Une (très) courte introduction à Python

A. Ridard

Il s'agit d'un langage de programmation de haut-niveau permettant :

- une programmation structurée
- une programmation orientée objet (non présentée dans ce document)

I. Pour bien commencer

1. Types de base

Les types de base sont :

- les **entiers** (int) : 421, 0, -192
- les **flottants** (float) : 3.14, 0.0, -1.7e-6
- les **booléens** (bool) : True et False

Les opérateurs numériques (applicables aux entiers et aux flottants) sont :

Python (Opérateurs numériques).

+ , − , *	addition , soustraction , multiplication
/	division (le résultat est un flottant)
**	exponentiation
// , %	quotient , reste de la division euclidienne

Les opérateurs booléens sont :

Python (Opérateurs booléens).

a and b	conjonction
a or b	disjonction
not a	négation
in	test de présence
< , > , <= , >= , == , !=	comparateurs

2. Listes (un premier conteneur)

Une liste est une séquence (ordonnée) de valeurs pouvant être de types différents et accessibles par leur index qui **commence à 0** :

Python (Indexation des séquences).

Pour les listes, tuples, chaines de caractères, ...

len(seq)	longueur de seq
seq[i]	l'élément d'index i de seq
seq[0]	le premier élément de seq
seq[len(seq)-1] ou seq[-1]	le dernier élément de seq
seq[a : b : p]	les éléments d'index a , a+p , ... , a+kp < b

Une liste fait partie des conteneurs [1] modifiables auxquels on peut appliquer les fonctions/méthodes [2] suivantes :

[1]. On en verra d'autres : les tuples, chaînes de caractères et dictionnaires

[2]. La distinction entre fonctions et méthodes sera faite lorsque l'on introduira la Programmation Orientée Objet

Python (Opérations sur les conteneurs).

min(cont) , max(cont)	minimum , maximum de cont
sum(cont)	somme des éléments de cont
+ , *	concaténation , duplication
cont.count(val)	nombre d'occurrences de val dans cont

Opérations spécifiques aux listes

lst.insert(idx,val)	insertion de val dans lst à la position idx
del(lst[idx])	suppression dans lst de l'élément d'index idx

3. Structure conditionnelle : SI

La structure conditionnelle permet à un bloc d'instructions de n'être exécuté que si une condition prédéterminée est réalisée. Dans le cas contraire, les instructions sont ignorées et la séquence d'exécution continue à partir de l'instruction qui suit immédiatement la fin du bloc.

Python (Structure conditionnelle).

if *b1* **:**
    |*bloc d'instructions si b1 est vrai*
elif *b2* **:**
    |*bloc d'instructions si b1 est faux et b2 vrai*
else :
    |*bloc d'instructions sinon*

Plusieurs elif possibles

4. Boucles : POUR et TANT QUE

a. Boucle itérative : POUR

La boucle itérative sert à répéter un bloc d'instructions un nombre prédéfini de fois.

Python (Boucle itérative).

for *i* in range($\overbrace{[a,]}^{\text{0 par défaut}}\underbrace{b}_{\text{exclu}}\overbrace{[,p]}^{\text{1 par défaut}}$) **:**
    |*bloc d'instructions*

i varie de a à b exclu avec un pas de p

b. Extension du for

Dans une boucle itérative, la variable d'itération peut prendre ses valeurs dans un conteneur.

Python (Extension du for).

for *i* in cont **:**
    |*bloc d'instructions*

i prend ses valeurs dans cont

c. Boucle conditionnelle : TANT QUE

La boucle conditionnelle sert à répéter un bloc d'instructions tant qu'une certaine condition est réalisée.

Python (Boucle conditionnelle).

while *b* **:**	instructions éxécutées
|*bloc d'instructions*	tant que b est vrai

5. Chaînes de caractères (un deuxième conteneur)

Comme une liste, une chaîne de caractères est une séquence [3] (ordonnée) ; les caractères étant accessibles par leur index qui **commence à 0**.

