

[TP1]

Analyser et manipuler des données avec Python

1. Variables et types

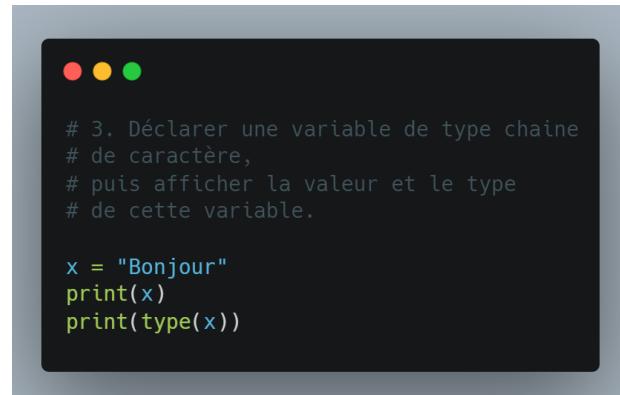
1.1 Variables

```
# 1. Déclarer et initialiser deux variables x et y :  
  
x = 2  
y = 5  
  
# puis calculer et afficher :  
# a. La somme  
a = x + y  
  
# b. La multiplication  
b = x * y  
  
# c. La division  
c = x / y
```

```
# 2. Afficher les types des résultats de vos calculs  
# a. Les résultats des calculs  
  
print(a)  
print(b)  
print(c)  
  
# b. Les types des résultats des calculs  
  
print(type(a))  
print(type(b))  
print(type(c))
```

Voici le résultat des prints :

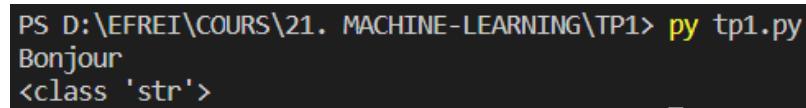
```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
7  
10  
0.4  
<class 'int'>  
<class 'int'>  
<class 'float'>
```



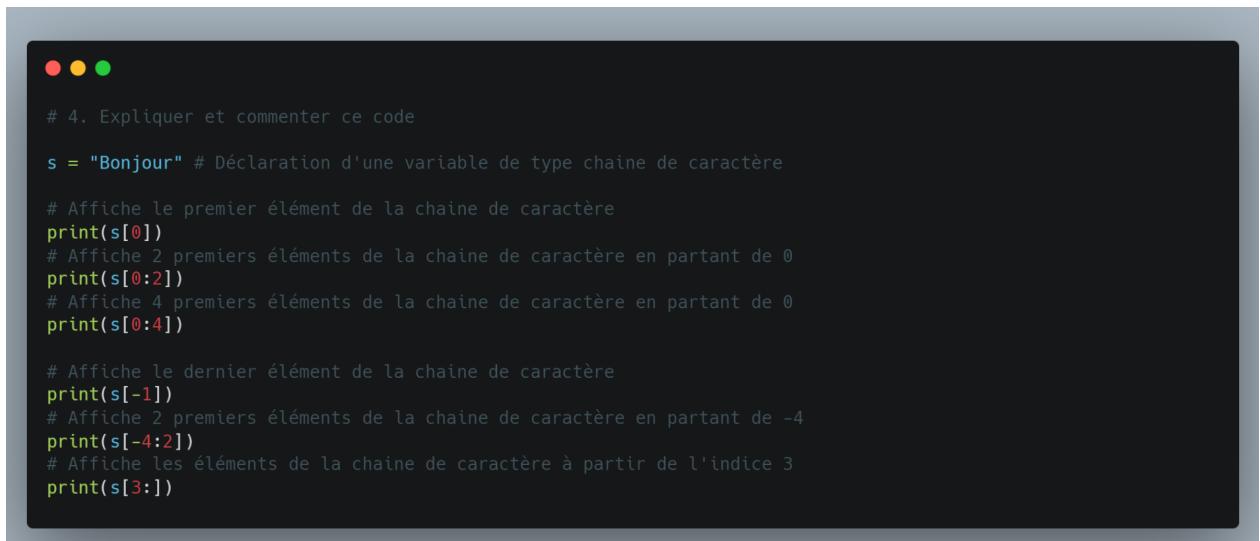
```
# 3. Déclarer une variable de type chaîne
# de caractère,
# puis afficher la valeur et le type
# de cette variable.

x = "Bonjour"
print(x)
print(type(x))
```

Voici le résultat :



```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
Bonjour
<class 'str'>
```



```
# 4. Expliquer et commenter ce code

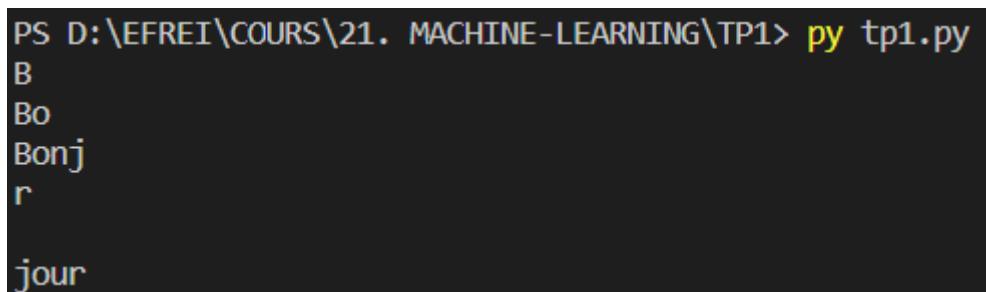
s = "Bonjour" # Déclaration d'une variable de type chaîne de caractère

# Affiche le premier élément de la chaîne de caractère
print(s[0])
# Affiche 2 premiers éléments de la chaîne de caractère en partant de 0
print(s[0:2])
# Affiche 4 premiers éléments de la chaîne de caractère en partant de 0
print(s[0:4])

# Affiche le dernier élément de la chaîne de caractère
print(s[-1])
# Affiche 2 premiers éléments de la chaîne de caractère en partant de -4
print(s[-4:2])
# Affiche les éléments de la chaîne de caractère à partir de l'indice 3
print(s[3:])
```

On affiche, dans un print, la chaîne de caractères 's' et on lui donne des indices et des valeurs à afficher.

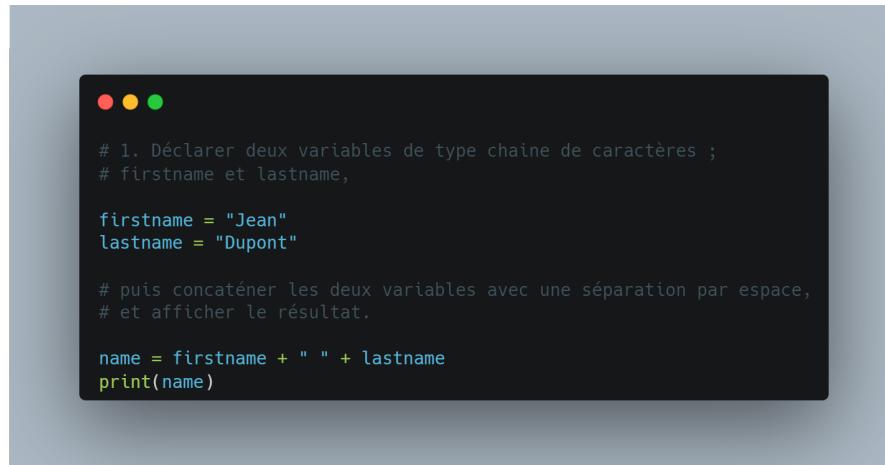
Voici les résultats des prints :



```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
B
Bo
Bonj
r

jour
```

