Les tableaux de données

Table des matières

I. Qu'est-ce qu'un tableau ?

Contexte

Les programmes tirent profit de la vitesse de calcul de la machine et les types de données nous permettent de préciser la nature de l'information à l'ordinateur. Pour aller encore plus loin, nous aimerions pouvoir manipuler plus facilement de nombreuses données.

Les tableaux vont nous permettre de regrouper dans la mémoire des informations qui peuvent être traitées par lot. L'ordinateur pourra donc lancer les calculs sur une série de données sans avoir besoin d'intervention extérieure. Les tableaux constituent un atout majeur lorsqu'il s'agit de faire réaliser énormément d'opérations à la machine en écrivant le moins d'instructions possible.

A. Manipuler les données par lot

Maintenant que nous connaissons les types simples, nous aimerions aller plus loin dans la gestion de nos données. Si nous voulons gérer un grand volume d'information, nous nous retrouvons limités. Nous voulons par exemple manipuler des données météorologiques relevées sur un mois, soit entre 28 et 31 valeurs. Nous pourrions définir 31 valeurs de type décimal qui serviraient à sauvegarder tous les relevés de température pris sur un mois :

Jour	Température
1	21
2	22
3	20
4	20
5	21
6	21
7	19
8	18
9	17
10	17

Jour	Température
11	17
12	19
13	20
14	21
15	22
16	22
17	23
18	24
19	23
20	22

Jour	Température
21	19
22	22
23	21
24	20
25	21
26	20
27	21
28	20
29	19
30	18

D'un point de vue algorithmique, nous aurions donc une déclaration de variables assez conséquente :

Méthode

```
Variable :
 Jour1, jour2, … jour30 : réel


Début
…
Fin
```

Chaque opération statistique comme le calcul de la moyenne par exemple, nous obligerait à faire appel à chacune de ces 30 variables :

Méthode

```
Variable :
 Jour1, jour2, … jour30 : réel
 moyenne : réel

Début
…
Moyenne = (jour1 + jour2 + … + jour30) / 30
Fin
```

Remarque

C'est une solution qui marche mais qui n'est pas viable pour le développeur. Si nous pouvons le concevoir pour un mois, cela devient impossible à gérer pour une année ou pour une période encore plus longue. Le fait de déclarer autant de variables nuit également à la compréhension du programme ou de l'algorithme.

Ce petit exemple nous montre deux problèmes auxquelles nous nous heurtons.

- Les différentes variables « *jour* » véhiculent toutes des informations sur la température, nous aimerions donc grouper ces valeurs.
- Le fait qu'il n'y ait pas toujours 31 jours par mois fait que nous allons déclarer des variables qui ne seront pas forcément toutes utilisées.

Nous avons déjà parlé de la rapidité de calcul de la machine et que le but du développeur est de faire faire à l'ordinateur un maximum de traitement à partir d'un minimum d'instructions. Avec cette solution, nous ne sommes clairement pas dans cette optique.

B. Plusieurs valeurs dans une variable

Sortons de l'algorithmie et de la programmation quelques instants. Si nous cherchions comment représenter des relevés de température pour en calculer des indicateurs (moyenne, valeur maximum, valeur minimum), il y a de fortes chances que nous pensions à les représenter dans un tableur.

Exemple

Un tableur est composé d'une grille de cellules et nous pouvons facilement y placer nos valeurs de température. Une fois les valeurs dans notre tableur, certaines fonctions vont nous permettre de calculer rapidement nos indicateurs :

Jour	Température
1	21
2	22
3	20
4	20
5	21
6	21
7	19
8	20
9	17
10	17
11	...

Moyenne	19,8	fonction moyenne()
Maximum	22	fonction max()
Minimum	17	fonction min()

Remarque

À partir du moment où toutes les valeurs nécessaires à nos calculs se retrouvent dans la même colonne, nous ne les considérons plus comme des valeurs isolées mais nous voulons traiter l'ensemble de la colonne. Si nous regardons les fonctions moyenne(cellule_début : cellule_fin), min(cellule_début : cellule_fin) et max(cellule_début : cellule_fin), nous voyons que le paramètre attendu est une plage de valeurs. La plage de valeurs est définie par la cellule de début et la cellule de fin et le tableur sait qu'il va devoir traiter toutes les valeurs entre ces deux cellules.

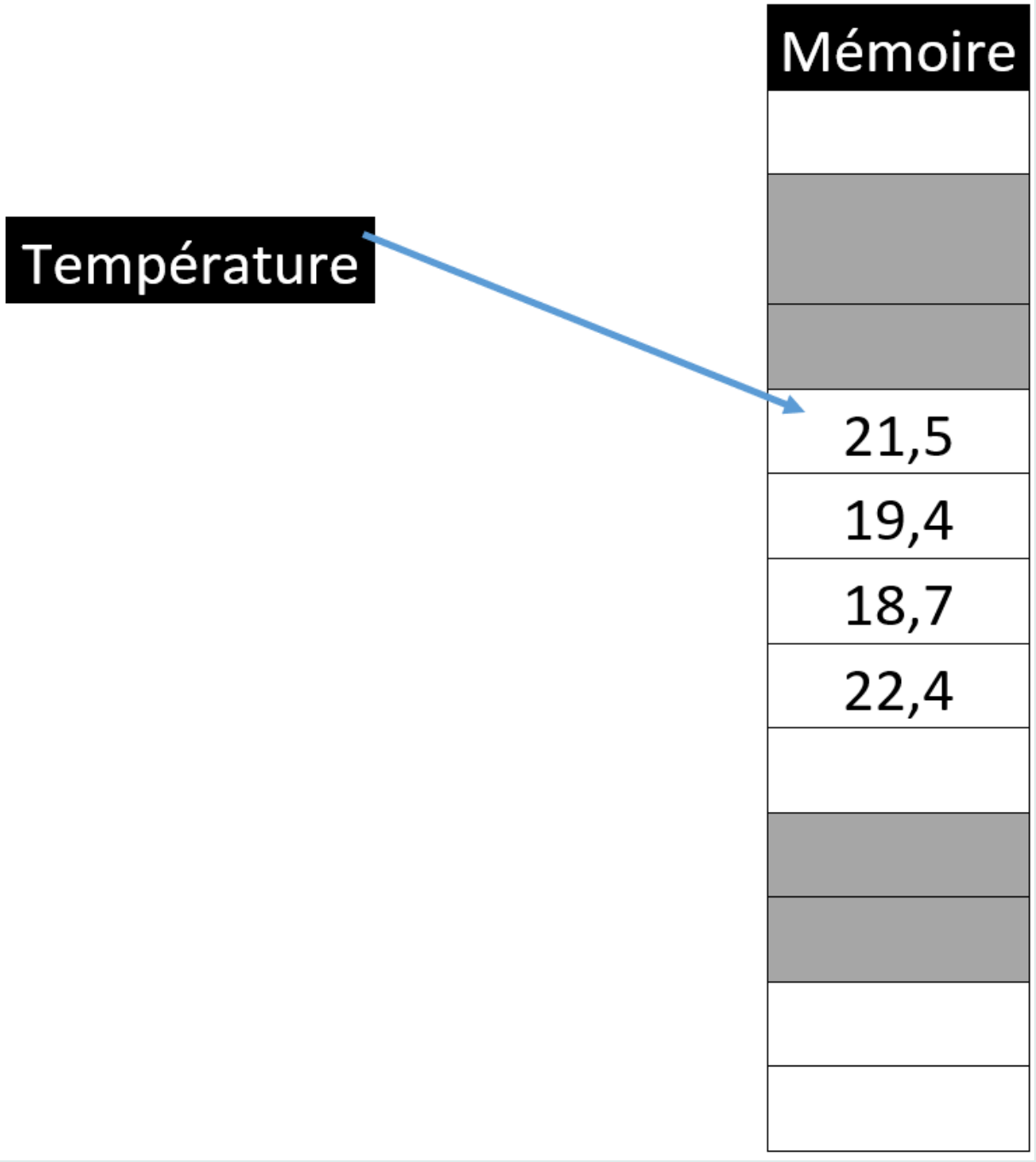
C'est ce genre de principe que nous aimerions retrouver dans nos langages de programmation. Nous voudrions pouvoir donner une liste de valeurs à la machine et la laisser la traiter. Cette liste, c'est ce que nous allons représenter avec un tableau.

C. Tableau dans la mémoire

Pour une variable de type simple (entier, booléen, réel), une zone mémoire est allouée et le nom de la variable pointe directement sur cette zone pour pouvoir récupérer la valeur stockée. Dans le cas d'un tableau, nous ne nous contentons pas d'une seule valeur mais d'une liste de valeurs. Ce n'est donc pas une seule case mémoire qui sera allouée cette fois mais autant de cases qu'il y a de valeurs dans notre tableau.

Fondamental

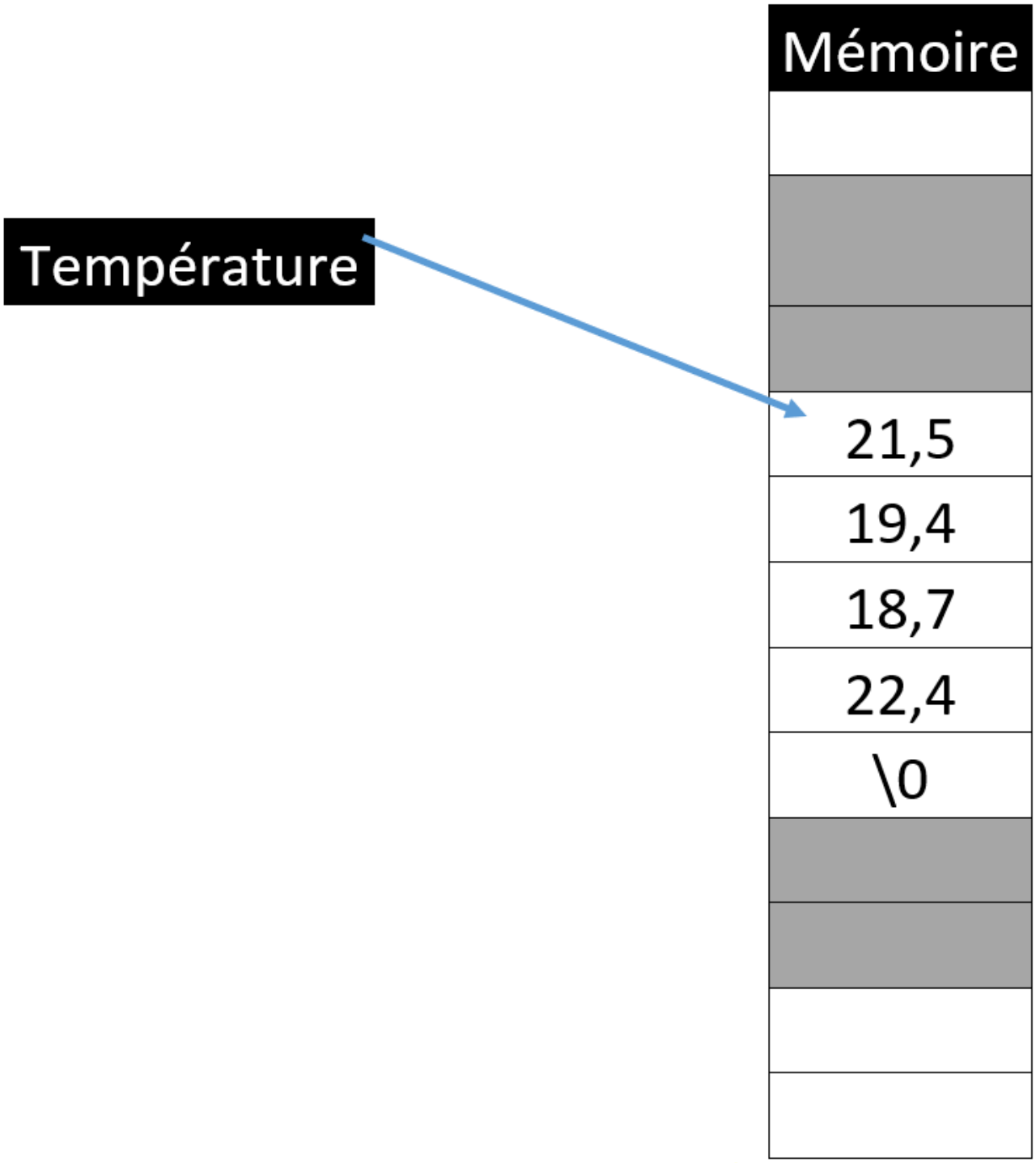
La représentation du tableau amène une contrainte supplémentaire. Les éléments du tableau sont stockés dans des zones contigües de la mémoire. Voici la représentation d'une variable de type tableau de réel contenant les éléments : 21.5, 19.4, 18.7, 22.4 :



