Yubsis Challenge

Generate a python script that lists all the repeats equal or longer than 9 bases in double strand dna sequences

Karim Mezhoud

2020-08-11

Contents

1 Challenge Description

A repeat, in the context of this test, is defined by a pair of locations on the dna with fully identical sequence. A repeat can be between 2 locations on the same strand or between one strand and the opposite strand. A repeat has a precise length, which is the maximal length of identity: if you extend it by one more base, the 2 sequences diverge.

As said, a "repeat" is between 2 locations. If the same sequence pattern of 9 bases is found at 3 (or more) different locations, this should result in as many different repeats as the number of pairs.

Write a python script that applies to a dna sequence of any length, containing only the 4 conventional bases, given in lower or uppercase. The result has to be a python list of all the found repeats of length >= 9, sorted by descending length. Each repeat (each pair) should be listed once, with its maximal length.

Most importantly, your agorithm should perform exactly what has been defined above. Care should be taken to fully describe the repeats in the final list : read once again how "repeat" has been defined. This will be tested on a set of defined sequences of 50 - 10 000 bp.

Of second priority, take care of the quality of your code. In particular, you can try to optimize the algorithm for clarity and performance (time of execution will be tested on random sequences of different length, potentially up to 1 million bp).

2 The goal of the challenge

- Find all repeat motifs of DNA that have more or equal to 9 bases. **Take Care about Overlapping**
- There are 4 conventional bases: `A,T,C,G`. The script must work with **upper and lower cases**.
- The DNA is a double strand helix. If we look the repeats in one strand, automatically, we can deduce repeats in the second strand.
- The input of the script is a simple DNA sequence that can reach 1.000,000 bp.
- The output is a list of motifs with their lengths, sorted from longest to shortest.

2.1 Subset the challenge to multiple subchallenges

- During computing the script need to screen the DNA `bp` by `bp` forming motifs with length longer or equal to `9bp+i`.
- For each motif (`9bp+i`) the script has to screen the DNA `bp` by `bp` and count all repeated motif.
- Testing all motifs with length from 9 to unknown length from DNA with 1.000,000 bp could take a long time. The idea is to limit iterations only to longest repeated motif.
  1. Identify the longest motifs with at less 2 repeats: the function returns the motifs and his length N.
  2. Iterate from 9 to N and count repeats.

2.2 Set Python environment

This is specific to Rstudio computing environments (IDE).

```
# Set python environment and version in RStudio ;-)
reticulate::use_python("/Users/Mezhoud/anaconda3/bin/python", required = TRUE)
reticulate::py_config()
```

```
## python:         /Users/Mezhoud/anaconda3/bin/python
## libpython:      /Users/Mezhoud/anaconda3/lib/libpython3.7m.dylib
## pythonhome:     /Users/Mezhoud/anaconda3:/Users/Mezhoud/anaconda3
## version:        3.7.5 (default, Oct 25 2019, 10:52:18)  [Clang 4.0.1 (tags/RELEASE_401/final)]
## numpy:          /Users/Mezhoud/anaconda3/lib/python3.7/site-packages/numpy
## numpy_version:  1.18.1
##
## NOTE: Python version was forced by use_python function
```

3 Generate DNA sequence

Here we define a function that generates a random DNA sequence using only `TACG`. As said in te description there is no gaps `-`.

3.1 DNA WITHOUT repeating motifs

```
import random

def generate_DNA(N, alphabet=['ACGT']):
    return ''.join([random.choice(alphabet) for i in range(N)])

random.seed(10)
dna = generate_DNA(100)

print(dna)
```

```
## ACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTA
```

- But we have to include some repeats in our DNA sequence.
- We will mix upper an lower case to better view the repeats.
- Also, the script must work with upper and lower cases.

3.2 Generate DNA WITH repeating motifs

- We defined 3 motifs repeats with 4, 10, and 12 bp.
- We defined lower case motif to better visualize the motifs in DNA, And take care about upper and lower input DNA.
- We randomly sample from `ATCG` and the 3 motifs.

```
import random

repeat4 = 'atca'
repeat10 =  'tcatg' * 2
repeat12 = 'tatg' * 3

def generate_DNA(N, alphabet=['A','C','G','T',repeat4,repeat10, repeat12]):
    return ''.join([random.choice(alphabet) for i in range(N)])

random.seed(10)
dna = generate_DNA(20)

print(dna)
```

```
## atcaATTatcaACTtatgtatgtatgTtatgtatgtatgGtcatgtcatgtatgtatgtatgCAatcaTGA
```

```
print(len(dna))
```

```
## 71
```

- As you can see the length of the DNA is **greater than 20bp**, because the function considers that `repeat4`, `repeat10` and `repeat12` as a single bp.

