Python programming

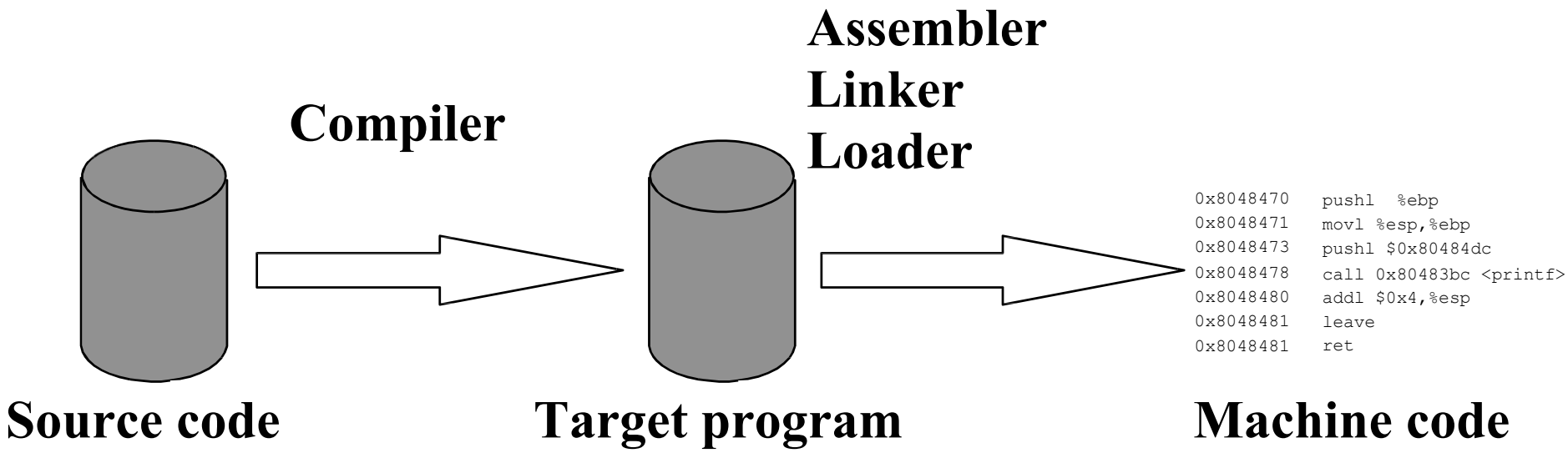
Python

- Created by Guido van Rossum
  - First version released in 1991
- Simple yet powerful language

  Allows writing basic scripts
- Suitable for complex projects thanks to its many libraries
- Interpreted language

  No compilation needed before execution

Compiled language

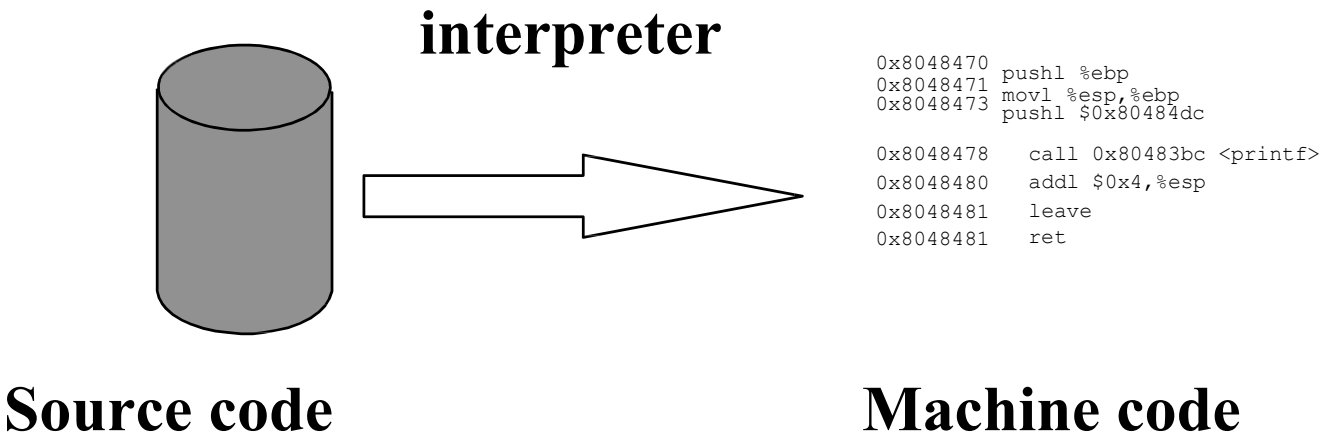


$ Open (or edit) `monProg.cpp`

$ Compile `monProg.cpp EN monProg`

$ Execute `monProg`

Interpreted language (ex: Matlab)



- Open (ou edit) monProg.py
- Interpreter monProg.py (python monProg.py)

Why prefer Python over other languages?

