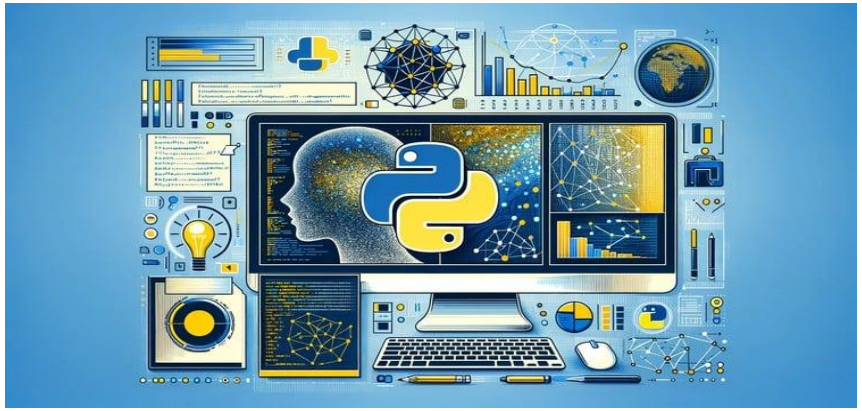
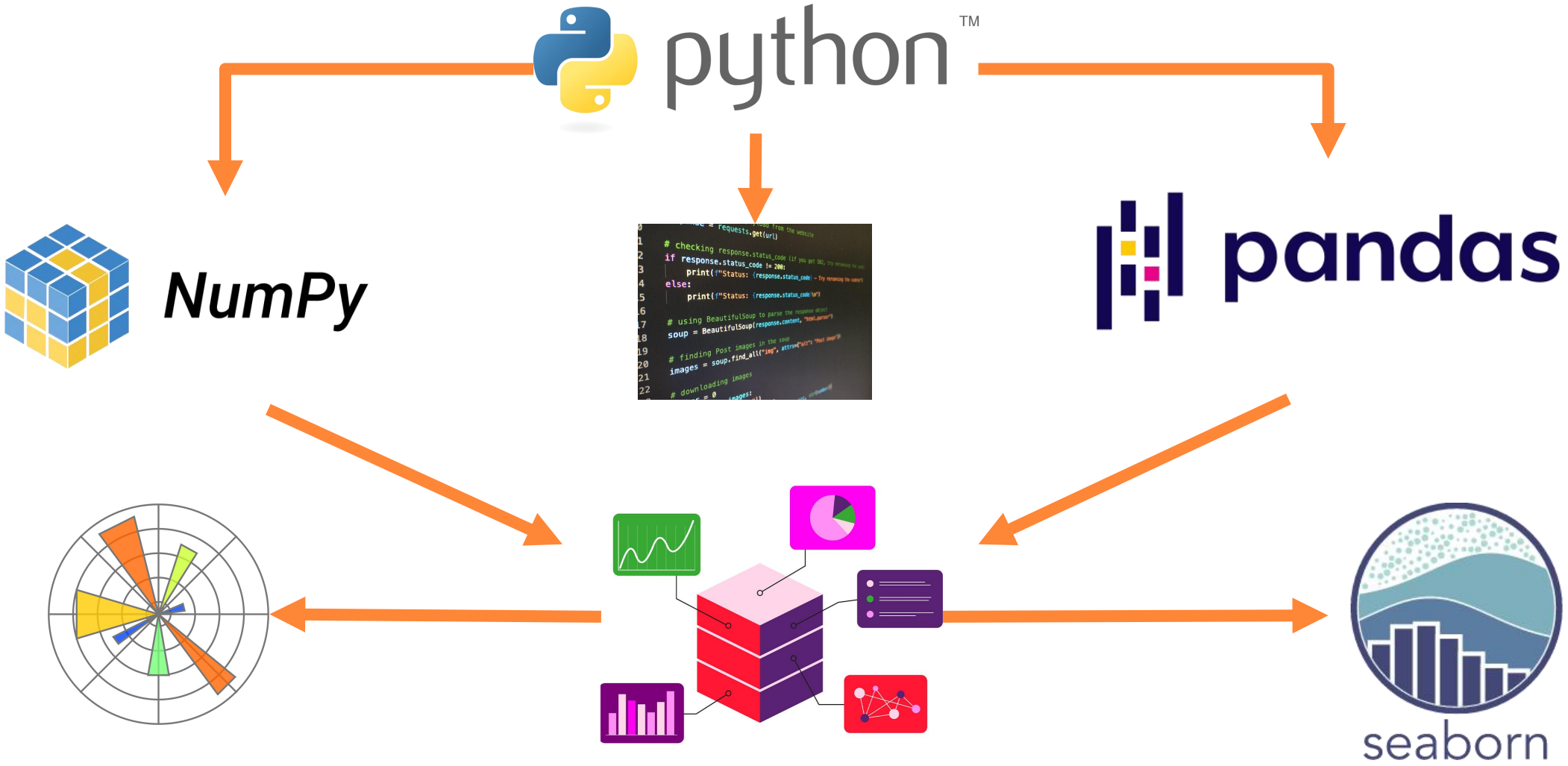
ADF PR

digital lifelong learning



Module : Analyse de données avec Python

Objectifs



Présentation générale du cours

- **Le nom du cours**
  - Analyse de données avec Python :
- **Volume horaire**
  - 08 heures
  - Cours + TP
- **Contenu**
  - **Chapitre 1** : Introduction au Python → un cours DE python
  - **Chapitre 2** : Python pour l'analyse de données → un cours EN python
- **Objectifs**
  - Syntaxe de base et structures de données en Python.
  - Exploration et traitement de données avec Python.

Plan

Chapitre 1 : introduction au Python

- Présentation de Python
- Les variables et les types de valeurs
- Les structures de contrôle
- Les données composites
- Les fonctions
- Les modules standard et paquets Python

Chapitre 2 : Python pour l'analyse de données

- Exploration de données avec Pandas et NumPy
- Visualisation avancée des données avec Matplotlib et Seaborn
- Machine Leraning et L'analyse prédictive

Chapitre 1 : introduction au Python

Syntaxe de base et structures de données en Python.

Présentation de Python

Historique et Définition

- Le langage de programmation Python a été créé en 1989 par Guido van Rossum
- La première version publique du langage est sortie en 1991.
- La dernière version de Python est la version 3
- Python est :
  - Multiplateforme.
  - Gratuit.
  - Un langage de “très haut niveau”
  - Un langage interprété.
  - Un langage orienté objet
  - Modulable et extensible.

Utilisation de python

- **Développement Web**
- **Science des Données et Analyse de Données**
  - Bibliothèques : Pandas, NumPy, Matplotlib, Seaborn, Plotly.
  - Analyse de données, visualisation, manipulation et nettoyage de grands ensembles de données.
- **Intelligence Artificielle et Machine Learning**
- **Développement d'Applications Desktop**
- **Développement d'Applications Scientifiques**
- **Sécurité et Cybersécurité**
- **Développement de Jeux Vidéo**
- **Traitement des Données et Fichiers**
- **IoT et Robotique**
- **Big Data et Cloud Computing**
- **Traitement d'Images et Vidéos**
- **Création d'APIs et Microservices**
- **…**

Comment utiliser pyhon ?

Comment utiliser pyhon ?

- Deux environnements faciles pour exécuter du code Python sans installation :

1. Google Colab (Google Labs) :

- Accessible en ligne, aucun logiciel à installer
- Aller sur : https://colab.research.google.com
- Cliquer sur "**Nouveau Notebook**" ou ouvrir depuis Google Drive

2. Jupyter Notebook :

- Utilisé localement ou sur serveur, très flexible
- Télécharger Anaconda : https://www.anaconda.com/products/distribution
- Installer Anaconda (Python inclus)
- Ouvrir Anaconda Navigator, puis lancer Jupyter Notebook

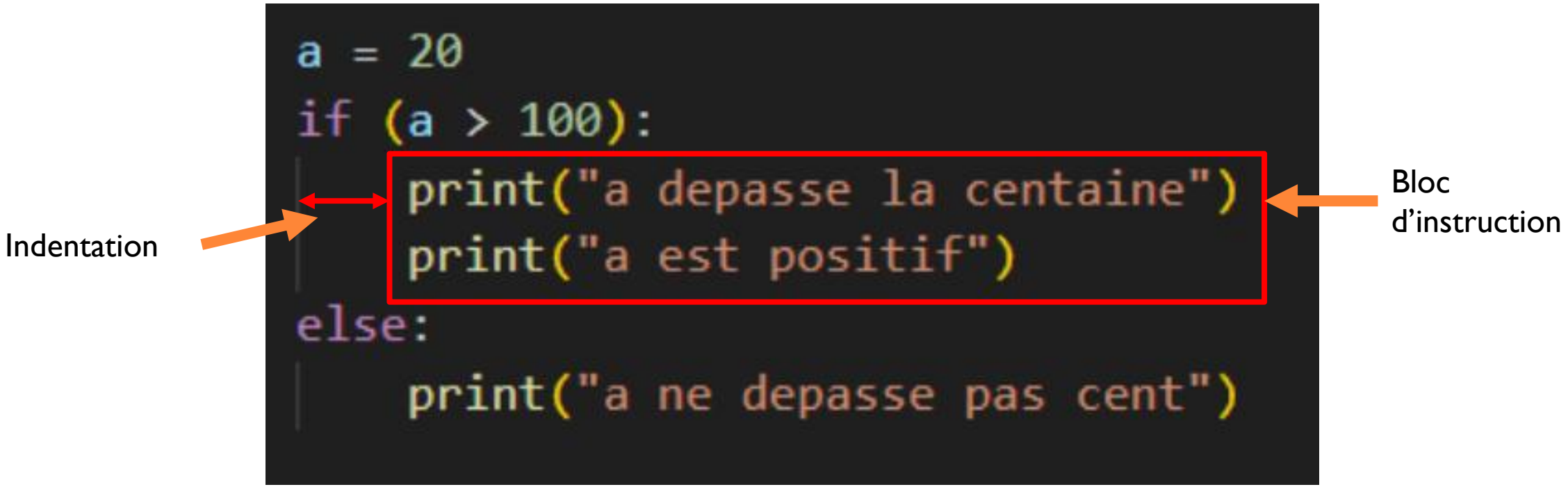
Syntaxe de base - Commentaires

- Pour ajouter un commentaire dans un code Python, on utilise le signe #.

```
1  # Votre premier commentaire en  Python.
2  print("Hello world!")
3
4  # D'autres commandes plus utiles pourraient suivre.
```

Syntaxe de base - Notion de bloc d'instructions et d'indentation

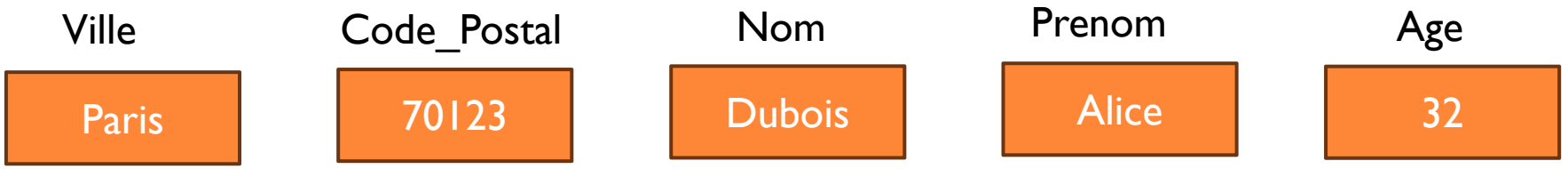
- En Python, l'indentation est utilisée pour définir des blocs de code, c'est-à-dire pour indiquer à l'interpréteur quelle instruction appartient à quelle autre. →l'ensemble des lignes indentées constitue un bloc d'instructions.



Les variables et les types de valeurs Python

Variables et opérateur d'affectation

- Une variable est une zone de la mémoire de l'ordinateur dans laquelle une valeur est stockée.



- Aux yeux du programmeur, une variable est définie par un nom, alors que pour l'ordinateur, il s'agit en fait d'une adresse, c'est-à-dire d'une zone particulière de la mémoire.
- En Python, la déclaration d'une variable et son initialisation (c'est-à-dire la première valeur que l'on va stocker dedans) se font en même temps.

```
1   >>> x = 2
```

Variables et opérateur d'affectation

- Python a « deviné » que la variable était un **entier**. On dit que Python est un langage au typage dynamique.
- Python a alloué (réservé) l'espace en mémoire pour y accueillir un entier. Chaque type de variable prend plus ou moins d'espace en mémoire. Python a aussi fait en sorte qu'on puisse retrouver la variable sous le nom x.
- Enfin, Python a assigné la valeur 2 à la variable x (affectation).

  ➔Dans d'autres langages (en C par exemple), il faut coder ces différentes étapes une par une. Python étant un langage dit de haut niveau, la simple instruction x = 2 a suffi à réaliser les 3 étapes en une fois !

```
1  >>> x = 2
2  >>> x
3  2
```

Variables et opérateur d'affectation

- Sous Python, on peut assigner une valeur à plusieurs variables simultanément.
- On peut aussi effectuer des affectations parallèles à l'aide d'un seul opérateur

```
>>> x = y = 8
>>> x
8
>>> y
8
>>> x,y = 14.2 , 20
>>> x
14.2
>>> y
20
>>>
```

Affectations multiples

Affectations parallèles

Nommage

- Le nom des variables en Python peut être constitué de :
  - lettres minuscules (a à z)
  - lettres majuscules (A à Z)
  - nombres (0 à 9)
  - caractère souligné (_).
  - Vous ne pouvez pas utiliser d'espace dans un nom de variable.
- Par ailleurs, un nom de variable ne doit pas débuter par un chiffre et il n'est pas recommandé de le faire débuter par le caractère _ (sauf cas très particuliers).
- De plus, il faut absolument éviter d'utiliser un mot « réservé » par Python comme nom de variable (par exemple : print, range, for, from, etc.).
- Et, bien que possible avec Python 3, l'utilisation de caractères accentués dans les noms des variables est fortement **déconseillée**.
- Enfin, Python est sensible à la casse, ce qui signifie que les variables TesT, test et TEST sont différentes.

Affichage

- Pour afficher la valeur à l'écran, il existe deux possibilités. :

1 - La première consiste à entrer au clavier le nom de la variable, puis <Entrée>.

```
>>> a = 2
>>> a
2
>>> msg = "Quoi de neuf ?"
>>> msg
'Quoi de neuf ?'
>>>
```

2 - l'instruction print :

```
>>> print(msg)
Quoi de neuf ?
```

Affichage

- la fonction print() affiche l'argument qu'on lui passe entre parenthèses et un retour à ligne.
- La fonction print() peut également afficher le contenu d'une variable quel que soit son type. Par exemple, pour un entier :

```
1  >>> var = 3
2  >>> print(var)
3  3
```

- Il est également possible d'afficher le contenu de plusieurs variables

```
1  >>> x = 32
2  >>> nom = "John"
3  >>> print(nom, "a", x, "ans")
4  John a 32 ans
```

- A l'intérieur d'un programme, vous utiliserez toujours l'instruction print.

Saisie de données

- Pour permettre à l'utilisateur d'un programme de saisir la valeur d'une variable x, on utilise la fonction input(),
- ATTENTION : la variable saisie est toujours de type str. Pour la convertir en nombre, il faut utiliser la fonction int() ou la fonction float() :

```
>>> x = input("saisir la valeur de x : ")
saisir la valeur de x : 12
>>> x
'12'
>>> type(x)
<class 'str'>
>>> x = int(x)
>>> x
12
>>> type(x)
<class 'int'>
```

```
x = int(input("saisir la valeur de x : "))

# Quelle est, à votre avis, l'utilité de la fonction type() ?
```

- Écrivez un programme en Python qui demande à l'utilisateur de saisir son nom, puis affiche un message de salutation personnalisé.

```
Entrez votre nom : Alice
Bonjour, Alice
```

Exercice

- Écrivez un programme en Python qui demande à l'utilisateur de **saisir** deux nombres entiers. Le programme doit ensuite calculer la somme de ces deux nombres et afficher le résultat.

Saisie de données

- Exemple

```
#exemple de programme Python
x = float(input("la valeur de x : "))
y = float(input("la valeur de y : "))

print("x :",x)
print("y :",y)

test = (x + y) / 2

print("Résultat  :",test)
```

Les types de variables

Types Simples

- Nombre (integer/float) : 976, 0.14, -9.99)
- Chaîne de caractères (str) : "Salut"
- Booléen (boolean) : True, False
- Aucune valeur : None

Types Composites

- Liste : [1, 2, 3]
- Tuple : (1, 2, 3)
- Dictionnaire : { "Nom" : "Ali", "Age" : 29}
- Ensemble (Set) : { "Ali", 29, "Etudiant"}

Les nombres

- entiers (integer ou int) : réels ou virgules flottantes (float)

```
>>> a, b, c = 1, 2, 5 # a,b et c seront de type int
>>> x, y, z = 22.9, 2., 0.12 # x,y et z seront de type float
```

Les nombres – Opérateurs arithmétiques

- Opérations sur les types numériques - opérateurs

Opération	Résultat
x + y	somme de x et y
x - y	différence de x et y
x * y	produit de x et y
x / y	quotient de x et y (division réelle)
x // y	quotient entier de x et y (division entière)
x % y	reste de x sur y (modulo)
abs(x)	valeur absolue de x
int(x)	x converti en nombre entier
float(x)	x converti en nombre à virgule flottante
x ** y	x à la puissance y

```
>>> 5 // 4
1
>>> 5 % 4
1
>>> 8 // 4
2
>>> 8 % 4
0
```

Les nombres – Opérateurs arithmétiques

- Opérations sur les types numériques - opérateurs combinés : permet d'effectuer une opération et une affectation en une seule étape,

Opérateur	Exemple	Raccourci pour
+=	x += 5	x = x + 5
-=	x -= 3	x = x – 3
*=	x *= 6	x = x * 6
//=	x //= 2	x = x // 2
**=	x**=4	x = x ** 4
%=	x%=5	x = x % 5

```
>>> i = 0
>>> i = i + 1
>>> i
1
>>> i += 1
>>> i
2
>>> i += 2
>>> i
4
```

Les chaines de caractères

```
>>> a = "bonjour"
>>> a
'bonjour'
>>> b = 'salut'
>>> b
'salut'
>>> c = """girafe"""
>>> c
'girafe'
>>> d = '''lion'''
>>> d
'lion'
```

il faut l'entourer de guillemets

- doubles
- simples,
- trois guillemets successifs doubles ou simples)

Les chaines de caractères

- Opérations sur les chaînes de caractères : Pour les chaînes de caractères, deux opérations sont possibles, l'addition et la multiplication :

```
>>> chaine = "Salut"
>>> chaine
'Salut'
>>> chaine + " Python"
'Salut Python'
>>> chaine * 3
'SalutSalutSalut'
```

Opérations

- Opérations illicites

```
>>> "toto" * 1.3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'float'
>>> "toto" + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

- Notez que Python vous donne des informations dans son message d'erreur.

Exercice

- Essayez de prédire le résultat de chacune des instructions suivantes, puis vérifiez-le dans l'interpréteur Python :

```
1) (1+2)**3                    = 27
2) "Da" * 4                    = 'DaDaDaDa’
3) "Da" + 3                    TypeError
4) ("Pa"+"La") * 2             = 'PaLaPaLa’
5) ("Da"*4) / 2                TypeError
6) str(4) * int("3")           = '444’
7) int("3") + float("3.2")     = 6.2
8) str(3) * float("3.2")       TypeError
9) str(3/4) * 2                '0.750.75'
```

Les Booléens

- Un booléen est un type simple de Python qui n'a que deux etats.
- Les Booléens sont basés sur la prise des décisions.
- Deux valeurs possibles : True (Vrai) ou False (faux)
- Exemple :

