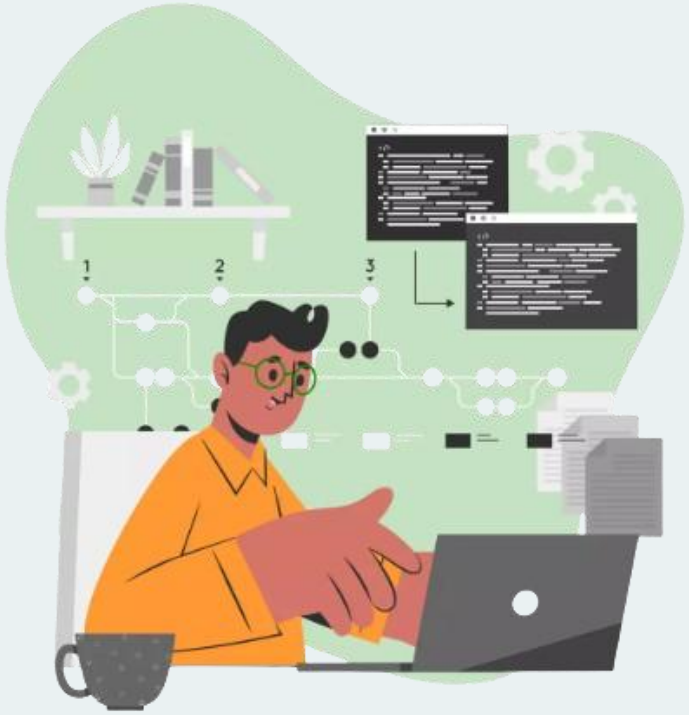
جامعة مولاي إسماعيل
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵎⵓⵍⴰⵢ ⵉⵙⵎⴰⵄⵉⵍ
UNIVERSITÉ MOULAY ISMAÏL



Intelligence Artificielle

Professeur Khalid BENABBES

Pr. BENABBES Khalid
Université Moulay Ismail, Meknès
E-mail: benabbeskhalid@gmail.com
Tél: 0700096627

La science des données (Data Science) et l'intelligence artificielle (IA)

Data Science (Données et analyses)?

-Discipline qui exploite les données pour extraire des informations utiles.

-Explorer, analyser et interpréter des volumes importantes données pour :
+Résoudre des problèmes,
+Guider la prise de décision
+Générer de la valeur.

intelligence artificielle (Modèles et automatisation) ?

-Domaine de l'informatique visant à créer des systèmes capables de simuler l'intelligence humaine.

-Développer des systèmes intelligents capables d'apprendre, raisonner et agir de manière autonome.
(Apprentissage, Raisonnement, Prise de décision**)**

Python ?

-Langage de programmation polyvalent et largement utilisé.

-Outil principal pour implémenter des solutions en science des données et en IA.

Statistique (Science)?

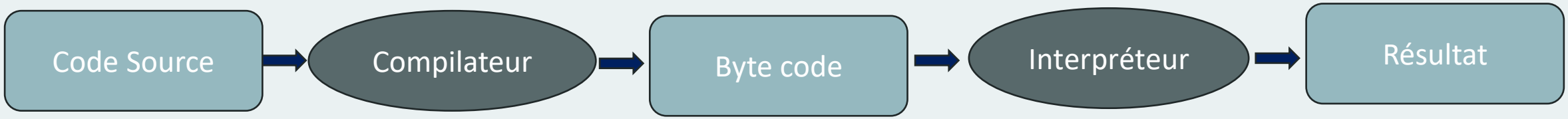
-Fondement mathématique pour l'analyse des données.

Plan

1. Langage Python et outils pour la science des données
2. Collection des données de sources variées
3. Nettoyage et visualisation des données (Préparation des données)
4. Augmentation des données
5. Analyse exploratoire des données (EDA)
6. Analyse statistique et prétraitement
7. Introduction au machine Learning et Deep Learning
8. Les Bases de l'Ingénierie des Données (Data Engineering)
9. Les Enjeux Éthiques en Sciences des Données
10. Déploiement des Modèles de Machine Learning et Deep Learning
11. Projets pratiques de science de données

Python et outils pour la science des données

Les bases de Python



Code Source (.py) → [Compilateur] → Bytecode (.pyc) → [Interpréteur PVM] → Résultat

-Compilé en bytecode puis interprété

Avantages: Interpréteur interactif permettant de tester ,n'importe quel petit bout de code, compilation transparente

Inconvénients: peut être lent

Que peut on faire avec Python?

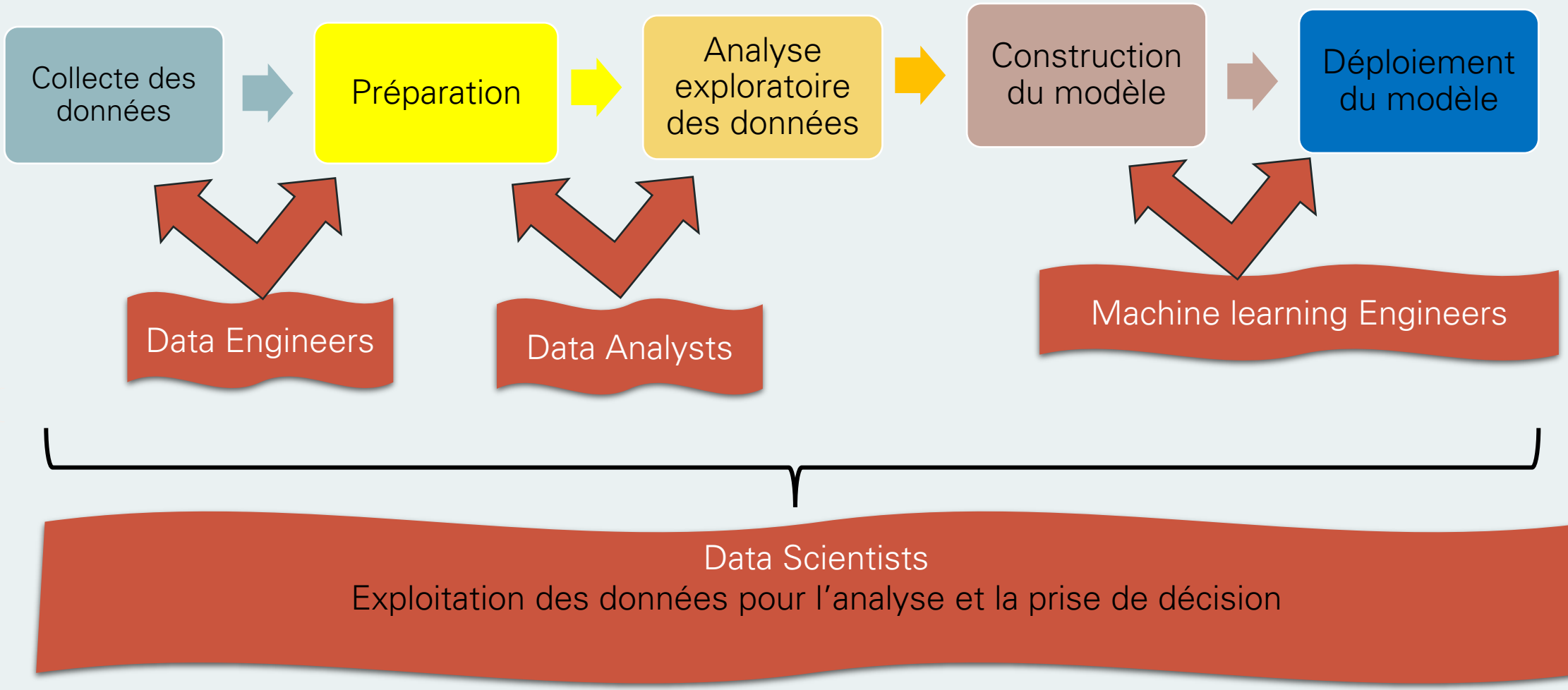
- **Développement web**: Django, Pyramid, Zope, Plone,…
- **Calcul Scientifique**: Numpy, Scipy,sage,..
- **Représentation graphique**: gnuplot, matplotlib, VTk,…
- **GUIs**: TkInter, PyGtk, PyQt,…
- **Manipulation d'images**: Pillow, OpenCv-Pythin,…
- **Base de données**: MySQL, PostgreSQL, Oracle,….