La variable « *température* » pointe sur la première case mémoire du tableau. Les éléments du tableau se suivent physiquement dans la mémoire si bien qu'il est aisé de parcourir un tableau puisqu'il suffit de parcourir la mémoire. Parcourir un tableau est une opération qui consiste à commencer par le premier élément et à traiter les éléments successivement selon leur position dans le tableau. La difficulté est de savoir à quel moment le tableau s'arrête. Pour cela, les langages de programmation modernes incluent dans le type tableau une information sur la taille du tableau. Dans notre exemple, nous avons un tableau de 4 éléments, la taille est donc de 4. Si nous sommes sur le premier élément du tableau, sa fin se trouve 4 zones mémoires plus loin.

Remarque

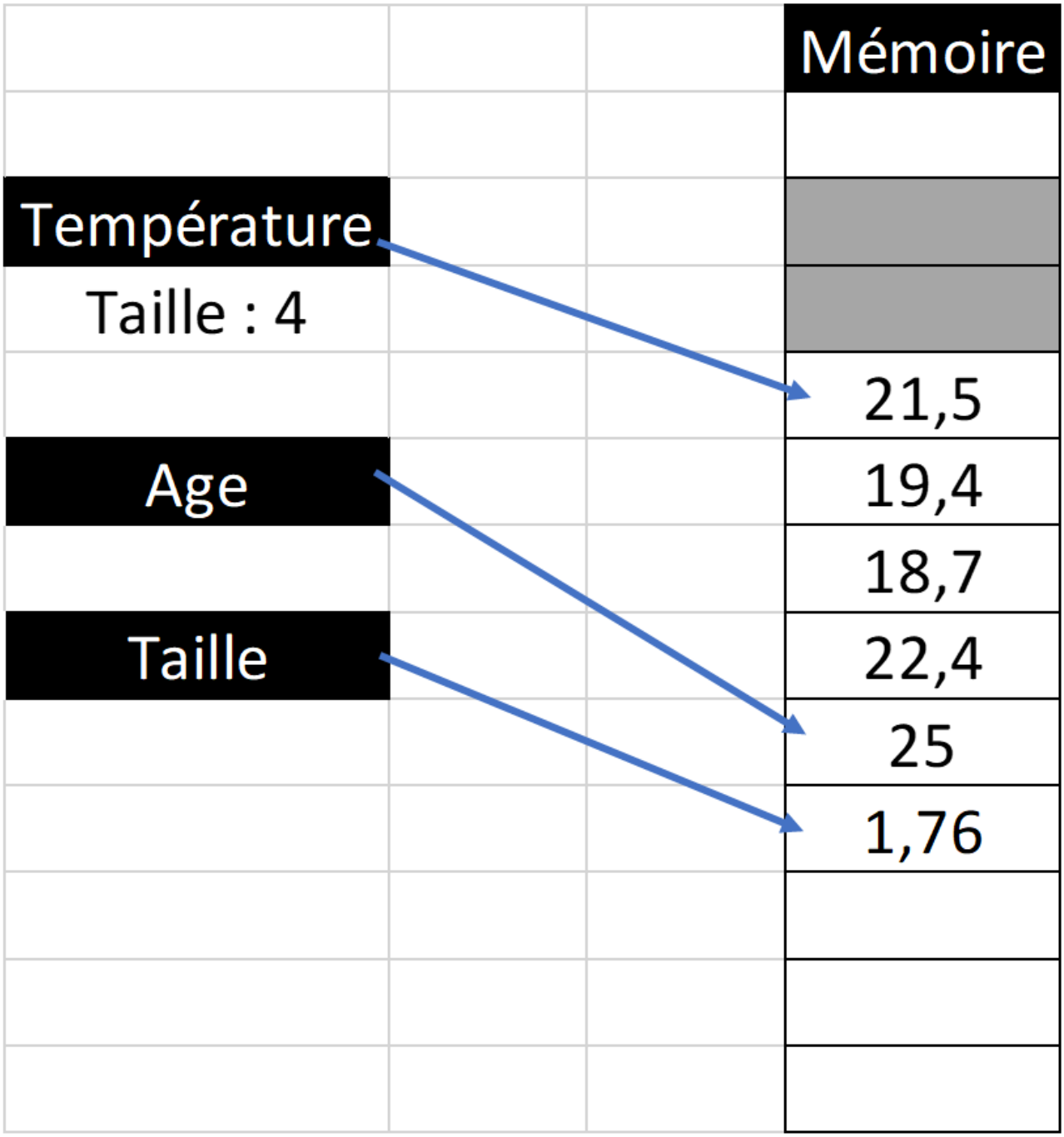
Certains langages, comme le C, s'appuient sur une autre règle pour connaître la fin d'un tableau. Au lieu de sauvegarder le nombre d'éléments, le langage C utilise un caractère spécial pour marquer la fin du tableau : « \0 ».



Il s'agit là d'un autre mécanisme mais qui fonctionne bien également. On pourra donc parcourir le tableau jusqu'à atteindre le fameux caractère « \0 » et être donc sûr d'être arrivé à la fin du tableau.

Fondamental

La notion de fin de tableau est très importante. Sans elle, nous pourrions continuer à parcourir la mémoire de l'ordinateur sans que les valeurs que nous prenons en compte aient encore un sens pour notre calcul :



Dans cet exemple, la taille du tableau de 4 nous indique que seules les 4 premières valeurs sont à prendre en compte dans le traitement du tableau. Sans cette information, nous pourrions très bien utiliser l'âge ou la taille et le résultat de notre calcul n'aurait plus aucun sens. Nous pourrions également nous retrouver dans un emplacement mémoire utilisé par un autre programme. Il s'agit dans ce cas-là d'un débordement de la mémoire attribuée. C'est une erreur qui peut être assez compliquée à corriger. Le mieux est donc de rester vigilant sur les dimensions des tableaux que nous utilisons.

D. Contenu d'un tableau

Jusqu'à présent, nous avons vu qu'un tableau peut contenir plusieurs éléments et que la taille du tableau est une information importante pour pouvoir le parcourir. Nous allons nous intéresser ici au contenu du tableau. Un tableau contient toujours un seul type d'éléments en algorithmie. Nous allons donc prendre l'habitude de déclarer un tableau d'entiers, un tableau de réels ou un tableau de booléens. Plusieurs langages de programmation ont étendu la notion de tableau pour pouvoir contenir des types différents. C'est le cas des listes en Python ou des arrays en Javascript.

Fondamental

Comme toutes les variables, les tableaux sont donc définis par un type qui correspond au type des éléments qui le composent. Encore une fois cette notion de type est importante et cela a un impact sur la représentation mémoire du tableau. Pour nous en convaincre, voici un petit descriptif des types en C++ et la taille qu'ils prennent en mémoire :

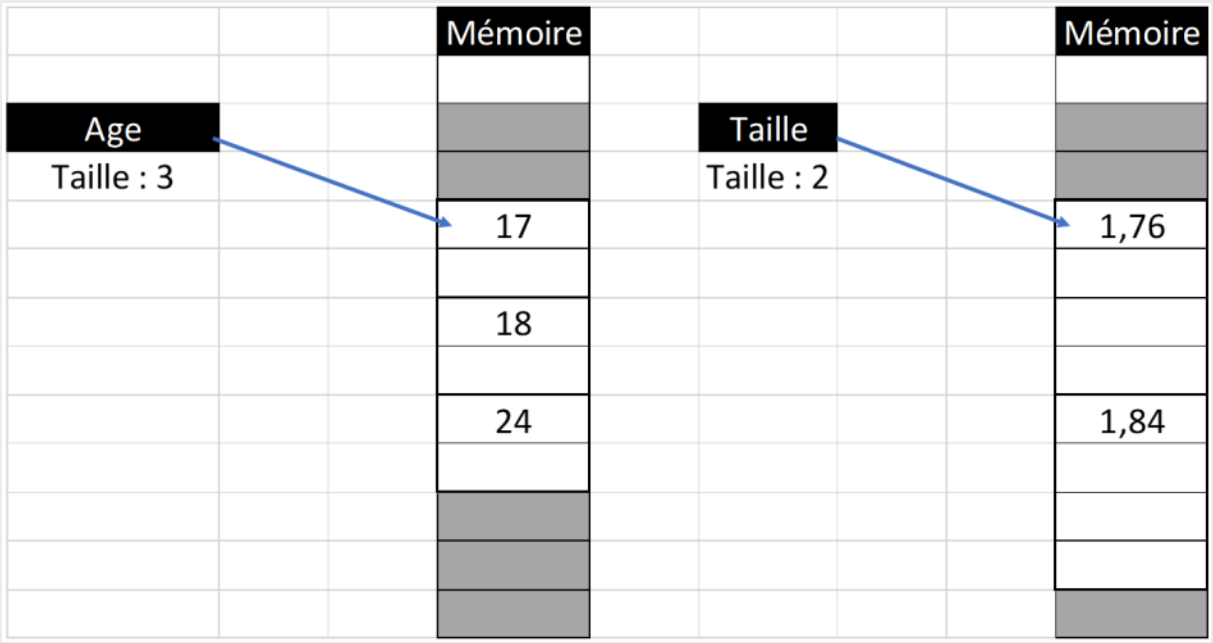
Type	Désignation	Taille en octet
int	Entier	2
Flot	Réel	4
double	Réel, double précision	8
char	caractère	1
bool	Boolean	1

Complément

Un octet est une unité de mesure de la quantité de données. Il correspond à 8 bits. Autrement dit, un octet est une valeur numérique sur 8 positions et chaque position peut valoir 0 ou 1. Il est donc possible de coder sur un octet $2^8 = 256$ valeurs différentes. Pour le cas d'un entier codé sur 2 octets, il est possible de représenter les nombres de 0 à **$(2^{16} - 1) = 65\,535$**. 2 est élevé à la puissance 16 car nous sommes sur deux octets (2 x 8 = 16).

