MultiThreading

November 29, 2019

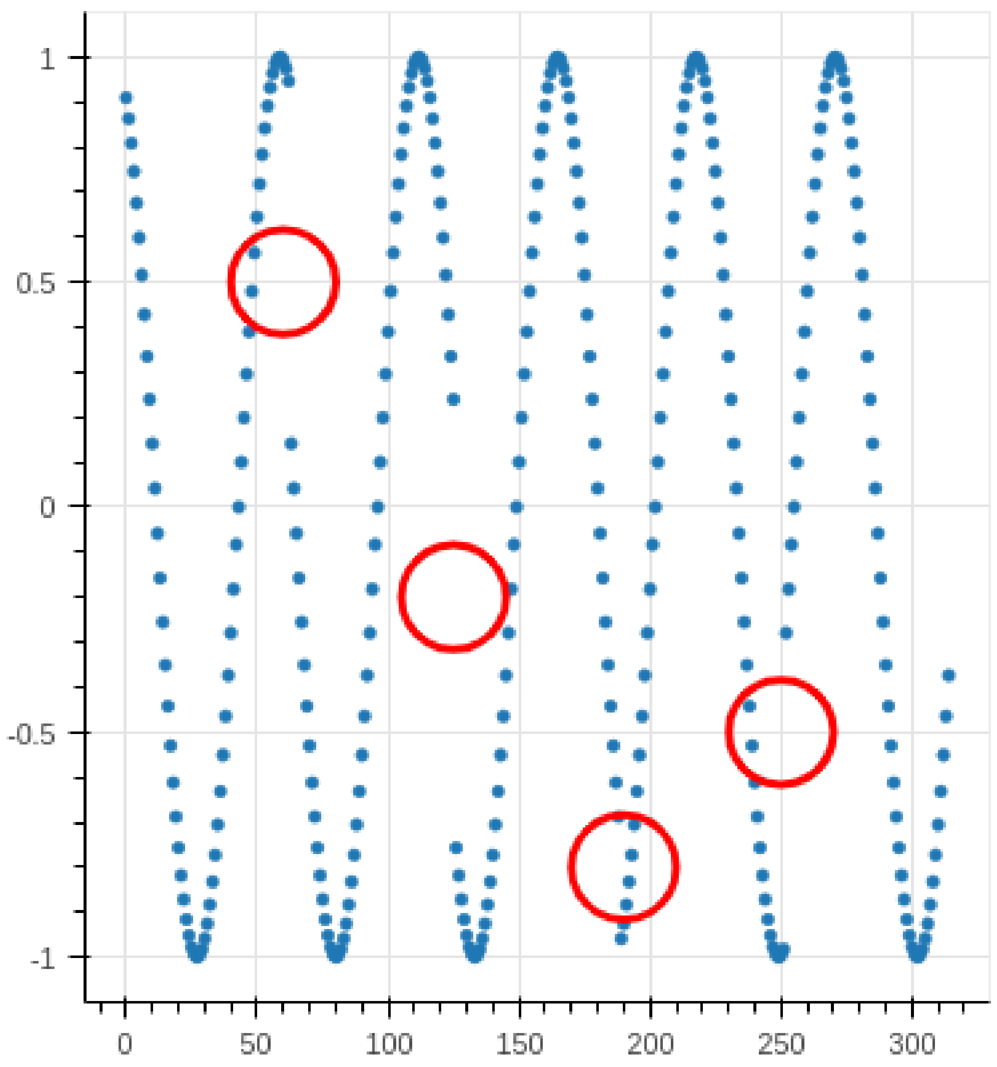
1 Why Multithread?

Let's start with an example. Imagine we want to acquire a signal and process the data aquired. So we've a Python script like the following one:

```
while True:
    raw_data=acquire()
    processed_data=process(raw_data)
```

The function *acquire()* collect the input and the function processes the data.