- Python is an easy-to-learn language, and its code is more readable, making it easier to maintain. It is sometimes up to 5 times more concise than Java, for example, which increases developer productivity and reduces the number of bugs.

- Python offers a very rich ecosystem for data science.
- The Python documentation is also extremely well-written, catering to both beginners and experts.
- Google, Yahoo, Microsoft, and NASA claim to use Python.
- Python is also very popular in the field of AI due to its high-quality libraries and computational power.

Installation and configuration

- Download: Go to the official Python website: ***python.org***
- Installation :
  - Windows: Download the installer, check the option "Add Python to PATH", then follow the instructions
  - Linux/MacOS : Python is often pre-installed. If not, use the package manager:
    - ✓ Linux : sudo apt install python3
    - ✓ macOS : brew install python

Choosing an IDE (Integrated Development Environment)

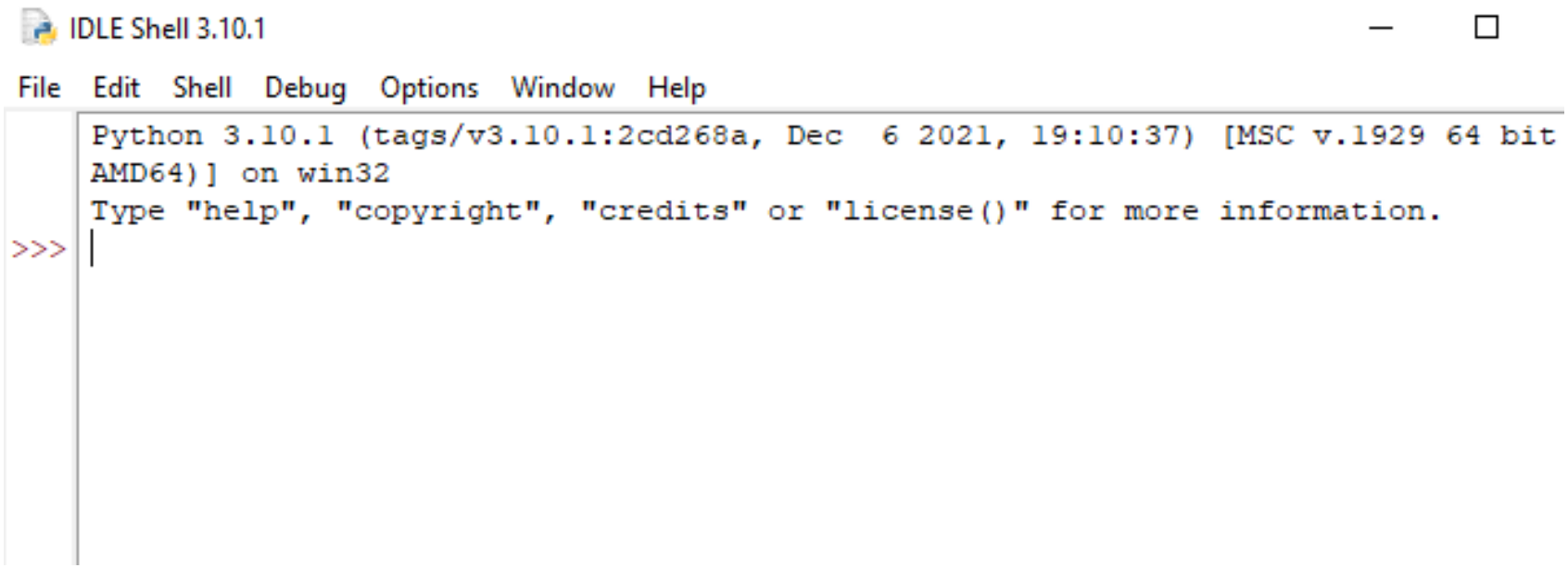
- Some popular IDEs for Python:
  - PyCharm :A full-featured IDE with powerful debugging tools.
  - VS Code : Lightweight and customizable, with extensions like the Python Extension.
  - Jupyter Notebook : Ideal for scientific or educational projects..
  - Thonny : Designed for beginners.
  - .

