TP 11- Proposition de solutions

Solution 1 1. Considérons l'instance (15,4) :

Appels	Sorties
pgcd_rec(15,4)	1
pgcd_rec(4,3)	1
pgcd_rec(3,1)	1
pgcd_rec(1,0)	1

On remplit premièrement la première colonne en descendant (celles des appels), puis la seconde en remontant (celle des sorties).
2. Version mopdifiée :

```
def pgcd_rec(a,b,n=1):
    if b==0:
        return a,n
    else:
        return pgcd_rec(b,abs(a%b),n+1)
```

Solution 2 Ce programme affiche un triangle de +. Par exemple :

```
>>> f(5)
+++++
++++
+++
++
+
```

Solution 3 Factorielle

factorielle.py

```
def factorielle(n):
    if n==0:
        return 1
    else:
        return n*factorielle(n-1)
```

Ce qui donne :

```
>>> factorielle(12)
479001600
>>> factorielle(997)
40359 ...
>>> factorielle(998)
[...]
RuntimeError: maximum recursion depth exceeded ...
```

Solution 4 Diverses fonctions

1. Calcul de $\sqrt{5}$.
Attention ! Afin d'éviter le double appel, il convient de précalculer u_{n-1}. La complexité redevient linéaire.

racine.py

```
def r5(n):
    if n==0:
        return 2
    else:
        u=r5(n-1)
        return (u+5/u)/2
```

2. $\sum_{k=1}^{n} \frac{1}{k}$

```
def somme(n):
    if n==0:
        return 0
    else:
        return somme(n-1)+1/n
```

3. Le coefficient binomial

$\binom{n}{k}$

```
def Cbin(k,n):
    if k>n or n<0:
        return 0
    elif k==0:
        return 1
    else:
        return Cbin(k-1,n-1)*n/k
```

4. Calcul d'une système récurrent :

```
u=lambda n:2 if n==0 else (u(n-1)+v(n-1))/2
v=lambda n:1 if n==0 else
                2*u(n-1)*v(n-1)/(u(n-1)+v(n-1))
```

Solution 5 Tri par sélection

Tri par sélection

```
def tri_selection_R(L):
    if len(L)<2:
        return L
    else:
        p=pmin(L)
        v=L.pop(p)
    return [v]+tri_selection_R(L)
```

Une version récursive de pmin :

```
def pmin_R(L):
    if len(L)==1:
        return 0
    p=pmin_R(L[:-1])
    if L[-1]<L[p]:
        return len(L)-1
    else:
        return p
```

Solution 6 1. Racine d'une fonction :

```
def dicho_f(f,a,b,p):
    if abs(b-a)<2*p:
        return (a+b)/2
    c=(a+b)/2
    if f(a)*f(c)<=0:
        return dicho_f(f,a,c,p)
    else:
        return dicho_f(f,c,b,p)
```

2. Dans une liste ordonnée

```
def dicho_L(x,L):
    if x<L[0] or x>L[-1] or len(L)==0:
        return False
    c=len(L)//2
    if x==L[c]:
        return True
    elif x<L[c]:
        return dicho_L(x,L[:c])
    else:
        return dicho_L(x,L[c+1:])
```

Version avec la position de l'élément, il faut rajouter une variable d'entrée pour repérer la position du premier élément de la liste en cours dans la liste initiale :

```
def dicho_L(x,L,p=0):
    if x<L[0] or x>L[-1] or len(L)==0:
        return False,-1
    c=len(L)//2
    if x==L[c]:
        return True,c+p
    elif x<L[c]:
        return dicho_L(x,L[:c],p)
    else:
        return dicho_L(x,L[c+1:],p+c+1)
```

Solution 7 Maximum d'une liste

```
def max_L(L):
    if len(L)==1:
        return L[0]
    if L[0]<L[1]:
        return max_L(L[1:])
    else:
        return max_L([L[0]]+L[2:])
```