```
>>> ch1 = "python"
>>> ch2 = "Python"
>>> ch1 == ch2
False
>>> ch2 = "python"
>>> ch1 == ch2
True
>>>
```

Si le feu est rouge, alors arrête-toi.

Les Booléens

- les expressions booléennes sont le plus souvent constituées :

Les opérateurs de comparaison

Opérateur	Signification
<	strictement inférieur
<=	inférieur ou égal
>	strictement supérieur
>=	supérieur ou égal
==	égal
!=	different

opérateurs logiques

Opérateur	Signification
AND	Renvoie True si toutes les deux expressions sont évaluées à True
OR	Renvoie True si une des comparaisons vaut True
NOT	Renvoie True si la comparaison vaut False (et inversement)

Les Booléens

Les opérateurs logiques :

- Les opérateurs logiques créent des conditions composées dans une formule
  - **AND** : Vrai lorsque les deux valeurs sont vraies
  - **OR** : Vrai si l'une ou l'autre des deux valeurs est vraie
  - **NOT** : Fait passer une valeur de faux à vrai, ou inversement
- Les tables de vérités :

A	B	A ET B
FAUX	FAUX	FAUX
FAUX	VRAI	FAUX
VRAI	FAUX	FAUX
VRAI	VRAI	VRAI

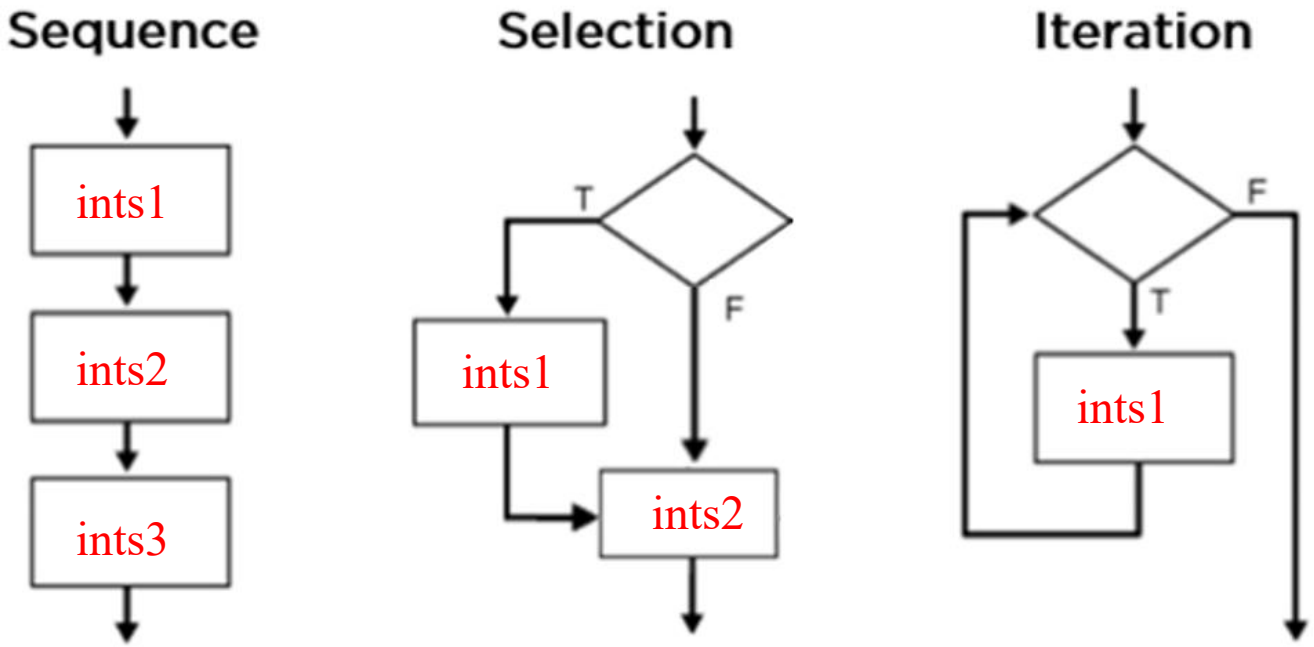
A	NON A
FAUX	VRAI
VRAI	FAUX

A	B	A OU B
FAUX	FAUX	FAUX
FAUX	VRAI	VRAI
VRAI	FAUX	VRAI
VRAI	VRAI	VRAI

Les structures de contrôle

Les structures de contrôle

- Les structures de contrôle sont les groupes d'instructions qui déterminent l'ordre dans lequel les actions sont effectuées dans le programme.

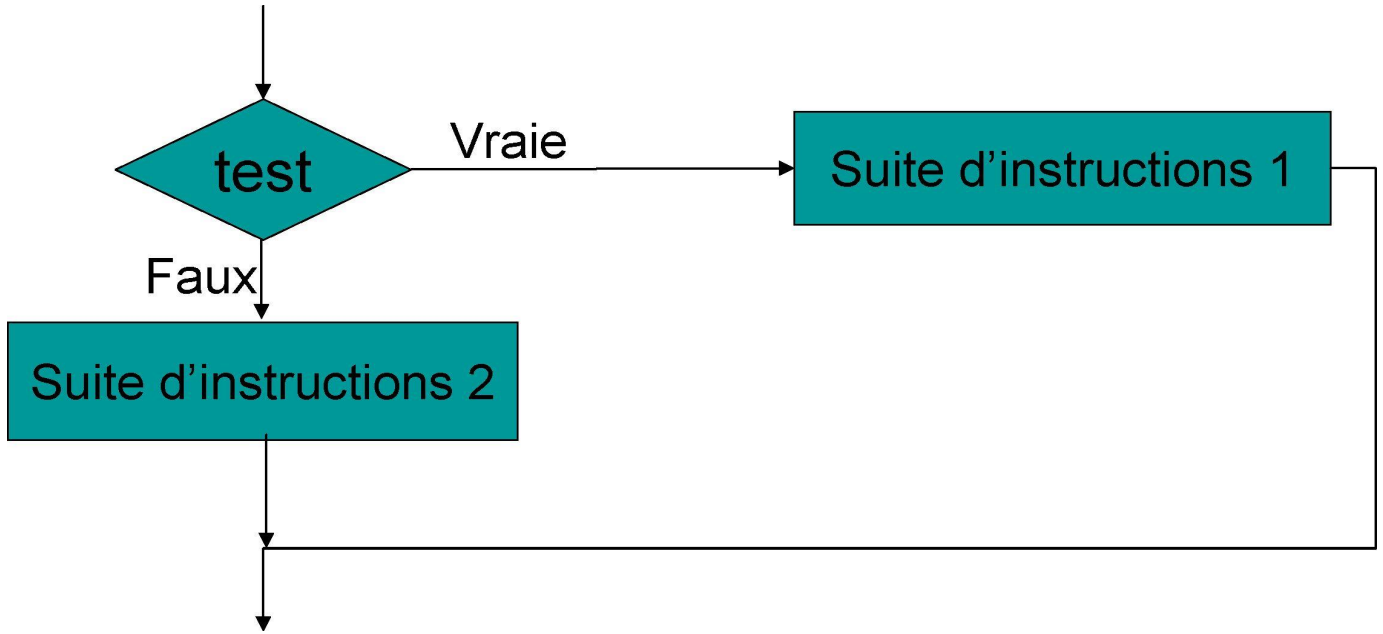


Séquence d'instructions

- Sauf mention explicite, les instructions d'un programme s'exécutent les unes après les autres, dans l'ordre où elles ont été écrites à l'intérieur du programme.
- Le « chemin » d'exécution est appelé un flux d'instructions, et les constructions qui le modifient sont appelées des instructions de contrôle de flux.
- Python exécute normalement les instructions de la première à la dernière, sauf lorsqu'il rencontre une structures de contrôle comme une instruction conditionnelle « if ». Une telle instruction va permettre au programme de suivre différents chemins suivant les circonstances.

Les instructions conditionnelles

- Une condition (test) est une expression écrite entre parenthèse à valeur booléenne.
- Les instructions conditionnelles (if en python) servent à n'exécuter une instruction ou une séquence d'instructions que si une condition est vérifiée.



Les instructions conditionnelles

Forme 1 : if

- La valeur de la condition sera interprétée en True ou False . Si la condition est correcte (évaluée à True) : le bloc d'instructions s'exécute

```
if condition :
        bloc d'instructions
```

```
>>> a = 150
>>> if (a > 0):
...     print("nombre positif ")
...
```

Indentation obligatoire

Les instructions conditionnelles

Forme 2 : if … else

- Si la condition mentionnée après if est VRAI (True), on exécute le `bloc 1`; si la condition est fausse, on exécute le `bloc 2` d'instructions.

```
if condition :
        bloc 1 d'instructions
else :
        bloc 2 d'instructions
```

```
>>> a = 150
>>> if (a > 0):
...     print("nombre positif ")
... else:
...     print("nombre négatif ou nul")
...
```

Exercice

- Écrire un programme python qui demande un nombre entier à l'utilisateur, puis qui teste et affiche s'il est négatif ou positif.

Les tests :

Forme 3 : Les tests imbriqués

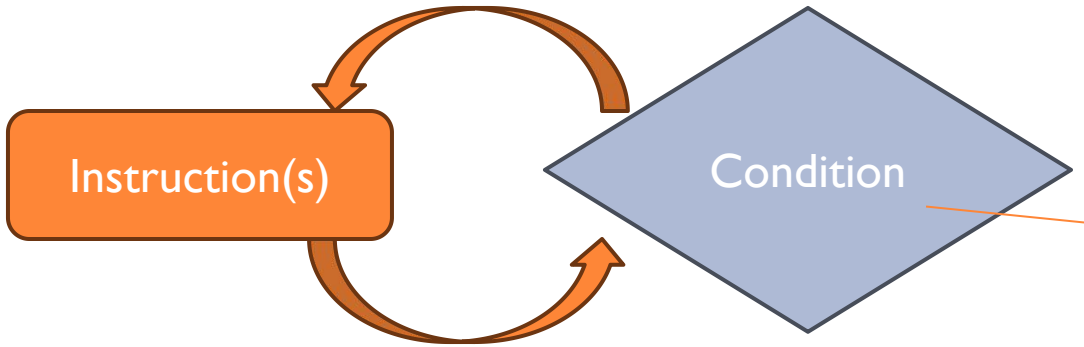
- On peut faire mieux encore en utilisant aussi l'instruction elif (contraction de « else if »)

```
if condition 1 :
        bloc 1 d'instructions
elif condition 2 :
        bloc 2 d'instructions
else :
        bloc 3 d'instructions
```

```
>>> a = 150
>>> if (a > 0) :
...     print("nombre positif ")
... elif (a < 0) :
...     print("nombre négatif ")
... else :
...     print("nombre nul")
...
```

Les boucles (Rappel)

- Instructions itératives : les boucles
- Les boucles servent à répéter l'exécution d'un groupe d'instructions un certain nombre de fois
- On distingue deux sortes de boucles en langages de programmation python :
  - Les boucles (while) : Répéter une action tant qu'une condition est vraie.
    - Exemple : Tant que je n'ai pas trouvé mes clés, je cherche dans chaque pièce de la maison.
  - Les boucles (for) : Répéter une action un nombre spécifique de fois.
    - Pour chaque jour de la semaine, je fais une activité différente.

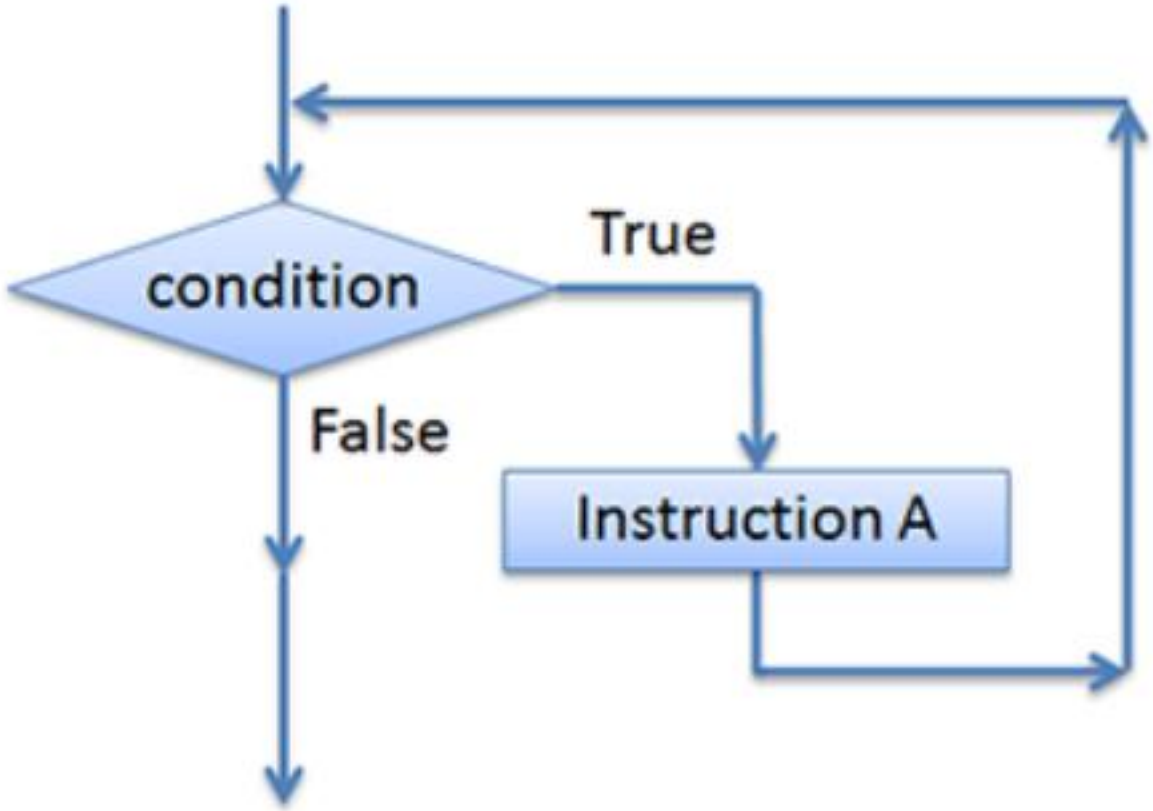


Une Condition est une expression qui peut êtres évaluée en **True** ou **False**

La boucle while

- La boucles while permet de répéter des instructions tant qu'une certaine condition est réalisée.

```
while condition :
        Blocs instruction A
```



La boucle while

```
i = 0
while ( i < 5 ):
    print("bonjour")
    i = i+1
print(i)
```

```
i = 0
1)  ( 0 < 5 ) ==> bonjour ==> i =0+1
2)  ( 1 < 5 ) ==> bonjour ==> i =1+1
3)  ( 2 < 5 ) ==> bonjour ==> i =2+1
4)  ( 3 < 5 ) ==> bonjour ==> i =3+1
5)  ( 4 < 5 ) ==> bonjour ==> i =4+1
6)  ( 5 < 5 ) *
```

```
bonjour
bonjour
bonjour
bonjour
bonjour
5
```

Les boucles - while

- La structure répétitive while permet d'effectuer une instruction ou des instructions (Bloc d'instructions) tant qu'une condition est satisfaite (évaluée à True)
- Ce qui signifie : tant que la condition est vraie, on exécute le bloc d'instructions.

```
while condition :
        Blocs instructions
```

Exemple 1

```
i=0
while i<10:
    print("bonjour")
    i+=1
```

Boucle correcte :
Bonjour s'affiche 10 fois.

Exemple 2

```
i=0
while i<10:
    print("bonjour")
```

Boucle infinie :
Bonjour s'affiche infiniment (sans arrêt)

Exemple pratique

- Supposons que nous voulons écrire un programme qui demande à l'utilisateur de saisir un mot de passe. La boucle continuera jusqu'à ce que l'utilisateur saisît le mot de passe correct. .
- Mot de passe correct : **PWD@2023**

Exemple pratique

Supposons que nous voulons écrire un programme qui demande à l'utilisateur de saisir un mot de passe. La boucle continuera jusqu'à ce que l'utilisateur saisît le mot de passe correct.

```
mdp = input("Saisir le mot de passe : ")
while ( mdp != "PWD@2023") :
    print(" mot de passe incorrect ")
    mdp = input("Saisir à nouveau le mot de passe : ")
print("Mot de passe correct - Bienvenue - ")
```

La boucle for

- Dans la plupart des langages de programmation, la structure répétitive for permet de répéter des instructions un certain nombre de fois.
- En python, une boucle for est utilisée pour itérer sur une séquence (c'est-à-dire une liste, un tuple, un dictionnaire, un set, un intervalle de valeurs, ou une chaine de caractères). Autrement dit, elle permet de parcourir une séquence du premier au dernier élément.

Exemple d'utilisation :

éléments de l'itérateur

```
for i in [0, 1, 2, 3]:
    print("i a pour valeur", i)
```

itérateur

Bloc de code

Affichage après exécution :

```
i a pour valeur 0
i a pour valeur 1
i a pour valeur 2
i a pour valeur 3
```

La boucle for

- La boucle For peut parcourir une séquence de nombres en utilisant la fonction "range". La fonction range () renvoie une séquence de nombres, commençant par `0` par défaut et incrémentée de `1` (par défaut), et s'arrête avant un nombre spécifié. Syntaxe : range(start, stop, step) avec :
  - start (facultatif) : Un nombre **entier** spécifiant à quelle position commencer. La valeur par défaut est 0.
  - stop (Requis) : Un nombre **entier** spécifiant à quelle position s'arrêter (non inclus).
  - step (facultatif) : Un nombre **entier** spécifiant l'incrémentation. La valeur par défaut est `1`

```
>>> for i in range(0,10,1):
        print (i)
```

0
1
2
3
4
5
6
7
8
9

Exercice

- Ecrivez un programme qui affiche le mot « Informatique » 10 fois

```
Informatique
Informatique
Informatique
Informatique
Informatique
Informatique
Informatique
Informatique
Informatique
Informatique
```

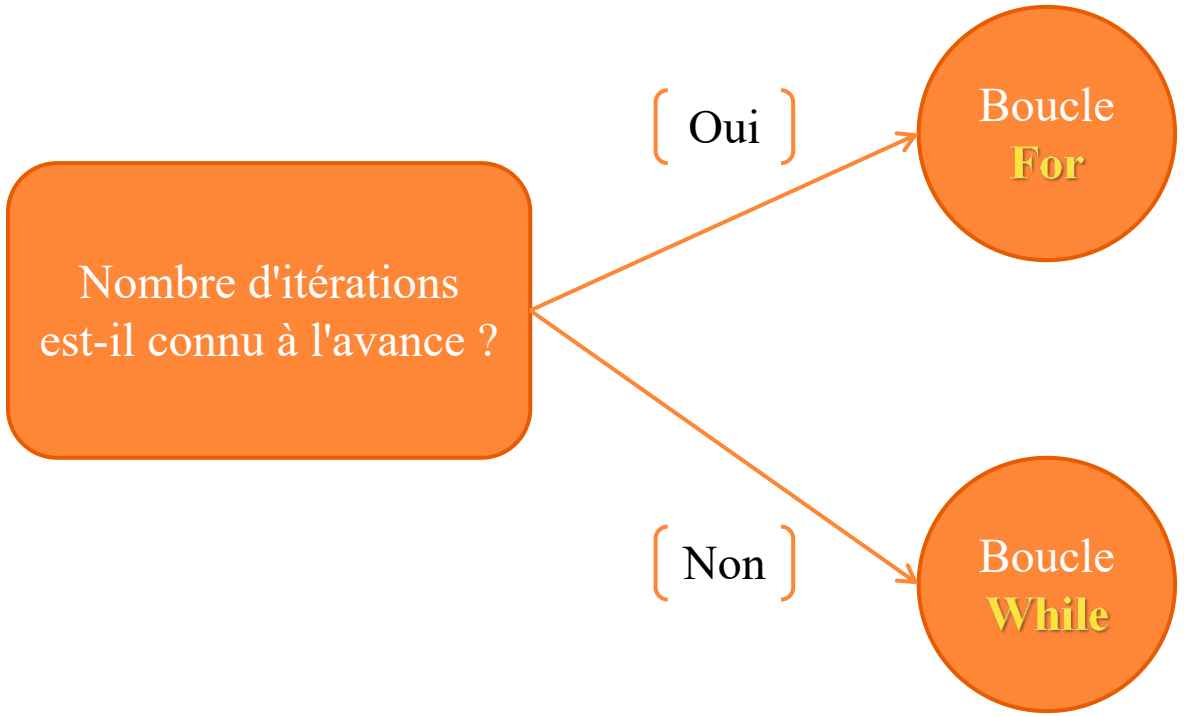
Solution 1 : while

