# TD n°21 - Exercices BAC

Thème : BAC

21

# TD : Exercices Divers BAC - Programmation

## Sujet 1 : Tableaux et Programmation

On rappelle que len est une fonction qui prend un tableau en paramètre et renvoie sa longueur.
C'est-à-dire le nombre d'éléments présents dans le tableau.

***Exemple*** : `len([12, 54, 34, 57])` vaut 4.

Le but de cet exercice est de programmer différentes réductions pour un site de vente de vêtements en ligne.
On rappelle que si le prix d'un article avant réduction est de x euros,

- son prix vaut 0, 5x si on lui applique une réduction de 50%,
- son prix vaut 0, 6x si on lui applique une réduction de 40%,
- son prix vaut 0, 7x si on lui applique une réduction de 30%,
- son prix vaut 0, 8x si on lui applique une réduction de 20%,
- son prix vaut 0, 9x si on lui applique une réduction de 10%.

Dans le système informatique du site de vente, l'ensemble des articles qu'un client veut acheter, appelé panier, est modélisé par un tableau de flottants.
Par exemple, si un client veut acheter un pantalon à 30,50 euros, un tee-shirt à 15 euros, une paire de chaussettes à 6 euros, une jupe à 20 euros, une paire de collants à 5 euros, une robe à 35 euros et un short à 10,50 euros, le système informatique aura le tableau suivant :

**Script Python**

```
tab = [30.5, 15.0, 6.0, 20.0, 5.0, 35.0, 10.5]
```

## Question 1. (a)

**Enoncé**

Écrire une fonction Python total_hors_reduction ayant pour argument le tableau des prix des articles du panier d'un client et renvoyant le total des prix de ces articles.

**Solution**

**Script Python**

```
def total_hors_reduction(tab):
    total = 0
    for pa in tab:
        total = total + pa
    return total
```

## Question 1. (b)

**Enoncé**

Le site de vente propose la promotion suivante comme offre de bienvenue : 20% de réduction sur le premier article de la liste, 30% de réduction sur le deuxième article de la liste (s'il y a au moins deux articles) et aucune réduction sur le reste des articles (s'il y en a).
Recopier sur la copie et compléter la fonction Python offre_bienvenue prenant en paramètre le tableau tab des prix des articles du panier d'un client et renvoyant le total à payer lorsqu'on leur applique l'offre de bienvenue.

Script Python

```
1  def offre_bienvenue (tab ):
2      """ tableau -> float """
3      somme =0
4      longueur =len(tab )
5      if longueur > 0 :
6          somme =tab [0]*...
7      if longueur > 1 :
8          somme = somme + ...
9      if longueur > 2 :
10         for i in range (2, longueur ):
11             somme =...
12     return ...
```

**Solution**

**Script Python**

```
def offre_bienvenue(tab):
    somme = 0
    longueur=len(tab)
    if longueur > 0:
        somme = tab[0]*0.8
    if longueur > 1:
        somme = somme + tab[1]*0.7
    if longueur > 2:
        for i in range(2,longueur):
            somme=somme+tab[i]
    return somme
```

Pour toute la suite de l'exercice, on pourra utiliser la fonction total_hors_reduction même si la question 1 n'a pas été traitée.

## Question 2.

**Enoncé**

Lors de la période des soldes, le site de vente propose les réductions suivantes :

- si le panier contient 5 articles ou plus, une réduction globale de 50%,
- si le panier contient 4 articles, une réduction globale de 40%,
- si le panier contient 3 articles, une réduction globale de 30%,
- si le panier contient 2 articles, une réduction globale de 20%,
- si le panier contient 1 article, une réduction globale de 10%.

Proposer une fonction Python prix_solde ayant pour argument le tableau tab des prix des articles du panier d'un client et renvoyant le total des prix de ces articles lorsqu'on leur applique la réduction des soldes.

