

# Tableau unidimensionnel [tb]

## Support de Cours

Karine Zampieri, Stéphane Rivière

Unisciel  algoprog  Version 18 mai 2018

## Table des matières

<b>1 Définitions et notations</b>	<b>3</b>
1.1 Qu'est-ce qu'un tableau ?	3
1.2 Déclaration et initialisation d'un tableau	4
1.3 Exemple : Déclaration et initialisation	5
1.4 Accès indiciel	6
1.5 Exemple : Accès indiciel	7
<b>2 Tableaux et paramètres</b>	<b>8</b>
2.1 Synonyme de type	8
2.2 Tableau et paramètres	9
2.3 Tableau et fonction	10
2.4 Exemple : Saisie et affichage d'un tableau	11
<b>3 Parcours d'un tableau</b>	<b>13</b>
3.1 Parcours complet	13
3.2 Parcours partiel	14
3.3 Parcours imbriqué	16

## Python - Tableau unidimensionnel (Cours)



**Mots-Clés** Tableau unidimensionnel, Parcours d'un tableau ■

**Requis** Structures de base, Structures conditionnelles, Algorithmes paramétrés, Structures répétitives, Schéma itératif ■

**Difficulté** ●●○



### Introduction

De par leur praticité, les **tableaux** sont omniprésents en algorithmique et programmation : une variable regroupant sous le même nom plusieurs valeurs de même type accessibles par leur position.

Ce module donne les définitions et notations de **tableau sur une dimension** puis décrit la transmission des tableaux ainsi que ses parcours (complet, partiel, imbriqué).

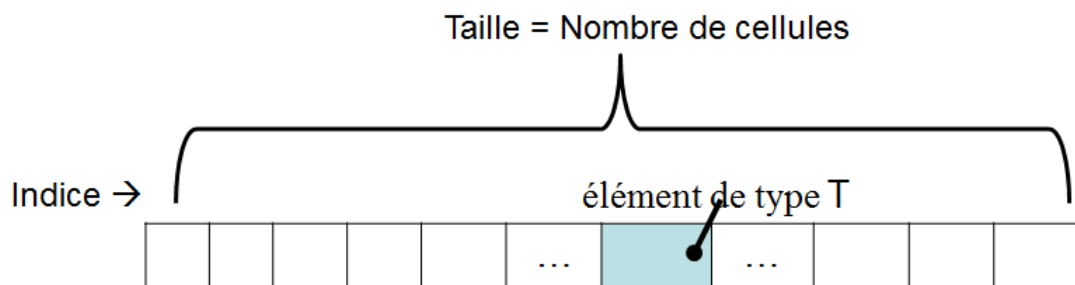
# 1 Définitions et notations

## 1.1 Qu'est-ce qu'un tableau ?



### Tableau, Type du tableau, Taille

Un **tableau** (sous-entendu monodimensionnel ou linéaire) est une collection **homogène** **indignée** sur  $\mathbb{N}$ , c.-à.d. une séquence d'éléments de même type portant tous le même nom mais se distinguant les uns des autres par un indice. Le **type du tableau** est le type de ses éléments. Sa **taille** ou *capacité* est le nombre (strictement positif) de ses éléments.



### Il n'y a pas de trou

Tous les éléments existent entre le premier et le dernier indice.



### Indice (ou index ou rang)

Entier donnant la **position** d'un élément dans la séquence. Cet indice varie entre la position du premier élément et la position du dernier élément, ces positions correspondant aux bornes de l'indice.



### Un Indice = Une case

A chaque valeur de l'indice ne correspond qu'une et une seule case du tableau, donc un élément.

### Taille statique v.s. variable

La taille d'un tableau ne peut pas être modifiée pendant son utilisation. On qualifie ce genre de tableau de **taille statique** (nombre fixe d'éléments). Nous verrons ultérieurement qu'il existe des tableaux de **taille variable** et appelés vecteurs ou tableaux dynamiques. Ce module ne traite que des premiers.

### Taille logique v.s. physique

Comme on utilisera un tableau plus grand que le nombre utile de ses éléments, on parle aussi de taille **logique**, dite aussi **effective** (le nombre d'éléments effectivement utilisés), que l'on oppose à la taille **physique** (la taille maximale du tableau).

## 1.2 Déclaration et initialisation d'un tableau



### Déclaration/Création d'un tableau

```
nomTab = [valInitTypeElement for x in range(taille)]
```

#### Explication

Déclare une variable dimensionnée. Avec : `TypeElement` le type (simple ou non) des éléments constitutifs du tableau, `nomTab` l'identifiant et `taille` son nombre d'éléments. La taille doit être une valeur entière positive (littéraux ou expressions constantes).



### Déclaration et initialisation

```
nomTab = [val1, ..., valN]
```

#### Explication

Déclare et initialise un tableau : la longueur de la liste détermine le nombre d'éléments.



