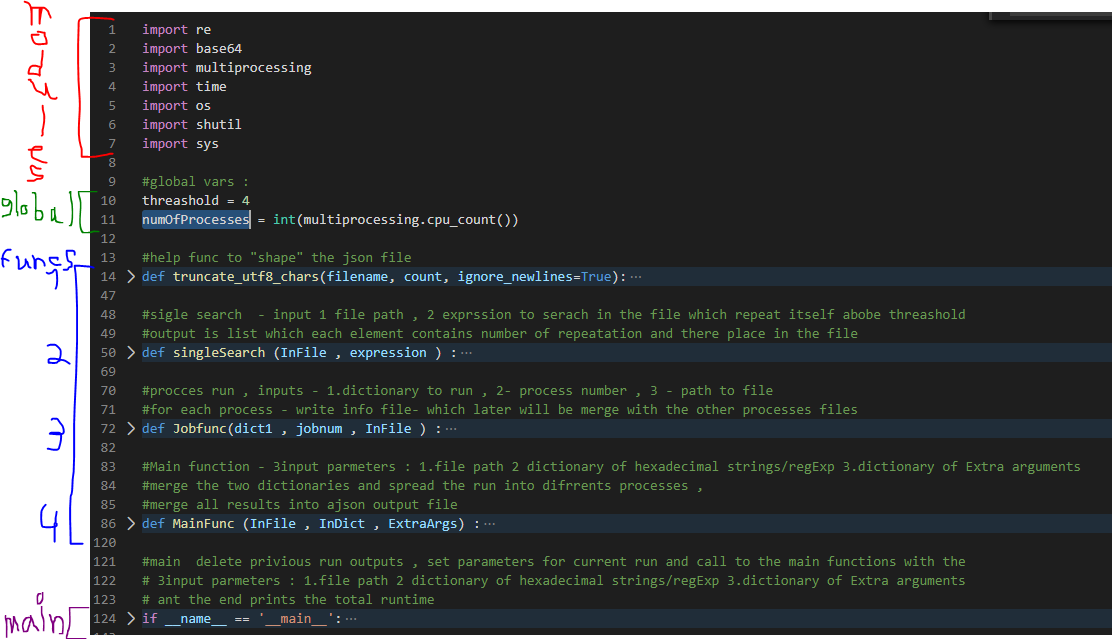
**matching patterns in binary files**

Target:

Find repeating patterns in a big img file,

Pattern to find is given from two dictionaries

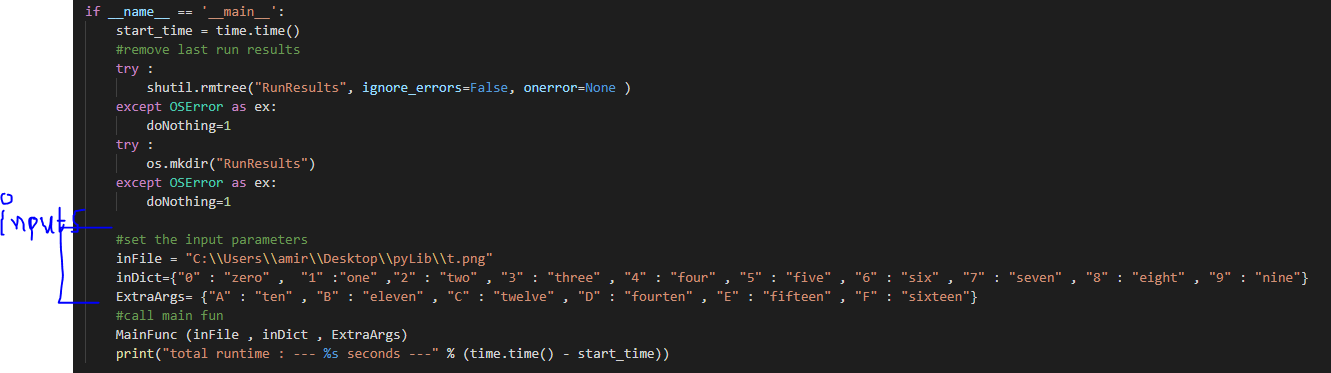
The codes signatures:



Main:

Inputs:

1. Path to the img file (or any file - it supports all files formats)
2. Dictionary1 - List/Map of hexadecimal strings/regular expressions
3. Dictionary2 - Extra arguments



The main deletes previous run outputs, gets the 2 dictionaries inputs - string of hexs (in my it single hexs letter but it can be any "hexs-string” combination with as any letters). we call the main function, and print the total time

4 Functions:

Function4 – MainFunc

main function - receive the 3 inputs from the main and make his magic – merge the two dictionaries and spread equal amount of hexs-string repetitions to each process. the function creates number of process as in the global var, and for each makes a parallel search at the same file with different “hexs-string” repetitions.

at the end of the process we merge the results from each process and put it in the Json out file

Function3 - JobFunc

Function to operate each process, receive dictionary which contain “hexs-string” to search their repetition in the input file (which is also an input parameter). Each “hexs string” from the dictionary will be send to a single search function (the next function). the output will be written to tmp file which in the end will be merged with the other process tmp files into the final Json file.

