Scientific Programming 26/08/2019

QCB MASTER

Before you start

Please write one single python script for the lab part and one text file with the answers to the theoretical questions.
IMPORTANT: Add your name and ID (matricola) on top of the .py and text files!

Theory

Please write the solution in a text file.

Exercise 1:

```
Let M be a square matrix - a list containing n lists, each of them of size
n. Return the computational complexity of function fun() with respect to n:

def fun(M):
    for row in M:
      for element in row:
          print(sum([x for x in row if x != element]))
```

Practical part

Exercise 1 (Part A):

The file `marker_data.tsv` is a tab separated file containing the following information regarding SNPs tiled into a SNP-array:

```
SNP_Array_ID    Allele    Chr    Pos    Flanking_Sequence
AX-105198889    A/G    Chr07    33233374
GACCCTAGCTACGATACTGGAAATCTGATTAAATA[A/G]TTAATTAATAGCTCCAAAGATTTTCTCCGTCGGAA
AX-115198284    A/G    Chr08    21371449
TAGCAGGAATATCCTGTGCTTTTCCTTGGCGCGGC[A/G]TCATCCATTACGACAGTCGCTCTTCCCCATGGAAT
AX-115181796    A/C    Chr11    5200547
AAAAGGGTCGAGGAGGCCAATCACATAGTTAAGTA[G/T]GTGTACAACCAGTGTGAAGATAATGAAGGTGGTGG
...
```

As the header says, the first column is the SNP_Array identifier, the second is the Allele (i.e. the type of SNP), the third is the chromosome where the SNP is located, the fourth is the position of the SNP in the chromosome and the fifth is a string representation of the flanking regions of the SNP (i.e. the sequence that is before and after the SNP and the SNP itself represented in the format "[x/x]").

Write the following methods:

1. `loadTsv(filename)`: gets the filename of the tsv file and returns a pandas dataframe with all the loaded information;

2. `getFlankingRegion(data, SNPid, fwd = True)`: if SNPid is in data, this method should return a string that is the forward or reverse flanking sequence of the SNP depending on if fwd is True or False. The forward flanking sequence is the sequence that proceeds the SNP, the reverse flanking sequence is the sequence that follows the SNP.
Considering the entries shown above:

```
getFlankingRegion(data, "AX-105198889", fwd = True)
```

should return the string: "GACCCTAGCTACGATACTGGAAATCTGATTAAATA"
while
```
getFlankingRegion(data, "AX-105198889", fwd = False)
```

should return the string: "TTAATTAATAGCTCCAAAGATTTTCTCCGTCGGAA" and
```
getFlankingRegion(data, "fake_id_not_present", fwd = False)
```

should return the empty string and print a warning message (i.e. "SNP "fake_id_not_present" not found").

3. `addGCcontent(data)` that modifies the dataframe data adding two columns: 'FWDgc' and 'REVgc' that are the GC content of the forward and reverse flanking region. Given a sequence, the GC content can be computed as: Count(G + C)/Count(A + T + G + C) . This method does not need to return anything but rather just to modify the dataframe data.
Hint: you can write an additional function (computeGC) that given a string, computes the GC content;

4. `computestats(data)` gets the data structure loaded and prints the total number of SNPs, the total number of Chromosomes, and for each chromosome, the total number of SNPs and the mean GC content for the FWD and REV flanking regions;

5. `plotData(data, chromosome)` given the input data and a chromosome (**optional, if not specified all data should be considered**) plots the distribution of all the SNP types in that chromosome as a bar chart and a whisker plot of the GC contents of the FWD and REV flanking regions. Note that if chromosome is not in data, a warning should be given. See the examples below.

Calling

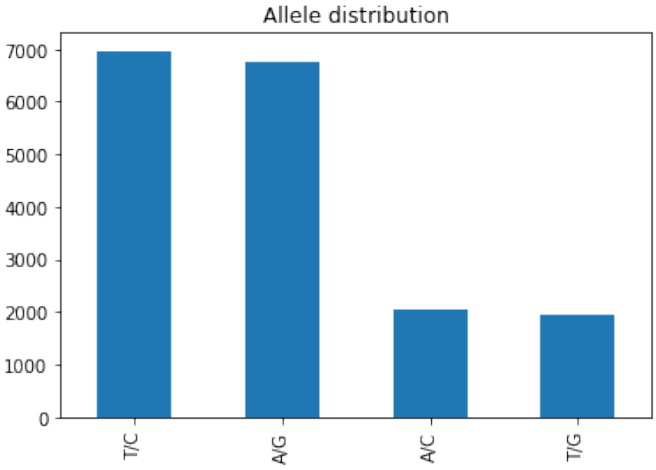
```
dataFile = "marker_data.tsv"
data = loadTSV(dataFile)

print(data.head(3))
print("")

snpid = "AX-115182233"
reg = getFlankingRegion(data,snpid, fwd = True)
if reg != None:
    print("FWD flanking region of {}:{}".format(snpid,reg))
reg = getFlankingRegion(data,snpid, fwd = False)
```