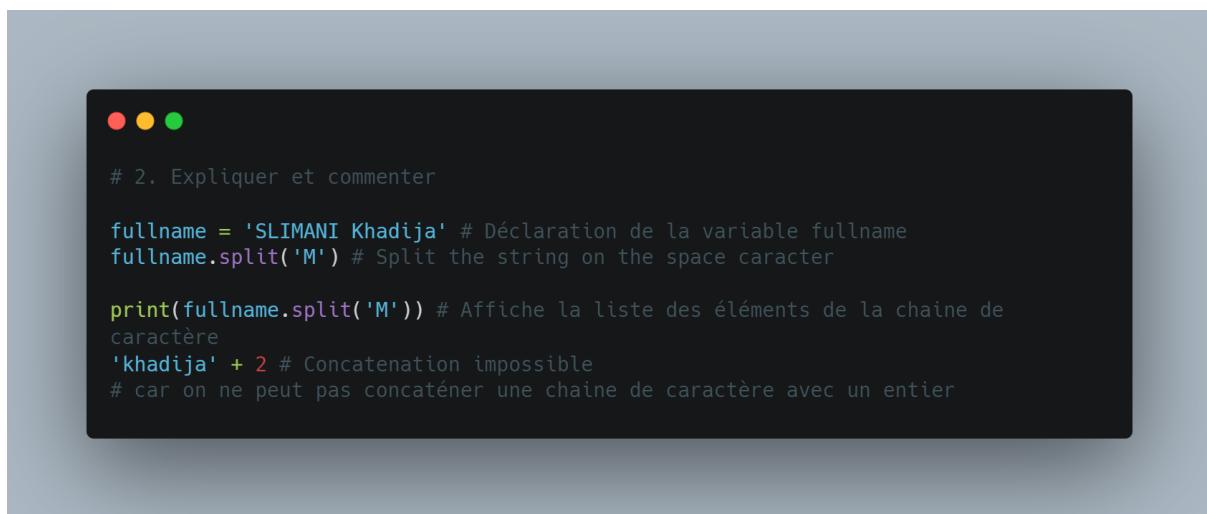
1.2 Opérations avec des chaînes



```
# 1. Déclarer deux variables de type chaîne de caractères ;  
# firstname et lastname,  
  
firstname = "Jean"  
lastname = "Dupont"  
  
# puis concaténer les deux variables avec une séparation par espace,  
# et afficher le résultat.  
  
name = firstname + " " + lastname  
print(name)
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
Jean Dupont
```



```
# 2. Expliquer et commenter  
  
fullname = 'SLIMANI Khadija' # Déclaration de la variable fullname  
fullname.split('M') # Split the string on the space character  
  
print(fullname.split('M')) # Affiche la liste des éléments de la chaîne de caractère  
'khadija' + 2 # Concaténation impossible  
# car on ne peut pas concaténer une chaîne de caractère avec un entier
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
['SLI', 'ANTI Khadija']  
Traceback (most recent call last):  
  File "D:\EFREI\COURS\21. MACHINE-LEARNING\TP1\tp1.py", line 77, in <module>  
    'khadija' + 2 # Concaténation impossible  
TypeError: can only concatenate str (not "int") to str
```

```
price = 3.24 # Déclaration de la variable price
num_items = 5 # Déclaration de la variable num_items
year = 2018 # Déclaration de la variable year

# Affiche la chaîne de caractère et les variables
print('the price or the year is ', year, 'is', price)
# Affiche la chaîne de caractère et la variable num_items
print("the price is %i" %num_items)
# Affiche la chaîne de caractère et la variable price
print("the price is %f" %price)
# Affiche la chaîne de caractère et les variables year et price
print("the price for the year %i is %f" %(year, price))
# Affiche la chaîne de caractère et les variables num_items, price et num_items*price
print('Jean bought () item(s) at a price of () each or a total of ()'.format(num_items, price, num_items*price))
```

Voici le résultat des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
the price or the year is 2018 is 3.24
the price is 5
the price is 3.240000
the price for the year 2018 is 3.240000
Jean bought 5 item(s) at a price of 3.24 each or a total of 16,2
```

1.3 Les fonctions

```
# 1. Ecrire une fonction add_numbers qui prend deux nombres
# et les additionne.

def add_numbers(x, y):
    return x + y
```

2. Est-ce qu'une fonction peut être affectée à une variable ?

Non plutôt l'inverse, une variable peut être affectée à une fonction. Il faut comprendre :

- Que le nom d'une variable passée à un argument n'a rien à voir avec le nom du paramètre correspondant dans la fonction
- Que les noms peuvent être identiques mais il faut comprendre qu'ils ne désignent pas la même chose.

1.4 Expressions conditionnelles

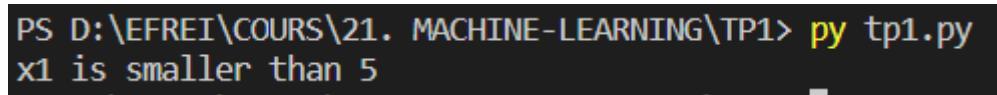


```
# Tester ce code et commenter

x1 = 2 # Déclaration de la variable x1
x2 = 7 # Déclaration de la variable x2

# Si x1 est supérieur à 5
if x1 > 5:
    # Affiche 'x1 is greater than 5'
    print('x1 is greater than 5')
# Sinon
else:
    # Affiche 'x1 is smaller than 5'
    print('x1 is smaller than 5')
```

Voici le résultat des prints :



```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
x1 is smaller than 5
```

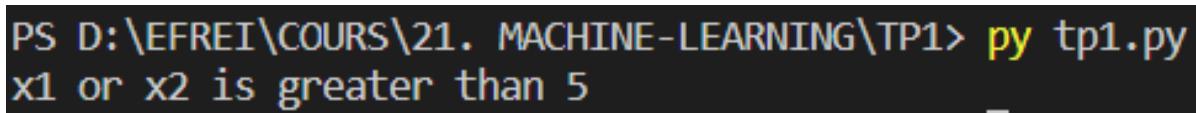


```
# Tester ce code et commenter

x1 = 2 # Déclaration de la variable x1
x2 = 7 # Déclaration de la variable x2

# Si x1 est supérieur à 5
if x1 > 5 and x2 > 5:
    # Affiche 'x1 and x2 are greater than 5'
    print('x1 and x2 are greater than 5')
# Sinon si x1 est supérieur à 5
elif x1 > 5 or x2 > 5:
    # Affiche 'x1 or x2 are greater than 5'
    print('x1 or x2 is greater than 5')
# Sinon
else:
    # Affiche 'x1 and x2 are smaller than 5'
    print('x1 and x2 are smaller than 5')
```

Voilà le résultat des prints :



```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
x1 or x2 is greater than 5
```

2. Types avancés

2.1 Listes

```
● ● ●  
# 1. Déclarer et initialiser une liste  
liste = [1,2,3]  
  
# 2. Afficher le contenu de la liste et son type  
print(liste)  
print(type(liste))
```

Voici le résultat des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
[1, 2, 3]  
<class 'list'>
```

```
● ● ●  
# 3. Les listes sont indexées de la même manière que les chaînes,  
# tester et commenter chaque ligne de ce code :  
  
x = [1,2,3,4,5,6,7,8,9,10] # Déclaration de la variable x  
  
# Affiche le premier élément de la liste  
print(x[0])  
# Affiche 2 premiers éléments de la liste en partant de 0  
print(x[1:3])  
# Affiche 4 premiers éléments de la liste en partant de 0  
print(x[:2])  
# Affiche le dernier élément de la liste  
print(x[-2:])  
# Affiche 2 premiers éléments de la liste en partant de -4  
print(x[-4:-2])  
# Affiche les éléments de la liste à partir de l'indice 3  
print(x[3::])
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
1
[2, 3]
[1, 2]
[9, 10]
10
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
# 4. Les listes sont une structure de données mutable
# ==> modifiée après la création,
# expliquer ce code :

x = [1,2,3,4,5,6,7,8,9,10] # Déclaration de la variable x

# Ajoute 3.3 à la fin de la liste
x.append(3.3) # Use append to append an object to a list
# Affiche la liste x
print(x)

# Change la valeur de l'indice 0 de la liste x de 1 à -1
x[0] = -1 # Change the value of the first element from 1 to -1
# Affiche la liste x
print(x)
```

La méthode `.append` en python permet d'ajouter un élément à une liste.
Il est possible de remplacer une valeur dans une liste en sélectionnant la valeur et d'attribuer une nouvelle valeur