Exemples :

- '' (chaîne vide)
- 'Hei!' (tous les caractères sont autorisés)
- "L'école HEI" (pour considérer l'apostrophe ', la chaîne doit être délimitée par ")

Une chaîne de caractères n'est pas modifiable

```
>>> ch='HEC'
>>> ch[2]='I'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Les fonctions/méthodes spécifiques aux chaînes de caractères sont :

Python (Chaines de caractères).

var = eval(ch)	conversion d'une chaine de caractères en entier, flottant, liste, ...
ch = str(var)	conversion d'un entier, flottant, liste, ... en chaine de caractères
" − ".join(['Jean','Pierre'])	donne 'Jean−Pierre'
"2;5.1;12" . split(';')	donne ['2','5.1','12']

(" − " : chaine de jointure ; ['Jean','Pierre'] : séquence de chaines ; ';' : chaine de séparation)

Conversions

```
>>> eval('421')
421
>>> eval('[421,3.14,True]')
[421, 3.14, True]
>>> str(True)
'True'
```

[3]. L'encadré **Indexation des séquences** est donc valable pour les chaînes de caractères

6. Fichiers textes

a. Des caractères particuliers

- \n permet de faire un saut de ligne
- \t permet de faire une tabulation
- \" permet de mettre " dans une chaîne délimitée par "
- \' permet de mettre ' dans une chaîne délimitée par '

```
>>> ch1="Voici une chaîne de caractères \nqui s'affichera sur deux lignes."
>>> ch1
"Voici une chaîne de caractères \nqui s'affichera sur deux lignes."
>>> print(ch1)
Voici une chaîne de caractères
qui s'affichera sur deux lignes.
>>> ch2="Voici une chaîne avec une\ttabulation"
>>> ch3="Le caractère \\ s'appelle antislash ou backslash pour les anglophones"
>>> ch4="Voici des \" dans une chaîne délimitée par des \""
>>> ch5='Voici des \' dans une chaîne délimitée par des \''
>>> print(ch2)
Voici une chaîne avec une	tabulation
>>> print(ch3)
Le caractère \ s'appelle antislash ou backslash pour les anglophones
>>> print(ch4)
Voici des " dans une chaîne délimitée par des "
>>> print(ch5)
Voici des ' dans une chaîne délimitée par des '
```

b. Lecture - Ecriture - Ajout

Python (Fichiers).

f=open('chemin\ nomfichier.txt', 'r' 'w' 'a')	ouverture en	lecture écriture ajout
f.readline()	lecture d'une ligne (caractère de fin de ligne compris) à partir de l'endroit où l'on se trouve dans le fichier	
f.readlines()	séquence des lignes du fichier	
f.write('texte à écrire')	écriture à partir de l'endroit où l'on se trouve dans le fichier	
f.close()	fermeture	

Lecture avec readline

```
>>> f=open('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info\\Lecture.txt','r')
>>> f.readline()
'Pour lire ce texte avec Pyhon,\n'
>>> f.readline()
'il existe différentes méthodes'
>>> f.readline()
''
>>> f.close()
```

- Ne pas oublier le \\ dans le chemin
- Le fichier doit exister

Lecture avec readlines

```
>>> f=open('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info\\Lecture.txt','r')
>>> lst=f.readlines()
>>> lst
['Pour lire ce texte avec Pyhon,\n', 'il existe différentes méthodes']
>>> x=''
>>> for ligne in lst:
...     x=x+ligne
...
>>> x
'Pour lire ce texte avec Pyhon,\nil existe différentes méthodes'
>>> print(x)
Pour lire ce texte avec Pyhon,
il existe différentes méthodes
>>> f.close()
```

Ecriture

```
>>> f=open('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info\\Ecriture.txt','w')
>>> f.write('Ce texte sera écrit\nsur deux lignes')
35
>>> f.close()
```

Si le fichier existe, son contenu est effacé, sinon le fichier est créé

Ajout (en fin de fichier)

```
>>> f=open('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info\\Ecriture.txt','a')
>>> f.write('Attention au retour à la ligne')
30
>>> f.close()
```

7. Gestion des répertoires

Module (**os** : gestion des répertoires).

import os	Les fonctions seront préfixées par os.
os.getcwd()	Quel est le répertoire de travail ?
os.chdir('chemin')	Changement du répertoire de travail
os.listdir()	Liste des fichiers dans le répertoire de travail