Les étapes clés d'un projet data Science

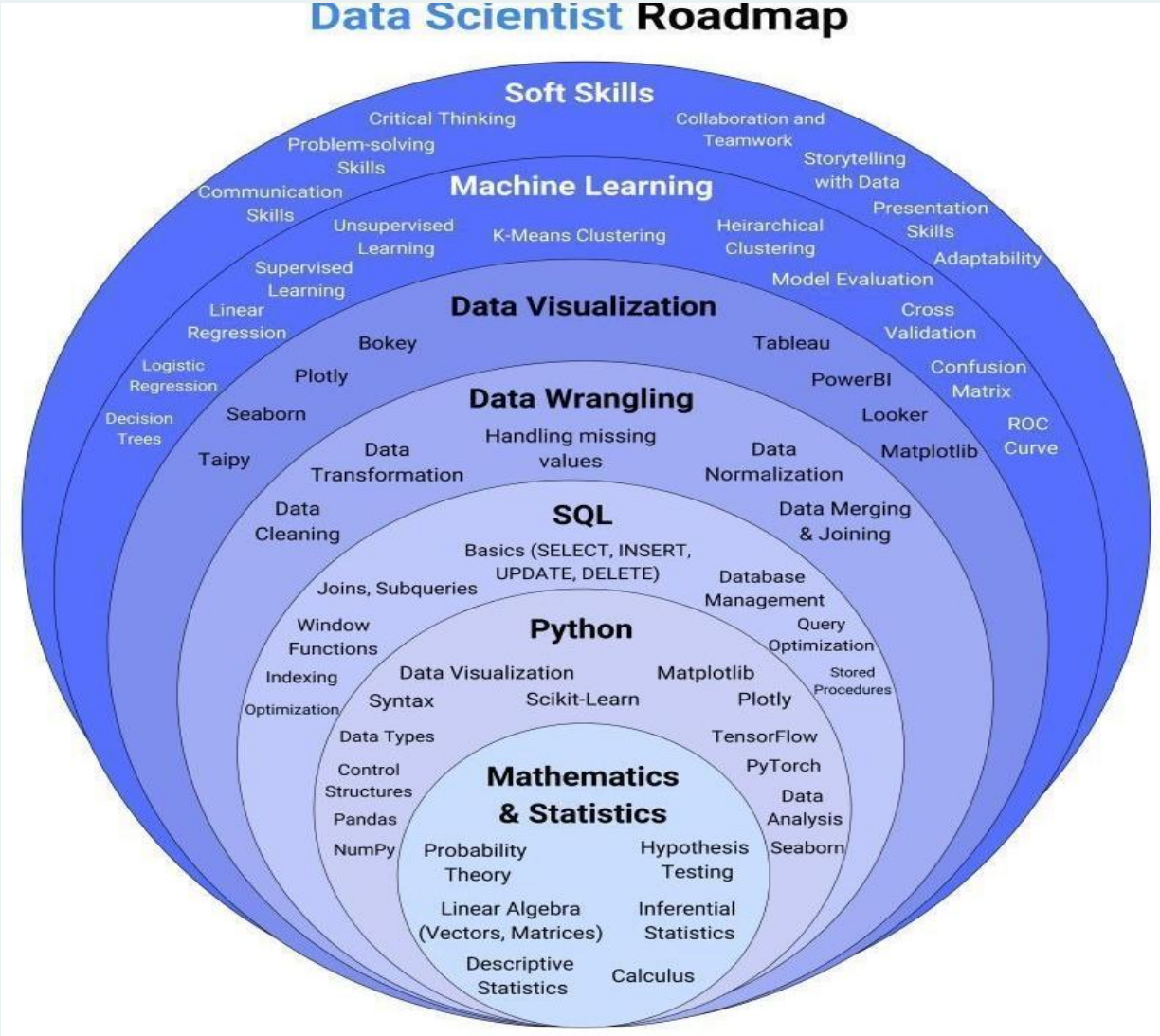
Dresser la problématique par le Data Scientist

- Qui est votre client?
- Qu’est ce que le client vous demande exactement de résoudre?
- Comment traduire leur demande ambiguë en un problème concret et bien défini?

Process du Data Scientist



Process du Data Scientist



Communication des résultats

- **Vous devez toujours expliquer et présenter les informations que vous avez trouvées.**
- **Ou, si votre objectif final est de créer un système (web App par exemple), vous devez généralement partager ce que vous avez développé.**
- **Il existe de nombreuses façons de communiquer vos résultats: rapports, diaporamas ou présentations.**
- **La visualisation des données sera toujours très précieuse.**

Visual Studio Code

Installation de l'environnement (Anaconda, Jupyter, Notebook).

Librairies Python pour la science de données

Manipulation et analyse de données

- ❑ **Pandas**
- ❑ **NumPy**

Visualisation de données

- ❑ **Matplotlib**
- ❑ **Seaborn**
- ❑ **Plotly**

Machine Learning

- ❑ **Scikit-learn**
- ❑ **TensorFlow et PyTorch**

Traitement de données volumineuses

- ❑ **Dask:** Similaire à Pandas.
- ❑ **PySpark:** Moteur de traitement de données distribué

Traitement de texte et NLP

- ❑ **NLTK**
- ❑ **SpaCy**

Les bases de Python

Types de données (Data Types)

- **Nombres (Numbers) :** Entiers (int), nombres flottants (float), nombres complexes (complex).
- **Chaînes de caractères (Strings) :** Séquences de caractères, comme "Bonjour".
- **Listes (Lists) :** Collections ordonnées et modifiables, comme [1, 2, 3].
- **Dictionnaires (Dictionaries) :** Collections de paires clé-valeur, comme {"nom": "Alice", "âge": 25}.
- **Tuples (Tuples) :** Collections ordonnées et immuables, comme (1, 2, 3).
- **Ensembles (Sets) :** Collections non ordonnées et sans doublons, comme {1, 2, 3}.
- **Booléens (Booleans) :** Valeurs True ou False.

