td1a_cenonce_session4

July 1, 2022

1 1A.2 - Modules, fichiers, expressions régulières

Le langage Python est défini par un ensemble de règle, une grammaire. Seul, il n'est bon qu'à faire des calculs. Les modules sont des collections de fonctionnalités pour interagir avec des capteurs ou des écrans ou pour faire des calculs plus rapides ou plus complexes.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

1.0.1 Fichiers

Les fichiers permettent deux usages principaux :

- récupérer des données d'une exécution du programme à l'autre (lorsque le programme s'arrête, toutes les variables sont perdues)
- échanger des informations avec d'autres programmes (Excel par exemple).

Le format le plus souvent utilisé est le fichier plat, texte, txt, csv, tsv. C'est un fichier qui contient une information structurée sous forme de matrice, en ligne et colonne car c'est comme que les informations numériques se présentent le plus souvent. Un fichier est une longue séquence de caractères. Il a fallu choisir une convention pour dire que deux ensembles de caractères ne font pas partie de la même colonne ou de la même ligne. La convention la plus répandue est :

- `\t` : séparateur de colonnes
- `\n` : séparateur de lignes

Le caractère `\` indique au langage python que le caractère qui suit fait partie d'un code. Vous trouverez la liste des codes : String and Bytes literals.
Aparté : aujourd'hui, lire et écrire des fichiers est tellement fréquent qu'il existe des outils qui font ça dans une grande variété de formats. Vous découvrirez cela lors de la séance 10. Il est utile pourtant de le faire au moins une fois soi-même pour comprendre la logique des outils et pour ne pas être bloqué dans les cas non prévus.
Ecrire et lire des fichiers est beaucoup plus long que de jouer avec des variables. Ecrire signifie qu'on enregistre les données sur le disque dur : elles passent du programme au disque dur (elles deviennent permanentes). Elles font le chemin inverse lors de la lecture.

Ecriture Il est important de retenir qu'un fichier texte ne peut recevoir que des chaînes de caractères.

[2]:
```
mat = [[1.0, 0.0],[0.0,1.0] ]                    # matrice de type liste de listes
with open ("mat.txt", "w") as f :                # création d'un fichier en mode␣
 ↪écriture
```

```
    for i in range (0,len (mat)) :                #
        for j in range (0, len (mat [i])) :       #
            s = str (mat [i][j])                  # conversion en chaîne de caractères
            f.write (s + "\t")                    #
        f.write ("\n")                            #

# on vérifie que le fichier existe :
import os
print([ _ for _ in os.listdir(".") if "mat" in _ ] )

# la ligne précédente utilise le symbole _ : c'est une variable
# le caractère _ est une lettre comme une autre
# on pourrait écrire :
# print([ fichier for fichier in os.listdir(".") if "mat" in fichier ] )
# on utilise cette convention pour dire que cette variable n'a pas vocation à rester
```

```
['mat.txt', 'matrix_dictionary.ipynb', 'seance4_excel_mat.txt',
'seance4_excel_mat.xlsx']
```

Le même programme mais écrit avec une écriture condensée :

[3]:
```
mat = [[1.0, 0.0],[0.0,1.0] ]             # matrice de type liste de listes
with open ("mat.txt", "w") as f :        # création d'un fichier
    s = '\n'.join ( '\t'.join( str(x) for x in row  ) for row in mat )
    f.write ( s )

# on vérifie que le fichier existe :
print([ _ for _ in os.listdir(".") if "mat" in _ ] )
```

```
['mat.txt', 'matrix_dictionary.ipynb', 'seance4_excel_mat.txt',
'seance4_excel_mat.xlsx']
```

On regare les premières lignes du fichier `mat2.txt` :

[4]:
```
import pyensae
%load_ext pyensae
%head mat.txt
```

[4]:
```
<IPython.core.display.HTML object>
```

Lecture

[5]:
```
with open ("mat.txt", "r") as f : # ouverture d'un fichier
    mat = [ row.strip(' \n').split('\t') for row in f.readlines() ]
print(mat)
```

```
[['1.0', '0.0'], ['0.0', '1.0']]
```

On retrouve les mêmes informations à ceci près qu'il ne faut pas oublier de convertir les nombres initiaux en `float`.

