

ALGORITHMIQUE

↔ Les tableaux ↔

I. Deux exemples pour se faire une idée

I.1. Un exemple à une dimension

Exemple

La liste des sept jours de la semaine constitue un `tableau` à une dimension, ou `liste`, contenant 7 éléments de type chaîne de caractères.

La déclaration de ce tableau est donnée ci-dessous.

Algorithme

En `pseudo-code`, on note :

```
Variables :  
  Nom_Jour = Tableau de chaînes de caractères à 1 dimension [0...6]  
  ...
```

Pseudo-Code

Remarque

Le troisième jour de la semaine (mercredi) est stocké dans `Nom_Jour[2]`.

I.2. Un exemple à deux dimensions

Exemple

Un relevé de notes de 5 devoirs réalisés dans une classe de 31 étudiants est un tableau à deux dimensions :

- l'indice de ligne correspondant à l'étudiant concerné ;
- l'indice de colonne correspondant au devoir réalisé.

La déclaration de ce tableau (également appelé **matrice**) est donnée ci-dessous.

Algorithme

En `pseudo-code`, on note :

```
Variables :  
  Notes = Tableau d'entiers à 2 dimensions [0...30] [0...4]  
  ...
```

Pseudo-Code

Remarque

La note du 6^e étudiant (par ordre alphabétique) au 3^e devoir est stockée dans `Notes[5][2]`.

Attention

On donne toujours, dans l'ordre : d'abord l'indice de ligne puis l'indice de colonne.

II. La théorie

II.1. Des variables à tiroir

Définitions

Un **tableau** est comparable à un ensemble de boîtes dans lesquelles on va stocker des informations. Il est repéré par un **nom** qui obéit aux mêmes règles que les noms de variables (**Nom_Jour** dans l'exemple 1 et **Notes** dans l'exemple 2).

Il possède aussi un **type** qui définit le type des données qui y vont être stockées (chaîne de caractères dans l'exemple 1 et entier dans l'exemple 2).

Un tableau est aussi caractérisé par une **dimension** :

- un tableau à une dimension est comparable à une suite de cases, chacune repérée par un indice;
- un tableau à deux dimensions est comparable à un meuble à tiroirs, chacun repéré par un indice de rangée (ou ligne) et un indice de colonne.

Chaque « case » ou « tiroir » s'appelle une cellule et les **indices commencent à 0**. Pour repérer une cellule précise dans un tableau, on place son indice entre crochets derrière le nom du tableau.

Remarque

Un tableau à 2 dimensions est en fait un tableau à 1 dimension (les lignes) dont les cellules sont des tableaux à 1 dimension (les colonnes) !

Illustration

Dans l'exemple 2, on peut considérer le relevé de notes comme un tableau à 1 dimension et 31 cellules (les étudiants), chaque cellule étant un tableau à 1 dimension et 5 cellules (les notes). **Notes** est un tableau à deux dimensions et **Notes[3]** est un tableau à 1 dimension et 5 cellules : les notes du quatrième étudiant.

II.2. Taille d'un tableau

Définition

La **taille** d'un tableau est son nombre de cellules quand il s'agit d'un tableau à 1 dimension, et son nombre de lignes et de colonnes pour un tableau à 2 dimensions. La taille sera précisée si elle est prévisible et figée.

III. Tableaux à une dimension

III.1. Affectation directe

Méthode 1 - Par déclaration

On peut définir un tableau de manière directe en **déclarant** toutes ses cellules.

Algorithme et Python

En **pseudo-code**, on note :

```
Nom_Jour ← ["lundi", "mardi", "mercredi", ..., "samedi", "dimanche"]
```

Pseudo-Code

En **python**, cela peut donner :

```
1 Nom_Jour = ["lundi", "mardi", "mercredi", ..., "samedi", "dimanche"]
```

Code Python

Méthode 2 - Par une boucle

On peut utiliser une boucle `Pour` afin de stocker un à un les éléments du tableau.

