Python-Initiation2

October 3, 2019

1 Initiation à python (suite)

On va cette fois s'intéresser à "résoudre" un problème et présenter quelques outils/constructions supplémentaires permettant de le résoudre.

1.1 Problème

On souhaite compter le nombre d'occurence des caractères alphabétiques (non accentués) d'un texte contenu dans un fichier.

Pour se faire, nous allons partir d'un programme permettant d'afficher individuellement chacun des caractères d'un fichier :

```
Bonjour,

Ca va bien ?
Demain je mange a la cantine, il y aura des haricots.

A demain, JB
```

Voici le programme (et son exécution) :

[2]:
```
fichier = open('contenu.txt','r')
for ligne in fichier:
    for mot in ligne.split():
        for lettre in mot:
            print("J'ai trouvé la lettre ",lettre)
fichier.close()
```

```
J'ai trouvé la lettre  B
J'ai trouvé la lettre  o
J'ai trouvé la lettre  n
J'ai trouvé la lettre  j
J'ai trouvé la lettre  o
J'ai trouvé la lettre  u
J'ai trouvé la lettre  r
J'ai trouvé la lettre  ,
J'ai trouvé la lettre  C
J'ai trouvé la lettre  a
J'ai trouvé la lettre  v
```

```
J'ai trouvé la lettre  a
J'ai trouvé la lettre  b
J'ai trouvé la lettre  i
J'ai trouvé la lettre  e
J'ai trouvé la lettre  n
J'ai trouvé la lettre  ?
J'ai trouvé la lettre  D
J'ai trouvé la lettre  e
J'ai trouvé la lettre  m
J'ai trouvé la lettre  a
J'ai trouvé la lettre  i
J'ai trouvé la lettre  n
J'ai trouvé la lettre  j
J'ai trouvé la lettre  e
J'ai trouvé la lettre  m
J'ai trouvé la lettre  a
J'ai trouvé la lettre  n
J'ai trouvé la lettre  g
J'ai trouvé la lettre  e
J'ai trouvé la lettre  a
J'ai trouvé la lettre  l
J'ai trouvé la lettre  a
J'ai trouvé la lettre  c
J'ai trouvé la lettre  a
J'ai trouvé la lettre  n
J'ai trouvé la lettre  t
J'ai trouvé la lettre  i
J'ai trouvé la lettre  n
J'ai trouvé la lettre  e
J'ai trouvé la lettre  ,
J'ai trouvé la lettre  i
J'ai trouvé la lettre  l
J'ai trouvé la lettre  y
J'ai trouvé la lettre  a
J'ai trouvé la lettre  u
J'ai trouvé la lettre  r
J'ai trouvé la lettre  a
J'ai trouvé la lettre  d
J'ai trouvé la lettre  e
J'ai trouvé la lettre  s
J'ai trouvé la lettre  h
J'ai trouvé la lettre  a
J'ai trouvé la lettre  r
J'ai trouvé la lettre  i
J'ai trouvé la lettre  c
J'ai trouvé la lettre  o
J'ai trouvé la lettre  t
J'ai trouvé la lettre  s
```

```
J'ai trouvé la lettre  .
J'ai trouvé la lettre  A
J'ai trouvé la lettre  d
J'ai trouvé la lettre  e
J'ai trouvé la lettre  m
J'ai trouvé la lettre  a
J'ai trouvé la lettre  i
J'ai trouvé la lettre  n
J'ai trouvé la lettre  ,
J'ai trouvé la lettre  J
J'ai trouvé la lettre  B
```

L'idée est donc de transformer le code de la boucle la plus interne (la plus imbriquée) de sorte que ce ne soit plus un affichage mais un «comptage» (on veut compter les occurences des lettres et non les afficher).

