

8 – Tableaux

8.1 Pyramide des âges

Imaginons un programme qui stoke la pyramide des âges des français (source : [pyramide des âges en 2021, source INSEE](#)).

Année de naissance	Âge révolu	Ensemble
2020	0	691 754
2019	1	714 585
2018	2	723 589
2017	3	741 105
2016	4	761 638
2015	5	782 397
2014	6	806 296
2013	7	813 727
2012	8	830 361
2011	9	836 626
2010	10	857 412
2009	11	846 079

...

Ce programme pourrait avoir deux fonctionnalités :

1. si l'utilisateur saisie un âge alors le programme affiche le nombre de français ayant cet âge-là
2. si l'utilisateur saisie deux âges, alors le programme affiche le nombre de français ayant un âge compris entre ces deux valeurs

Réaliser un programme qui réalise cela pour les âges compris entre 0 an(inclus) et 5 ans (exclus).

```
[1]: # essayer d'écrire le programme ici..  
...
```

On voit bien qu'il faudra utiliser beaucoup de variables. Une variable pour chaque année. C'est très pénible et heureusement, le langage Python propose une alternative bien pratique...

8.2 Notion de tableau

Création d'un tableau Un tableau permet de stocker plusieurs valeurs dans une seule variable et d'y accéder ensuite facilement. En Python, on construit un tableau en énumérant ses valeurs entre crochets et séparées par des virgules.

```
>>> t = [2, 3, 5]
```

```
[2]: t = [2, 3, 5]
```

Ici, on a déclaré une variable `t` contenant un tableau. Ce tableau contient trois entiers. Les valeurs sont **ordonnées**.

- la première valeur est 2
- la deuxième 3
- la troisième 5

On peut se représenter un tableau comme des cases consécutives

2	3	5
---	---	---

C'est en effet ainsi qu'un **tableau est organisé dans la mémoire de l'ordinateur** : ses valeurs y sont rangées consécutivement.

Indices d'un tableau Pour accéder à une valeur contenue dans le tableau `t`, il faut utiliser la notation `t[i]` où `i` désigne la numéro de la case à laquelle on veut accéder.

Les cases sont numérotées à partir de zéro.

Ainsi, la valeur contenue dans la première case du tableau est

```
>>> t[0]
```

```
2
```

```
[3]: t[0]
```

```
3]: 2
```

Écrire un code qui essaye d'accéder à une case en dehors des limites du tableau.

```
[4]: t[3]
```

```
↳
```

```
IndexError                                Traceback
↳ (most recent call last)
```

```
  /tmp/ipykernel_118370/3611909921.py in <module>
----> 1 t[3]
```

```
IndexError: list index out of range
```

On dit que 0 est l'**indice** de la première case.

La **taille** d'un tableau est son nombre de cases.

On obtient la taille d'un tableau avec l'opération `len(t)`.

```
>>> len(t)
```

```
3
```

```
[5]: len(t)
```

```
5]: 3
```

Les indices d'un tableau t prennent donc des valeurs comprises entre 0 et $\text{len}(t)-1$.

Ce qui donne la représentation mentale suivante :

0	1	2	2	3	5
---	---	---	---	---	---

Mais **attention**, seules les valeurs sont stockées en mémoire de l'ordinateur. Les indices n'ont pas besoin d'être matérialisés.

Retour sur la pyramide des âges Créer un tableau `pda` contenant les 10 premiers effectifs de la pyramide des âges.

```
[6]: pda = [691754, 714585, 723589, 741105, 761638, 782397, 806296, 813727, 830361, 836626]
```

Dans la case i du tableau `pda` on retrouve le nombre de français ayant exactement l'âge i .

Créer un programme qui demande un âge à l'utilisateur et affiche le nombre de français ayant cet âge-là.

Modification du contenu d'un tableau Le contenu d'un tableau peut être modifié. Pour cela, on utilise un **affectation** exactement comme on le ferait avec une variable.