Getting Started with the Python Interpreter

- Starting the Interpreter:
  - Open a terminal or command prompt.
  - Type python or python3 (depending on the version).
  - You should see the interactive prompt >>>..
- Running a Python Script:
  - Create a file named script.py with the following content:
    - ✓ print("Welcome to Python !")
  - In the terminal, run : python script.py

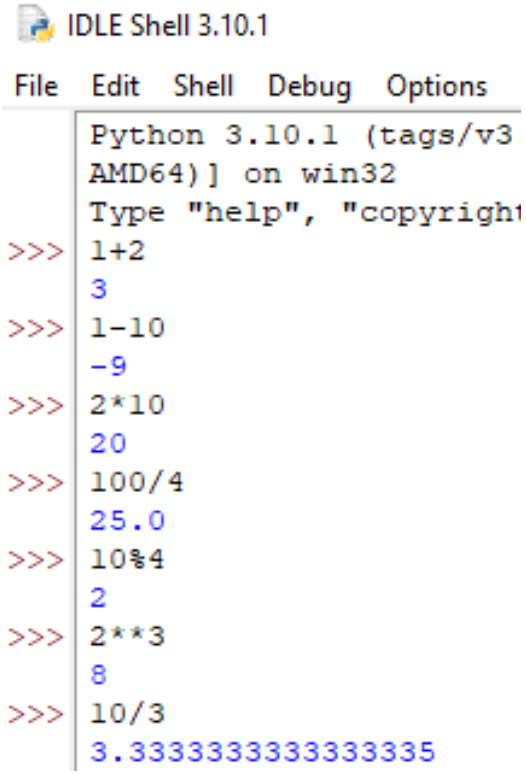
Programming in Python

- Two execution modes:
  - Use the Python interpreter (interactive mode)
  - Write a program in a file and run it (script mode)



Calculations

- One of the first features of an interpreter is performing calculations.
- All operators can be used.
- The double asterisk (**) represents exponentiation



Variables

- A variable is a memory location in the computer where a value is stored.
- A variable is created when you assign a value to it for the first time.

```
x = 5
y = "John"
print(x)
print(y)
```

- The print() function allows you to display strings or any other object.
- Variables do not need to be declared with a specific type and can even change type after being defined.

```
x = 4          # x is of type int
x = "Sally"    # x is now of type str
print(x)
```

Variables types

- The type of an object defines how it is represented in memory.
- The computer's memory is made up of a continuous sequence of bytes.
- The basic types in Python relate to strings, integers, and floating-point numbers (real numbers):
  - ✓ Les type caractères : Le mot-clef **str**
  - ✓ Character type: The keyword str
  - ✓ Integer type: int
  - ✓ Floating-point type: float
  - ✓ Boolean type: bool
  - ✓ Binary types: bytes
  - ✓ No type: NoneType

Variables

- To find out the type of a variable, you can use the type() function.

```
>>> v = 15
>>> type(v)
<class 'int'>
>>> v = "Bonjour"
>>> type(v)
<class 'str'>
>>> v = 3.1
>>> type(v)
<class 'float'>
```

Variables

❖ Castign :

If you want to specify the data type of a variable, you can do it using type casting

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

Operators

- Arithmetic:
  - + : addition
  - - : soustraction
  - * : multiplication
  - / : division (résultat float)
  - // : division entière
  - % : modulo (reste de division)
  - ** : puissance

Operators

- **Logical Operators :**
  - and: true if all conditions are true
  - or: true if at least one condition is true
  - not: inverts a condition
- **Comparatifs :**
  - ==: equal to
  - !=: not equal to
  - <, >, <=, >=: less than, greater than (or equal to)

Operators

- A boolean expression can contain the following comparison operator

Operator	Explication
x == y	x is equal to y
x != y	x is not equal y
x > y	x is greater than y
x < y	x is less than y
x >= y	x is greater or equal y
x <= y	x is less or equal y

- Exemple :

```
>>> 3 < 4
True
>>> 3 > 4
False
```

Instruction if ... else

- Logical operator and :

```
>>> 2 < 3 and 3 < 4
True
>>> 2 < 3 and 3 > 4
False
```

