TP n°4 : corrigé

PTSI Lycée Eiffel

décembre 2014

1 Lecture et analyse de programmes

Il faut évidemment tester les fonctions avec votre Python préféré. La première fonction calcule simplement $f(x) = 3x^2 - 3x$. La deuxième affiche le nombre qui est le plus petit entre x et x^2. La dernière calcule le plus petit entier n pour lequel $\sum_{k=1}^{n} k$ devient plus grand que x.

2 Quelques petits exemples pour commencer à programmer vous-même

1. Un exemple parmi tant d'autres possibles pour la puissance :

```
def puissance(x,n) :
    p=1
    for i in range(n) :
        p=p*x
    return p
```

   Cette version ne fonctionnera qu'avec un entier n positif, on peut toujours rajouter un teste pour gérer les puissances négatives en ajoutant simplement un inverse à la fin.

2. Il faut simplement faire attention à bien garder en permanence deux termes de la suite en mémoire, et à les modifier correctement. La solution classique pour cela est d'utiliser une variable temporaire, mais on peut en Python affecter simultanément deux variables, autant en profiter (bien sûr, en pratique, les affectations ne sont pas simultanément, c'est Python qui gère tout seul la variable temporaire).

```
def fibonacci(n) :
    a,b=0,1
    for i in range(n-1) :
        a,b=b,a+b
    return b
```

3. Plein de possibilités ici, notamment certaines qui utilisent un passage par les chaines de caractères pour isoler les chiffres. Pour faire quelque chose de plus mathématique, on peut procéder de la façon suivante : pour isoler le dernier chiffre d'un nombre, on prend le reste de sa division par 10, et pour supprimer ce dernier chiffre on divise le nombre (de façon euclidienne) par 10.

```
def sommechiffres(n) :
    a=n ; s=0
```

```
while a!=0 :
        s=s+a%10
        s=s//10
return s
```

3 Pour s'entraîner avec la récursivité

Pour le calcul de puissance, on peut faire une première version récursive simple, où on se contente de multiplier x par x^{n-1} (quand $n \geqslant 1$) pour calculer x^n. En Python, cela peut donner ceci :

```
def recpuiss(x,n) :
    if n==0 :
        return 1
    else :
        return x*recpuiss(x,n-1)
```

Une autre version nettement plus efficace, où on divise la puissance par 2 à chaque fois que c'est possible :

```
def recpuiss(x,n) :
    if n==0 :
        return 1
    elif n==0 :
        return x
    elif n%==2 :
        return recpuiss(x,n/2)**2
    else :
        return x*recpuiss(x,(n-1)/2)**2
```

Le nombre de calculs nécessaires avec cette dernière méthode est de l'ordre de $2 * \log_2(n)$ au pire. Passons à un Fibonacci bêtement récursif :

```
def recfibo(n) :
    if n==0 :
        return 0
    elif n==1 :
        return 1
    else :
        return recfibo(n-1)+recfibo(n-2)
```

Ce programme est en fait complètement minable, car il ne se rend pas compte qu'il va devoir recalculer plein de fois les mêmes nombres. Par exemple, pour calculer F_{20}, il va tenter de calculer $F_{18} + F_{19}$, mais en calculant les deux morceaux de façon indépendate, donc en faisant deux nouveaux appels à la fonction pour F_{18}, et deux autres pour F_{19}. En multipliant par deux le nombre d'appels à chaque étape, le temps d'exécution du programme explose très rapidement. Dernier programme pour lequel la récursion marche très bien, la somme des chiffres :

```
def recsommechiffres(n) :
    if n<10 :
        return n
    else :
        return (n%10 + recsommechiffres(n//10))
```

4 Amusons-nous un peu avec les nombres entiers

1. Le plus simple est évidemment de tester si notre nombre est divisible par chacun des entiers plus petits que lui. Plus précisément, on fera varier les entiers à tester entre 2 et $\sqrt{n}$, ce qui est suffisant. Si on trouve un diviseur, on sort de la boucle en retournant la valeur False. Si on arrive en fin de boucle, c'est que le nombre est premier, et on ressort la valeur True.

```
def prems(n) :
    for i in range(2,int(sqrt(n))+1) :
        if n%i==0 :
            return False
    return True
```

2. On peut faire très rapide en utilisant la fonction précédente et une manipulation intelligente des listes en Python

```
def listeprems(n) :
    return [i for i in range(2,n+1) if prems(i)==True]
```

3. L'algorithme d'Euclide est très simple à programmer avec les doubles affectations de Python :

```
def euclide(n,p) :
    a,b=n,p
    while b!=0 :
        a,b=b,a%b
    return a
```

   Une version récursive est en effet possible quoique relativement superflue :

```
def receuclide(n,p) :
    if p==0 :
        return n
    else :
        return receuclide(p,n%p)
```

4. On peut reprendre quasiment le même programme qu'à la première question : si on croise un diviseur de n (ce sera forcément le plus petit), on retourne sa valeur, sinon c'est que n est premier et on retourne tout simplement la valeur de n.

```
def facteurmin(n) :
    for i in range(2,int(sqrt(n))+1) :
        if n%i==0 :
            return i
    return n
```

5. Je donne directement une version récursive car c'est vraiment efficace pour un programme de ce type : on cherche le plus petit facteur premier, et on recommence après avoir divisé n par ce facteur.

```
def listefacteurs(n) :
```

```
if n==1 :
    return []
else :
    p=facteurmin(n)
    return [p]+listefacteurs(n/p)
```

5 Pour ceux qui ont tout trouvé trop facile

Bon, je n'ai pas tout trouvé facile, alors je ne fais pas cette partie ! Plus sérieusement, je la mettrai à jour, mais plus tard.