Les bases de Python

Opérateurs de comparaison (Comparison Operators):

- == : égal à
- != : différent de
- > : supérieur à
- < : inférieur à
- >= : supérieur ou égal à
- <= : inférieur ou égal à

Conditions :

- if, elif, else

```
age = 18
If( age >= 18):
    print("Majeur")
else:
    print("Mineur")
```

Conditions :

- if, elif, else

```
age = 18
if age <= 18:
    print("Majeur")
elif age <= 13:
    print("Majeur")
else:
    print("Mineur")
```

Boucles :

- For

```
for i in range(3):
    print(i)  # Affiche 0, 1, 2
```

- range()

```
for i in range(2, 5):
    print(i)  # Affiche 2, 3, 4
```

- While

```
compteur = 0
while compteur < 3:
    print(compteur)
    compteur += 1
```

Les bases de Python

List compréhension:

```
carres = [x**2 for x in range(5)]
print(carres)  # Affiche [0, 1, 4, 9, 16]

carres2 = [x**2 for x in range(5) if x%2==0]
print(carres)  # Affiche [0, 1, 4, 9, 16]
```

Fonctions (Functions)

```
def dire_bonjour(nom):
    return f"Bonjour, {nom}!"

print(dire_bonjour("Alice"))  # Affiche "Bonjour, Alice!"
```

Expressions lambda (Lambda Expressions)

```
carre = lambda x: x**2
print(carre(5))  # Affiche 25
```

map() et filter() :

```
nombres = [1, 2, 3, 4]
carres = list(map(lambda x: x**2, nombres))  # [1, 4, 9, 16]
pairs = list(filter(lambda x: x % 2 == 0, nombres))  # [2, 4]
```

Les méthodes associées à des objets:

```
ma_liste = [1, 2, 3]
ma_liste.append(4)  # Ajoute 4 à la liste
print(ma_liste)     # Affiche [1, 2, 3, 4]
ma_liste.pop() #supprimer le dernier élément de la liste
print(ma_liste)     # Affiche [1, 2, 3]
ma_liste.insert(i, x) # Ajouter un élement 4 à  la position 1
print(ma_liste)     # Affiche [1, 4, 2, 3]
ma_liste.extend([5, 6, 7]) #Ajout de plusieurs éléments
print(ma_liste) # [1, 99, 2, 3, 4, 5, 6, 7]
```

Les méthodes associées à des objets:

- ma_liste = [10, 20, 30, 40, 50] #Supprimer un élément spécifique (par valeur)
- ma_liste.remove(30)
- print(ma_liste) # [10, 20, 40, 50]
- element_supprime = ma_liste.pop(1) # Supprimer un élément par son index (ex: 2ème élément)
- print(ma_liste) #[10, 40, 50]
- print("Élément supprimé :", element_supprime) #20
- del ma_liste[0] #Supprime le premier élément
- del ma_liste
- print(ma_liste) # [40, 50]
- ma_liste.clear() #Supprimer tous les éléments
- print(ma_liste) # []

Différences entre array, liste, tuple, dict et set en Python

`list`	Collection ordonnée d'éléments	✅ Oui	✅ Oui	✅ Oui	✅ Indexation rapide	`[1, 2, 3]`
`tuple`	Comme `list`, mais **immuable**	❌ Non	✅ Oui	✅ Oui	✅ Indexation rapide	`(1, 2, 3)`
`set`	Collection **non ordonnée** d'éléments uniques	✅ Oui	❌ Non	❌ Non (uniques)	❌ Pas d'indexation	`{1, 2, 3}`
`dict`	Stocke des paires **clé-valeur**	✅ Oui	✅ Oui (Python 3.7+)	❌ Clés uniques	✅ Accès rapide par clé	`{"nom": "Alice", "age": 25}`

Les chaînes de caractères

- Exemples avec s = "Bonjour"

Obtenir la longueur d'une chaîne (= le nombre de caractères)	len(s) => 7
Obtenir un caractère de la chaîne	s[0] => "B" s[-1] => "r"
Obtenir une partie de la chaîne	s[0:3] => "Bon"
Concaténer deux chaînes	"Hi " + s
Rechercher si une chaîne est incluse dans une autre	s.find("jour") => 3 # Trouvé en position 3 (-1 si pas trouvé)

Les chaînes de caractères

Exemple: affichage total des caractères d'une chaîne via la méthode d'itérateur

```
s = "Python"
for x in s:
    print(x)
# affiche :
"""
P
y
t
h
o
n
"""
```

Les chaînes de caractères

▪ Opération sur les chaînes de caractères

➞ Concaténation de deux chaînes de caractères

Pour faire la concaténation de deux chaînes de caractères, on utilise l'opérateur '+' :

Example

```
s1 = "Learn "
s2 = "Python"
# concaténation de s1 et s2
s = s1 + s2
print(s) # affiche : 'Learn Python'
```

➞ Extraire une sous chaîne de caractères

On extrait une sous chaîne de s depuis **la i ème** position, jusqu'à la **j ème** non incluse en utilisant la syntaxe :

```
substring = string[i : j]
```

Exemple.

```
s = "Python"
substring = s[2 : 5]
print(substring) # affiche : 'tho'
```

Les chaînes de caractères

- ## Les fonctions de chaînes de caractères en Python

Exemple. transformation d'une chaîne en minuscule

```
1 s="CRMEF OUJDA"
2 s = s.lower()
3 print(s) # affiche crmef oujda
```

Exemple. remplacement d'une occurrence par une autre

```
1 s="CRMEF OUJDA"
2 s = s.replace("CRMEF","ENS")
3 print(s) # affiche ENS OUJDA
```

Exemple. Nombre de caractères d'une chaîne

```
1 #-*- coding : utf-8 -*-
2 s = "CRMEF OUJDA"
3 n = len(s)
4 print("le nombe de caractères de la chaîne s est : " ,
      n)
5 # affiche le nombre de caractères de la chaîne s est     :
       11
```

Les chaînes de caractères

- ## Les fonctions de chaînes de caractères en Python

Exemple. String.format

```
1 nom = "David"
2 age = 37
3 s = 'Bonjour , {}, vous avez {} ans'.format(nom,age)
4 print(s) # affiche 'Bonjour, David, vous avez 37 ans'
```

Exemple. extraire une sous chaîne

```
1 s = "CRMEF OUJDA"
2 s1 = s[6 :9]
3 print(s1)     # affiche OUJ
4 s2 = s[6 :]
5 print(s2)     # affiche OUJDA
6 s3 = s[ :4]
7 print(s3)     # affiche CRME
```

La gestion du tuple

- Les **tuples** correspondent aux listes à la différence qu'ils sont **non modifiables**. Ils utilisent les **parenthèses** au lieu des crochets :

```
Entrée [1]: x=(1,2,3)
            x
Out[1]: (1, 2, 3)

Entrée [2]: x[2]
Out[2]: 3

Entrée [3]: x[0:2]
Out[3]: (1, 2)

Entrée [4]: x[2]=5
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[4], line 1
----> 1 x[2]=5

TypeError: 'tuple' object does not support item assignment
```

La gestion du tuple

- **Création d'un tuple en utilisant le constructeur tuple()**

Il existe une autre méthode pour créer un tuple qui consiste à utiliser le constructeur **Tuple()**

Exemple. Création d'un tuple en utilisant le constructeur tuple() :

```
1 myTuple = tuple (("cartable", "cahier", "livre"))
2 # notez les doubles parenthèses rondes
3 print(myTuple)
```

La gestion du tuple

Boucle à travers un tuple

Vous pouvez parcourir les éléments d'un tuple en utilisant une boucle for.

