Département de Mathématiques 2018/2019
Module: Analyse numérique 1
Fiche de T.P. N°1

Initiation à PYTHON et SPYDER

Python est un langage de programmation créé par Guido van Rossum, dont la première version est sortie en 1991. Il permet de concevoir toutes sortes de programmes, comme des jeux, des logiciels, des progiciels, etc.
Il permet une programmation structurée (à l'aide de fonctions) et une programmation orientée objet (interaction de briques logicielles élémentaires appelées "objets").
Python est un langage interprété c'est à dire un programme est traduit en langage-machine au moment de son exécution ; en opposition avec les langages compilés (par exemple FORTRAN, C, C++), où le programme est traduit en langage-machine une fois pour toutes.
Python est un logiciel libre **(www.python.org)** et portable c'est à dire qu'il peut fonctionner sous différents systèmes d'exploitation (Windows, Linux, Mac OS) d'installation très simple.
Il est possible d'associer des bibliothèques à **Python** afin d'étendre ses possibilités. Par exemple:
Pour le calcul scientifique: NumPy, SciPy, Sympy,
Pour la visualisation :maplotlib
Pour la simulation : simPy;
Les programmes **Python** sont écrits en utilisant des mots usuels (très souvent de l'anglais) et des symboles mathématiques familiers. suivant les étapes
Taper le programme. Le sauvegarder dans un fichier nommé nom.py. L'exécuter dans l'interpréteur Python.
Anaconda est une plateforme gratuite développée par la société continuum analytics qui a l'avantage de rassembler tout le nécessaire pour l'utilisation scientifique de Python c'est à dire le langage Python,l'éditeur **Spyder** ainsi que ses modules scientifiques.
Spyder (**S**cientific **PY**thon **D**evelopment **E**nvi**R**onment) est un environnement de développement orienté vers un usage sientifique de Python avec les fenêtres consoles, Editeur, Explorateur de variables, Explorateur de fichiers, historique, ...
Lancer Spyder, dans Affichage/Volets sélectionner uniquement Editeur et console IPython (ou Python console)
Dans Outils/Préférences dans l'onglet répertoire de travail global définir le répertoire tpananum dans c:\windows. Puis relancer Spyder.

La "calculatrice" Python

Tester la console Python ou IPython en l'utilisant comme calculatrice.
A d d i t i o n et s o u s t r a c t i o n :
>>> 20+80
>>> 6-5
M u l t i p l i c a t i o n :
>>> 2*5
Puissance :
>>> 5**2
D i v i s i o ns :
>>> 23/3
>>> 2 3 . 0 / 3 . 0

>>>23//3 # Quotient de la division euclidienne
>>> 49%10 # Reste de la division euclidienne
Affichage du résultat à l'ecran :
>>> **p r i n t** (1 + 2 est une a d d i t i o n)
La priorité (ordre) des opérations est respectée (mêmes règles qu'en maths) :
1 parenthèses ()
2 puissance **
3 multiplication *, division et reste
4 addition + et substraction -
>>> 1 + 2 * 3
>>> (1 + 2) * 3
Remarque: La priorité des opérateurs // et % est la même que celle des opérateurs de multiplication et division. Les calculs de quotient et reste se font donc avant les additions et soustractions.
>>> print(10 + 20 // 3 + 42 % 5)
Un commentaire commence par le caractère # et s'étend jusqu'à la fin de la ligne.
>>> # Voici un commentaire
>>> 9+2 # en voici un autre
Les expressions booléennes
Deux valeurs possibles : False, True.
Opérateurs de comparaison : ==,!=, >, >=, <, <=.
>>> 2 > 8 # False
>>> 2 <= 8 < 15 # True
Opérateurs logiques: not, or, and.
>>> (3==3) or (9 >24) # True (des le premier membre)
>>> (9 > 24) and (3==3) # False (des le premier membre)
Les opérateurs logiques et de comparaisons sont à valeurs dans False, True.
Calcul approché
Les calculs avec des nombres décimaux ne sont pas exacts mais approchés. ce qui peut donner lieu à des résultats surprenants.
>>>1/3 # 17 chiffres sont conservés au maximum
0.3333333333333333
>>> print(1.0 - 0.00000000000000000000001) # erreur d'absoption
1.0
Expliquer les résultats des instructions suivantes:
>>> nbre=1000*1000*1000*1000*1000*1000
>>> nbreplus1=nbre+1
>>> nbreplus1-nbre
>>> nbreplus1=nbreplus1/1
>>> nbreplus1-nbre
# test d'égalité
>>> 0.1+0.2==0.3
False !!!!
>>> print(0.1+0.2-0.3)
5.551115123125783e-17
Comment est-ce possible?