```
n = 0
while( n < 10):
  print("Informatique")
  n = n + 1
```

Solution 2 : for

```
for m in range(0,10,1):
  print("Informatique")
```

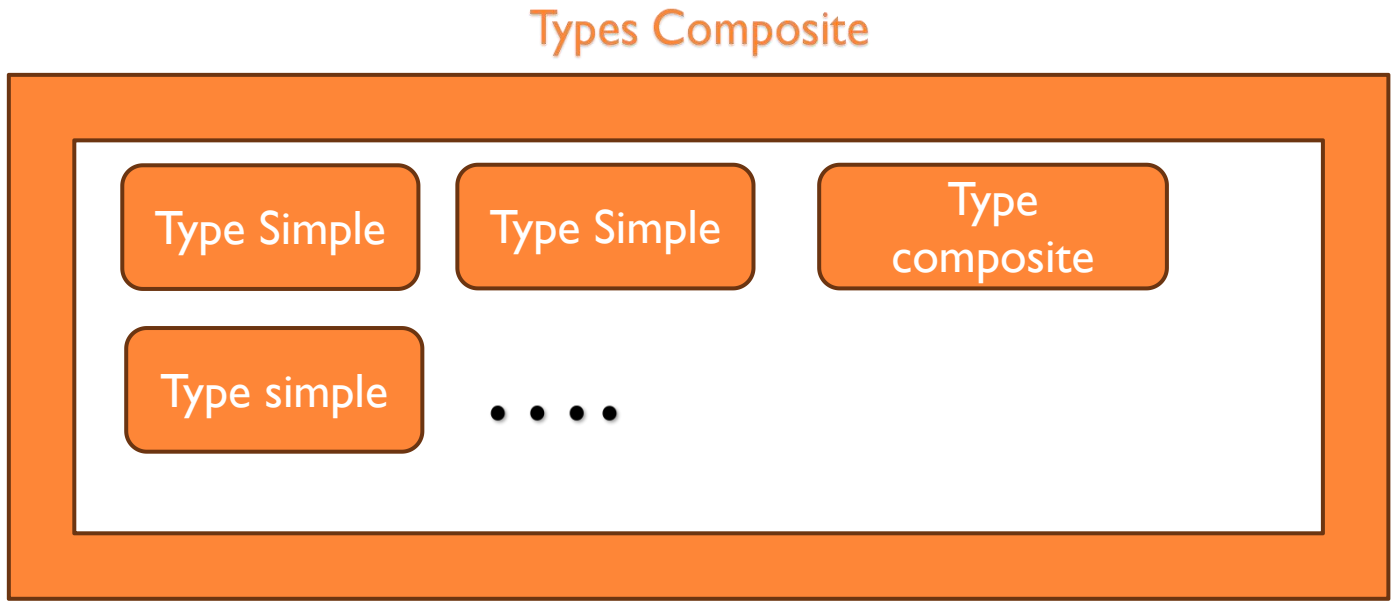
Choisir la structure de boucle adaptée



Les données composites

Les données composites

- Une donnée (variable ou expression) de type composite est une entité qui rassemble dans une seule structure un ensemble d'entités plus simples.
- Les types de données composites sont constitués d'autres types de données,
- Types Composites : Liste, Tuple, Dictionnaire, Ensemble (Set)



Les listes

- Une liste est une structure de données qui contient une série de valeurs.
- Python autorise la construction de liste contenant des valeurs de types différents.
- Une liste est déclarée par une série de valeurs séparées par des virgules, et le tout encadré par des crochets.

```
nom_liste = [ 10, 20,“Alice", 30, “Bob“ ]
```

```
>>> animaux = ["girafe", "tigre", "singe", "souris"]
>>> tailles = [5, 2.5, 1.75, 0.15]
>>> mixte = ["girafe", 5, "souris", 0.15]
>>> animaux
['girafe', 'tigre', 'singe', 'souris']
>>> tailles
[5, 2.5, 1.75, 0.15]
>>> mixte
['girafe', 5, 'souris', 0.15]
>>>
```

Les listes

- Les listes sont des séquences, c'est-à-dire des collections ordonnées d'objets. On peut accéder à chacun d'entre eux individuellement si l'on connaît son indice (index) dans la liste.
- La liste peut être indexée avec des nombres négatifs ou postifs selon le modèle suivant :

```
liste          : ["girafe", "tigre", "singe", "souris"]
indice positif :        0        1        2         3
indice négatif :       -4       -3       -2        -1
```

```
>>> animaux = ["girafe", "tigre", "singe", "souris"]
>>> animaux[-2]
'singe'
>>> animaux[0]
'girafe'
>>> animaux[3]
'souris'
>>>
```

Les listes

- Tranches : Un autre avantage des listes est la possibilité de sélectionner une partie d'une liste en utilisant un indiçage construit sur le modèle [m:n+1] pour récupérer tous les éléments, du m-ième au n-ième (de l'élément m inclus à l'élément n+1 exclu). On dit alors qu'on récupère une tranche de la liste.

```
>>> animaux = ["girafe", "tigre", "singe", "souris"]
>>> animaux[0:2]
['girafe', 'tigre']
>>> animaux[0:3]
['girafe', 'tigre', 'singe']
>>> animaux[1:]
['tigre', 'singe', 'souris']
>>> animaux[:1]
['girafe']
>>> animaux[:]
['girafe', 'tigre', 'singe', 'souris']
>>> animaux[1:-1]
['tigre', 'singe']
>>> animaux[-3:]
['tigre', 'singe', 'souris']
```

Les listes

- Tranches : On peut aussi préciser le pas en ajoutant un symbole deux-points supplémentaire et en indiquant le pas par un entier.

```
>>> animaux = ["girafe", "tigre", "singe", "souris"]
>>> animaux[0:3:2]
['girafe', 'singe']
>>> x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x[::1]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x[::2]
[0, 2, 4, 6, 8]
>>> x[::3]
[0, 3, 6, 9]
>>> x[0:9:3]
[0, 3, 6]
>>> x[1:6:3]
[1, 4]
```

Les listes

- Opération sur les listes : Tout comme les chaînes de caractères, les listes supportent l'opérateur + de concaténation, ainsi que l'opérateur * pour la duplication.

```
>>> ani1 = ["girafe", "tigre"]
>>> ani2 = ["singe", "souris"]
>>> ani1 + ani2
['girafe', 'tigre', 'singe', 'souris']
>>> ani1 * 3
['girafe', 'tigre', 'girafe', 'tigre', 'girafe', 'tigre']
>>>
```

Les listes

- Une liste Python peuvent être modifiée. On dit que les listes sont mutables.
- 4 types de mutations :
  - Ajout : l'opérateur de concaténation + ou la méthode append(),

    La méthode append() ajoute un seul élément à la fin d'une liste;
  - Modification : par substitution;
  - Insertion; La méthode insert() insère un objet dans une liste à un indice déterminé;
  - Suppression. L'instruction del supprime un élément d'une liste à un indice déterminé

```
>>> a = [1, 2, 3]
>>> a
[1, 2, 3]
>>> a = a + [5] # Ajout par addition
>>> a
[1, 2, 3, 5]
>>> a.append(10) # Ajout par la méthode append()
>>> a
[1, 2, 3, 5, 10]
>>> a[2] = 15 # Modification
>>> a
[1, 2, 15, 5, 10]
>>> a.insert(3, -24) # Insertion
>>> a
[1, 2, 15, -24, 5, 10]
>>> del a[3] # Suppression par l'instruction del
>>> a
[1, 2, 15, 5, 10]
```

Les listes - Méthodes

- `len(nomListe)` : Affiche la taille de la liste
- `sum(nomListe)` : Calcule la somme des éléments de la liste
- `max(nomListe)` : Cherche la valeur maximale dans la liste
- `min(nomListe)` : Cherche la valeur minimale dans la liste
- `nomListe.remove(Element)` : Retire un élément de la liste
- `nomListe.sort()` : Trie la liste
- `nomListe.pop()` : Retire le dernier élément de la liste

```
>>> x = [33,24,19,46,38]
>>> len(x)
5
>>> sum(x)
160
>>> max(x)
46
>>> min(x)
19
>>> x.remove(19)
>>> x
[33, 24, 46, 38]
>>> x.sort()
>>> x
[24, 33, 38, 46]
>>> x.pop()
46
>>> x
[24, 33, 38]
```


Pr.Abdelouafi ikidid ~ abdelouafi.ikidid@gmail.com

Exercices

- Exercice 1 :

Constituez une liste semaine contenant les 7 jours de la semaine.

1. À partir de cette liste, comment récupérez-vous seulement les 5 premiers jours de la semaine d'une part, et ceux du week-end d'autre part ? Utilisez pour cela l'indiçage.
2. Trouvez deux manières pour accéder au dernier jour de la semaine.

- Exercice 2 : Remplir une liste avec 10 entiers donnés par l'utilisateur .Puis, le programme trouve le plus grand et le plus petit de ces nombres.

Exercice 2

```
>>> jours = ['lun','mar','mer','jeu','ven','sam','dim']
>>> jours[0:5]  # 5 premiers jours
>>> jours[:5]   # 5 premiers jours
>>> jours[5:7]  # week-end
>>> jours[-2:]  # week-end
>>> jours[6]    # dernier jour
>>> jours[-1]   # dernier jour
>>> jours
```

Exercice 2

```
l=[]
#Remplissage
for i in range(10):
  l.append(int(input("Donner un nombre:")))
#Recherche le pp et pg
#1- initialisation
min=l[0]
max=l[0]
#2 -recherche dans le reste
for elt in l:
   if min>elt:
     min=elt
   if max<elt:
     max=elt
#3 affichage
print("Le plus petit :",min)
print("Le plus grand:",max)
print(l)
```

Les chaines de caractères

Chaînes de caractères ↔ Données composites

- Une chaîne de caractères est une donnée composite de type string str qui est une suite d'entités (ou items) plus simples, ➔ les caractères.
- On peut considérer une chaîne de caractères comme une séquence ordonnée de variables qui contiennent chacune un caractère. Les caractères sont indexés à partir de 0.
- Si on crée une variable de type str avec l'affectation chaine = 'Hello' , on obtient sa longueur avec len(chaine) et on accède au caractère d'indice i par son nom qui est chaine[i].
- Le type str est un type de variable non mutable, c'est pourquoi on ne peut modifier le premier caractère avec chaine[0]='V' par exemple.

Les chaines de caractères

Exemple

On observe que :

- on peut désigner un caractère directement par son indice ;
- on peut compter à partir de la fin ;
- on peut extraire des tranches;
- il n'est pas possible de changer une partie d'une chaîne.

```
>>> phrase = 'Python est un super langage de programmation'
>>> phrase[0:6]
'Python'
>>> phrase[0]
'P'
>>> phrase[-10]
'g'
>>> phrase[-10:]
'grammation'
>>> phrase[7:]
'est un super langage de programmation'
>>> phrase[:7]
'Python '
>>> phrase[0] = "p"
Traceback (most recent call last):
  File "<pyshell#55>", line 1, in <module>
    phrase[0] = "p"
TypeError: 'str' object does not support item assignment
>>> chaine = phrase[0:7] + 'v3'
>>> chaine
'Python v3'
```

Exercice

1. Écrivez un script qui détermine si une chaîne contient ou non le caractère «a».
2. Écrivez un script qui compte le nombre d'occurrences du caractère «e» dans une chaîne.

Les tuples

- Les tuples (« n-uplets » en français) sont des objets séquentiels correspondant aux listes (itérables, ordonnés et indexables) mais ils sont toutefois non modifiables.
- Les tuples sont un autre type séquentiel de données.
- Un tuple est une donnée immuable à la différence d'une liste qui est altérable. Cela signifie qu'on ne va pas pouvoir modifier les valeurs d'un tuple après sa création.
- L'intérêt des tuples par rapport aux listes réside dans leur **immutabilité**
- Pratiquement, on utilise les parenthèses au lieu des crochets pour les créer :

```
>>> t = (1, 2, 3)
>>> t
(1, 2, 3)
>>> type(t)
<class 'tuple'>
>>> t[2]
3
>>> t[0:2]
(1, 2)
>>> t[2] = 15
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    t[2] = 15
TypeError: 'tuple' object does not support item assignment
>>>
```

Les tuples

- Les opérateurs + et * fonctionnent comme pour les listes (concaténation et duplication) :

```
>>> (1, 2) + (3, 4)
(1, 2, 3, 4)
>>> (1, 2) * 4
(1, 2, 1, 2, 1, 2, 1, 2)
>>>
```

Les dictionnaires

- Les dictionnaires sont un autre type séquentiel de données.
- Leur différence réside dans leur façon d'indexer les valeurs.
- On a la liberté de choisir nous mêmes nos clefs (ou index ou indice) et attribuer la clef de notre choix à chaque valeur.
- Les valeurs d'un dictionnaire ne sont pas ordonnées à la différence des valeurs d'une séquence.
- On accède aux valeurs d'un dictionnaire par des clés

```
ani1 = {}
ani1["nom"] = "girafe"
ani1["taille"] = 5.0
ani1["poids"] = 1100
print(ani1)
```

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

nom	taille	poids
girafe	5.0	1100

Les dictionnaires : Itération sur les clés

- Si on souhaite voir toutes les associations clés / valeurs, on peut itérer sur un dictionnaire de la manière suivante :

```
>>> ani2 = {'nom': 'singe', 'poids': 70, 'taille': 1.75}
>>> for key in ani2:
...     print(key, ani2[key])
...
...
nom singe
poids 70
taille 1.75
>>>
```

Les dictionnaires

```
>>> d = {"nom": "Ali", "age":30, "sport":["Football", "Tennis"]}
>>> d["nom"] #Afficher les éléments du dictionnaire
'Ali'
>>> d["sport"] #Afficher les éléments du dictionnaire
['Football', 'Tennis']
>>> d["email"] = "ali123@est.ma" #Ajouter un élément au dictionnaire
>>> d["age"] = 40 #Modifier un élément du dictionnaire
>>> d
{'nom': 'Ali', 'age': 40, 'sport': ['Football', 'Tennis'], 'email': 'ali123@est.ma'}
>>> del d["email"] #Supprimer un élément du dictionnaire
>>> d
{'nom': 'Ali', 'age': 40, 'sport': ['Football', 'Tennis']}
>>>
```

Les ensembles (Sets)

- Les ensembles sont un autre type de données composites.
- Un ensemble est une collection d'éléments modifiables, non ordonnée, sans index et qui ne peut pas posséder l'élément dupliqué.

```
>>> s = {4, 5, 5, 12}
>>> s
{12, 4, 5}
>>> type(s)
<class 'set'>
>>>
```

Les ensembles (Sets)

- La fonction interne à Python set() convertit un objet itérable passé en argument en un nouveau set (opération de casting)

```
>>> set([1, 2, 4, 1])
{1, 2, 4}
>>> set((2, 2, 2, 1))
{1, 2}
>>> set(range(5))
{0, 1, 2, 3, 4}
>>> set({"clé_1": 1, "clé_2": 2})
{'clé_2', 'clé_1'}
>>> set(["ti", "to", "to"])
{'ti', 'to'}
>>> set("Python version 3")
{'n', 'e', 'h', 'v', 'P', 'o', 's', 'r', 'i', '3', ' ', 'y', 't'}
>>>
```

Les ensembles (Sets)

- Nous avons dit plus haut que les sets ne sont pas ordonnés ni indexables, il est donc impossible de récupérer un élément par sa position. Il est également impossible de modifier un de ses éléments par l'indexation

```
>>> s = set([1, 2, 4, 1])
>>> s[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
>>> s[1] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support item assignment
```

- Par contre, les sets sont itérables :

```
>>> for element in s:
...     print(element)
...
1
2
4
```

Les ensembles (Sets)

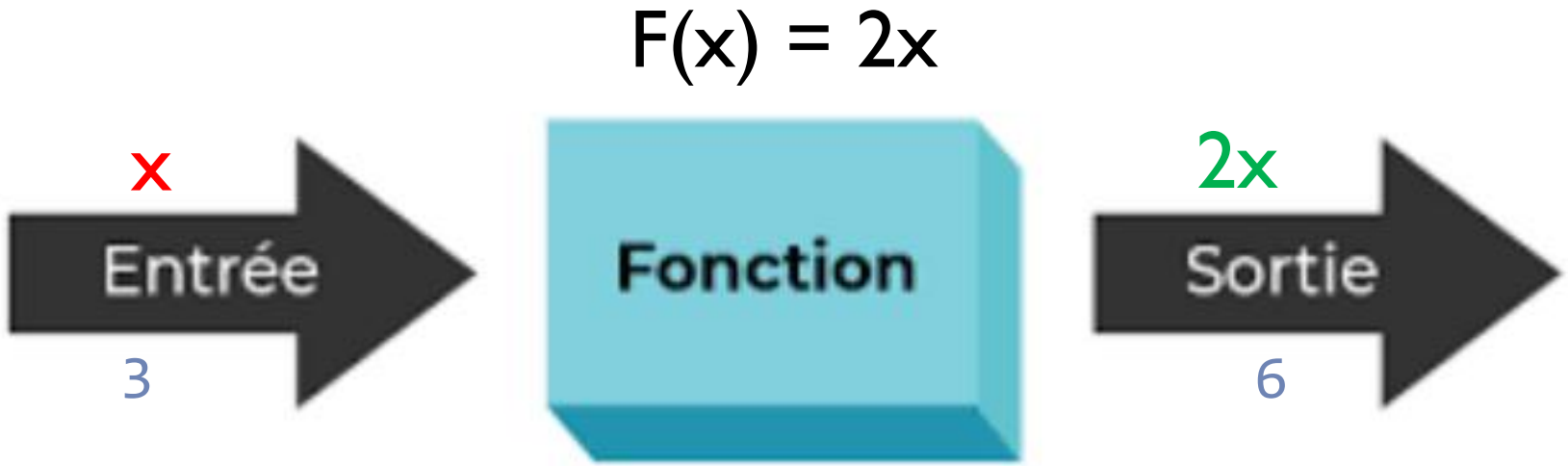
- Les sets ne peuvent être modifiés que par des méthodes spécifiques.

```
>>> s = set(range(5))
>>> s
{0, 1, 2, 3, 4}
>>> s.add(4)
>>> s
{0, 1, 2, 3, 4}
>>> s.add(472)
>>> s
{0, 1, 2, 3, 4, 472}
>>> s.discard(0)
>>> s
{1, 2, 3, 4, 472}
```

- les sets ne supportent pas les opérateurs + et *

Les fonctions

Qu’est ce qu’une fonction

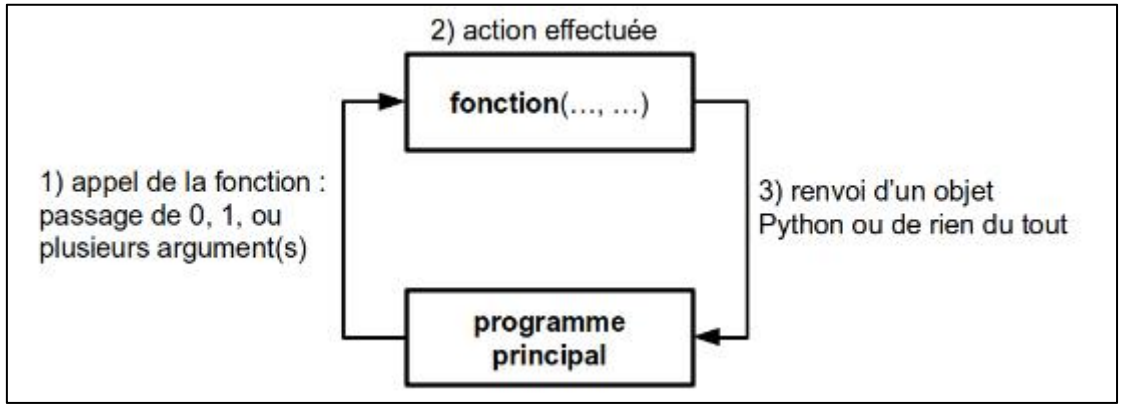



```
print("Bonjour")
```

→ Bonjour

Fonctions : Principe et généralités

- Les fonctions permettent de décomposer un programme complexe en une série de sous-programmes plus simples, lesquels peuvent à leur tour être décomposés eux-mêmes en fragments plus petits, et ainsi de suite…
  - Par exemple si nous disposons d'une fonction capable de calculer une racine carrée, nous pouvons l'utiliser un peu partout dans nos programmes sans avoir à la réécrire à chaque fois.
- Vous connaissez déjà certaines fonctions Python. Par exemple range() ou len(). Pour l'instant, une fonction est à vos yeux une sorte de « boîte noire » :
  - À laquelle vous passez aucune, une ou plusieurs variable(s) entre parenthèses. Ces variables sont appelées arguments.
  - Qui effectue une action.
  - Et qui renvoie un objet Python ou rien du tout



Fonctions : Déclaration d'une fonction

- Pour définir une fonction, Python utilise le mot-clé `def`. Si on souhaite que la fonction renvoie quelque chose, il faut utiliser le mot-clé `return`.