Exemple. Parcourez les éléments et imprimez les valeurs :

```
myTuple = ("cartable", "cahier", "livre")
for x in myTuple:
    print (x)
# Affiche tous les éléments du tuple.
```

Vérifier si un élément existe dans un tuple

Pour déterminer si un élément spécifié est présent dans un tuple, utilisez le mot-clé in

Exemple. Vérifiez si **"cartable"** est présent dans le tuple :

```
myTuple = ("cartable", "cahier", "livre")
if("cartable" in myTuple):
    print("Oui, 'cartable' est dans myTuple")
```

Longueur d'un tuple: on utilise la méthode **len() :**

Exemple. nombre d'éléments d'un tuple :

```
myTuple = ("cartable", "cahier", "livre")
print(len(myTuple))
# Affiche 3
```

- **La gestion du tuple**

➞ Ajout ou suppression d'éléments impossible à un tuple

Exemple. Ajout d'éléments **impossible** à un tuple :

```
1 myTuple = ("cartable", "cahier", "livre")
2 myTuple [3] = "Stylo" # Ceci provoquera une erreur !
```

➞ Suppression d'un tuple

Les tuples ne sont pas modifiables, vous ne pouvez donc pas en supprimer d'éléments, mais vous pouvez supprimer complètement le tuple à l'aide du mot clé **del**

Exemple. Supprimer complètement un tuple :

```
1 myTuple = ("cartable", "cahier", "livre")
2 del myTuple
3 print(myTuple) #cela générera une erreur car le tuple n'existe plus
```

- **La gestion de l'ensemble (Set)**

➞ **Définir des ensembles en Python**

Un ensemble en Python (Python set) est une collection non ordonnée et non indexée. En Python, les ensembles sont écrits avec des accolades.

Exemple. Création d'un ensemble :

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 print(mySet)
```

Remarque 6*:* Les ensembles ne sont pas ordonnés, les éléments apparaîtront donc dans un ordre aléatoire.

- **La gestion de l'ensemble (Set)**

➞ **Définir des ensembles en Python**

Un ensemble en Python (Python set) est une collection non ordonnée et non indexée. En Python, les ensembles sont écrits avec des accolades.

Exemple. Création d'un ensemble :

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 print(mySet)
```

Remarque 6*:* Les ensembles ne sont pas ordonnés, les éléments apparaîtront donc dans un ordre aléatoire.

- **Les ensembles en Python (Python Set)**

  → **Accès aux éléments d'un ensemble Python**

Vous ne pouvez pas **accéder** aux éléments d'un ensemble en faisant référence à un **index**, car les ensembles ne sont pas **ordonnés**, les éléments n'ont pas d'index. Mais vous pouvez **parcourir** les éléments de l'ensemble à l'aide d'une **boucle for** ou demander si une valeur spécifiée est présente dans un ensemble à l'aide du mot **clé in**.

Exemple. Affichage des éléments d'un ensemble

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 for x in mySet:
3       print(x)
```

Exemple. vérification d'appartenance d'un élément

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 print("Crayon" in mySet) # affiche : True
3 print("Cahier" in mySet) # affiche : False
```

- **Les ensembles en Python (Python Set)**

  → **Longueur ou cardinal d'un ensemble Python**

  Pour connaitre la longueur (cardinal) d'un ensemble Python, on utilise la méthode **len()** Exemple. longueur d'un ensemble python

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 cardinal = len(mySet)
3 print("card(mySet) = ", cardinal)
4 # affiche card(mySet) = 3
```

- **Les ensembles en Python (Python Set)**

  → **Ajouter un ou plusieurs éléments à un ensemble Python**

  - Pour ajoutez un élément à un ensemble Python, on utilise la méthode add() :
  Exemple. Ajout d'un élément à l'ensemble

Exemple. Ajout d'un élément à l'ensemble

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 mySet.add ("Cahier")
3 print(mySet)
```

Exemple. ajouter plusieurs éléments

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 mySet.update (["Cahier", "Cartable", "Trousse"])
3 print(mySet)
```

- **Les ensembles en Python (Python Set)**

  → **Supprimer un élément d'un ensemble Python**

Pour supprimer un élément d'un ensemble Python, deux choix s'offrent à vous:
la méthode **remove()** ou la méthode **discard()**

Exemple. supprimer "Crayon" par la méthode **remove()**

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 mySet.remove("Crayon")
3 print(mySet) # affiche {'Gomme', 'Stylo'}
```

Remarque : Si l'élément à supprimer n'existe pas, **remove()** générera une erreur.

Exemple. supprimer **"Crayon"** par la méthode **discard() :**

```
1 mySet = {"Stylo", "Crayon", "Gomme"}
2 mySet.discard("Crayon")
3 print(mySet) # affiche {'Gomme', 'Stylo'}
```

Remarque 9. Vous pouvez également utiliser la méthode **pop()** pour supprimer un élément, mais cette méthode supprimera le dernier élément. Rappelez-vous que les ensembles ne sont pas ordonnés et vous ne saurez pas quel élément sera supprimé.

Les ensembles en Python (Python Set)

Vider un ensemble Python

Pour vider ensemble Python, on se sert de la méthode clear()

Exemple: vider un ensemble python

```
mySet = {"Stylo", "Crayon", "Gomme"}
mySet.clear()
print(mySet) # affiche set{} qui veut dire un ensemble vide
```

Supprimer un ensemble

Pour supprimer un ensemble Python, on utilise la commande **del**

Exemple. Supprimer un ensemble

```
mySet = {"Stylo", "Crayon", "Gomme"}
del mySet
print(mySet)
# affiche le message d'erreur: builtins.NameError: name 'mySet' is not defined
```

Remarque 9. Vous pouvez également utiliser la méthode **pop()** pour supprimer un élément, mais cette méthode supprimera le dernier élément. Rappelez-vous que les ensembles ne sont pas ordonnés et vous ne saurez pas quel élément sera supprimé.

- **Les ensembles en Python**

  → **Récapitulatif des méthodes associées à un ensemble Python**

1. **add() :** ajoute un élément à l'ensemble
2. **clear()** : supprime tous les éléments de l'ensemble
3. **copy()** : retourne une copie de l'ensemble
4. **difference ()** : retourne un ensemble contenant la différence entre deux ensembles ou plus.
5. **difference_update() :** supprime les éléments de cet ensemble qui sont également inclus dans un autre ensemble spécifié
6. **discard() :** supprimer l'élément spécifié
7. **intersection() :** retourne un ensemble, qui est l'intersection de deux autres ensemble
8. **intersection_update() :** supprime les éléments de cet ensemble qui ne sont pas pré- sents dans d'autres ensembles spécifiés.
9. **isdisjoint() :** indique si deux ensembles ont une intersection ou non.
10. **issubset() :** indique si un autre jeu contient ce jeu ou non.
11. **issuperset() :** indique si cet ensemble contient un autre ensemble ou non.
12. **pop() :** supprime un élément de l'ensemble
13. **remove() :** supprime l'élément spécifié
14. **symmetric_difference() :** retourne un ensemble avec les différences symétriques de deux ensembles

La gestion du dictionnaire

- Les dictionnaires sont des collections non ordonnées d'objets, c-à-d qu'il n'y a pas de notion d'ordre (i.e. pas d'indice). On accède aux valeurs d'un dictionnaire par des clés.
- **Exemples:**

```
a={}  # Créer un dictionnaire vide a
b=dict () # Créer aussi un dictionnaire vide b
a['nom'] = ' girafe' # ajouter la clé 'nom' avec la valeur 'girafe'
a['taille'] = 5.0 # ajouter la clé 'taille' avec la valeur 5.0
a['poids'] = 1100 # ajouter la clé 'poids' avec la valeur 1100
print(a)
print(a['taille'])
```

```
{'nom': ' girafe', 'taille': 5.0, 'poids': 1100}
5.0
```