Exemple

Une variable de type double utilise deux fois plus de mémoire de stockage qu'une variable de type float. Le fait de définir un type pour le tableau permet au langage de savoir que chaque élément de la mémoire va utiliser la même taille de zone mémoire :



Le tableau « *Age* » est un tableau d'entiers (int). Chaque élément utilise donc deux octets en mémoire. Le tableau « *Taille* » est un tableau de réels (float). Chaque élément utilise donc 4 octets en mémoire. La taille mémoire utilisée par chaque élément permet de déterminer la taille mémoire occupée par le tableau en entier :

- Tableau « *Age* » : 3 éléments de 2 octets → Tableau de 3 x 2 = 6 octets
- Tableau « *Taille* » : 2 éléments de 4 octets → Tableau de 2 x 4 = 8 octets

Exemple

Si vous avez déjà programmé en Python, vous avez dû vous apercevoir que le type « *list* » se rapproche beaucoup du tableau que nous venons de décrire. Dans une liste Python, nous pouvons cependant stocker des éléments de types différents. Le type liste de Python n'est pas à proprement dit un tableau mais nous pouvons le voir comme une extension du tableau. C'est également le cas des arrays en Javascript.

Remarque

Cette première partie nous a permis de comprendre la structure du tableau ainsi que sa représentation en mémoire. Nous allons voir dans la partie suivante comment créer un tableau et le manipuler.

Exercice : Quiz

[solution n°1 p.31]

Question 1

Un tableau en termes d'algorithmie est :

- ○ Une façon de stocker plusieurs valeurs dans une seule variable
- ○ Un tableau de bord qui vous donne des indicateurs sur votre programme
- ○ Une fenêtre dans laquelle vous tapez votre code

Question 2

Dans la mémoire, de quelle façon un programme stocke-t-il un tableau ?

- Chaque valeur est stockée là où il y a de la place, il n'est pas nécessaire que les zones mémoires soient contigües
- Les valeurs sont stockées dans des zones mémoires contigües et la variable tableau pointe sur le premier élément
- Les tableaux sont stockés dans une mémoire spéciale qui leur est dédiée

Question 3

Un même tableau peut contenir :

- Des données de types différents
- Des données de types différents si elles sont numériques (entier, réel, booléen)
- Un seul type de données

Question 4

Pourquoi est-il important de connaitre la taille du tableau ?

- Pour ne pas se retrouver à traiter un élément de la mémoire qui ne correspond pas au tableau
- Pour pouvoir le comparer aux autres tableaux
- Ce n'est important de connaitre sa taille, ce qui compte ce sont les valeurs

Question 5

Si je mets des données dans un tableur, quel élément de mon tableur représente le mieux un tableau ?

- Une cellule d'une feuille de travail
- Une colonne d'une feuille de travail
- Un classeur

III. Construire et manipuler un tableau

A. Construire un tableau

Fondamental

Si un tableau peut contenir plusieurs éléments, il reste toutefois une variable de notre programme. Comme toute variable, nous allons devoir le déclarer avant de pouvoir l'utiliser. Un tableau doit avoir un nom, un type et une taille. Le type d'un tableau indique la présence d'un tableau ainsi que le type des éléments qu'il va contenir.

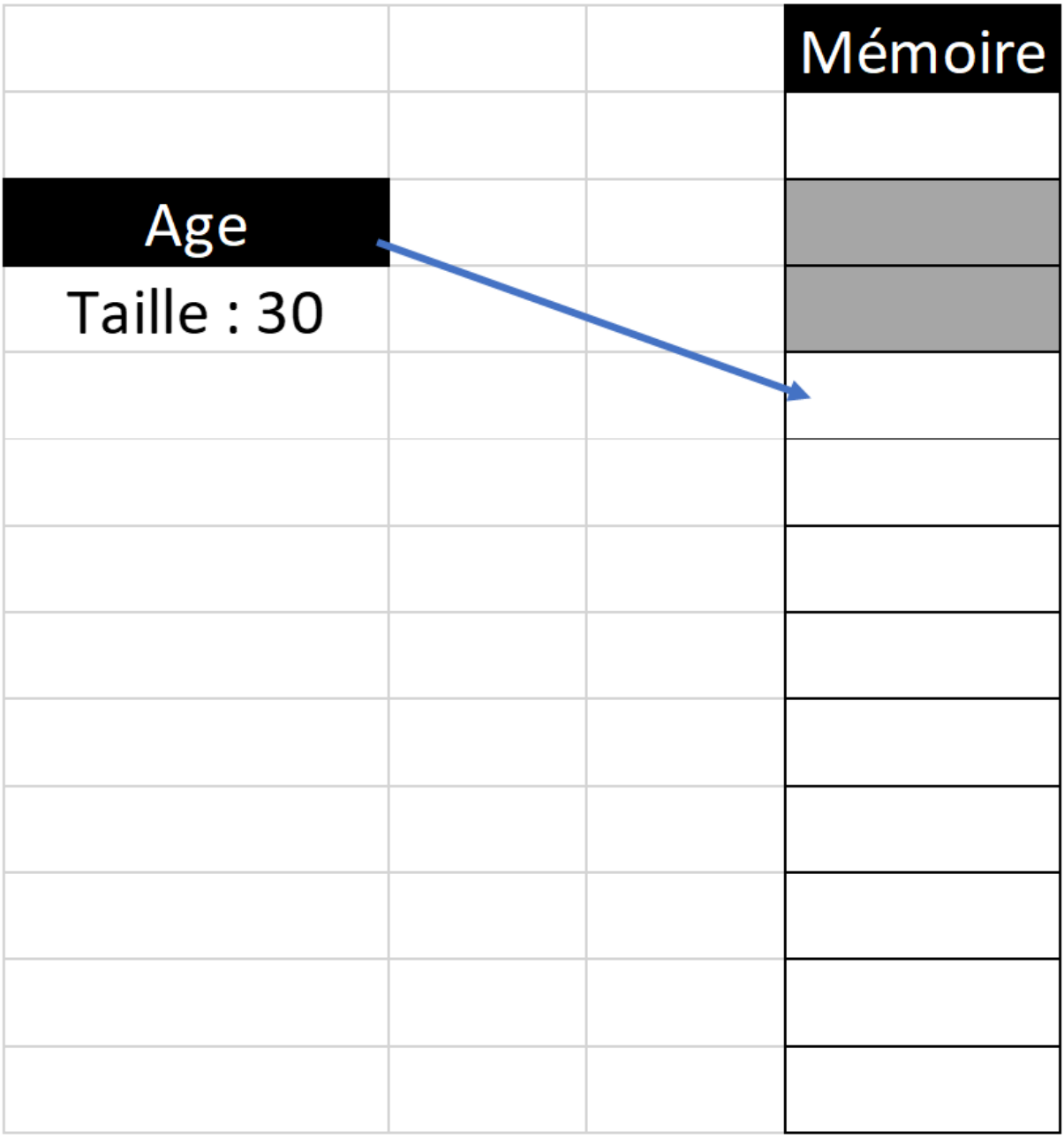
```
Variables :
 Age : entier[30]
 Taille : réel[30]

Début
…
```

```
Fin
```

La notation [ ] indique que nous sommes en présence d'un tableau. Le nombre entre [ ] permet d'indiquer la taille du tableau. « *Age* » est un tableau d'entiers comportant 30 éléments. « *Taille* » est un tableau de réels comportant 30 éléments. Dans ce cas-là, les tableaux sont considérés comme non initialisés : nous connaissons leur taille mais le contenu n'est pas encore défini.

Spécifier la taille dès le départ est important car le programme va chercher un emplacement mémoire susceptible de pouvoir stocker le tableau. Si je déclare un tableau de 30 éléments, il faut que je trouve en mémoire un emplacement de 30 zones mémoires contigües :



Même si les éléments du tableau ne sont pas encore renseignés, la zone mémoire dédiée au stockage est réservée. La question du dimensionnement du tableau est donc très importante. Elle doit pouvoir contenir tous les éléments manipulés sans pour autant saturer la mémoire.

Il est possible de définir un tableau à partir des éléments qui le composent :

```
Variables :
 Age : entier[ ] = [22, 18, 25, 33]
 Taille : réel[ ] = [1.76, 1.84, 1.81, 1.79]

Début
…
Fin
```