```
def   nomFonction(parametre1,parametre2,…) :
          instructions
          return  valeur_de_retour
```

- l'indentation de ce bloc d'instructions (qu'on appelle le corps de la fonction) est obligatoire.
- valeur_de_retour = récupérable dans une variable
- Exemple :

```
>>> def carre(x):
...     return x**2
...
>>> print(carre(2))
4
```

Fonctions : Déclaration d'une fonction

Fonction sans paramètres sans retour

```
def myFunction() :
    instruction1
    instruction2
    …

> myFunction()
```

Fonction sans paramètres avec retour

```
def myFunction() :
    instruction1
    instruction2
    …
    return valeur

> var = myFunction()
```

Fonction avec paramètres sans retour

```
def myFunction(p1, p2, …) :
    instruction1
    instruction2
    …

> myFunction(valeur1, valeur2, …)
```

Fonction avec paramètres avec retour

```
def myFunction(p1, p2, …) :
    instruction1
    instruction2
    …
    return valeur

> retour = myFunction(valeur1, valeur2, …)
```

Fonctions : Déclaration d'une fonction

Fonction sans paramètres sans retour

```
>>> def hello():
...     print("bonjour")
...
>>> hello()
bonjour
```

Fonction sans paramètres avec retour

```
def filiere() :
... return "SIR"
...
>>> filiere()
'SIR'
```

Fonction avec paramètres sans retour

```
>>> def bonjour(p) :
... print("bonjour ", p)
...
>>> bonjour("mohamed" )
bonjour mohamed
```

Fonction avec paramètres avec retour

```
>>> def fois(x, y):
...     return x*y
...
>>> fois(2, 3)
6
>>> prod = fois(2, 3)
>>> prod
6
```

Fonctions : Fonctions prédéfinies

Built-in Functions				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

On peut découvrir toutes les fonctions built-in ici.

Exemples

Fonction	Description	Exemple
print()	Affiche une valeur	print("Bonjour")
len()	Donne la taille d'une chaîne ou d'une liste	len("Python") → 6
type()	Donne le type d'une variable	type(3.14) → <class 'float'>
sum()	Fait la somme d'une liste de nombres	sum([1,2,3]) → 6
max()	Donne la plus grande valeur d'une séquence	max([10, 20, 5]) → 20
min()	Donne la plus petite valeur	min([10, 20, 5]) → 5
round()	Arrondit un nombre	round(3.14159, 2) → 3.14
sorted()	Trie une liste	sorted([5, 2, 9]) → [2, 5, 9]

Exercices

- **Exercice 1 :**

1. Proposer une fonction en python qui permet de calculer la somme des éléments d'une liste
2. Utiliser cette fonction pour calculer la somme des éléments d'une liste L dont les éléments sont générés aléatoirement.

- **Exercice 2 :**

1. Proposer une fonction qui construit une liste les diviseurs d'un nombre la fonction doit retourner la liste construite.
2. tester la fonction avec un programme de test.

- **Exercice 3 : Utilisation des fonctions récursives**
- Une fonction récursive est une fonction qui s'appelle elle-même, toutefois, il faut faire attention à la condition d'arrêt sinon on risque d'exécuter la fonction à l'infinie

1. Proposer une fonction qui calcule la factorielle d'un nombre
2. Utiliser la fonction pour calculer la somme des factorielles des 100 premiers nombres positifs

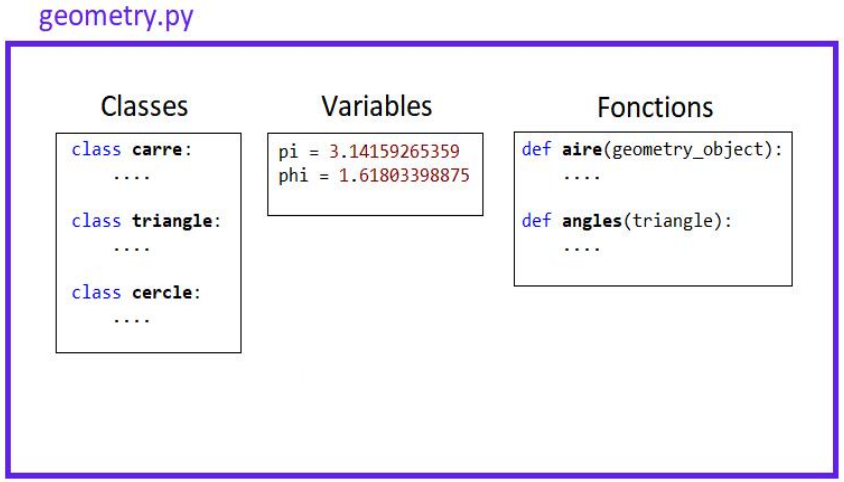
Les modules standard et paquets Python

Définition d'un module

- Module Python : On appelle "module" tout fichier constitué de code Python (c'est-à-dire tout fichier avec l'extension .py) importé dans un autre fichier ou script.
- On les appelle aussi bibliothèques ou *libraries*.
- Il y a trois types de modules Python :
  - Les modules standards qui ne font pas partie du langage en soi mais sont intégrés automatiquement par Python ;
  - Les modules développés par des développeurs externes qu'on va pouvoir utiliser ;
  - Les modules qu'on va développer nous mêmes.
- Pour utiliser un module, on utilise la commande import.

Définition d'un module

- Un module est un fichier Python (.py) contenant un ensemble de fonctions, de classes et de variables prédéfinies et fonctionnelles, que vous pouvez utiliser comme bon vous semble dans votre code !
- Par exemple, si vous travaillez sur une problématique faisant intervenir de la **géométrie**, vous pourriez avoir besoin de :
  - classes :
    - carré – défini par la longueur de son côté,
    - triangle – défini par la longueur de ses trois côtés,
    - cercle – défini par son rayon,
    - etc. ;
  - variables :
    - pi : constante indispensable pour calculer l'aire d'un cercle, égale à 3,1415...,
    - phi : constante représentant le nombre d'or, égale à 1,6180... ;
  - fonctions :
    - aire : qui prend en paramètre un objet géométrique (carré, triangle, etc.) et calcule son aire,
    - angles : qui prend en paramètre un triangle, et calcule les angles internes de ce dernier ,
    - etc.



Importation de modules

- Exemple :

```
>>> import random                    # 1
>>> random.randint(0, 10)            # 2
4                                    # 3
```

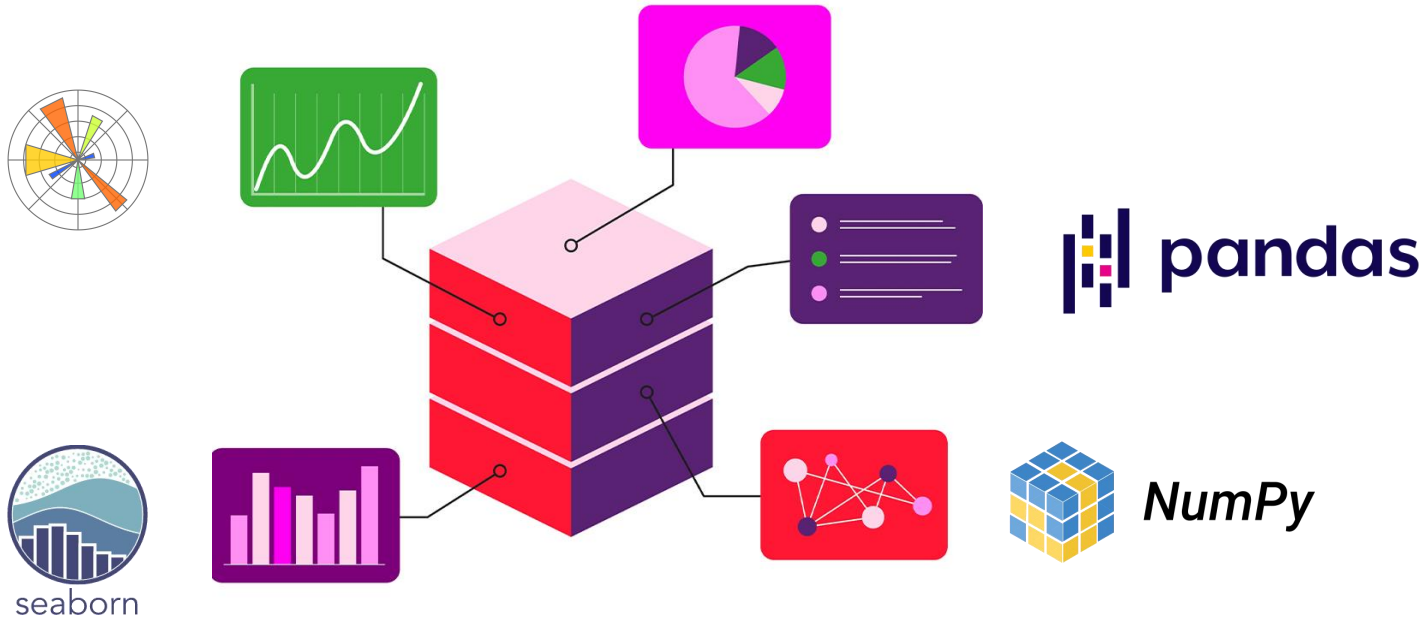
- Ligne 1, l'instruction import donne accès à toutes les fonctions du module `random`.
- Ensuite, ligne 2, nous utilisons la fonction `randint(0, 10)` du module `random`. Cette fonction renvoie un nombre entier tiré aléatoirement entre 0 inclus et 10 inclus.

Quelques modules courants

- math : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- sys : interaction avec l'interpréteur Python, passage d'arguments.
- os : dialogue avec le système d'exploitation.
- random : génération de nombres aléatoires.
- time : accès à l'heure de l'ordinateur et aux fonctions gérant le temps.
- ...
- Pour la liste complète, reportez-vous à la page des modules sur le site de Python.

Exemples

Module	Fonction ou constante	Description	Exemple
math	sqrt(x)	Racine carrée de x	math.sqrt(16) → 4.0
	pow(x, y)	Puissance x^y	math.pow(2, 3) → 8.0
	pi	Constante π	math.pi → 3.1415...
	log(x)	Logarithme népérien (base e)	math.log(10) → 2.302...
datetime	date.today()	Donne la date du jour	datetime.date.today()
	now()	Donne la date et l'heure actuelles	datetime.datetime.now()
random	randint(a, b)	Nombre entier aléatoire entre a et b inclus	random.randint(1, 10)
	choice(liste)	Choisit un élément au hasard dans une liste	random.choice(['A', 'B'])
	random()	Nombre réel aléatoire entre 0 et 1	random.random() → 0.54...
	uniform(a, b)	Nombre réel aléatoire entre a et b	random.uniform(5, 10) → 7.63...



Chapitre 2 : Python pour l'analyse de données

Exploration et traitement de données avec Python.

Les librairies Python pour l'analyse de données

Dans le domaine de l'analyse de données en particulier, voici une liste de librairies incontournables que nous allons voir dans le cadre de ce cours:

- numpy : librairie de calcul scientifique (algèbre, calcul matriciel, stochastique, etc.)
- pandas : librairie permettant la représentation, la manipulation et l'analyse de données sous forme de tableaux (ou DataFrames)
- matplotlib : librairie de visualisation graphique de données.
- seaborn : librairie de visualisation statistique avancée reposant sur matplotlib, offrant des graphiques élégants, lisibles et optimisés pour l'analyse des relations entre les variables (nuages de points, cartes de chaleur, distributions, etc.)

Les librairies Python pour l'analyse de données

Pour importer un package, il faut tout d'abord qu'il soit installé sur la machine (ce qui est déjà fait). Ensuite, il faut utiliser les mots-clés suivants:

`import package as alias` ou `from package import subpackage as alias`

Exemple :

Syntaxe 1 : Importer tout le package avec un alias

```
import numpy as np
```

Cela vous permet d'utiliser toutes les fonctionnalités de numpy en écrivant simplement np.

Par exemple :

```
import numpy as np
resultat = np.sqrt(16)  # Utilise la fonction racine carrée de numpy
print(resultat)  # Affiche : 4.0
```

Syntaxe 2 : Importer uniquement une partie spécifique du package

```
from numpy import sqrt
```

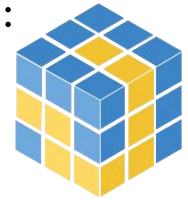
Cela vous permet d'utiliser directement la fonction sans utiliser l'alias :

```
from numpy import sqrt
resultat = sqrt(16)  # Pas besoin de préciser numpy
print(resultat)  # Affiche : 4.0
```

Numpy

Qu'est-ce que NumPy ?

- Le terme NumPy est en fait l'abréviation de « Numerical Python ». Il s'agit d'une bibliothèque Open Source en langage Python.
- NumPy (Numerical Python) est une bibliothèque Python spécialisée dans :
  - Le calcul scientifique.
  - La manipulation efficace de tableaux de données (appelés ndarray).
  - Les opérations mathématiques vectorisées.
  - Le calcul matriciel, les fonctions statistiques et les algèbres linéaires.



NumPy

http://www.numpy.org

- Pourquoi NumPy en Analyse de Données ?
  - NumPy est la base de nombreuses bibliothèques comme Pandas, SciPy, Scikit-learn.
  - Il permet de traiter rapidement de grandes quantités de données numériques.
  - Il optimise les performances grâce à des opérations vectorisées (plus rapides que les boucles Python classiques).
- On utilise cet outil pour la programmation scientifique en Python, et notamment pour la programmation en Data Science, pour l'ingénierie, les mathématiques ou la science.

Utilisation

- Il faut au départ importer le package numpy avec l'instruction suivante :

```
import numpy as np
```

- Exemple

```
>>> import numpy as np
>>> np.pi
3.141592653589793
```

Tableaux - numpy.array()

Un tableau NumPy (appelé ndarray : n-dimensional array) est une structure de données qui permet de stocker et manipuler des séries de valeurs numériques sous forme de vecteurs (1D), matrices (2D), ou tableaux multidimensionnels (3D et plus).

C'est l'équivalent optimisé d'une liste Python, conçu pour les calculs scientifiques rapides.

- Création d'un tableau avec numpy.array()

```
numpy.array(liste_de_valeurs)
```

- Exemple1 :

```
import numpy as np
tableau = np.array([10, 20, 30, 40])
print(tableau)
```

```
Résultat :
[10 20 30 40]
```

- Exemple2

```
>>> import numpy as np
>>> a = np.array([1, 2, 3, 4])
>>> a
array([1, 2, 3, 4])
>>> type(a)
<class 'numpy.ndarray'>
>>> a[0]
1
>>> a[3]
4
>>>
```

Tableau 2D

- Il est possible de créer un tableau 2D en utilisant une liste de listes au moyen de crochets imbriqués. Les listes internes correspondent à des lignes du tableau.

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

- Exemple

```
>>> import numpy as np
>>> b = np.array([[1, 2, 3], [4, 5, 6]])
>>> b
array([[1, 2, 3],
       [4, 5, 6]])
>>> type(b)
<class 'numpy.ndarray'>
>>> b[0,1]
2
>>> b[1,2]
6
```

Propriétés utiles d'un tableau NumPy

```
import numpy as np
price_list = [ 100000, 300000, 250000]
price_array = np.array(price_list)
print(price_array)
print("shape:", price_array.shape)
print("mean:", price_array.mean())
print("argmax:", price_array.argmax())
print("cumsum:", price_array.cumsum())
```

shape : donne la forme du tableau.
Résultat : (3,) ce qui signifie un tableau à une dimension contenant 3 éléments.

mean() : calcule la moyenne des valeurs.
Résultat : (100000 + 300000 + 250000) / 3 = 216666.666...

argmax() : donne l'indice de la plus grande valeur.
Résultat : 1, car 300000 est à la position 1 (en commençant à 0).

cumsum() : retourne la somme cumulée des éléments.
Résultat : [100000 400000 650000]

- 100000
- 100000 + 300000 = 400000
- 400000 + 250000 = 650000

Fonctions mathématiques avec NumPy

Fonctions trigonométriques

numpy.sin(x)	sinus
numpy.cos(x)	cosinus
numpy.tan(x)	tangente
numpy.arcsin(x)	arcsinus
numpy.arccos(x)	arccosinus
numpy.arctan(x)	arctangente

Fonctions hyperboliques

numpy.sinh(x)	sinus hyperbolique
numpy.cosh(x)	cosinus hyperbolique
numpy.tanh(x)	tangente hyperbolique
numpy.arcsinh(x)	arcsinus hyperbolique
numpy.arccosh(x)	arccosinus hyperbolique
numpy.arctanh(x)	arctangente hyperbolique

Fonctions mathématiques avec NumPy

Fonctions diverses

x**n	x à la puissance n, exemple : x**2
numpy.sqrt(x)	racine carrée
numpy.exp(x)	exponentielle
numpy.log(x)	logarithme népérien
numpy.abs(x)	valeur absolue
numpy.sign(x)	signe

Pandas

Qu'est-ce que Pandas ?

- Pandas est une librairie python qui permet de manipuler et analyser facilement des données.
- manipuler des tableaux de données avec des étiquettes de variables (**colonnes**) et d'individus (**lignes**).
- Ces tableaux sont appelés DataFrames.
- On peut facilement lire et écrire ces Dataframes à partir ou vers un fichier tabulé.
- On peut facilement tracer des graphes à partir de ces DataFrames grâce à **matplotlib**.

https://pandas.pydata.org/

Pourquoi utiliser Panda Python ?

- Fournit une structure de donnée appelée Dataframe rapide et efficace pour la manipulation des données avec indexation intégrée ;
- Dispose d'outils pour lire et écrire dans des fichiers de différents formats (.Csv, .Txt, .Xlsx, .Sql, .Hdf5, etc…) ;
- Offre une flexibilité pour traiter les données de type hétérogènes ou manquantes ;
- Est open source ;
- Fournit une documentation très détaillée et facile à lire ;
- Est utilisée dans une grande variété de domaines universitaires et commerciaux, notamment la finance, les neurosciences, l'économie, les statistiques, la publicité, l'analyse web…

Utilisation

- Il faut au départ importer le package pandas avec l'instruction suivante :

```
import pandas as pd
```

DataSet

id	Prenom	Nom	email	genre	Note_CC	Note_TP	Note_EF	age
1	Danya	Cockman	dcockman0@fda.gov	Female	3	9	6	16
2	Gary	Blase	gblase1@tinyurl.com	Genderqueer	13	17	0	17
3	Marlowe	Korlat	mkorlat2@goodreads.com	Male	14	9	1	17
4	Blondie	Della	bdella3@ycombinator.com	Female	16	13	12	17
5	Sherline	Satcher	ssatcher4@bloglovin.com	Female	4	11	17	16
6	Nolie	Ivantyev	nivantyev5@prnewswire.com	Female	18	5	15	16
7	Francene	Killiner	fkilliner6@jimdo.com	Female	7	15	15	18
8	Aylmer	Dorning	adorning7@jugem.jp	Male	17	19	7	15
9	Christian	Backsal	cbacksal8@state.gov	Female	18	17	15	15

https://www.mockaroo.com/

Lecture de données

- le nom des colonnes est dans le fichier xlsx et l'index des lignes est numérique (de 0 a n-1)

importe la bibliothèque pandas

nom du fichier Excel

la feuille Excel (l'onglet)