### Python : Tableau numpy

Pour déclarer/créer un tableau numpy

```
import numpy as np
nomTab = np.zeros(taille, TypeElement)
```

Pour déclarer/initialiser un tableau numpy :

```
import numpy as np
nomTab = np.array([val1, ..., valN], dtype=TypeElement, order='C')
```

### 1.3 Exemple : Déclaration et initialisation



#### Exemple Python

```
tab = [0 for x in range(8)]  
NTEMPMAX = 366  
temp = [0.0 for x in range(NTEMPMAX)]  
message = 25 * " "
```

#### Explication

La variable `tab` est un tableau de 8 cases de type entier, `temp` un tableau de `NTEMPMAX` cases de réels et `message` un tableau de 25 caractères.

## 1.4 Accès indicial



### Accès indicial

`tab[k]`

### Explication

Accède à la case d'indice `k` d'un tableau `tab`.

Le temps d'accès à l'élément est fixe.

### Numérotation des cases

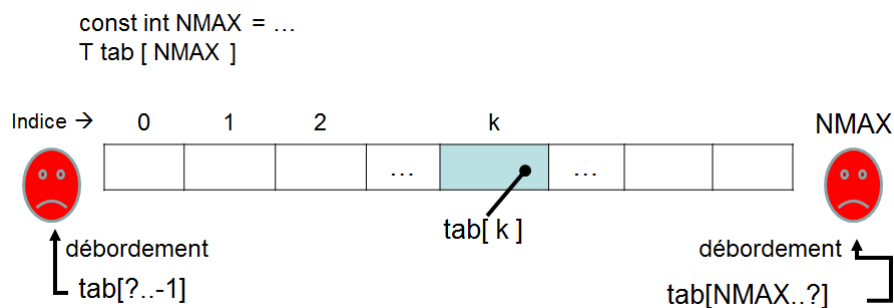
Chaque langage de programmation possède sa propre convention.

- **alg** : Les cases sont numérotées de 1 (par défaut) à `TMAX` (taille du tableau).
- **C/C++, Java, Python** : Ils commencent à indiquer un tableau à partir de 0. Ce principe est dit l'**indexation en base 0**.
- **Basic** : Il débute la numérotation à partir de 1 ou 0.
- **Ada** : Il permet de numéroté les cases à partir d'une valeur quelconque.



### Dépassement des bornes

Les langages contrôlent le débordement des bornes d'un tableau et déclenchent une erreur qui généralement arrête le programme.



## 1.5 Exemple : Accès indiciel



### Exemple Python

Soit le tableau déclaré ainsi :

```
tab = [0 for x in range(100)]
```

### Explication

Il est interdit d'utiliser `tab[-1]` ou `tab[100]`. De plus, chaque élément `tab[j]` (avec `j` dans `[0..99]`) doit être manié avec la même précaution qu'une variable simple, c.-à-d. qu'on ne peut utiliser un élément du tableau qui n'aurait pas été préalablement affecté ou initialisé.

## 2 Tableaux et paramètres

### 2.1 Synonyme de type



#### Synonyme de type

Alias d'un type existant (lorsqu'un nom de type est trop long ou est difficile à manipuler).



#### Synonyme de type

N'existe pas.



#### Typedef = Définition

N'introduit pas de nouveau type mais un **nouveau nom** pour le type.



## 2.2 Tableau et paramètres

Le passage par valeur a pour conséquence de créer une copie de l'entité passé en paramètre effectif. Dans le cas d'un tableau, cette opération est coûteuse en temps et en mémoire.

L'algorithmique ne se préoccupe pas de cet aspect technique. Par contre il convient de tenir compte des caractéristiques du langage de programmation pour réaliser le passage des paramètres le plus adéquat.

L'autre point particulier est qu'il serait utile de pouvoir appeler le même module avec des tableaux de « tailles » différentes. Pour permettre cela, il convient de passer également la taille logique `n` en paramètre.



### Définitions Python

Soient les définitions suivantes :

```
TMAX = ...
```



### Tableau et paramètres

Le prototype des modules sera :

```
... ssprg(tab,n,...)
```

### Explication

La transmission des tableaux se fait **toujours** par référence d'où la possibilité de les modifier.

## 2.3 Tableau et fonction



### Tableau et fonction

Une fonction peut fournir un résultat qui soit un tableau.

## 2.4 Exemple : Saisie et affichage d'un tableau

La saisie et l'affichage sont deux opérations de base d'un tableau. Nous allons donc écrire deux procédures utilitaires `saisirTab` et `afficherTab`. Voici d'abord l'algorithme :



### Script Python

```
# Taille maximale des ITableaux
TMAX = 20

def PGSaisirAff():
    tab = [0 for x in range(TMAX)]
    nelems = UtilsTB.saisirTab(tab)
    UtilsTB.afficherTab(tab, nelems)

PGSaisirAff()
```

### Explication

L'algorithme déclare un tableau d'entiers `tab` de taille maximale `TMAX`. La procédure `saisirTab` a pour effet de demander le nombre d'éléments, de saisir les valeurs dans `tab` puis elle renvoie le nombre d'éléments saisis, ce dernier étant mémorisé dans l'entier `nelems`. La procédure `afficherTab` affiche les `nelems` éléments de `tab`.