Algorithme et Python

En `pseudo-code`, on peut proposer :

```
Pour i allant de 0 à 6 Faire
  Afficher("Saisir le contenu de la case numéro ",i)
  Saisir(Nom_Jour[i])
FinPour
```

Pseudo-Code

En `python`, cela peut donner :

```
1 Nom_Jour = [" " for i in range(0,7)] #crée un tableau vide à 7 cases
2 for i in range(0,7): #i va de 0 à 6
3     print(f"Case n°{i}")
4     Nom_Jour[i] = input("Entrer le contenu cette case :")
```

Code Python

III.2. Extension d'un tableau

Propriété

La taille d'un tableau n'est pas toujours prévisible dès le début d'un algorithme. On peut dans ce cas déclarer un tableau qui sera initialisé vide `[]` c'est à dire avec 0 cellule, et au gré des besoins, on ajoute une nouvelle cellule au tableau en y stockant une valeur. La taille du tableau évolue en conséquence.

On conviendra que la procédure `Étendre(NomTableau,Valeur)` ajoute une cellule au tableau `NomTableau` avec `Valeur` comme contenu.

Python

En `python`, on peut procéder comme suit :

```
>>> Machin = [2,5,4]
>>> len(Machin)
3
>>> Machin.append(8)
>>> len(Machin)
4
>>> Machin
[2, 5, 4, 8]

>>> Machin = [2,5,4]
>>> Machin = Machin + [8]
>>> Machin
[2, 5, 4, 8]
```

Début de la console `python`


Fin de la console `python`

III.3. Affichage des éléments

Méthode

On a vu que pour afficher **un** élément d'une liste (à n éléments) on utilisait son indice, via la commande `List[k]`. Si on veut afficher **tous** les éléments un par un, on peut utiliser une boucle `Pour`.

Algorithmme et Python

En , on peut proposer :

```
List ← [...]
n ← longueur(List)
Pour i allant de 0 à n-1 Faire
    Afficher(List[i])
FinPour
```

 Pseudo-Code


En , cela peut donner :

```
1 List = [.....]
2 n = len(List) #ou toute autre liste déjà définie !
3 for i in range(n): #qui va bien de 0 à n-1 !
4     print(List[i])
```



 Code Python

III.4. Recherche d'un élément

Méthode

Pour rechercher un élément donné (désigné par ) dans le tableau  (qui doit exister!), on parcourt un à un tous les éléments du tableau et on les compare à l'élément cherché.


Algorithmme

On utilise une boucle  et on affiche chaque position de  dans le tableau.

```
List ← [...]
n ← longueur(List)
Saisir(val)
trouve ← 0
Pour i allant de 0 à n-1 Faire
    Si List[i] = val Alors
        Afficher(val, "se trouve en position", i)
        trouve ← 1
    FinSi
FinPour
Si trouve = 0 Alors
    Afficher(val, "n'est pas dans la liste")
FinSi
```

 Pseudo-Code

Python


En , on obtient :

```
1 .
2 .
3 .
4 .
5 .
6 .
7 .
8 .
9 .
10 .
```


 Code Python


III.5. Nombre d'occurrences (apparitions) d'une valeur saisie par l'utilisateur

i Idée

On va utiliser une variable de comptage et on va  la liste élément par élément.


Algorithme

On utilise une boucle  Pour :

 Pseudo-Code

```
# Saisie du tableau List à n éléments et stockage de la taille
List ← [...]
n ← longueur(List)
# Saisie de la valeur à compter
Afficher("Saisir la valeur à compter")
Saisir(val)
# Initialisation du compteur
nb ← 0
# Boucle de comptage
Pour i allant de 0 à n-1 Faire
    Si List[i] = val Alors
        nb ← nb + 1
    FinSi
FinPour
# Affichage du résultat
Afficher(val, "apparaît", nb, "fois")
```

Python



En , on obtient :

 Code Python


```
1 # Saisie (manuelle) du tableau contenant n termes et stockage de la taille
2 List = []
3 n = int(input("Taille de la liste : "))
4 for i in range(0,n):
5     print(f"Case n°{i}")
6     element = int(input("Saisir la valeur :"))
7     List = List + [element]
8 # Saisie de la valeur à compter
9 val = int(input("Entrer la valeur à compter :"))
10 # Initialisation du compteur
11 nb = 0
12 # Boucle de comptage
13 for i in range(0,n):
14     if List[i] == val :
15         nb = nb + 1
16 print(f"{val} apparaît {nb} fois")
```

III.6. Recherche du plus grand (petit) élément

i Idée


On utilise une variable , initialisée à . Puis on parcourt les autres éléments de la liste en les comparant à max et en remplaçant max par toute valeur qui lui est supérieure. A la fin, max contient le plus grand nombre de la liste.

