TP 1-Le langage Python : rappels

1 Avant de commencer

1.1 Enregistrez votre travail régulièrement !

1.2 Commentaires

Beaucoup de TP dureront plusieurs séances ; il est donc important, en plus de sauvegarder votre travail, que votre code soit « propre » :

- donnez des noms explicites à vos fonctions et vos variables ;
- commentez votre code : la commande # permet d'apposer des commentaires (les expressions qui suivent # sont ignorées par Python).

Exemple 1

```
# Exemple 1

def exemple1(x)    # ceci est un commentaire
    return 2*x
```

1.3 Aide

En cas de doute, vous pouvez consulter l'aide de Python grâce à la commande (dans la console) :

```
help(fonction)
```

où `fonction` est le nom de la commande dont on cherche les fonctionnalités.

2 La syntaxe Python

2.1 Opérations de base, opérateurs logiques

Opérations de base

+	−	*	**	/
Addition	Soustraction	Multiplication	Puissance	Division

Opérateurs logiques et de comparaison

- Opérateurs de comparaison.

==	>	<	>=	<=	! =
Égal	Supérieur	Inférieur	Supérieur ou égal	Inférieur ou égal	Différent

- Opérateurs logiques.

and	or	not
Et	Ou	Négation

Les opérateurs de comparaison et les opérateurs logiques renvoient un *booléen* c'est-à-dire **True** ou **False**.

- Taper dans la console et observer le résultat.

```
2 < 3
```

- Taper dans la console et observer le résultat.

```
2 == 3
```

2.2 Structures de contrôle

2.2.1 Variables et affectation

Variables et affectation

Le symbole « = » sert à affecter une valeur à une variable.

- Taper dans la console et observer le résultat.

```
x = 2
```

- Que fait le code suivant ?

```
a,b = 2,3
```

- Que fait le code suivant ?

```
a = 2
b = 3
a,b=b,a
```

2.2.2 Les fonctions

Les fonctions

La syntaxe d'une fonction est la suivante :

```
def ma_fonction(parametre_1,\dots,parametre_n):
        instructions
        return resultat_1,...,resultat_p
```

où

- `ma_fonction` est le nom de la fonction,
- `parametre_1,...,parametre_n` sont les paramètres de la fonction (il peut y en avoir zéro, un ou plusieurs),
- `instructions` est une succession d'instructions qui vont être exécutées à l'appel de la fonction,
- `resultat_1,...,resultat_p` sont les résultats retournés par la fonction (il peut y en avoir zéro, un ou plusieurs).

Exemple 2

La fonction suivante :

```
def f(x):
        return x**2+2*x+3
```

est nommée f ; elle prend en paramètre une variable x et renvoie la valeur de $x^2 - 2x + 3$.

- Créer une fonction nommée `moyenne` qui prend en paramètres deux variables et renvoie leur moyenne.

2.2.3 Instructions condiltionnelles

Instructions conditionnelles

Les instructions conditionnelles permettent de mettre des conditions à l'exécution d'instructions. La syntaxe Python est la suivante :

```
if condition1 :
        instruction1.1
        instruction1.2
        ...
elif condition2 :
        instruction2.1
        instruction2.2
        ...
elif condition3 :
        ...
...
else :
        instructions_else
```

Le code ci-dessus exécute :

- les instructions `instruction1.1`, `instruction1.2`,...**si** la condition `condition1` est satisfaite ;
- les instructions `instruction2.1`, `instruction2.2`,...**si** la condition `condition2` est satisfaite ;
- etc ;
- les instructions `instructions_else` **si** aucune des conditions précédentes n'est satisfaite.

Si l'on veut qu'aucune instruction ne soit exécutée lorsque qu'une condition est réalisée, on utilise la commande `pass` à la place des instructions.

Exemple 3

On suppose qu'une variable x est déjà affectée.

```
if x < 0 :
        print(''signe strictement negatif'' '')
elif x == 0 :
        pass
else :
        print(''signe strictement positif'' '')
```

Le programme ci-dessus :

- *affiche « signe strictement négatif »si la variable x est strictement négative ;*
- *ne fait rien si la variable x vaut* 0 *;*
- *affiche « signe strictement positif »dans les autres cas.*

- Créer une fonction nommée f qui, pour un réel donné, renvoie sa valeur absolue.

