TP 1: Débuts avec Python

6 mars 2014

Dans ce TP, nous allons faire nos premiers pas avec Python. Nous allons graduellement prendre connaissance de la syntaxe, réinvestir les notions abordées dans le premier cours et écrire nos premiers programmes en Python. Pour toute question, suggestion ou remarque qui vous viendrait hors de la séance de TP, n'hésitez à m'écrire à l'adresse *edbonnet@hotmail.com*.

1 Prise en main

Créez un fichier *hello.py*. Écrivez l'instruction suivante : *print "hello world"*. Puis dans un terminal, compilez avec *python hello.py*. La fonction *print* affiche son argument. Ce petit programme s'appelle communément un *hello world* ; il s'agit d'un tout premier pas dans un langage de programmation qui permet de se familiariser avec la syntaxe de base et à la compilation.

2 Variables

En Python, les variables n'ont pas besoin d'être déclarées. L'affectation d'une valeur v dans une variable x s'écrit $x = v$. Python est un langage dynamiquement typé. Vous n'avez donc pas non plus besoin de préciser le type d'une nouvelle variable. On se contentera pour l'instant des types entier et chaîne de caractères. *x=2* initialise une variable x à la valeur 2. *s="bonjour"* stock la chaîne de caractères "bonjour" dans la variable s. Les opérations de base sur les entiers ont la syntaxe infixe habituelle (+,-,*,/). L'opérateur + est dit surchargé car il effectue aussi (entre autres) la concaténation de deux chaînes de caractères. Par exemple "bonj"+"our" renvoie la chaîne "bonjour".

Que va afficher l'exécution des lignes suivantes ?

```
x=1
s="bla"
x=3-x
print s+"s"
print 3*x+4
```

Vérifiez votre hypothèse en recopiant le code précédent et en compilant. Bien, et que va faire le bout de code suivant ?

```
x=4
s="3"
print s+x
```

Pour éviter ce problème de type, on pourrait utiliser les fonctions de conversions de type *str* et *int*. Remplacez la troisième ligne par *print int(s)+x* puis par *print s+str(x)*. Que se passe-t-il ?

3 Les blocs par l'indentation

Un bloc est un morceau de code cohérent. Il est précédé de " :" et sa fin est marquée par un réalignement. Voici quelques exemples de blocs avec une conditionnelle.

```
if n == 0:
    n = 1
    if m == 0:
        m = 2 * n
    else:
        m = n - 1
```

Notez au passage que le test d'égalité == ne doit pas être confondu avec l'affectation =.

Y a-t-il une différence avec le code suivant :

```
if n == 0:
    n = 1
    if m == 0:
        m = 2 * n
else:
    m = n - 1
```

4 Premières fonctions

Une fonction en Python prend un certain nombre d'arguments (ou paramètres) entre parenthèses, possiblement 0, séparés par des virgules. Pour définir une fonction, vous pouvez utiliser le mot clé *def*. La valeur de retour de la fonction est précédée par le mot clé *return*. Le corps de la fonction constitue un block. Ainsi, si on écrit par exemple la fonction carrée, le code pourra être :

```
def carree(n):
    return n*n
```

L'appel d'une fonction se fait en passant à la fonction des valeurs pour ces paramètres : *carree(4)* renverra la valeur 16.

Que fait la fonction suivante ? Au fait, % est l'opérateur modulo.

```
def f(n,m):
    s = "pair"
    if n < m:
        if m%2 == 0:
            return s
        else:
            return "im"+s
    else:
        if n%2 == 0:
            return s
        else:
            return "im"+s
```

La fonction factorielle est définie par $n! = \Pi_{i=1}^{n} i$. Pour la question suivante, on pourra utiliser la syntaxe *for i in c :* où c est un *objet itérable* et i décrit cet "ensemble". La fonction *range* permet de créer des intervalles d'entiers : *range(i,j)* avec $i \leqslant j$ renvoie l'objet itérable $\{i, i+1, \ldots, j-1, j\}$. Par défaut, $range(k)$ renvoie $range(0, k-1)$. Par exemple, ce morceau de code calcule dans s la somme des k premiers entiers.

```
s=0
for i in range(1,k):
    s = s + i
```

Exercice 1 *Écrire une fonction itérative qui calcule la factorielle d'un nombre.*

De façon équivalente, on peut définir la factorielle inductivement par $0! = 1$ et $\forall n \geqslant 1$, $n! = n \times (n-1)!$.

Exercice 2 *Écrire une version récursive de la précédente fonction.*

Maintenant, écrivez une fonction avec deux paramètres :

Exercice 3 *Écrire une fonction itérative qui calcule la puissance d'un entier par un entier.*

Exercice 4 *Écrire une version récursive de la fonction puissance.*

Remarquez que si n est pair : $a^n = (a^{\frac{n}{2}})^2$, et si n est impair : $a^n = a(a^{\lfloor \frac{n}{2} \rfloor})^2$.

Exercice 5 *En déduire une version récursive améliorée de la fonction puissance.*

Asymptotiquement, combien de multiplications réalise-t-on avec la version améliorée ?

5 Suites de Syracuse

Une suite de syracuse est définie par la relation de récurrence $u_{n+1} = \frac{u_n}{2}$ si n est pair et $u_{n+1} = 3n + 1$ sinon. La donnée du premier terme $u_1 > 0$ définit donc une unique suite qu'on notera $syr(u_1)$. On conjecture depuis des siècles que tout premier terme mène inévitableblement au cycle 4-2-1-4-2-1 ....

Exercice 6 *Écrire une fonction qui étant donné un premier terme u_1, affiche tous les termes de la suite $syr(u_1)$ jusqu'à la première occurrence du nombre 1.*

Dans la suite, on considère toujours une suite $(u_n) = syr(u_1)$ comme tronquée à partir du plus petit indice l tel que $u_l = 1$. En effet, après cet indice l appelé *durée de vol*, la suite (u_n) radote et n'a plus grand intérêt.

Exercice 7 *Écrire une fonction qui étant donné u_1, renvoie la durée de vol de $syr(u_1)$.*

Une autre caractéristique naturelle des suites de Syracuse est le plus grand entier atteint appelé *hauteur de vol* et est défini par $\max\{u_i | 1 \leqslant i \leqslant l\}$.

Exercice 8 *Écrire une fonction qui étant donné u_1, renvoie la hauteur de vol de $syr(u_1)$.*

Exercice 9 *Afficher les deux caractéristiques des 1000 premières suites de Syracuse, à savoir $syr(1), syr(2), \ldots, syr(1000)$.*

Pour les deux questions suivantes, on peut utiliser des boucles *while*. La syntaxe est tout simplement :

```
while cond:
    instr
```

Exercice 10 *Calculer le plus petit entier u_1 tel que la hauteur de vol de u_1 est supérieur à $100u_1$.*

Exercice 11 *Calculer le plus petit entier u_1 tel que la durée de vol de u_1 est supérieur à $100u_1$.*