If we try to plot the data collected we may found something like in this picture



You can see how some samples are lost by the *acquire* function. This happens because the acquisition isn't continuos: after the first cycle of the signal is a collected some preprocessing is done, this operations take time and cause the delay in the acquisition

1.1 Multi-threading with the *threading* module

To avoid this kind of issues we can use the multithreading capabilities of Python with the module *threading*. There are also other tools to achieve the same results but we will just use this one. To create our custom thread class we want to use the code below:

[3]:
```
import threading
class MyThread(threading.Thread):
    def __init__(self, threadID):
        threading.Thread.__init__(self)
        #Setup thread
        self.threadID = threadID
    def run(self):
        pass
        #Do something
```

Inside the *init* method we will the define the parameters of the thread while inside the *run* method we will specify the instructions that the trade will execute

2 Exercises

2.1 Exercise 1

In this exercise we want to test if we can improve the the speed of a little piece of code by using multi-threading. We want to execute a GET request to a list of websites in two ways:

1. using a simple for cycle
2. using separate threads for each request

We want to measure the execution time of both of the solutions to check which of them is the fastest

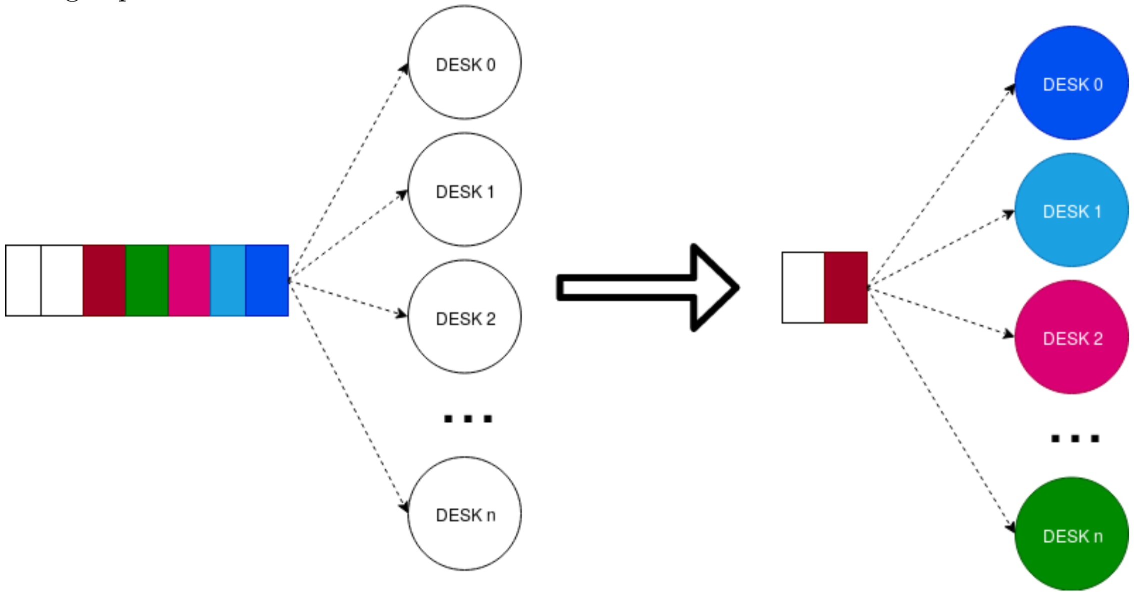
Tips and tricks

1. You can use this list of website below
2. To measure the execution time you can use this code

[26]:
```
#Code to avaluate execution time
import time
#Websites list
website_list= ["http://yahoo.com",
               "http://google.com",
               "http://amazon.com",
               "http://ibm.com",
               "http://apple.com",
               "https://www.microsoft.com",
               "https://www.youtube.com/" ,
               "https://www.polito.it/" ,
               "http://www.wikipedia.org",
               "https://www.reddit.com/",
               "https://www.adobe.com/",
               "https://wordpress.org/",
               "https://github.com/",
               "https://www.google.com/maps/"]
start = time.time()
#Code to measure
stop=time.time()
execution_time=stop-start
```

2.2 Exercise 2

In this exercise we want to simulate the queue for that we can find in a lot of supermarket nowadays, a single queue that than sends the customer to the first cash desk available.