Function2 – singleSearch

Get as input file and single “hexs-string” expression – to search for. Output is list with all the places and number of the repetitions

How do we find the repetitions? (algorithm)?

1. Convert input file into a hex string with Base 64 module (was the most efficient module for me)
2. Replace all the “hexs-string” expression received to spaces
3. Count the repetitions for each space, and if we have some space in sequence in a row above threshold, I will add the number and the location into output

Function1 – truncate\_utf8\_chars

Not really important – helps deletes unwanted data from the final json output file

Modules:

I used only built in libraries (as modules) at python 3.6

Ref for built in libs:

<https://docs.python.org/3.6/py-modindex.html#cap-s>

global:

two global variables :

1. Threshold – number of repeated patterns, for example if we looking for ‘A’:

fsdfasdfsd**AAAAA**sdfdvgf – we have 5 sequences above the threshold 4.

fsdfasdfsd**AAA**sdfdvgf – 3 will be skipped because it is bellowing the threshold 4.

(the threshold is threshold +, meaning in this example 4 will not be skipped)

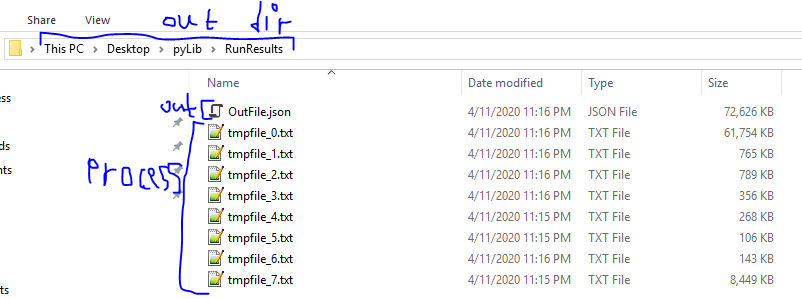
1. numOfProcesses – number of running processes. The solution is multiprocessing based and as such I gave the ability to control the number of it – I set it to the number of cores in the system (for example in my computer is 8)

outputs:

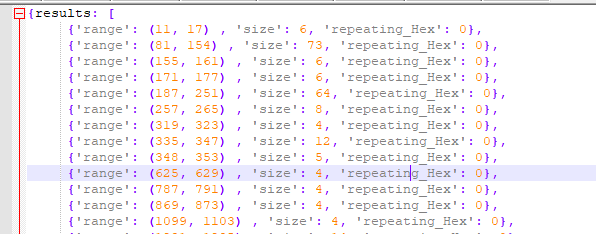
all outputs are will output to “RunResults” library in the running environment, and will include:

1. OutFile json file - with the results
2. tmpFile\_<process\_num> - tmp results from each process

example (for 8 cores):



Json file:



I didn’t delete the tnpFiles in order to check and debug if I will want to, all of them will be deleted in the next run

Remarks:

* The code can run both on Win and Linux OS (include Ubuntu 18.04)
* I choose the convert the file into Hexadecimal, and the dictionaries keys will be the pattern I will search – also in hexadecimal, same in the given example:

5D00008000': 'lzma',  
  ' 27051956': 'uImage',

…..

* I preferred the inputs parameters will be in the code and not part of the python file run (for example python file.py <var1> <var2> <var3>….) because I don’t think it will be convenient for the dictionary inputs (inputs 2,3)
* I used multiprocessing to speed up the process, If I wouldn’t have the limitation of python built in modules, I would have use spark framework, which suit exactly for this case (multi searches on text file)
* I checked it over pretty big files and got pretty good results about 70 seconds for 80 MG file, with 16 searches, matches and 8 cores in my computer. The time will be change by the following parameters:

1. Number of cores (the more of the, less times)
2. Number of “hexs-string” to searches (the more of them more search time)
3. Number of matches (the more of them, the more run time)

* I didn’t check nor try to find edge cases, I focused in reducing the run time