TP 2 : Types, listes et boucle for

Python fait une distinction claire entre les différents "types" de données utilisées : ainsi un texte est un objet totalement différent d'un nombre pour lui... et un "nombre" est encore trop vague : est-ce un entier ? un décimal ? un complexe ?

L'objectif de ce TP est de découvrir en particulier le type "liste" et d'écrire quelques algorithmes associés.

La fonction `type()` permet d'identifier le type d'une variable.

Exécutez les commandes suivantes :

dans le shell
```
import numpy as np
a=1
type(a)
a=2.5
type(a)
a="vache"
type(a)
a='vache'
type(a)
a=np.array(['vache',0,1/5])
type(a)
```

Les cinq types ci-dessus seront les plus fréquemment utilisés. Ce TP va se concentrer sur les 4 premiers (int, float, str et les tableaux), le dernier sera employé un peu plus tard.

1 Les nombres entiers et décimaux

Dans le "shell", écrire les instructions suivantes :

dans le shell
```
a=1
b=2.5
c=a+b
c
type(c)
d=c-b
d
type(d)
```

Python identifie de manière distincte les deux types de données, mais il agit de manière transparente pour les manipuler. Ainsi, vous n'avez pas à vous préoccuper du fait que votre nombre peut être entier ou décimal : python s'adapte, mais sachez qu'il est meilleur avec les entiers qu'avec les autres réels (type float)...

Danger !

L'exemple précédent avec la variable d vous montre aussi qu'une fois que vous êtes passé chez les décimaux, vous ne revenez pas au type des entiers !

Remarque : le nom "float" (pour "flottant") provient du fait que la virgule dans un nombre décimal est vu comme un objet qui "flotte" dans un entier.

2 Les tableaux

Outil central de l'informatique, la programmation avec les tableaux de Python est très simple.

2.1 Définition

En python, on écrit les tableaux avec `np.array` et le texte entre crochets : `[` et `]`, avec les éléments séparés par des virgules.

à taper dans le shell

```
A=np.array([5,"choucroute", 9,"lasagne"])
A[1]
A[0]
A[-1]
A[-2]
len(A)
A[1]="paella"
A
```

Comme vous pouvez le voir, on peut facilement modifier les valeurs dans un tableau instructions suivantes :

dans le shell

```
A=np.array(["café",7,"cacao",3])
A+1
B=np.array([2,2,5])
C=np.array([1,4,3])
B+C
A+np.array([1])
A+np.array(["lait",2,"thé",5])
2*A
A[0:2]
A[3:5]
```

Soyez très attentifs aux deux derniers exemples, qui permettent d'extraire des parties de liste....

Remarquez enfin qu'on peut créer des listes vides :

dans le shell

```
A=np.array([])
len(A)
```

Exercice 1
Ecrire une fonction `permute(L)` qui échange le premier et le dernier élément d'un tableau `L`.

Par exemple, `permute(np.array([0,2,3,8]))` devra répondre `np.array([8,2,3,0])`

> Pas trop méchant. Il faut stocker, comme dans l'exercice avec caramel et chocolat, la valeur du premier élément dans une variable temporaire avant de faire l'échange.
>
> ```
> def permute(L):
>     temp=L[0]
>     L[0]=L[-1]
>     L[-1]=temp
>     return L
> ```

Exercice 2
Ecrire une fonction `supprime(L,n)` qui prend comme argument un tableau `L` et un entier n et qui répond la liste `L` sans l'élément qui avait la place n.

Par exemple si votre tableau de départ était `{np.array([0,4,5,6,2])` et que n vaut 2, la liste renvoyée doit être `np.array([0,4,6,2])`.