We want to create a list of customers that has their own serve-time that is random number (you can use numpy.random to generate random numbers with particular distributions) and a list of cash desk. We want to analyze the result of the simulation so we want to know the total time needed to serve all the customers, the average time and the number of customer served by each cash desk

2.3 Exercise 3

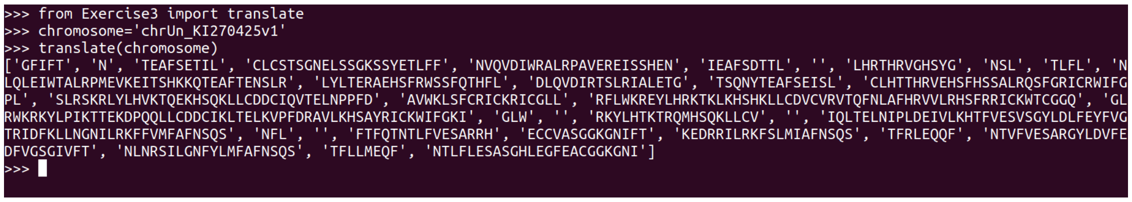
We want to create a script that uses multithreading to convert a dna sequence to the correspondent protein. A DNA sequence is composed by a concatenation of ***nucletides*** that can have four possible values :'A', 'T', 'C', 'G'. Each group of 3 nucleotides is called ***codon*** and which indicicates a particular ***aminoacid***. A sequence of aminoacid correnspond to a protein (inside the file *usefulthings.txt* you can find the dictionary to convert a condon to a aminoacid. Inside the file *chromosomes.json* there is a list of chromosomes. We want to create a python script that uses multithread that do what follows:

1. Perform a get request to this api
   https://api.genome.ucsc.edu/getData/sequence?genome=hg38;chrom=
2. Convert each codon to the corresponding aminoacid
3. Return the sequence that define the protein

Pay attention that some codon in the table correspnd to 'STOP', this indicates the finish of the previous protein and the start of a new one. To start do no use all the list of chromosomes but go step by step.

As you did for the first exercise yo can check whethe the code with multithreading is faster than a simple for-loop

Below you can see an example of the result



```
>>> from Exercise3 import translate
>>> chromosome='chrUn_KI270425v1'
>>> translate(chromosome)
['GFIFT', 'N', 'TEAFSETIL', 'CLCSTSGNELSSGKSSYETLFF', 'NVQVDIWRALRPAVEREISSHEN', 'IEAFSDTTL', '', 'LHRTHRVGHSYG', 'NSL', 'TLFL', 'N
LQLEIWTALRPMEVKEITSHKKQTEAFTENSLR', 'LYLTERAEHSFRWSSFQTHFL', 'DLQVDIRTSLRIALETG', 'TSQNYTEAFSEISL', 'CLHTTHRVEHSFHSSALRQSFGRICRWIFG
PL', 'SLRSKRLYLHVKTQEKHSQKLLCDDCIQVTELNPPFD', 'AVWKLSFCRICKRICGLL', 'RFLWKREYLHRKTKLKHSHKLLCDVCVRVTQFNLAFHRVVLRHSFRRICKWTCGGQ', 'GL
RWKRKYLPIKTTEKDPQQLLCDDCIKLTELKVPFDRAVLKHSAYRICKWIFGKI', 'GLW', '', 'RKYLHTKTRQMHSQKLLCV', '', 'IQLTELNIPLDEIVLKHTFVESVSGYLDLFEYFVG
TRIDFKLLNGNILRKFFVMFAFNSQS', 'NFL', '', 'FTFQTNTLFVESARRH', 'ECCVASGGKGNIFT', 'KEDRRILRKFSLMIAFNSQS', 'TFRLEQQF', 'NTVFVESARGYLDVFE
DFVGSGIVFT', 'NLNRSILGNFYLMFAFNSQS', 'TFLLMEQF', 'NTLFLESASGHLEGFEACGGKGNI']
>>> 
```