1.2 L'alternative

Le premier problème est que l'on souhaite ne compter que les caractères alphabétiques et ignorer les signe de ponctuation, etc. Il faut donc opérer un filtre de sorte que si le caractère capturé corrspondant à une lettre on fait quelque chose, sinon on l'ignore. Ainsi seule les caractères alphabétiques seront comptabilisés. On a donc à faire à une conditionnelle *si une-certaine-condition alors faire-quelque-chose sinon faire-autre-chose*. La forme syntaxique à employer est :

```
if une-certaine-condition:
    faire-quelque-chose
else:
    faire-autre-chose
```

et dans le cas où il n'y à rien à faire si la condition est fausse, l'instruction peut-être réduite en :

```
if une-certaine-condition:
    faire-quelque-chose
```

Comment peut-on écrire *une-certaine-condition* ? Il y a un langage disponible permettant de les exprimer toutes mais que nous ne détaillerons pas ici (il sera à découvrir au fur et à mesure). La condition qui nous intéresse pour le problème est : *le-caractère-est-alphabétique*, comment cela s'écrit-il en python ? Pour y répondre nous allons devoir utiliser un module…

1.3 Les modules python

Python est un éco-système, c'est-à-dire que ce n'est pas seulement un langage de programmation mais qu'il est accompagné d'un ensemble de choses très utiles. En effet, de nombreux calculs et traitements sont récurrents dans les programmes qu'on écrit : afficher quelque chose (`print()`), connaître la longueur d'une liste (`len()`), découper une chaîne de caractères en mots (`.split()`), etc. Comme ces fonctionnalités sont nombreuses, elles sont classées à la manière dont on classe les ouvrages dans une bibliothèque (le rayon des ouvrages de mathématique, celui de psychologie, de physique, etc). En Python les fonctions sont rangées dans des rayons que l'on appelle des modules.

Une question se pose naturellement «Comment puis-je savoir qu'une fonction qui fait telle et telle

chose existe ?». C'est le niveau d'expertise du programmeur qui entre en jeu. Avec le temps, on sait que la probabilité que telle fonction existe ou pas dans un module Python. Par exemple, si l'on souhaite calculer la moyenne de valeurs données, est-ce que je dois le faire moi-même par un petit programme ou existe t-il une fonction donnée pour le faire ? La réponse est : Python est très utilisé par des tas de gens qui utilisent Python pour faire des statistiques; il est donc fort probable qu'il existe un module dédié à ça et contenant une fonction de calcul de moyenne. Pour le vérifier allons dans la documentation Python. On y trouve un index général des modules dans lequel il y a statistics! Devinez ce qu'on y trouve! Comme par hasard une des premières fonctions mentionnées s'appelle `mean()`, je vous laisse déviner ce qu'elle permet de faire (lisez).

Maintenant, il n'y a plus qu'un conseil à vous donner : une fois les éléments syntaxiques de base du langage acquis (il y en a assez peu en fait) votre expertise Python augmentera :

- par la pratique de résolution de problèmes de plus en plus compliqués (vous acquerrez des compétences algorithmiques)
- par la consultation de la documentation afin de connaître de mieux en mieux certains modules (vous acquerrez des compétences Python)

Il n'y a pas d'autre recette pour devenir un expert Python.

Maintenant, y a t-il une fonction permettent de tester si un caractère est une lettre ? Très probablement. On imagine assez bien que c'est un besoin si récurrent qu'il y a sans doute une solution toute prête! Pour cela nous allons consulter une autre partie de la documentation Python : Référence de la Bibliohtèque Standard. C'est tellement commun comme besoin que ce n'est même pas dans un module mais dans le module standard qui ne recquiert donc rien de particulier pour son utilisation. Dans ce module on trouve une entrée Le Type Séquence de Texte (rappelez-vous une chaîne de caractères est en fait une séquence). Là, balladez-vous jusqu'à la collection de fonction `is`*quelque-chose*, l'une d'entre elle est `isalpha` et donc la documentation est assez explicite (peut-être même trop, alors oubliez la partie finale sur les caractères Unicode - et oui les documentations ont plusieurs niveaux de lecture selon l'expertise). Bref, on a exactement ce qui nous faut :