La gestion du dictionnaire

- Les dictionnaires sont des collections non ordonnées d'objets, c-à-d qu'il n'y a pas de notion d'ordre (i.e. pas d'indice). On accède aux valeurs d'un dictionnaire par des clés.
- **Exemples:**

```
a={}  # Créer un dictionnaire vide a
b=dict () # Créer aussi un dictionnaire vide b
a['nom'] = ' girafe' # ajouter la clé 'nom' avec la valeur 'girafe'
a['taille'] = 5.0 # ajouter la clé 'taille' avec la valeur 5.0
a['poids'] = 1100 # ajouter la clé 'poids' avec la valeur 1100
print(a)
print(a['taille'])
```

```
{'nom': ' girafe', 'taille': 5.0, 'poids': 1100}
5.0
```

La gestion du dictionnaire

- **Manipulation des dictionnaires**
  - **Opérations sur les dictionnaires**

Opération	signification
x in d	Vrai si x est une des clés de d
x not in d	Réciproque de la ligne précédente
d[i]	Retourne l'élément associé à la clé i
len(d)	Nombre d'éléments de d
min(d)	La plus petite clé
max(d)	La plus grande clé
del d[i]	Supprime l'élément associé à la clé i
list(d)	Retourne une liste contenant toutes les clés du dictionnaire d.
dict(x)	Convertit x en un dictionnaire si cela est possible

La gestion du dictionnaire

- **Manipulation des dictionnaires**
  - **Méthodes prédéfinies de manipulation des dictionnaires**

Méthode	signification
d.copy()	Retourne une copie de d.
d.get(k[, x])	Retourne d[k] si la clé k existe. Sinon la valeur None est retournée à moins que le paramètre optionnel x soit renseigné ; auquel cas, c'est ce paramètre qui sera retourné.
d.clear()	Supprime tous les éléments du dictionnaire.
d.update(d1)	Pour chaque clé k de d, d[k] = d1[k]
d.setdefault(k[, x])	Retourne d[k] si la clé k existe, sinon, affecte x à d[k].
d.popitem()	Retourne un élément et le supprime du dictionnaire.

La gestion du dictionnaire

- **Manipulation des dictionnaires**
  - **Méthodes prédéfinies de manipulation des dictionnaires**

Méthode	signification
d.copy()	Retourne une copie de d.
d.get(k[, x])	Retourne d[k] si la clé k existe. Sinon la valeur None est retournée à moins que le paramètre optionnel x soit renseigné ; auquel cas, c'est ce paramètre qui sera retourné.
d.clear()	Supprime tous les éléments du dictionnaire.
d.update(d1)	Pour chaque clé k de d, d[k] = d1[k]
d.setdefault(k[, x])	Retourne d[k] si la clé k existe, sinon, affecte x à d[k].
d.popitem()	Retourne un élément et le supprime du dictionnaire.

La gestion du dictionnaire

Manipulation des dictionnaires

➔Les méthodes keys( ), values( ) et items( )

- La méthode **keys()** renvoie la liste des clés d'un dictionnaire
- La méthode **values()** renvoie la liste des valeurs d'un dictionnaire
- La méthode **items()** renvoie la liste des tuples (clé, valeur) d'un dictionnaire

```
a={'nom':'girafe', 'taille':5.0, 'poids':1100}  # Créer un dictionnaire a
print(a.keys())  # la liste des clés d'un dictionnaire
print(a.values())  # la liste des valeurs d'un dictionnaire
print(a.items())  # la liste des tuples (clé, valeur) d'un dictionnaire
```

```
dict_keys(['nom', 'taille', 'poids'])
dict_values(['girafe', 5.0, 1100])
dict_items([('nom', 'girafe'), ('taille', 5.0), ('poids', 1100)])
```

Les fonctions définies par l'utilisateur

- **Syntaxe:**

```
def nom_fonction(parametres):
    # instructions
```

- **Exemple:** Fonction qui permet de calculer la somme de deux nombres x et y

```
#définition de la fonction
def somme(x, y):
    S=x+y
    return S
#appel de la fonction
a=3
b=5
s=somme(a,b)
print("la somme de ", a, "et", b, "est: ", s)
```

```
la somme de  3 et 5 est:  8
```

Démonstration:

manipulation de données simples

Manipulation des données avec NumPy
Manipulation des données avec Pandas
Visualisation des données avec Matplotlib et Seaborn

Manipulation des données avec NumPy

- NumPy (ou Numpy) est une bibliothèque d'algèbre linéaire pour Python. Elle est essentielle en **science des données avec Python**, car presque toutes les bibliothèques de l'écosystème **PyData** reposent sur NumPy comme l'un de leurs principaux composants.
- Fournit des tableaux multidimensionnels (ndarray) et des fonctions pour les manipuler.
- **Performances optimisées** : Les opérations sont exécutées en C, ce qui les rend très rapides.
- **Opérations vectorisées** : Permet d'éviter les boucles explicites en Python, ce qui améliore l'efficacité.
- **Base de nombreuses bibliothèques** : Pandas, SciPy, Scikit-learn, etc., reposent sur NumPy.
- **Génération des nombres aléatoires**
- **Indexing and slicing (Indexation et découpage)**
- **Efficacité et performance de la mémoire**
- **des aspects les plus importants de Numpy :** les vecteurs, les tableaux, les matrices et la génération de nombres.

Manipulation des données avec NumPy

Si vous avez **Anaconda**, installez **NumPy** en ouvrant votre terminal ou invite de commande et en tapant :
#conda install numpy

Une fois NumPy installé, vous pouvez l'importer en tant que bibliothèque :
#import numpy as np

NB: Les tableaux Numpy sont essentiellement de deux types : les vecteurs et les matrices.
-Les vecteurs sont strictement des tableaux à 1 dimension et les matrices sont à 2 dimensions (mais vous devez noter qu'une matrice peut toujours avoir une seule ligne ou une seule colonne).

1. Création de tableaux

- **Tableau 1D**

```
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

- **Tableau 2D**

```
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
print(arr_2d)
```

- **Tableaux avec des valeurs prédéfinies :**

```
zeros = np.zeros((3, 3))  # Tableau de zéros
ones = np.ones((2, 2))    # Tableau de uns
identity = np.eye(3)      # Matrice identité
```

2. Manipulation des tableaux

- **Accéder aux éléments** :

```
print(arr[0])        # Premier élément d'un tableau 1D
print(arr_2d[1, 2])  # Élément à la 2ème ligne, 3ème colonne
```

- **Modifier les éléments**

```
arr[0] = 10
arr_2d[1, 2] = 99
```

- **Taille et forme :**

```
print(arr.shape)      # Forme du tableau
print(arr_2d.ndim)    # Nombre de dimensions
print(arr.size)       # Nombre total d'éléments
```

- **Redimensionner un tableau :**

```
arr_reshaped = arr.reshape(5, 1)  # Redimensionne en 5 lignes, 1 colonne
```

Manipulation des données avec NumPy

3. Opérations mathématiques

- **Opérations de base :**

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(a + b)  # Addition élément par élément
print(a * b)  # Multiplication élément par élément
print(np.dot(a, b))  # Produit scalaire
```

