# TP 6 RÉCURSIVITÉ

```python
## Exercice 1

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



## Exercice 2

def maxr(a):
    if len(a) == 1:
        return a[0]
    else:
        M = maxr(a[1:])
        if M > a[0]:
            return M
        else:
            return a[0]



## Exercice 3

d = {}

def F(n):
    if n not in d:
        if n <= 1:
            res = n
        else:
            res = F(n-1) + F(n-2)
        d[n] = res
    return d[n]




# cpt = 0
#
# def compte(n):
#     global cpt
#     if n == 2:
#         cpt += 1
#     if n <= 1:
#         return n
#     else:
#         return compte(n-1) + compte(n-2)
```

```
## Exercice 4

t_ex = [[3], [7, 4], [2, 4, 6], [8, 5, 9, 3]]

def triangle_iteratif(t):
    n = len(t)
    # 1) création du tableau
    S = [[0 for j in range(n)] for i in range(n)]

    # 2) initialisation / cases "faciles"
    for j in range(n):
        S[n-1][j] = t[n-1][j]

    # 3) boucle pour remplir le tableau
    i = n-2
    while i >= 0:
        for j in range(i+1):
            S[i][j] = t[i][j] + max(S[i+1][j], S[i+1][j+1])
        i = i - 1

    # 4) on exploite la ou les case(s) utile(s)
    return S[0][0]



def Srec(t, i, j):
    if i == len(t)-1:
        return t[len(t)-1][j]
    else:
        return t[i][j] + max(Srec(t, i+1, j), Srec(t, i+1, j+1))


d = {}

def Srecmemo(t, i, j):
    if (i,j) not in d:
        if i == len(t)-1:
            res = t[len(t)-1][j]
        else:
            res = t[i][j] + max(Srecmemo(t, i+1, j), Srecmemo(t, i+1, j+1))
        d[(i,j)] = res
    return d[(i,j)]


def S(tab):
    d = {}
    def aux(t, i, j):
        if (i,j) not in d:
            if i == len(t)-1:
                res = t[len(t)-1][j]
            else:
                res = t[i][j] + max(aux(t, i+1, j), aux(t, i+1, j+1))
            d[(i,j)] = res
        return d[(i,j)]

    return aux(tab, 0, 0)



## Exercice 5


def opti(P, V, i, w):
    if i == 0:
        return 0
    elif P[i-1] > w:
        return opti(P, V, i-1, w)
    else:
        a = opti(P, V, i-1, w)
        b = V[i-1] + opti(P, V, i-1, w-P[i-1])
        return max(a, b)
```

```python
dico_opti = {}

def opti_memo(P, V, i, w):
    if (i, w) not in dico_opti:
        if i == 0:
            res = 0
        elif P[i-1] > w:
            res = opti_memo(P, V, i-1, w)
        else:
            a = opti_memo(P, V, i-1, w)
            b = opti_memo(P, V, i-1, w-P[i-1])
            res = max(a, V[i-1] + b)
        dico_opti[(i, w)] = res
    return dico_opti[(i, w)]


Pex = [3, 2, 1, 4]
Vex = [4, 3, 1, 9]
Pmaxex = 8

assert opti_memo(Pex, Vex, 4, Pmaxex) == 14
```