>>> 0.1
0.1
En fait, Python n'affiche qu'une approximation décimale de la valeur stockée en binaire. Il affiche une valeur arrondie pour améliorer la lisibilité.
Le module decimal propose un moyen pratique de « voir » la valeur exacte stockée pour n'importe quel flottant.
>>> from decimal import Decimal
>>> Decimal(0.1)
Decimal('0.1000000000000000055511151231257827021181583404541015625') !!!!
Par contre, lorsqu'on travaille avec des nombres entiers, les résultats ne sont jamais arrondis.
En particulier, les nombres peuvent devenir aussi grands qu'on le souhaite!
>>> print(12345678987654321998999999999999999 * 98765432101234567899999999999999)
1219326319798811256645633290134984787658024444343310000000000001
Python prétend que ses entiers sont sans limites. Il peut, par exemple, calculer facilement un googol:
>>> 10**100
10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
En conclusion:

- Il ne faut pas oublier que la valeur stockée est une approximation de la fraction décimale d'origine.
- Il faut utiliser autant que possible des entiers plutôt que des nombres décimaux.
- Il faut eviter les tests d'égalité sur des nombres décimaux.

Les données et les variables
La variable est un nom donné à une valeur. Informatiquement, c'est une référence à une adresse mémoire.
Remarques :
Les noms des variables sont conventionnellement écrits en minuscule ; ils commencent par une lettre ou le caractère souligné(_), puis, éventuellement, des lettres, des chiffres ou encore du caractère _. Python est sensible à la casse.
Ils doivent être différents des mots réservés de Python :
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
L'affectation
On affecte une variable par une valeur en utilisant le signe = ; dans une affectation, le membre de gauche reçoit le membre de droite.
Remarque : Affecter n'est pas comparer !
l'affectation a un effet mais n'a pas de valeur :
>>>a=b
la comparaison a une valeur mais n'a pas d'effet :
>>> a==b
L'affectation peut avoir plusieurs formes
a=10 # forme de base
a+=1 # meme chose que a = a + 1, si a est deja connue;

b = c = 10 # affectations multiples
d, e, f = 1.5, 2.5, 3.5 # affectation de tuple
g, h = h, g # permutations des valeurs de g et h
m, n = [5,7] # affectation de liste (par position).

Remarque

Une variable garde en mémoire une information tant que l'interpréteur Python est ouvert. Si vous quittez le programme puis le redémarrez toutes vos variables seront effacées.

Type des données

A chaque type de données des opérations peuvent être associées. le type entier ou **int,** le type flottant ou **float,** le type chaine de caractère ou **str.**

Les types des variables seront visibles à laide de la fonction **type.**

```
>>> a=1
>>> type(a)
>>> b=a/2
>>> type(b)
>>> a=float(a)
>>> print(a)
>>> type(a)
>>> A="j'aime les maths"
>>> a=str(a)
>>> print(a)
>>> type(a)
>>> a+2
>>> A+a
>>> 20*A
```

Les entrées / sorties

L'instruction **input** permet de saisir une entrée au clavier. Comme c'est une évaluation, elle effectue un typage dynamique.

```
>>> n = input("Entrez un entier : ")
Entrez un entier : 7
>>> type(n)
<type 'int'>
>>> n = input("Entrez un entier : ")
Entrez un entier : 1.1
>>> type(n)
<type 'float'>
```