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 3.3]
[-1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 3.3]
```

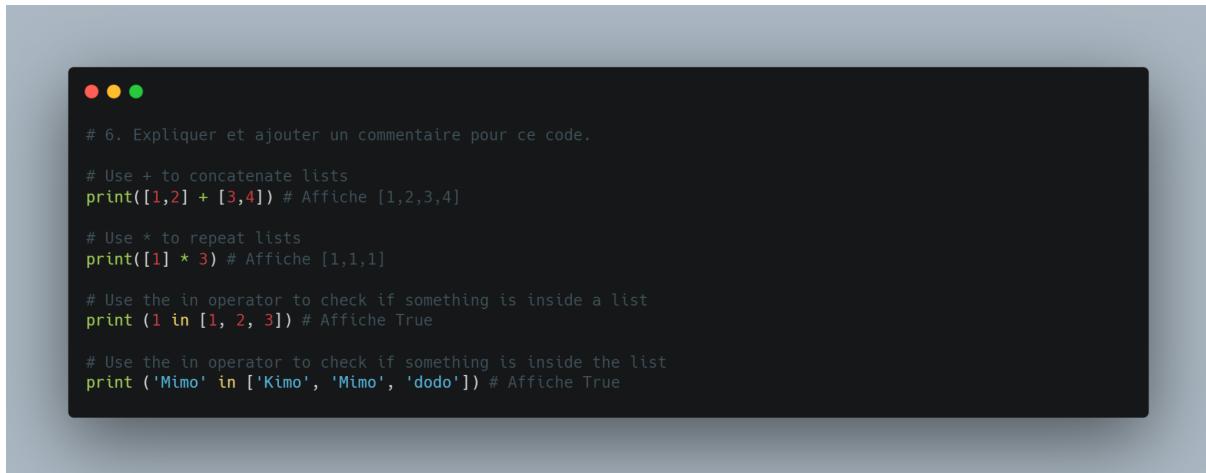
```
# 5. Ecrire le code qui permet de parcourir
# et afficher chaque élément de la liste.

nums = [1,2,3,4,5,6,7,8,9,10] # Déclaration de la variable nums

# Pour chaque élément de la liste nums
for num in nums:
    # Affiche l'élément
    print(num)
```

Voici les résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
1
2
3
4
5
6
7
8
9
10
```



```
# 6. Expliquer et ajouter un commentaire pour ce code.

# Use + to concatenate lists
print([1,2] + [3,4]) # Affiche [1,2,3,4]

# Use * to repeat lists
print([1] * 3) # Affiche [1,1,1]

# Use the in operator to check if something is inside a list
print (1 in [1, 2, 3]) # Affiche True

# Use the in operator to check if something is inside the list
print ('Mimo' in ['Kimo', 'Mimo', 'dodo']) # Affiche True
```

Il est possible de fusionner deux listes, de répéter autant de fois une liste, et de voir si une valeur est dans une liste. Cela n'impacte pas le type de liste, cela peut être aussi bien des chaînes de caractères (string) ou des nombres (int).

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[1, 2, 3, 4]
[1, 1, 1]
True
True
```

2.2 Tuples

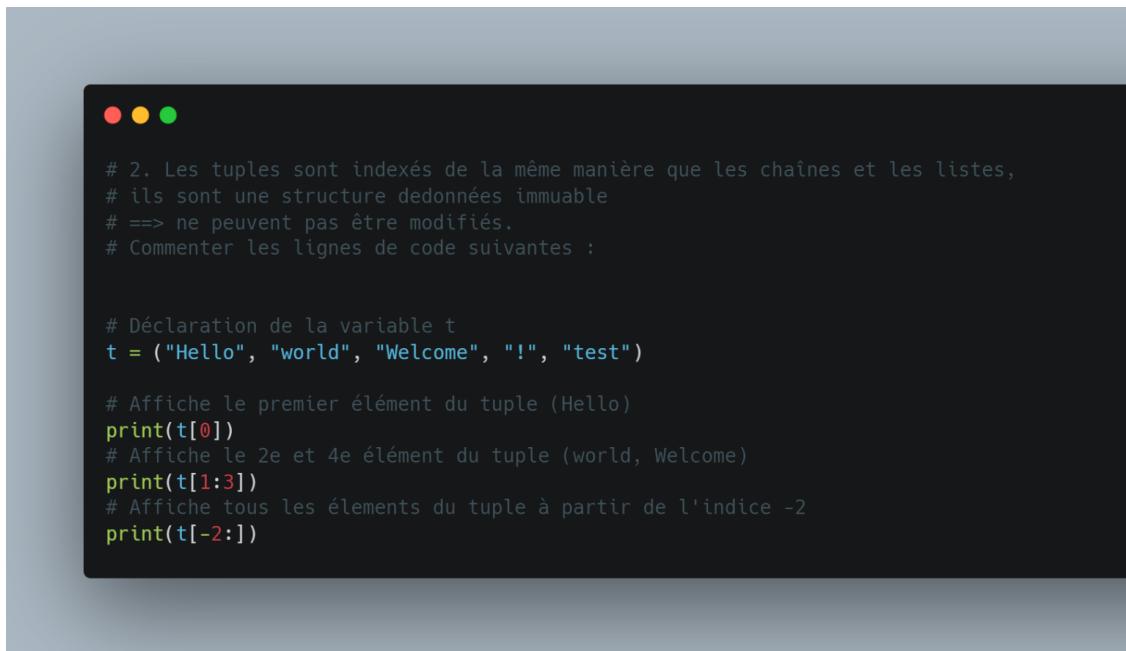


```
# 1. Déclarer un tuple,
# puis afficher son contenu et son type.

mon_tuple = ("Hello", "world")
print(mon_tuple)
print(type(mon_tuple))
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
('Hello', 'world')
<class 'tuple'>
```



The screenshot shows a terminal window with three colored window control buttons (red, yellow, green) at the top. The terminal displays the following Python code:

```
# 2. Les tuples sont indexés de la même manière que les chaînes et les listes,
# ils sont une structure dédonnées immuable
# ==> ne peuvent pas être modifiés.
# Commenter les lignes de code suivantes :

# Déclaration de la variable t
t = ("Hello", "world", "Welcome", "!", "test")

# Affiche le premier élément du tuple (Hello)
print(t[0])
# Affiche le 2e et 4e élément du tuple (world, Welcome)
print(t[1:3])
# Affiche tous les éléments du tuple à partir de l'indice -2
print(t[-2:])
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
Hello
('world', 'Welcome')
('!', 'test')
```



The screenshot shows a terminal window with three colored window control buttons (red, yellow, green) at the top. The terminal displays the following Python code, which attempts to modify a tuple:

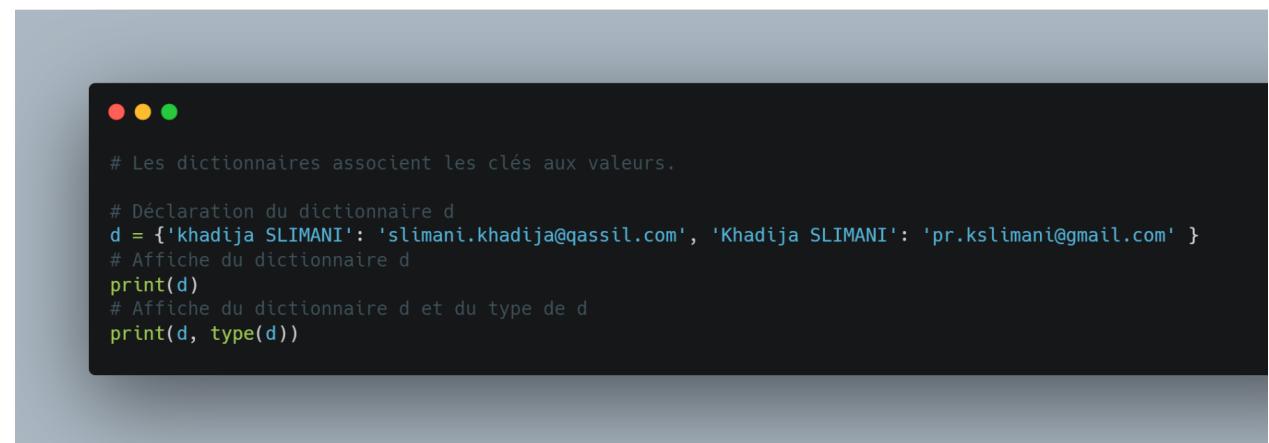
```
# Déclaration de la variable t
t = ("Hello", "world", "Welcome", "!", "test")

# Change the value of the first element from 1 to -1
# Does not work because tuples are immutable
t[0] = -1
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
Traceback (most recent call last):
  File "D:\EFREI\COURS\21. MACHINE-LEARNING\TP1\tp1.py", line 262, in <module>
    t[0] = -1
TypeError: 'tuple' object does not support item assignment
```

2.3 Dictionnaire

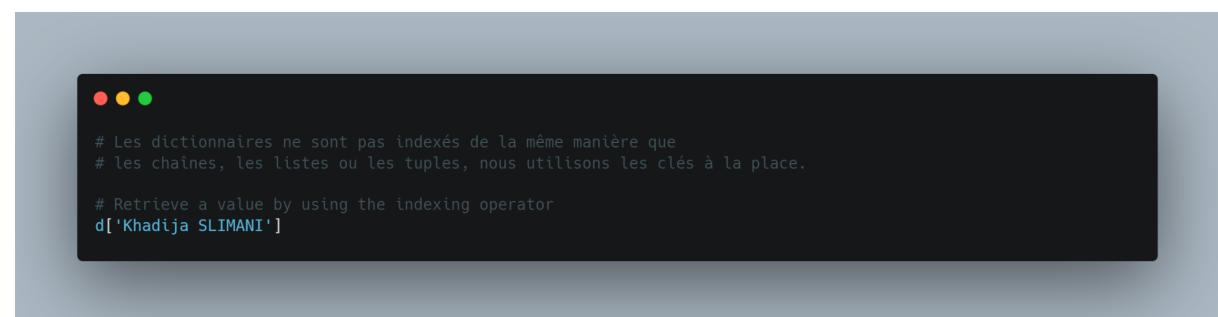


```
# Les dictionnaires associent les clés aux valeurs.