**Solution**

**Script Python**

```
def prix_solde(tab):
    longueur = len(tab)
    reduc = 1
    if longueur == 1 :
        reduc = 0.9
    if longueur == 2 :
        reduc = 0.8
    if longueur == 3  :
        reduc = 0.7
    if longueur == 4 :
        reduc = 0.6
    if longueur >= 5 :
        reduc = 0.5
    return reduc*total_hors_reduction(tab)
    ```

    autre possibilité plus “courte” :
```python
def prix_soldeb(tab):
    return total_hors_reduction(tab)*(1-0.1*min(5,len(tab)))
```

## Question 3. (a)

**Enoncé**

Écrire une fonction minimum qui prend en paramètre un tableau tab de nombres et renvoie la valeur minimum présente dans le tableau.

**Solution**

Dans cette question nous partons du principe que tab n'est pas vide.

**Script Python**

```
def minimum(tab):
    mini = tab[0]
    for pa in tab:
        if pa < mini:
            mini = pa
    return mini
```

## Question 3. (b)

**Enoncé**

Pour ses bons clients, le site de vente propose une offre promotionnelle, à partir de 2 articles achetés, l'article le moins cher des articles commandés est offert.
Écrire une fonction Python offre_bon_client ayant pour paramètre le tableau des prix des articles du panier d'un client et renvoyant le total à payer lorsqu'on leur applique l'offre bon client.

**Solution**

**Script Python**

```
def offre_bon_client(tab):
    longueur = len(tab)
    total = total_hors_reduction(tab)
    if longueur >= 2:
        total = total - minimum(tab)
    return total
```

**Question 4.**
Afin de diminuer le stock de ses articles dans ses entrepôts, l'entreprise imagine faire l'offre suivante à ses clients : en suivant l'ordre des articles dans le panier du client, elle considère les 3 premiers articles et offre le moins cher, puis les 3 suivants et offre le moins cher et ainsi de suite jusqu'à ce qu'il reste au plus 2 articles qui n'ont alors droit à aucune réduction.
***Exemple :*** Si le panier du client contient un pantalon à 30,50 euros, un tee-shirt à 15 euros, une paire de chaussettes à 6 euros, une jupe à 20 euros, une paire de collants à 5 euros, une robe à 35 euros et un short à 10,50 euros, ce panier est représenté par le tableau suivant :

**Script Python**

