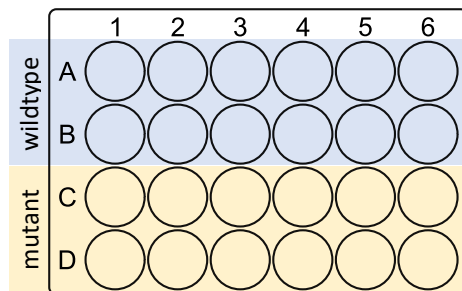
```
3 How          3
.
1 I            3.1
4 wish        3.14
1 I           3.141
5 could       3.1415
9 recollect    3.14159
2 of          3.141592
6 circle      3.1415926
5 round       3.14159265
3 the         3.141592653
5 exact       3.1415926535
8 relation    3.14159265358
9 Arkimedes   3.141592653589
7 unwound     3.1415926535897
```

Note that the data type of your pi may be string or float. Should rounding errors occur with pi as float, these can be ignored.

The program should still work according to the same principle on a different pi poem representing a different number of decimal places. You may assume that there are no special characters other than comma and full stop.

Question 5

The file *densities.csv* contains optical density (OD) measurements for two different yeast strains: wildtype and mutant. Each line in the file corresponds to the density of one yeast cell culture grown in a single well on a 24-well microtiter plate. The format of each line is as follows: well row (labeled A to D), well column (numbered 1 to 6), OD value. Rows A and B contain wildtype, rows C and D contain mutant yeast cultures.



First three lines in densities.csv

A,1,0.226

B,1,0.217

C,1,0.023

```
letter_to_index = {'A': 0, 'B': 1, 'C': 2, 'D': 3}
```

Write a program that:

- creates a NumPy array representing the microtiter plate (4 rows, 6 columns) filled with the OD measurements extracted from the file. You may use the dictionary *letter_to_index* to convert the well row labels to array indices.
- calculates the mean OD for the wildtype (rows A and B) and the mutant (rows C and D) yeast cultures. You may use `numpy.mean()` for this.
- identifies and counts the number of non-viable wells, indicated by an OD measurement less than 0.05, for both wildtype and mutant yeast cultures.

Your program should print the NumPy array containing the OD values as well as the mean OD's and number of non-viable wells per yeast strain:

```
[[0.226 0.219 0.223 0.221 0.225 0.232]
 [0.217 0.026 0.237 0.225 0.228 0.236]
 [0.023 0.122 0.126 0.139 0.021 0.03 ]
 [0.113 0.114 0.129 0.147 0.131 0.025]]
wildtype
mean density: 0.20958333333333337
number of non-viable wells: 1
mutant
mean density: 0.09333333333333334
number of non-viable wells: 4
```

Note that the printed output of the mean and the non-viable well counts does not need to be in a specific format, but it should be clearly stated, what the printed values are and which strain they describe.

Your program should still work according to the same principle if the file *densities.csv* contained different measurements in a different order. You may assume that the microtiter plate has always 4 rows and 6 columns, wildtype in row A and B, mutant in row C and D, and that there is a measurement for every well.

For opening the file, your program should simply use the file name without path information (i.e., relative path).

Question 6

```
students = [['11-287-233', 'Justus', 'Meier', 'Justus', 'C'],
            ['08-609-029', 'Luna Meier', 'Meier', 'Luna', 'A'],
            ['11-237-784', 'Mhood', 'Hood', 'Maria', 'C'],
            ['08-169-553', 'Shea_Toby', 'Shea', 'Toby', 'B'],
            ['08-364-725', 'linC', 'Lin-Schmidt', 'Cason', 'B'],
            ['08-959-799', 'perez123', 'Perez Sanchez', 'Maria', 'B'],
            ['09-106-042', 'shayrom', 'Romero', 'Shayna', 'A']]

results = [['Justus',12,13,5], ['Luna Meier',10,11,14], ['Shea_Toby',7,6,2],
           ['Mhood',8,10,3], ['linC',13,10,15], ['perez123',2,3,0], ['shayrom',5,8,4]]
```

You are given two lists containing student information and assessment scores:

- *students* consists of lists with student number, username, last name, first name and group
- *results* contains lists with username and three scores from a recent online assessment.

