Langage Python : révisions

On utilisera Python 3. En salle informatique, se connecter sous Linux.

Les différentes façons de faire du Python

Mode interactif. On peut utiliser Python de façon interactive : taper `python3` dans une console pour démarrer. Vous pouvez alors taper des commandes en Python. Exemple :

```
>>> a=3
>>> b=5
>>> a+b
8
>>> a*b
15
```

Il suffit de taper `ctrl-D` pour sortir du mode interactif. On peut définir des fonctions en mode interactif mais ce n'est pas la méthode adaptée pour écrire des programmes.

Avec un éditeur de texte. Lancer un éditeur de texte (par exemple *emacs*) et taper votre programme (ne pas oublier de sauver). Par exemple le programme `fichier.py` :

```
def somme(a,b):
    return a+b

print(somme(3,5))
```

Ouvrir un terminal, se placer dans le répertoire où se trouve le programme (avec la commande `cd`) et taper `python3 fichier.py`. Le programme est exécuté. La flèche vers le haut permet d'accéder aux commandes précédentes. Il est donc facile de modifier le programme puis de le relancer.

Noter que dans le mode interactif, le résultat de la commande s'écrit en-dessous ce qui n'est pas le cas quand on lance un programme (pour s'en convaincre, changer le programme en supprimant la fonction `print`).

Jupyter. Le notebook jupyter prend place dans un navigateur. La feuille est constituée d'une suite de cellules qui peuvent être du code Python, dans lesquelles on peut définir des fonctions Python et les appeler, ou des cellules de texte en *markdown*, avec la possibilité d'écrire des formules mathématiques en *latex*. Quand on modifie une cellule définissant une fonction, il ne faut pas oublier de l'évaluer à nouveau pour que les modifications soient prises en compte.

Les environnements de programmation. Ils sont indispensables pour développer de gros programmes. Puissants, ils peuvent être aussi difficile à maîtriser. Si vous souhaitez en essayer un vous pouvez par exemple essayer *pyzo*.

Les commentaires en Python

Les commentaires en Python suivent le symbole `#`. Ceci est très utile pour ajouter des explications ou commenter des lignes de code (elles ne seront pas exécutées).

Types de base, opérations, conversion

Le plus simple est d'utiliser le mode interactif pour cette partie, dans jupyter ou à défaut dans une console.

1. La commande `type()` permet de connaître le type d'un objet. De quel type sont les objets `True`, `10`, `2.3`, `'bonjour'` ?
2. Tester les opérations `+` et `*` entre deux entiers, entre un entier et une chaîne de caractères, entre deux chaînes de caractères. Quand ces opérations sont-elles possibles et que font-elles ?
3. Les opérations booléennes s'écrivent `and`, `or`, `not`. Que renvoie `not (False or True)` ?
4. Les opérations de comparaison se font avec `==`,`<`, `<=`, etc. Attention à ne pas utiliser l'affectation `=` à la place du test d'égalité. Que vaut `2==3` ?
5. La conversion vers une chaîne de caractères se fait avec `str()`. Tester la conversion de différents types d'objets vers une chaîne de caractères.
6. Essayer de convertir une chaîne de caractères en entier. Quand cela est-il possible ?

   Remarque : les variables n'ont pas de type en Python. Par exemple, il est tout à fait possible de faire :

```
>>> a=3
>>> a='bonjour'
```

La variable `a` contient successivement des objets de types différents.

Instructions de test

On rappelle sur un exemple la syntaxe des tests.

```
x=2

if x==1:
    print('egal a 1')
elif x==2:
    print('egal a 2')
elif x==3:
    print('egal a 3')
else:
    print('autre valeur que 1,2 ou 3')
```

Comme pour les fonctions, c'est l'indentation qui permet de délimiter les blocs. Les parties en `elif` et `else` sont optionnelles.

1. Écrire une fonction `maximum(a,b)` qui renvoie le plus grand des entiers a, b passés en paramètres (pour cet exercice, sans utiliser la fonction `max` de Python).
2. Écrire une fonction `milieu(a,b,c)` qui renvoie l'élément du milieu si a, b, c sont distincts et le plus grand élément sinon.

Les listes

1. Exécutez la commande suivante :

   ```
   >>> L=[1,3,'ab',2,'c',0]
   ```

   Examinez le type de `L`, sa longueur (fonction `len()`). Accéder aux différents éléments de `L` en inspectant par exemple `L[3]`. Quel est l'indice du premier élément de L ? L'indice du dernier ?