4 Find the Longest motif with at less 2 repeats: avoid overlapping motifs

4.1 Algorithm

- To avoid overlapping during DNA screening, the script must **subtract or omit** the motif that starts at position `i` and ends at position `j`. Where `i = range(1 to n)` and `j = range(i+1 to n)`.
- We can define `motif(i,j)` as an object that store the **matching** and **non-overlapping** substring at position `(i to j)`.
- If `substring[i-1] == substring[j-1] && (j-i) > motif(i-1,j-1)` # **the second condition for avoid overlapping**
  - then `motif(i, j) = motif(i-1, j-1) + 1`,
- else `motif(i, j) = 0`

4.2 `longestMotif` Function

```python
# Returns the longest  motif (non-overlapping)
def longestMotif(dna):
    # unsure if DNA is in upper Cases
    dna = dna.upper()
    n = len(dna)
    motif = [[0 for x in range(n + 1)]
                for y in range(n + 1)]

    longest_motif = "" # To store result
    longest_motif_length = 0 # To store length of result

    # building table in bottom-up manner
    index = 0
    for i in range(1, n + 1):
        for j in range(i + 1, n + 1):

            # (j-i) > motif[i-1][j-1] to remove
            # overlapping
            if (dna[i - 1] == dna[j - 1] and
                motif[i - 1][j - 1] < (j - i)):
                motif[i][j] = motif[i - 1][j - 1] + 1

                # updating maximum length of the
                # motif and updating the finishing
                # index of the suffix
                if (motif[i][j] > longest_motif_length):
                    longest_motif_length = motif[i][j]
                    index = max(i, index)

            else:
                motif[i][j] = 0

    # If we have non-empty motif, then insert
    # all bps from first bp to
    # last bpof motif
    if (longest_motif_length > 0):
        for i in range(index - longest_motif_length + 1,
                                    index + 1):
            longest_motif = longest_motif + dna[i - 1]

    return longest_motif

# Driver Code
# if __name__ == "__main__":
#     random.seed(10)
#     dna = generate_DNA(1000).upper()
#     return(longestMotif(dna))
```

4.2.1 Longest motif counter

- This function allows us to compare the counting with (our function) and without (`count()` method) overlapping.

```
## Count occurence with overlapping
def count_substring(string,sub_string):

    # Insure if string and sub_string are in upper cases
    string = string.upper()
    sub_string = sub_string.upper()

    l=len(sub_string)
    count=0
    for i in range(len(string)-len(sub_string)+1):
        if(string[i:i+len(sub_string)] == sub_string ):
            count+=1
    return count
```

4.2.2 Run an example of `longestMotif` function

```
import timeit
import random

random.seed(10)
start = timeit.default_timer()
dna = generate_DNA(1000).upper()
stop = timeit.default_timer()
print("> Generate DNA with ", len(dna), " bp",  round(stop - start, 3), "s")
```

```
## > Generate DNA with  4189  bp 0.066 s
```

```
start = timeit.default_timer()
longest_motif = longestMotif(dna)
stop = timeit.default_timer()
print('> Find longest motifs: ', round(stop - start, 3), "s")
```

```
## > Find longest motifs:  7.014 s
```

```
print("> Longest motif: " + str(longest_motif))
```

```
## > Longest motif: ATCATATGTATGTATGTATGTATGTATGTATGTATGTATG
```

```
print("> Number of repeated longest motif WITHOUT overlapping: " + str(dna.upper().count(longest_motif)
```

```
## > Number of repeated longest motif WITHOUT overlapping: 2 With: 40 bp
```

```
print("> Number of repeated longest motif WITH overlapping: " + str(count_substring(dna,longest_motif))
```

```
## > Number of repeated longest motif WITH overlapping: 2
```

- Well, At this step we can limit iterations to count all possible repeated motifs to maximum length of 41bp. This may save time of the processing.