[5]:
```
fichier = open('contenu.txt','r')
for ligne in fichier:
    for mot in ligne.split():
        for lettre in mot:
            if lettre.isalpha():
                print("J'ai trouvé la lettre ",lettre)
fichier.close()
```

```
J'ai trouvé la lettre  B
J'ai trouvé la lettre  o
J'ai trouvé la lettre  n
J'ai trouvé la lettre  j
J'ai trouvé la lettre  o
J'ai trouvé la lettre  u
J'ai trouvé la lettre  r
J'ai trouvé la lettre  C
J'ai trouvé la lettre  a
J'ai trouvé la lettre  v
J'ai trouvé la lettre  a
```

```
J'ai trouvé la lettre  b
J'ai trouvé la lettre  i
J'ai trouvé la lettre  e
J'ai trouvé la lettre  n
J'ai trouvé la lettre  D
J'ai trouvé la lettre  e
J'ai trouvé la lettre  m
J'ai trouvé la lettre  a
J'ai trouvé la lettre  i
J'ai trouvé la lettre  n
J'ai trouvé la lettre  j
J'ai trouvé la lettre  e
J'ai trouvé la lettre  m
J'ai trouvé la lettre  a
J'ai trouvé la lettre  n
J'ai trouvé la lettre  g
J'ai trouvé la lettre  e
J'ai trouvé la lettre  a
J'ai trouvé la lettre  l
J'ai trouvé la lettre  a
J'ai trouvé la lettre  c
J'ai trouvé la lettre  a
J'ai trouvé la lettre  n
J'ai trouvé la lettre  t
J'ai trouvé la lettre  i
J'ai trouvé la lettre  n
J'ai trouvé la lettre  e
J'ai trouvé la lettre  i
J'ai trouvé la lettre  l
J'ai trouvé la lettre  y
J'ai trouvé la lettre  a
J'ai trouvé la lettre  u
J'ai trouvé la lettre  r
J'ai trouvé la lettre  a
J'ai trouvé la lettre  d
J'ai trouvé la lettre  e
J'ai trouvé la lettre  s
J'ai trouvé la lettre  h
J'ai trouvé la lettre  a
J'ai trouvé la lettre  r
J'ai trouvé la lettre  i
J'ai trouvé la lettre  c
J'ai trouvé la lettre  o
J'ai trouvé la lettre  t
J'ai trouvé la lettre  s
J'ai trouvé la lettre  A
J'ai trouvé la lettre  d
J'ai trouvé la lettre  e
```

```
J'ai trouvé la lettre  m
J'ai trouvé la lettre  a
J'ai trouvé la lettre  i
J'ai trouvé la lettre  n
J'ai trouvé la lettre  J
J'ai trouvé la lettre  B
```

Hourra! Cette fois nous n'avons plus que les lettres!

Y a t'il d'autres possibilités ? Oui, nous aurions pu utiliser par exemple :

`if lettre in string.ascii_letters:`

qui nécessite d'utiliser le module `string`et donc l'usage de la directive :

`import string`

En effet, sauf pour le module standard, tous les autres modules doivent être «importés» si l'on souhaite les utiliser. Sinon Python serait obligé d'importer tous les modules dans tous les programmes (inutile! Pourquoi aller chercher des livres de mathématiques si vous n'en faites pas ?). Le programme s'écrirait alors :