```
import pandas as pd
data = pd.read_excel("NOTES_DATA.xlsx", sheet_name="f1")
print(data)
```

Chargement des données depuis un fichier xlsx

```
      id   Prenom       Nom                    email   genre  Note_CC  \
0      1   Pierce  Prozescky  pprozescky0@@biblegateway.com    Male        2
1      2   Jolene    Seddon   jseddon1@barnesandnoble.com  Female        6
2      3     Abel     Coppo        acoppo2@canalblog.com    Male       12
3      4    Keefe    O'Neal            koneal3@weibo.com    Male        5
4      5    Decca   Averies          daveries4@nhs.uk    Male       11
..   ...      ...       ...                         ...     ...      ...
995  996   Chicky    Hitter        chitterrn@cam.ac.uk    Male        9
996  997   Jolene  Spaughton  jspaughtonro@thetimes.co.uk  Female        2
997  998  Carlita      Belt        cbeltrp@symantec.com  Female       15
998  999    Lisha    Raspin          lraspinrq@google.pl  Female        6
999 1000  Annabel    Sissons     asissonsrr@guardian.co.uk  Female        8

     Note_TP  Note_EF  age
0          5        0   16
1         18       15   18
2          4       11   15
3         11       10   15
```

Affichage des informations

- `data.info()` : imprime des infos sur le Dataframe

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   id       1000 non-null   int64
 1   Prenom   1000 non-null   object
 2   Nom      1000 non-null   object
 3   email    1000 non-null   object
 4   genre    1000 non-null   object
 5   Note_CC  1000 non-null   int64
 6   Note_TP  1000 non-null   int64
 7   Note_EF  1000 non-null   int64
 8   age      1000 non-null   int64
dtypes: int64(5), object(4)
memory usage: 70.4+ KB
```

Sélection

- `data.head(), data.tail()` : les 5 premières ou les 5 dernières.

```
data.head()
```

	id	Prenom	Nom	email	genre	Note_CC	Note_TP	Note_EF	age
0	1	Pierce	Prozescky	pprozescky0@biblegateway.com	Male	2	5	0	16
1	2	Jolene	Seddon	jseddon1@barnesandnoble.com	Female	6	18	15	18
2	3	Abel	Coppo	acoppo2@canalblog.com	Male	12	4	11	15
3	4	Keefe	O'Neal	koneal3@weibo.com	Male	5	11	10	15
4	5	Decca	Averies	daveries4@nhs.uk	Male	11	16	19	15

```
data.tail()
```

	id	Prenom	Nom	email	genre	Note_CC	Note_TP	Note_EF	age
995	996	Chicky	Hitter	chitterrn@cam.ac.uk	Male	9	6	3	15
996	997	Jolene	Spaughton	jspaughtonro@thetimes.co.uk	Female	2	16	9	17
997	998	Carlita	Belt	cbeltrp@symantec.com	Female	15	15	12	16
998	999	Lisha	Raspin	lraspinrq@google.pl	Female	6	7	7	16
999	1000	Annabel	Sissons	asissonsrr@guardian.co.uk	Female	8	12	20	17

Sélection

- Sélection colonne

```
>>> data["email"]
0                    sluisetti0@nymag.com
1                 hclilverd1@mozilla.com
2       myoselevitch2@sciencedaily.com
3                  ekivelle3@plala.or.jp
4                   bspringate4@nasa.gov
                      ...
995          iclemenconrn@imageshack.us
996        amagrannellro@microsoft.com
997              cazemarrp@so-net.ne.jp
998                 cgiffonrq@jigsy.com
999              lborthwickrr@fotki.com
Name: email, Length: 1000, dtype: object
```

```
>>> data["email"][4]
'bspringate4@nasa.gov'
```

Sélection ligne

```
>>> data.loc[4]
id                                5
Prenom                      Bernhard
last_name                  Springate
email          bspringate4@nasa.gov
genre                          Male
Note_CC                       12.37
Note_TP                        9.76
Note_EF                        8.08
Name: 4, dtype: object
```

Sélection de ligne dans la colonne

```
>>> data["email"][4:10]
4                bspringate4@nasa.gov
5                 amiles5@state.tx.us
6              ksandyfirth6@weebly.com
7     jgallety7@washingtonpost.com
8           sbloomer8@macromedia.com
9                 ldeeney9@apache.org
Name: email, dtype: object
```

Sélection de plusieurs colonnes

```
>>> data[["Prenom", "Note_CC" ]].head()
      Prenom  Note_CC
0   Sharline     1.43
1      Horst     4.28
2  Mufinella    11.62
3    Edouard    12.82
4   Bernhard    12.37
```

- `data.columns` : les noms des colonnes

```
>>> data.columns
Index(['id', 'Prenom', 'last_name', 'email', 'genre', 'Note_CC', 'Note_TP',
       'Note_EF'],
      dtype='object')
```

- `data.index` : les noms des lignes

```
>>> data.index
RangeIndex(start=0, stop=1000, step=1)
```

- `data.shape` : renvoie la dimension du dataframe sous forme (nombre de lignes, nombre de colonnes)

```
>>> data.shape
(1000, 8)
```

Calcul sur les données

```
>>> data["Moyenne"] = data["Note_CC"]*0.25 + data["Note_TP"]*0.25 + data["Note_EF"]*0.50
>>> data
        id     Prenom    last_name  ... Note_TP Note_EF  Moyenne
0        1   Sharline     Luisetti  ...    2.99   10.16   6.1850
1        2      Horst     Clilverd  ...    1.80    6.80   4.9200
2        3  Mufinella  Yoselevitch  ...   14.12   11.25  12.0600
3        4    Edouard      Kivelle  ...   17.75    9.29  12.2875
4        5   Bernhard    Springate  ...    9.76    8.08   9.5725
..     ...        ...          ...  ...     ...     ...      ...
995    996      Ibbie    Clemencon  ...    7.25    2.70   8.1025
996    997     Arthur   Magrannell  ...   15.54    8.59  10.6725
997    998      Clair       Azemar  ...   13.28    0.60   4.9075
998    999     Cherin       Giffon  ...    2.12    5.13   7.0325
999   1000       Linn    Borthwick  ...   14.16   10.55   8.8625

[1000 rows x 9 columns]
>>>
```

Suppression d'une colonne

```
>>> del data["Moyenne"]
>>> data
        id     Prenom    last_name  ... Note_CC Note_TP  Note_EF
0        1   Sharline     Luisetti  ...    1.43    2.99    10.16
1        2      Horst     Clilverd  ...    4.28    1.80     6.80
2        3  Mufinella  Yoselevitch  ...   11.62   14.12    11.25
3        4    Edouard      Kivelle  ...   12.82   17.75     9.29
4        5   Bernhard    Springate  ...   12.37    9.76     8.08
..     ...        ...          ...  ...     ...     ...      ...
995    996      Ibbie    Clemencon  ...   19.76    7.25     2.70
996    997     Arthur   Magrannell  ...    9.97   15.54     8.59
997    998      Clair       Azemar  ...    5.15   13.28     0.60
998    999     Cherin       Giffon  ...   15.75    2.12     5.13
999   1000       Linn    Borthwick  ...    0.19   14.16    10.55

[1000 rows x 8 columns]
>>>
```

Dataframes et indexation

- Quand on boucle sur un dataframe, on boucle sur les noms des colonnes :

```
>>> for x in data:
...     print(x)
...
...
id
Prenom
last_name
email
genre
Note_CC
Note_TP
Note_EF
```

- On peut boucler sur les lignes d'un dataframe, chaque ligne se comportant comme un namedtuple :

```
>>> for x in data.itertuples():
...   print(x.Prenom)
...
...
Sharline
Horst
Mufinella
Edouard
Bernhard
Aleta
Karlan
Julio
Saw
```

Accès selon une condition :

```
>>> data[data["Note_EF"] > 18 ]
...
      id     Prenom   last_name  ... Note_CC Note_TP  Note_EF
9     10    Laurena      Deeney  ...    6.18    7.01    18.49
13    14   Emmanuel  Middleweek  ...    1.30    2.41    18.36
28    29    Rickert      Tuting  ...   14.40    3.32    19.73
35    36  Mellicent      Addams  ...    4.61   19.62    18.83
38    39    Ethelda     Jaffray  ...    0.40   14.18    19.54
..   ...        ...         ...  ...     ...     ...      ...
937  938     Moishe      Rablen  ...   10.90    9.73    18.63
940  941  Harriette      Vinick  ...    8.72   19.34    19.27
949  950    Amelina       Siggs  ...   18.88   18.98    19.96
972  973     Stacie    Phythien  ...    0.59   19.82    18.34
976  977   Benedikt  Reddlesden  ...   18.93   13.56    18.72

[96 rows x 8 columns]
```

Mesures Statistiques

- `min_value = data["nom_de_la_colonne"].min()`
- `max_value = data["nom_de_la_colonne"].max()`
- `sum_value = data["nom_de_la_colonne"].sum()`
- `prod_value = data["nom_de_la_colonne"].prod()`
- `count_value = data["nom_de_la_colonne"].count()`

Fonctions sur les dataframes

- `data["Col"].mean()` : renvoie la moyenne de la colonne `Col` (en ignorant les NaN)

fonctions similaires à mean :

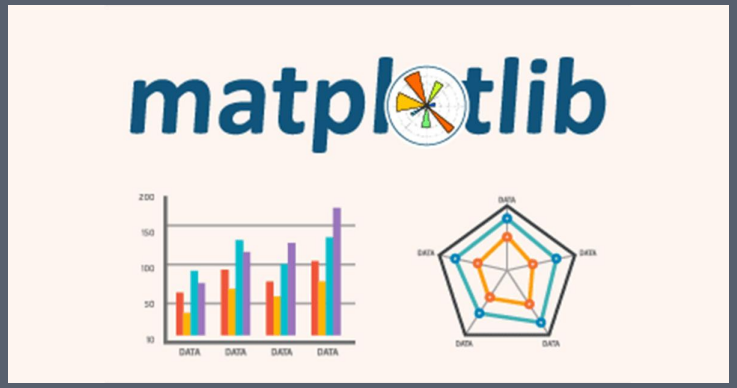
- min, max
- sum, prod : la somme, le produit.
- Mesures de Tendance Centrale
  - 1 – le mode `mode_value = data["nom_de_la_colonne"].mode()`
  - 2 – la médiane `mediane_value = data["nom_de_la_colonne"].median()`
  - 3 – la moyenne `mediane_rendement = data["Rendement"]. mean()`

L'opération de groupage

- La fonction groupby() est l'une des fonctions les plus utiles lorsqu'il s'agit de traiter des dataframes volumineux dans Pandas.
- Une opération groupby consiste généralement à diviser la dataframe, à appliquer une fonction et à combiner les résultats.

```
moyenne_ef_par_genre = data.groupby('genre')['Note_EF'].mean()
print(moyenne_ef_par_genre)
```

```
genre
Female    10.054737
Male      10.299048
Name: Note_EF, dtype: float64
```



Matplotlib et Techniques d’exploration des données

Analyse exploratoire des données (AED)

- **Qu'est-ce que l'Analyse exploratoire des données ?**
- L’analyse exploratoire des données (AED) est une approche initiale de l’analyse des données qui vise à résumer leurs principales caractéristiques, souvent à l’aide de méthodes visuelles et statistiques. Elle permet de :
  - Comprendre la structure des données disponibles.
  - Détecter les valeurs manquantes, aberrantes ou extrêmes.
  - Identifier les tendances, motifs et relations entre les variables.
- L'AED constitue une étape cruciale avant l'application de modèles d'apprentissage automatique ou prédictif, car elle permet de poser les bonnes hypothèses et de mieux préparer les données.

Analyse exploratoire des données (AED)

Visualisation des données

La visualisation des données est essentielle pour :

- Identifier des tendances globales.
- Détecter des valeurs aberrantes.
- Communiquer efficacement les résultats.

Types de visualisations utiles :

1. **Histogrammes** :
   - Utilisés pour visualiser la distribution des variables numériques.
   - Exemple : Analyser la fréquence des rendements mensuels d'un portefeuille financier.
2. **Diagrammes de dispersion** :
   - Montrent la relation entre deux variables numériques.
   - Exemple : Étudier la relation entre les taux d'intérêt et les rendements obligataires.
3. **Boxplots (Boîtes à moustaches)** :
   - Illustrent la dispersion et les valeurs aberrantes d'une variable.
   - Exemple : Comparer les rendements mensuels de différents fonds d'investissement.
4. **Séries temporelles** :
   - Visualisent les évolutions de variables au cours du temps.
   - Exemple : Suivre l'évolution des prix d'une action.

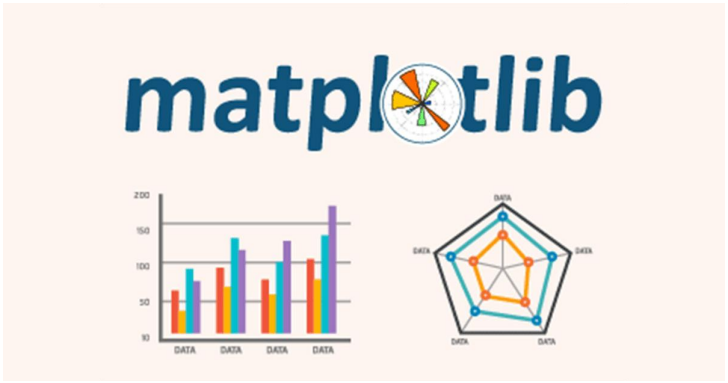
Analyse exploratoire des données (AED)

Visualisation des données

Visualisation des données en Python

- Pour tracer des courbes nous avons besoin de la bibliothèque Matplotlib utilisée dans ce cours.
- Si vous ne disposez pas de cette bibliothèque, vous pouvez cette commande pour installer l'environnement adapté.

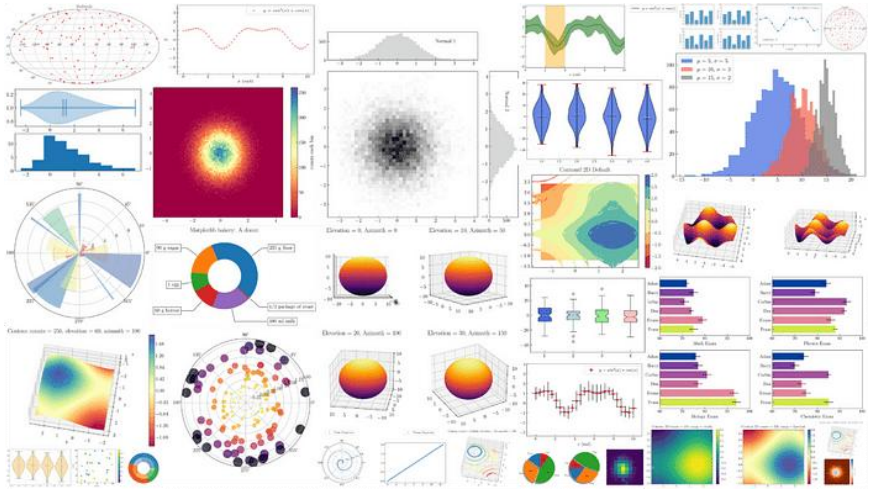
```
py -m pip install matplotlib
```



Analyse exploratoire des données (AED)

Visualisation des données

- Matplotlib est une bibliothèque python qui dessine des graphiques. Nul besoin de connaissances en interfaces graphiques pour créer un graphique dynamique avec possibilité de zoom et de sauvegarde par l'utilisateur. Il est d'ailleurs possible de sauvegarder les graphiques en format matriciels comme le PNG, JPEG , etc. et vectoriels comme le PDF et le SVG .



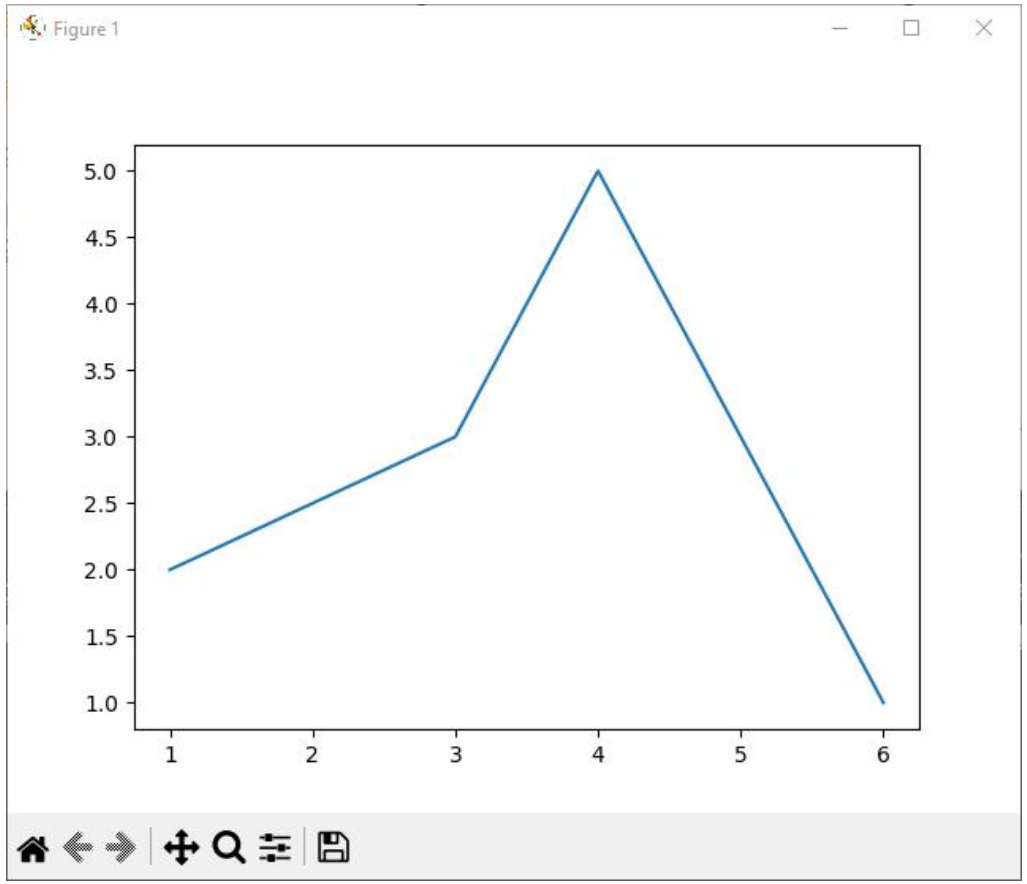
Création d'une courbe

- Importation de module : `import matplotlib.pyplot as plt`
- Chaque représentation graphique a une fonction correspondante avec Matplotlib :
  - nuage de points ou scatter plot, en anglais : `scatter()` ;
  - diagrammes en ligne ou en courbes : `plot()` ;
  - diagrammes en barres : `bar()` ;
  - histogrammes : `hist()` ;
  - diagrammes circulaires : `pie()`

diagrammes en ligne ou en courbes : `plot()`

- L'instruction `plot()` permet de tracer des courbes qui relient des points dont les abscisses et ordonnées sont fournies en arguments.
- Exemple

```
import matplotlib.pyplot as plt
x = [1,3,4,6]
y = [2,3,4,1]
plt.plot(x, y)
plt.show() # affiche la figure à l'écran
```

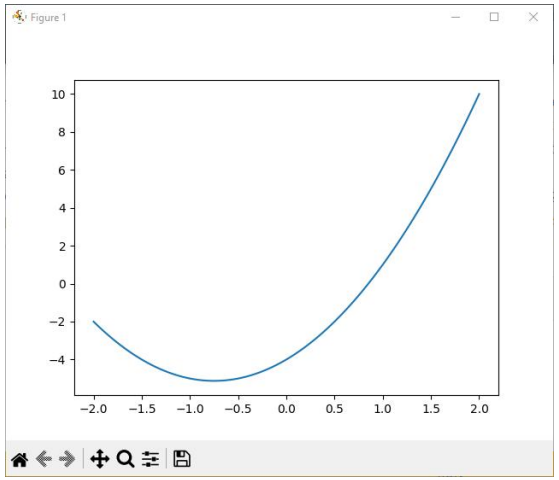


Création d'une courbe

- Exemple 2

$y = 2x^2+3x-4$ entre -2 et 2 en utilisant 100 points

```
>>> x = np.linspace(-2, 2, 100)
>>> y = 2*x**2+3*x-4
...
>>> plt.plot(x,y)
...
    [<matplotlib.lines.Line2D object at 0x000002109439C680>]
>>> plt.show()
```



Les fonctions de base

- `plt.show()` : Pour afficher le résultat.
- `plt.plot(liste_x,liste_y)` : placera les points de coordonnées (x_1,y_1), (x_2,y_2), ..., (x_n, y_n)
- Exemple :

```
>>> plt.plot([1,3,4],[2,1,6])
...
[<matplotlib.lines.Line2D object at 0x00000210900A6C60>]
>>> plt.show()
```

- plt.axis(x_min, x_max, y_min, y_max) : Cette fonction permet de modifier les axes du repère qui sera affiché
- plt.grid() : Affiche un quadrillage en plus sur notre repère.

