Informatique - Notes de cours 10

Fonctions récursives

M. Marmorat

24 juin 2025

1 Syntaxe d'une fonction récursive

On a rencontré plusieurs fois des fonctions définies par :

- une valeur dans un ou plusieurs "cas de base", et
- pour les autres cas, une relation "de récurrence" permettant de se ramener à un cas "plus simple".

Par exemple, pour la suite (u_n) définie par : $u_0 = 5$ et $\forall n \geq 0,\ u_{n+1} = (u_n + 3)^2$:

- le "cas de base" correspond à $n = 0$
- la relation $u_{n+1} = (u_n + 3)^2$ permet de ramener le calcul de u_{n+1} à un cas "plus simple" : le calcul de u_n.

En langage Python, on peut tirer partie d'une telle struture récursive pour définir des fonctions. En fait, Python accepte de **faire appel à une fonction `f` au sein même de la définition de `f`**. On parle d'*appel récursif* à `f`.

La syntaxe générale est la suivante :

```
def f(arguments) :
    if cas de base :
        return valeur du cas de base
    else :
        return valeur utilisant f(arguments_bis)
```

Par exemple, pour la suite (u_n) ci-dessus :

Attention, l'appel récursif à `f` est autorisé, mais c'est à vous de vous assurer que cet appel permet à Python de redescendre in fine au cas de base.
Que se passe-t-il par exemple dans la situation suivante ?

```
def f(n) :
    if n == 0 :
        return 1
    else :
        return (f(n)+3)**2
```

De manière générale, les appels récursifs à la fonction doivent toujours être "plus simples" que ceux qui sont en train d'être définis. Le plus souvent, on appelera simplement la fonction sur des arguments plus petits, les cas de base correspondant aux plus petites valeurs possibles.

2 Exemples classiques

2.1 Boucle `for` vs fonction récursive

L'utilisation des fonctions récursives est particulièrement adaptée au calcul des suites récurrentes que nous traitions jusqu'à maintenant avec des boucles `for`.

Exercice 1. Écrire une fonction Python récursive prenant en argument $n \in \mathbb{N}$ et renvoyant u_n où (u_n) est la suite donnée par $u_0 = 4$ et $\forall n \geq 0,\ u_{n+1} = n + u_n^2$.

Exercice 2. Écrire une fonction Python récursive prenant en argument $n \in \mathbb{N}$ et renvoyant u_n où (u_n) est la suite donnée par $u_0 = 4$ et $\forall n \geq 1,\ u_n = \dfrac{1 + u_{n-1}}{2 + u_{n-1}}$.

Attention, si on réalise la fonction précédente sous la forme suivante :

```
def f(n):
    if n==0 :
        return 4
    else :
        return (1+f(n-1))/(2+f(n-1))
```

combien d'appels récursifs à la fonction `f` sont nécessaire pour calculer u_4 ?

Moralité : toujours utiliser *avec parcimonie* les appels récursifs : il faut en faire le moins possible ! Les codes :

```
x = (1+f(n-1))/(2+f(n-1))
```

```
u = f(n-1)
x = (1+u)/(2+u)
```

sont mathématiquement équivalents, mais pas informatiquement !

Parfois, la relation de récurrence n'est pas explicitement donnée, c'est alors à vous de la trouver.

Exercice 3. Écrire une fonction récursive `facto` prenant en argument $n \in \mathbb{N}$ et renvoyant $n!$.

2.2 Exponentiation rapide

L'intérêt des fonctions récursives par rapport aux boucles `for` réside dans le fait que l'appel récursif de la fonction peut se faire avec n'importe quelle valeur "plus proche des cas de bases". Si `f` est une fonction définie récursivement, calculer `f(n)` ne se fait pas toujours à l'aide de l'appel récursif de `f(n-1)`. On peut faire un appel récursif à n'importe quelle valeur `f(k)` tant que cet appel nous "rapproche" des cas de base.

Un exemple classique d'utilisation de fonctions récursives est ce qu'on appelle *l'exponentiation rapide*. C'est une méthode permettant de calculer plus "rapidement" x^n pour $x \in \mathbb{R}$ et $n \in \mathbb{N}$ que la technique "naïve".

Exercice 4 (exponentiation naïve)**.** Écrire une fonction récursive `puissance1` prenant en argument un réel x et un entier naturel n et renvoyant x^n.

Exercice 5 (exponentiation rapide).

1. Soient $x \in \mathbb{R}$ et $n \in \mathbb{N}$. Si n est pair, comment calculer x^n en fonction de $x^{n/2}$? Si n est impair, comment calculer x^n en fonction de $x^{(n-1)/2}$?
2. Écrire une fonction récursive `puissance2` prenant en argument un réel x et un entier naturel n et renvoyant x^n.

Remarque 1. Combien de multiplications sont nécessaires au calcul de x^{12} dans `puissance1(x,12)` et dans `puissance2(x,12)` ? Plus généralement, combien de multiplications sont nécessaires au calcul de x^n dans `puissance1(x,n)` et dans `puissance2(x,n)` ?

3 Entraînement

Exercice 6. 1. Écrire une fonction récursive Python prenant en argument $n \in \mathbb{N}$ et renvoyant u_n où (u_n) est la suite définie par $u_0 = 2$ et : $\forall n \in \mathbb{N}^*,\ u_n = 1 + 2u_{n-1} + 5u_{n-1}^2 - 2u_{n-1}^3$.

2. Écrire une fonction récursive Python prenant en argument $n \in \mathbb{N}^*$ et renvoyant v_n où (v_n) est la suite définie par $v_1 = 3$ et : $\forall n \in \mathbb{N}^*,\ v_{n+1} = \dfrac{v_n + 2}{v_n + n}$.

Exercice 7. On définit la fonction suivante, qui sera appelée avec `a` et `b` des entiers naturels non nuls :

```
def f(a,b):
    if b==1 :
        return a
    else :
        return a + f(a,b-1)
```

1. Quel est le cas de base de cette fonction récursive ? Pourquoi n'y a-t-il pas de boucle infinie ?
2. Que renvoie `f(3,4)` ? De manière générale, que renvoie `f(a,b)` pour `a`, `b` $\in \mathbb{N}^*$?
3. Vérifier que la relation de récurrence de la ligne 5 est cohérente avec la valeur de `f(a,b)` que vous avez donnée question 2.

Exercice 8. Écrire une fonction récursive prenant en paramètre un entier n et renvoyant le n-ème terme de la suite de Fibonacci.

Exercice 9. Écrire une fonction récursive `nb_puiss_2` prenant en argument un entier `n` et renvoyant le nombre de fois que cet entier est divisible par 2, c'est-à-dire l'entier k tel que $n = 2^k m$ avec m impair.