- **Fonctions mathématiques :**

```
print(np.sqrt(a))     # Racine carrée
print(np.exp(a))      # Exponentielle
print(np.sin(a))      # Sinus
```

3. Agrégation

- **Somme, moyenne, min, max :**

```
print(np.sum(arr))      # Somme des éléments
print(np.mean(arr))     # Moyenne
print(np.min(arr))      # Valeur minimale
print(np.max(arr))      # Valeur maximale
```

- **Agrégation par axe :**

```
print(np.sum(arr_2d, axis=0))  # Somme par colonne
print(np.sum(arr_2d, axis=1))  # Somme par ligne
```

4. Indexation et découpage (slicing)

- **Indexation :**

```
print(arr[1:4])  # Éléments de l'indice 1 à 3
```

- **Découpage 2D :**

```
print(arr_2d[:, 1])  # Toutes les lignes, 2ème colonne
print(arr_2d[1, :])  # 2ème ligne, toutes les colonnes
```

Statistiques descriptives

Les statistiques descriptives sont essentielles pour comprendre et analyser des données.

1. Mesures de tendance centrale

-Ces mesures indiquent où se situe le centre des données**.**

- **Moyenne:** La moyenne est la somme des valeurs divisée par le nombre total de valeurs.
  -Elle est sensible aux valeurs extrêmes.

```
import numpy as np

data = [10, 20, 30, 40, 50]
moyenne = np.mean(data)
print("Moyenne :", moyenne)
```

- **Médiane:** La médiane est la valeur centrale d'un ensemble de données triées.
  -Elle est moins sensible aux valeurs extrêmes.

```
mediane = np.median(data)
print("Médiane :", mediane)
```

- **Mode:** la valeur la plus fréquente dans un ensemble de données..

```
from scipy import stats

mode = stats.mode(data)
print("Mode :", mode.mode[0])
```

Statistiques descriptives

Les statistiques descriptives sont essentielles pour comprendre et analyser des données.

1. Mesures de dispersion

Ces mesures indiquent à quel point les données sont dispersées.

La variance: mesure la dispersion des données par rapport à la moyenne.

```
variance = np.var(data)
print("Variance :", variance)
```

Écart-type: la racine carrée de la variance. Il donne une mesure de dispersion dans les mêmes unités que les données.

-**Science des données** : Utilisé pour normaliser les données, détecter des outliers, etc.

```
ecart_type = np.std(data)
print("Écart-type :", ecart_type)
```

Statistiques descriptives

Les statistiques descriptives sont essentielles pour comprendre et analyser des données.

1. Corrélation et covariance

-Ces mesures indiquent la relation entre deux variables**.**

- **La covariance:** mesure comment deux variables varient ensemble. Une covariance positive indique que les variables évoluent dans le même sens, tandis qu'une covariance négative indique qu'elles évoluent en sens opposé.

```
x = [10, 20, 30, 40, 50]
y = [5, 15, 25, 35, 45]

covariance = np.cov(x, y)[0, 1]  # np.cov retourne une matrice de covariance
print("Covariance :", covariance)
```

- **La Corrélation:** mesure la force et la direction de la relation linéaire entre deux variables. Elle est souvent calculée avec le coefficient de Pearson, qui varie entre -1 (corrélation négative parfaite) et 1 (corrélation positive parfaite).

```
correlation = np.corrcoef(x, y)[0, 1]  # np.corrcoef retourne une matrice de corrélation
print("Corrélation :", correlation)
```

Manipulation des données avec NumPy

5. Manipulation avancée

- **Concaténation :**

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.concatenate((a, b))
```

- **Transposition :**

```
arr_transposed = arr_2d.T
```

- **Tri :**

```
sorted_arr = np.sort(arr)
```

6. Sauvegarde et chargement

- **Sauvegarder un tableau :**

```
np.save('mon_tableau.npy', arr)
```

- **Charger un tableau :**

```
loaded_arr = np.load('mon_tableau.npy')
```

Démonstration:

Des calcules mathématiques et statistiques (Array).

Ex1: Exercice pratique (1/3)

1. Créez une matrice 3x3 avec des valeurs aléatoires.
2. Calculez la transposée de la matrice.
3. Multipliez la matrice par sa transposée.
4. Créez un tableau 2D représentant des données tabulaires (par exemple, 5 lignes et 3 colonnes).
5. Calculez la moyenne de chaque colonne.
6. Trouvez la valeur maximale et minimale de chaque ligne.

Ex1: Exercice pratique (2/3)

1. Créez un tableau 2D de forme (4, 4).
2. Extrayez les éléments diagonaux.
3. Remplacez la deuxième colonne par des zéros.

Ex1: Exercice pratique (Statistiques) (3/3)

1. Générez un tableau de 100 nombres aléatoires suivant une distribution normale.
2. Calculez la moyenne, la médiane et l'écart-type.
3. Tracez un histogramme des données (en utilisant Matplotlib)
4. Donner la formule mathématique pour calculer la médiane
5.

1. Ex4: Questions de cours
   Donner l'importance de la variance et noter la formule mathématique correspondante et sa relation par rapport à l'écart type
   Espérance (Expected Value) ?
   Variable discrètes vs continue?
   Variable qualitative vs quantitative?
   Variables ordinales vs nominales?

1.

Ex1: Exercice pratique (5)

1. Somme et soustraction des deux matrices:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

2. Multiplier deux matrices A et B:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad v = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$$

3. Deviser la matrice A sur B:

$$A = \begin{pmatrix} 4 & 6 \\ 8 & 10 \end{pmatrix}, \quad v = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

$$A = \begin{pmatrix} 4 & 6 \\ 8 & 10 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix}$$

2. Calculer le déterminant de:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

1.

Corr: Exercice pratique (1/3)

1. Somme et soustraction des deux matrices:

$$C = A + B = \begin{pmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$$

1. Multiplier deux matrices A et B:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

$$C = A \cdot B = \begin{pmatrix} (1 \cdot 5 + 2 \cdot 7) & (1 \cdot 6 + 2 \cdot 8) \\ (3 \cdot 5 + 4 \cdot 7) & (3 \cdot 6 + 4 \cdot 8) \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

$$b = A \cdot v = \begin{pmatrix} (1 \cdot 5) + (2 \cdot 6) \\ (3 \cdot 5) + (4 \cdot 6) \end{pmatrix} = \begin{pmatrix} 17 \\ 39 \end{pmatrix}$$

1.

$$b_1 = (1 \cdot 7) + (2 \cdot 8) = 7 + 16 = 23$$

$$b_2 = (3 \cdot 7) + (4 \cdot 8) = 21 + 32 = 53$$

$$b_3 = (5 \cdot 7) + (6 \cdot 8) = 35 + 48 = 83$$

$$b = \begin{pmatrix} 23 \\ 53 \\ 83 \end{pmatrix}$$

1. Deviser la matrice A sur B:

$$\begin{pmatrix} \frac{4}{2} & \frac{6}{3} \\ \frac{8}{2} & \frac{10}{3} \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 4 & \frac{10}{3} \end{pmatrix}$$

$$\begin{pmatrix} \frac{4}{2} & \frac{6}{3} \\ \frac{8}{4} & \frac{10}{5} \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

2. Calculer le déterminant de:

$$\det(A) = 1 \cdot \det \begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix} - 2 \cdot \det \begin{pmatrix} 4 & 6 \\ 7 & 9 \end{pmatrix} + 3 \cdot \det \begin{pmatrix} 4 & 5 \\ 7 & 8 \end{pmatrix}.$$

$$\det(A) = 1 \cdot (45 - 48) - 2 \cdot (36 - 42) + 3 \cdot (32 - 35).$$

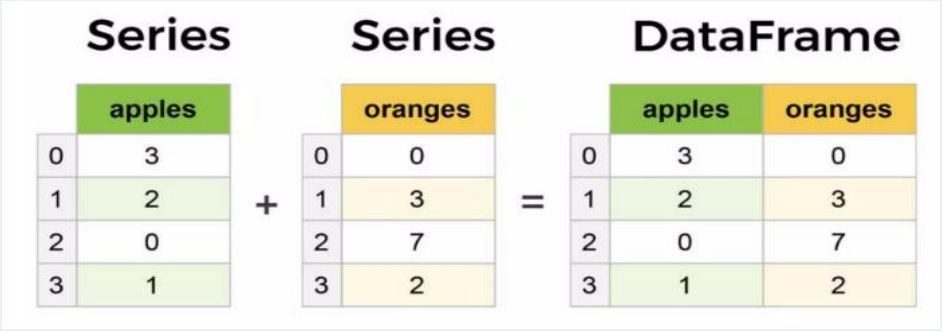
$$\det(A) = 1 \cdot (-3) - 2 \cdot (-6) + 3 \cdot (-3) = -3 + 12 - 9 = 0.$$

Opération	Dimensions de **A**	Dimensions de **B**	Dimensions du résultat
Somme (A + B)	$m \times n$	$m \times n$	$m \times n$
Soustraction (A - B)	$m \times n$	$m \times n$	$m \times n$
Multiplication (A \cdot B)	$m \times n$	$n \times p$	$m \times p$
Division élément par élément (A \oslash B)	$m \times n$	$m \times n$	$m \times n$

Manipulation des données avec Pandas

Pandas : Librairie Python pour extraire, préparer et éventuellement analyser, des données

- ❑ Chargement et exploration des données (CSV, Excel, JSON).
- ❑ Contient les classes Series et DataFrame (tables de données)
- ❑ Nettoyage des données : gestion des valeurs manquantes, suppression des doublons.
- ❑ Filtrage, sélection et agrégation des données.



- ❑ **Périmètre de Pandas** : Se restreint à la manipulation, lecture et prétraitement des données.
- ❑ Utilisation d'autres packages, comme **Scikit-learn**, pour l'analyse de données avancée.

Manipulation des données avec Pandas

- **1. Importer Pandas**
- Pour utiliser Pandas, commencez par l'importer :