Tracé de fonctions plus complexes

- Tracé de $y = \cos(x) + 3\sin(2x)$

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> import math
>>> abscisses = np.linspace(-4,4,100)
>>> ordonnées = [math.cos(x)+3*math.sin(2*x) for x in abscisses]
>>> plt.plot(abscisses,ordonnées)
    [<matplotlib.lines.Line2D object at 0x000002A9BD24F560>]
>>> plt.show()
```

Tracé de plusieurs fonctions

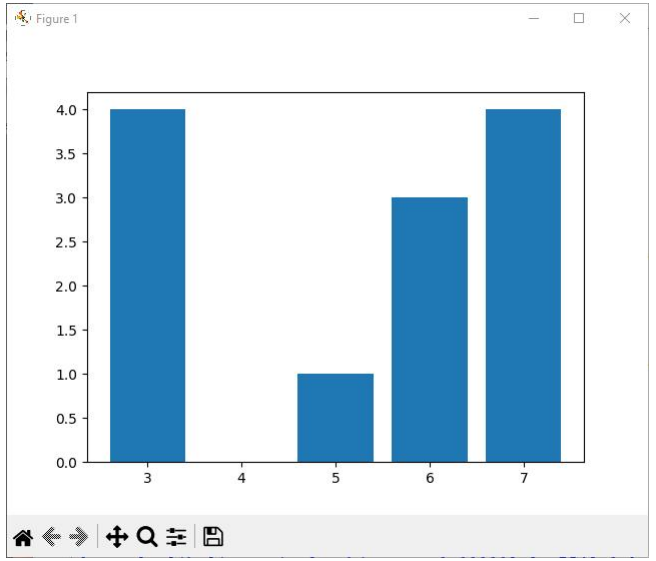
- $y = -2x + 3$
- $y = x^2 - 4x + 4$

```
>>> x = np.linspace(-1,3,100)
>>> y = -2*x+3
>>> plt.plot(x,y)
[<matplotlib.lines.Line2D object at 0x000002A9BD2C8830>]
>>>
>>> y = x**2 - 4*x + 4
>>> plt.plot(x,y)
[<matplotlib.lines.Line2D object at 0x000002A9BD7548C0>]
>>> plt.show()
```

Tracé de diagrammes en bâtons et histogrammes

- Pour les diagrammes en bâtons, on utilise la fonction `bar(valeurs,effectifs)`.
- `Exemple`

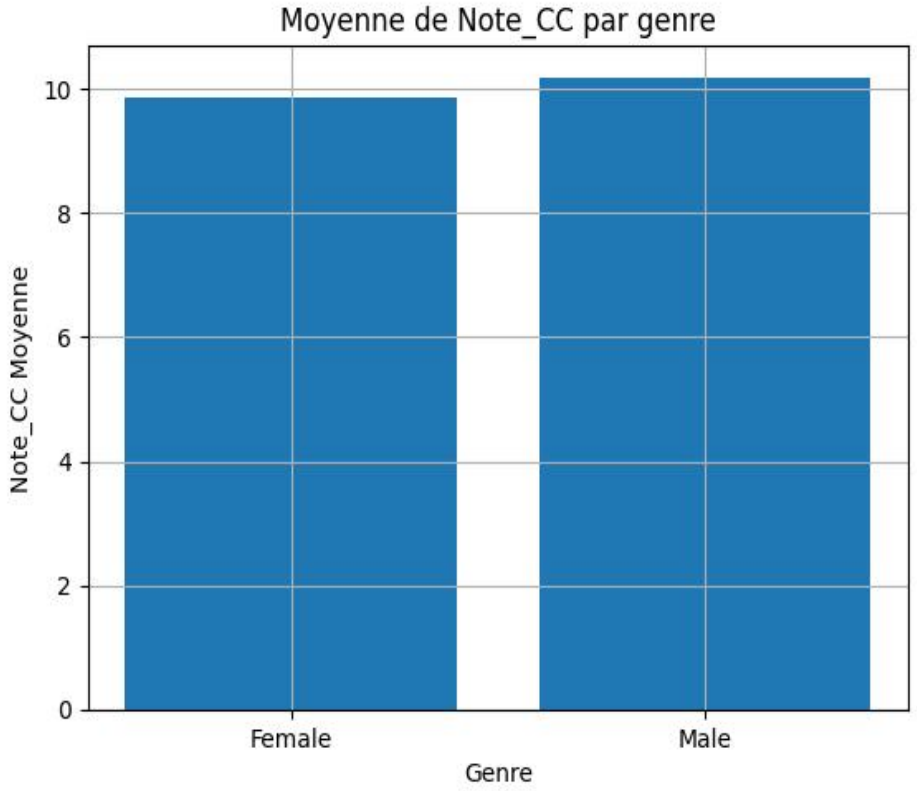
```
import matplotlib.pyplot as plt
x = [ 3, 5, 6, 7]
y = [ 4, 1, 3, 4]
plt.bar(x,y)
plt.show()
```



Tracé de diagrammes en bâtons et histogrammes

- Comparer les performances moyennes aux différentes évaluations (Note_TP) selon le genre. Utile pour détecter d'éventuelles disparités ou biais de performance selon le genre.

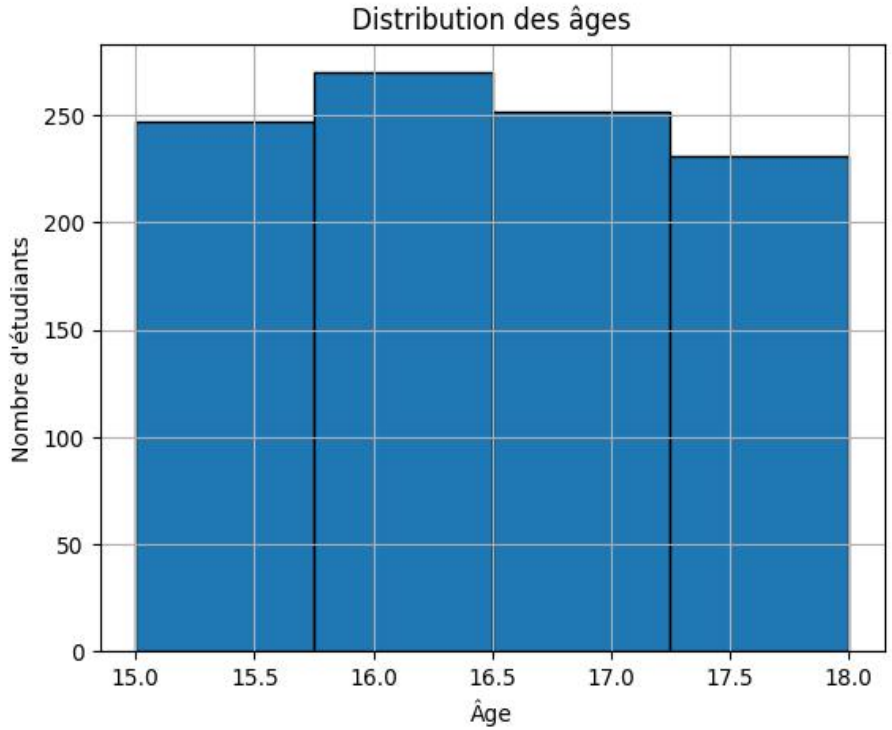
```
import matplotlib.pyplot as plt
# Calcul des moyennes par genre
mean_notes = data.groupby('genre')['Note_TP'].mean()
# Exemple avec la moyenne de Note_CC uniquement
x = mean_notes.index
y = mean_notes
plt.bar(x, y)
plt.title("Moyenne de Note_CC par genre")
plt.xlabel("Genre")
plt.ylabel("Note_CC Moyenne")
plt.grid(True)
plt.show()
```



Tracé de diagrammes en bâtons et histogrammes

- Histogram : Distribution des âges
- Visualiser la répartition des étudiants selon leur âge. Cela permet de connaître les classes d'âge dominantes et d'ajuster les méthodes pédagogiques selon les tranches d'âge.

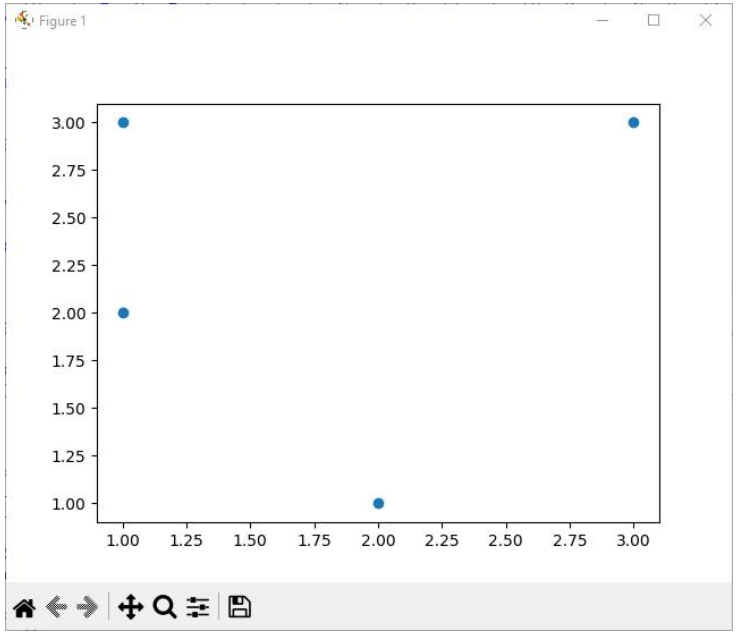
```
import matplotlib.pyplot as plt
plt.hist(data['age'], bins=4, edgecolor='black')
plt.title('Distribution des âges')
plt.xlabel('Âge')
plt.ylabel('Nombre d\'étudiants')
plt.grid(True)
plt.show()
```



Affichages de nuages de points

- `plt.scatter(abscisses, ordonnées)`
- Exemple

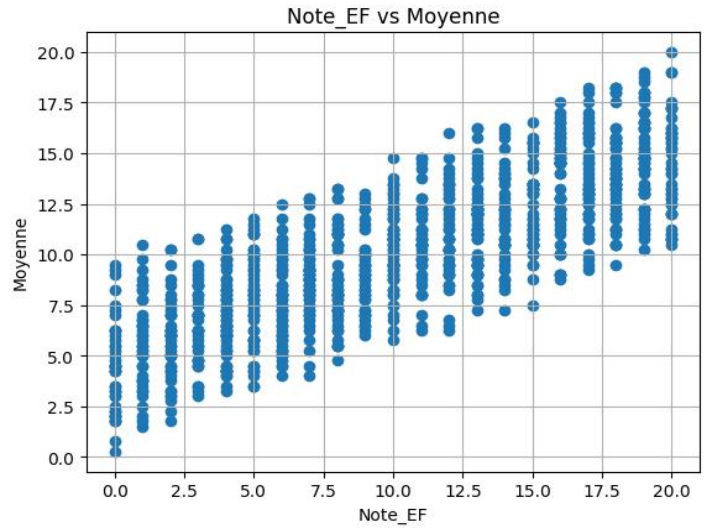
```
import matplotlib.pyplot as plt
import numpy as np
x = [ 1, 3, 2, 1]
y = [ 2, 3, 1, 3]
plt.scatter(x,y)
plt.show()
```



Affichages de nuages de points

- Objectif d'analyse :
- Observer s'il existe une corrélation entre les notes de l'examen final (EF) et la moyenne genrele (Moyenne). Cela permet de savoir si les étudiants qui réussissent en EF ont aussi tendance à bien performer en Moyenne.

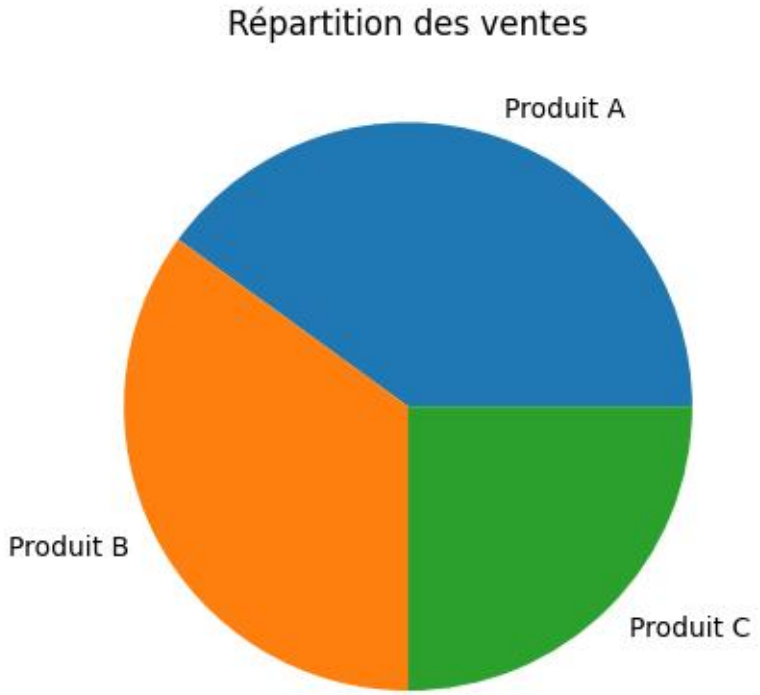
```
import matplotlib.pyplot as plt

data["Moyenne"] = data["Note_CC"] * 0.25 + data["Note_TP"] * 0.25 + data["Note_EF"] * 0.50

plt.scatter(data['Note_EF'], data['Moyenne'])

plt.title('Note_EF vs Moyenne')

plt.xlabel('Note_EF')

plt.ylabel('Moyenne')

plt.grid(True)

plt.show()
```



pie

- Un graphique en secteurs (*pie chart*) est utilisé pour représenter des proportions ou des parts d'un tout. Chaque part (secteur) correspond à un pourcentage de la somme totale.

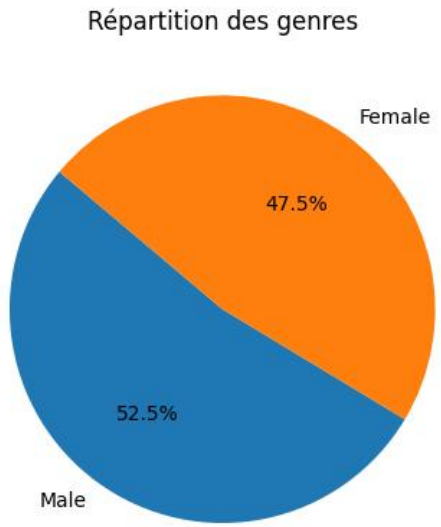
```
import matplotlib.pyplot as plt
# Données
lab = ['Produit A', 'Produit B', 'Produit C']
valeurs = [40, 35, 25]
# Graphique
plt.pie(valeurs, labels=lab)
plt.title("Répartition des ventes")
plt.show()
```



pie

- Objectif d'analyse :
- Mesurer la proportion de chaque genre dans l'échantillon d'étudiants. Cette information est importante pour comprendre la composition démographique de la classe ou du groupe d'étude.

```
import matplotlib.pyplot as plt
genre_counts = data['genre'].value_counts()
plt.pie(genre_counts, labels=genre_counts.index, autopct='%1.1f%%',  startangle=140)
plt.title('Répartition des genres')
plt.show()
```



Seaborn

Générez des graphiques complexes

Seaborn

- La librairie Seaborn vient proposer une alternative à Matplotlib. C'est également une librairie permettant de générer des graphiques, tout comme Matplotlib
- il propose de multiples modèles graphiques prédéfinis de bonne qualité esthétique, en modifiant les options graphiques par défaut de Matplotlib ;
- il ajoute une interaction avec les data frames afin de faciliter grandement la génération de graphiques à partir de ceux-ci ;
- il propose un catalogue – très – dense de fonctions graphiques pour répondre le plus précisément possible à une problématique donnée
- Seaborn étant une surcouche de Matplotlib, il y a donc de nombreuses ressemblances entre les deux librairies

Utilisation

- Installation de Seaborn

```
python -m pip install seaborn
```

- Importation des bibliothèques

```
import seaborn as sns
import matplotlib.pyplot as plt
```

- Chargement des données

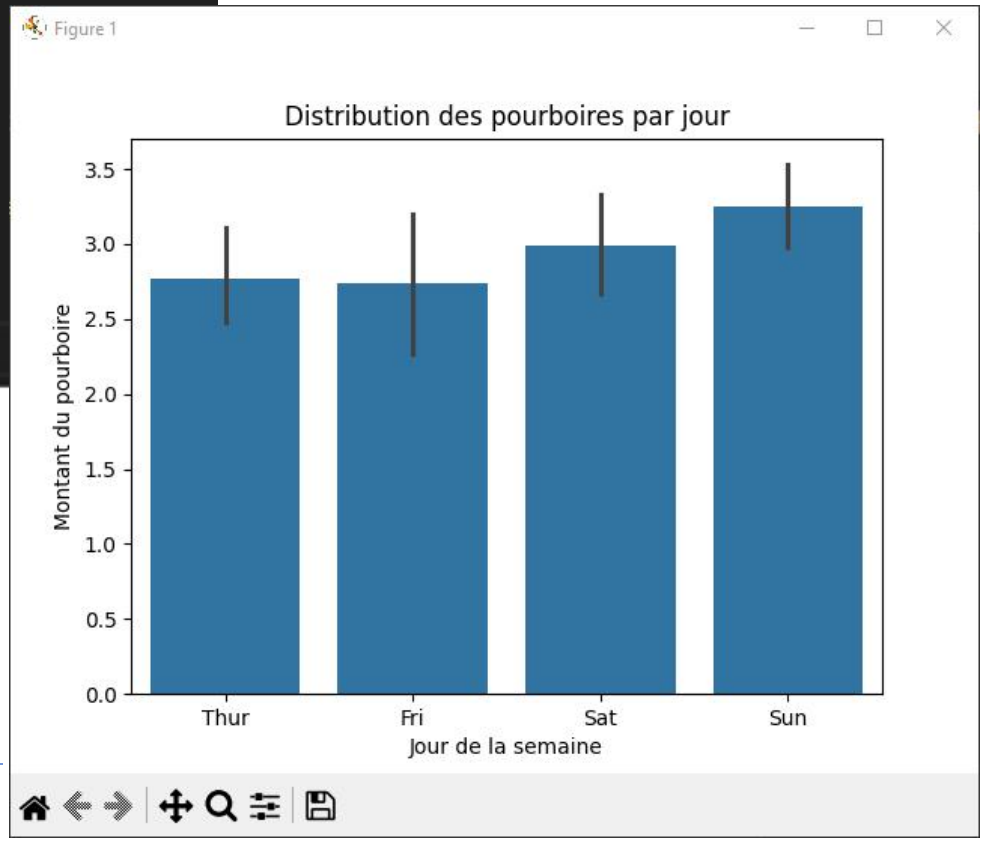
```
tips_data = sns.load_dataset('tips')
```

