CORRIGÉ - TP5 : RÉCURSIVITÉ

1 Exemple introductif

1. On a :
   `fact(4)=4 * fact(3)`
   `fact(3)=3 * fact(2)`
   `fact(2)=2 * fact(1)`
   `fact(1)=1 * fact(0)`
   `fact(0)=1`
   On a donc `fact(4)=4 * 3 * 2 * 1 * 1`.
2. On obtient :

```
>>> fact(4)
24
```

3. On modifie la fonction `fact` :

```
def fact(n):
    print(n)
    if n==0:
        return 1 #Cas de base
    else:
        return n*fact(n-1) #Appel recursif
```

et on obtient :

```
>>> fact(4)
4
3
2
1
0
24
```

4. L'appel `fact(-4)` conduit à une erreur. En effet, on a une suite d'appels à la fonction `fact` avec des arguments strictement décroissant. `n` n'atteint jamais la valeur 0, le programme ne se termine pas.

2 Premières applications de la récursivité

1.

```
def pgcd(a,b):
    if b==0:
        return a
    else:
        return pgcd(b,a%b)
```

2. (a)

```
def tracer_triangle(n):
    if n>0:
        afficher_ligne(n)
        tracer_triangle(n-1)
```

(b)

```
def tracer_triangle2(n):
    if n>0:
        tracer_triangle2(n-1)
        afficher_ligne(n)
```

3. Voici une première version de la fonction `u` :

```
def u(a,n):
    if n==0:
        return a
    return (u(a,n-1)+a/u(a,n-1))/2
```

Cette version appelle à chaque fois deux fois la fonstion `u`. La complexité est exponentielle. En testant, on s'aperçoit que le temps d'exécution est très long.

On propose une nouvelle version ne faisant cette fois qu'un seul appel récursif :

```
def u(a,n):
    if n==0:
        return a
    r = u(a,n-1)
    return (r+a/r)/2
```

4. On reprend ce qu'on a vu dans le TP précédent :

```
def expo_rapide(k,n):
    if n==0:
        return 1
    r = expo_rapide(k,n//2)
    if n%2==0:
        return r*r
    else:
        return k*r*r
```

5.

```
def bin(n):
    if n==0:
        return []
    return bin(n//2)+[n%2]
```

6. (a) On propose cette première version :

```
def fibo(n):
    if n==0 or n==1:
        return 1
    return fibo(n-2) + fibo(n-1)
```

(b) Cette version a une complexité linéaire :

```
def fibo(n):
    def aux(n): #aux(n) renvoie (F_(n-1),F_(n)), avec F_(-1) = 0
        if n==0:
            return(0,1)
        else:
            a,b = aux(n-1)
            return(b,a+b)
    return aux(n)[1]
```

3 Recherche dichotomique

1.

```
def recherche_dicho_bornes(L,x,a,b):
    if a > b:
        return False
    m = (a+b)//2
    if x<L[m]:
        return recherche_dicho_bornes(L,x,a,m-1)
    elif x>L[m]:
        return recherche_dicho_bornes(L,x,m+1,b)
    else:
        return True
```

2. On a :

```
def recherche_dicho(L,x):
    return recherche_dicho_bornes(L,x,0,len(L)-1)
```

4 Tours de Hanoï

```
def hanoi():
    def aux(d,t,a,n):
    #d est le numero de la pile de depart,
    #t le numero de la pile transitoire,
    #a le numero de la pile d'arrivee,
    #n le nombre de disques a deplacer.
        if n>0:
            aux(d,a,t,n-1)
            print(d,"->",a)
            aux(t,d,a,n-1)
    aux(1,2,3,7)
```

5 Flocon de Koch

1.

```
from turtle import *
```

2.

```
def tracer_carre():
    forward(100)
    right(90)
    forward(100)
    right(90)
    forward(100)
    right(90)
    forward(100)
```

3.

```
def koch(n,l):
    if n==0:
        forward(l)
    else:
        koch(n-1,l/3)
        left(60)
        koch(n-1,l/3)
        right(120)
        koch(n-1,l/3)
        left(60)
        koch(n-1,l/3)

def flocon(n,l):
    for _ in range(3):
        koch(n,l)
        right(120)
```

Fin