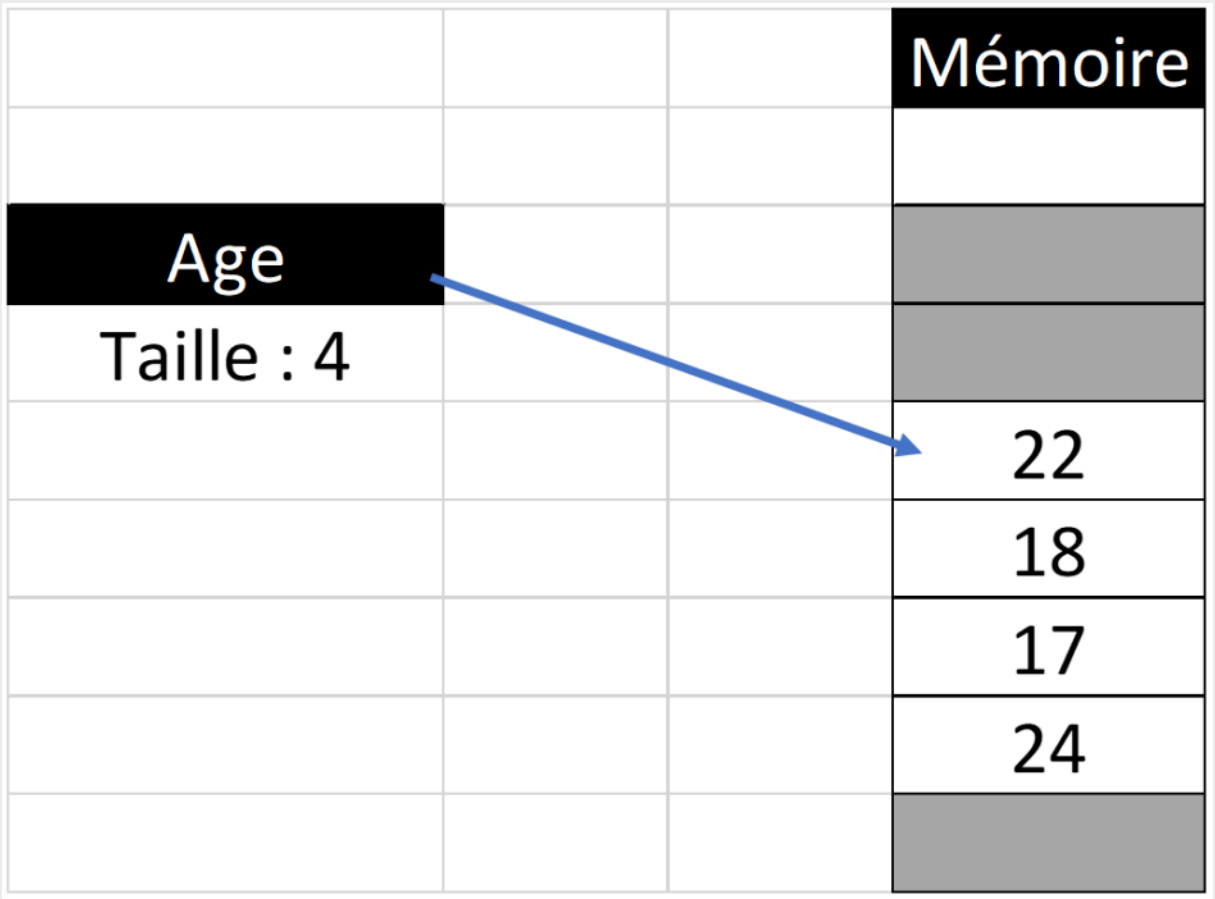
Remarque

Cette fois-ci, la taille n'est plus spécifiée, mais la définition du tableau nous indique les éléments constituant le tableau. « *Age* » est un tableau d'entiers contenant 4 éléments : 22, 18, 25 et 33. Une fois de plus, nous retrouvons tous les éléments nécessaires pour pouvoir déclarer un tableau.

Avec les listes Python, nous avons la possibilité de rajouter un élément à une liste existante. Il est intéressant de comprendre ce qui se passe en mémoire lorsque nous faisons un liste.append(element). En effet, il est possible de gagner énormément de temps de traitement sur cette instruction.

Exemple

Imaginons un tableau de 4 éléments :



Si je veux ajouter un 5ième élément à ce tableau, je m'aperçois que la zone mémoire après la valeur 24 n'est pas disponible. Je vais donc devoir trouver un nouvel emplacement mémoire qui contient 5 emplacements libres contigus et je vais recopier mon tableau à cet endroit. Ici, nous sommes sur 4 éléments mais si j'ajoute un élément à un tableau de plusieurs milliers d'éléments, il y a de fortes chances que ce tableau soit copié dans un autre emplacement mémoire.

Remarque

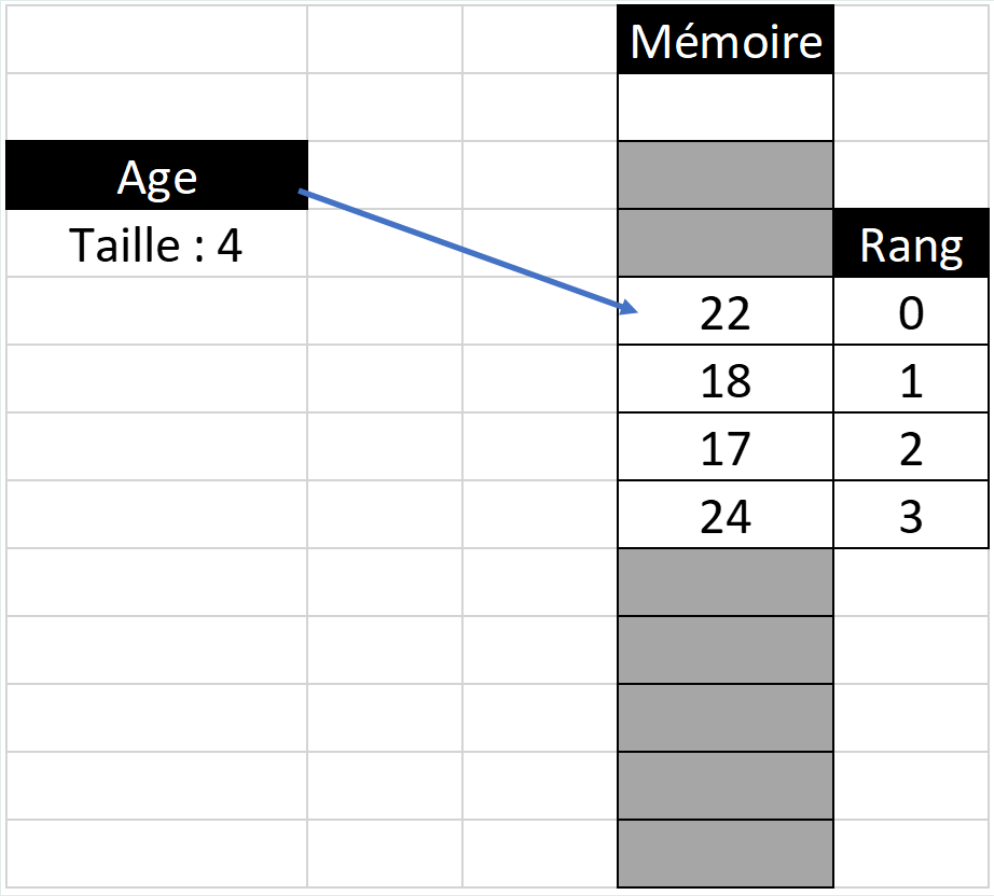
Copier des milliers d'éléments prend du temps. Si vous savez à l'avance combien d'éléments va contenir votre tableau, le mieux est de le déclarer directement à la bonne taille pour éviter les recopies inutiles. Il y a toujours des cas où on ne peut pas prévoir le nombre d'éléments à traiter et dans ces cas-là, nous n'avons pas d'autres choix que de sacrifier un peu de temps de traitement.

B. Accéder aux éléments d'un tableau

Fondamental

Un tableau est un élément de structure incontournable pour traiter un grand nombre de valeurs qui représentent la même notion. Cela ne veut pas dire pour autant qu'il devient impossible d'accéder aux éléments d'un tableau de manière individuelle. D'après la représentation mémoire que nous avons présentée, nous savons que la variable tableau pointe sur le premier élément du tableau en mémoire. Nous allons utiliser le fait que les éléments du tableau sont stockés dans des zones mémoire contigües pour pouvoir y accéder facilement.

Lors de la déclaration du tableau, nous avons parlé des crochets [ ]. Les crochets vont nous permettre d'accéder facilement à un élément particulier d'un tableau en fonction de sa position. À chaque élément du tableau correspond un indice de rang qui va nous permettre d'y accéder.



Remarque

Nous pouvons dire que « 22 » est l'élément du tableau au rang « 0 », « 18 » est l'élément du tableau au rang « 1 » et ainsi de suite. L'opérateur crochet va nous permettre d'accéder directement à un élément en connaissance son rang :

- Age[0] va nous renvoyer la valeur 22 (22 est la valeur dans le tableau « *age* » de l'élément au rang 0).
- Age[1] va nous renvoyer la valeur 18 (18 est la valeur dans le tableau « *age* » de l'élément au rang 1).
- Etc.

Remarque

Nous savons maintenant comment déclarer un tableau et comment accéder à ses éléments. Voici un petit exemple d'algorithme qui reprend ces notions.

```
Variables :
 Age : entier[ ] = [22, 18, 25, 33]
 Taille : réel[ ] = [1.76, 1.84, 1.81, 1.79]


Début
 Afficher 'La personne 1 est âgée de ' + age[0] + ' ans'
 Afficher 'La personne 2 est âgée de ' + age[1] + ' ans'
 Afficher 'La personne 3 est âgée de ' + age[2] + ' ans'
 Afficher 'La personne 4 est âgée de ' + age[3] + ' ans'
Fin
```

Attention

Pour quelqu'un qui débute en programmation, il peut être perturbant que les indices de tableau commencent à 0 et non à 1. Pour que ce soit plus clair, le rang dans le tableau correspond au nombre de décalages à faire à partir du début du tableau :

- Age[0] : la variable « *age* » de type tableau d'entiers pointe déjà sur le premier élément de tableau il y a donc 0 décalage à faire et on retourne juste la valeur pointée : 22.
- Age[1] : la variable « *age* » de type tableau d'entiers pointe sur le second élément de tableau il y a donc 1 décalage à faire pour se retrouver sur la zone mémoire suivante : 18.
- Et ainsi de suite.

Attention

S'il y a 4 éléments dans le tableau, ces 4 éléments seront désignés par des indices allant de 0 à 3. Si on veut généraliser cette règle à n'importe quel tableau, un tableau de n éléments verra ses éléments désignés par des indices allant de 0 à n-1.

C. Modifier un tableau

Fondamental

Un tableau permet donc de stocker des valeurs en mémoire en les regroupant sous le nom d'une seule et même variable tout en nous permettant d'accéder à chaque élément de manière individuelle. Cependant, les valeurs peuvent être amenées à changer et il est nécessaire de pouvoir modifier les valeurs d'un tableau sans avoir à en recréer un.

Les éléments d'un tableau étant accessibles directement à travers leur indice, il est tout à fait possible de modifier également un élément en y accédant directement :

```
Variables :
 Age : entier[ ] = [22, 18, 25, 33]


Début
 Age[1] ← 24
 Afficher 'La personne 2 est âgée de ' + age[1] + ' ans'
Fin
```

Remarque

Age[1] représente l'élément qui se trouve à l'indice 1 du tableau, à savoir la valeur 18 lors de l'initialisation. Le fait que age[1] se retrouve à gauche de l'opération d'affectation indique que sa valeur va être modifiée. Après modification, age[1] vaudra donc 24 et le tableau age sera composé de 4 éléments : [22, 24, 25, 33]. Pour que la modification puisse être faite, il faut bien entendu que la valeur de modification coïncide avec le type du tableau. Il n'est pas possible d'affecter une valeur réelle à un élément d'un tableau d'entiers.