- Vérifier votre fonction en calculant $f(2)$, $f(0.5)$, $f(-3)$ et $f(-5)$.

2.2.4 Les boucles

Boucle `for`

La syntaxe est la suivante :

```
for element in iterable :
        instruction
```

où

- `iterable` est une liste, un range ...
- `element` est une variable qui prend successivement les valeurs contenues dans `iterable`,
- `instruction` est un bloc d'instructions qui va être exécuté successivement pour chaque valeurs de `element`.

Exemple 4

La commande

```
for i in range(0,10,2):
        print(i)
```

va exécuter l'instruction `print(i)` *successivement pour chaque valeur* i *de* `range(0,10,2)`*. On obtient donc :*

```
0
2
4
6
8
```

- Entrer les instructions suivantes :

```
n = int(input('Donner unevaleur de n:'))
S = 0
for k in range(1,n+1):
        S = S + k
print(S)
```

- Tester pour différentes valeur de n. Expliquer ce que renvoie ce programme.

- Modifier le programme ci-dessous pour qu'il calcule $\sum_{k=0}^{n}(k^2-3k+1)$.

Boucle `while`

La boucle `while` permet d'exécuter des instructions tant qu'une condition est respectée. La syntaxe est la suivante :

```
while condition :
        instructions
```

où

- `condition` est une condition (c'est-à-dire une expression qui prend la valeur vraie ou fausse),
- `instructions` est un bloc d'instructions qui va être exécuté tant que `condition` est vraie.

On considère une suite $(u_n)_{n\in\mathbb{N}}$ telle que

$$\forall n \in \mathbb{N},\ e^{-\sqrt{u_n}} \leqslant u_n - n \leqslant e^{-\sqrt{n}}.$$

- Compléter le programme Python suivant afin qu'il affiche un entier n pour lequel $u_n - n$ est inférieur ou égal à 10^{-4}.

```
n=0
while -----  :
        n = -----
print(n)
```

3 Bibliothèques

3.1 Importation de bibliothèques

Un grand nombre de fonctions ou programmes particulièrement utiles dans certains domaines sont prédéfinis et rangés dans des modules externes. C'est le cas par exemple des fonctions racine carrée, exponentielle, ...qui sont déjà définies et rangées dans le module maths. Ces modules peuvent ensuite être regroupés dans ce que l'on appelle des librairies (aussi appelées package ou paquet ou encore bibliothèques). La multitude de librairies disponibles sur Python est ce qui en fait une de ses grandes forces.

Afin de pouvoir accéder à ces fonctions il faut au préalable importer ces modules et/ou librairies à l'aide d'une des commandes suivantes :

```
from ...import *  ou  import ...as ....
```

3.2 Numpy

La biblothèque Numpy contient le module et permet donc d'accèder à toutes les fonctions mathématiques usuelles ainsi qu'aux valeurs de certaines constantes.

Dans toute la suite, on considère que la bibliothèque Numpy a été importée sous le diminutif `np` grâce à la commande :

```
import numpy as np
```

Constantes et fonction mathématiques classiques

- `np.e`, `np.pi` : les constantes e et π.
- `np.exp`, `np.log`, `np.sqrt`, `np.abs`, `np.floor` : les fonctions exponentielle, logarithme, racine carrée, valeur absolue et partie entière. Ces fonctions peuvent s'appliquer à des variables numériques ou vectorielles (vecteurs ou matrices).

Le principal avantage de la bibliothèque Numpy par rapport au module Math est qu'elle permet la manipulation de vecteurs et de matrices.

Définir une matrice ou un vecteur

1. Pour définir un vecteur ou une matrice en entrant les coordonnées on utilise la commande `np.array` (voir l'exemple ci-dessous).
2. La commande `np.zeros([n,p])` permet de créer la matrice nulle de taille $n \times p$.
   La commande `np.ones([n,p])` permet de créer la matrice de taille $n \times p$ dont tous les coefficients valent 1.
   La commande `np.eye(n)` permet de créer la matrice identité de taille n.
3. La commande `np.linspace(a,b,n)` permet d'obtenir un vecteur de taille `n` dont les coordonnées sont régulièrement espacées de `a` à `b`.
4. La commande `np.arange(a,b,c)` permet d'obtenir un vecteur dont les coordonnées sont espacées de `a` à `b` par pas de `c`.