# Déclaration du dictionnaire d
d = {'khadija SLIMANI': 'slimani.khadija@qassil.com', 'Khadija SLIMANI': 'pr.kslimani@gmail.com' }
# Affiche du dictionnaire d
print(d)
# Affiche du dictionnaire d et du type de d
print(d, type(d))
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
{'khadija SLIMANI': 'slimani.khadija@qassil.com', 'Khadija SLIMANI': 'pr.kslimani@gmail.com'}
{'khadija SLIMANI': 'slimani.khadija@qassil.com', 'Khadija SLIMANI': 'pr.kslimani@gmail.com'} <class 'dict'>
```



```
# Les dictionnaires ne sont pas indexés de la même manière que
# les chaînes, les listes ou les tuples, nous utilisons les clés à la place.

# Retrieve a value by using the indexing operator
d['Khadija SLIMANI']
```

```
# Déclaration du dictionnaire d
d = {'khadija SLIMANI': 'slimani.khadija@qassil.com', 'Khadija SLIMANI': 'pr.kslimani@gmail.com' }

# 1. Récupérer les clés du dictionnaire
d.keys()
# afficher les clés du dictionnaire
print(d.keys())

# 2. Récupérer les valeurs du dictionnaire
d.values()
# Afficher les valeurs du dictionnaire
print(d.values())

# 3. Itérer sur tous les éléments (afficher « clés : valeur » de tous les éléments)
for key, value in d.items():
    print(key, value)
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
dict_keys(['khadija SLIMANI', 'Khadija SLIMANI'])
dict_values(['slimani.khadija@qassil.com', 'pr.kslimani@gmail.com'])
khadija SLIMANI slimani.khadija@qassil.com
Khadija SLIMANI pr.kslimani@gmail.com
```

3. Dates et heures

```
# Détailler et expliquer chaque ligne de ce code :

# Importation du module datetime sous le nom dt
import datetime as dt
# Importation du module time sous le nom tm
import time as tm
```

On utilise des librairies afin de faire des fonctionnalités complexes comme la gestion des dates et des horaires. On utilise la méthode ‘import’ de python pour appeler ses librairies. Dans notre cas on importe la librairie ‘datetime’, qui nous sert à obtenir la date, on la nomme sous le nom de dt.

On importe aussi la librairie ‘time’ afin d’obtenir l’heure, on la nomme tm.

```
# Affiche le temps en secondes depuis le 1er janvier 1970
print(tm.time())

# Affiche la date et l'heure actuelle
print(dt.datetime.now())
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
1668619768.0605278
2022-11-16 18:29:28.061072
```

On affiche dans un premier temps le temps et en secondes de maintenant à partir du 1er janvier 1970. Puis on affiche la date et l'heure actuelle.



```
# Déclaration de la variable dtnow qui contient la date et l'heure actuelle
dtnow = dt.datetime.fromtimestamp(tm.time())
dtnow
```

On définit une variable ‘dtnow’ qui contient la date et l'heure actuelle.



```
# get year, month, day, etc. from a datetime
# Récupérer l'année, le mois, le jour, l'heure, la minute
# et la seconde de la date et l'heure actuelle
dtnow.year, dtnow.month, dtnow.day, dtnow.hour, dtnow.minute, dtnow.second
```

On récupère l'année, le mois, le jour, l'heure, la minute et la seconde de la date et l'heure actuelle.



```
# Affiche le mois actuel
print(dtnow.month)

# Strftime met le mois actuel en abrégé
dtnow.strftime('%b')
# Affiche le mois actuel en abrégé
print(dtnow.strftime(' %b'))
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
11
Nov
```

On peut afficher le nombre du mois, lorsque l'on affiche ‘dtnow.month’. De plus la méthode strftime permet de donner la valeur du mois pour ‘dtnow’ en lettre et en abrégé.

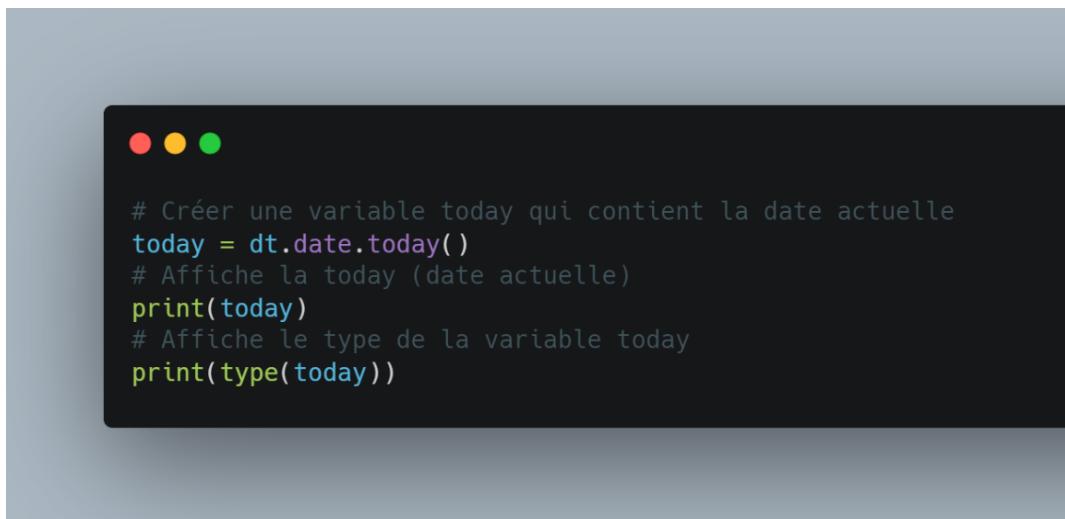


```
# Création d'un objet timedelta qui contient 100 jours
delta = dt.timedelta(days = 100) # create a timedelta of 100 days
# Affiche le type de la variable delta
print(type(delta))
# Affiche le contenu de la variable delta
print(delta)
```

Voici le résultat des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
<class 'datetime.timedelta'>
100 days, 0:00:00
```

On définit une variable delta qui contient 100 jours, puis on affiche le type de la variable delta et on affiche le contenu.



```
# Créer une variable today qui contient la date actuelle
today = dt.date.today()
# Affiche la today (date actuelle)
print(today)
# Affiche le type de la variable today
print(type(today))
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
2022-11-16
<class 'datetime.date'>
```

On crée une variable ‘today’ qui contient la date actuelle (année, mois, jours - au format américain), on affiche son contenu puis son type.

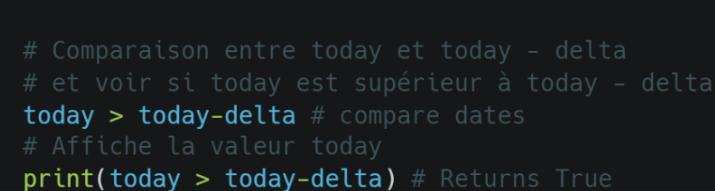


```
# Today prends la valeur de today - delta
today = date.today() - timedelta(days=100)
# Affiche la valeur today
print(today)
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
2022-08-08
```

On affecte une valeur à `today`, qui est sa propre valeur moins la valeur de `delta` (100 jours).
Et on affiche la nouvelle valeur de `today`.



```
# Comparaison entre today et today - delta
# et voir si today est supérieur à today - delta
today > today-delta # compare dates
# Affiche la valeur today
print(today > today-delta) # Returns True
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
True
```

On fait une comparaison entre `today` et `today-delta` pour bien savoir si `today` est bien supérieur que `today-delta`, vu que la nouvelle valeur de `today` est une soustraction avec `delta`.