- Logical operator or

```
>>> 2 < 3 or 3 < 4
True
>>> 2 < 3 or 3 > 4
True
```

Input-Output

- The simplest way to produce output is by using the print() function.
- In Python, we have the input() function to allow the user to enter dynamic values.

```
>>> a
'5'
```

```
>>> a = int(input('Saisir une valeur '))
Saisir une valeur 5
>>> a
5
```

Instructions conditionnelles

- The if statement is the simplest control structure. Its syntax in Python involves the concept of a block. Let's first explore this concept in more detail.
- Bloc of instructions (Indentation) : A block is defined by indentation, achieved by shifting the beginning of the instructions to the right using spaces at the start of the line. All instructions within the same block must be indented exactly at the same level (i.e., shifted to the right by the same number of spaces). A block can contain one or more instructions, including compound instructions (tests, loops, etc.).

Instruction if

- Syntaxe :

```
if condition:
    Instruction A
```

  - A condition is a boolean expression, meaning it is an expression that evaluates to either True or False.
    The instruction A is executed only if the condition is satisfied (i.e., if it evaluates to True).

- Example of condition true :

```
x = 15
if x > 10:
    print(x, "is greater than 10")
print("end")
```

Instruction if ... else

- Syntaxe :

```
if condition:
    Instruction A
else:
    Instruction B
```

- Exemple :

```
x = 5
if x > 0 :
    print(x, "est positif")
else :
    print(x, "est négatif ou nul")
print("Fin")
```

Instruction if ... else elif

- Condition **elif** Il est possible d'ajouter autant de conditions précises que l'on souhaite en ajoutant le mot clé elif.

```
a = 5
if a > 5:
    a = a + 1
elif a == 5:
    a = a + 1000
else:
    a = a - 1
```

Instruction if ... else

❖ Exemple :

Selon la température de l'eau

- ✓ Temp =< 0 C'est de la glace
- ✓ 0<Temp <100 C'est du liquide
- ✓ Temp > 100 C'est de la vapeur

Ecrire un programme en Python permettant de dire si il s'agit du liquide, de la glace ou de la vapeur

Loops : for loop

- The **for** statement is a compound statement, meaning a statement whose header ends with a colon, followed by an indented block that forms the body of the loop.
- Example of use:

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

- Affichage après exécution :

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

for Loop

- It is possible to achieve the same result without providing the list of values, by using the range() function..

```
for i in range(4):
    print("i a pour valeur", i)
```

- The range() function generates a list of consecutive integers and takes three parameters: range(start, stop, step). The start and step parameters are optional. By default, start = 0 and step = 1. They are specified only if they differ from the default values.

```
for i in range (0,10,2):
   print(i)

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

Iterating over a string with a for loop

- To iterate over a string with a for loop in Python, you can use the for loop structure in combination with the character index.
- Here is an example:

```
my_string= "Example of string"

for caracter in my_string:
    print(caracter)
```

for loop

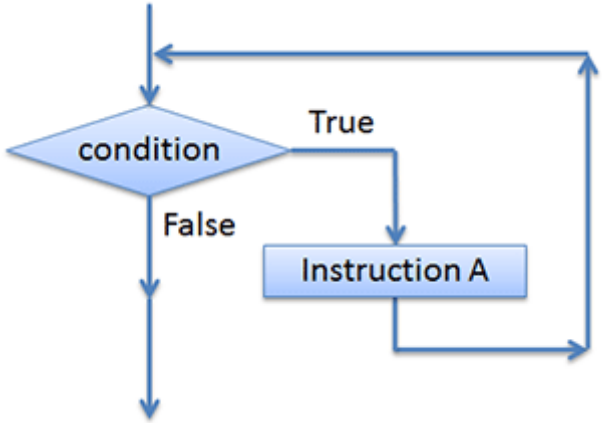
- Example :

1. Write a Python program that displays "Bonjour" 10 times.
2. Write a Python program that displays all numbers between 10 and 20.
3. Write a Python program that displays all even numbers between 0 and 100.
4. Write a Python program that displays the sum of numbers from 0 to 100.

While loop

- Syntax :

```
while condition:
    Instruction A
```



- The while keyword means 'as long as' in English. The body of the loop (i.e., the indented block of instructions) will be repeated as long as the condition is true.
- Example :

```
x = 1
while x < 10:
    print("x a pour valeur", x)
    x = x * 2
print("Fin")
```

Comment choisir entre boucle for et boucle while

In general, if we know the number of iterations to execute before starting the loop, we choose a for loop. On the other hand, if the decision to stop the loop can only be made by a test, we choose a while loop.

- Transformation of a for loop into a while loop :

```
for i in range(4):
    print("i a pour valeur", i)
```

- The program above is equivalent to :

```
i = 0
while i < 4:
    print("i a pour valeur", i)
    i = i + 1
```

Examples

- Example 1 :

  Using a while loop, write a Python program that prompts the user to enter a positive number.

- Example 2 :

Write a Python program that asks the user to enter an integer n, then use a while loop to find the smallest divisor of n (other than 1).

Les instructions break et continue

- L'instruction break L'instruction break permet de « casser » l'exécution d'une boucle (while ou for). Elle fait sortir de la boucle et passer à l'instruction suivante.