```
tab = [30.5, 15.0, 6.0, 20.0, 5.0, 35.0, 10.5]
```

Pour le premier groupe (le pantalon à 30,50 euros, le tee-shirt à 15 euros, la paire de chaussettes à 6 euros), l'article le moins cher, la paire de chaussettes à 6 euros, est offert. Pour le second groupe (la jupe à 20 euros, la paire de collants à 5 euros, la robe à 35 euros), la paire de collants à 5 euros est offerte.
Donc le total après promotion de déstockage est 111 euros.
On constate que le prix après promotion de déstockage dépend de l'ordre dans lequel se présentent les articles dans le panier.

### Question 4.(a)

**Enoncé**

Proposer un panier contenant les mêmes articles que ceux de l'exemple mais ayant un prix après promotion de déstockage différent de 111 euros.

**Solution**

plusieurs solutions possibles
[30.5, 20.0, 35.0, 15.0, 6.0, 5.0, 10.5] => total après promotion = 122 - 20 - 5 = 97

### Question 4. (b)

**Enoncé**

Proposer un panier contenant les mêmes articles mais ayant le prix après promotion de déstockage le plus bas possible.

**Solution**

[35, 30.5, 20.0, 15.0, 10.5, 6.0, 5.0] => total après promotion = 122 - 20 - 6 = 96

### Question 4. (c)

=== "Enoncéé Une fois ses articles choisis, quel algorithme le client peut-il utiliser pour modifier son panier afin de s'assurer qu'il obtiendra le prix après promotion de déstockage le plus bas possible ? On ne demande pas d'écrire cet algorithme.

**Solution**

Pour avoir le prix après promotion de déstockage le plus bas possible, il faut trier le tableau dans l'ordre décroissant. On peut donc utiliser un algorithme de tri (tri par sélection, tri par insertion ou tri fusion)

## Sujet 2 : Programmation en Général

*Cet exercice porte sur la programmation en général.*

Étant donné un tableau non vide de nombres entiers relatifs, on appelle sous-séquence une suite non vide d'éléments voisins de ce tableau. On cherche dans cet exercice à déterminer la plus grande somme possible obtenue en additionnant les éléments d'une sous-séquence.

Par exemple, pour le tableau ci-dessous, la somme maximale vaut 18.
Elle est obtenue en additionnant les éléments de la sous-séquence encadrée en gras ci-dessous (6 ; 8 ; -6 ; 10).

| -8 | -4 | **6** | **8** | **-6** | **10** | -4 | -4 |
|---|---|---|---|---|---|---|---|

### Question 1. a.

**Enoncé**

Quelle est la solution du problème si les éléments du tableau sont tous positifs ?

**Solution**

Si les éléments du tableau sont tous positifs, il suffit d'additionner tous les éléments du tableau pour obtenir la somme maximale (la sous-séquence correspond à l'ensemble du tableau).

### Question 1. b.

**Enoncé**

Quelle est la solution du problème si tous les éléments sont négatifs ?

**Solution**

Si les éléments du tableau sont tous négatifs, il suffit de prendre l'élément le plus grand du tableau (la sous-séquence est réduite à un seul élément)

Dans cette question, on examine toutes les sous-séquences possibles.

## Question 2. a.

**Enoncé**

Écrire le code Python d'une fonction `somme_sous_sequence(lst, i, j)` qui prend en argument une liste et deux entiers `i, j` et renvoie la somme de la sous-séquence délimitée par les indices i et j (inclus).

**Solution**

**Script Python**

```
def somme_sous_sequence(lst, i, j):
    somme = 0
    for ind in range(i,j+1):
        somme = somme + lst[ind]
    return somme
```

## Question 2. b.

**Enoncé**

La fonction `pgsp` ci-dessous permet de déterminer la plus grande des sommes obtenues en additionnant les éléments de toutes les sous-séquences possibles du tableau lst.

**Script Python**

```
def pgsp(lst):
    n = len(lst)
    somme_max = lst[0]
    for i in range(n):
        for j in range(i, n):
            s = somme_sous_sequence(lst, i, j)
            if s > somme_max :
                somme_max = s
    return somme_max
```

Parmi les quatre choix suivants, quel est le nombre de comparaisons effectuées par cette fonction si le tableau lst passé en paramètre contient 10 éléments ?

- 10
- 55
- 100
- 1055

**Solution**

Pour un tableau de 10 éléments, nous avons 55 comparaisons $(10+9+8+7+6+5+4+3+2+1=55)$.

### Question 2. c.

**Enoncé**

Recopier et modifier la fonction pgsp pour qu'elle renvoie un tuple contenant la somme maximale et les indices qui délimitent la sous-séquence correspondant à cette somme maximale.

**Solution**