Puis on affiche le résultat dans un `print` où on a bien la valeur `True`.

4. Compréhensions Lambda et List

```
# Commenter et expliquer le résultat de ce code :  
  
# Déclaration de la fonction my_function qui prends 3 paramètres a, b et c  
# et qui retourne la somme de a et b,  
# le lambda sert à créer une fonction en une seule ligne  
my_function = lambda a, b, c : a + b
```

On crée une fonction ‘my_function’ qui possède une fonction lambda qui permet de créer une fonction sur une seule ligne.

‘my_function’ est composée de 3 variables (a, b et c) et on calcule l’addition entre a et b.

```
# Appel de la fonction my_function avec les paramètres 1, 2 et 3,  
my_function(1,2,3)  
  
# Affiche le résultat = 3  
print(my_function(1,2,3))
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
3
```

On appelle la fonction ‘my_function’ avec les paramètres 1, 2 et 3. Cela va calculer le résultat de la fonction avec les paramètres.

Comme on peut le voir sur le print, où le résultat est 3.

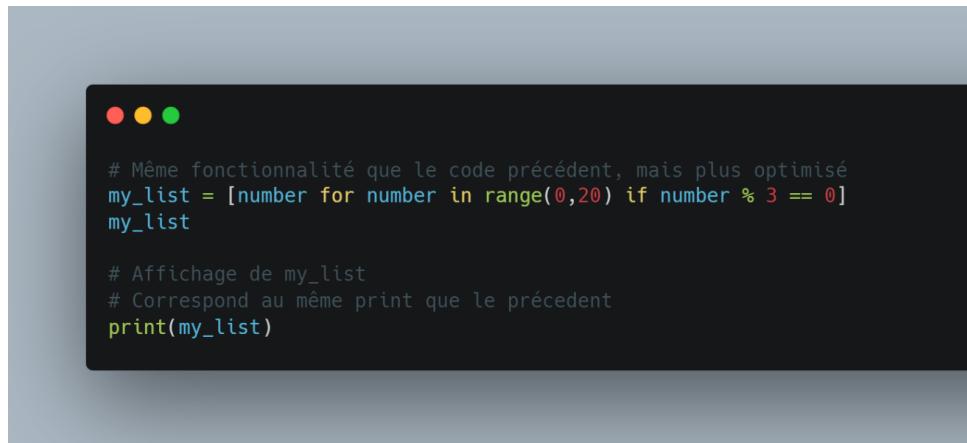


```
# Création d'une liste vide
my_list = []
# Pour chaque nombre de 0 à 20
for number in range(0, 20):
    # si le nombre est divisible par 3
    if number % 3 == 0:
        # ajouter le nombre à la liste my_list
        my_list.append(number)
# affiche la liste my_list avec un message
print('my_list contains the following items:', my_list)
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
my_list contains the following items: [0, 3, 6, 9, 12, 15, 18]
```

On crée une liste vide. En parallèle on crée une boucle entre 0 et 20, et à chaque fois que le nombre est divisible par 3, alors on ajoute ce nombre dans la liste. Puis on affiche cette liste. Les nombres divisibles par 3 entre 0 et 20 sont : 0, 3, 6, 9, 12, 15, 18.



```
# Même fonctionnalité que le code précédent, mais plus optimisé
my_list = [number for number in range(0,20) if number % 3 == 0]
my_list

# Affichage de my_list
# Correspond au même print que le précédent
print(my_list)
```

Voici le résultat du print :

```
[0, 3, 6, 9, 12, 15, 18]
```

Cette nouvelle attribution à `my_list` correspond exactement à la précédente, mais toutes les étapes et les conditions sont condensés directement dans la fonction. Et on obtient le même résultat.

5. Python numérique (NumPy)



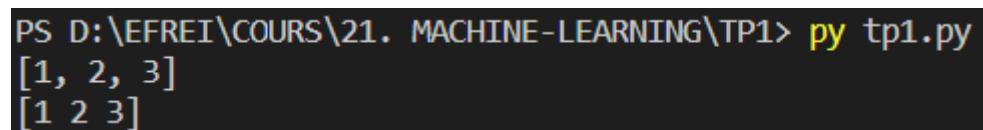
```
# Import de la librairie numpy, avec le nommage np
import numpy as np
```

5.1 Création de tableaux

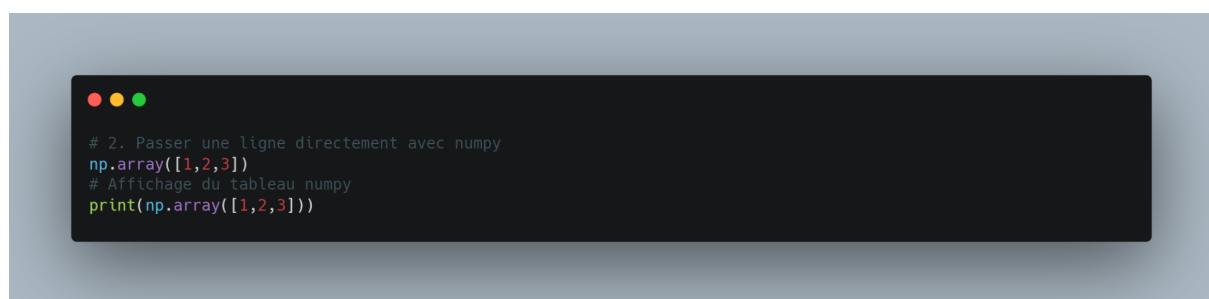


```
# 1. Créer une liste
liste = [1,2,3]
# Afficher la liste
print(liste)
# et la convertir en un tableau numpy.
np.array(liste)
# affichage du tableau numpy
print(np.array(liste))
```

Voici les résultats des prints :

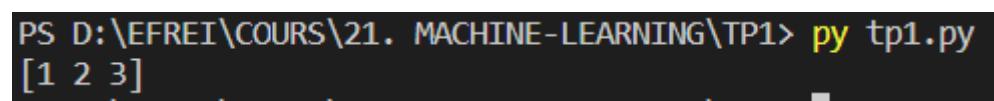


```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[1, 2, 3]
[1 2 3]
```

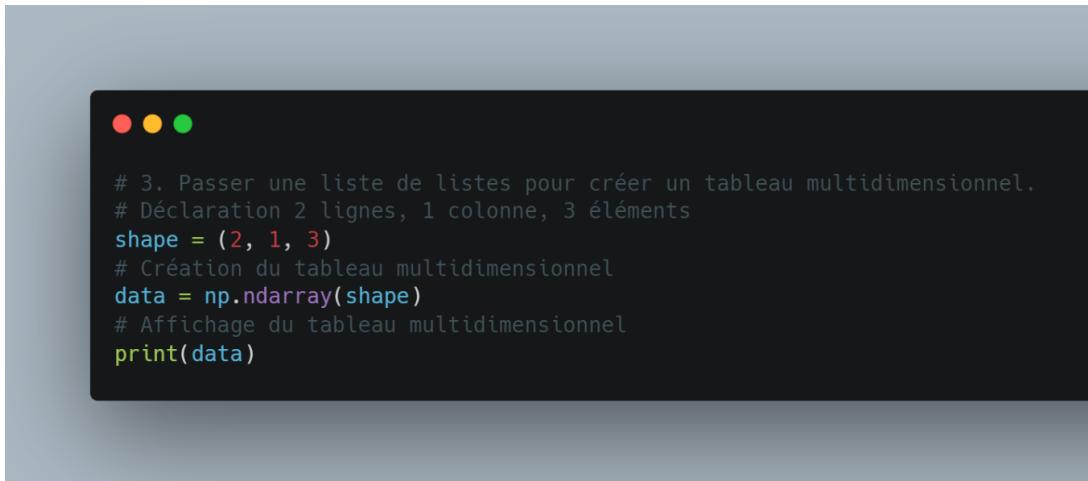


```
# 2. Passer une ligne directement avec numpy
np.array([1,2,3])
# Affichage du tableau numpy
print(np.array([1,2,3]))
```

Voici le résultat du print :



