MODULES ET GRAPHIQUES

Les *modules* (package en anglais) 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. Toute installation de Python dispose déjà d'une bibliothèque standard qui contient de nombreux modules. Cependant, d'autres modules seront nécessaires en cours d'année.

Syntaxe pour les modules :

- `import <mod>` pour importer le module.
- `<mod>.<fonc>(...)` permet d'utiliser la fonction `<fonc>` du module `<mod>`, avec « . . . » pour les arguments de `<fonc>`.

1 Quelques modules simples

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 flottant 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` qui prend en argument deux flottants $a < b$ et retourne un flottant aléatoire compris entre a et b.

Module numpy. Le module `numpy` contient les constantes et fonctions usuelles mathématiques, mais pas que !

```
import numpy as np     # "as np" => np devient un alias, cf lignes suivantes.

print( np.cos(0) )     # np.cos remplace maintenant à numpy.cos

print( np.pi )         # pas de parenthèses car np.pi est une "constante"
```

Exercice 2. Calculer avec Python la valeur $e^{\pi}\arctan(5)$.

Tableaux numpy (ou array) vs listes. Les fonctions `numpy` retournent ou bien des flottants, ou bien un nouveau type de variable : les *tableaux numpy* (notés `ndarray` dans le Workspace). Ce type ressemble aux listes (ou liste de listes) mais il y a des exceptions notables :

- En 2 dimensions, un tableau numpy doit être rectangulaire : toutes les lignes doivent avoir le même nombre de colonnes et réciproquement, tout comme les matrices en mathématiques. Il ne peut pas y avoir de « trou ». Idem pour n dimensions.
- Les opérations arithmétiques sur les tableaux numpy s'effectuent « élement par élément ».

La fonction `np.array(...)` permet de convertir une liste donnée en argument en tableau numpy (comme `int(...)` pour les entiers, etc.). Rentrez les commandes suivantes dans la console :

```
L = [1,2,3]                  # L est une liste mais pas une array
L+L

T = np.array( L )            # Conversion en une array (ou tableau numpy)
T

T+T                          # opérations élément par élément
T*2
T**T

S = np.array( [1,2,3,4] )    # S est de taille 4
S+T                          # erreur car ...
```

Comme on peut le voir, le résultat de `A+A` n'est pas le même selon que `A` est une liste ou un tableau numpy.

Enfin, le module `numpy` dispose d'une fonction très utile pour les graphes : `np.linspace(a,b,n)` retourne un tableau numpy de `n` points équidistants allant de `a` à `b` inclus.

Exercice 3. Écrire un script qui affiche un tableau numpy contenant les carrés des 51 flottants équidistants de 0 à 10. On utilisera la fonction `linspace`.

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 est un module qui contient de nombreux sous-modules. On utilisera surtout le sous-module `pyplot`qui contient la fonction `plot`. Étant donnés deux listes / tableaux numpy X et Y de même taille n, l'instruction **`plot(X,Y)`**...

- ... trace les points de coordonnées [1] $(X_0,Y_0), (X_1,Y_1), \cdots, (X_{n-1},Y_{n-1})$,
- ... puis trace une ligne brisée qui relie le point d'indice i au point d'indice $i+1$.

Copier et compiler le code suivant :

```
import matplotlib.pyplot as plt  # pyplot est un sous-module de matplotlib

X = [0,1,2]
Y = [2,5,3]

plt.close()    # ferme les graphiques existants (nécessaire si on recompile)
plt.plot(X,Y)
plt.show()     # affiche le graphique
```

1. Où on note $X_i :=$ `X[i]` et $Y_i :=$ `Y[i]`

Tracé d'une fonction. Pour tracer une fonction, il est important de prendre une liste d'abscisses `x` avec des points nombreux et équidistants. La fonction `np.linspace` est un incontournable.

```
def f(t) :
    return np.cos(2*np.pi*t)

n = 11                      # nombre de points
X = np.linspace(0,10,n)     # tableau numpy [ 0, ... , 10 ] avec n points

Y = [ f(x) for x in X ]     # liste [ f(X[0]) , f(X[1]) , ... , f(X[n-1]) ]

plt.close()
plt.plot(X,Y,'-x')          # "-x" permet de mettre une croix sur chaque point
plt.show()
```

Question 1. Pourquoi le graphe retourne une droite horizontale ?

Exercice 4. Recompiler le script pour `n` valant $21, 41, 81, 201$ et enfin 1001. Dans le dernier cas, il faut enlever `"-x"` pour ne plus afficher que la « courbe ».

Question 2. Pour le tracé avec $n = 1001$, zoomer plusieurs fois avec la loupe sur un pic du cosinus. L'état de la « courbe » vous surprend-il ?

Appendice : 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, 101 )
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 avec g écrit "mathématiquement"
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.

3 Exercices d'approfondissement

Exercice 5. Écrire une fonction `liste_alea` qui prend en argument deux entiers n et p et retourne une liste de n nombres *entiers* aléatoires entre 1 et p.

Exercice 6. Recopier et renommer cette fonction en `liste_alea_dist`, puis la modifier pour qu'elle renvoie une liste d'entiers distincts (avec $n \leq p$).

Question 3. On note $t(n)$ le temps d'exécution d'une fonction donnée pour un argument de taille n (dans le pire cas). Un calcul théorique montre qu'il existe $K > 0$ et $\alpha \in \mathbb{R}$ tels que $t(n) \simeq Kn^\alpha$ (on suppose que les autres termes sont négligeables devant n^α). Écrire $\ln t(n)$ comme une fonction de $\ln n$. À quoi correspond α dans cette relation ?

Exercice 7 (Vérification graphique de la complexité)**.** On reprend la fonction `plusProchesValeurs` du TP précédent. On avait trouvé une complexité quadratique, donc pour n assez grand, le temps d'exécution $T(n)$ devrait se comporter comme Kn^2 quand n est grand (en négligeant les autres termes). On se propose de le vérifier en pratique. Compléter le script suivant.

```
def plusProchesValeurs(L) :
    ...
    ...  # recopier la fonction ici, cf corrigé du TP4 si besoin
    ...

import numpy as np
import time
import matplotlib.pyplot as plt

N = ...       # tableau numpy [1000, ... , 4000] avec 11 valeurs
Times = []    # liste des temps d'exécution lorsque n parcourt N

for n in N:

    L = ...  # liste de taille n : [0, 1, ..., n-1]

    # avec la fonction time.time(), chronométrer
    # le temps  d'exécution de plusProchesValeurs(L)

    Times.append( ... )  # ajouter ce temps à la liste Times

plt.close()
X = np.log(N)
Y = np.log(Times)
Z = 2*np.log(N)-16  # la valeur 16 est arbitraire

plt.plot(X,Y,'b-x') # en bleu, ln(Tn) en fonction de n
plt.plot(X,Z,'r')   # en rouge, une droite de pente 2
plt.show()
```

Est-ce qu'une complexité d'ordre n^2 vous semble cohérent ?

Note : le fait que le graphe bleu ne soit pas une droite est lié à plusieurs causes : D'une part, on a négligé des termes proportionnels à n dans le calcul de la complexité. D'autre part, le processeur est sollicité de manière irrégulière par votre ordinateur et donc la puissance de calcul restante est variable au cours du temps. Ces deux causes ont un effet atténué lorsque n est très grand, mais il est déconseillé d'aller trop loin avec une complexité quadratique...

Enfin, un derniere dernière cause est que l'on a négligé certaines instructions dans le décompte d'opérations élémentaires, par exemple les affectations (`d=dist`) ou les évaluations de listes (`L[i]` ou `L[j]`).