[6]:
```
with open ("mat.txt", "r") as f :                # ouverture d'un fichier
    mat = [ [ float(x) for x in row.strip(' \n').split('\t') ] for row in f.
 ↪readlines() ]
```

```
print(mat)
```

```
[[1.0, 0.0], [0.0, 1.0]]
```

Voilà qui est mieux. Le module os.path propose différentes fonctions pour manipuler les noms de fichiers. Le module os propose différentes fonctions pour manipuler les fichiers :

[7]:
```
import os
for f in os.listdir('.'):
    print (f)
```

```
.ipynb_checkpoints
data
exemple_fichier.txt
image.png
image.png.gv
images
integrale_rectangle.ipynb
integrale_rectangle_correction.ipynb
j2048.ipynb
j2048_correction.ipynb
mat.txt
matrix_dictionary.ipynb
monmodule.py
monmodule3.py
page.html
pp_exo_deviner_un_nombre.ipynb
pp_exo_deviner_un_nombre_correction.ipynb
README.txt
seance4_excel.txt
seance4_excel.xlsx
seance4_excel_mat.txt
seance4_excel_mat.xlsx
td1a_cenonce_session1.ipynb
td1a_cenonce_session2.ipynb
td1a_cenonce_session3.ipynb
td1a_cenonce_session4.ipynb
td1a_cenonce_session5.ipynb
td1a_cenonce_session6.ipynb
td1a_correction_session1.ipynb
td1a_correction_session2.ipynb
td1a_correction_session3.ipynb
td1a_correction_session4.ipynb
td1a_correction_session5.ipynb
td1a_correction_session6.ipynb
td1a_pyramide_bigarree.ipynb
td1a_pyramide_bigarree_correction.ipynb
td2_1.png
texte_langue.ipynb
texte_langue_correction.ipynb
voeux.zip
VOEUX01.txt
VOEUX05.txt
VOEUX06.txt
```

```
VOEUX07.txt
VOEUX08.txt
VOEUX09.txt
VOEUX74.txt
VOEUX75.txt
VOEUX79.txt
VOEUX83.txt
VOEUX87.txt
VOEUX89.txt
VOEUX90.txt
VOEUX94.txt
__pycache__
```

with De façon pragmatique, l'instruction with permet d'écrire un code plus court d'une instruction : `close`. Les deux bouts de code suivant sont équivalents :

[8]:
```
with open("exemple_fichier.txt", "w") as f:
    f.write("something")
```

[9]:
```
f = open("exemple_fichier.txt", "w")
f.write("something")
f.close()
```

L'instruction `close` *ferme* le fichier. A l'ouverture, le fichier est réservé par le programme Python, aucune autre application ne peut écrire dans le même fichier. Après l'instruction `close`, une autre application pour le supprimer, le modifier. Avec le mot clé `with`, la méthode `close` est implicitement appelée.

à quoi ça sert ? On écrit très rarement un fichier texte. Ce format est le seul reconnu par toutes les applications. Tous les logiciels, tous les langages proposent des fonctionnalités qui exportent les données dans un format texte. Dans certaines circonstances, les outils standards ne fonctionnent pas - trop grops volumes de données, problème d'encoding, caractère inattendu -. Il faut se débrouiller.

[10]:
```
```

1.0.2 Exercice 1 : Excel → Python → Excel

Il faut télécharger le fichier seance4_excel.xlsx qui contient une table de trois colonnes. Il faut :

- enregistrer le fichier au format texte,
- le lire sous python
- créer une matrice carrée 3x3 où chaque valeur est dans sa case (X,Y),
- enregistrer le résultat sous format texte,
- le récupérer sous Excel.

1.0.3 Autres formats de fichiers

Les fichiers texte sont les plus simples à manipuler mais il existe d'autres formats classiques~:

- html : les pages web
- xml : données structurées
- [zip](http://fr.wikipedia.org/wiki/ZIP_(format_de_fichier), gz : données compressées
- wav, mp3, ogg : musique
- mp4, Vorbis : vidéo
- ...

1.0.4 Modules

Les modules sont des extensions du langages. Python ne sait quasiment rien faire seul mais il bénéficie de nombreuses extensions. On distingue souvent les extensions présentes lors de l'installation du langage (le module math) des extensions externes qu'il faut soi-même installer (numpy). Deux liens :

- modules officiels
- modules externes

Le premier réflexe est toujours de regarder si un module ne pourrait pas vous être utile avant de commencer à programmer. Pour utiliser une fonction d'un module, on utilise l'une des syntaxes suivantes :

[11]:
```
import math
print (math.cos(1))

from math import cos
print (cos(1))

from math import *    # cette syntaxe est déconseillée car il est possible qu'une␣
 ↪fonction
print (cos(1))        # porte le même nom qu'une des vôtres
```

```
0.5403023058681398
0.5403023058681398
0.5403023058681398
```

1.0.5 Exercice 2 : trouver un module (1)

Aller à la page modules officiels (ou utiliser un moteur de recherche) pour trouver un module permettant de générer des nombres aléatoires. Créer une liste de nombres aléatoires selon une loi uniforme puis faire une permutation aléatoire de cette séquence.