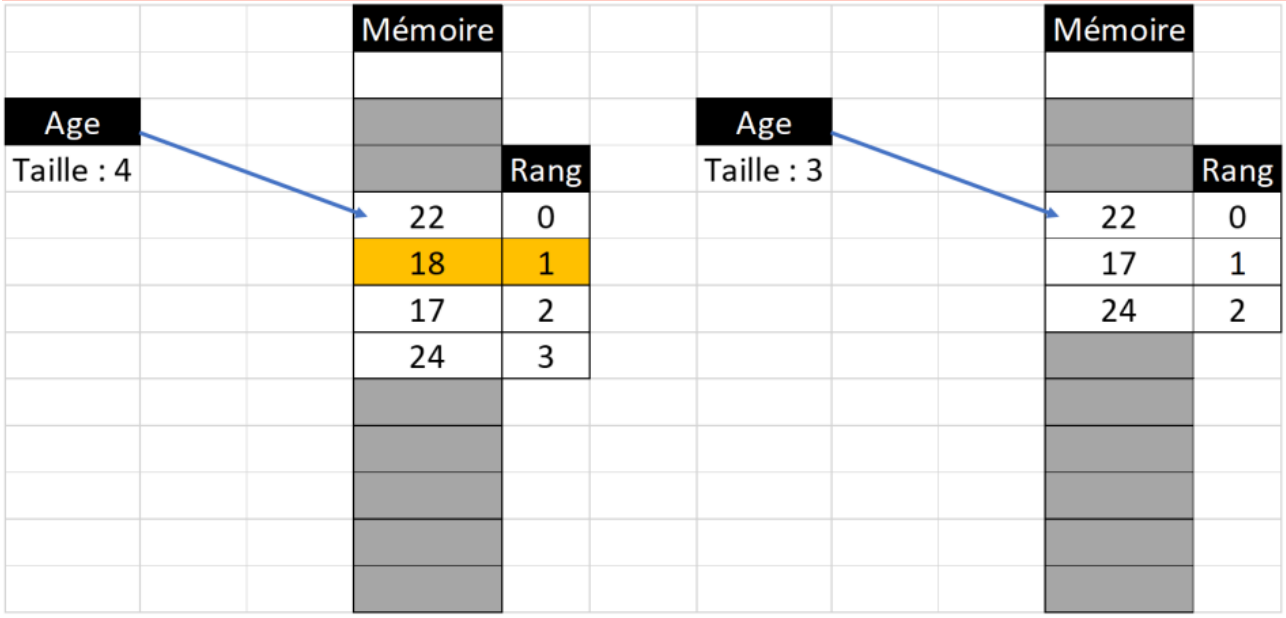
Attention

La plupart des langages de programmation permettent les opérations d'ajout suivantes :

- Ajout d'un élément à gauche (append left). Dans ce cas, un élément est ajouté avant le premier élément et la variable du tableau pointe désormais sur ce nouvel élément.
- Ajout d'un élément à droite (append ou append right). Dans ce cas, un élément est ajouté en fin de tableau.

Dans la plupart des cas, ces opérations entrainent une recopie du tableau vers une zone mémoire assez grande pour pouvoir le contenir. Si vous connaissez à l'avance la taille maximale de votre tableau, déclarez cette taille dès le début et effectuez des modifications en accédant à chaque élément par son indice. Votre programme sera plus rapide car il évitera de nombreuse recopies en mémoire.

La plupart des langages permettent également de supprimer un élément bien particulier du tableau. Dans ce cas tous les éléments qui suivent cet élément subiront une copie :



Sur cet exemple, deux copies ont été réalisées :

- 17 vers le rang 1
- 24 vers le rang 2

Suivant la taille du tableau, une suppression peut également prendre beaucoup de temps à cause des recopies successives.

La majorité des langages de programmation vont vous permettre de trier un tableau. Les algorithmes de tri pourraient représenter une partie de cours entier tant il y a dire sur le sujet. L'importance des tris en programmation vient du fait qu'il est plus facile de chercher dans un tableau trié. Pour s'en convaincre, il suffit d'imaginer un dictionnaire. Personne ne se servirait d'un dictionnaire si les mots à l'intérieur n'étaient pas triés. En programmation, c'est exactement la même chose.

Exemple

Nous voulons savoir si la valeur 24 se trouve dans le tableau âge. Si le tableau n'est pas trié, il faudra comparer chaque élément du tableau. Si le tableau est trié, nous savons que si nous atteignons une valeur > 24, ce n'est plus la peine de continuer, toutes les autres seront > 24 également. Cela permet d'éviter des recherches inutiles et donc de gagner du temps.

D. Parcours de tableau avec une boucle

Une fois toutes nos valeurs rassemblées au sein d'une même variable tableau, nous aimerions construire des algorithmes qui traitent des tableaux entiers et non plus élément par élément. Nous aimerions obtenir par exemple la moyenne des valeurs du tableau sans pour autant avoir à accéder à chaque élément par son indice. Pour cela, nous allons utiliser une structure algorithmique qui permet d'itérer sur une ou plusieurs instructions : la boucle.

Définition

Une boucle est une structure de contrôle qui permet de répéter plusieurs fois les mêmes instructions. Cela permet au développeur de ne pas avoir à se répéter. Plusieurs styles de boucles existent selon que l'on veut répéter les instructions un nombre de fois déterminé (par exemple 10 fois) ou alors jusqu'à ce qu'une condition soit vraie (avance jusqu'à tomber sur un obstacle). Le sujet des boucles sera détaillé dans une autre partie du cours, nous allons juste nous intéresser ici au parcours des tableaux.

Remarque

Voici un petit exemple de boucle qui affiche les nombres de 1 à 10.

```
Début
 Pour i de 1 à 10 Faire
  Afficher i
 Fait
Fin
```

Remarque

La variable i est déclarée directement dans l'entête de la boucle. Lors du premier tour de la boucle, elle vaut 1, puis elle est incrémentée de 1 pour valoir 2 dans le second tour. La boucle s'arrête une fois la borne maximum atteinte, ici 10.

Si on reprend ce que l'on vient d'expliquer sur les tableaux, il est donc possible accéder à chaque élément par son indice :

- Tab[0] : premier élément du tableau tab
- Tab[1] : second élément du tableau tab
- Etc.

Pour accéder à l'élément suivant, on ajoute donc juste 1 à l'indice en vérifiant que nous sommes toujours dans le tableau à l'aide de la taille. L'indice étant un nombre entier, nous pouvons le remplacer par une variable de type entier.

```
Variables :
 Indice : entier
 Tab : [] = [22, 24, 19, 18]


Début
 Indice ← 0
 Afficher 'L'element ' + Indice + ' est ' + tab[indice]
 Indice ← indice +1
 Afficher 'L'element ' + Indice + ' est ' + tab[indice]
 Indice ← indice +1
 Afficher 'L'element ' + Indice + ' est ' + tab[indice]
 Indice ← indice +1
 Afficher 'L'element ' + Indice + ' est ' + tab[indice]
```

```
Fin
```

Ce petit algorithme va simplement écrire :

- L'element 0 est 22
- L'element 1 est 24
- L'element 2 est 19
- L'element 3 est 18

Remarque

Nous avons juste accédé aux différents éléments grâce à notre variable « indice ». L'algorithme est encore long et redondant. Entre chaque affichage, il faut ajouter 1 à l'indice. L'idéal serait d'avoir une structure à laquelle on pourrait dire « *tu me fais varier indice de 0 à 3 et pour chaque valeur de cet indice, tu me fais un affichage*. » Nous avons de la chance, c'est exactement ce que fait une boucle. Une boucle va répéter une opération un certain nombre de fois et va contrôler le nombre de boucles grâce à un compteur. Notre algorithme précédent peut donc s'écrire avec une boucle :

```
Variable :
 Indice : entier
 Tab : entier[] = [22, 24, 19, 18]

Début
 Pour indice de 0 à 3 Faire
  Afficher 'L'element ' + Indice + ' est ' + tab[indice]
 Fait
Fin
```

Remarque

Dans le premier tour de la boucle, l'indice va valoir 0. Dans le second, il va valoir 1 et la boucle va s'arrêter lorsque l'indice vaut 3. Cette structure est intéressante car si nous devons traiter un tableau de 10 000 éléments, il suffit de modifier la valeur finale du compteur (pour indice de 0 à 9 999). En tant que développeur, cela nous fait moins de code à écrire et du coup moins de code à maintenir.

Exercice : Quiz

[solution n°2 p.32]

Question 1

Quelle déclaration de variable correspond à un tableau ?

- ○ Température : entier[]
- ○ Température : entier
- ○ Température : réel

Question 2

Je déclare un tableau : t : entier[] = [1, 2, 3, 4, 5]. Que vaut t[2] ?

- 2
- 3
- 4

Question 3

Que contient le tableau Age à la fin de l'algorithme ?

```
Variables :
 Age : entier [] = [22, 18, 25, 33]

Début
 Age[1] ← 35
 Age[2] ← 23
Fin
```

- [22, 35, 23, 33]
- [22, 18, 25, 33]
- [35, 23, 25, 33]

Question 4

Que contient le tableau Age à la fin de l'algorithme ?

```
Variables :
 Age : entier [] = [22, 18, 25, 33]

Début
 Age[1] ← Age[0] * 2
 Age[2] ← Age[3] - 5
Fin
```

- [44, 18, 25, 28]
- [22, 44, 28, 33]
- [44, 36, 20, 33]

Question 5

Que fait l'algorithme suivant ?

```
Variables :
 Indice : entier
 Tab : entier[ ] = [22, 24, 19, 18]

Début
 Pour indice de 0 à 3 Faire
```

```
    Tab[indice] ← Tab[indice] * 2
  Fait
Fin
```

- Il multiplie par deux tous les éléments du tableau Tab
- Il multiplie par deux le premier élément du tableau Tab
- Il multiplie par deux le dernier élément du tableau Tab

V. Tableaux et dimensions

A. Limites du tableau à 1 dimension

Utiliser un tableau est déjà une grande avancée pour regrouper des valeurs qui ont un lien et qui peuvent être traitées par lot. Malheureusement, nous allons rencontrer des cas qu'il est difficile de représenter avec un tableau simple.

Exemple

Imaginons que nous voulions stocker les distances en kilomètres qui séparent des villes. Nous pourrions très facilement représenter ces distances sous forme de grille dans un tableur.