Algorithmme

On utilise une boucle  Pour :

```
# Tableau et taille
List ← [...]
n ← longueur(List)
# Initialisation de la variable max
maxi ← List[0]
# Boucle de parcours de la liste
Pour i allant de 1 à n-1 Faire
    Si List[i] > maxi Alors
        maxi ← List[i]
    FinSi
FinPour
Afficher(max)
```

 Pseudo-Code

En , on obtient :

```
1 .
2 .
3 .
4 .
5 .
6 .
7 .
```

 Code Python


Exercice

Écrire un algorithme, en pseudo-code et en , permettant de déterminer le plus petit nombre d'une liste.


```
.
.
.
.
.
.
.
.
.
.
```

 Pseudo-Code

```
1 .
2 .
3 .
4 .
5 .
6 .
7 .
```


 Code Python


Python

En , les commandes `max(List)` et `min(List)` permettent de renvoyer les valeurs maximale et minimal du tableau à *une dimension* `List`.


III.7. Commandes spécifiques à Python

Python

En , on a des commandes directes permettant de déterminer des éléments particuliers d'une liste.

Début de la console 

```
>>> liste = [3,10,15,10,2,4,0,5,7,11,13,10,15,16,15]
>>> len(liste)
15
>>> liste.count(15)      #compte le nombre de fois où 15 apparaît
3
>>> liste.index(2)       #indice de la 1ère apparition de 2
4
>>> liste.sort()         #tri croissant
>>> liste
[0, 2, 3, 4, 5, 7, 10, 10, 10, 11, 13, 15, 15, 15, 16]
```

Fin de la console 


IV. Tableaux à deux dimensions

IV.1. Affectation directe

i Idée


Les méthodes d'affectation sont les mêmes que pour un tableau à une dimension.

Algorithmme et Python

En , on note :

```
Tabl ← [[3,1,6,0],[5,2,4,2]]
```


 Pseudo-Code

En , cela donne :

```
1 Tabl = [[3,1,6,0],[5,2,4,2]]
```


 Code Python

Algorithmme et Python

En , on a :

```
Pour i allant de 0 à 1 Faire
  Pour j allant de 0 à 3 Faire
    Afficher("Contenu de la case L",i,"et C",j)
    Saisir(Tabl[i][j])
  FinPour
FinPour
```

 Pseudo-Code

En , cela donne :

```
1 tabl = [[0,0,0,0],[0,0,0,0]]
2 # ou tabl = [[0 for j in range(0,4)] for i in range(0,2)]
3 for i in range(0,2):
4     for j in range(0,4):
5         print(f"Case L{i} et C{j}")
6         tabl[i][j] = input("Entrer le contenu cette case :")
```

 Code Python

IV.2. Affichage des éléments

Méthode

Pour afficher :

- un élément d'un tableau, on utilise `Tabl[i][j]`;
- une ligne d'un tableau, on utilise `Tabl[i]`;
- une colonne d'un tableau, on utilise une boucle `Pour`.

Algorithme et Python

En `pseudo-code`, on a :

```
# Saisie de Tabl
Tabl ← [.....]

# Élément i,j
Afficher(Tabl[i][j])

# Ligne numéro k
Afficher(Tabl[k])

# Colonne numéro c
Pour i allant de 0 à n-1 Faire
    Afficher(Tabl[i][c])
FinPour
```

Pseudo-Code

En `python`, cela donne :

```
1 # Saisie de Tabl
2 Tabl = [.....]
3
4 # Élément i,j
5 print(Tabl[i][j])
6
7 # Ligne n°k
8 print(Tabl[k])
9
10 # Colonne n°c
11 for i in range(0,n) :
12     print(Tabl[i][c])
```

Code Python

IV.3. Création d'un tableau vide ou aléatoire en python

Idée

Le module `random` permet de générer des nombres aléatoires, on peut donc « créer » des tableaux aléatoires.

