MODULES, GRAPHIQUES ET FICHIERS EXTERNES

Si Python est un des langages de programmation les plus utilisés aujourd'hui, c'est parce qu'il existe une multitude de *modules* (package en anglais) qui contiennent des fonctions déjà « toutes faites ». Quand on veut réaliser une tâche précise, bien souvent il existe un module qui peut s'en charger en quelques lignes.

Une *bibliothèque* est un ensemble de modules. Toute installation de Python dispose déjà d'une *bibliothèque standard* avec de nombreux modules et outils. Cependant, d'autres modules seront nécessaires en cours d'année.

1 Quelques modules simples

Pour utiliser une `<fonction>` d'un `<module>` donné, la syntaxe est toujours la même : `<module> . <fonction>` suivi des parenthèses avec les arguments.

Module random. Le module `random` contient tout ce qui permet de simuler le hasard. Voici les deux fonctions les plus utiles :

```
import random              # On importe le module

p = random.randint(5,8)    # Génère un nombre entier aléatoire entre 5 et 8 inclus
print(p)
q = random.random()        # Génère un nombre décimal aléatoire entre 0 et 1
print(q)
```

Une fois importé, le module apparait dans le Workspace avec le type « module ». Il n'est alors plus nécessaire de l'importer à chaque fois qu'on veut utiliser une fonction du module.

Exercice 1. Écrire une fonction `alea(a,b)` qui retourne un nombre *décimal* aléatoire entre `a` et `b`.

Module numpy. Le module `numpy` contient les constantes et fonctions usuelles qu'on utilise en mathématiques, mais pas que !

```
import numpy as np     # np devient un alias, cf plus bas.

print( np.cos(0) )     # np.cos équivaut maintenant à numpy.cos
print( np.pi )
```

Exercice 2. Calculer $e^{\pi}\arccos\left(\frac{1}{5}\right)$.

Tableaux numpy (ou array) vs listes. En plus, le module `numpy` est particulièrement utile pour créer des tableaux de nombre ou faire des calculs avec. Mais pour cela, il faut convertir les listes ou les tableaux en *array* (ou *tableaux numpy*).

```
import numpy as np     # si vous l'avez déjà compilé, pas besoin de le remettre

L = [1,3]              # L est une liste mais pas une array
print(L+L)
T = np.array( L )      # Conversion en une array (ou tableau numpy)
print(T+T)
```

Avec un tableau numpy, les opérations `+ - * /` ... se font « élément par élément ». Pour les listes, ces opérations n'ont pas le même sens, comme on le voit plus haut. Il est donc important de savoir si ce qu'on manipule est une *array* ou pas.

En tapant `T` dans la console, on voit le mot-clé `array(...)` : cela permet de vérifier que `T` est une array. Le Workspace peut aussi nous renseigner par la colonne « Type » : `ndarray` pour `T`, et `list` pour `L`. A noter : toute fonction du package `numpy` retournera une array plutôt qu'une liste.

Exercice 3. Écrire un script qui affiche un tableau numpy contenant les carrés des entiers de 0 à 20.

2 Bibliothèque matplotlib

Matplotlib est une bibliothèque destinée à la faire toutes sortes de graphiques : courbes, histogrammes, nuages de points, etc. Vous pouvez consulter l'impressionnant éventail de possibilités sur le site https://matplotlib.org, sur la page « Examples ». On y trouve aussi des tutoriels très bien faits (en anglais). Pensez-y pour vos TIPE !

Module pyplot. Matplotlib, en tant que bibliothèque, contient de nombreux modules. On utilisera surtout le module `pyplot`. Copier et compiler le code suivant :

```
import matplotlib.pyplot as plt
X = [0,1,2]
Y = [2,5,3]
plt.close()      # ferme les graphiques existants (nécessaire si on recompile)
plt.plot(X,Y)    # trace les points de coordonnées ( X[i] , Y[i] )
plt.show()       # affiche le graphique
```

Tracé d'une fonction. Pour tracer une fonction, il est important de prendre une liste d'abscisses `x` avec des points nombreux et équidistants. Le module `numpy` dispose de LA fonction ad hoc : `np.linspace(a,b,n)` retourne une *array* de `n` points équidistants allant de `a` à `b` inclus.

```
def f(t) :
    return t**2-2

X = np.linspace(0,2,100)  # array de 100 points allant de 0 à 2
Y = [ f(x) for x in X ]   # liste [ f(X[0]) , f(X[1]) , ... , f(X[100]) ]

plt.close()      # ferme les graphiques existants (nécessaire si on recompile)
plt.plot(X,Y)    # trace les points de coordonnées ( X[i] , Y[i] )
plt.show()       # affiche le graphique
```