Distance (kilomètres)	Paris	Lyon	Nice	Nantes	Strasbourg	Montpellier	Lille	Rennes	Reims	Saint-Étienne	Angers	Grenoble	Nîmes	Aix-en-Provence	Brest
Paris		462	931	380	488	746	219	348	143	522	293	575	710	759	589
Lyon	462		472	681	494	302	689	718	487	62	594	107	251	299	970
Nice	930	471		1 143	790	326	1 157	1 186	955	490	1 062	465	279	176	1 440
Nantes	380	682	1 145		860	824	597	107	515	662	88	786	874	972	296
Strasbourg	487	491	788	860		791	549	827	347	550	773	533	739	787	1 069
Montpellier	746	303	327	823	791		963	895	787	322	771	297	56	154	1 120
Lille	217	690	1 159	598	522	963		572	199	749	511	803	938	986	759
Rennes	347	718	1 186	106	827	894	572		483	699	126	823	948	1 013	241
Reims	143	487	955	516	347	786	198	483		546	429	600	735	783	725
Saint-Étienne	523	64	491	661	552	322	750	698	548		574	155	271	319	950
Angers	293	595	1 062	88	773	770	511	128	429	576		700	824	890	377
Grenoble	574	106	466	786	534	296	801	823	599	155	699		245	293	1 075
Nîmes	711	252	281	872	740	55	938	949	736	271	825	246		108	1 169
Aix-en-Provence	757	298	178	970	786	153	984	1 013	781	317	889	291	106		1 267
Brest	590	971	1 441	298	1 070	1 121	760	241	725	952	378	1 076	1 171	1 269	

La difficulté ici est que pour retrouver une distance, il nous faut deux informations : la ville de départ et la ville d'arrivée. C'est exactement le même principe que sur une grille de bataille navale. Une case est identifiée à la fois par sa colonne et par sa ligne. D'ailleurs, notre tableur référence également les lignes avec des nombres et les colonnes avec des lettres, ce qui nous permet rapidement de savoir ce qu'il y a dans la case D26.

Si nous essayons de rapprocher cette grille du fonctionnement d'un tableau, nous aimerions mettre en place un tableau qui, au lieu d'accéder à un élément par un indice permettrait d'accéder à un élément à l'aide de deux indices : un pour la ligne et un pour la colonne. C'est ce que va nous permettre de faire un tableau à 2 dimensions. Mais avant d'en expliquer les propriétés, nous allons le construire pas à pas, en fonction de ce que nous connaissons déjà.

Exemple

Prenons une grille de distances un peu plus simple.

Distance (kilomètres)	Paris	Lyon	Strasbourg
Paris		462	488
Lyon	462		494
Strasbourg	487	491	

Nous savons utiliser les tableaux, déclarons donc 3 tableaux, 1 pour chaque ville, ainsi que des variables entières représentant les positions des villes dans le tableau (Paris : 0, Lyon : 1, Strasbourg : 2).

```
Variables :
 Dist_Paris : entier[] = [0, 462, 488]
 Dist_Lyon : entier[] = [462, 0, 494]
 Dist_Strasbourg : entier[] = [487, 491, 0]
 Ind_paris : entier = 0
 Ind_lyon : entier = 1
 Ind_strasbourg : entier = 2

Début
…
Fin
```

Remarque

Avec ces quelques variables, je peux déjà résoudre mon problème de distance. Si je pars de Paris, il me suffit de prendre le tableau « dist_Paris ». Imaginons que je veuille récupérer la distance Paris → Strasbourg, je vais donc chercher dist_Paris[ind_strasbourg] (donc dist_Paris[2]),ce qui me retourne 488 km. En effet, le tableau dist_paris contient 3 valeurs :

- Indice 0 : 0 km - distance Paris Paris
- Indice 1 : 462 km - distance Paris Lyon
- Indice 2 : 488 km - distance Paris Strasbourg

J'arrive donc bien, avec cette méthode, à récupérer la distance que je cherche en prenant le tableau correspondant à la ville de départ et l'indice qui correspond à la ville d'arrivée. Cependant, cette méthode me fait déclarer autant de tableaux que je gère de villes, la France possédant plus de 36 000 communes, je ne peux pas utiliser ce principe à grande échelle.

B. Tableaux à plusieurs dimensions

Rappel

Nous avons vu que les tableaux pouvaient contenir plusieurs éléments de types simples (entier, réel, booléen, caractères). Imaginons maintenant qu'un tableau puisse contenir plusieurs éléments qui sont eux-mêmes des tableaux. Pour le comprendre, reprenons notre exemple de distance et faisons-le évoluer encore un peu. Dans mon exemple précédent, j'avais déclaré 3 tableaux :

- Dist_Paris : entier[ ] = [0, 462, 488]
- Dist_Lyon : entier[ ] = [462, 0, 494]
- Dist_Strasbourg : entier[ ] = [487, 491, 0]

Rappel

À partir de ces 3 tableaux, je vais déclarer un tableau qui va contenir 3 éléments : dist_Paris, dist_Lyon et dist_Strasbourg :

Dist : entier[][] = [dist_paris, dist_Lyon, dist_strasbourg]

Dist est de type entier[][], c'est un tableau de tableau d'entiers. En effet, dist_Paris, dist_Lyon et dist_Strasbourg sont des tableaux d'entiers. Si je remplace les variables dist par leurs valeurs, j'obtiens : Dist : entier[][] = [[0, 462, 488], [462, 0, 494], [487, 491, 0]].

Rappel

Je viens ici de déclarer un tableau à 2 dimensions qui n'est rien d'autre qu'un tableau dont chaque élément est lui-même un tableau. Si on reprend ce que nous savons déjà sur les tableaux et les indices, nous pouvons donc dire que :

- Dist[0] est le premier élément du tableau : [0, 462, 488],
- Dist[1] est le second élément du tableau : [462, 0, 494],
- Dist[2] est le troisième élément du tableau : [487, 491, 0].

Et donc la distance qui sépare Paris de Strasbourg est donc le 3^{ième} élément du premier élément de dist : Dist[0][2] est le troisième élément de [0, 462, 488] soit 488.

Et si nous utilisons les indices suivants :

- Ind_paris : entier = 0
- Ind_lyon : entier = 1
- Ind_strasbourg : entier = 2

Alors :

Dist[Ind_paris][Ind_strasbourg] est le troisième élément de [0, 462, 488] soit 488.

Nous avons donc bien fabriqué un tableau qui nous permet de rechercher un élément à partir de deux indices (ici, indice de la ville de départ et indice de la ville d'arrivée). Ce tableau à deux dimensions va nous servir à chaque fois que nous aurons à représenter une grille, une carte ou un plateau de jeu. Il n'y a aucune limite dans le nombre de dimensions que l'on peut mettre en place dans la déclaration d'un tableau.

C. Plusieurs dimensions en mémoire

Fondamental

On peut penser que la représentation mémoire d'un tableau à plusieurs dimensions peut être assez complexe mais elle suit les règles que nous avons vues jusqu'ici. En termes d'espace mémoire, la taille d'un tableau à 2 dimensions est facilement calculable. Si nous cherchons combien il y a d'éléments dans un tableau de n lignes sur k colonnes, nous nous rendons rapidement compte qu'il y en a n x k. La première chose que fait un programme pour stocker un tableau à deux dimensions, c'est déterminer de combien d'éléments il est constitué.

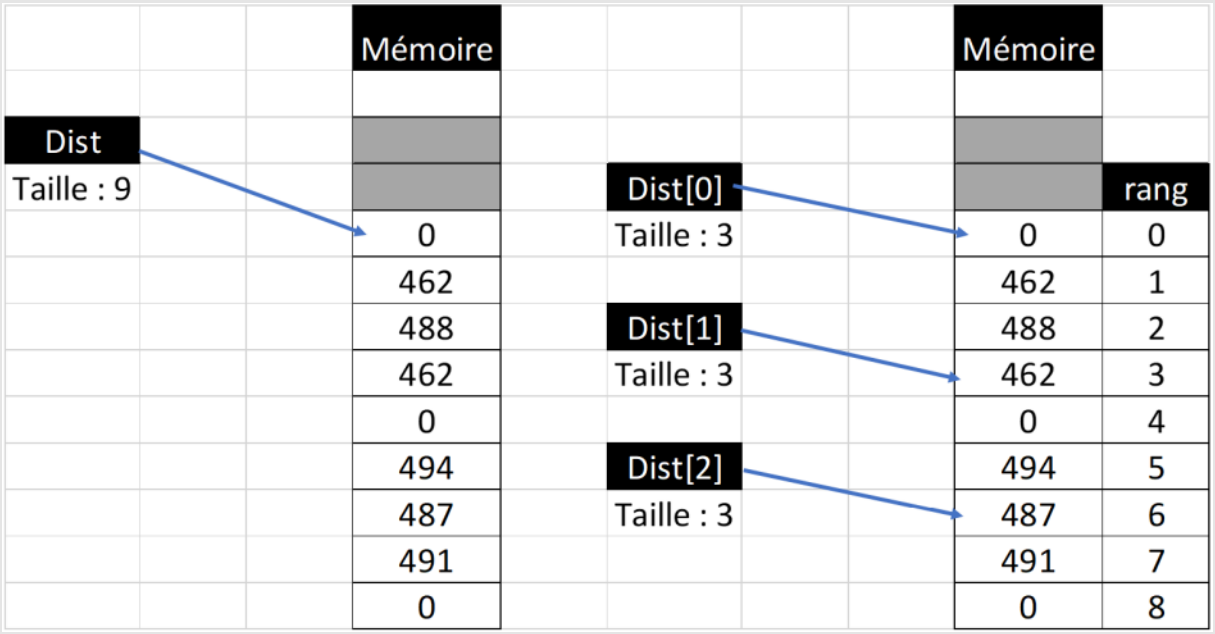
Méthode

Si je reprends l'exemple des distances entre les villes et du tableau des distances que j'ai déclaré :

Variables :

Dist : entier[ ][ ] = [[0, 462, 488], [462, 0, 494], [487, 491, 0]]

Dist est un tableau de 3 lignes sur 3 colonnes, il s'agit donc d'un tableau qui contient 3 x 3 = 9 éléments entiers. Le programme va donc réserver 9 emplacements contigus en mémoire. Ensuite, il va stocker le premier élément qui est [0, 462, 488]. Stocker un tableau dans une variable, nous savons le faire, il suffit de stocker chacun des 3 éléments dans les zones mémoires disponibles. Et nous procédons de même pour tous les éléments du tableau dist :



Comme nous l'avons déjà vu pour les caractères ou les nombres réels, l'ordinateur ne sait pas ce qu'est un tableau à deux dimensions. Néanmoins, les langages de programmation vont pouvoir à l'aide de quelques astuces simuler ce genre de tableau.

D. Tableau homogène avec Python et Numpy

Attention

Pour représenter un tableau en Python, nous utilisons communément le type liste. Attention, une liste Python n'est pas un tableau au sens où nous l'avons décrit dans ce cours. C'est une structure de données qui va bien plus loin.

Sur le plan des similitudes, une liste Python est un ensemble de valeurs manipulées au travers d'une seule variable. Nous pouvons accéder à un élément particulier avec l'aide des [] et même modifier un élément particulier. Cependant, Python permet d'avoir des éléments de plusieurs types dans la même liste :

l = [3, 'banane', 5.5, True]

Cette manière de définir une liste est tout à fait valide en Python. La liste l contient 4 éléments, un entier 3, une chaîne de caractères 'banane', un réel 5.5 et un booléen True. Un tableau tel que nous l'avons décrit est défini par le type d'éléments qu'il contient. On ne peut donc pas mettre des éléments de plusieurs types dans le même tableau.