```
>>> tips_data
     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female     No   Sun  Dinner     2
1         10.34  1.66    Male     No   Sun  Dinner     3
2         21.01  3.50    Male     No   Sun  Dinner     3
3         23.68  3.31    Male     No   Sun  Dinner     2
4         24.59  3.61  Female     No   Sun  Dinner     4
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner     3
240       27.18  2.00  Female    Yes   Sat  Dinner     2
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2

[244 rows x 7 columns]
```

Exemples de visualisations avec Seaborn

- Diagramme en barres - Distribution des pourboires par jour

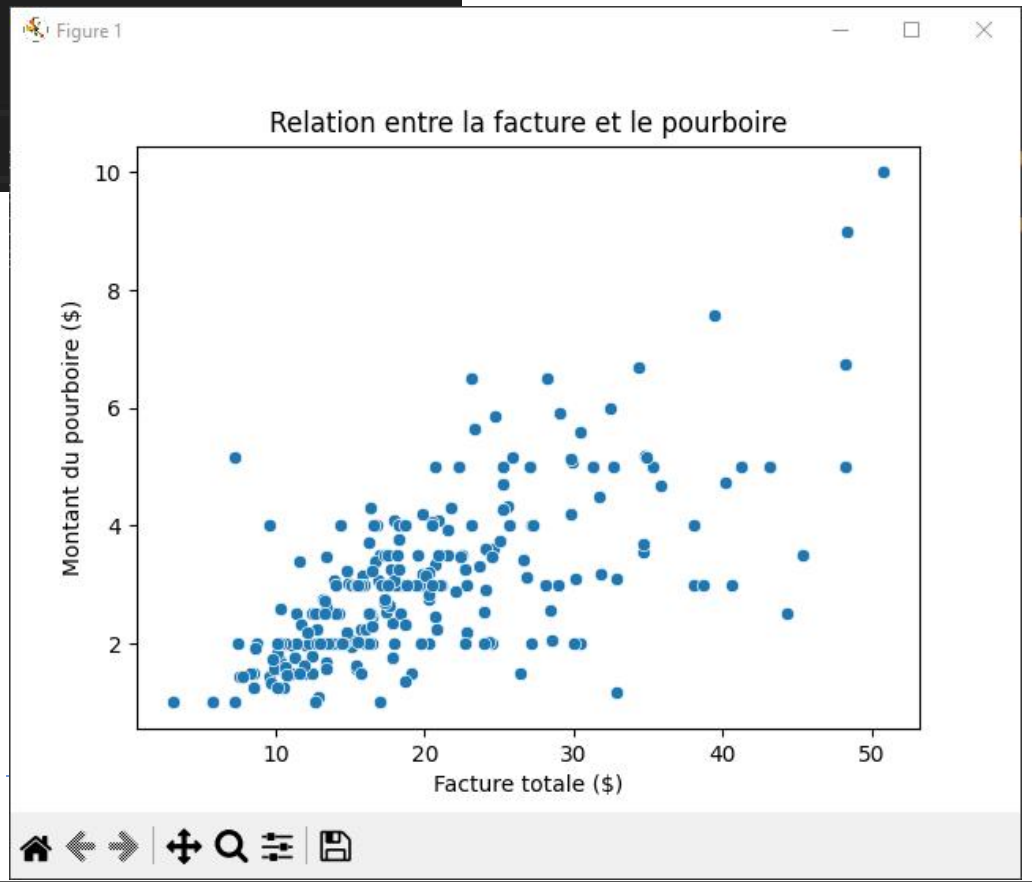
```python
import seaborn as sns
import matplotlib.pyplot as plt
tips_data = sns.load_dataset('tips')
sns.barplot(x='day', y='tip', data=tips_data)
plt.title('Distribution des pourboires par jour')
plt.xlabel('Jour de la semaine')
plt.ylabel('Montant du pourboire')
plt.show()
```



Exemples de visualisations avec Seaborn

- Nuage de points – Relation entre la facture et le pourboire

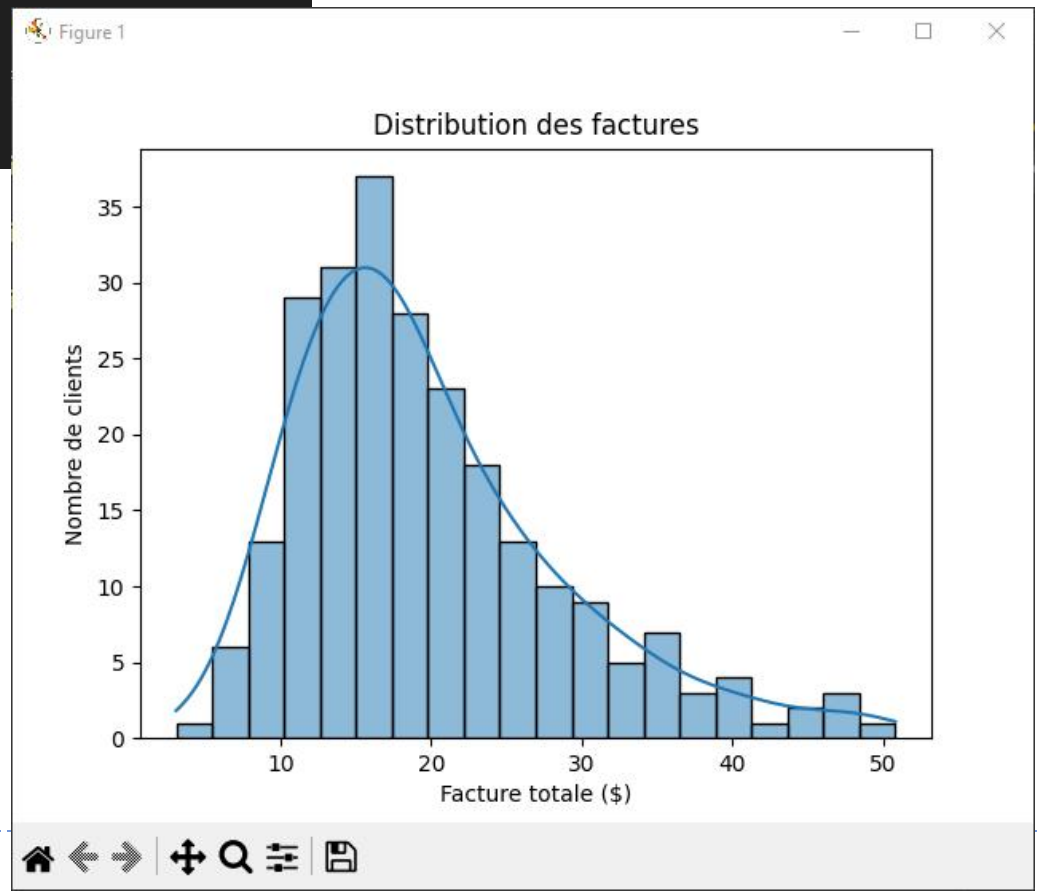
```
sns.scatterplot(x='total_bill', y='tip', data=tips_data)
plt.title('Relation entre la facture et le pourboire')
plt.xlabel('Facture totale ($)')
plt.ylabel('Montant du pourboire ($)')
plt.show()
```



Exemples de visualisations avec Seaborn

- Histogramme - Distribution des factures

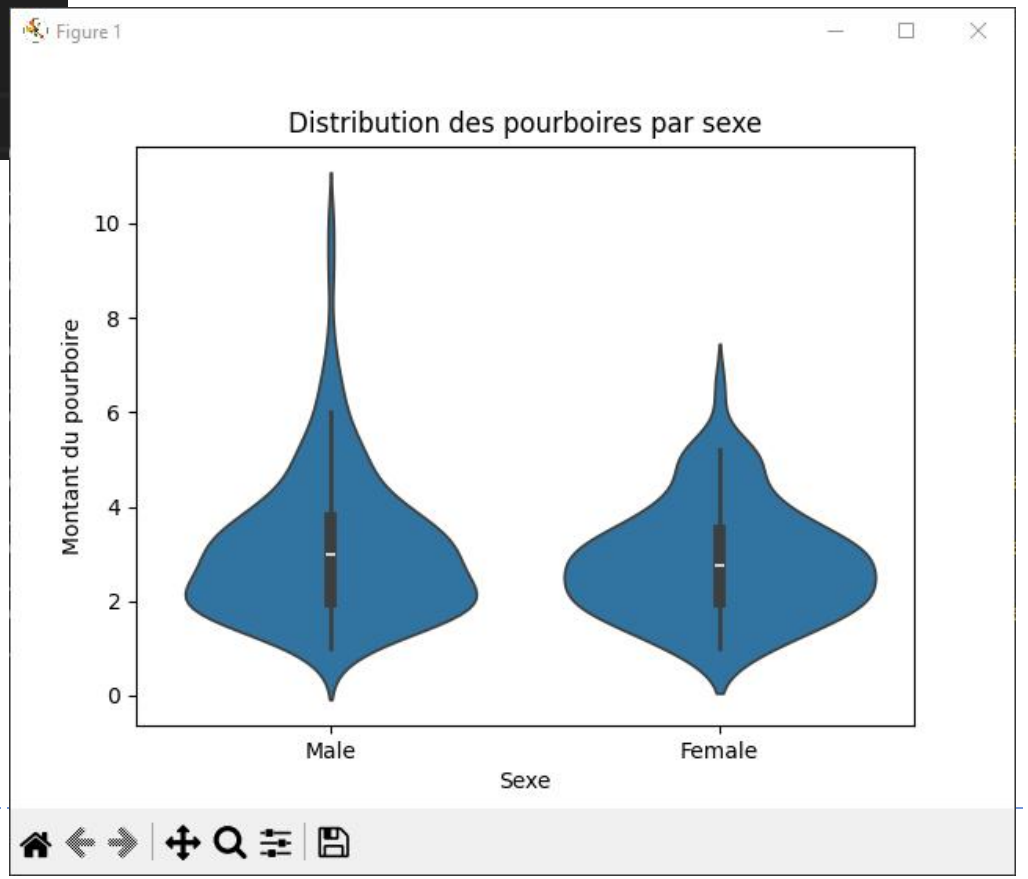
```
sns.histplot(tips_data['total_bill'], bins=20, kde=True)
plt.title('Distribution des factures')
plt.xlabel('Facture totale ($)')
plt.ylabel('Nombre de clients')
plt.show()
```



Exemples de visualisations avec Seaborn

- Diagramme en violon - Distribution des pourboires par sexe

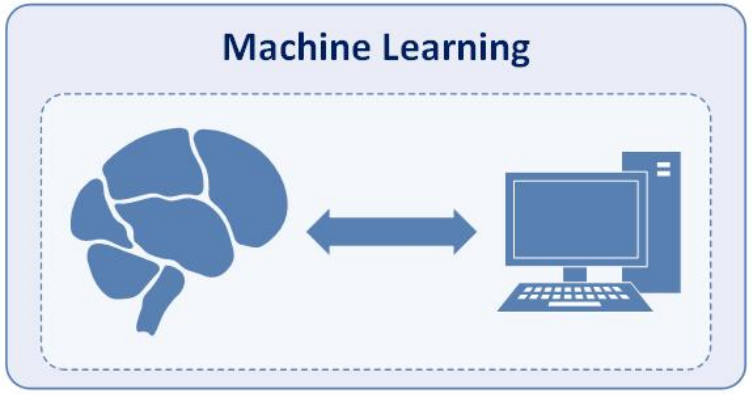
```
sns.violinplot(x='sex', y='tip', data=tips_data)
plt.title('Distribution des pourboires par sexe')
plt.xlabel('Sexe')
plt.ylabel('Montant du pourboire')
plt.show()
```

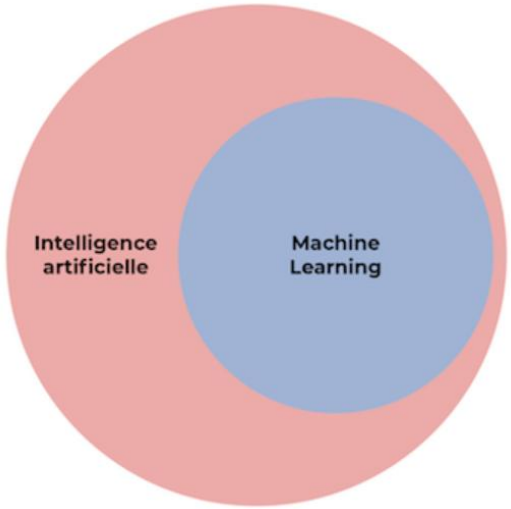


Machine Leraning et L'analyse prédictive

Machine Learning (Apprentissage automatique) : Définition

- Qu'est-ce que c'est l'apprentissage (Learning) ?
- Dans le contexte de l'intelligence artificielle (IA), **l'apprentissage** désigne le processus par lequel un système ou un modèle informatique améliore ses performances sur une tâche donnée à partir de données ou d'expériences, sans être explicitement programmé pour chaque situation.
- Cela correspond à ce que l'on appelle généralement **l'apprentissage automatique** (ou **machine learning**).



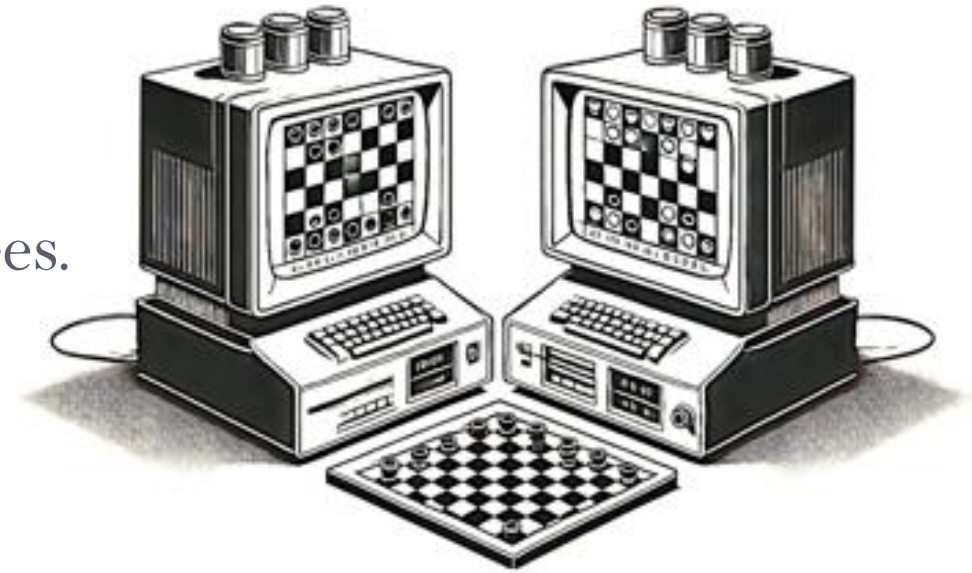


Le Machine Learning est une sous-discipline de l'IA

Machine Learning (Apprentissage automatique) : Définition

- Le Machine Learning a une tâche précise à accomplir : prédire.

- **Processus** :
  - Collecte de données.
  - Création d'un modèle basé sur ces données.
  - Évaluation et amélioration du modèle.



APPRENTISSAGE automatique : Modèle

- L'apprentissage automatique (ML) nous permet de construire des systèmes informatiques qui apprennent tout seuls à partir des données qu'ils utilisent. Encore mieux : ils sont capables d'améliorer leur performance au cours du temps, en s'enrichissant de nouvelles données.
- Une fois que le problème est défini, le programme d'apprentissage automatique va avoir besoin d'un modèle sur lequel s'appuyer.

Un modèle est une représentation mathématique d'un problème donné.

Fonctionnement du machine learning

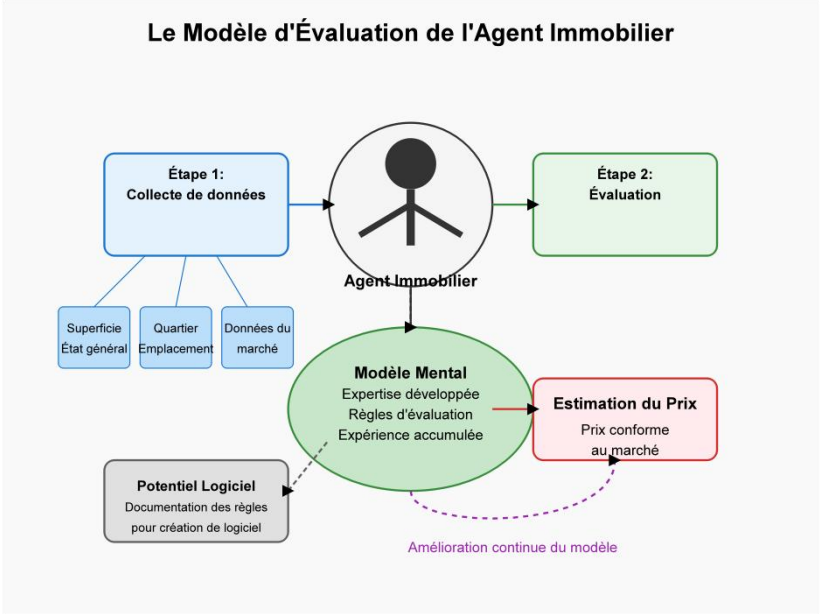
- L'idée centrale du machine learning est d'apprendre à partir de données.
- Le Machine Learning utilise des algorithmes pour développer des modèles prédictifs à partir de jeux de données (datasets, en anglais)
- Le machine learning repose sur deux piliers fondamentaux :
  - — Les données, qui sont les exemples à partir duquel l'algorithme va apprendre ;
  - — L'algorithme d'apprentissage, qui est la procédure que l'on fait tourner sur ces données pour produire un modèle. On appelle **entraînement** le fait de faire tourner un algorithme d'apprentissage sur un jeu de données.

APPRENTISSAGE automatique : Modèle

Exemple : un problème que pourrait rencontrer un agent immobilier : faire estimer un bien au prix de vente le plus conforme au marché immobilier. Pour réaliser cette évaluation, notre professionnel de l'immobilier va effectuer deux étapes :

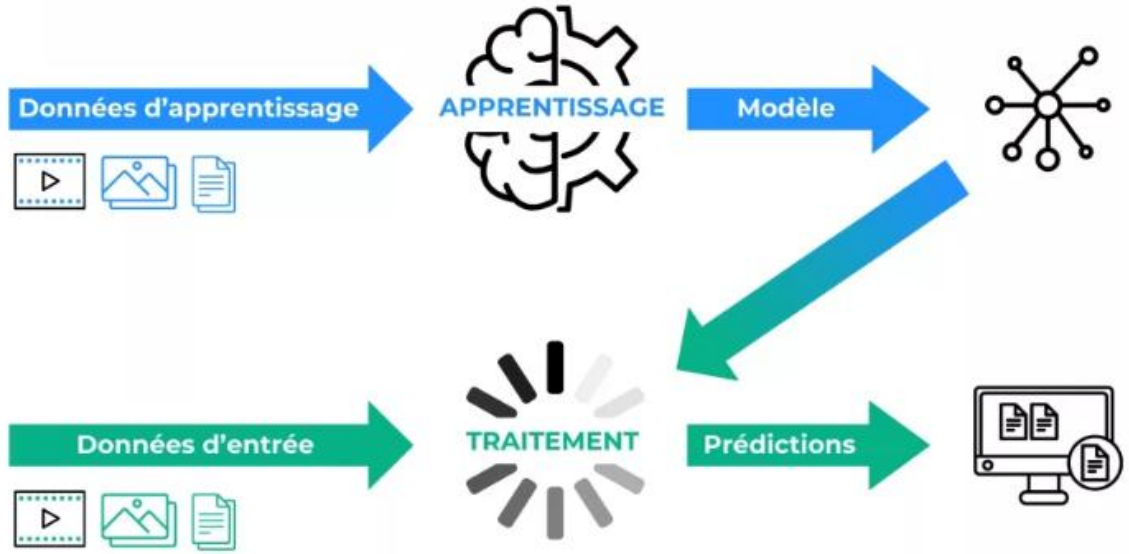
1. **il recueille des données** sur les caractéristiques clés du bien immobilier (par exemple : l'emplacement géographique, la superficie, l'état général, etc.) ;
2. **il procède ensuite à une évaluation** fondée sur des données publiques disponibles, ainsi que sur sa propre expertise immobilière.

Au fil du temps, il va pouvoir développer sa connaissance du marché qu'il cible. Ainsi, l'agent, fort de toutes ses connaissances immobilières et des données disponibles, va devenir de plus en plus apte à fournir une évaluation de bien. On pourrait dire qu'il a développé un modèle **d'évaluation des prix.**



Fonctionnement du machine learning

- Le machine learning consiste à entraîner un algorithme au sein d'une base d'apprentissage. On lui fait reconnaître des motifs récurrents ou « patterns » pour aboutir à un modèle réalisant des prédictions.
- Une fois ce modèle développé, celui-ci est sollicité par la machine lors de traitements de nouvelles données, pour aboutir à une réponse ou à une action finale. Au fur et à mesure des entraînements successifs et grâce à l'évolution du contexte, l'algorithme améliore ses performances.
- Tout ce processus à lieu automatiquement et vous n'aurez qu'à renseigner les données initiales pour l'apprentissage ! Plus on le nourrit et plus il devient précis.



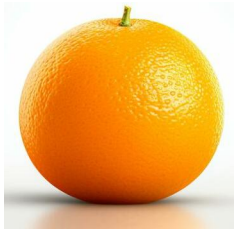
Machine Learning (Apprentissage automatique)

1. Données comme exemples d'apprentissage

2. Apprentissage à partir des données

Si un fruit est rouge et rond, c'est peut-être une pomme

3. Évaluation des réponses



nouvelle image qu'elle n'a jamais vue

4. Amélioration continue

Introduction à la Modélisation Prédictive

- La modélisation prédictive, ou Predictive Modeling en anglais, regroupe un ensemble de méthodes permettant de collecter et d'analyser des données définies, de manière à les interpréter pour en déduire des **pronostics** concernant des tendances futures, des événements à venir ou bien le comportement des consommateurs à l'avenir.
- La modélisation prédictive consiste à utiliser des données historiques pour anticiper des comportements ou des résultats futurs. Cette approche s'appuie sur des outils mathématiques et des algorithmes pour fournir des insights exploitables.

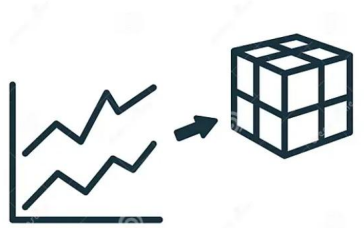
Exemples

Imaginons que vous soyez responsable des données d'une plateforme de contenu en ligne dont le business model repose sur le nombre d'abonnements souscrits. On vous pose 2 questions :

- Quel est le profil des utilisateurs qui s'abonnent ? → **La modélisation statistique**.
- Comment prédire si un nouvel utilisateur va s'abonner ?→ **La modélisation prédictive** (machine learning)

Applications :

- Marché : Prévision de la demande et des prix de vente