Exemple 5

1. *La commande* `np.array([[1,1,0],[1,2,3],[0,1,0]])` *crée la matrice :*
$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 3 \\ 0 & 1 & 0 \end{pmatrix}.$$
2. *La commande* `np.zeros([2,2])` *crée la matrice :*
$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$
3. *La commande* `np.linspace(2,12,5)` *crée le vecteur :*
$$(2, 4.5, 7, 9.5, 12).$$
4. *La commande* `np.arange(0, 1.3,0.2)` *crée le vecteur :*
$$(0, 0.2, 0.4, 0.6, 0.8, 1, 1.2).$$

▸ Créer les vecteurs suivantes sans rentrer les coefficients à la main :
1. $a = [1.1, 2.1, \ldots, 99.1, 100.1]$,
2. $b = [2, 4, 6, \ldots, 96, 98]$,
3. $c = [100, 98, 96, \ldots, 4, 2]$,
4. d le vecteur de taille 50 ne contenant que des 1.

- Créer les matrices suivantes sans rentrer les coefficients à la main (sauf pour la A) :
  1. $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$,
  2. B la matrice à 2×3 ne contenant que des 1,
  3. $C = I_5$,

Manipulation des matrices

Soit `M` une matrice préalablement définie dans Python.

1. Soit `M` une matrice préalablement définie dans Python.
   (a) La commande `M[i,j]` donne le coefficient de la i-ième ligne et j-ième colonne de `M`.
   (b) La commande `M[i,:]` donne la i-ième ligne de `M`.
   (c) La commande `M[:,j]` donne la j-ième colonne de `M`.

   Attention, en python la numérotation des lignes et des colonnes commence à zéro !
2. Soient M et N deux matrices préalablement définies dans Python et `a` un réel.
   (a) `M+N` : somme des matrices (qui doivent donc être de la même taille).
   (b) `M*N` : produit coefficients par coefficients des matrices (qui doivent donc être de la même taille).
   (c) `M+a` : ajout de `a` à tous les coefficients de `M`.
   (d) `M*a` : multiplication par `a` de tous les coefficients de `M`.
   (e) `M**a` : passage à la puissance `a` de tous les coefficients de `M`.
   (f) `np.dot(M,N)` : produit matriciel de `M` et de `N` (à condition que les tailles des matrices le permettent).
   (g) `np.shape(M)` : taille de la matrice.
   (h) `np.transpose(M)` : transposée de la matrice.

- Créer le vecteur de taille 50 ne contenant que des 5 (sans rentrer les coefficients à la main).

- Créer le vecteur $x = [5, 7, 2, 14, 8, 1]$ et écrire les commandes permettant d'extraire l'élément 7. Écrire une commande pour remplacer le 8 par un 3.

- Créer la matrice suivante sans rentrer les coefficients à la main :

$$D = \begin{pmatrix} 2 & 3 & 3 \\ 3 & 2 & 3 \\ 3 & 3 & 2 \end{pmatrix}.$$

- Créer la matrice $E = \begin{pmatrix} 1 & -2 & 3 \\ -4 & 5 & -6 \\ 7 & -8 & 9 \end{pmatrix}$ et donner la commande permettant d'extraire la deuxième ligne.

- Effectuer le produit matriciel de la matrice E avec la matrice D.

3.3 Librairie `numpy.linalg`

La librairie `numpy.linealg`

1. Exemple d'importation :

```
import numpy.linalg as al
```

2. Commandes utiles :
   - `al.inv(M)` : inverse de `M` lorsqu'elle est inversible.
   - `al.matrix_power(M,n)` : calcule la puissance `n`-ième de `M` lorsque `M` est carrée.
   - `al.solve(A,B)` : résolution du système $Ax = B$.

- Soit $F = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Calculer avec Python A^n pour $n = 2, 3, 4$ et 5.

- Conjecturer la forme de A^n et démontrer votre conjecture par récurrence.

- On considère le système suivant :

$$\begin{cases} x & + & 2y & = & 1 \\ 3x & + & 5y & = & 2 \end{cases}$$

Résoudre ce système à la main puis vérifier votre résultat avec Python.