Remarque

On pourrait être tenté de croire qu'une liste est donc un tableau, en mieux. Une liste vous permet de stocker n'importe quoi sans avoir à contrôler le contenu systématiquement. Python ne provoquera aucune erreur si vous ajoutez une chaîne de caractères dans une liste qui ne contient que des entiers. Cependant, le développeur devra être vigilant sur le contenu des listes et des instructions qu'il désire y appliquer.

Reprenons la liste définie au-dessus : l = [3, 'banane', 5.5, True]

Exemple

Que va-t-il se passer si je décide d'ajouter 3 à chaque élément de la liste ? Pas de problème pour le premier puisqu'il s'agit d'un numérique. Cependant, Python va provoquer une erreur lorsqu'il va se rendre compte que je lui demande d'ajouter 3 à une 'banane'. Cette souplesse que propose Python peut donc également être source d'erreur si le développeur n'est pas vigilant.

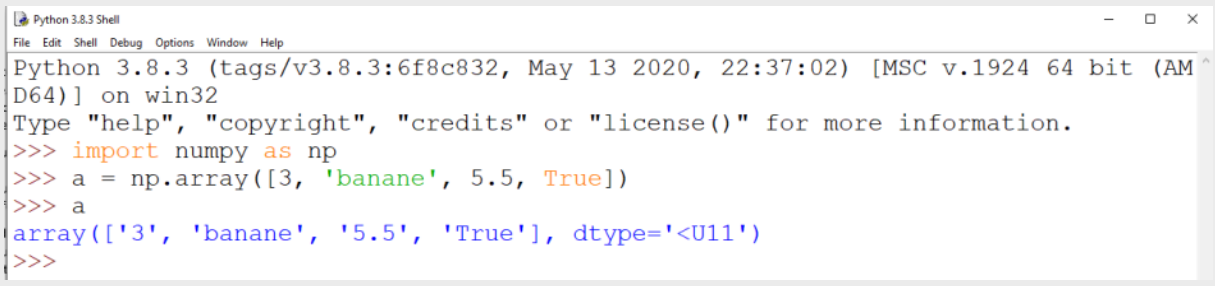
Remarque

Le fait que les données ne soient pas du même type dans une liste va aussi nous faire perdre du temps en contrôle. À chaque opération que nous voudrons faire sur la liste, Python sera obligé de vérifier qu'elle se prête bien au contenu de la liste. Si je veux la moyenne de la liste l, il y a de fortes chances pour que Python s'insurge une fois de plus.

Mais du coup, est-ce qu'il est possible de représenter vraiment un tableau en Python ? Oui, mais pas dans la version de base de Python, il faudra passer par une bibliothèque qui s'appelle Numpy. En effet, Numpy est une bibliothèque Python qui permet de représenter des tableaux et des matrices (tableaux à 2 dimensions) et constitue la base de nombreux modules de calcul scientifique pour Python. Les tableaux Numpy perdent l'aspect hétérogène des listes Python mais gagnent en vitesse de calcul. Les tableaux Numpy ne contenant qu'un seul type de données, il n'est plus nécessaire de faire autant de vérifications que pour les listes, et les traitements se passent donc plus vite.

Exemple

Si je crée un tableau Numpy à partir de la liste l ci-dessus :



Vous voyez que Numpy a tenté de trouver le type qui permettait de représenter toutes les valeurs et il a choisi de tout convertir en chaînes de caractères. Ce n'est pas forcément la meilleure chose pour faire des calculs mais il n'a pas eu le choix.

Complément

Numpy, comme la plupart des bibliothèques Python est un projet OpenSource. Vous aurez plus d'informations sur le site officiel à l'adresse web Numpy[1].

Exercice : Quiz

[solution n°3 p.33]

Question 1

Un tableau à deux dimensions permet de résoudre des problèmes :

- ○ Lorsqu'un élément du tableau est déterminé par un couple de valeurs
- ○ Lorsque les problèmes traitent de géométrie
- ○ Lorsque l'on travaille à deux sur le même projet

Question 2

Je déclare un tableau à deux dimensions : t : entier[][] = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]. Quelle est la valeur de l'élément t[1] ?

- ○ 2
- ○ [4, 5, 6]
- ○ [1, 2, 3]

Question 3

Je déclare un tableau à deux dimensions : t : entier[][] = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]. Quelle est la valeur de l'élément t[0][2] ?

- ○ 2
- ○ 3
- ○ 7

Question 4

Comment est stocké un tableau à plusieurs dimensions en mémoire ?

- ○ Il est stocké dans une mémoire elle-même à plusieurs dimensions
- ○ Il est stocké comme un tableau à une dimension : les éléments se suivent sur des zones mémoires contigües
- ○ Il est stocké dans un fichier .xls ou .csv

Question 5

1 https://numpy.org/

Il n'y a pas de différence entre un tableau et une liste Python.

- Vrai
- Faux

VII. Essentiel

Les tableaux nous permettent de regrouper les valeurs sous le nom d'une seule variable. Cette possibilité est très pratique pour pouvoir traiter ces données par lot, pour faire des analyses statistiques par exemple.

Même si des valeurs sont rassemblées au sein d'un tableau, il est possible d'accéder et de modifier chaque élément du tableau de manière indépendante. Nous utilisons les crochets pour accéder à un élément particulier d'un tableau : t[0] désigne le premier élément du tableau t.

Les langages de programmation implémentent toujours la structure tableau mais peuvent y apporter quelques évolutions comme la liste en Python ou le vecteur en C++. Le fait de travailler avec ces structures va permettre aux développeurs d'accéder directement à de nombreuses fonctionnalités propres au traitement des tableaux (moyenne, tri, recherche du max, recherche du min).

Les tableaux à plusieurs dimensions permettent de traiter des problèmes qui demandent une représentation spécifique. Nous pouvons ici chercher un élément avec plusieurs clés d'entrée :

- X et y pour un point sur un graphe
- Colonne et ligne pour une grille
- Ville de départ, ville d'arrivée dans un tableau de distance

Les tableaux à 2 dimensions servent également à représenter des matrices et des graphes. Les graphes recouvrent un grand nombre d'algorithme concernant les problèmes de plus court chemin, d'arbre couvrant ou de gestion de flux. Il est également possible de créer des tableaux possédant plus de 2 dimensions.

VIII. Auto-évaluation

A. Exercice

Nous allons solutionner le problème qui nous a servi de fil rouge tout au long de ce cours. Nous allons traiter nos relevés de températures sur un mois et nous allons calculer quelques informations statistiques sur ces données.

Naturellement, nous allons nous appuyer sur la structure tableau qui va nous permettre de gérer toutes ces valeurs simplement. Pour que notre solution soit la plus flexible et la plus réutilisable possible, nous allons permettre à un utilisateur d'entrer lui-même ses données.

Question 1

[solution n°4 p.35]

Déclaration du tableau

Dans cette première question, nous allons déclarer un tableau qui va pouvoir contenir les relevés de températures sur un mois (31 éléments) :

- Déclarer une variable de type tableau d'entier de 31 éléments
- Initialiser le tableau avec 31 éléments 0

Question 2

[solution n°5 p.35]

Alimentation du tableau

Maintenant que notre tableau est défini, nous allons permettre à un utilisateur de l'alimenter :

- Mettre en place une boucle de parcours du tableau en prenant soin de faire coïncider les bornes inférieures et supérieures de la boucle avec la taille du tableau. Vous pouvez utiliser la fonction longueur(tab) qui vous donne le nombre d'éléments dans la variable tableau tab.
- Dans la boucle, permettez à un utilisateur de modifier l'élément avec le mot clé « *Saisir* ».

Question 3

[solution n°6 p.35]

Moyenne de températures sur un mois

Nous voulons connaître la température moyenne du tableau. Pour cela, il nous faut :

- Déclarer une variable de type entier qui va contenir la somme des températures.
- Cette variable doit être initialisée à 0 au début de l'algorithme.
- Parcourir le tableau des températures et à chaque tour de boucle ajouter la température courante à la somme.
- Après la boucle, donnez la température moyenne en divisant la somme des températures par le nombre d'éléments dans le tableau des températures.

B. Test

Exercice 1

[solution n°7 p.36]

Redonner le bon type à chacune des valeurs ci-dessous.

2
[2, 5, 7]
8.17
[2.5, 5.7, 10.1]
Entier[]
Entier
Réel
Réel[]

Exercice 2

[solution n°8 p.36]

Faites en sorte que la phrase ci-dessous soit vraie.

Il est possible d'accéder directement à un élément bien précis d'un tableau en utilisant *[sa valeur | son type | son indice]*.

Exercice 3

[solution n°9 p.37]

Faites en sorte que la phrase ci-dessous soit vraie :

Le parcours d'un tableau peut se faire facilement à l'aide *[d'une boucle | d'une déclaration | d'une condition]*.

Exercice 4

[solution n°10 p.37]

Quelles sont les déclarations et les opérations qui se rapportent à des types simples, à des tableaux à 1 dimension, à des tableaux à deux dimensions :

`a ← 5 + 7` `a : entier` `a[0] ← a[0] x 2` `a : entier[] = [1, 2, 3]` `a : entier[][] = [[1, 2, 3],[4,5,6],[7,8,9]]`

`a[0][0] = a[0][1] + a[1][0]`

Type simple	Tableau à 1 dimension	Tableau à deux dimensions

Exercice 5 : Quiz

[solution n°11 p.37]

Exercice

Faites en sorte que la phrase ci-dessous soit vraie.

- J’ai le droit de déclarer un tableau à 3 dimensions
- Je n’ai pas le droit de déclarer un tableau à 3 dimensions
- Je ne peux pas déclarer un tableau à 3 dimensions

Solutions des exercices

Exercice p. 10 Solution n°1

Question 1

Un tableau en termes d'algorithmie est :

- [x] Une façon de stocker plusieurs valeurs dans une seule variable
- [ ] Un tableau de bord qui vous donne des indicateurs sur votre programme
- [ ] Une fenêtre dans laquelle vous tapez votre code

Un tableau est une façon de faire comprendre à la machine que nous voulons gérer une liste de valeurs pour effectuer des traitements par lot.

Question 2

Dans la mémoire, de quelle façon un programme stocke-t-il un tableau ?

- [ ] Chaque valeur est stockée là où il y a de la place, il n'est pas nécessaire que les zones mémoires soient contigües
- [x] Les valeurs sont stockées dans des zones mémoires contigües et la variable tableau pointe sur le premier élément
- [ ] Les tableaux sont stockés dans une mémoire spéciale qui leur est dédiée

Tous les éléments du tableau sont stockés dans des zones contigües, ce qui permet de passer rapidement à l'élément suivant en consultant la zone mémoire suivante.