- Comptabilité : Prévision des flux de trésorerie
- Économie : Prévision du taux de chômage ou de croissance



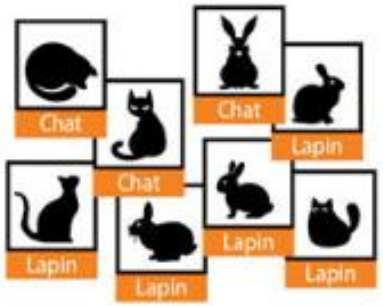
Dataset

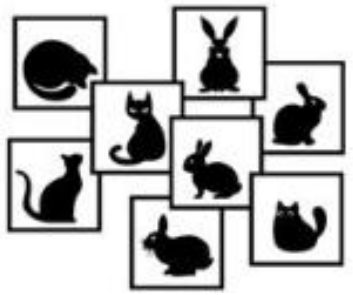
Données = carburant du ML

- Tout modèle prédictif repose sur un jeu de données. **Sans données, pas de Machine Learning.**
- Comme le suggère la définition proposée par Wikipedia, les algorithmes de l'apprentissage automatique sont basés sur des données. On parle aussi **d'échantillons** (samples), **d'observations**, ou **d'exemples**. → données (dataset).
- Deux grandes familles de jeux de données peuvent être utilisées :

  — les données **étiquetées** : chaque observation x_n est fournie avec une étiquette (label) y_n que l'on cherche à prédire ;

  — les données **non-étiquetées** : comme le nom l'indique, aucune étiquette n'est fournie.





Dataset

- Dans l'apprentissage supervise le dataSet contient toujours deux types de variables :
  - **La variable cible**, sujet de la prédiction (y : Target).
  - **Les autres variables** potentiellement prédictrices (x : Features).
- Par exemple, prenons un jeu de données comprenant l'âge, la taille et le poids d'une centaine de collégiens. Si on souhaite **prédire le poids** des enfants en fonction de leur **taille** et de leur **âge**, la variable cible sera le poids et les variables prédictrices seront l'âge et la taille.

Technique de la Modélisation Prédictive

Régression linéaire

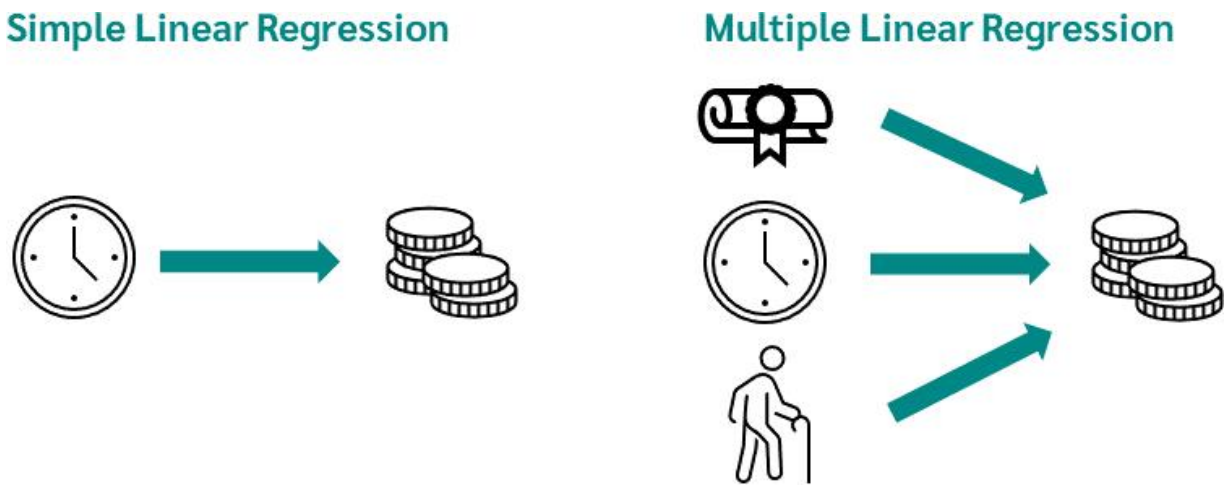
- Modèle qui prédit une valeur numérique continue à partir de variables explicatives, en supposant une relation linéaire entre elles.
- **Exemple** : prédire le prix d'une maison en fonction de sa surface.

K-Nearest Neighbors (k-NN)

- Algorithme de classification ou régression qui prédit la sortie d'un point en se basant sur les k points les plus proches dans les données d'entraînement.
- **Exemple** : prédire si un client achètera (Oui/Non) selon les comportements de ses k voisins les plus proches.

Technique de la Modélisation Prédictive - **Régression linéaire**

- La régression linéaire est une méthode statistique utilisée pour modéliser la relation entre une variable dépendante (y) et une ou plusieurs variables indépendantes (x).
- L'objectif est de trouver une relation linéaire qui permet de prédire les valeurs futures de $\boldsymbol{y}$ à partir de $\boldsymbol{x}$.
- Selon qu'il y a une ou plusieurs variables indépendantes, on distingue l'analyse de régression linéaire simple et l'analyse de régression linéaire multiple.



Technique de la Modélisation Prédictive - **Régression linéaire**

Régression Linéaire Simple

- Une seule variable indépendante est utilisée pour prédire y.
- Exemple : Prédire le prix d'une action en fonction du temps.
- Formule

$$y = \beta_0 + \beta_1 x + \epsilon$$

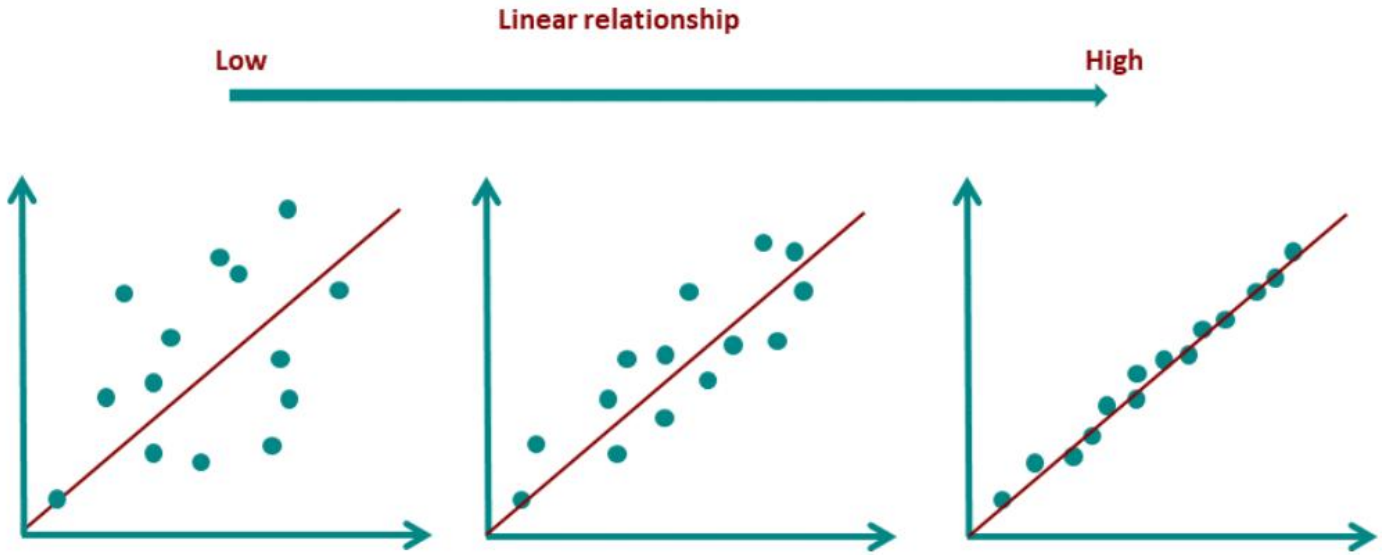
où :

- β_0 : intercept (valeur de y quand $x = 0$),
- β_1 : coefficient de régression (pente de la droite),
- ϵ : terme d'erreur (représente les variations non expliquées).

Technique de la Modélisation Prédictive - **Régression linéaire**

Régression Linéaire Simple

- L'objectif d'une régression linéaire simple est de prédire la valeur d'une variable dépendante en fonction d'une variable indépendante.
- Plus la relation linéaire entre la variable indépendante et la variable dépendante est grande, plus la prédiction est précise.
- Visuellement, la relation entre les variables peut être représentée par un diagramme de dispersion. Plus la relation linéaire entre les variables dépendantes et indépendantes est importante, plus les points de données se situent sur une ligne droite



Technique de la Modélisation Prédictive - **Régression linéaire**

Régression linéaire multiple

- Plusieurs variables indépendantes sont utilisées pour prédire *y*.
- Exemple : Prédire le rendement d'un portefeuille basé sur plusieurs facteurs économiques (taux d'intérêt, inflation, volatilité).
- Formule :

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$

où :

- $x_1, x_2, \ldots, x_n$: variables explicatives.

Technique de la Modélisation Prédictive - **Régression linéaire**

Régression linéaire multiple

- Contrairement à la régression linéaire simple, la régression linéaire multiple permet de prendre en compte plus de deux variables indépendantes.
- La variable à estimer est appelée variable dépendante (critère). Les variables utilisées pour la prédiction sont appelées variables indépendantes (prédicteurs).
- La régression linéaire multiple est fréquemment utilisée dans la recherche sociale empirique ainsi que dans les études de marché. Dans ces deux domaines, il est intéressant de déterminer l'influence de différents facteurs sur une variable. Par exemple, quels sont les déterminants qui influencent la santé ou le comportement d'achat d'une personne ?

Simple Linear Regression

$$\hat{y} = b \cdot x + a$$

➡

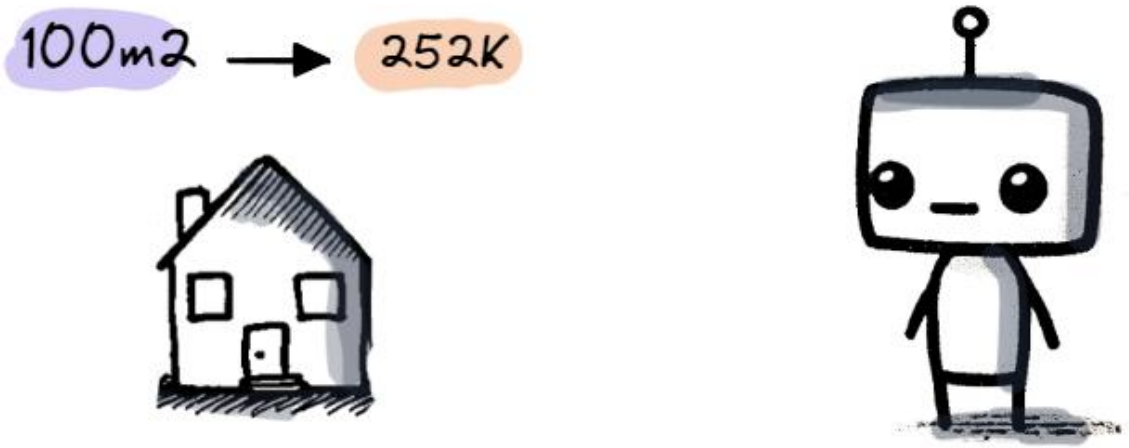
Multiple Linear Regression

$$\hat{y} = b_1 \cdot x_1 + b_2 \cdot x_2 + \ldots + b_k \cdot x_k + a$$

Régression : Exemple illustratif

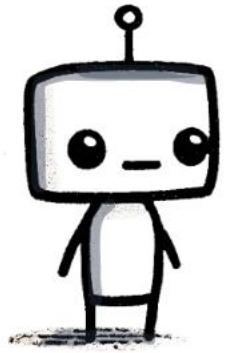
- Imaginez que vous souhaitiez prédire le **prix** d'une maison en fonction de sa surface habitable.
- Pour ce faire, on montre des exemples de maisons à notre machine.

Voici une maison, sa surface habitable est de 100 m2 et son prix est de 252 000 euros.



Régression : Exemple illustratif

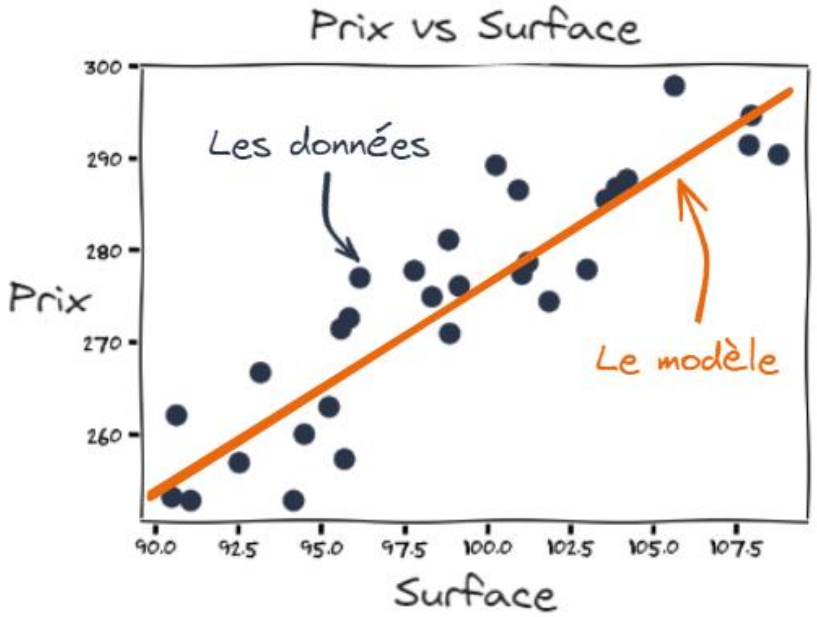
- Dans l'apprentissage supervisé, ces exemples de questions/réponses sont présentés à la machine sous forme de jeu de données (X, Y), où X représente les variables d'entrée, et Y la sortie attendue.



	X	Y
	92	259K
	95	237K
	98	272K
	93	234K
	100	252K
	110	300K

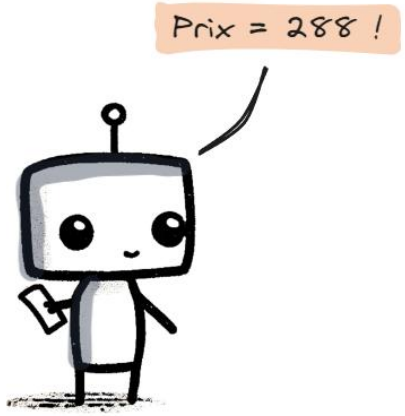
Régression : Exemple illustratif

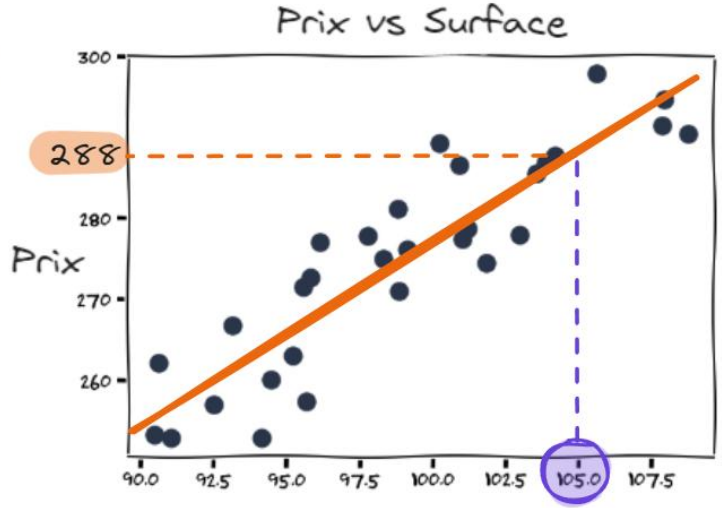
- Grâce à ce jeu de données, la Machine est capable d'apprendre un **modèle** permettant de prédire la valeur de Y en fonction de X. Pour ce faire, elle effectue une auto-évaluation en recherchant le modèle qui lui offre les meilleures performances par rapport au jeu de données fourni.



Régression : Exemple illustratif

- Une fois ce modèle développé, il est possible de s'en servir pour faire de futures **prédictions**.
- Par exemple, si nous avons une nouvelle maison, dont la surface habitable est de 105 m2 :





Alors notre machine peut utiliser notre modèle pour prédire le prix de cette nouvelle maison.

Régression

- — 1. Prévision du cours d'une action
- — 2. Évaluation du risque de crédit
- — 3. Modélisation de la relation risque-rendement (CAPM)
- — 4. Prévision des taux d'intérêt
- — 5. Prévision des ventes
- — 6. Prévision du bénéfice net
- — 7. Estimation des amortissements futurs
- — 8. Prévision du budget de fonctionnement
- — 9. Détection d’anomalies comptables
- ....

LAB

- Apprendre à utiliser la régression linéaire multiple pour prédire le nombre de ventes en fonction des dépenses publicitaires sur différents canaux (TV, radio, journaux).

tv	radio	journaux	ventes
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	9.3
151.5	41.3	58.5	18.5
180.8	10.8	58.4	12.9

FIN