2. Tester les opérations élémentaires sur les listes. Toutes se font en temps constant (il s'agit d'un coût *amorti* pour `append` et `pop`, c'est-à-dire que le coût moyen d'une telle opération est constant quand on le considère sur tout le déroulement de l'exécution).
   - Création d'une liste vide `[]` ;
   - Longueur d'une liste `len(L)` ;
   - Lecture et modification d'un élément, par exemple `L[1]` ou `L[2]=7` ;
   - Ajout d'un élément à la fin d'une liste : `L.append(3)` ;
   - Suppression du dernier élément d'une liste : `x=L.pop()`.

   Il existe de nombreuses fonctions et méthodes s'appliquant aux listes. Attention au coût de ces opérations lorsqu'on évalue la complexité d'une fonction. Ci-dessous deux exemples de telles fonctions :
   - Concaténation de deux listes avec `+` ;
   - Création d'une sous-liste avec la syntaxe `L[debut:fin]` (à noter : les arguments `début` et `fin` sont optionnels).
3. Écrire une fonction `concatenation(L,M)` retournant une nouvelle liste qui est la concaténation des listes `L` et `M` à partir des opérations de base sur les listes uniquement.
4. De même, écrire une fonction `sousliste(L,debut,fin)` (il n'est pas demandé de faire en sorte de gérer les arguments optionnels).

La boucle `for`

On rappelle la syntaxe par l'exemple suivant :

```
L=[1,3,'ab',2,'c',0]
for x in L:
  print(x)
```

1. Ecrire une fonction `somme(L)` retournant la somme des éléments de la liste `L`.

On veut souvent faire une boucle du type :

"pour i allant de 1 à 10"

En Python, on utilise pour cela la fonction `range` qui crée la liste d'entiers. Pour la boucle ci-dessus on écrit :

```
for i in range(1,11):
```

La syntaxe est `range(fin)`, ou `range(debut,fin)` ou encore `range(debut,fin,pas)`.

Comme l'objet créé par `range` n'est pas réellement la liste d'entiers mais un objet se comportant comme tel, on convertit l'objet retourné par `range` pour voir s'afficher la liste : par exemple `list(range(4))`.

2. Que retournent les appels suivants ? (Essayer de répondre avant de tester.)

`list(range(10))` `list(range(2, 12))` `list(range(2, 12, 3))`

3. Utiliser la fonction `range` pour produire les listes suivantes :

$[0, 1, 2, 3, 4]$ $[1, 3, 5, 7, 9, 11]$ $[13, 12, 11, 10, 9, 8]$.

La boucle `while`

1. On définit la suite (u_n) par $u_0 = 0$ et $u_{n+1} = u_n^2 + 3u_n + 1$ pour tout $n \in \mathbb{N}$. Écrire une fonction `rang(A)` qui retourne le plus petit entier n tel que $u_n \geqslant A$.
2. Que calcule la fonction suivante ?

```
def f(n):
  k=1
  while(k<n):
    k=3*k
  return k
```

Exercice : affichage de damiers

1. Écrire un programme qui affiche un damier carré de taille n. Par exemple :

```
>>> damier(5)
.#.#.
#.#.#
.#.#.
#.#.#
.#.#.
```

2. Écrire un programme qui affiche un damier carré de taille n avec des cases de taille c. Par exemple :

```
>>> damier2(4,3)
...###...###
...###...###
...###...###
###...###...
###...###...
###...###...
...###...###
...###...###
...###...###
###...###...
###...###...
###...###...
```

Exercice : nombre d'occurrences dans une liste de listes

Le but de cet exercice est de compter le nombre d'occurrences d'un élément dans une liste de listes.

1. Écrire une fonction `nbocc(L,x)` prenant en entrée une liste L et retournant le nombre d'occurrences de x dans L. Par exemple, `nbocc`$([1,3,3,5,3,8,3],3)$ doit renvoyer 4.
2. Écrire une fonction `listeocc(LL,x)` prenant en entrée une liste de listes d'entiers $LL = [L_0, L_1, \ldots, L_k]$ et un entier x et retournant la liste $[n_0, n_1, \ldots, n_k]$ où n_i est le nombre d'occurrences de x dans L_i.
3. Écrire une fonction `totalocc(LL,x)` prenant en entrée une liste de listes d'entiers et retournant le nombre total d'occurrences de x dans les sous-listes de `LL`.