Home Work #3

Implement the following in Python 3 (3.9 recommended) using no other library besides `re`.

§1. `get_market_cap(filename)`

Write a Python function `get_market_cap(filename)` that reads the input `filename`, which contains information regarding the market capitalization values of various companies at different times in history (for example, see `market-cap-history.txt`). It must make heavy use of the `re` library to extract each company name, the amount and unit of its value, and the year it was recorded. It should return the information in a dictionary with the structure given below.

Some of the vagaries of natural language are beyond `re`. Thankfully, there are patterns: all sentences are in separate lines; the first sentence can be ignored; 'now' and 'today' both mean the current year; 'same time' refers to the year in the previous sentence; 'around' can be taken to be synonymous with 'in'; and the absence of any temporal word can be presumed to indicate the current year. Although there are only trillion and million in the example, we cannot rule out billions.

The following summarizes the information to be gleaned from the example file and the structure of the corresponding dictionary:

Name	*Amount*	*Unit*	*Year*
Dutch East India Company	8.28	trillion	1637
Mississippi Company	6.8	trillion	1720
South Sea Company	4.5	trillion	1720
Saudi Aramco	1.89	trillion	2022
Apple has	1.3	trillion	2022
PetroChina	1.24	trillion	2007
Microsoft	1.2	trillion	2022
Standard Oil	1	trillion	1911
Alphabet	931	million	2022
Amazon	886	million	2022

Structure of dictionary (corresponding to example):

```
{'company': 'Dutch East India Company':
         {'value': 8.28, 'unit': trillion, 'year': 1637},
   ...,
 'company': 'Amazon': {'value': 886, 'unit': million, 'year': 2022}
}
```

§2. `get_digit_patterns(filename)`

Write a Python function `get_digit_patterns` that reads the input `filename`, which is is a text file containing a long sequence of digits, and then uses regular expressions to detect subsequences of the form *ddx*...*ddx* for each of the ten decimal digits *d* ($d \in [0..9]$), where *x* is any decimal digit. The pattern in a subsequence should not be repeated within it.

It would print out those patterns in lines of the form

```
found 001 in 359:975
...
found 998 in 78:768
```

indicating that the slice 359:975 of the digit sequence is of the form 001...001, ..., and the slice 78:768 is of the form 998...998. It also means that the slice 362:972 does not contain the pattern 001...001,... the slice 81:765 does not contain the pattern 998...998.

The detected patterns should also be returned in a dictionary with the following structure (by example):

```
{'d': 0: {'x': 1, start: 359, end:975},
  ...
 'd': 9: {'x': 8, start: 78, end:768}
}
```

The digit sequence in the input file would be broken into lines purely for convenience (for example, `digits_of_pi.txt` contains a sequence of 1000 digits in 20 lines).

- The input text files are presumed to be in the current directory.
- Use raw strings and multi-line pattern specifications relying on the `re.VERBOSE` mode. Add appropriate comments within the pattern definition.
- Submit a Jupyter Notebook whose cells I can run in order. (Control the number of cells.)
- The notebook should start with your name and summarize what it achieves (and list requirements that are not fully operational).

  Each function should start with a docstring describing it.
- The last cell should contain

```
CMP_DICT = get_market_cap('market_cap_history.txt')

DGT_DICT = get_digit_patterns(digits_of_pi.txt')
```