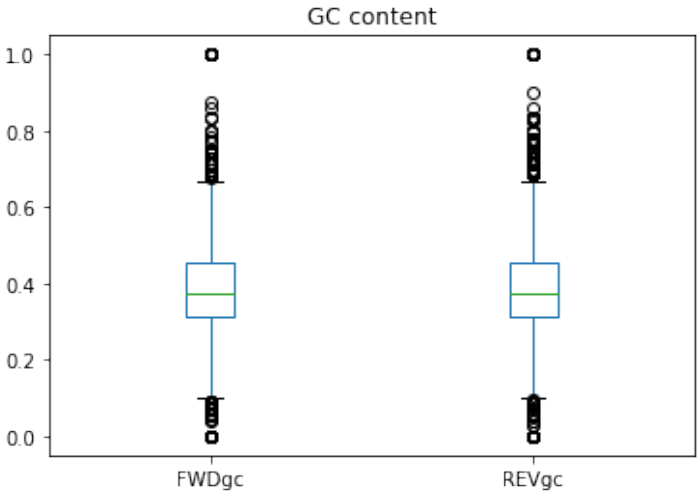
```
if reg != None:
    print("REV flanking region of {}:{}".format(snpid,reg))
print("")
snpid = "AX-123456789"
reg = getFlankingRegion(data,snpid, fwd = True)
if reg != None:
    print("FWD flanking region of {}:{}".format(snpid,reg))

print("")
addGCcontent(data)
print("Data now:")
print(data.head(3))


computestats(data)

plotData(data)

plotData(data,"Chr02")

plotData(data, "ChrXX")
```