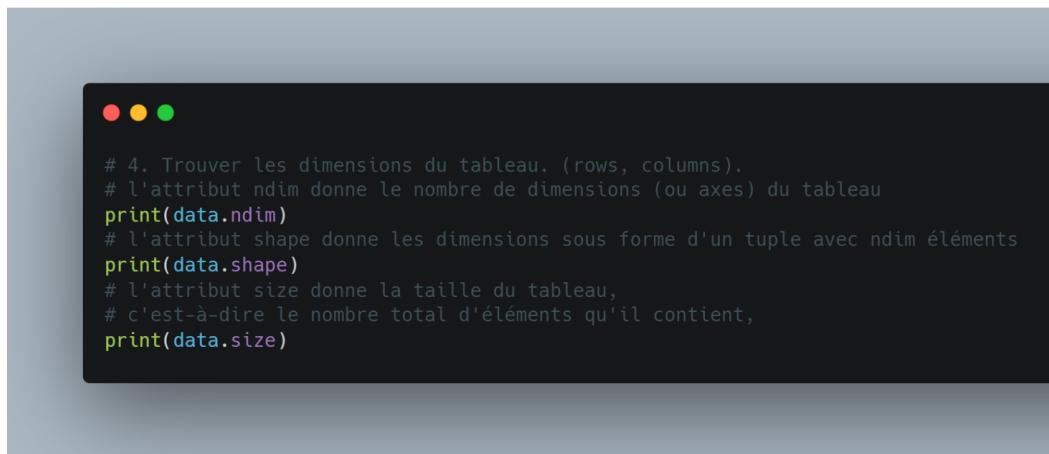
```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[1 2 3]
```



```
# 3. Passer une liste de listes pour créer un tableau multidimensionnel.  
# Déclaration 2 lignes, 1 colonne, 3 éléments  
shape = (2, 1, 3)  
# Création du tableau multidimensionnel  
data = np.ndarray(shape)  
# Affichage du tableau multidimensionnel  
print(data)
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
[[[1.e-323 0.e+000 0.e+000]]  
 [[0.e+000 0.e+000 0.e+000]]]
```



```
# 4. Trouver les dimensions du tableau. (rows, columns).  
# l'attribut ndim donne le nombre de dimensions (ou axes) du tableau  
print(data.ndim)  
# l'attribut shape donne les dimensions sous forme d'un tuple avec ndim éléments  
print(data.shape)  
# l'attribut size donne la taille du tableau,  
# c'est-à-dire le nombre total d'éléments qu'il contient,  
print(data.size)
```

Voici le résultat des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
3  
(2, 1, 3)
```

5. Expliquer le rôle et le résultat obtenu après l'utilisation de ces fonctions la classe numpy :

```
# a. arrange()
# arrange retourne un objet de type numpy.ndarray.
# Création d'un tableau numpy en utilisant arrange
arr1 = np.arange(1,13)
# Afficher le tableau numpy
print(arr1)
# Afficher le type de m
print(type(arr1))
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[ 1  2  3  4  5  6  7  8  9 10 11 12]
<class 'numpy.ndarray'>
```

```
# b. reshape()
# Reshape permet de redimensionner un tableau numpy.
arr2 = np.reshape(arr1,(4,3))
# Afficher le nouveau tableau
print(arr2)
# Afficher le type du nouveau tableau
print(type(arr2))
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
<class 'numpy.ndarray'>
```

```
● ● ●

# c. linspace()
# linspace retourne un tableau numpy avec des éléments répartis uniformément
# dans un intervalle spécifié.
arr3 = np.linspace(0, 10, 5)
# Afficher le tableau numpy
print(arr3)
# Afficher le type du tableau numpy
print(type(arr3))
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[ 0.  2.5  5.  7.5 10. ]
<class 'numpy.ndarray'>
```

```
● ● ●

# d. ones()
# ones retourne un tableau numpy rempli de 1.
arr4 = np.ones((3, 4))
# Afficher le tableau numpy
print(arr4)
# Afficher le type du tableau numpy
print(type(arr4))
```

Voici le résultat des prints :

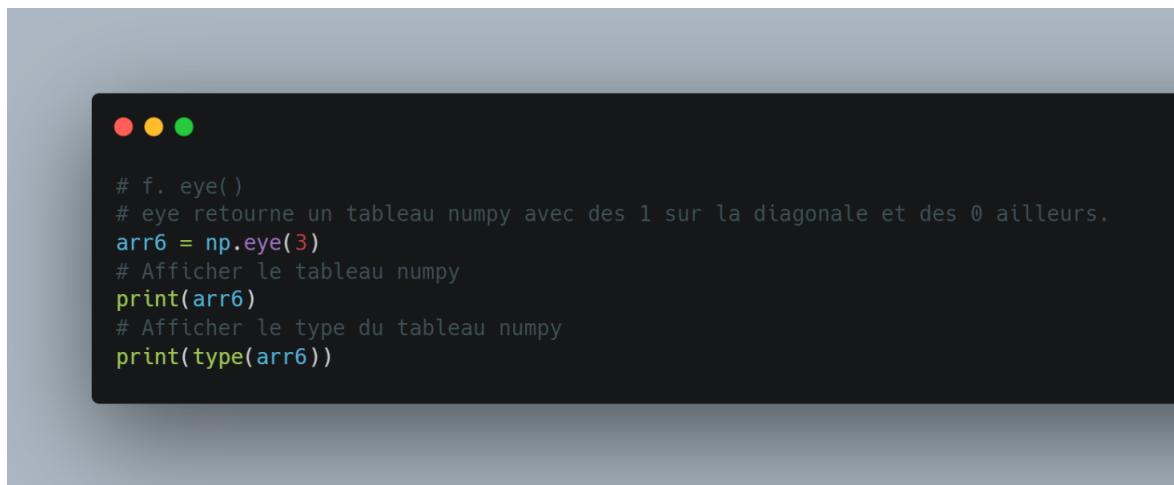
```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[[1.  1.  1.  1.]
 [1.  1.  1.  1.]
 [1.  1.  1.  1.]]
<class 'numpy.ndarray'>
```



```
# e. zeros()
# zeros retourne un tableau numpy rempli de 0.
arr5 = np.zeros((3, 4))
# Afficher le tableau numpy
print(arr5)
# Afficher le type du tableau numpy
print(type(arr5))
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[[0.  0.  0.  0.]
 [0.  0.  0.  0.]
 [0.  0.  0.  0.]]
<class 'numpy.ndarray'>
```

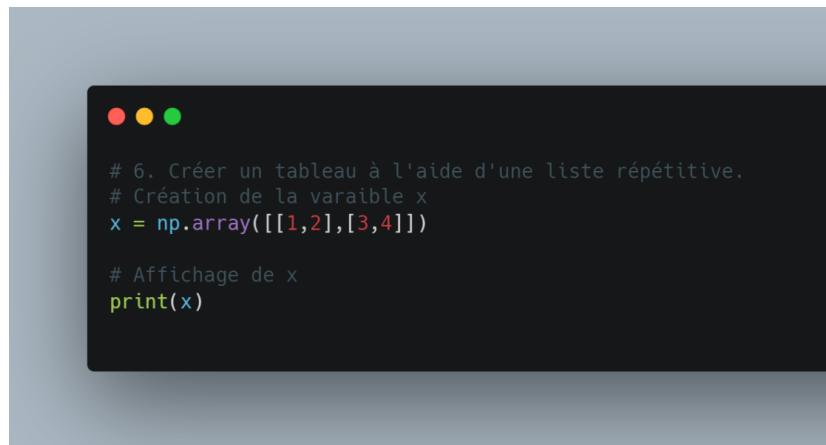


```
# f. eye()
# eye retourne un tableau numpy avec des 1 sur la diagonale et des 0 ailleurs.
arr6 = np.eye(3)
# Afficher le tableau numpy
print(arr6)
# Afficher le type du tableau numpy
print(type(arr6))
```

Voici les résultats des prints :

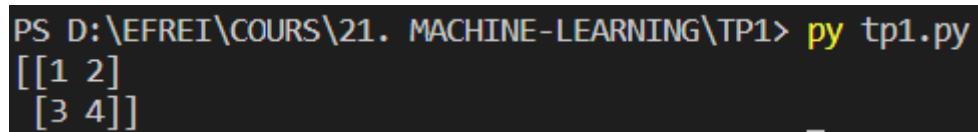
```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[[1.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]]
<class 'numpy.ndarray'>
```

On crée une variable x, avec deux tableaux numpy : [1,2] et [3,4].



