Python for Data Scientists Assignment 2

Problem 1

- Simplest solution → simple data structure (list, tuple,..)
- Efficient solution → dictionary

{(0, 'c'): ['class', 'case', 'course'], (1, 'l'): ['class'], (2, 'a'): ['class'], (3, 's'): ['class’], …(2, 'r'): ['word'], (3, 'd'): ['word']}

Problem 1

```
words = ['class', 'case', 'course', 'dictionary', 'java', 'list', 'program',
'python', 'tuple', 'word']

def dictionary(l) :
    d = {}
    for word in l :
        for i,c in enumerate(word) :
            d[i,c] = d.get ((i,c), []) + [word]
    return d

def words_letter_position (d, letter, position) :
    return d.get((position, letter), [] )

d = dictionary(words)
r = words_letter_position(d, 'a', 1)
print ("résultat=",r)
```

Problem 1

- It is more efficient to use a dictionary for lookup of elements because it takes less time to traverse in the dictionary than a list.

- More time is needed to fetch a single element in a list than that of a dictionary because dictionary uses hash table for implementing the arrangement.

Problem 2

Efficient solution:

- Define a function which take a string as an input and return a string that contains all of the letters in order
- Use a dictionary to save all anagrams

```
if signature(word) not in d:
    d[t] = [word]
else:
    d[t].append(word)
```

- Print all anagrams :

```
for v in d.values():
    if len(v) > 1:
        print(len(v), v)
```

Problem 2

Efficient solution:

- Print all anagrams in order: make a list of (length, word pairs), use sort

```
t = []
for v in d.values():
    if len(v) > 1:
        t.append((len(v), v))
t.sort()
```

Problem 2

Efficient solution:

- Add a filter length to the dictionary : select only the words in d that have n letters

```
res = {}
for word, anagrams in d.items():
    if len(word) == n:
        res[word] = anagrams
return res
```

Time: 1.0149905681610107