Question 3

Un même tableau peut contenir :

- [ ] Des données de types différents
- [ ] Des données de types différents si elles sont numériques (entier, réel, booléen)
- [x] Un seul type de données

Comme toutes les autres variables, un tableau ne peut contenir qu'un seul type d'élément. Attention à ne pas confondre un tableau avec une liste de Python, par exemple. La liste peut contenir des éléments de types différents mais elle est considérée comme une extension de la notion de tableau.

Question 4

Pourquoi est-il important de connaitre la taille du tableau ?

- [x] Pour ne pas se retrouver à traiter un élément de la mémoire qui ne correspond pas au tableau
- [ ] Pour pouvoir le comparer aux autres tableaux
- [ ] Ce n'est important de connaitre sa taille, ce qui compte ce sont les valeurs

La variable tableau pointe sur le premier élément du tableau, on en connait donc le début. Il est tout aussi important d'en connaitre la fin pour ne pas traiter des éléments en dehors du tableau. En connaissant le début du tableau et sa taille, il est facile de calculer la fin du tableau.

Question 5

Si je mets des données dans un tableur, quel élément de mon tableur représente le mieux un tableau ?

- ○ Une cellule d'une feuille de travail
- ◉ Une colonne d'une feuille de travail
- ○ Un classeur

Q La cellule ne peut contenir qu'un seul élément. Le tableau contient une liste d'éléments, c'est donc bien la colonne qui se rapproche le plus de la représentation d'un tableau.

Exercice p. 19 **Solution n°2**

Question 1

Quelle déclaration de variable correspond à un tableau ?

- ◉ Température : entier[]
- ○ Température : entier
- ○ Température : réel

Q Un tableau est déclaré en tant que tableau grâce au []. Les crochets indiquent qu'il ne s'agit pas seulement d'une variable de type entier mais d'une variable de type tableau qui va contenir des éléments entiers.

Question 2

Je déclare un tableau : t : entier[] = [1, 2, 3, 4, 5]. Que vaut t[2] ?

- ○ 2
- ◉ 3
- ○ 4

Q Les indices utilisés pour accéder à un élément du tableau sont des valeurs numériques débutant avec 0. Dans ce cas, t[0] = 1, t[1] = 2 et t[2] = 3.

Question 3

Que contient le tableau Age à la fin de l'algorithme ?

```
Variables :
 Age : entier [] = [22, 18, 25, 33]

Début
 Age[1] ← 35
 Age[2] ← 23
Fin
```

- ◉ [22, 35, 23, 33]
- ○ [22, 18, 25, 33]
- ○ [35, 23, 25, 33]

Q Lorsque l'on accède à un élément particulier du tableau, seul cet élément est modifié. Age[1] ← 35 remplace le second élément (18) par 35. Il ne faut pas oublier que les indices commencent à 0. Age[2] ← 23 remplace le 3^{e} élément (25) par 23.

Question 4

Que contient le tableau Age à la fin de l'algorithme ?

```
Variables :
 Age : entier [] = [22, 18, 25, 33]

Début
 Age[1] ← Age[0] * 2
 Age[2] ← Age[3] - 5
Fin
```

- ○ [44, 18, 25, 28]
- ◉ [22, 44, 28, 33]
- ○ [44, 36, 20, 33]

Un élément de tableau peut être accédé à la fois en lecture et en écriture. Age[0] correspond au premier élément du tableau : 22. 22 est multiplié par 2, ce qui donne 44. 44 est stocké dans age[1] donc à l'emplacement 1. Age[3] correspond au 4e élément du tableau : 33. 5 est soustrait de 33, ce qui donne 28. 28 est stocké dans age[2] donc à l'emplacement 2.

Question 5

Que fait l'algorithme suivant ?

```
Variables :
 Indice : entier
 Tab : entier[ ] = [22, 24, 19, 18]

Début
 Pour indice de 0 à 3 Faire
  Tab[indice] ← Tab[indice] * 2
 Fait
Fin
```

- ◉ Il multiplie par deux tous les éléments du tableau Tab
- ○ Il multiplie par deux le premier élément du tableau Tab
- ○ Il multiplie par deux le dernier élément du tableau Tab

Nous utilisons une boucle pour répéter l'opération d'affectation pour tous les éléments du tableau. La variable indice va prendre successivement les valeurs 0, 1, 2, 3 et va nous permettre de parcourir tous les éléments du tableau. Chaque élément est doublé.

Exercice p. 26 **Solution n°3**

Question 1

Un tableau à deux dimensions permet de résoudre des problèmes :

- [x] Lorsqu'un élément du tableau est déterminé par un couple de valeurs
- [ ] Lorsque les problèmes traitent de géométrie
- [ ] Lorsque l'on travaille à deux sur le même projet

Un tableau à deux dimensions permet de représenter des problèmes du type bataille navale ou distance entre villes. Chaque élément est identifié par deux valeurs : colonne et ligne pour une case de la bataille navale, ville de départ et ville d'arrivée pour le tableau des distances.

Question 2

Je déclare un tableau à deux dimensions : t : entier[][] = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]. Quelle est la valeur de l'élément t[1] ?

- [ ] 2
- [x] [4, 5, 6]
- [ ] [1, 2, 3]

T est un tableau qui contient 3 éléments qui sont eux-mêmes des tableaux. t[0] = [1, 2, 3], t[1] = [4, 5, 6].

Question 3

Je déclare un tableau à deux dimensions : t : entier[][] = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]. Quelle est la valeur de l'élément t[0][2] ?

- [ ] 2
- [x] 3
- [ ] 7

T[0][2] indique que je cherche l'élément d'indice 2 dans l'élément t[0]. t[0] est le premier élément du tableau t, à savoir le tableau [1, 2, 3]. Dans cet élément, je vais chercher l'élément d'indice 2 qui est 3.

Question 4

Comment est stocké un tableau à plusieurs dimensions en mémoire ?

- [ ] Il est stocké dans une mémoire elle-même à plusieurs dimensions
- [x] Il est stocké comme un tableau à une dimension : les éléments se suivent sur des zones mémoires contigües
- [ ] Il est stocké dans un fichier .xls ou .csv

L'ordinateur ne sait pas que votre tableau a plusieurs dimensions, il va juste stocker les éléments en mémoire les uns derrières les autres. C'est le programme qui vous permettra d'accéder aux éléments au travers d'un couple d'indices.

Question 5

Il n'y a pas de différence entre un tableau et une liste Python.

- [ ] Vrai
- [x] Faux

Le tableau et la liste Python partagent beaucoup de similitudes mais connaissent aussi de grandes différences. Il est possible de stocker des types hétérogènes dans les listes Python par exemple.

p. 27 **Solution n°4**

```
Variables :
 Temperature : entier[31]

Début
 Pour i de 0 à 30 Faire
  Temperature[i] ← 0
 Fait
Fin
```

p. 27 **Solution n°5**

```
Variables :
 Temperature : entier[31]
 Indice : entier

Début
 Pour i de 0 à 30 Faire
  Temperature[i] ← 0
 Fait
 Pour indice de 0 à longueur(Temperature) Faire
  Saisir Temperature[indice]
 fpour
Fin
```

p. 28 **Solution n°6**

```
Variables :
 Temperature : entier[31]
 Indice : entier
 Somme : entier

Début
 Pour i de 0 à 30 Faire
  Temperature[i] ← 0
 Fait
 Somme ← 0
```

```
 Pour indice de 0 à longueur(Temperature) Faire
  Saisir Temperature[indice]
 fpour
 Pour indice de 0 à longueur(Temperature) Faire
  Somme ← Somme + Temperature[indice]
 fpour
 Afficher 'La température moyenne est : ' + Somme / longueur(Temperature)
Fin
```

Exercice p. 28 **Solution n°7**

Redonner le bon type à chacune des valeurs ci-dessous.

Entier[]
[2, 5, 7]
Entier
2
Réel
8.17
Réel[]
[2.5, 5.7, 10.1]

Type	Valeur
Entier	2
Réel	8.17
Entier[]	[2, 5, 7]
Réel[]	[2.5, 5.7, 10.1]

Les tableaux sont toujours déclarés avec les crochets suivants le type des éléments qu'ils contiennent. Un tableau d'entiers est donc de type entier[], un tableau de réels est de type réel[].

Exercice p. 28 **Solution n°8**

Faites en sorte que la phrase ci-dessous soit vraie.

Il est possible d'accéder directement à un élément bien précis d'un tableau en utilisant `son indice`.

L'indice d'un élément est son rang dans le tableau. En connaissant l'indice d'un élément, il est très facile d'accéder à l'élément en question avec les crochets : tab[indice]. La valeur ne peut pas être utilisée pour chercher un élément puisque l'on peut estimer que deux éléments d'un tableau pourraient avoir la même valeur. Tous les éléments d'un tableau ont le même type, le type ne détermine donc pas un élément précis du tableau.

Exercice p. 28 Solution n°9

Faites en sorte que la phrase ci-dessous soit vraie :

Le parcours d'un tableau peut se faire facilement à l'aide `d'une boucle`.

C'est la boucle qui permet de répéter une instruction plusieurs fois. La condition est une structure qui permet de faire un choix alors que la déclaration est l'instruction qui crée le tableau en lui donnant un nom, un type, une dimension et parfois son contenu.

Exercice p. 28 Solution n°10

Quelles sont les déclarations et les opérations qui se rapportent à des types simples, à des tableaux à 1 dimension, à des tableaux à deux dimensions :

Type simple	Tableau à 1 dimension	Tableau à deux dimensions
a : entier	a : entier[] = [1, 2, 3]	a : entier[][] = [[1, 2, 3],[4,5,6],[7,8,9]]
a ← 5 + 7	a[0] ← a[0] x 2	a[0][0] = a[0][1] + a[1][0]

Cas	Opération
Type simple	a : entier a ← 5 + 7
Tableau à 1 dimension	a : entier[] = [1, 2, 3] a[0] ← a[0] x 2
Tableau à deux dimensions	a : entier[][] = [[1, 2, 3], [4,5,6],[7,8,9]] a[0][0] = a[0][1] + a[1][0]

Les types simples n'utilisent pas les [], ni dans les déclarations, ni quand ils sont utilisés. Si on accède à un élément d'un tableau avec un seul indice, c'est un tableau à une dimension. Si on accède à un élément d'un tableau avec deux indices, il s'agit d'un tableau à 2 dimensions.

Exercice p. 29 Solution n°11

Exercice

Faites en sorte que la phrase ci-dessous soit vraie.

- ◉ J'ai le droit de déclarer un tableau à 3 dimensions
- ○ Je n'ai pas le droit de déclarer un tableau à 3 dimensions
- ○ Je ne peux pas déclarer un tableau à 3 dimensions

Le nombre de dimensions d'un tableau n'est pas limité. Nous avons vu que de toute façon, la machine va le stocker comme s'il s'agissait d'un tableau à une dimension. Il est donc tout à fait possible de déclarer des tableaux à 3, 4, 5 ou n dimensions.