```
# 6. Créer un tableau à l'aide d'une liste répétitive.  
# Cr ation de la variable x  
x = np.array([[1,2],[3,4]])  
  
# Affichage de x  
print(x)
```

Voici le r sultat du print :

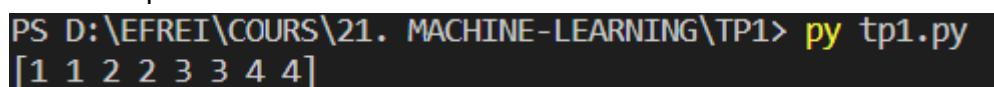


```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
[[1 2]  
 [3 4]]
```



```
# 7. R p te les  l ments d'un tableau en utilisant la fonction .repeat().  
# cr ation de la variable x2  
x2 = np.repeat(x, 2)  
# afficher x2  
print(x2)
```

Voici le r sultat du print :

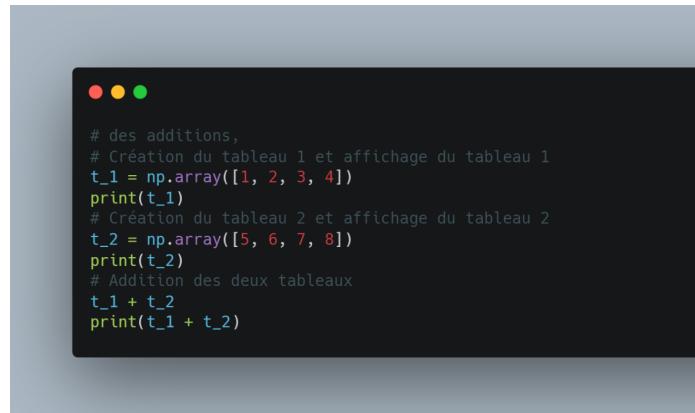


```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
[1 1 2 2 3 3 4 4]
```

5.2 Opérations

1. Utiliser +, -, *, / et ** pour effectuer

des additions,

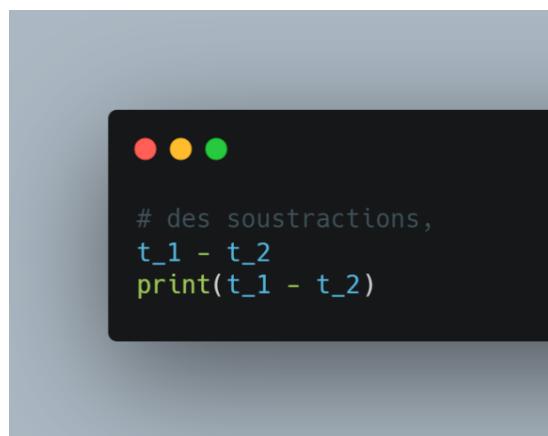


```
# des additions,
# Cr ation du tableau 1 et affichage du tableau 1
t_1 = np.array([1, 2, 3, 4])
print(t_1)
# Cr ation du tableau 2 et affichage du tableau 2
t_2 = np.array([5, 6, 7, 8])
print(t_2)
# Addition des deux tableaux
t_1 + t_2
print(t_1 + t_2)
```

Voici les r sultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[1 2 3 4]
[5 6 7 8]
[ 6  8 10 12]
```

des soustractions,

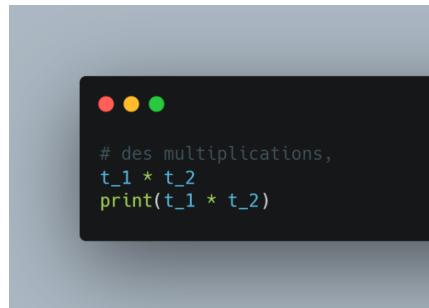


```
# des soustractions,
t_1 - t_2
print(t_1 - t_2)
```

Voici le r sultat des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[-4 -4 -4 -4]
```

des **multiplications**,



```
# des multiplications,
t_1 * t_2
print(t_1 * t_2)
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[ 5 12 21 32]
```

des **divisions**

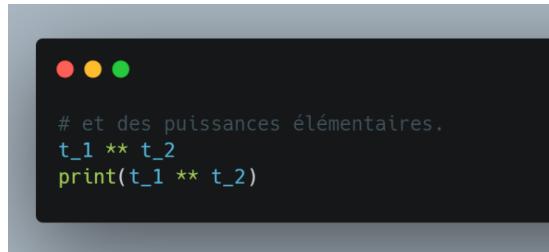


```
# des divisions
t_1 / t_2
print(t_1 / t_2)
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[0.2          0.33333333 0.42857143 0.5         ]
```

et des **puissances élémentaires**.

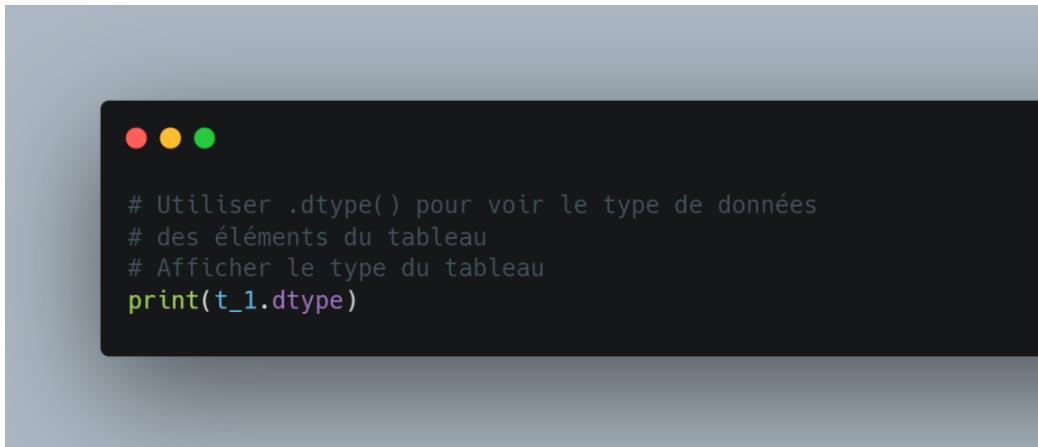


```
# et des puissances élémentaires.
t_1 ** t_2
print(t_1 ** t_2)
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[     1      64   2187 65536]
```

2. Utiliser .dtype() pour voir le type de données des éléments du tableau.

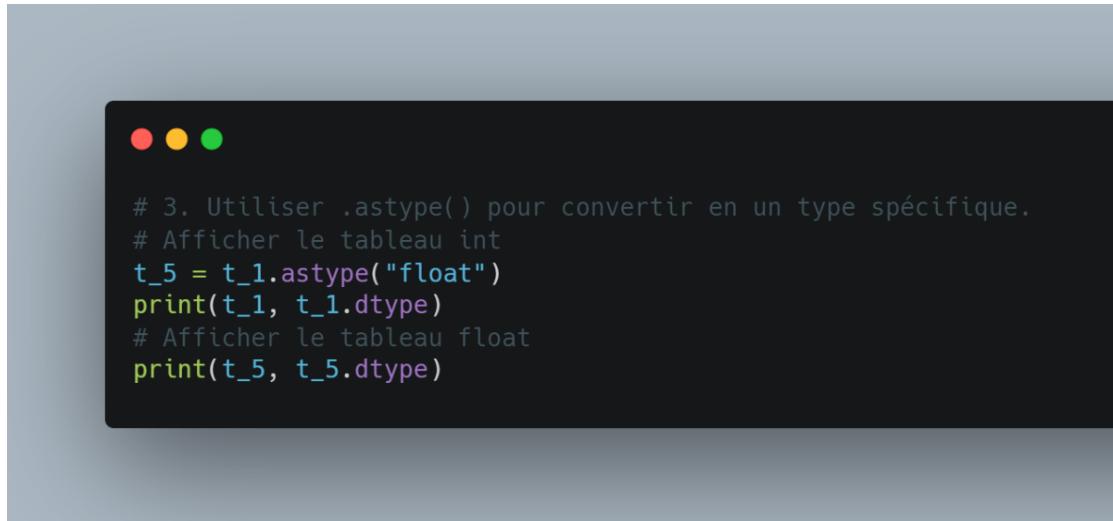


```
# Utiliser .dtype() pour voir le type de données
# des éléments du tableau
# Afficher le type du tableau
print(t_1.dtype)
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
int32
```

3. Utiliser .astype() pour convertir en un type spécifique.