L'instruction print permet d'afficher des sorties à l'écran :

```
>>> a=2
>>> b=5
>>> print ("La somme a+b est : ", a+b)
La somme a+b est : 7
```

L'opérateur \n peut être utiliser pour afficher un retour à la ligne comme,

```
>>> print("a=",'\n',10)
```

L'opérateur % peut aussi être utilisé pour formater des chaînes par exemple:

```
>>> print("a= %5.3f "%2)
>>> print( "il faut manger %2d fruits et légumes par jour." %5)
```

Instructions conditionnelles

La syntaxe de IF - ELIF - ELSE

```
if {conditions}:
    {executer ce code}
elif {conditions}:
    {executer ce code}
elif {conditions}:
    {executer ce code}
else:
    {executer ce code}
```

Remarques :

Il est possible, après un IF de tester autant de conditions que l'on souhaite avec des ELIF (y compris 0) ; par contre, il ne peut y avoir qu'un seul ELSE à la fin.

Le caractère : est obligatoire à la fin de chaque ligne contenant IF, ELIF ou ELSE.

Trouver, par exemple, le minimum de deux nombres :

```
x, y =4, 3
# Ecriture classique :
if x<y:
    min=x
else:
    min=y
```

Il arrive souvent qu'il y ait deux résultats simples possibles après le test d'une condition. Il est dans ce cas un peu lourd d'utiliser la structure if ...: else: .... Cela peut être écrit en une ligne.

```
# Utilisation de l'opérateur ternaire : <expression1> if <condition> else <expression2>
min = x if x < y else y
```

Un autre exemple :

```
a=input("Introduire un réel : ")
if a >0:
    print (" a est positif")
elif a<0:
    print ("a est négative)
else:
    print("a est nul")
```

Question : Saisissez un flottant. S'il est positif ou nul, affichez sa racine, sinon affichez un message d'erreur.

Utiliser :

```
>>> from math import sqrt
```

Boucle WHILE

But : Répéter une portion de code.

La syntaxe de WHILE

```
while {condition a laquelle la boucle continue}:
    {code execute dans la boucle}
# indentation necessaire, d'habitude 4 espaces;
{code qui ne sera pas execute dans la boucle}
# car il n'est pas indente.
```

Exemple

```
x = 10
while x != 0:
    x = x - 1
    print ("X decroit, valeur non nulle egale a ", x)
print ("La boucle est finie, x est nul")
```

Question 1 : Que fait l'exemple suivant?

Exemple

```
n =int( input("Entrez un entier [1 .. 10] : "))
while (n < 1) or (n > 10):
    n = int(input("Entrez un entier [1 .. 10], S.V.P. : "))
```

Question 3 : Initialisez deux entiers : a = 0 et b = 10.

(i) Écrire une boucle affichant et incrémentant la valeur de a tant qu'elle reste inférieure à celle de b.

(ii) Écrire une autre boucle décrémentant la valeur de b et affichant sa valeur si elle est impaire. Boucler tant que b n'est pas nul.

Indentation

Le code doit obligatoirement être indenté !

Remarques :

La tabulation sert à délimiter les blocs de code. C'est une bonne pratique dans tout langage. Créer les indentations nécessaires dans l'exemple suivant:

```
x = 10
while x > 0:
print (x)
if x > 5:
print ("Nombre plus grand que 5")
elif x % 2 != 0:
print( "C'est un nombre impaire")
print ("Il n'est pas plus grand que 5")
else:
print ("Ce nombre n'est pas plus grand que 5")
print ("Il n'est pas impaire, non plus")
x = x - 1
print ("nous avons diminue x de 1")
print ("on continue la boucle tant que x reste superieur a 0")
print ("Maintenant x n'est plus superieur a 0")
print ("la boucle est finie.")
```