```
python
    def pgsp(lst):
        n = len(lst)
        somme_max = lst[0]
        i_max = 0
        j_max = 0
        for i in range(n):
            for j in range(i,n):
                s = somme_sous_sequence(lst,i,j)
                if s > somme_max:
                    somme_max = s
                    i_max = i
                    j_max = j
        return (somme_max, i_max, j_max)
```

**Question 3.**
On considère dans cette question une approche plus élaborée. Son principe consiste, pour toutes les valeurs possibles de l'indice $i$, à déterminer la somme maximale $S(i)$ des sous-séquences qui se terminent à l'indice $i$.
En désignant par `lst[i]` l'élément de lst d'indice $i$, on peut vérifier que

- S(0) = lst[0]
- et pour $i \geq 1$ :
  - $S(i) = lst[i]$ si $S(i-1) \leq 0$ ;
  - $S(i) = lst[i] + S(i-1)$ si $S(i-1) > 0$.

## Question 3. a.

**Enoncé**

Recopier et compléter le tableau ci-dessous avec les valeurs de $S(i)$ pour la liste considérée en exemple.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -8 | -4 | 6 | 8 | -6 | 10 | -4 | -4 |

**Solution**

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| lst[i] | -8 | -4 | 6 | 8 | -6 | 10 | -4 | -4 |
| S(i) | -8 | -4 | 6 | 14 | 8 | 18 | 14 | 10 |

### Question 3. b.

**Enoncé**

La solution au problème étant la plus grande valeur des $S(i)$, on demande de compléter la fonction `pgsp2` ci-dessous, de sorte que la variable sommes_max contienne la liste des valeurs $S(i)$.

**Script Python**

```
def pgsp2(lst):
    sommes_max = [lst[0]]
    for i in range(1, len(lst)):
        # à compléter

return max(sommes_max)
```

**Solution**

**Script Python**

```
def pgsp2(lst):
    somme_max = [lst[0]]
    for i in range (1,len(lst)):
        if somme_max[i-1] <= 0:
            somme_max.append(lst[i])
        else :
            somme_max.append(lst[i]+somme_max[i-1])
    return max(somme_max)
```

### Question 3. c.

**Enoncé**

En quoi la solution obtenue par cette approche est-elle plus avantageuse que celle de la question 2.b. ?

**Solution**

Cette solution est plus avantageuse, car la complexité en temps de l'algorithme est en $O(n)$ alors que dans le cas précédent il était en $O(n^2)$.

## Sujet 3 : Tableau - Parcours - Programmation en Général

*Cet exercice porte sur l'algorithmique et la programmation en Python. Il aborde les notions de tableaux de tableaux et d'algorithmes de parcours de tableaux.*

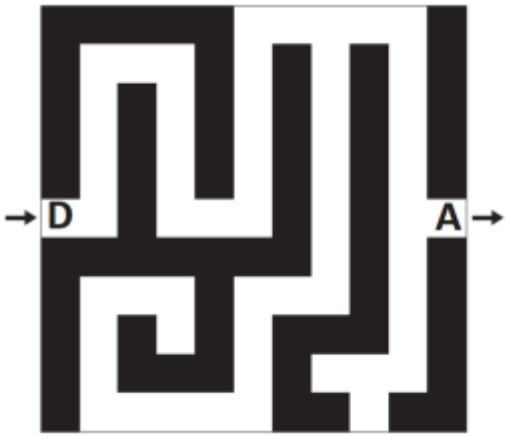
**Partie A : Représentation d'un labyrinthe**

On modélise un labyrinthe par un tableau à deux dimensions à $n$ lignes et $m$ colonnes avec $n$ et $m$ des entiers strictement positifs.
Les lignes sont numérotées de 0 à $n-1$ et les colonnes de 0 à $m-1$.
La case en haut à gauche est repérée par $(0,0)$ et la case en bas à droite par $(n-1,m-1)$.

Dans ce tableau :

- 0 représente une case vide, hors case de départ et arrivée,
- 1 représente un mur,
- 2 représente le départ du labyrinthe,
- 3 représente l'arrivée du labyrinthe.

Ainsi, en Python, le labyrinthe ci-dessous est représentée par le tableau de tableaux `lab1`.



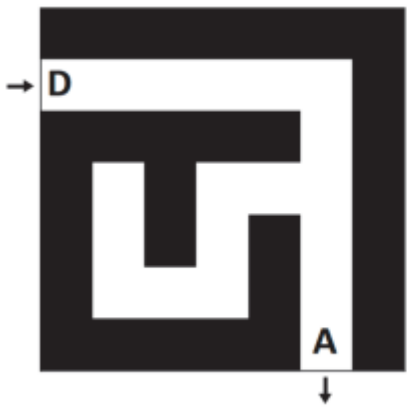

```
lab1 = [[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1],
        [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
        [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
        [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
        [2, 0, 1, 0, 0, 0, 1, 0, 1, 0, 3],
        [1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1],
        [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
        [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1]]
```

## Question A.1.

**Enoncé**

Le labyrinthe ci-dessous est censé être représenté par le tableau de tableaux lab2.
Cependant, dans ce tableau, un mur se trouve à la place du départ du labyrinthe.
Donner une instruction permettant de placer le départ au bon endroit dans lab2.

