Algorithmique et Programmation 3
TP 1: Débuts avec Python

01/10/2020

1 Prise en main (rappels)

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 (rappels)

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 (rappels)

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
```

4.1 Fonction Factorielle

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 $\{i, i+1, \dots, j-1\}$. 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.

Exercice 2. De façon équivalente, on peut définir la factorielle inductivement par $0! = 1$ et $\forall n \geqslant 1$, $n! = n \times (n-1)!$. Écrire une version récursive de la précédente fonction.

4.2 Suite de Fibonacci

Exercice 3. La définition de la suite de Fibonacci est la suivante :

$$f_n = \begin{cases} 0 & \text{si } n = 0, \\ 1 & \text{si } n = 1, \\ f_{n-1} + f_{n-2} & \text{si } n \geq 2. \end{cases}$$

Écrivez un algorithme récursif **fibo_rec** qui, étant donné un entier n, calcule f_n.

Exercice 4. L'algorithme itératif ci-dessous (Algorithme 1) permet aussi de calculer f_n.

Algorithm 1: fibo_it(n)

```
if n = 0 then
    retourner 0
end
else
    x ← 0
    y ← 1
    for i = 2 à n do
        temp ← x + y
        x ← y
        y ← temp
    end
    retourner y
end
```

Écrivez une function **fibo_it** qui implante l'Algorithme 1.

Exercice 5. Vérifiez que les deux fonctions **fibo_rec** et **fibo_it** retournent bien la même valeur, puis comparez le temps d'exécution de ces deux fonctions pour différentes valeurs de n.

Exercice 6. Écrivez une fonction récursive **_fibo_smart_rec_aux** qui, étant donné un entier $n \geq 1$, renvoie la paire $(f_n,\ f_{n-1})$. En exploitant cette fonction auxiliaire, écrivez une fonction **fibo_smart_rec** qui, étant donné un entier n, renvoie f_n. Comparez son résultat et son temps de calcul à ceux des fonctions précédentes pour différentes valeurs de n.

Exercice 7. Population de lapins
On considère un modèle simplifié de l'évolution d'une population de lapins. Comme ceux-ci se reproduisent rapidement (en comparaison de leur durée de vie), on suppose que sur l'intervalle de temps considéré, il n'y a que des naissances et aucun décés.
A chaque instant entier n, on note p_n le nombre de couples de lapins pubères et j_n le nombre de couples de lapins juvéniles, qui ne peuvent pas encore procréer. A l'instant $n = 0$, il y a seulement un couple de lapins pubères. Entre l'instant $n-1$ et l'instant n :

(a) chaque couple de lapins qui étaient pubères à l'instant $n-1$ donne naissance à un couple de lapins juvéniles,

(b) tous les lapins qui étaient juvéniles à l'instant $n-1$ deviennent des lapins pubères.

1. Donnez des relations de récurrence reliant les suites (p_n) et (j_n).
2. Déduisez-en une relation de récurrence portant uniquement sur la suite (p_n), et une autre portant uniquement sur la suite (j_n).
3. En notant l_n le nombre total de couples de lapins à l'instant n, donnez une relation de récurrence pour la suite (l_n).
4. Exprimez p_n, j_n et l_n en fonction des éléments de la suite de Fibonacci (f_n).
5. Quelle analogie voyez-vous entre ce modèle de population et les algorithmes des exercices 3 et 4 ?

4.3 Fonction Puissance

Maintenant, on va travailler avec une fonction avec deux paramètres :

Exercice 8. Écrire une fonction itérative qui calcule la puissance d'un entier par un entier.

Exercice 9. Écrire une version récursive de la fonction puissance.

Exercice 10. Remarquez que si n est pair : $a^n = (a^{\frac{n}{2}})^2$, et si n est impair : $a^n = a(a^{\lfloor \frac{n}{2} \rfloor})^2$. En déduire une version récursive améliorée de la fonction puissance. Asymptotiquement, combien de multiplications réalise-t-on avec la version améliorée ?