5 Find all repeating motifs (motifs >= 9bp, motifs <= longest)

```
from collections import Counter as cntr
import pandas as pd

random.seed(10)
start = timeit.default_timer()
dna = generate_DNA(10000).upper()
stop = timeit.default_timer()
print("> Generate DNA with ", len(dna), " bp",  round(stop - start, 3), "s")
```

```
## > Generate DNA with  42821  bp 0.05 s
```

```
start = timeit.default_timer()
substrs = cntr()

maximum = 41 # longest motif with at less two repeats
minimum = 9  # the minimum length of motif is 9 bp

for n in range(minimum,maximum): # look from 9 to 41 bp
  for i in range(0,len(dna)-n):  # loop for counter
    substrs[dna[i:i+n]] += 1
  motifs = substrs#.most_common

stop = timeit.default_timer()
print('> Get all repeated substring: ', round(stop - start, 3), "s")

# Convert to dict
```

```
## > Get all repeated substring:  2.391 s
```

```
d = dict(motifs)
start = timeit.default_timer()
# filter only repeats at less twice
# sort by motif length decreasing
d = dict((k, v) for k, v in sorted(d.items(), key=lambda item: len(item[0]),reverse = True) if v >= 2)
# Sort by value decreasing
#d = dict((k, v) for k, v in sorted(d.items(), key=lambda item: item[1],reverse = True) if v >= 2)
stop = timeit.default_timer()
print('> Filter and sort motifs with at less two repeats: ', round(stop - start, 3), "s")
```

```
#print(d)
```

```
## > Filter and sort motifs with at less two repeats:  0.796 s
```

```
start = timeit.default_timer()

ls = []
# reformat output
#for key, value in d.items():
#    print(key, len(str(key)) ,value)

for key, value in d.items():
    temp = [key,len(str(key)),value]
    ls.append(temp)

stop = timeit.default_timer()
print('> Reformating dictionary to list, include length of motifs (bp): ', round(stop - start,3), "s")
```

```
## > Reformating dictionary to list, include length of motifs (bp):  0.255 s
```

```
print("> At all there are: ", len(ls), " motifs, with at less two repeats.")
```

```
## > At all there are:  86420  motifs, with at less two repeats.
```

```
print("> Print motifs, length , repeats: ", *ls[1:20], sep = "\n")
```

```
## > Print motifs, length , repeats:
## ['TTATGTATGTATGTCATGTCATGTATGTATGTATGTCATG', 40, 3]
## ['TATGTATGTATGTCATGTCATGTATGTATGTATGTCATGT', 40, 5]
## ['ATGTATGTATGTCATGTCATGTATGTATGTATGTCATGTC', 40, 5]
## ['TGTATGTATGTCATGTCATGTATGTATGTATGTCATGTCA', 40, 5]
## ['GTATGTATGTCATGTCATGTATGTATGTATGTCATGTCAT', 40, 5]
## ['TATGTATGTCATGTCATGTATGTATGTATGTCATGTCATG', 40, 5]
## ['ATGTATGTCATGTCATGTATGTATGTATGTCATGTCATGT', 40, 3]
## ['TGTATGTCATGTCATGTATGTATGTATGTCATGTCATGTA', 40, 2]
## ['GTATGTCATGTCATGTATGTATGTATGTCATGTCATGTAT', 40, 2]
## ['TATGTCATGTCATGTATGTATGTATGTCATGTCATGTATG', 40, 2]
## ['ATGTCATGTCATGTATGTATGTATGTCATGTCATGTATGT', 40, 2]
## ['TGTCATGTCATGTATGTATGTATGTCATGTCATGTATGTA', 40, 2]
## ['GTCATGTCATGTATGTATGTATGTCATGTCATGTATGTAT', 40, 3]
## ['TCATGTCATGTATGTATGTATGTCATGTCATGTATGTATG', 40, 4]
## ['CATGTCATGTATGTATGTATGTCATGTCATGTATGTATGT', 40, 4]
## ['ATGTCATGTATGTATGTATGTCATGTCATGTATGTATGTA', 40, 4]
## ['TGTCATGTATGTATGTATGTCATGTCATGTATGTATGTAT', 40, 4]
## ['GTCATGTATGTATGTATGTCATGTCATGTATGTATGTATG', 40, 4]
## ['ATGTATGTATGTATGTCATGTCATGTATGTATGTATGATC', 40, 2]
```

