**Master on Bioinformatics**

**Module 1. Programming in Bioinformatics**

Department of Computer Architecture and Operating Systems (CAOS)

**Exam 2019-20**



**Exercise 1.** A combination of two (or more) strings is a string created by interleaving the characters of the input strings. A combination is regular if a fixed number of characters are taken from each string at a time (always in the same fixed order).

1. **[0.5p]** Implement a function that, given two strings, outputs the regular combination of both input strings (one character at a time). **Note:** You cannot use the zip function.

def **combine**(a,b):

[...]

print(combine(*"AAA"*,*"GGGGG"*)) **# Prints "AGAGAGGG"**

print(combine(*"GGGGG"*,*"AAA"*)) **# Prints "GAGAGAGG"**

1. **[0.5p]** Implement the same function recursively (i.e. combining one character at a time).

def **combine\_recursive**(a,b):

[...]

1. **[1p]** Assuming both input strings have the same length, implement the same function using a divide and conquer approach. That is, subdividing recursively the task of combining two string into two problems of combining both halves of the strings.

def **combine\_divide\_\_conquer**(a,b):

[...]

print(combine\_divide\_\_conquer(*"AAAAA"*,*"GGGGG"*)) **# Prints "AGAGAGAGAG"**

print(combine\_divide\_\_conquer(*"GGGGG"*,*"AAAAA"*)) **# Prints "GAGAGAGAGA"**

print(combine\_divide\_\_conquer(*"ACGTA"*,*"GGCCA"*)) **# Prints "AGCGGCTCAA"**

1. **[1p]** Extend the first combine function (non-recursive) so it can combine a list of strings.

def **combine\_list**(slist):

[...]

print(combine\_list([*"AA"*,*"GGGGG"*,*"TTTTTT"*])) **# Prints "AGTAGTGTGTGTT"**

**Exercise 2.** Some FASTQ files encode unimportant chunks of the sequences using lower case letters (i.e. masked sequences).

1. **[1p]** Given a FASTQ file containing sequences with potentially masked chunks, implement a function that opens the FASTQ, reads the sequences, and counts how many bases have been masked.

|  |  |
| --- | --- |
| **> cat sample.fastq**  @SequenceA  acacgGACGGGtgactACGGTG  +  JHHHGHLJJJIIIKHHHHIHII  @SequenceB  ACACCAtggtgACAGCATACGACACgggg  +  HHHHIHIIHHHHIHIIHHHHIHHHHIHII | def **fastq\_count\_masked\_bases**(file\_path):  [...]    print(fastq\_count\_masked\_bases(*"sample.fastq"*)) **# Print 19** |

1. **[1p]** Given a FASTQ file containing sequences with potentially masked chunks, implement a function that generates another FASTQ file containing only the unmasked parts of the sequences.

|  |  |
| --- | --- |
| **> cat sample.fastq**  @SequenceA  acacgGACGGGtgactACGGTG  +  JHHHGHLJJJIIIKHHHHIHII  @SequenceB  ACACCAtggtgACAGCATACGACACgggg  +  HHHHIHIIHHHHIHIIHHHHIHHHHIHII | **> cat sample.masked.chunks.fastq**  @SequenceA.1  ACACG  +  JHHHG  @SequenceA.2  TGACT  +  IIKHH  @SequenceB.1  TGGTG  +  IIHHH  @SequenceB.2  GGGG  +  IHII |

def **fastq\_isolated\_masked\_chunks**(input\_fastq\_path,output\_fastq\_path):

[...]

fastq\_isolated\_masked\_chunks(*"sample.fastq"*,*"sample.masked.chunks.fastq"*)

**Exercise 3.** Given the SAM file ‘sample.sam’ and using ONLY linux command line tools.

1. **[0.5p]** Extract the alignment at position 500 (ignoring headers).

>

1. **[0.5p]** From the nucleotides that appear masked in the SAM file, find which one appears masked more times.

>

1. **[0.5p]** Implement a BASH script that splits the sam file into mapped and unmapped reads (e.g. ‘sample.mapped.sam’ and ‘sample.unmapped.sam’). Then, for each unmapped sequence, generates a separated FASTQ file (with the corresponding sequence). Finally, merges together all FASTQ files in a FASTA file called ‘unmapped.fasta’.

INPUT=$1

[...]

**Exercise 4:** We want to store the grades from all the master students and for all the assignments of the course. We need a data structure capable of representing the grades for all the submitted assignments. For that, we propose the following data-structure in python.

students\_grades = {

*"5214784"* : [(*"boxes"*,10.0),(*"fastx"*,8.0),(*"gameOfLive"*,10.0)],

*"2365489"* : [(*"boxes"*,8.0),(*"gameOfLive"*,10.0)],

*"1371629"* : [(*"gameOfLive"*,10.0),(*"fastx"*,3.0),(*"boxes"*,9.5)]

}

1. **[1p]** Given the student ID, implement a function that computes the final grade as the average of all grades obtained in each assignment.

def **student\_final\_grade**(students\_grades,student\_id):

[...]

student\_final\_grade(students\_grades,*"1371629"*) **# Prints "Student's final grade is: 7.5"**

student\_final\_grade(students\_grades,*"9999999"*) **# Prints "Student not found"**

1. **[1p]** Extend the previous function to break down the student's grades per assignment and include the general average grade obtained for each assignment.

def **student\_detailed\_final\_grade**(students\_grades,student\_id):

[...]

**Exercise 5.** An irregular combination of two strings is a string created by interleaving its characters, taking any number of characters from each string at a time.

1. **[1p]** Given three strings, implement a recursive function that checks if the third string is the result of any kind of combination of the first two.

def **is\_combination**(a,b,c):

[...]

print(is\_combination(*"AAA"*,*"GGGG"*,*"AAGAGGG"*)) **# True**

print(is\_combination(*"AAA"*,*"GGGG"*,*"AAGAGG"*)) **# False**

print(is\_combination(*"TATA"*,*"AGAG"*,*"TAGAAGTA"*)) **# True**

1. **[0.5p]** After solving the previous problem, we have noticed that many of the recursive calls are repeated. We would like to avoid repeating computations. Implement again the function using the memoization technique (or dynamic programming) in order to skip redundant computations

def **is\_combination\_memo**(a,b,c):

[...]