CHAPITRE 2

STRATÉGIES DE PROGRAMMATION

```
## Exemple 1

def gain1(a):
    g = 0
    for i in range(len(a)):
        for j in range(i+1, len(a)):
            if a[j]-a[i] > g:  # meilleure affaire ?
                g = a[j]-a[i]
    return g


def gain1_bis(a):
    g = 0
    i0, j0 = 0, 0 # dates de la meilleure vente
    for i in range(len(a)):
        for j in range(i+1, len(a)):
            if a[j]-a[i] > g: # meilleure affaire ?
                g = a[j]-a[i]
                i0, j0 = i, j
            elif a[j]-a[i] == g and j-i < j0-i0: # durée plus courte ?
                i0, j0 = i, j
    return(g, i0, j0)


''' Si la chute du taux i par rapport au taux i-1 est supérieure
        au gain courant en i-1 alors le gain courant en i est nul:
            gainCourant_i = max (0 , gainCourant_{i-1} + a[i] - a[i-1] ).
    Dans un tour de boucle, on tient constamment à jour le gain courant
        et on garde trace dans une variable du meilleur gain courant. '''


def gain2(a):
    gc = 0 # gain courant
    g = 0  # gain i.e. le max des gains courants
    for i in range(1,len(a)):
        if gc+a[i]-a[i-1] > 0:
            gc = gc + a[i]-a[i-1]
        else:
            gc = 0
        if gc > g:
            g = gc
    return g
```

```
def gain2_bis(a):
    gc = 0
    g = 0
    i = 0 # indice du prix minimum
    i0, j0 = 0, 0
    for j in range(1,len(a)):
        if gc + a[j] - a[j-1] > 0:
            gc += a[j] - a[j-1]
        else:
            gc = 0
            i = j
        if g < gc :
            g = gc
            i0, j0 = i, j
        if g == gc and j-i < j0-i0:
            i0, j0 = i, j
    return (g, i0, j0)




## Exemple 2

def monnaie_a_rendre (somme_a_rendre, systeme):
    """ entrée : int somme_a_rendre, list d'int systeme
        détermine le nombre de pièces pour chaque type du systeme
            afin d'obtenir somme_a_rendre
        utilise pour cela un algorithme glouton :
            on commence par les pièces de plus grande valeur
        sortie : list d'int correspondant au nombre de chaque pièce
    """

    liste_monnaie = [0 for _ in systeme]  # liste de 0 de même longueur que systeme
    k = len(systeme) - 1

    while somme_a_rendre > 0 and k >= 0:
        nombre_pieces = somme_a_rendre // systeme[k]
        somme_a_rendre = somme_a_rendre % systeme[k]
        liste_monnaie[k] = nombre_pieces
        k = k - 1

    return liste_monnaie


assert monnaie_a_rendre(8, [1, 4, 6]) == [2, 0, 1]
assert monnaie_a_rendre(49, [1, 3, 6, 12, 24, 30]) == [1, 0, 1, 1, 0, 1]
assert monnaie_a_rendre(11, [2, 4, 6]) == [0, 1, 1]
```

```
## Exemple du déplacement sur un damier

# initialisation
''' f(i,0) = somme des m_{k,0} pour k allant de 0 à i
    f(0,j) = somme des m_{0,k} pour k allant de 0 à j
'''

# formule de récurrence
''' f(i,j) = m_{i,j} + min (f(i-1,j) + f(i,j-1))
'''

# version itérative

def damier(m):
    """ entrée : tableau (liste de listes) d'entiers
                    correspondant à un damier pondéré
        sortie : entier correspondant au parcours de poids minimal
    """
    n, p = len(m), len(m[0])
    f = [[0 for j in range(p)] for i in range(n)]  # tableau de 0
    f[0][0] = m[0][0]                              # case initiale
    for i in range(n-1):                           # première colonne
        f[i+1][0] = m[i+1][0] + f[i][0]
    for j in range(p-1):                           # première ligne
        f[0][j+1] = m[0][j+1] + f[0][j]
    for i in range(1, n):                          # autres cases
        for j in range(1, p):
            f[i][j] = m[i][j] + min(f[i-1][j], f[i][j-1])
    return f[n-1][p-1]       # le résultat est le contenu de la "dernière case"


# pour créer un exemple :
# from random import randint
# D = [[randint(1,10) for _ in range(3)] for _ in range(4)]


# version avec un chemin optimal reconstitué
def damier_bis(m):
    n, p = len(m), len(m[0])
    f = [[0 for j in range(p)] for i in range(n)]
    f[0][0] = m[0][0]
    for i in range(n-1):
        f[i+1][0] = m[i+1][0] + f[i][0]
    for j in range(p-1):
        f[0][j+1] = m[0][j+1] + f[0][j]
    for i in range(1, n):
        for j in range(1, p):
            f[i][j] = m[i][j] + min(f[i-1][j], f[i][j-1])
    c = [(n, p)]
    i, j = n-1, p-1
    while i>0 and j>0:
        if f[i][j] - m[i][j] == f[i-1][j]:
            i -= 1
        else:
            j -= 1
        c.append((i, j))
    if i == 0:
        while j>0:
            j -= 1
            c.append((i,j))
    else:
        while i>0:
            i -= 1
            c.append((i,j))

    c.reverse()

    return f[n-1][p-1], c
```

```python
# version récursive naïve

def damier_rec(m, a, b):
    """ entrée : m tableau (liste de listes) d'entiers
                    correspondant à un damier pondéré
                a, b (int) coordonnées d'une case
        sortie : entier correspondant au parcours de poids minimal
    """
    if a == 0 and b == 0:
        return m[0][0]
    if a == 0:
        return m[0][b] + damier_rec(m, 0, b-1)
    if b == 0:
        return m[a][0] + damier_rec(m, a-1, 0)
    gauche = damier_rec(m, a-1, b)
    haut = damier_rec(m, a, b-1)
    return m[a][b] + min(gauche, haut)



# version récursive avec mémoïsation

cases = {}  # dictionnaire pour mémoïser

def damier_rec_memo(m, a, b):
    if (a, b) not in cases:
        if a == 0 and b == 0:
            res = m[0][0]
        elif a == 0:
            res = m[0][b] + damier_rec_memo(m, 0, b-1)
        elif b == 0:
            res = m[a][0] + damier_rec_memo(m, a-1, 0)
        else:
            gauche = damier_rec_memo(m, a-1, b)
            haut = damier_rec_memo(m, a, b-1)
            res = m[a][b] + min(gauche, haut)
        cases[(a, b)] = res
    return cases[(a,b)]



# version avec le dictionnaire dans la fonction principale

def damier_rec_memo_main(m):
    n, p = len(m), len(m[0])
    cases = {}
    def aux(a, b):
        if (a, b) not in cases:
            if a == 0 and b == 0:
                res = m[0][0]
            elif a == 0:
                res = m[0][b] + aux(0, b-1)
            elif b == 0:
                res = m[a][0] + aux(a-1, 0)
            else:
                gauche = aux(a-1, b)
                haut = aux(a, b-1)
                res = m[a][b] + min(gauche, haut)
            cases[(a, b)] = res
        return cases[(a,b)]
    return aux(n-1, p-1)
```