Remarque


Cette méthode peut être très pratique pour générer des tableaux et ainsi tester des `scripts` sur des tableaux aléatoires, sans avoir besoin de les créer « à la main »!

Il est donc vivement conseillé de connaître cette technique pour éviter de s'embêter à créer des tableaux, parfois « grands »!

Python

On peut, en , proposer :

```

Début de la console 
>>> # Tableau de 0 avec 3 lignes et 4 colonnes
>>> # On commence par l'indice des colonnes puis l'indice des lignes
>>> tableau1 = [[0 for j in range(0,4)] for i in range(0,3)]
>>> tableau1
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

>>> # Tableau de nombres aléatoires entre 1 et 20 à 2 lignes et 5 colonnes
>>> from random import *
>>> tableau2 = [[randint(1,20) for j in range(0,5)] for i in range(0,2)]
>>> tableau2
[[20, 1, 2, 3, 16], [1, 19, 7, 1, 16]]
>>> tableau3 = [[randint(1,20) for j in range(0,5)] for i in range(0,2)]
>>> tableau3
[[12, 14, 13, 18, 15], [2, 20, 9, 9, 16]]

>>> # On récupère les dimensions d'un tableau
>>> # Le nombre de lignes est la longueur du tableau
>>> # Le nombre de colonnes est la longueur d'une ligne du tableau
>>> len(tableau2)
2
>>> len(tableau2[0])
5

>>> tableau4 = [[randint(1,20) for j in range(0,6)] for i in range(0,3)]
>>> tableau4
[[13, 16, 1, 12, 7, 17], [3, 9, 8, 9, 1, 3], [12, 12, 8, 20, 8, 6]]
>>> # Élément 1,4
>>> print(tableau4[1][4])
1

>>> # Affichage sous forme de tableau
>>> for i in range(0,3):
...     print(tableau4[i])
...
[13, 16, 1, 12, 7, 17]
[3, 9, 8, 9, 1, 3]
[12, 12, 8, 20, 8, 6]
>>> # Ligne n°2
>>> print(tableau4[2])
[12, 12, 8, 20, 8, 6]
>>> # Colonne n°5
>>> for i in range(0,3) :
...     print(tableau4[i][5])
...
17
3
6

```

Fin de la console 


IV.4. Recherche d'un élément


Méthode

Comme dans le cas d'un tableau à une dimension, on parcourt un à un tous les éléments du tableau tant que l'élément n'a pas été trouvé et tant qu'il reste des éléments à examiner.

Le parcours se fait de la façon suivante : l'indice de ligne étant fixé, on fait varier l'indice de colonne. Il faut donc deux boucles : une boucle sur l'indice de ligne (i) dans laquelle est imbriquée une boucle sur l'indice de colonne (j).

Algorithmme et Python

En  pseudo-code, on peut proposer :

 Pseudo-Code

```


# Saisie de Tabl et stockage des dimensions
Tabl ← [.....]
n ← longueur(Tabl)
p ← longueur(Tabl[0])

# Saisie de la valeur à rechercher
Afficher("saisir la valeur à chercher :")
Saisir(val)

# Initialisation d'un booléen qui indiquera que l'élément n'a pas été trouvé
trouve ← 0

# Boucles de parcours du tableau
Pour i allant de 0 à n-1 Faire
  Pour j allant de 0 à p-1 Faire
    Si Tabl[i][j] = val Alors
      Afficher(val, "a été trouvé à la", i, "ième ligne et", j, "ème colonne")
      trouve ← 1
    FinSi
  FinPour
FinPour

# Toutes les positions de val sont affichées
# Message en cas d'échec de la recherche
Si trouve = 0 Alors
  Afficher(val, "ne figure pas dans le tableau")
FinSi
  
```



En  python, cela donne :

 Code Python

```

1 .
2 .
3 .
4 .
5 .
6 .
7 .
8 .
9 .
10 .
11 .
12 .
  
```

Remarques

Contrairement aux tableaux à une dimension pour lesquels on a des commandes  prédéfinies, on ne peut pas déterminer (directement) le maximum, le minimum ou le nombre d'occurrences d'un élément... Il faut passer par un balayage des lignes avec une boucle  Pour...