[7]:
```
import string
fichier = open('contenu.txt','r')
for ligne in fichier:
    for mot in ligne.split():
        for lettre in mot:
            if lettre in string.ascii_letters:
                print("J'ai trouvé la lettre ",lettre)
fichier.close()
```

```
J'ai trouvé la lettre  B
J'ai trouvé la lettre  o
J'ai trouvé la lettre  n
J'ai trouvé la lettre  j
J'ai trouvé la lettre  o
J'ai trouvé la lettre  u
J'ai trouvé la lettre  r
J'ai trouvé la lettre  C
J'ai trouvé la lettre  a
J'ai trouvé la lettre  v
J'ai trouvé la lettre  a
J'ai trouvé la lettre  b
J'ai trouvé la lettre  i
J'ai trouvé la lettre  e
J'ai trouvé la lettre  n
J'ai trouvé la lettre  D
J'ai trouvé la lettre  e
J'ai trouvé la lettre  m
J'ai trouvé la lettre  a
J'ai trouvé la lettre  i
```

```
J'ai trouvé la lettre  n
J'ai trouvé la lettre  j
J'ai trouvé la lettre  e
J'ai trouvé la lettre  m
J'ai trouvé la lettre  a
J'ai trouvé la lettre  n
J'ai trouvé la lettre  g
J'ai trouvé la lettre  e
J'ai trouvé la lettre  a
J'ai trouvé la lettre  l
J'ai trouvé la lettre  a
J'ai trouvé la lettre  c
J'ai trouvé la lettre  a
J'ai trouvé la lettre  n
J'ai trouvé la lettre  t
J'ai trouvé la lettre  i
J'ai trouvé la lettre  n
J'ai trouvé la lettre  e
J'ai trouvé la lettre  i
J'ai trouvé la lettre  l
J'ai trouvé la lettre  y
J'ai trouvé la lettre  a
J'ai trouvé la lettre  u
J'ai trouvé la lettre  r
J'ai trouvé la lettre  a
J'ai trouvé la lettre  d
J'ai trouvé la lettre  e
J'ai trouvé la lettre  s
J'ai trouvé la lettre  h
J'ai trouvé la lettre  a
J'ai trouvé la lettre  r
J'ai trouvé la lettre  i
J'ai trouvé la lettre  c
J'ai trouvé la lettre  o
J'ai trouvé la lettre  t
J'ai trouvé la lettre  s
J'ai trouvé la lettre  A
J'ai trouvé la lettre  d
J'ai trouvé la lettre  e
J'ai trouvé la lettre  m
J'ai trouvé la lettre  a
J'ai trouvé la lettre  i
J'ai trouvé la lettre  n
J'ai trouvé la lettre  J
J'ai trouvé la lettre  B
```

Si le résultat produit est le même, les deux formes sont-elles équivalentes ? La réponse est non. Le premier cas est «meilleur» en ce sens qu'il permet d'obtenir les lettres alphabétiques de jeu de

caractères utilisé (donc dépend de la langue!) alors que la seconde ne «capture» que les lettres standard a-ZA-Z. Dans notre cas c'est équivalent, mais si on écrit une application multilingue c'est bien différent (songez aux accents, aux caractères bizzares œ, etc).

1.4 Les dictionnaires

Notre dernier sous-problème (la décomposition de problèmes en sous-problèmes est une démarche scientifique très classique) est d'obtenir non l'affichage mais le comptage des occurences…

Une petite réflexion s'impose. Je pourrais créer autant de compteurs que de lettres, soit des variables indépendantes, soit un tableau de valeurs où chaque entrée correspondrait au comptage d'une lettre. L'iconvénient est que du coup, le programme s'adapterait mal à de nouvelles lettres. Une solution plus élégante est d'utiliser une dictionnaire.

Les dictionnaires Python sont des structures de données qui contiennent des couples clé:valeur. La clé est le point d'entrée pour la valeur associée comme dans un dictionnaire où un mot est l'entrée pour sa définition. Regardons un peu comme cela fonctionne :

[12]:
```
unDictionnaire = { "einstein":1879, "chomsky":1928, "kant":1724, "de Gouges":
↪1748 }
print(unDictionnaire)
```

```
{'einstein': 1879, 'chomsky': 1928, 'kant': 1724, 'de Gouges': 1748}
```

Comme on peut le comprendre aisément, la structure du dictionnaire permet d'associer, ici à un nom de personnage célèbre sa date de naissance. De sorte qu'on puisse retrouver cette information à volonté :

[17]:
```
print("Einstein est né en",unDictionnaire["einstein"])
```

```
Einstein est né en 1879
```

Finalement un dictionnaire se comporte comme un tableau dont les indices ne seraient pas des nombres mais des valeurs quelconques!