```
import pandas as pd
```

- **2. Chargement des données**
- Pandas peut charger des données à partir de plusieurs formats.

```
df = pd.read_csv('fichier.csv')
```

```
df = pd.read_excel('fichier.xlsx', sheet_name='Feuille1')
```

```
df = pd.read_json('fichier.json')
```

2. Création d'un DataFrame à partir d'un dictionnaire

```
import pandas as pd

# Données
data = {
    'Nom': ['Alice', 'Bob', 'Charlie', ''],  # Une valeur manquante pour le nom
    'Age': [25, 30, 35, 0],                  # Une valeur manquante pour l'âge
    'Ville': ['Paris', 'Lyon', 'Marseille', '']  # Une valeur manquante pour la ville
}

# Création du DataFrame
df = pd.DataFrame(data)

# Affichage du DataFrame
print(df)
```

- **4. Exploration des données**
- Une fois les données chargées, vous pouvez les explorer.

Afficher les premières lignes

```
print(df.head())  # Affiche les 5 premières lignes
```

Afficher les dernières lignes

```
df.tail()
```

```
df.tail(10)
```

3. Création d'un DataFrame à partir de données aléatoires

```
import pandas as pd
import numpy as np

# Création d'un DataFrame avec des valeurs aléatoires
df_random = pd.DataFrame(np.random.randn(5, 4), columns=['W', 'X', 'Y', 'Z'])

# Affichage du DataFrame
print(df_random)
```

Manipulation des données avec Pandas

```python
import pandas as pd

# Création de plusieurs séries
s1 = pd.Series([10, 20, 30], name='Colonne1')
s2 = pd.Series(['A', 'B', 'C'], name='Colonne2')
s3 = pd.Series([True, False, True], name='Colonne3')

# Création du DataFrame à partir des séries
df = pd.DataFrame({s1.name: s1, s2.name: s2, s3.name: s3})
print(df)
```

```python
import pandas as pd

# Création de plusieurs séries
s1 = pd.Series([10, 20, 30], name='Colonne1')
s2 = pd.Series(['A', 'B', 'C'], name='Colonne2')
s3 = pd.Series([True, False, True], name='Colonne3')

# Combinaison des séries en un DataFrame
df = pd.concat([s1, s2, s3], axis=1)
print(df)
```

Manipulation des données avec Pandas

- **4. Exploration des données**
- Une fois les données chargées, vous pouvez les explorer.

Afficher les premières lignes

```
print(df.head())  # Affiche les 5 premières lignes
```

Afficher les dernières lignes

```
df.tail()
```

```
df.tail(10)
```

Informations sur le DataFrame

```
print(df.info())  # Affiche les types de données et les valeurs non nulles
```

Statistiques descriptives

```
print(df.describe())  # Résumé statistique des colonnes numériques
```

Dimensions du DataFrame

```
print(df.shape)  # Nombre de lignes et de colonnes
```

Afficher les colonnes

```
print(df.columns)  # Liste des colonnes
```

Afficher les index (lignes)

```
print(df.index)
```

Manipulation des données avec Pandas

- **5. Nettoyage des données**

- **Corrigez les erreurs de format dans les colonnes textuelles.**

```
df['text_column'] = df['text_column'].str.strip().str.lower()
```

Gestion des valeurs manquantes

```
print(df.isnull().sum())  # Compte les valeurs manquantes par colonne
```

Supprimer les lignes avec des valeurs manquantes :

```
df_cleaned = df.dropna()
```

Remplacer les valeurs manquantes :

```
df_filled = df.fillna(0)  # Remplacer par 0
df_filled = df.fillna(df.mean())  # Remplacer par la moyenne
```

Supprimer les doublons

```
df_no_duplicates = df.drop_duplicates()
```

Supprimer les doublons basés sur des colonnes spécifiques

```
df.drop_duplicates(subset=['Nom', 'Âge'], inplace=True)
```

Compter les occurrences de chaque JobTitle

```
job_counts = df['JobTitle'].value_counts()
```

```
df['JobTitle'].value_counts(dropna=False)
```

6. Filtrage et sélection des données

- Sélection de colonnes

```
df_selected = df[['colonne1', 'colonne2']]
```

- Filtrer avec une condition :

```
df_filtered = df[df['colonne1'] > 10]
```

- Filtrer avec plusieurs conditions :

```
df_filtered = df[(df['colonne1'] > 10) & (df['colonne2'] == 'valeur')]
```

7. Sélection par index

```
valeur = df.iloc[2, 1]  # Ligne 2, Colonne 1
```

```
ligne = df.iloc[2, :]  # Ligne 2, toutes les colonnes
```

```
sous_ensemble = df.iloc[1:4, 0:2]  # Lignes 1 à 3, Colonnes 0 à 1
```

Sélection par étiquette