5.1 Wanted output format

We can change the output format as we like. Here the output is a list of all motifs from the longest to the shortest with their lengths.

```
ls = []
# reformat output
#for key, value in d.items():
```

```
#    print(key, len(str(key)) ,value)
start = timeit.default_timer()
for key, value in d.items():
    temp = [key,len(str(key))]
    ls.append(temp)

stop = timeit.default_timer()
print('> Reformating dictionary to list, include length of motifs (bp): ', round(stop - start,3), "s")
```

```
## > Reformating dictionary to list, include length of motifs (bp):  0.233 s
```

```
print("> At all there are: ", len(ls), " motifs, with at less two repeats.")
```

```
## > At all there are:  86420  motifs, with at less two repeats.
```

```
print("> Print head of longest motifs, length: ", *ls[0:20], sep = "\n")
```

```
## > Print head of longest motifs, length:
## ['GTTATGTATGTATGTCATGTCATGTATGTATGTATGTCAT', 40]
## ['TTATGTATGTATGTCATGTCATGTATGTATGTATGTCATG', 40]
## ['TATGTATGTATGTCATGTCATGTATGTATGTATGTCATGT', 40]
## ['ATGTATGTATGTCATGTCATGTATGTATGTATGTCATGTC', 40]
## ['TGTATGTATGTCATGTCATGTATGTATGTATGTCATGTCA', 40]
## ['GTATGTATGTCATGTCATGTATGTATGTATGTCATGTCAT', 40]
## ['TATGTATGTCATGTCATGTATGTATGTATGTCATGTCATG', 40]
## ['ATGTATGTCATGTCATGTATGTATGTATGTCATGTCATGT', 40]
## ['TGTATGTCATGTCATGTATGTATGTATGTCATGTCATGTA', 40]
## ['GTATGTCATGTCATGTATGTATGTATGTCATGTCATGTAT', 40]
## ['TATGTCATGTCATGTATGTATGTATGTCATGTCATGTATG', 40]
## ['ATGTCATGTCATGTATGTATGTATGTCATGTCATGTATGT', 40]
## ['TGTCATGTCATGTATGTATGTATGTCATGTCATGTATGTA', 40]
## ['GTCATGTCATGTATGTATGTATGTCATGTCATGTATGTAT', 40]
## ['TCATGTCATGTATGTATGTATGTCATGTCATGTATGTATG', 40]
## ['CATGTCATGTATGTATGTATGTCATGTCATGTATGTATGT', 40]
## ['ATGTCATGTATGTATGTATGTCATGTCATGTATGTATGTA', 40]
## ['TGTCATGTATGTATGTATGTCATGTCATGTATGTATGTAT', 40]
## ['GTCATGTATGTATGTATGTCATGTCATGTATGTATGTATG', 40]
## ['ATGTATGTATGTATGTCATGTCATGTATGTATGTATGATC', 40]
```

```
print("> Print tail of shortest motifs, length: ", *ls[-20:-1], sep = "\n")
```

```
## > Print tail of shortest motifs, length:
## ['CAATCACGA', 9]
## ['GTATGTTGA', 9]
## ['AAGATCAAT', 9]
## ['CAAATCAAA', 9]
## ['AATCATTTC', 9]
## ['TCATGAGTT', 9]
## ['TCATGAAAT', 9]
## ['CATGAAATC', 9]
## ['CATATCAAG', 9]
## ['ATGGGCTCA', 9]
## ['TGGGCTCAT', 9]
## ['GTATGGAGG', 9]
## ['TATGGAGGT', 9]
## ['TCATGTCTT', 9]
## ['CATGTCTTA', 9]
## ['TGTACATCA', 9]
## ['ATCACCGAT', 9]
## ['TCACCGATC', 9]
## ['CACCGATCA', 9]
```