```
# 3. Utiliser .astype() pour convertir en un type spécifique.
# Afficher le tableau int
t_5 = t_1.astype("float")
print(t_1, t_1.dtype)
# Afficher le tableau float
print(t_5, t_5.dtype)
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[1 2 3 4] int32
[1. 2. 3. 4.] float64
```

4.Expliquer le résultat de print(x**2) ; (par exemple : x= [1 2 3])

```
# 4.Expliquer le résultat de print(x**2) ;
# (par exemple : x= [1 2 3] )
# Création d'une liste x
x=[1,2,3]
# Affiche de x avec les valeurs en puissance de 2
print(x**2)
```

Cette opération permet de faire des puissances sur une liste de nombre. Dans notre cas, toutes valeurs vont avoir une puissance de 2 (1^2 , 2^2 , 3^2).

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[1,4,9]
```

5.3 Fonctions mathématiques

Numpy a de nombreuses fonctions mathématiques intégrées qui peuvent être exécutées sur des tableaux.

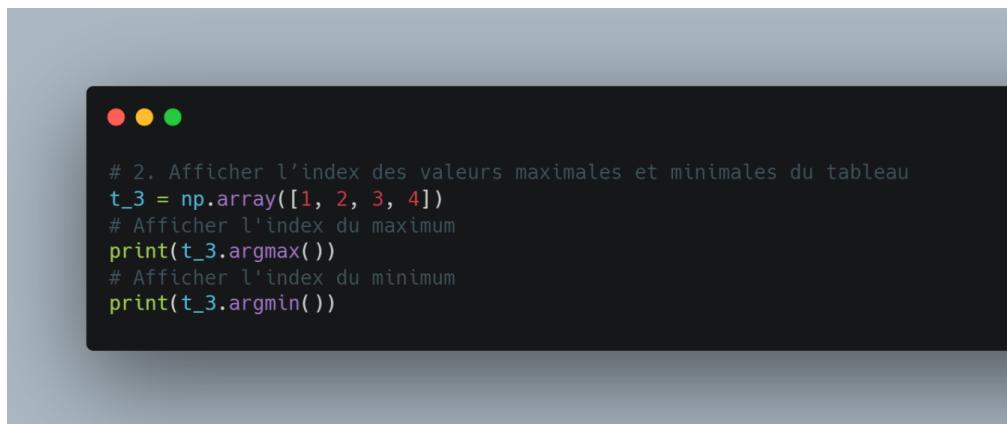
1. Créer un tableau en utilisant numpy, puis calculer la somme, la moyenne, le maximum, le minimum et l'écart-type.

```
## 5.3 Fonctions mathématiques
# 1. Créer un tableau en utilisant numpy
# Création du tableau
t_3 = np.array([1, 2, 3, 4])
# puis calculer la somme
print(t_3.sum())
# la moyenne
print(t_3.mean())
# le maximum
print(t_3.max())
# le minimum
print(t_3.min())
# l'écart-type
print(t_3.std())
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
10
2.5
4
1
1.118033988749895
```

2. Afficher l'index des valeurs maximales et minimales du tableau :



```
# 2. Afficher l'index des valeurs maximales et minimales du tableau
t_3 = np.array([1, 2, 3, 4])
# Afficher l'index du maximum
print(t_3.argmax())
# Afficher l'index du minimum
print(t_3.argmin())
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
3
0
```

L'index '3' correspond à la valeur 4 et l'index '0' correspond à la valeur 1

5.4. Indexation / Tranchage

1. Expliquer le résultat de: s[-5 ::-2]



```
s = np.arange(13)**2
print(s[0], s[4], s[-1])
# 1. Expliquer le résultat de: s[-5 ::-2]
print(s[-5::2])
```

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
0 16 144
[ 64 100 144]
```

Le résultat est le tableau 's' inversé avec un pas de 2.

2. Considérons le tableau multidimensionnel r,

```
# 2. Considérons le tableau multidimensionnel r,
r = np.arange(36)
r.resize((6, 6))
print(r)

# Utilisez la notation entre parenthèses pour trancher :
# `tableau[ligne, colonne]`
r[2,2]
print(r)

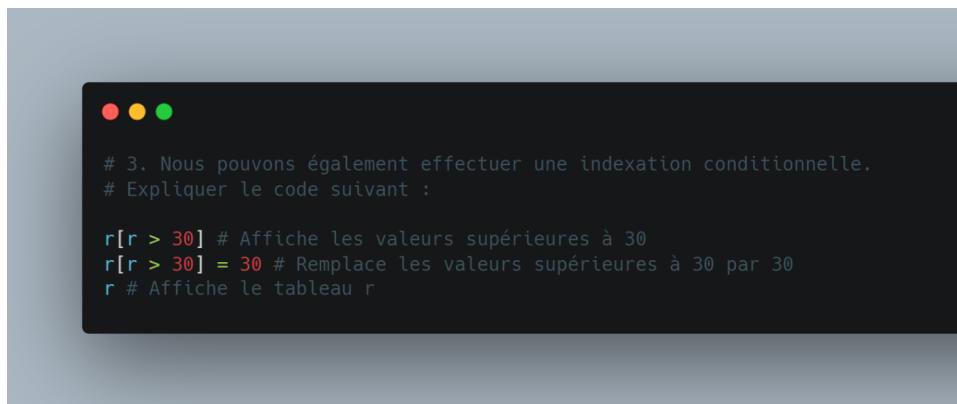
# Et utilisez : pour sélectionner
# une plage de lignes ou de colonnes
r[3, 3:6]
print(r)

# Ici, nous sélectionnons toutes les lignes
# jusqu'à (et n'incluant pas) la ligne 2,
# et toutes les colonnes jusqu'à (et n'incluant pas) la dernière colonne.
r[:2, :-1]
print(r)
```

Voici le résultat des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
```

3. Nous pouvons également effectuer une indexation conditionnelle. Expliquer le code suivant :



Tout d'abord, le 'r[r > 30]' affiche les valeurs supérieures à 30. Puis, avec le 'r[r > 30] = 30' on remplace les valeurs supérieures à 30 par 30. Enfin, on affiche le tableau r.

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 30 30 30 30 30]]
```

5.5. Copie de données

1. Définissez les valeurs de cette tranche sur zéro ([:] sélectionne l'ensemble du tableau)
2. Afficher r, vos remarques ?
3. Ecrire un code qui permet de créer une copie qui n'affectera pas le tableau d'origine.
4. Afficher encore une fois le tableau d'origine r, vos remarques ?

```
r2 = r[:3, :3]
# print(r2)
# 1. Définissez les valeurs de cette tranche sur
# zéro ([:] sélectionne l'ensemble du tableau)
r2[:] = 0
# 2. Afficher r, vos remarques ?
print(r) # r2 est une vue de r, donc r est modifié
# 3. Ecrire un code qui permet de créer une copie qui
# n'affectera pas le tableau d'origine.
r_copy = r.copy()
# 4. Afficher encore une fois le tableau d'origine r, vos remarques ?
print(r) # r n'est pas modifié
```

Tout d'abord, r2 est une vue de r, donc notre print(r) montre bien que r est modifié. Pour cela on crée une copy de r : 'r_copy'. Lorsque l'on affiche 'r_copy', on voit que r n'est pas modifié.

Voici les résultats des prints :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py
[[ 0  0  0  3  4  5]
 [ 0  0  0  9 10 11]
 [ 0  0  0 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 30 30 30 30 30]]
[[ 0  0  0  3  4  5]
 [ 0  0  0  9 10 11]
 [ 0  0  0 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 30 30 30 30 30]]
```

5.5.1. Itérer sur des tableaux

Créons un nouveau tableau 4 par 3 de nombres aléatoires 0-9.

1. Itérer le tableau par ligne.

```
● ● ●  
# 1. Itérer le tableau par ligne.  
for row in test:  
    print(row)
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
[2 6 4]  
[3 3 9]  
[2 3 4]  
[4 6 1]
```

2. Itérer le tableau par index.

```
● ● ●  
# 2. Itérer le tableau par index.  
for i in range(len(test)):  
    print(test[i])
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
[0 1 2]  
[1 2 1]  
[4 7 1]  
[1 5 2]
```

3. Itérer le tableau par ligne et index.

```
● ● ●  
# 3. Itérer le tableau par ligne et index.  
for i, row in enumerate(test):  
    print('row', i, 'is', row)
```

Voici le résultat du print :

```
PS D:\EFREI\COURS\21. MACHINE-LEARNING\TP1> py tp1.py  
row 0 is [3 6 9]  
row 1 is [6 4 3]  
row 2 is [2 8 9]  
row 3 is [4 5 8]
```