Note that the student numbers and usernames are unique to the students.

Write a program that:

- a) finds the student with the highest total score (total score = sum of three scores). Print the highest total score plus the student's student number, last name, first name, username and group.
- b) summarizes the results per group and prints the following for each group (in ascending order, starting with group A):
 - for each group member, sorted by their individual total score, the total score followed by the student number, last name and first name.
 - the group's average total score.

The output of your program should look similar to this:


```
a)
best student: 38 points
08-364-725 Lin-Schmidt Cason linC B
b)
group A
17 09-106-042 Romero Shayna
35 08-609-029 Meier Luna
group average: 26.0
group B
5 08-959-799 Perez Sanchez Maria
15 08-169-553 Shea Toby
38 08-364-725 Lin-Schmidt Cason
group average: 19.333333333333332
group C
21 11-237-784 Hood Maria
30 11-287-233 Meier Justus
group average: 25.5
```

The program should still work according to the same principle with lists of the same structure as *students* and *results* but containing different students, a different number of students, a different number of groups (A-Z), or a different number of scores per student. You may assume that each student from the list *students* appears in the *results* list as well. You may also assume that there is only one highest total score.

Question 7

In the Polymerase Chain Reaction (PCR), primers are short DNA sequences that initiate the synthesis of a target DNA fragment. Effective primer design is crucial for the specificity and efficiency of the PCR.

One key factor in primer design is to avoid inverted repeats (IR) within the primer sequence. An IR consists of two segments that are reverse complements of each other, with a spacer sequence in between. The reverse complement of a sequence is the sequence read in the reverse direction and the nucleotides (nt) replaced by their complements (A->T, T->A, C->G, G->C).

For example, consider the IR . The reverse complement of the first 5 nt long segment, TAGCT, is AGCTA, which is the same as the last 5 nt in the sequence fragment. In between the example IR segments is a 3 nt long spacer, CGT.

Such IR's in a primer can lead to the formation of secondary structures such as hairpins, reducing the primer's efficacy.

```
primers = ['TCAGCTAGCTCGTAGCTACAGGC', 'CAGGTCACCTGTTAGACTCAGTCG',  
           'ACGAGTCGAGCGTAGTCTAC', 'TGAAGTGTGAATAGTACTCACGAG',  
           'CGCTCATGTATCATGAGCGCA', 'TGAAGTCGAATAGTTCGACTCA']
```

Write a program that evaluates a given list of potential primers, to identify IR's within them. Your program should specifically identify all IR's with

- a minimal IR segment length of 3 nt and
- a spacer of exactly 3 nt.

Ensure that the program correctly identifies the longest IR at a given location without including sub-segments as separate inverted repeats. For example, AGCTcgtAGCA is part of TAGCTcgtAGCAT and should not be counted as a second IR.

Your program should print each primer sequence together with any IR's found within that match the IR segment length and spacer criteria.

Note that your output does not need to be formatted as follows, but it should be clear which IR's and primer belong to one another. It is not needed to format the spacer with lower case letters, nor to position the IR's under the corresponding sequence in the primer.

```
Primer: TCAGCTAGCTCGTAGCTACAGGC  
        TAGCTcgtAGCTA  
  
Primer: CAGGTCACCTGTTAGACTCAGTCG  
        CAGGtcaCCTG      GACTcaGTC  
  
Primer: ACGAGTCGAGCGTAGTCTAC  
        GTAgtcTAC  
  
Primer: TGAAGTGTGAATAGTACTCACGAG  
  
Primer: CGCTCATGTATCATGAGCGCA  
        CGCTCATGtatCATGAGCG  
  
Primer: TGAAGTCGAATAGTTCGACTCA  
        AGTCGAAtagTTCGACT  
        AGTtcgACT
```

Your program should still work according to the same principle on a different list of primers.