Il faut créer une nouvelle liste, et y copier les éléments de la liste de départ en sautant l'élément d'indice n.

```
def supprime(L,n):
    M=np.array([])
    for k in range(len(L)):
        if k!=n:
            M=np.append(M,L[k])
    return M
```

La fonction `np.append` permet d'ajouter un élément à la fin d'une liste.
On teste par exemple avec

```
print(supprime(np.array([0,4,5,6,2]),2))
```

2.2 L'instruction "range"

L'instruction `range(a,b)` retourne un objet de type "range", constitué de $b-a$ éléments allant de a à $b-1$. Avec un troisième argument, on peut donner "le pas".
Testez les commandes suivantes (l'instruction `list()` permet ici de lire le contenu du range) :

dans le shell
```
A=range(1,10)
A
type(A)
list(A)
A[1]
A[10]
B=range(10)
list(B)
C=range(0,10,2)
list(C)
D=range(10,0,-1)
list(D)
```

3 La boucle for

3.1 Définition

boucle for
```
for variable in liste:
    instructions
    ...
    instructions
suite du programme
```

Ce bloc est très proche de la boucle `while` : il consiste à éxécuter le bloc d' `instructions` (indenté) pour chaque valeur de la `variable` qui parcourt l'itérateur (comprenez la `liste`) qui peut être aussi bien une "vraie liste" qu'un couple, un mot ou un `range`.

Ecrire par exemple le script suivant dans l'éditeur et l'exécuter.

dans l'éditeur
```
for k in range(1,10):
    print(k)
```

3.2 Liste quelconque :

La liste n'est pas forcément qu'une liste de nombres :

dans l'éditeur
```
L=np.array(["John", "Paul","George","Ringo"])
for k in L:
    print(k+" était un membre des Beatles.")
```

3.3 Imbrication de boucles :

On peut inclure une boucle for dans une boucle for, pour faire lire une deuxième liste.
C'est le même principe que les listes de nombres à double indice qu'on a rencontré en cours pour les sommes doubles. Par exemple :

dans l'éditeur

```
L=np.array([1,2,3,4])
M=np.array([1,2,3,4])
for i in L:
    print("valeur de i=",i)
    print("Somme avec j :")
    for j in M:
        print(i+j)
```

4 Commenter vos programmes !

Nous allons écrire des programmes de plus en plus complexes, ce qui nécessite un peu d'organisation.
Dans l'éditeur, quand vous écrivez vos programmes, il faut prendre l'habitude d'ajouter des commentaires en les introduisant d'un #. Cela permet d'une séance à l'autre de se rappeler ce qu'on avait voulu faire, et ça permet aussi d'expliquer à un lecteur ce que vous avez programmé

```
1  def racine(a,b,c) : #Ce programme donne le nombre de racines réelles de ax²+bx+c
2      delta=b*b-4*a*c #j'enregistre delta pour me simplifier la vie
3      if delta > 0 : #c'est le cas deux racines
4          print("il y a deux racines réelles")
5      elif delta== 0 : #ne pas oublier le double égal !
6          print("une racine double")
7      else :
8          print("pas de racine réelle")
9
10 #Avec un double dièse, ça fait un Titre souligné dans Pyzo!
11
12 ##Un programme qui ne sert à rien :
13 def rien():
14     return
15
16 ##Un programme qui dit bonjour :
17 def salut():
18     return("Bonjour !")
```

De même, tâchez de donner des noms de variables et de fonctions qui soient explicites, comme dans le programme ci-dessus. Gardez toujours en mémoire que quelqu'un d'autre que vous devra peut-être lire et comprendre votre code !

5 Boucle for, somme et liste : quelques exercices

Exercice 3

1. Reprise du TP 1 : écrire un programme qui calcule la somme $\sum_{k=0}^{n} k$ en utilisant la boucle `for`
2. Ecrire une fonction qui prend comme argument un entier n et qui retourne $\sum_{k=0}^{n} k^2$. Vérifiez que cela correspond bien à $\frac{n(n+1)(2n+1)}{6}$

1.
```
def somme1(n):
    S=0
    for k in range(n+1):
        S=S+k
    return S
```
Cette structure est à connaître par coeur ! C'est la structure de base pour faire des sommes en python.
Vous constatez qu'on s'arrête à n en utilisant `range(n+1)` car le dernier élément n'est pas inclus dans le range.