```
>>> import os
>>> os.getcwd()
'C:\\Users\\aridard2\\Documents\\WinPython-32bit-3.3.3.3\\python-3.3.3\\Scripts'
>>> os.chdir('C:\\Users\\aridard2\\Desktop\\2015_2016\\Info')
>>> f=open('Ecriture.txt','r')
>>> lst=f.readlines()
>>> x=''
>>> for ligne in lst:
...     x=x+ligne
...
>>> print(x)
Ce texte sera écrit
sur deux lignesAttention au retour à la ligne
>>> f.close()
```

II. Tableaux numpy

Module (**numpy** : gestion des tableaux).

import numpy as np	Les fonctions seront préfixées par np.
np.array(liste)	Création d'un tableau à partir d'une liste
np.linspace(min,max],nbPoints)	Création d'un tableau 1D
np.arange(min,max[,pas)	Création d'un tableau 1D
np.fromfunction(fct,(nbLig,nbCol))	Création d'un tableau 2D
np.where(condition,val si vrai, val sinon)	Modification d'un tableau
tableau.reshape(nbElts sur axe 0,...)	Permet de reformer un tableau
tableau.shape	Nombre d'éléments sur chaque axe
tableau.dtype	Type commun des éléments d'un tableau

Les opérations et les fonctions s'appliquent terme à terme

Dans les exemples de cette section, le module numpy a déjà été importé avec l'instruction *import numpy as np*.

Construction à partir d'une liste

```
>>> tab1=np.array([1.,2.,3.])
>>> tab1.shape
(3,)
>>> tab1.dtype
dtype('float64')
>>> tab2=np.array([[1,2,3],[4,5,6]])
>>> tab2.shape
(2, 3)
>>> tab2.dtype
dtype('int32')
```

Contrairement aux listes, tous les éléments doivent être du même type

Attention

```
>>> np.array([421,3.14,True])
array([ 421.  ,    3.14,    1.  ])
>>> np.array([421,3.14,'True'])
array(['421', '3.14', 'True'],
dtype='<U4')
```

Construction à partir d'une fonction

```
>>> def fct(i,j):
...     return(j>=i)
...
>>> np.fromfunction(fct,(3,3))
array([[ True,  True,  True],
[False,  True,  True],
[False, False,  True]], dtype=bool)
```

Construction à l'aide de arange ou linspace (1D)

```
>>> tab1=np.arange(0,12,2)
>>> tab1
array([ 0,  2,  4,  6,  8, 10])
>>> tab1.dtype
dtype('int32')
>>> tab2=np.linspace(0,10,6)
>>> tab2
array([  0.,   2.,   4.,   6.,   8.,  10.])
>>> tab2.dtype
dtype('float64')
```

- range et arange ne génèrent pas des objets de même type (classe)
- Contrairement à range, arange autorise les flottants

Attention

```
>>> type(range(0,12,2))
<class 'range'>
>>> type(np.arange(0,12,2))
<class 'numpy.ndarray'>
>>> np.arange(0,3,0.5)
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5])
>>> range(0,3,0.5)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer
```

Modification de la forme

```
>>> tab1=np.arange(0,12,2)
>>> tab1
array([ 0,  2,  4,  6,  8, 10])
>>> tab2=tab1.reshape(2,3)
>>> tab2
array([[ 0,  2,  4],
[ 6,  8, 10]])
```

Modification multiple du contenu

```
>>> tab=np.array([[1,2,3],[4,5,6]])
>>> tab[1:]=0
>>> tab
array([[1, 2, 3],
[0, 0, 0]])
```

Modification en parallèle du contenu

```
>>> tab=np.array([[1,2,3],[4,5,6]])
>>> tab[1:]=[1,2,3]
>>> tab
array([[1, 2, 3],
[1, 2, 3]])
```

Modification conditionnelle du contenu

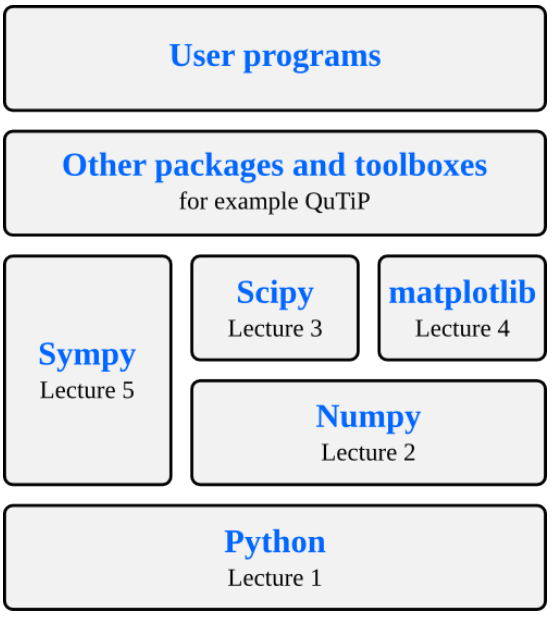
```
>>> tab=np.array([[1,2,3],[4,5,6]])
>>> tab=np.where(tab%2==0,tab,0)
>>> tab
array([[0, 2, 0],
[4, 0, 6]])
```

Les opérations et les fonctions s'appliquent terme à terme.

Traitement vectoriel