5.2 allRepeatesMotifs Function

```
from collections import Counter as cntr
import pandas as pd

def allRepeatedMotifs(dna, minimum, maximum):

    # insure that DNA in upper cases
    dna = dna.upper()
    maximum = maximum # longest motif with at less two repeats
    minimum = minimum  # the minimum length of motif is 9 bp
    start = timeit.default_timer()
    substrs = cntr()
    for n in range(minimum,maximum): # look from 9 to 41 bp
      for i in range(0,len(dna)-n):  # loop for counter
        substrs[dna[i:i+n]] += 1
    motifs = substrs#.most_common

    stop = timeit.default_timer()
    print('> Get all repeated substring: ', round(stop - start, 3), "s")

    # Convert to dict
    d = dict(motifs)
    start = timeit.default_timer()
    # filter only repeats at less twice
    # sort by motif length decreasing
    d = dict((k, v) for k, v in sorted(d.items(), key=lambda item: len(item[0]),reverse = True) if v >=
    # Sort by value decreasing
    #d = dict((k, v) for k, v in sorted(d.items(), key=lambda item: item[1],reverse = True) if v >= 2)
    stop = timeit.default_timer()
    print('> Filter and sort motifs with at less two repeats: ', round(stop - start, 3), "s")

    start = timeit.default_timer()

    ls = []
    # reformat output
    #for key, value in d.items():
    #    print(key, len(str(key)) ,value)

    for key, value in d.items():
        temp = [key,len(str(key))]
        ls.append(temp)

    stop = timeit.default_timer()
    print('> Reformating dictionary to list, include length of motifs (bp): ', round(stop - start,3), "
    print("> At all there are: ", len(ls), " motifs, with at less two repeats.")
    #print("> Print motifs, length , repeats: ", *ls[1:20], sep = "\n")
    return(ls)
```

5.2.1 Run an example of allRepeatedMotifs Function

```
random.seed(10)
start = timeit.default_timer()
dna = generate_DNA(10000).upper()
stop = timeit.default_timer()
print("> Generate DNA with ", len(dna), " bp",  round(stop - start, 3), "s")
```

```
## > Generate DNA with  42821  bp 0.08 s
```

```
ls = allRepeatedMotifs(dna, 9,41)
```

```
## > Get all repeated substring:  2.068 s
## > Filter and sort motifs with at less two repeats:  0.753 s
## > Reformating dictionary to list, include length of motifs (bp):  0.132 s
## > At all there are:  86420  motifs, with at less two repeats.
```

```
print( *ls[0:20], sep = "\n")
```

```
## ['GTTATGTATGTATGTCATGTCATGTATGTATGTATGTCAT', 40]
## ['TTATGTATGTATGTCATGTCATGTATGTATGTATGTCATG', 40]
## ['TATGTATGTATGTCATGTCATGTATGTATGTATGTCATGT', 40]
## ['ATGTATGTATGTCATGTCATGTATGTATGTATGTCATGTC', 40]
## ['TGTATGTATGTCATGTCATGTATGTATGTATGTCATGTCA', 40]
## ['GTATGTATGTCATGTCATGTATGTATGTATGTCATGTCAT', 40]
## ['TATGTATGTCATGTCATGTATGTATGTATGTCATGTCATG', 40]
## ['ATGTATGTCATGTCATGTATGTATGTATGTCATGTCATGT', 40]
## ['TGTATGTCATGTCATGTATGTATGTATGTCATGTCATGTA', 40]
## ['GTATGTCATGTCATGTATGTATGTATGTCATGTCATGTAT', 40]
## ['TATGTCATGTCATGTATGTATGTATGTCATGTCATGTATG', 40]
## ['ATGTCATGTCATGTATGTATGTATGTCATGTCATGTATGT', 40]
## ['TGTCATGTCATGTATGTATGTATGTCATGTCATGTATGTA', 40]
## ['GTCATGTCATGTATGTATGTATGTCATGTCATGTATGTAT', 40]
## ['TCATGTCATGTATGTATGTATGTCATGTCATGTATGTATG', 40]
## ['CATGTCATGTATGTATGTATGTCATGTCATGTATGTATGT', 40]
## ['ATGTCATGTATGTATGTATGTCATGTCATGTATGTATGTA', 40]
## ['TGTCATGTATGTATGTATGTCATGTCATGTATGTATGTAT', 40]
## ['GTCATGTATGTATGTATGTCATGTCATGTATGTATGTATG', 40]
## ['ATGTATGTATGTATGTCATGTCATGTATGTATGTATGATC', 40]
```

