Département de Mathématiques 2018/2019
Module: Analyse numérique 1
Fiche de T.P. N°2

Initiation à PYTHON (Listes, boucles For et fonctions)

Les listes : définition et exemples

Une liste est une collection hétérogène, ordonnée et modifiable d'éléments, séparés par des virgules et entourée de crochets.

```
>>> fruits = ['figue', 'raisin', 'abricot', 'poire']
type(fruits) # list
```

Accès à des éléments

```
>>> print( fruits[2]) # abricot
>>>fruits[1:3] # ['raisin', 'abricot']
fruits[:2] # ['figue', 'raisin']
>>> fruits[:] # ['figue', 'raisin', 'abricot', 'poire']
>>> fruits[-1] # 'poire'
>>> fruits[1]=13
>>> print (fruits) # ['figue', 13, 'abricot', 'poire']
>>> print (fruits[0:2]) # ['figue', 13]
```

Opérations de liste de base

Les listes répondent aux opérateurs + et * de la même manière que les chaînes; ils veulent dire concaténation et répétition. Le résultat est une nouvelle liste.

```
>>> [1, 2, 3] + [4, 5, 6] # [1, 2, 3, 4, 5, 6]
>>> [1,2,3]*4 # [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> len([1, 2, 3]) # 3
>>> len(fruits) # 4
```

Fonction range

```
>>> liste_1=range(4) # [0, 1, 2, 3]
>>> liste_2=range(4,10) # [4, 5, 6, 7, 8, 9]
>>> liste_3=[liste_1,liste_2]
>>> print liste_3 # [[0, 1, 2, 3], [4, 5, 6, 7, 8, 9]]
>>> 18 in fruits # False
>>> 13 in fruits # True
```

Fonction append

```
>>> fruits.append(18)
>>> print fruits
# ['figue', 13, 'abricot', 'poire', 'pomme', 18]
```

Fonction del

```
>>> del fruits[1]
>>> print fruits
# ['figue','abricot', 'poire', 'pomme', 18]
```

Plus généralement, pour afficher les méthodes de la classe "liste" :

```
help(list)
```

Boucle FOR

But : Répéter une portion de code pour chaque valeur d'une séquence

La syntaxe de FOR

```
for {valeur} in {sequence}:
```

```
    {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.
```

Exemples

```
for lettre in "bonjour":
    print (lettre)
    # b o n j o u r
for i in [1,2,3]:
    print (i)
    # 1 2 3
```

Les listes en compréhension : définition et exemples

Liste en compréhension est une expression qui permet de générer une liste de manière très compacte. Elle est équivalente à une boucle for qui construirait la même liste en utilisant la méthode append().

Exemple, premiere forme :

```
resultat_1 = [x+1 for x in range(5)]
# a le meme effet que :
resultat_1 = []
for x in range(5):
    resultat_1.append(x+1)
```

Exemple, deuxieme forme :

```
resultat_2 = [x+1 for x in range(5) if x > 2]
# a le meme effet que :
resultat_2 = []
for x in range(5):
    if x > 2:
        resultat_2.append(x+1)
```

Exemple, troisieme forme :

```
resultat_3 = [x*y for x in range(5) for y in range(2)]
# a le meme effet que :
resultat_3 = []
for x in range(5):
    for y in range(2):
        resultat_3.append(x*y)
```

Exercices

1. Calculer π avec la formule de Wallis $\pi = 2 \prod_{i=1}^{\infty} \frac{4i^2}{4i^2 - 1}$.

2. Ecrire un programme qui affiche tous les nombres parfaits entre 2 et nmax.

Un nombre parfait est égal à la somme de ses diviseurs lui exclu.

Les fonctions

On a déjà pu rencontrer des fonctions prédéfinies à plusieurs reprises. Ces fonctions proviennent de deux sources différentes; ce sont :

soit des fonctions faisant partie intégrante du langage Python comme print, len et abs, par exemple ;

soit des fonctions ayant été écrites par d'autres programmeurs et mises à disposition en important le module qui les contient, comme la fonction sqrt du module math.

```
>>> abs(-3)
>>> from math import sqrt
>>> print(sqrt(2))
1.4142135623730951
```

On peut créer des fonctions python à l'aide de l'operateur **def** la syntaxe est la suivante:

```
def nom_fonction(liste des parametres):
    """Documentation de la fonction."""
    { code execute dans la fonction }
    # indentation necessaire
    return {valeur retournee dans le programme principal}
{ code qui ne sera pas execute dans la fonction }
# car il n'est pas indente.
```

Remarques :

Dans la définition précédente, nom_fonction est le nom qui sera utilisé ultérieurement pour appeler la fonction.

Le bloc d'instructions est obligatoire. S'il ne fait rien (ou le code n'est pas encore écrit), on utilise l'instruction **pass.**

La documentation est fortement conseillée.

Le passage des arguments se fait par affectation. Chaque argument de la définition de la fonction correspond, dans l'ordre, à un paramètre de l'appel.

Exemples

1. Utilisation sans liste de paramètres :

```
def somme():
    """ somme des 10 premiers entiers"""
    s=0
    for x in range(11):
        s+=x
    return s
# appel de la fonction
print ("La somme est :", somme())
# La somme est 55
help(somme)
# Help on function somme in module __main__:
# somme()
# somme des 10 premiers entiers
```

2.

```
from math import pi
def cube(x):
    return x**3
def volumeSphere(r):
    return 4.0 * pi * cube(r) / 3.0
# Saisie du rayon et affichage du volume
```

```
rayon = float(input("Le rayon de la sphere :"))
print ("Volume de la sphere =",volumeSphere(rayon))
```

3. Fonction avec appel récursif

```
def fibo(n):
    """ Retourne le nombre de Fibonacci n """
    if n == 0 or n == 1:
        return n
    else:
        return fibo( n - 1 ) + fibo( n - 2 )
# Appel de la fonction
print( fibo(9))
```

Exercices

3. Ecrire une fonction VolCyl qui retourne le volume d'un cylindre. Les paramètres sont le rayon R et la hauteur H. Tester cette fonction par des appels avec différents arguments.

4. Ecrire une fonction SolEq pour calculer les solutions réelles de l'équation $ax^2 + bx + c = 0$. Tester pour (1, -3, 2), (1, -2, 1) et (1, 1, 1).