2.
```
def somme2(n):
    S=0
    for k in range(n+1):
        S=S+k**2
    return S
```
Vous pouvez vérifier avec
```
n=5
print(somme2(n))
print(n*(n+1)*(2*n+1)/6)
```

Exercice 4

Ecrire une fonction `moyenne` qui prend comme argument une liste de nombre et retourne la moyenne des éléments de la liste.
On pourra la tester sur la liste $[1, 3, 4, 8, 6, 2, 9]$ dont la moyenne est environ $4,71$

Il faut additionner les éléments de la liste, puis diviser par le nombre d'éléments. On utilise le fait que l'instruction `for` permet de parcourir directement les éléments d'une liste.

```
def moyenne(L):
    S=0
    for k in L:
        S=S+k
    return S/len(L)
```

On teste par exemple avec

```
print(moyenne(np.array([1,3,4,8,6,2,9])))
```

Exercice 5 On considère la somme double $S_n = \sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} i + j$

1. Vérifiez (mathématiquement) que cette somme vaut $n^2(n+1)$
2. Ecrire une fonction qui prend en argument un nombre n, et qui calcule $\sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} i + j$.
3. Vérifier votre programme avec la formule que vous avez démontré à la première question.
4. Trouvez le plus petit n tel que $S_n \geq 5000$.

1. On peut écrire
$$\begin{aligned} S_n &= \sum_{i=1}^{n}\sum_{j=1}^{n} i + j \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} i + \sum_{i=1}^{n}\sum_{j=1}^{n} j \\ &= \sum_{i=1}^{n} ni + \sum_{j=1}^{n}\sum_{i=1}^{n} j \\ &= n\frac{n(n+1)}{2} + n\frac{n(n+1)}{2} \\ &\boxed{= n^2(n+1)} \end{aligned}$$
2. On peut faire une double boucle `for` imbriquée :
3.
```
def somme_double(n):
    S=0
```

```
    for i in range(1,n+1):
        for j in range(1,n+1):
            S=S+i+j
    return S
```

4. On teste avec

```
n=5
print(somme_double(n))
print(n**2*(n+1))
```

5. On peut faire une boucle `while` pour trouver le plus petit n tel que $S_n \geq 5000$:

```
n=1
while somme_double(n)<5000:
    n+=1
print(n)
```

Exercice 6 Soit $S_n = \sum_{0 \leq i \leq j \leq n} 2^{i+j}$

1. Ecrire un programme qui calcule S_n. Votre programme doit répondre par exemple $S_2 = 35$
2. Déterminez une formule donnant S_n et vérifiez la avec votre programme.
3. Construisez une liste qui contienne les 100 premières valeurs de la suite S_n dans l'ordre croissant, puis inversez-la pour les obtenir dans l'ordre décroissant (interdiction d'utiliser la fonction built-in reverse !)

1. On commence par écrire :

$$S_n = \sum_{i=0}^{n} \sum_{j=i}^{n} 2^{i+j}$$

On peut faire une double boucle `for` imbriquée en prenant garde à la condition $i \leq j$:

```
def somme_double2(n):
    S=0
    for i in range(n+1):
        for j in range(i,n+1):
            S=S+2**(i+j)
    return S
```

2. On peut écrire

$$\begin{aligned}
S_n &= \sum_{i=0}^{n}\sum_{j=i}^{n} 2^{i+j} \\
&= \sum_{i=0}^{n} 2^i \sum_{j=i}^{n} 2^j \\
&= \sum_{i=0}^{n} 2^i \left(2^i + 2^{i+1} + \cdots + 2^n\right) \\
&= \sum_{i=0}^{n} 2^i \left(2^i \frac{2^{n-i+1}-1}{2-1}\right) \\
&= \sum_{i=0}^{n} 2^{2i} \left(2^{n-i+1}-1\right) \\
&= \sum_{i=0}^{n} 2^{n+i+1} - \sum_{i=0}^{n} 2^{2i} \\
&= 2^{n+1}\frac{2^{n+1}-1}{2-1} - \frac{2^{2(n+1)}-1}{2^2-1} \\
&\boxed{= 2^{n+1}(2^{n+1}-1) - \frac{2^{2(n+1)}-1}{3}}
\end{aligned}$$

3. On peut créer la liste avec une boucle `for`, puis la renverser avec une autre boucle `for` :

```
L=np.array([])
for n in range(100):
    L=np.append(L,somme_double2(n))
M=np.array([])
for k in range(len(L)-1,-1,-1):
    M=np.append(M,L[k])
print(M)
```