6 Implementation of the main function

6.1 yubsisChallenge Function

```
def yubsisChallenge(dna, minimum = 9):

  longest_motif = longestMotif(dna)
  print("> Longest motif: " + str(longest_motif))
  print("> Number of repeated longest motif WITHOUT overlapping: " + str(dna.upper().count(longest_moti
  print("> Number of repeated longest motif WITH overlapping: " + str(count_substring(dna,longest_motif

  motifs = allRepeatedMotifs(dna=dna ,minimum=minimum, maximum=len(longest_motif)+1)
  return(motifs)
```

6.2 Run and Example

```
import timeit
import random
from collections import Counter as cntr
import pandas as pd

random.seed(10)
start = timeit.default_timer()
dna = generate_DNA(1000)
stop = timeit.default_timer()
print("> Generate DNA with ", len(dna), " bp",  round(stop - start, 3), "s")
```

```
## > Generate DNA with  4189  bp 0.052 s
```

```
final_list = yubsisChallenge(dna, minimum = 9)
```

```
## > Longest motif: ATCATATGTATGTATGTATGTATGTATGTATGTATGTATG
## > Number of repeated longest motif WITHOUT overlapping: 2 With: 40 bp
## > Number of repeated longest motif WITH overlapping: 2
## > Get all repeated substring:  0.161 s
## > Filter and sort motifs with at less two repeats:  0.066 s
## > Reformating dictionary to list, include length of motifs (bp):  0.008 s
## > At all there are:  7135  motifs, with at less two repeats.
```

```
print(*final_list[0:20], sep = "\n")
```

```
## ['ATCATATGTATGTATGTATGTATGTATGTATGTATGTATG', 40]
## ['ATCATATGTATGTATGTATGTATGTATGTATGTATGTAT', 39]
## ['TCATATGTATGTATGTATGTATGTATGTATGTATGTATG', 39]
## ['GTATGTATGTATGTCATGTCATGTTATGTATGTATGCA', 38]
## ['ATCATATGTATGTATGTATGTATGTATGTATGTATGTA', 38]
## ['TCATATGTATGTATGTATGTATGTATGTATGTATGTAT', 38]
## ['CATATGTATGTATGTATGTATGTATGTATGTATGTATG', 38]
## ['TATGTATGTATGTTATGTATGTATGTATGTATGTATG', 37]
## ['GTATGTATGTATGTCATGTCATGTTATGTATGTATGC', 37]
## ['TATGTATGTATGTCATGTCATGTTATGTATGTATGCA', 37]
## ['ATCATATGTATGTATGTATGTATGTATGTATGTATGT', 37]
## ['TCATATGTATGTATGTATGTATGTATGTATGTATGTA', 37]
## ['CATATGTATGTATGTATGTATGTATGTATGTATGTAT', 37]
## ['ATATGTATGTATGTATGTATGTATGTATGTATGTATG', 37]
## ['TATGTATGTATGTCATGTCATGTATGTATGTATGTC', 36]
## ['TATGTATGTATGTTATGTATGTATGTATGTATGTAT', 36]
## ['ATGTATGTATGTTATGTATGTATGTATGTATGTATG', 36]
## ['GTATGTATGTATGTCATGTCATGTTATGTATGTATG', 36]
## ['TATGTATGTATGTCATGTCATGTTATGTATGTATGC', 36]
## ['ATGTATGTATGTCATGTCATGTTATGTATGTATGCA', 36]
```

```
print(*final_list[-20:-1], sep = "\n")
```

```
## ['ATGTATGAA', 9]
## ['TGTATGAAT', 9]
## ['GTATGAATC', 9]
## ['TATGAATCA', 9]
## ['TGTATGCAC', 9]
## ['ACCGTATGT', 9]
## ['CCGTATGTA', 9]
## ['TCAATCAAT', 9]
## ['CAATCAATC', 9]
## ['ATGAATCAA', 9]
## ['CATATCACT', 9]
## ['TGAATCAAT', 9]
## ['GAATCAATC', 9]
## ['GTATGTCTA', 9]
## ['TATGTCTAT', 9]
## ['ATGTCTATG', 9]
## ['TGTCTATGT', 9]
## ['GTCTATGTA', 9]
## ['CATCTCATG', 9]
```

6.3 Straight way to run script

All python functions are hosted in github at this link.

