TP 5

RÉCURSIVITÉ

```python
## Exercice 1

def s_entiers(n):
    """
    Entrée : n (int) (n >= 1)
    Sortie : somme des entiers de 1 à n (type int)
    """
    if n == 0:
        return 0
    else:
        return n + s_entiers(n - 1)


# ----------------------------------------
assert s_entiers(1) == 1
assert s_entiers(2) == 3
assert s_entiers(10) == 10*(10 + 1)//2
# ----------------------------------------



## Exercice 2

from math import sqrt

def V(n):
    if n == 0:
        return sqrt(2)
    else:
        return sqrt(U(n-1) + 2)

''' Le cas de base est n=0. Lors de l'exécution de V(n), l'appel récursif
    porte sur n-1 donc on se "rapproche strictement" du cas de base et on
    l'atteint en un nombre fini d'étapes d'où la terminaison. '''


def W(n):
    if n == 0:
        return 0
    elif n%2 == 0:
        return W(n//2)
    else:
        return 1 - W(n//2)

''' Le cas de base est n=0. Lors de l'exécution de W(n), l'appel récursif
    porte sur n//2 (quel que soit le cas) donc on se "rapproche strictement"
    du cas de base et on l'atteint en un nombre fini d'étapes d'où la terminaison. '''
```

```
## Exercice 3

def suites_rec(a, b, n):
    if n == 0:
        return a, b
    else:
        u, v = suites_rec(a, b, n-1)
        return (u+v)/2, sqrt(u*v)


def suites_iter(a, b, n):
    u, v = a, b
    for k in range(1, n):
        u_ancien = u
        u = (u+v)/2
        v = sqrt(u_ancien * v)
    return u, v


from time import time
t0=time()
for _ in range(1000):
    suites_rec(5,10,500)
t1=time()
print('version récursive: ',(t1-t0)/1000)
t0=time()
for _ in range(1000):
    suites_iter(5,10,500)
t1=time()
print('version iterative: ',(t1-t0)/1000)

t0=time()
for _ in range(1000):
    suites_rec(1,2,500)
t1=time()
print('version récursive: ',(t1-t0)/1000)
t0=time()
for _ in range(1000):
    suites_iter(1,2,500)
t1=time()
print('version iterative: ',(t1-t0)/1000)



## Exercice 4

def exp_rec(x, n):
    """
    Entrée : x (float), n (int)
    Sortie : x**n (float)
    Calcul récursif simple.
    """

    if n == 0: # cas de base
        return 1
    else:
        return x*exp_rec(x, n - 1)


# ----------------------------
assert exp_rec(5, 0) == 1
assert exp_rec(5, 1) == 5
assert exp_rec(5, 7) == 5**7
# ----------------------------
```

```python
## Exercice 5

def expor_rec(x, n):
    """
    Entrée : x (float), n (int)
    Sortie : x**n (float)
    Calcul récursif d'exponentiation rapide.
    (n impair => (n-1)/2 = n//2)
    """

    if n == 0: # cas de base
        return 1
    elif n%2 == 0:
        return expor_rec(x*x, n//2)
    else:
        return x*expor_rec(x*x, n//2)


assert expor_rec(5, 0) == 1
assert expor_rec(5, 1) == 5
assert expor_rec(5, 7) == 5**7



## Exercice 6

def mr(a):
    """
    Entrée : a (list)
    Sortie : la valeur maximale de la liste a (int ou float)
    """

    if len(a) == 1:
        return a[0]
    else:
        x = mr(a[1:])
        if a[0] >= x:
            return a[0]
        else:
            return x


assert mr([1]) == 1
assert mr([1, 2]) == 2
assert mr([2, 1]) == 2
assert mr([1, 2, 3]) == 3
assert mr([1, 3, 2]) == 3
assert mr([3, 1, 2]) == 3



## Exercice 7

def inverse(ch):
    """
    Entrée : ch chaîne de caractères (str)
    Sortie : la chaîne ch inverséee (str)
    """

    if ch == '':
        return ''
    else:
        return inverse(ch[1:]) + ch[0]


assert inverse('') == ''
assert inverse('a') == 'a'
assert inverse('ab') == 'ba'
assert inverse('abc') == 'cba'
assert inverse('abcd') == 'dcba'
```

```python
## Exercice 8

def chemin_rec(t):
    def aux(t, i, j):
        if i == len(t)-1:
            return t[i][j]
        a = aux(t, i+1, j)
        b = aux(t, i+1, j+1)
        return t[i][j] + max(a, b)
    return aux(t, 0, 0)


# deuxième version avec mémoïsation

def chemin_rec_memo(t):
    d = {}
    def aux(t, i, j):
        if (i,j) not in d:
            if i == len(t)-1:
                d[(i,j)] = t[i][j]
            else:
                a = aux(t, i+1, j)
                b = aux(t, i+1, j+1)
                d[(i,j)] = t[i][j] + max(a, b)

        return d[(i,j)]

    return aux(t, 0, 0)



## Exercice 9

def monnaie(v, s):
    """
    Entrée :
        v (type 'int' ou 'float') : valeur de la somme à rendre.
        s (type 'list') : système de monnaie.
    Sortie :
        nombre minimal de pièces à rendre calculé récursivement ('int')
    On utilise :
        monnaie(v, s) = 1 + min { monnaie(v - p, s) pour p <= v }
    """

    if v == 0:
        return 0
    else:
        return 1 + min(monnaie(v - p, s) for p in s if v >= p)


# version avec mémoïsation

def monnaie_memo(v, s):
    def np(val, syst):
        if val in memo:
            return memo[val]
        else:
            if val == 0:
               return 0
            else:
                # Ne sont considérées que les pièces qui vérifient l'inégalité p <= val
                nombre_pieces = 1 + min([np(val - p, syst) for p in syst if p <= val])
                memo[val] = nombre_pieces
                return nombre_pieces

    memo = {}
    return np(v, s)
```