Ainsi, pour modifier le contenu de la seconde case du tableau `t` pour y remplacer la valeur 3 par la valeur 17 :

```
>>> t[1] = 17
```

```
[7]: t[1] = 17
```

Modifier le tableau `pda` pour y traduire une naissance.

```
[8]: pda[0] = pda[0] + 1
      pda
```

```
[8]: [691755,
      714585,
      723589,
      741105,
      761638,
      782397,
      806296,
      813727,
      830361,
      836626]
```

8.3 – Parcours d'un tableau

En utilisant le tableau `pda` de la partie précédente, proposer un programme permettant d'obtenir le nombre de français ayant entre 1 et 8 ans.

```
[9]: pda = [691754, 714585, 723589, 741105, 761638, 782397, 806296, 813727, 830361, 836626]
      pda[1] + pda[2] + pda[3] + pda[4] + pda[5] + pda[6] + pda[7] + pda[8]
```

```
[9]: 6173698
```

Comment s'y prendre quand le tableau est beaucoup plus long ?

Une meilleure solution consiste à utiliser une **boucle** pour parcourir les cases du tableau concernées en **accumulant** le nombre total dans une variable.

On commence par initialiser la variable `n` à 0 :

```
n = 0
```

Puis on effectue une boucle `for` donnant successivement à la variable `i` toutes les valeurs entre 1 et 8, et on ajoute à chaque fois la valeur `pda[i]` à la variable `n` :

```
for i in range(1, 9):  
    n += pda[i]
```

On peut vérifier qu’après la boucle la variable `n` contient bien la valeur précédente.

```
[10]: n = 0  
for i in range (1, 9):  
    n += pda[i]  
  
n
```

```
0]: 6173698
```

Écrire un programme qui demande à l’utilisateur

- un âge minimal
- un âge maximal

et qui renvoie le nombre de français dont l’âge est compris entre le minimal (inclu) et le maximal (exclu).

Par exemple :

```
>>> âge minimum (inclu) : 1  
>>> âge maximum (exclu) : 9  
il y a 6173698 personnes qui ont entre 1 et 9 ans
```

Pour parcourir **toutes les cases** d’un tableau `t`, on peut utiliser une boucle `for` allant de 0 inclus à `len(t)` exclu.

Proposer un programme qui affiche le nombre total de français contenus dans votre tableau `pda`.

```
[11]: population_totale = 0  
for i in range (0, len(pda)):  
    population_totale += pda[i]  
  
print(population_totale)
```

```
7702078
```

8.4 – Construire de grands tableaux

Pour construire des tableaux d'une taille arbitraire, on peut utiliser la syntaxe :

```
>>> t = [0] * 1000
```

On donne la taille du tableau après le symbole `*` (ici 1000) et entre crochet une valeur qui sera donnée à toutes les cases du tableau (ici 0).

On obtient donc un tableau de taille 1000 dont toutes les cases contiennent pour l'instant la valeur 0.

Créer un tel tableau et vérifier sa taille.

```
[12]: t = [0] * 1000  
      len(t)
```

```
2]: 1000
```

Une fois le tableau créé, on peut le **remplir** avec des valeurs de son choix.

Remplir le tableau `t` avec la valeur des carrés des 1000 premiers entiers.

```
[13]: for i in range(0, 1000):  
      t[i] = i * i
```

Attention, l'opération `[0]*1000` n'implique pas une multiplication (même si on voit le symbole `*`). Il s'agit là d'une opération spécifique aux tableaux.

Il est possible de *concaténer* deux tableaux, c'est-à-dire de construire un nouveau tableau contenant bout à bout les éléments des deux tableaux. On le fait avec l'opération `+` :

```
>>> [8, 13, 21] + [34, 55]  
[8, 13, 21, 34, 55]
```

Tableaux et chaînes de caractères Les chaînes de caractères offrent certaines ressemblance avec les tableaux.

- taille d'une chaîne de caractère : `len(ch)`
- accès au i-ème caractère d'une chaîne : `ch[i]`
- concaténation : `+`
- répétition d'un valeur : `*`

1. Créer la variable `ch` initialisée à la chaîne de caractères "bonjour".
2. Afficher la longueur de la chaîne.
3. Vérifier que le 3ème caractère de la chaîne est bien le `n`
4. Concaténer cette chaîne avec la deuxième chaîne " le monde !"

```
[14]: ch = "bonjour"
      print(len(ch))
      print(ch[2])
      ch = ch + " le monde !"
      print(ch)
```

7

n


bonjour le monde !