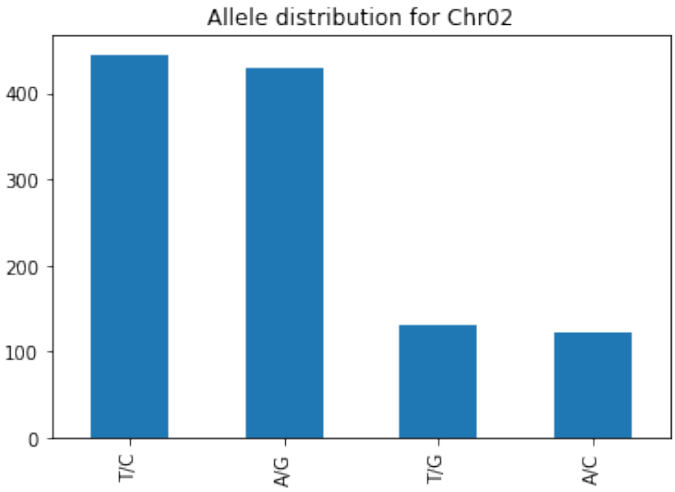
should return

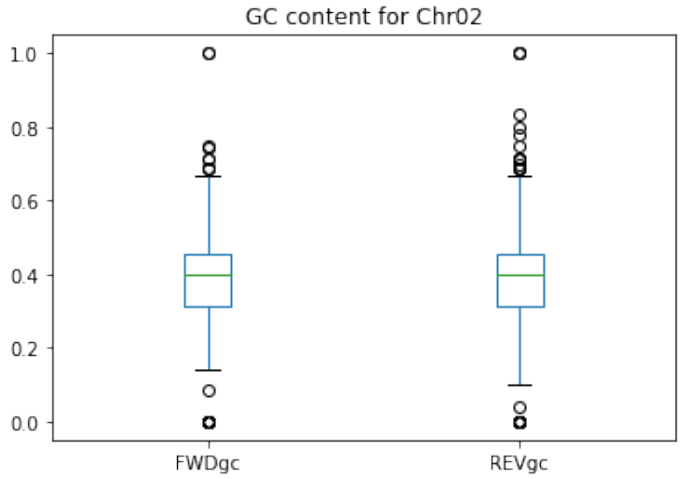
```
   SNP_Array_ID Allele    Chr       Pos  \
0  AX-105198889    A/G  Chr07  33233374
1  AX-115198284    A/G  Chr08  21371449
2  AX-115181796    A/C  Chr11   5200547

                                  Flanking_Sequence
0  GACCCTAGCTACGATACTGGAAATCTGATTAAATA[A/G]TTAATT...
1  TAGCAGGAATATCCTGTGCTTTTCCTTGGCGCGGC[A/G]TCATCC...
2  AAAAGGGTCGAGGAGGCCAATCACATAGTTAAGTA[G/T]GTGTAC...

FWD flanking region of AX-115182233:GGAAGTTGAATAAATAAACTTACAAGAATGTTAGC
REV flanking region of AX-115182233:TGGGCATAAATAATTCCACCGTCATCAGTATTAAC

SNP AX-123456789 not found

Data now:
   SNP_Array_ID Allele    Chr       Pos  \
0  AX-105198889    A/G  Chr07  33233374
1  AX-115198284    A/G  Chr08  21371449
2  AX-115181796    A/C  Chr11   5200547

                                  Flanking_Sequence     FWDgc     REVgc
0  GACCCTAGCTACGATACTGGAAATCTGATTAAATA[A/G]TTAATT...  0.371429  0.342857
1  TAGCAGGAATATCCTGTGCTTTTCCTTGGCGCGGC[A/G]TCATCC...  0.542857  0.485714
2  AAAAGGGTCGAGGAGGCCAATCACATAGTTAAGTA[G/T]GTGTAC...  0.428571  0.457143
Total number of SNPs: 17689

Total number of Chromosomes: 18

Chr00 has 127 SNPs.
    Mean FWDgc: 0.386
    Mean REVgc: 0.374
Chr01 has 770 SNPs.
    Mean FWDgc: 0.381
    Mean REVgc: 0.388
Chr02 has 1127 SNPs.
    Mean FWDgc: 0.388
```

```
    Mean REVgc: 0.387
Chr03 has 1094 SNPs.
    Mean FWDgc: 0.378
    Mean REVgc: 0.380
Chr04 has 914 SNPs.
    Mean FWDgc: 0.385
    Mean REVgc: 0.381
Chr05 has 1274 SNPs.
    Mean FWDgc: 0.399
    Mean REVgc: 0.395
Chr06 has 922 SNPs.
    Mean FWDgc: 0.386
    Mean REVgc: 0.384
Chr07 has 823 SNPs.
    Mean FWDgc: 0.381
    Mean REVgc: 0.383
Chr08 has 930 SNPs.
    Mean FWDgc: 0.390
    Mean REVgc: 0.384
Chr09 has 1020 SNPs.
    Mean FWDgc: 0.388
    Mean REVgc: 0.392
Chr10 has 1165 SNPs.
    Mean FWDgc: 0.386
    Mean REVgc: 0.386
Chr11 has 1196 SNPs.
    Mean FWDgc: 0.381
    Mean REVgc: 0.388
Chr12 has 1103 SNPs.
    Mean FWDgc: 0.385
    Mean REVgc: 0.381
Chr13 has 1036 SNPs.
    Mean FWDgc: 0.391
    Mean REVgc: 0.382
Chr14 has 914 SNPs.
    Mean FWDgc: 0.373
    Mean REVgc: 0.376
Chr15 has 1371 SNPs.
    Mean FWDgc: 0.382
    Mean REVgc: 0.377
Chr16 has 916 SNPs.
    Mean FWDgc: 0.385
    Mean REVgc: 0.388
Chr17 has 987 SNPs.
    Mean FWDgc: 0.387
    Mean REVgc: 0.385
```


Allele distribution
7000
6000
5000
4000
3000
2000
1000
0
T/C
A/G
A/C
T/G


GC content
1.0
0.8
0.6
0.4
0.2
0.0
FWDgc
REVgc


Allele distribution for Chr02
400
300
200
100
0
T/C
A/G
T/G
A/C




```
Warning. ChrXX is not present in the dataset
```

Exercise 2 (part B):

Priority queues are queues in which elements are extracted depending on their priority (i.e. elements with higher priority will be extracted first).
Write a class `PriorityQueue` implementing a simple priority queue and specify the methods below.
Hint: the priority queue can be a list (or deque) with elements being tuples like (priority, element).
NOTE:
For simplicity, we will assume that elements of the PriorityQueue are all of the same type.

