

## TP n° 09 – Fonctions récursives

**Définition.** Une fonction récursive est une fonction dont le calcul nécessite d'invoquer la fonction elle-même.

**Exercice 1.** On considère le programme suivant :

---

```

1 def f(x,p):
2     if p==0 :
3         return(1)
4     else:
5         return(x*f(x,p-1))

```

---

1. Recopiez le programme et exécutez-le. En général, que renvoie la fonction  $f$  ?

**Solution 1.**  $f(x,p)$  renvoie la valeur de  $x^p$ .

2. Modifiez le programme pour qu'il affiche les valeurs successives de  $x$  et de  $p$ .

**Solution 2.**

```

def f(x,p):
    if p==0
        return(1)
    else:
        print(x,p)
        return(x*f(x,p-1))

```

3. Que se passe-t-il si on efface les lignes 2, 3 et 4 ?

**Solution 3.** Le message d'erreur suivant s'affiche : "maximum recursion depth exceeded". On parle de dépassement de la pile. Le programme ne s'arrête jamais car  $p$  décroît sans limite.

**Exercice 2.** Exponentiation rapide.

Dans cet exercice, nous allons optimiser le programme précédent grâce au principe suivant :

$$\text{pour } x \in \mathbb{R}, \text{ et } p \in \mathbb{N}, \quad x^p = \begin{cases} 1 & \text{si } p = 0 \\ (x^2)^{\frac{p}{2}} & \text{si } p \text{ est pair} \\ x \times (x^2)^{\frac{p-1}{2}} & \text{si } p \text{ est impair} \end{cases}$$

La fonction `expon` prend comme entrée deux valeurs  $x$  et  $p$ .

`expon(x,n)` :

- Si  $p$  est nul, renvoyer 1.
- Si  $p$  est pair, la fonction s'appelle elle-même, avec comme argument  $x^2$  et  $\frac{p}{2}$ .
- Si  $p$  est impair, renvoyer  $x \times \text{expon}(x^2, \frac{p-1}{2})$ .

1. Ecrivez une fonction récursive `expon` qui suit l'algorithme ci-dessus.

**Solution 4.**

```

def expon(x,p):
    if p==0:
        return(1)
    else:
        if p%2==0:
            return(expon(x**2,p//2))
        else:
            return(x*expon(x**2,(p-1)//2))

```

2. On a donc deux fonctions,  $f$  de l'exercice 1 et la fonction `expon` qui renvoient le même résultat. Proposez une façon de comparer la complexité des deux fonctions.

Deux méthodes :

- on utilise la fonction `time` de la bibliothèque `time`, présentée au TP n°7 ;
- (facultatif) on ajoute un compteur au programme à l'aide d'une variable globale.

**Solution 5.**

Méthode 1 :

```
start=time.time()
(expon(x,p))
end=time.time()
print(end-start)
```

```
start=time.time()
(f(x,p))
end=time.time()
print(end-start)
```

Méthode 2 :

```
compteur=0

def expon_compteur(x,p):
    if p==0:
        return(1)
    else:
        global compteur
        compteur=compteur+1
        if p%2==0:
            return(expon_compteur(x**2,p//2))
        else:
            return(x*expon_compteur(x**2,(p-1)//2))

print(expon_compteur(x,p),compteur)

compteur=0

def f_compteur(x,p):
    if p==0 :
        return(1)
    else:
        global compteur
        compteur=compteur+1
        return(x*f_compteur(x,p-1))
```

**Exercice 3.** Dichotomie récursive.

Soit  $f$  une fonction qui s'annule sur l'intervalle  $[a, b]$ . L'objectif est de calculer une valeur approchée de  $c$  tel que  $f(c) = 0$ . On procède par dichotomie, en divisant à chaque étape l'intervalle de travail en deux. On note  $n$  le nombre d'étapes effectuées (ce nombre  $n$  décidera donc de la précision de la réponse). La fonction `dicho` prend comme argument  $f, a, b$  et  $n$  et renvoie une valeur approchée de  $c$ .

1. Donnez un critère simple pour vérifier si deux nombres  $x$  et  $y$  sont de signes opposés.

**Solution 6.**  $xy < 0$ .

2. Prenez une feuille et un stylo. A la manière de l'exercice 2, écrivez un algorithme qui décrit les étapes de la dichotomie en version récursive. Il démarre par : `dicho(f,a,b,n)`.

**Solution 7.** `dicho(f,a,b,n)`

On pose :  $c = \frac{a+b}{2}$ .

- Si  $n = 0$ , renvoyer  $c$ .
- Si  $f(a)f(c)$  est négatif, renvoyer `dicho(f,a,c,n-1)`.
- Sinon, renvoyer `dicho(f,c,b,n-1)`.

3. Tant que le programme sur feuille n'est pas correct, revenez à l'étape 2.

Implémentez l'algorithme précédent en langage python. Soit une fonction  $f(x) = x^3 + x^2 - 5$ , déterminer  $x \in [0, 2]$ , tel que  $f(x) = 0$ .

**Solution 8.**

```
def dichot(f,a,b,n):
    c=(a+b)/2.
    if n==0:
        return(c)
    else:
        if f(a)*f(c)<0:
            return(dichot(f,a,c,n-1))
        else:
            return(dichot(f,c,b,n-1))
```

**Exercice 4.** Enumeration des sous-listes.

Soit  $L$  une liste. L'objectif de l'exercice est d'écrire un programme qui renvoie toutes les sous-listes de  $L$ . Par exemple, pour  $L = [1, 2, 3]$ , ses sous-listes sont :

$$[1, 2, 3], [2, 3], [1, 3], [3], [1, 2], [2], [1], []$$

1. Ecrivez une fonction `SousListe` qui prend comme argument une liste et renvoie la liste de ses sous-listes.

**Solution 9.**

```
def SousListe(L):
    if len(L)==0:
        return([[]])
    DernierTerme=L[len(L)-1]
    AncienneSousListe=SousListe(L[0:len(L)-1])
    NouvelleSousListe=[]
    for i in AncienneSousListe:
        NouvelleSousListe.append(i+[DernierTerme])
    return(NouvelleSousListe+AncienneSousListe)
```

2. En maths, la fonction construit l'ensemble  $\mathcal{P}(L)$ . Vous savez qu'il est de cardinal  $2^{\text{Card}(L)}$ . Vérifiez la longueur de `SousListe(L)`.

**Solution 10.** `len(SousListe(L))==2**len(L)`

**Exercice 5.** Rendu de monnaie récursif.

La liste `monnaie_euro=[50,20,10,5,2,1]` contient l'ensemble des billets/pièces (dont le montant est un entier) du système monétaire européen dans l'ordre décroissant. Etant donné une valeur `restant_du`, on veut déterminer la liste des pièces nécessaires pour atteindre ce montant, obtenue par un algorithme glouton.

Proposez une fonction récursive `rendu` qui prend comme argument `restant_du` et `monnaie_a_rendre` et renvoie la liste demandée.

Par exemple : `rendu(216, [0,0,0,0,0,0])` doit renvoyer `[4,0,1,1,0,1]`.

**Solution 11.**

```
monnaie_euro=[50,20,10,5,2,1]
```

```
def rendu(m,L):
    if m==0:
        return(L)
    else:
        i=0
        while i<len(monnaie_euro) and m<monnaie_euro[i]:
            i=i+1
        L[i]=L[i]+1
        m=m-monnaie_euro[i]
        return(rendu(m,L))
```