Faculté des Sciences
Aix*Marseille Université

Licence Sciences et humanités, 3e année

Premier TP

vendredi 18 novembre 2016
T. Coulbois

Prise en main de Python

D'abord il vous faut un environnement de programmation. Je propose l'utilisation d'un serveur jupyter, par exemple celui de Sage : `sagemath.org`. (Mais tout IDE (environnement de développement intégré) convient : Eclipse, Pycharm, Wing, etc.).

Créez-vous donc un compte sur le nuage de Sage et ouvrez une feuille de calcul Jupyter.

1 Calculs

1.1 Des entiers

Essayez : `7/3, 7//3, 7%3, 2^3, 2**3, 18@3, 18/0`.

Hiérarchie des opérations : `3+4*5` ou `(3+4)*5`

Quel est le plus grand nombre entier ? Utilisez la variable prédéfinie `_` pour rappeler le dernier résultat : `1*1000` puis répétez beaucoup de fois `_*1000`

1.2 Des nombres réels flottants

Ils sont notez avec une virgule (`1.0`) ou avec la notation exponentielle (`1e-2`, `1.32e5` ou `1.32E5`)

Essayez : `8.0/4` ou `1.0/3`, comparer avec `8/4` et `1/3`.

1.3 Des variables

Python stocke les données dans sa mémoire. On peut donner un nom à ces données. C'est un **nom de variable**. Pour l'ordinateur c'est une référence vers un emplacement de sa mémoire.

Essayez : `a=2, 2a=4, b=2*a, uneVariable=5, une_autre=6.7, try=23, mètres=34.7`

On affecte une valeure à une variable avec le signe `=`

`n=7` puis `type(n)`, `msg="Quoi de neuf ?"` puis `type(msg)`, `pi=3.14159` puis `type(pi)`.

On peut afficher ou utiliser une variable, on peut la ré-affecter : `n=n+1`

1.4 Des chaînes de caractères

Les chaînes de caractères sont entourés de guillemets : `"licence", "l'eau vive", ""`, d'apostrophe : `'eggs', 'une "beauté"')` ou de triples guillemets ce qui permet d'étendre sur plusieurs lignes.

Pour des chaînes `c0='', c1='abc', c2='defg'`, on peut obtenir leur longueur `type(c0), len(c1), c1+c2` et faire des opérations `c3=c1*3`. On peut extraire les caractères `c2[0], c3[5]` et les sous-chaînes `c3[2:5], c3[:6], c3[1:], c3[-1], c3[::2], c3[1:6:-1]`.

1.5 Des listes

On peut en créer `range(6), range(2,7), range(2,15,3), range(8,3,-1)`, calculer leur longueur, changer leurs valeurs, les concaténer, les découper et faire des listes de n'importe quoi.

`l=[1,2,'ab',7.3,7,1], l[2], l[:-1], l[2]='bonjour',len(l), l+range(3)`

1.6 Conversions

Convertir un type de donnée en un autre est souvent naturel : `a=int("34"), x=float(12), int(3.4), str(34)`

2 Programmer

2.1 Affichage et saisies

Vos programmes vont devoir afficher des résultats et des calculs intermédiaires. C'est l'objet de la commande `print`, jusqu'ici nous n'affichions que le résultat d'éxécution de commandes.

`print("a: "), print("pi= ",3.1419)`.

Il est parfois utile de demander son avis à l'utilisateur pour un programme intéractif :

`input("Donnez votre valeur")`. Attention on récupère une chaîne de caractères qu'il faut peut-être convertir en nombre.

Écrire un programme qui demande à l'utilisateur deux nombres, calcule leur somme, différence, produit et rapport et les affiche.

Écrire un programme qui demande à l'utilisateur deux nombres et affiche joliment le quotient et le reste de la division euclidienne : `Division euclidienne: 13 = 5 * 2 + 3"`.

2.2 Condition

```
a = int(input("a : "))
if a > 100 :    # n'oubliez pas le deux-points...
    print("<a> dépasse la centaine")   # ni l'indentation
elif a == 100 :
    print("<a> vaut 100")
else :
    print("<a> ne vaut pas cher !")
```

Écicre un programme qui demande à l'utilisateur des coefficients `a`, `b` et `c` et affiche les racines de $ax^2 + bx + c$.

2.3 Boucles

Il y a essentiellement deux types de boucles : les boucles `for` et les boucles `while`.

```
for i in range(10):
    print(i,i*i)
```

Programmer le calcul de la somme des 100 premiers entiers. Des 20 premiers carrés.

Que fait le programme ?

(notez le `end=' '` qui remplace le retour à la ligne par un espace.

```
n = int(input("Donnez un nombre entier: "))
d = 1
while d < n :
    if n % d == 0:
        print(d, end=' ')
    d = d + 1
```

Trouver tous les nombres parfaits plus petits que 1000 (un nombre est **parfait** s'il est égal à la somme de ses diviseurs stricts).

3 Exercice : la suite de Syracuse

Une suite de Syracuse est une suite de nombres entiers qui vérifie

$$u_{n+1} = \begin{cases} \frac{u_n}{2} \text{ si } u_n \text{ est pair} ; \\ 3u_n + 1 \text{ si } u_n \text{ est impair.} \end{cases}$$

Par exemple pour $u_0 = 11$, la suite est $11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, \ldots$ En s'arrêtant au premier 1, cette suite de nombre est le **vol** de 11, 52 son **altitude maximale** et 15 la **longueur de son vol**.

Écrire un programme qui demande un entier à l'utilisateur et affiche le vol de cet entier. Modifier le programme pour afficher la longueur du vol et l'altitude maximale.

3.1 Fonctions

Une fonction est un bloc de code qui pourra être utilisé dans la suite du programme.

Elle est composée d'une ligne d'en-tête qui commence par le mot clé `def`, du nom de la fonction et d'une liste de paramètres, elle se termine par le caractère deux points. Ensuite, il est conseillé d'écrire une ligne de description (entre guillemets). Puis le contenu de la fonction est indenté. Une fonction se termine et retourne la valeur déclarée par `return`.

```
def doubler(x):
    "Retourne le double de x"
    return 2 * x
```

Écrire une fonction `etapeSuivante(u)` qui calcul l'étape suivante de la suite de Syracuse : `etapeSuivante(7)` retourne 22.

Reprendre le programme sur la suite de Syracuse pour utiliser cette fonction.

3.2 Récursivité

Une fonction peut s'appeler elle-même.

```
def altitudeMaximal(u):
     "Retourne l'altitude maximale de x."
     v=etapeSuivante(u)
     return max(u,altitudeMaximale(v))
```

4 Au secours !

Il y a beaucoup de documentation et d'aide en ligne.

D'abord, en ligne de commande taper la touche de tabulation 

complète la commande par les commandes connues de Python.

Ensuite, toujours en ligne de commande, éxécuter une commande suivi du point d'interrogation affiche la documentation de cette commande : `max?`.

```
Docstring:
max(iterable[, key=func]) -> value
max(a, b, c, ...[, key=func]) -> value

With a single iterable argument, return its largest item.
With two or more arguments, return the largest argument.
Type:      builtin_function_or_method
```

Connectez-vous sur `docs.python.org` et en particulier le tutoriel : `docs.python.org/3/tutorial`.