[23]:
```
activité = { "einstein":"physicien", "chomsky":"linguiste", "kant":
↪"philosophie", "de Gouges":"juriste"}
print(activité)
qui = "chomsky"
print(qui,"exercait comme",activité[qui])
```

```
{'einstein': 'physicien', 'chomsky': 'linguiste', 'kant': 'philosophie', 'de
Gouges': 'juriste'}
chomsky exercait comme linguiste
```

On peut tester si une clé est présente :

[29]:
```
qui = "de Gouges"
if qui in activité:
    print(qui,"a une activité :",activité[qui])
else:
```

```
    print(qui,"n'a pas d'activité connue")

qui = "jojo"
if qui in activité:
    print(qui,"a une activité :",activité[qui])
else:
    print(qui,"n'a pas d'activité connue")
```

```
de Gouges a une activité : juriste
jojo n'a pas d'activité connue
```

Ainsi *clé* `in` *dictionnaire* permet de tester l'existence d'une entrée de dictionnaire pour la *clé*.

On va donc pouvoir résoudre notre problème! Il suffit d'avoir un dictionnaire dont les clés seraient les lettres et la valeur le nombre d'occurences :

[30]:
```
import string
fichier = open('contenu.txt','r')
resultat = {}
for ligne in fichier:
    for mot in ligne.split():
        for lettre in mot:
            if lettre in string.ascii_letters:
                if lettre in resultat:
                    resultat[lettre] += 1 # il y a déjà une entrée pour la␣
 ↪lettre
                else:
                    resultat[lettre] = 1 # c'est la première fois qu'on␣
 ↪rencontre cette lettre
fichier.close()
print(resultat)
```

```
{'B': 2, 'o': 3, 'n': 7, 'j': 2, 'u': 2, 'r': 3, 'C': 1, 'a': 11, 'v': 1, 'b':
1, 'i': 6, 'e': 7, 'D': 1, 'm': 3, 'g': 1, 'l': 2, 'c': 2, 't': 2, 'y': 1, 'd':
2, 's': 2, 'h': 1, 'A': 1, 'J': 1}
```

Note : la syntaxe `+= 1` permet simplement de modifier une variable numérique en lui ajoutant la valeur 1.

Pouvons nous améliorer les choses ? On pourrait par exemple souhaiter compter les occurences indépendemment de leur casse (distinction majuscule/minuscule). Une façon de faire serait de simplement, avant comptage, transformer les lettres en minuscules :

[32]:
```
import string
fichier = open('contenu.txt','r')
resultat = {}
for ligne in fichier:
    for mot in ligne.split():
        for lettre in mot:
            if lettre in string.ascii_letters:
```

```
                l = lettre.lower()
                if l in resultat:
                    resultat[l] += 1 # il y a déjà une entrée pour la lettre
                else:
                    resultat[l] = 1 # c'est la première fois qu'on rencontre␣
 ↪cette lettre
fichier.close()
print(resultat)
```

```
{'b': 3, 'o': 3, 'n': 7, 'j': 3, 'u': 2, 'r': 3, 'c': 3, 'a': 12, 'v': 1, 'i':
6, 'e': 7, 'd': 3, 'm': 3, 'g': 1, 'l': 2, 't': 2, 'y': 1, 's': 2, 'h': 1}
```

Rappellez-vous...la documentation... Transformer une lettre en sa sœur minuscule est certainement une fonctionnalité standard!

Maintenant pouvons-nous prsenter les résultats de meilleure manière ?