### Exemple d'exécution

```
Nombre d'éléments dans [1..20]? 10
t[0]? 45
t[1]? 4
t[2]? 1
t[3]? -56
t[4]? 22
1t[5]? 34
t[6]? 49
t[7]? 12
t[8]? 0
t[9]? -27
[45 4 1 -56 22 34 49 12 0 -27 ]
```



### Fonction saisirTab

```
def saisirTab(t):
    TMAX = len(t)
    n = -1
    while not (1 <= n and n <= TMAX):
        print("Nombre de valeurs dans [1..", TMAX, "]? ", sep="", end="")
        n = int(input())
    for j in range(0, n):
        print("t[", j, "]? ", sep="", end="")
        t[j] = int(input())
    return n
```

### Explication

La fonction `saisirTab` effectue une saisie sécurisée du nombre d'éléments dans `n` (entier) compris dans `[1..TMAX]` puis saisit un à un `n` entiers et les stocke dans un `ITableau t` et renvoie `n`.



### Procédure afficherTab

```
def afficherTab(t, n):  
    print("[", end="")  
    for j in range(0, n):  
        print(t[j], " ", sep="", end="")  
    print(""])
```

### Explication

La procédure `afficherTab` affiche à la queue-leu-leu (séparés par un espace) les `n` éléments d'un `ITableau t`.

## 3 Parcours d'un tableau

Les tableaux interviennent dans de nombreux problèmes : il est primordial de savoir les parcourir en utilisant des algorithmes **corrects, efficaces et lisibles**.

Cette section examine les situations courantes et quelles solutions conviennent. On peut aussi envisager d'autres parcours (à l'envers, une case sur deux, ...) mais ils ne représentent aucune difficulté nouvelle.



### Déclarations

```
TMAX = ...  
tab = [0 for x in range(TMAX)]
```

### 3.1 Parcours complet

Pour parcourir **complètement** un tableau, la répétitive **Pour** est le moyen le plus simple comme dans l'algorithme suivant où « **traiter** » va dépendre du problème concret posé : afficher, modifier, sommer...



### Parcours complet

```
for j in range(0, n):  
    traiter tab[j]
```

## 3.2 Parcours partiel

Certains algorithmes se contentent de parcourir successivement les différents éléments du tableau jusqu'à rencontrer un élément satisfaisant une certaine condition. Par exemple :

- On cherche la présence d'un élément et on vient de le trouver.
- On vérifie qu'il n'y a pas de 0 et on vient d'en trouver un.

### Parcours complet, boucle TantQue

Un tel **parcours partiel** est le plus souvent basé sur une répétitive conditionnelle. La première étape est donc de transformer le **Pour** en **TantQue** ce qui donne :

```
j = 0
while j < n:
    traiter tab[j]
    j += 1
```



### Remarque

On peut à présent introduire le test d'arrêt. A la fin de la boucle, une contrainte est qu'on voudra savoir si oui ou non on s'est arrêté prématurément et, si c'est le cas, à quel indice. Il existe essentiellement deux solutions, avec ou sans variable booléenne. En général, la solution [A] sera plus claire si le test est court.



### (A) Parcours partiel, Sans variable booléenne

```
j = 0
while j < n and test_sur_tab[j]_dit_que_on_continue:
    traiter tab[j]
    j += 1
# Arrêt prématuré à l'indice j
arretPremature = (j < n)
```



### Ne testez pas tab(j.) à l'extérieur de la boucle

Car **j** n'est peut-être pas valide.



### (B) Parcours partiel, Avec variable booléenne

```
trouve = False
j = 0
while j < n and not trouve:
    if (test_sur_tab[j]_dit_que_on_a_trouvé):
        trouve = True
    else:
        j += 1
```

**Attention**

Choisissez un nom de booléen adapté au problème et initialisez-le à la bonne valeur. Par exemple, si la variable s'appelle « `bcontinue` » :

- Initialisez la variable à `Vrai`.
- Le test de la boucle est « `...Et bcontinue` ».
- Mettez la variable à `Faux` pour sortir de la boucle.

### 3.3 Parcours imbriqué

Certains algorithmes sur les tableaux font appel à des **boucles imbriquées**. La boucle principale sert généralement à parcourir les cases une à une, tandis que le traitement de chaque case dépend du parcours simple d'une partie du tableau (par exemple toutes les cases restantes) ce qui correspond à la boucle interne.



#### Parcours imbriqué

```
for j in range(0, n):  
    for k in range(..., ...):  
        traiter tab[k]
```