

TP corrigé : récursivité 1

1 Algorithmes classiques en récursif

```
[1]: # 1.  
def somme(n):  
    if n == 0:  
        return 0  
    else:  
        return somme(n-1) + n*n  
  
somme(3) # test
```

[1]: 14

```
[2]: # 2.  
def binom(n, k):  
    if k == 0: # 0 parmi n vaut 1  
        return 1  
    if n == 0: # k parmi 0 vaut 0  
        return 0  
    else:  
        return binom(n-1, k-1) + binom(n-1, k)  
  
binom(4, 2) # test
```

[2]: 6

```
[3]: # 3.  
# il est important de faire en sorte que f s'annule sur l'intervalle [a, b],  
# même dans les appels récursifs  
def dichotomie(f, a, b, epsilon):  
    m = (a+b)/2  
    if abs(f(m)) < epsilon:  
        return m  
    if f(a)*f(m) <= 0: # f s'annule sur l'intervalle [a, m]  
        return dichotomie(f, a, m, epsilon)  
    else: # f s'annule sur l'intervalle [m, b]  
        return dichotomie(f, m, b, epsilon)  
  
def g(x): # pour tester dichotomie
```

```

    return x**2 - 2

dicho(g, 0, 2, 0.001) # approximation d'un zéro de g, c'est à dire racine de 2

```

[3]: 1.41455078125

2 Algorithme d'Euclide

```

[4]: # 1.
def pgcd(a, b):
    if b == 0:
        return a
    else:
        return pgcd(b, a%b)

pgcd(12, 18) # test

```

[4]: 6

```

[5]: # 2.
# Par appel récursif on obtient d, u, v tels que d = u*b + v*r.
# Comme a = bq + r, r = a - bq
# Donc d = u*b + v*(a - bq) = v*a + (u - q*v)*b
# Les coefficients de Bézout a et b sont donc v et u - q*v

def bezout(a, b):
    if b == 0:
        return (a, 1, 0)
    (d, u, v) = bezout(b, a%b)
    return (d, v, u - (a//b)*v)

bezout(7, 25) # 1 = (-7)*7 + (-1)*25

```

[5]: (1, -7, 2)

3 Rendu de monnaie

```

[6]: # 1.

def rendu(n, L):
    if len(L) == 0:
        if n == 0: return 1
        else: return 0
    return rendu(n - L[-1], L[:-1]) + rendu(n, L[:-1])

```

```
rendu(6, [1, 2, 3, 5]) # test
```

[6]: 2

```
[7]: # 2.  
def ajouter(e, LL):  
    for i in range(len(LL)):  
        LL[i].append(e)
```

```
[8]: # 3.  
def rendu2(n, L):  
    if len(L) == 0:  
        if n == 0: return [[]]  
        else: return []  
    L1 = rendu2(n - L[-1], L[:-1])  
    ajouter(L[-1], L1)  
    return L1 + rendu2(n, L[:-1])  
  
rendu2(6, [1, 2, 3, 5]) # test
```

[8]: [[1, 5], [1, 2, 3]]