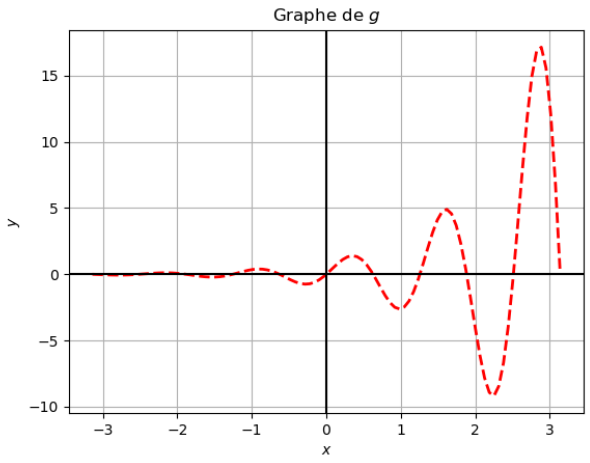
Gare ! `X` est un tableau numpy, mais `Y` est une liste. La syntaxe pour définir `Y` est à retenir : on appelle ça une *liste en compréhension*.

Un *joli* tracé de fonction. On peut améliorer grandement la présentation d'un graphique de diverses façons :

```
def g(t) :
    return np.sin(5*t)*np.exp(t)

X = np.linspace(-np.pi,np.pi,100)
Y = [ g(x) for x in X ]

plt.close()
plt.plot(X,Y, color='r', linewidth=2,
    linestyle='--')
plt.grid()                      # grille
plt.title('Graphe de $g$')      # titre
plt.axhline(color='black')      # axe x
plt.axvline(color='black')      # axe y
plt.xlabel('$x$')     # intitulé axe x
plt.ylabel('$y$')     # intitulé axe y
plt.show()
```



La fonction `plt.plot` peut prendre des arguments supplémentaires pour préciser la couleur (`r` : rouge, `b` : bleu, `g` : vert, etc.), l'épaisseur et le style du trait (`-` tirets, `:` pointillés, etc.). On peut les mettre dans l'ordre qu'on veut, mais toujours après `X,Y`.

Enfin, `$g$` permet d'afficher g plutôt que g.

Exercice 4. Afficher sur le même graphique les courbes de $h : x \to x^\alpha$ sur $[0,2]$, pour α dans $[0.25, 0.5, 1, 1.5, 2]$. Indication : utiliser une boucle qui parcourt les valeurs de α, avec `plt.plot(...)` dans la boucle. Mettre `plt.close()` avant la boucle et `plt.show()` après.

3 Lecture et écriture de fichiers

Dossier courant. Python peut récupérer le contenu d'un fichier et/ou le modifier. Mais il faut dire à Python où se trouve le fichier en question. Par défaut, Python cherche le fichier dans le *dossier courant*. Taper `cd` dans la console pour vérifier quel est le dossier courant.

Il est souvent pratique que le dossier courant soit le même que le dossier qui contient votre fichier `.py`. Pour changer de dossier courant, taper `cd C:/Chemin/vers/le/dossier` dans la console. Pour ne pas avoir à le refaire à chaque fois qu'on lance Pyzo, aller dans Shell > Configuration des shells et modifier le champ « StartDir ».

Lecture du fichier Téléchargez le fichier `poeme.txt`. depuis le site et placez-le dans votre dossier courant. La fonction `open` de Python possède plusieurs options : `'r'` (pour *read*) signifie que le fichier est ouvert en lecture seule, on ne peut pas le modifier.

```
f = open('poeme.txt','r')   # f est une variable qui "pointe" vers le fichier
texte = f.readlines()      # On stocke toutes les lignes dans une liste
f.close()                   # On ferme le fichier, ne pas oublier !
print(texte)
```

Ce n'est pas très lisible. On remarque que `texte` est une liste dont chaque élément correspond à une ligne du fichier. On peut donc remplacer `print(texte)` par :

```
for ligne in texte :
    print(ligne)
```

Mais il y a un saut de ligne entre chaque ligne ! Cela est dû au caractère `\n` à la fin de chaque ligne, qui correspond à un retour à la ligne. On peut le retirer avec la méthode `.rstrip()` :

```
for ligne in texte :
    print(ligne.rstrip())
```

Enfin, si on n'a pas besoin de stocker le texte entier, on peut parcourir le fichier ligne par ligne directement :

```
f = open('poeme.txt','r')
for ligne in f :
    print(ligne.rstrip())
f.close()
```

Attention : cela signifie que `f.close()` doit arriver plus tard... Il est en général recommandé d'ouvrir le fichier, d'extraire ce dont on a besoin, puis de le refermer aussitôt.