Attention, contrairement aux tableaux, les caractères d'une chaîne ne peuvent pas être modifiés. On dit qu'une chaîne est immuable.

```
[15]: ch[2] = "Z"
```



TypeError

Traceback (most recent call last)

```
/tmp/ipykernel_118370/1875237351.py in <module>
----> 1 ch[2] = "Z"
```



```
TypeError: 'str' object does not support item_
↪assignment
```

8.5 – Tableaux et variables

Activité

Le bloc de code ci-dessous fait appel à un outil **indispensable** : [Python Tutor](#). Je vous invite à l'utiliser en autonomie chaque fois que vous voulez comprendre ou visualiser un algorithme, une erreur, etc.

1. Exécuter le bloc de code
2. Visualiser l'exécution du code pas à pas (en cliquant sur Forward >) et répondre aux questions suivantes :
 1. Comment s'appelle l'instruction de la ligne 1?
 2. Est-ce que l'instruction de la ligne 4 modifie le contenu de la variable `x`?
 3. Même question pour l'instruction de la ligne 7.
 4. Proposer une explication...

```
[ ]: from tutor import tutor

x = 1
x = 2
y = x
y = 3
t = [1, 2, 3]
u = t
u[2] = 7
print(t[2])

tutor()
```

... bloc de réponse ...

Petite explication de ce phénomène d'**effet de bords** lié aux tableaux.

En réalité, ce n'est pas un tableau qui est stocké en mémoire, mais l'*adresse mémoire* du tableau (c'est pour cela que la représentation adoptée est celle d'une

flèche : l'adresse mémoire désigne l'emplacement où se trouve le tableau).

Du coup, les deux variables `t` et `u` désignent **le même** tableau ! Et donc toute modification du contenu du tableau `u` sera visible dans le tableau `t`.

8-6 – Tableaux et fonctions

Tableau en argument Dans le même ordre d'idée, nous allons passer un tableau en argument d'une fonction dans l'exemple ci-dessous.

Activité

- Observer le code ci-dessous et répondre aux questions suivantes :
 - À quel type de valeur correspondent les variables `x` et `t` lorsqu'elles sont initialisées ?
 - Quelles sont les valeurs initiales des variables `x` et `t` ?
 - Que fait la fonction `f` ?
- Exécuter le bloc de code ci-dessous.
- Visualiser l'exécution du code pas à pas et répondre aux questions suivantes :
 - Quelles sont les valeurs des variables `x` et `t` **après** l'exécution complète de l'instruction ligne 8 ?
 - Expliquer pourquoi la variable `x` est inchangée alors que `t` a été modifiée ?

```
[ ]: from tutor import tutor

x = 1
t = [1, 2, 3]

def f(a, b):
    a = a + 1
    b[a] = 7

f(x, t)

tutor()
```

... bloc de réponse ...

Comme on le constate, le contenu du tableau `t` a été modifié par la fonction, mais pas le contenu de la variable `x` : **une fonction peut modifier le contenu d'un tableau qui lui est passé en argument.**

Nous écrirons par la suite de nombreuses fonctions opérant ainsi sur les tableaux. Par exemple en créant des fonctions qui trient le contenu d'un tableau.

Tableau renvoyé par une fonction Une fonction peut aussi renvoyer un tableau comme résultat.

Pour bien comprendre ce qu'il se passe, illustrons cela par un autre exemple.

Activité

1. Lire le code ci-dessous et répondre aux questions suivantes :
 1. Que fait la fonction `carres(n)` lorsqu'on lui passe un nombre entier `n` en argument ?
 2. Lorsqu'on exécute l'instruction `c = carres(4)`, la fonction s'exécute. En arrivant à la ligne `return`, combien y a-t-il de variables locales à la fonction en mémoire ?
 3. Une fois que la fonction s'est complètement exécutée, combien reste-t-il de variables locales à la fonction en mémoire ?
2. Exécuter le bloc de code ci-dessous.
3. Visualiser l'exécution du code pas à pas et répondre aux questions suivantes :
 1. Est-ce que la variable `t` a survécu à l'appel à la fonction ?
 2. Est-ce que le contenu de la variable `t` a survécu ?

```
[ ]: from tutor import tutor
```

```
def carres(n):  
    t = [0] * n  
    for i in range(n):  
        t[i] = i * i  
    return t  
  
c = carres(4)  
  
tutor()
```