1. `__init__(self)` that creates an empty PriorityQueue;

2. `__len__(self)` returns the number of elements in the PriorityQueue;

3. `isEmpty(self)` returns True if the PriorityQueue is empty, False otherwise;

4. `__str__(self)` that returns the string representation of the PriorityQueue with elements sorted by decreasing priority. If a PriorityQueue contains the elements [1,3,8,7,9,0] with corresponding priorities [1,12,4,7,3,7],

```
PQ = PriorityQueue()
PQ.enqueue_multi([1, 3, 8, 7, 9, 0], [1, 12, 4, 7, 3,7])
print(PQ)
```

should print the string:

```
PriorityQueue:
Element: 3    Priority: 12
Element: 7    Priority: 7
Element: 0    Priority: 7
Element: 8    Priority: 4
Element: 9    Priority: 3
Element: 1    Priority: 1
```

5. `enqueue(self,elem, prior)` enqueues the elment elem with priority prior (the length of the PriorityQueue is increased by one). The method does not return anything;

6.`dequeue(self)` removes and returns the element (or one of the elements) with the highest priority;

7.`top(self)` returns a tuple with list of ***ALL*** the elements with the highest priority and the single value that is the highest priority. The size of the PriorityQueue must not be changed by this method;

8.`enqueue_multi(self, elements, priorities)` where elements and priorities are lists of the same size respectively of elements and of priorities (that are assumed to be numbers) and adds all the elements with their corresponding priority to the priority queue (i.e. for each i in range(len(elements)), elements[i] has priority priorities[i]);

9.`merge(self, otherQueue, priority_offset=0)` merges the PrioriyQueue otherQueue into this PriorityQueue adding the specified priority_offset (which is an integer and is optional) to the priority of each element in otheQueue. The PriorityQueue otherQueue is empty at the end of the method (see examples below).

Calling

```
PQ = PriorityQueue()
print("Length: {}\nIs empty? {}".format(len(PQ), PQ.isEmpty()))
print(PQ)
print("Dequeueing an element...")
PQ.dequeue()
PQ.enqueue("PQ_el1", 11)
print("\nQueue now:")
print(PQ)
print("")
PQ.enqueue_multi(['PQ_el2','PQ_el3','PQ_el4'], [1,2])
PQ.enqueue_multi(['PQ_el2','PQ_el3','PQ_el4', 'some_text', 'AATA', 'other_txt',
'BB'], [4,11,4,7,72,11,1])
print("\n" + str(PQ))
for i in range(3):
    top_els = PQ.top()
    print("Next element(s) to dequeue is/are {} (priority:
{})".format(top_els[0],top_els[1]))
    el = PQ.dequeue()
    print("\there it is: {}".format(el))
print("\nThe queue now has {} elements".format(len(PQ)))
print(PQ)

secondQueue = PriorityQueue()
secondQueue.enqueue_multi(['elem1', 'elem2', 'elem3'], [10,9,7])
print("\nsecondQueue:\n{}".format(secondQueue))
PQ.merge(secondQueue, 2)
print("\nsecondQueue now (length:
{}):\n{}".format(len(secondQueue),secondQueue))
print("\nPQ now (length: {}):\n{}".format(len(PQ), PQ))
```

should return:

```
Length: 0
Is empty? True
PriorityQueue:
Dequeueing an element...
Warning: the priority queue is empty

Queue now:
```

```
PriorityQueue:
Element: PQ_el1    Priority: 11

Warning: cannot add to PriorityQueue as elements and priorities have different
sizes

PriorityQueue:
Element: AATA    Priority: 72
Element: other_txt    Priority: 11
Element: PQ_el3    Priority: 11
Element: PQ_el1    Priority: 11
Element: some_text    Priority: 7
Element: PQ_el4    Priority: 4
Element: PQ_el2    Priority: 4
Element: BB    Priority: 1
Next element(s) to dequeue is/are ['AATA'] (priority:72)
    here it is: AATA
Next element(s) to dequeue is/are ['PQ_el1', 'PQ_el3', 'other_txt']
(priority:11)
    here it is: PQ_el1
Next element(s) to dequeue is/are ['PQ_el3', 'other_txt'] (priority:11)
    here it is: PQ_el3

The queue now has 5 elements
PriorityQueue:
Element: other_txt    Priority: 11
Element: some_text    Priority: 7
Element: PQ_el4    Priority: 4
Element: PQ_el2    Priority: 4
Element: BB    Priority: 1

secondQueue:
PriorityQueue:
Element: elem1    Priority: 10
Element: elem2    Priority: 9
Element: elem3    Priority: 7

secondQueue now (length: 0):
PriorityQueue:

PQ now (length: 8):
PriorityQueue:
Element: elem1    Priority: 12
Element: other_txt    Priority: 11
Element: elem2    Priority: 11
Element: elem3    Priority: 9
Element: some_text    Priority: 7
Element: PQ_el4    Priority: 4
Element: PQ_el2    Priority: 4
Element: BB    Priority: 1
```