Avec un document csv. Python peut traiter d'autres formats que les `.txt`. Récupérer le fichier « population.csv ». Ouvrez-le avec un programme externe pour visualiser les données qu'il contient.

Exercice 5. Dans Pyzo, afficher toutes les lignes de population.csv comme ci-dessus. A quoi correspond le point-virgule ; ?

Outre le problème du point virgule, on ne veut pas garder la première ligne qui contient les en-têtes. Enfin, les nombres sont stockés sous forme de chaines de caractères, et on voudrait les stocker sous forme de nombres dans Python. Voici une façon de contourner ces 3 problèmes :

```
f = open("population.csv", "r")
f.readline()                    # permet de sauter la 1ère ligne
texte = f.readlines()           # stocke le reste des lignes
f.close()

annee,pop = [],[]               # initialisation des listes
for ligne in texte :
    L = ligne.rstrip()          # on enlève le \n
    a,p = L.split(';')          # on sépare le contenu selon le ;

    annee.append(float(a))      # a et p sont des chaines...
    pop.append(float(p))        # float() les convertit en nombres "flottants"
print(annee)
print(pop)
```

En réalité, lorsqu'on ouvre un fichier avec open, Python place un « curseur » au début du fichier : la lecture s'effectue à partir de ce curseur. L'instruction `f.readline()` (sans s) retourne une ligne à partir du curseur, et place le curseur au début de la ligne suivante. Ensuite, l'instruction `f.readlines()` lit le fichier jusqu'au bout... mais à partir du curseur ! On a donc bien sauté la première ligne.

La fonction `f.readlines()` place le curseur à la fin du fichier :

```
f = open('poeme.txt','r')     # curseur au début
print( f.readlines() )        # curseur début -> fin, on a tout lu
print( f.readlines() )        # curseur fin -> fin, on a rien lu
f = open('poeme.txt','r')     # curseur au début
print( f.readlines() )        # curseur début -> fin, on a tout lu
```

Exercice 6. À partir des listes annee et pop obtenues précédemment, faire un graphique de la population en fonction de l'année.

Écriture du fichier. Pour modifier un fichier, il ne faut pas l'ouvrir en lecture seule (option `'r'`) : on verra les options `r+`, `a+` et `w+`. Ensuite, on utilise `f.write(...)`. Commençons par l'option `r+` :

```
f = open('poeme.txt','r+')  # le curseur est placé AU DÉBUT
f.write('Et blablabla')
f.close()                   # Regarder le fichier à part
```

On voit que `f.write` a remplacé « `Que j'aime à` » par « `Et blablabla` ». Le problème vient du fait que l'écriture se fait à partir du curseur et « écrase » le contenu déjà présent. Corriger le poème, puis utiliser l'option `a+` :

```
f = open('poeme.txt','a+')  # le curseur est placé À LA FIN
f.write('Et blablabla')
f.close()                   # Vérifier que tout va bien dans le fichier
```

⚠ Pour écrire dans un fichier, on peut aussi utiliser `f=open('...','w+')` avec `w+` pour *write*. Mais cette commande *efface* le contenu du fichier. On l'utilise en général quand on a stocké dans une chaine tout le nouveau contenu du fichier.

4 Exercices d'approfondissement

Exercice 7. Écrire une fonction `liste_alea(n,p)` qui retourne une liste de `n` nombres *entiers* aléatoires entre `1` et `p`. Modifier cette fonction en `liste_alea_dist(n,p)`pour qu'elle renvoie une liste d'entiers distincts (avec $n \leq p$).

Exercice 8. Rajouter la donnée `2022;8.0` à population.csv et reprendre l'exercice 6. Vous pouvez faire un joli graphique !

Exercice 9. Écrire un fichier `emeop.txt`, qui contient les lignes de `poeme.txt` mais en ordre inverse.