```
valeur = df.loc[2, 'Nom']  # Ligne d'index 2, Colonne 'Nom'
```

```
ligne = df.loc[2, :]  # Ligne d'index 2, toutes les colonnes
```

```
sous_ensemble = df.loc[1:3, 'Nom':'Âge']  # Lignes 1 à 3, Colonnes 'Nom' à 'Âge'
```

.iloc[] : Utilise les indices numériques (position).
.loc[] : Utilise les étiquettes (noms des lignes et des colonnes).

Manipulation des données avec Pandas

- **Corrigez les erreurs de format dans les colonnes textuelles.**

```
df['text_column'] = df['text_column'].str.strip().str.lower()
```

- **7. Agrégation des données**

- Agrégation simple

```
print(df['colonne1'].sum())  # Somme
print(df['colonne1'].mean())  # Moyenne
print(df['colonne1'].max())  # Maximum
print(df['colonne1'].min())  # Minimum
```

Agrégation par groupe

```
df_grouped = df.groupby('colonne2')['colonne1'].mean()  # Moyenne par groupe
```

Agrégation avec agg

```
df_agg = df.groupby('colonne2').agg({
    'colonne1': ['sum', 'mean'],
    'colonne3': 'max'
})
```

6. Manipulation avancée

- Ajouter une nouvelle colonne

```
df['nouvelle_colonne'] = df['colonne1'] * 2
```

- Appliquer une fonction à une colonne

```
df['colonne1'] = df['colonne1'].apply(lambda x: x * 2)
```

- Filtrer avec plusieurs conditions :

```
df_filtered = df[(df['colonne1'] > 10) & (df['colonne2'] == 'valeur')]
```

- Trier les données

```
df_sorted = df.sort_values(by='colonne1', ascending=False)
```

- Renommer les colonnes

```
df=df.rename({'Nom':'Nom1'},axis=1)
```

- Remplacer des valeurs de la colonne

```
df['Ville']=df['Ville'].replace('Paris', 'Nantes')
```

Manipulation des données avec Pandas

```
# Créer un DataFrame complexe
df = pd.DataFrame({
    'ID': [1, 2, 3, 4, 5],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Age': [24, 30, 22, 35, 29],
    'Salary': [50000, 60000, 55000, 70000, 65000],
    'Department': ['HR', 'IT', 'Finance', 'Marketing', 'Sales'],
    'JoinDate': ['2018-01-01', '2019-05-15', '2017-11-20', '2020-03-10', '2018-07-23']
})

# Sélectionner toutes les colonnes sauf 'ID' et 'JoinDate'
cols = df.columns.difference(['ID', 'JoinDate'])
df_selected = df[cols]

print(df_selected)
```

Manipulation des différents types de données

Pour un Data Scientist, comprendre les différents types de données est super important. Ça l'aide à savoir quel genre d'analyse faire, quel outil utiliser, et à quel niveau de complexité il doit s'attendre.

Il y a deux grandes catégories : les données qualitatives et les données quantitatives.

- **Les données qualitatives** sont celles qui se divisent en catégories. Par exemple, une marque de voiture, un code postal, ou même une évaluation de type "bon ou pas bon". Si les catégories n'ont pas de classement particulier, on parle de données **nominales** (comme les couleurs ou les pays). Si ces catégories ont un ordre, c'est des données **ordinales**, comme le niveau de satisfaction.

- **Les données quantitatives**, c'est du numérique, avec des valeurs mesurables, comme le prix ou la température. Elles se divisent en deux types : les données d'intervalle, où il n'y a pas de zéro réel (comme la température), et les données de ratio, où le zéro a une vraie signification (comme la distance ou l'âge).

Manipulation des données avec Pandas

Sauvegarde des données

- Sauvegarder en CSV

```
df.to_csv('fichier_modifie.csv', index=False)
```

- Sauvegarder en Excel

```
df.to_excel('fichier_modifie.xlsx', index=False)
```

- Sauvegarder en Json

```
df.to_json('fichier_modifie.json', orient='records')
```

Sauvegarde des données

```
df.to_csv('fichier_modifie.csv', index=False)
```

- Sauvegarder en Excel

```
df.to_excel('fichier_modifie.xlsx', index=False)
```

- Sauvegarder en Json

```
df.to_json('fichier_modifie.json', orient='records')
```

6. Manipulation avancée

- Retyper les valeurs

```
print(df.dtypes)
df['colonne'] = df['colonne'].astype(int)
df['colonne'] = df['colonne'].astype(float)
df['colonne'] = df['colonne'].astype(str)
df['colonne'] = df['colonne'].astype(bool)
df['colonne'] = df['colonne'].astype(bool)
df['colonne'] = pd.to_datetime(df['colonne'])
df['colonne'] = pd.to_numeric(df['colonne'])
```

```
# Convertir la colonne 'B' en entier, forcer les erreurs à NaN
df['TotalPayBenefits'] = pd.to_numeric(df['TotalPayBenefits'], errors='coerce')
```

- Séparer les colonnes numériques et catégorielles

```
numerical_df = df.select_dtypes(include=['number'])  # Sélectionner les colonnes numériques
non_numerical_df = df.select_dtypes(exclude=['number'])  # Sélectionner les colonnes non numériques
```

```
xnum=df.loc[:,df.columns != 'Nom'];
xcat=df.loc[:,df.columns == 'Nom'];
```

Encodage des variables catégorielles (One-Hot Encoding, Label Encoding)

❑ **Encodage des variables (préparation des données) | Label encoding**

```
#df.dropna()
#df.dtypes

import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Créer une instance de LabelEncoder
le = LabelEncoder()

# Appliquer le LabelEncoder à la colonne 'Couleur'
df['JobTitle'] = le.fit_transform(df['JobTitle'])
df
```

- Concaténer les colonnes normalisées avec la colonne catégorielle

```
client = pd.concat([xnum_normalized, xcat], axis=1)
```

Encodage des variables catégorielles (One-Hot Encoding, Label Encoding)

❑**Encodage des variables (préparation des données) | One-Hot Encoding**

```
▷Initialize Reactive Jupyter | Sync all Stale code
#hot encoding
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Exemple de DataFrame avec des données catégorielles
data = {
    'Nom': ['Alice', 'Bob', 'Charlie', 'David'],
    'Couleur': ['Rouge', 'Vert', 'Bleu', 'Rouge'],
    'Ville': ['Paris', 'Lyon', 'Marseille', 'Paris']
}
df = pd.DataFrame(data)
# Afficher le DataFrame original
print("DataFrame original :")
print(df)

# Initialiser le OneHotEncoder
encoder = OneHotEncoder(sparse=False)  # sparse=False pour obtenir un tableau dense

# Appliquer OneHotEncoder à la colonne 'Couleur' et 'Ville'
encoded_data = encoder.fit_transform(df[['Couleur', 'Ville']])

# Convertir les résultats en DataFrame
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(['Couleur', 'Ville']))

# Ajouter les colonnes encodées au DataFrame original
df_encoded = pd.concat([df.drop(['Couleur', 'Ville'], axis=1), encoded_df], axis=1)

# Afficher le DataFrame après l'One-Hot Encoding
print("\nDataFrame après One-Hot Encoding :")
print(df_encoded)
```