[33]:
```
import string
fichier = open('contenu.txt','r')
resultat = {}
for ligne in fichier:
    for mot in ligne.split():
        for lettre in mot:
            if lettre in string.ascii_letters:
                l = lettre.lower()
                if l in resultat:
                    resultat[l] += 1 # il y a déjà une entrée pour la lettre
                else:
                    resultat[l] = 1 # c'est la première fois qu'on rencontre␣
 ↪cette lettre
fichier.close()

for c,v in resultat.items():
    print("Le caractère %s est présent %s fois." % (c,v))
```

```
Le caractère b est présent 3 fois.
Le caractère o est présent 3 fois.
Le caractère n est présent 7 fois.
Le caractère j est présent 3 fois.
Le caractère u est présent 2 fois.
Le caractère r est présent 3 fois.
Le caractère c est présent 3 fois.
Le caractère a est présent 12 fois.
Le caractère v est présent 1 fois.
Le caractère i est présent 6 fois.
Le caractère e est présent 7 fois.
Le caractère d est présent 3 fois.
Le caractère m est présent 3 fois.
```

```
Le caractère g est présent 1 fois.
Le caractère l est présent 2 fois.
Le caractère t est présent 2 fois.
Le caractère y est présent 1 fois.
Le caractère s est présent 2 fois.
Le caractère h est présent 1 fois.
```

Deux choses à dire. La première c'est qu'on peut itérer sur les couples clé/valeur mais il faut pour cela extraire du dictionnaire la séquence des couples :

`for c,v in resultat.items():`

`c` et `v` capturent respectivement la clé et la valeur de tous les couples (items) contenus dans le dictionnaire.

La seconde concerne l'affichage qui est ici un peu plus compliqué que d'habitude parce qu'on souhaite un contrôlé plus fin (sans insister plus que ça sur la finesse possible) :

`print("Le caractère %s est présent %s fois." % (c,v))`

La chaîne à afficher est ici *paramétrée*, ce qui signifie qu'elle contient des emplacements/trous qui seront comblés avec les informations qui suivent. Les trous sont symbolisés par **`%s`** et donc la chaîne de caractère représente la forme générale de l'affichage final. Ce qui suit **`%`** représente l'ensemble des valeurs qui rempliront les trous (dans l'ordre). On affiche donc dans le premier trou le caractère et dans le second la valeur.

Et si je souhaite obtenir un affichage trié dans l'ordre alphabétique ?

[36]:
```
import string
fichier = open('contenu.txt','r')
resultat = {}
for ligne in fichier:
    for mot in ligne.split():
        for lettre in mot:
            if lettre in string.ascii_letters:
                l = lettre.lower()
                if l in resultat:
                    resultat[l] += 1 # il y a déjà une entrée pour la lettre
                else:
                    resultat[l] = 1 # c'est la première fois qu'on rencontre␣
 ↪cette lettre
fichier.close()

for c in sorted(resultat):
    print("Le caractère %s est présent %s fois." % (c,resultat[c]))
```

```
Le caractère a est présent 12 fois.
Le caractère b est présent 3 fois.
Le caractère c est présent 3 fois.
Le caractère d est présent 3 fois.
Le caractère e est présent 7 fois.
```

```
Le caractère g est présent 1 fois.
Le caractère h est présent 1 fois.
Le caractère i est présent 6 fois.
Le caractère j est présent 3 fois.
Le caractère l est présent 2 fois.
Le caractère m est présent 3 fois.
Le caractère n est présent 7 fois.
Le caractère o est présent 3 fois.
Le caractère r est présent 3 fois.
Le caractère s est présent 2 fois.
Le caractère t est présent 2 fois.
Le caractère u est présent 2 fois.
Le caractère v est présent 1 fois.
Le caractère y est présent 1 fois.
```

Trier est une opération assez commune non ? Elle est dans le module standard!

Exercice Sauriez-vous trier dans l'ordre inverse ? Cherchez dans la documentation de sorted!

```
Le caractère y est présent 1 fois.
Le caractère v est présent 1 fois.
...
```

Exercice Sauriez-vous écrire un programme qui cette fois compte les occurences des mots!

```
Le mot bonjour est présent 1 fois.
Le mot revoir est présent 1 fois.
...
```

Exercice À partir du précédent, sauriez-vous obtenir la longueur moyenne des mots lus ?