```




```
lab2 = [[1, 1, 1, 1, 1, 1, 1],
        [1, 0, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 0, 1],
        [1, 0, 1, 0, 0, 0, 1],
        [1, 0, 1, 0, 1, 0, 1],
        [1, 0, 0, 0, 1, 0, 1],
        [1, 1, 1, 1, 1, 3, 1]]
```

**Solution**

**Script Python**

```
lab2[1][0] = 2
```
```

## Question A.2.

**Enoncé**

Écrire une fonction `est_valide(i, j, n, m)` qui renvoie `True` si le couple $(i, j)$ correspond à des coordonnées valides pour un labyrinthe de taille $(n, m)$, et `False` sinon.
On donne ci-dessous des exemples d'appels.

**Script Python**

```
>>> est_valide(5, 2, 10, 10)
True
>>> est_valide(-3, 4, 10, 10)
False
```

**Solution**

**Script Python**

```
def est_valide(i,j,n,m):
    return i>=0 and j>=0 and i<n and j<m
```

### Question A.3.

=== "Enoncé On suppose que le départ d'un labyrinthe est toujours indiqué, mais on ne fait aucune supposition sur son emplacement. Compléter la fonction depart(lab)ci-dessous de sorte qu'elle renvoie, sous la forme d'un tuple, les coordonnées du départ d'un labyrinthe (représenté par le paramètre lab). Par exemple, l'appel depart(lab1) doit renvoyer le tuple (5, 0).

**Texte**

```python
def depart(lab) :
    n = len(lab)
    m = len(lab[0])
    ...
```

Solution

**Script Python**

```python
def depart(lab):
    n = len(lab)
    m = len(lab[0])
    for i in range(n):
        for j in range(m):
            if lab[i][j]==2:
                return (i,j)
```

### Question A.4.

**Enoncé**

Écrire une fonction nb_cases_vides(lab) qui renvoie le nombre de cases vides d'un labyrinthe (comprenant donc l'arrivée et le départ).
Par exemple, l'appel nb_cases_vides(lab2) doit renvoyer la valeur 19.

**Solution**

**Script Python**