6.3.1 Sourcing from github

```
# sourcing scripts from github
import requests
exec(requests.get('https://raw.githubusercontent.com/kmezhoud/Yubsis/master/src/generate_DNA.py').text)
exec(requests.get('https://raw.githubusercontent.com/kmezhoud/Yubsis/master/src/longestMotif.py').text)
exec(requests.get('https://raw.githubusercontent.com/kmezhoud/Yubsis/master/src/count_substring.py').te
exec(requests.get('https://raw.githubusercontent.com/kmezhoud/Yubsis/master/src/allRepeatedMotifs.py').
exec(requests.get('https://raw.githubusercontent.com/kmezhoud/Yubsis/master/src/yubsisChallenge.py').te
```

6.3.2 Generate DNA sequence if needed

```
import timeit

random.seed(10)
start = timeit.default_timer()
dna = generate_DNA(1000)
stop = timeit.default_timer()
print("> Generate DNA with ", len(dna), " bp",  round(stop - start, 3), "s")
```

```
## > Generate DNA with  4189  bp 0.08 s
```

6.3.3 Test the script with your DNA sequences

```
# dna is a sequence of `ATCG`  in upper or lower or mixed cases
final_list = yubsisChallenge(dna, minimum = 9)
```

```
## > Longest motif: ATCATATGTATGTATGTATGTATGTATGTATGTATGTATG
## > Number of repeated longest motif WITHOUT overlapping: 2 With: 40 bp
## > Number of repeated longest motif WITH overlapping: 2
## > Get all repeated substring:  0.129 s
## > Filter and sort motifs with at less two repeats:  0.061 s
## > Reformating dictionary to list, include length of motifs (bp):  0.005 s
## > At all there are:  7135  motifs, with at less two repeats.
```

```
print(*final_list[0:20], sep = "\n")
```

```
## ['ATCATATGTATGTATGTATGTATGTATGTATGTATGTATG', 40]
## ['ATCATATGTATGTATGTATGTATGTATGTATGTATGTAT', 39]
## ['TCATATGTATGTATGTATGTATGTATGTATGTATGTATG', 39]
## ['GTATGTATGTATGTCATGTCATGTTATGTATGTATGCA', 38]
## ['ATCATATGTATGTATGTATGTATGTATGTATGTATGTA', 38]
## ['TCATATGTATGTATGTATGTATGTATGTATGTATGTAT', 38]
## ['CATATGTATGTATGTATGTATGTATGTATGTATGTATG', 38]
## ['TATGTATGTATGTTATGTATGTATGTATGTATGTATG', 37]
## ['GTATGTATGTATGTCATGTCATGTTATGTATGTATGC', 37]
## ['TATGTATGTATGTCATGTCATGTTATGTATGTATGCA', 37]
## ['ATCATATGTATGTATGTATGTATGTATGTATGTATGT', 37]
## ['TCATATGTATGTATGTATGTATGTATGTATGTATGTA', 37]
## ['CATATGTATGTATGTATGTATGTATGTATGTATGTAT', 37]
## ['ATATGTATGTATGTATGTATGTATGTATGTATGTATG', 37]
## ['TATGTATGTATGTCATGTCATGTATGTATGTATGTC', 36]
## ['TATGTATGTATGTTATGTATGTATGTATGTATGTAT', 36]
## ['ATGTATGTATGTTATGTATGTATGTATGTATGTATG', 36]
## ['GTATGTATGTATGTCATGTCATGTTATGTATGTATG', 36]
## ['TATGTATGTATGTCATGTCATGTTATGTATGTATGC', 36]
## ['ATGTATGTATGTCATGTCATGTTATGTATGTATGCA', 36]
```

```
print(*final_list[-20:-1], sep = "\n")
```

```
## ['ATGTATGAA', 9]
## ['TGTATGAAT', 9]
## ['GTATGAATC', 9]
## ['TATGAATCA', 9]
## ['TGTATGCAC', 9]
## ['ACCGTATGT', 9]
## ['CCGTATGTA', 9]
## ['TCAATCAAT', 9]
## ['CAATCAATC', 9]
## ['ATGAATCAA', 9]
## ['CATATCACT', 9]
## ['TGAATCAAT', 9]
## ['GAATCAATC', 9]
## ['GTATGTCTA', 9]
## ['TATGTCTAT', 9]
## ['ATGTCTATG', 9]
## ['TGTCTATGT', 9]
## ['GTCTATGTA', 9]
## ['CATCTCATG', 9]
```