```
>>> tab=np.arange(1,7).reshape(2,3)
>>> tab
array([[1, 2, 3],
[4, 5, 6]])
>>> tab%2==1
array([[ True, False,  True],
[False,  True, False]], dtype=bool)
>>> tab+1
array([[2, 3, 4],
[5, 6, 7]])
>>> tab**2
array([[ 1,  4,  9],
[16, 25, 36]])
>>> np.sin(tab)
array([[ 0.84147098,  0.90929743,  0.14112001],
[-0.7568025 , -0.95892427, -0.2794155 ]])
```

III. Les modules utilisés en Maths : numpy, scipy, matplotlib et sympy



Les lectures suggérées sont ici

Prenez un peu de temps pour la lecture 0 du site précédent, vous y trouverez entre autres des informations sur l'installation d'Anaconda et l'utilisation de Jupyter notebook...

PYTHON EN BREF

Python (Types de base).

int , float , bool — entier , flottant , booléen

Python (Opérateurs numériques).

+ , − , * — addition , soustraction , multiplication
/ — division (le résultat est un flottant)
** — exponentiation
// , % — quotient , reste de la division euclidienne

Python (Conteneurs).

Non modifiables

str	'hei' ou "hei"	chaines de caractères
tuple	(421,3.14,'hei')	t-uplets

Modifiables

list	[421,3.14,'hei']	listes
dict	{clé : valeur}	dictionnaires

Python (Opérations sur les conteneurs).

min(cont) , max(cont) — minimum , maximum de cont
sum(cont) — somme des éléments de cont
+ , * — concaténation , duplication
cont.count(val) — nombre d'occurrences de val dans cont

Opérations spécifiques aux listes

lst.insert(idx,val) — insertion de val dans lst à la position idx
del(lst[idx]) — suppression dans lst de l'élément d'index idx

Python (Chaines de caractères).

var = eval(ch) — conversion d'une chaine de caractères en entier, flottant, liste, ...
ch = str(var) — conversion d'un entier, flottant, liste, ... en chaine de caractères

"−".join(['Jean','Pierre']) — donne 'Jean−Pierre'
(chaine de jointure ; séquence de chaines)

"2 ;5.1 ;12" .split(";") — donne ['2','5.1','12']
(chaine de séparation)

Python (Indexation des séquences).

Pour les listes, tuples, chaines de caractères, ...

len(seq) — longueur de seq
seq[i] — l'élément d'index i de seq
seq[0] — le premier élément de seq
seq[len(seq)-1] ou seq[-1] — le dernier élément de seq
seq[a : b : p] — les éléments d'index a , a+p , ... , a+kp < b

Python (Affichage/Saisie).

print(ch,"contient",len(ch),"caractères") — affichage
e=input("message facultatif") — affectation à e de la chaine de caractères saisie

Python (Déclaration de fonction).

```
def nomfct (a,b) :
    bloc d'instructions
    return(res)
```

avec a et b des paramètres éventuels

Python (Fichiers).

f=open('chemin\ nomfichier.txt' , 'r' / 'w' / 'a') — ouverture en lecture / écriture / ajout

f.readline() — lecture d'une ligne (caractère de fin de ligne compris) à partir de l'endroit où l'on se trouve dans le fichier

f.readlines() — séquence des lignes du fichier

f.write('texte à écrire') — écriture à partir de l'endroit où l'on se trouve dans le fichier

f.close() — fermeture

Python (Structure conditionnelle).

```
if b1 :
    |bloc d'instructions si b1 est vrai
elif b2 :
    |bloc d'instructions si b1 est faux et b2 vrai
else :
    |bloc d'instructions sinon
```

Plusieurs elif possibles

Python (Boucle itérative).

for *i* in range([*a*,] *b* [,*p*]) **:** — i varie de a à b exclu avec un pas de p
(0 par défaut ; 1 par défaut ; exclu)

```
    |bloc d'instructions
```

Extension du for aux conteneurs

for *i* in cont **:** — i prend ses valeurs dans cont

```
    |bloc d'instructions
```

Python (Boucle conditionnelle).

while *b* **:** — instructions éxécutées tant que b est vrai

```
    |bloc d'instructions
```