```
def nb_cases_vides(lab):
    n = len(lab)
    m = len(lab[0])
    compt = 0
    for i in range(n):
        for j in range(m):
            if lab[i][j]==2 or lab[i][j]==3 or lab[i][j]==0:
                compt = compt + 1
    return compt
```

**Partie B : Recherche d'une solution dans un labyrinthe**

On suppose dans cette partie que les labyrinthes possèdent un unique chemin allant du départ à l'arrivée sans repasser par la même case. Dans la suite, c'est ce chemin que l'on appellera solution du labyrinthe.
Pour déterminer la solution d'un labyrinthe, on parcourt les cases vides de proche en proche.
Lors d'un tel parcours, afin d'éviter de tourner en rond, on choisit de marquer les cases visitées.
Pour cela, on remplace la valeur d'une case visitée dans le tableau représentant le labyrinthe par la valeur 4.

### Question B.1.

**Enoncé**

On dit que deux cases d'un labyrinthe sont voisines si elles ont un côté commun.
On considère une fonction `voisines(i, j, lab)` qui prend en arguments deux entiers $i$ et $j$ représentant les coordonnées d'une case et un tableau `lab` qui représente un labyrinthe.
Cette fonction renvoie la liste des coordonnées des cases voisines de la case de coordonnées $(i, j)$ qui sont valides, non visitées et qui ne sont pas des murs. L'ordre des éléments de cette liste n'importe pas.
Ainsi, l'appel `voisines(1, 1, [[1, 1, 1], [4, 0, 0], [1, 0, 1]])` renvoie la liste `[(2, 1), (1, 2)]`.
Que renvoie l'appel `voisines(1, 2, [[1, 1, 4], [0, 0, 0], [1, 1, 0]])` ?

**Solution**

L'appel de la fonction renvoie : [(2, 2), (1, 1)]

**Question 2.**
On souhaite stocker la solution dans une liste chemin. Cette liste contiendra les coordonnées des cases de la solution, dans l'ordre. Pour cela, on procède de la façon suivante.

- Initialement :
  - déterminer les coordonnées du départ : c'est la première case à visiter ;
  - ajouter les coordonnées de la case départ à la liste chemin.
- Tant que l'arrivée n'a pas été atteinte :
  - on marque la case visitée avec la valeur 4 ;
  - si la case visitée possède une case voisine libre, la première case de la liste renvoyée par la fonction `voisines` devient la prochaine case à visiter et on ajoute à la liste chemin ;
  - sinon, il s'agit d'une impasse. On supprime alors la dernière case dans la liste chemin. La prochaine case à visiter est celle qui est désormais en dernière position de la liste chemin.

## Question B.2. a.

=== "Enoncé Le tableau de tableaux lab3 ci-dessous représente un labyrinthe.

**Texte**

```
```python
lab3 =  [[1, 1, 1, 1, 1, 1],
        [2, 0, 0, 0, 0, 3],
        [1, 0, 1, 0, 1, 1],
        [1, 1, 1, 0, 0, 1]]
```

La suite d'instructions ci-dessous simule le début des modifications subies par la liste chemin lorsque l'on applique la méthode présentée.
```python
# entrée: (1, 0), sortie (1, 5)
chemin = [(1, 0)]
chemin.append((1, 1))
chemin.append((2, 1))
chemin.pop()
chemin.append((1, 2))
chemin.append((1, 3))
chemin.append((2, 3))
```

Compléter cette suite d'instructions jusqu'à ce que la liste chemin représente la solution.
Rappel : la méthode pop supprime le dernier élément d'une liste et renvoie cet élément.
```

Solution

**Script Python**

```python
# entrée: (1, 0), sortie (1, 5)
chemin = [(1, 0)]
chemin.append((1,1))
chemin.append((2,1))
chemin.pop()
chemin.append((1,2))
chemin.append((1,3))
chemin.append((2,3))
chemin.append((3,3))
chemin.append((3,4))
chemin.pop()
chemin.pop()
chemin.pop()
chemin.append((1,4))
chemin.append((1,5))
```

## Question B.2. b.

**Enoncé**

Recopier et compléter la fonction solution(lab) donnée ci-dessous de sorte qu'elle renvoie le chemin solution du labyrinthe représenté par le paramètre lab.

On pourra pour cela utiliser la fonction voisines.

**Script Python**

```
def solution(lab):
    chemin = [depart(lab)]
    case = chemin[0]
    i = case[0]
    j = case[1]
    ---
```

Par exemple, l 'appel `solution(lab2)` doit renvoyer `[(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5), (6, 5)]`.*

**Solution**

**Script Python**

```
def solution(lab):
    chemin = [depart(lab)]
    case = chemin[0]
    i = case[0]
    j = case[1]
    while lab[i][j] != 3:
        lab[i][j]=4
        v = voisines(i,j,lab)
        if len(v) != 0 :
            prochaine = v.pop()
            chemin.append(prochaine)
            i = prochaine[0]
            j = prochaine[1]
        else :
            chemin.pop()
            n = len(chemin)
            i = chemin[n-1][0]
            j = chemin[n-1][1]
    return chemin
```