1.0.6 Exercice 3 : trouver un module (2)

Trouver un module qui vous permette de calculer la différence entre deux dates puis déterminer le jour de la semaine où vous êtes nés.

1.0.7 Module qu'on crée soi-même

Il est possible de répartir son programme en plusieurs fichiers. Par exemple, un premier fichier `monmodule.py` qui contient une fonction :

[12]:
```
# fichier monmodule.py
import math

def fonction_cos_sequence(seq) :
    return [ math.cos(x) for x in seq ]

if __name__ == "__main__" :
    print ("ce message n'apparaît que si ce programme est le point d'entrée")
```

```
ce message n'apparaît que si ce programme est le point d'entrée
```

La cellule suivante vous permet d'enregistrer le contenu de la cellule précédente dans un fichier appelée `monmodule.py`.

[13]:
```
code = """
# -*- coding: utf-8 -*-
import math
def fonction_cos_sequence(seq) :
    return [ math.cos(x) for x in seq ]
if __name__ == "__main__" :
    print ("ce message n'apparaît que si ce programme est le point d'entrée")
"""
with open("monmodule.py", "w", encoding="utf8") as f :
    f.write(code)
```

Le second fichier :

[14]:
```
import monmodule

print ( monmodule.fonction_cos_sequence ( [ 1, 2, 3 ] ) )
```

```
[0.5403023058681398, -0.4161468365471424, -0.9899924966004454]
```

Note : Si le fichier `monmodule.py` est modifié, *python* ne recharge pas automatiquement le module si celui-ci a déjà été chargé. On peut voir la liste des modules en mémoire dans la variable `sys.modules` :

[15]:
```
import sys
list(sorted(sys.modules))[:10]
```

[15]:
```
['IPython',
 'IPython.core',
 'IPython.core.alias',
 'IPython.core.application',
 'IPython.core.async_helpers',
 'IPython.core.autocall',
 'IPython.core.builtin_trap',
 'IPython.core.compilerop',
 'IPython.core.completer',
 'IPython.core.completerlib']
```

Pour retirer le module de la mémoire, il faut l'enlever de `sys.modules` avec l'instruction `del sys.modules['monmodule']`. *Python* aura l'impression que le module *monmodule.py* est nouveau et il l'importera à nouveau.

1.0.8 Exercice 4 : son propre module

Que se passe-t-il si vous remplacez `if __name__ == "__main__":` par `if True :`, ce qui équivaut à retirer la ligne `if __name__ == "__main__":` ?

1.0.9 Expressions régulières

Pour la suite de la séance, on utilise comme préambule les instructions suivantes :

[16]:
```
import pyensae.datasource
discours = pyensae.datasource.download_data('voeux.zip', website = 'xd')
```

La documentation pour les expressions régulières est ici : regular expressions. Elles permettent de rechercher des motifs dans un texte :

- *4 chiffres / 2 chiffres / 2 chiffres* correspond au motif des dates, avec une expression régulière, il s'écrit : `[0-9]{4}/[0-9]{2}/[0-9]{2}`
- *la lettre a répété entre 2 et 10 fois* est un autre motif, il s'écrit : `a{2,10}`.

[17]:
```
import re   # les expressions régulières sont accessibles via le module re
expression = re.compile("[0-9]{2}/[0-9]{2}/[0-9]{4}")
texte = """Je suis né le 28/12/1903 et je suis mort le 08/02/1957. Ma seconde femme␣
 ↪est morte le 10/11/63."""
cherche = expression.findall(texte)
print(cherche)
```

```
['28/12/1903', '08/02/1957']
```

Pourquoi la troisième date n'apparaît pas dans la liste de résultats ?

1.0.10 Exercice 5 : chercher un motif dans un texte

On souhaite obtenir toutes les séquences de lettres commençant par *je* ? Quel est le motif correspondant ? Il ne reste plus qu'à terminer le programme précédent.

[18]:
```

```

1.0.11 Exercice 6 : chercher un autre motif dans un texte

Avec la même expression régulière, rechercher indifféremment le mot *securite* ou *insecurite*.

[19]:
```

```

On peut passer du temps à construire des expressions assez complexes surtout quand on oublie quelques Petites subtilités avec les expressions régulières en Python.

1.1 Exercice 7 : recherche les urls dans une page wikipédia

On pourra prendre comme exemple la page du programme Python.

1.2 Exercice 8 : construire un texte à motif

A l'inverse des expressions régulières, des modules comme Mako ou Jinja2 permettent de construire simplement des documents qui suivent des règles. Ces outils sont très utilisés pour la construction de page web. On appelle cela faire du templating. Créer une page web qui affiche à l'aide d'un des modules la liste des dimanches de cette année.

[20]:
```

```