```
for i in range(10):
    print("debut iteration", i)
    print("bonjour")
    if i == 2:
        break
    print("fin iteration", i)
print("apres la boucle")
```

```
debut iteration 0
bonjour
fin iteration 0
debut iteration 1
bonjour
fin iteration 1
debut iteration 2
bonjour
apres la boucle
```

Les instructions break et continue

- L'instruction continue L'instruction continue permet de passer prématurément au tour de boucle suivant. Elle fait continuer sur la prochaine itération de la boucle.

```
for i in range(4):
    print("debut iteration", i)
    print("bonjour")
    if i < 2:
        continue
    print("fin iteration", i)
print("apres la boucle")
```

```
debut iteration 0
bonjour
debut iteration 1
bonjour
debut iteration 2
bonjour
fin iteration 2
debut iteration 3
bonjour
fin iteration 3
apres la boucle
```

Les chaînes de caractères

- En Python, les chaînes de caractères (strings) sont des séquences de caractères Unicode, et elles sont définies à l'aide de guillemets simples (') ou doubles ("). Voici quelques opérations et fonctionnalités courantes liées aux chaînes de caractères en Python :
- Création de chaînes de caractères :

```
# Utilisation de guillemets simples
chaine_simple = 'Bonjour, Python!'

# Utilisation de guillemets doubles
chaine_double = "Hello, Python!"
```

Les chaînes de caractères

- Concaténation de chaînes :

```
chaine1 = 'Bonjour'
chaine2 = 'Python'
# Concaténation avec l'opérateur '+'
resultat = chaine1 + ' ' + chaine2
print(resultat)  # Affiche : Bonjour Python
```

- Indexation et découpage (slicing) :

```
chaine = 'Python'
# Accéder à un caractère par son indice
premier_caractere = chaine[0]  # 'P'
# Découpage d'une chaîne
sous_chaine = chaine[1:4]  # 'yth'
# Utilisation d'indices négatifs
dernier_caractere = chaine[-1]  # 'n'
```

Les chaînes de caractères

- Méthodes de chaînes courantes :

```
chaine = '   Python est génial!   '

# Supprimer les espaces avant et après la chaîne
chaine_strip = chaine.strip()

# Convertir en majuscules et minuscules
chaine_majuscules = chaine.upper()
chaine_minuscules = chaine.lower()

# Remplacer des sous-chaînes
chaine_remplace = chaine.replace('Python', 'Java')

# Trouver l'indice de la première occurrence
indice = chaine.find('est')
```

Les listes

- Une liste est une structure de données qui contient une série de valeurs.
- Les listes sont utilisées pour stocker plusieurs éléments dans une seule variable.
- 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.

```
# Création d'une liste de nombres
ma_liste = [1, 2, 3, 4, 5]

# Affichage de la liste
print("Ma liste de nombres : ", ma_liste)
```

Les listes

- Les éléments de liste sont triés, modifiables et autorisent les valeurs en double.
- Les éléments de liste sont indexés, le premier élément a un index [0], Le deuxième élément a un index [1] ,etc.
- Une liste peut contenir différents types de données :
- Les éléments de liste sont indexés et vous pouvez y accéder en vous référant au numéro d'indice, le premier élément a l'indice 0.

```
thislist = [1, 2, 3, 4, 5]
print(thislist[1])
```

Les listes

- Indexation négative : L'indexation négative signifie commencer par la fin

  -1 fait référence au dernier élément, -2 fait référence à l'avant-dernier élément, etc.

```
thislist = [1, 2, 3, 4, 5]
print(thislist[-1])
```

- Vous pouvez spécifier une plage d'indice, Lors de la spécification d'une plage, la valeur renvoyée sera une nouvelle liste avec le éléments spécifiés.

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
print(thislist[:4])
print(thislist[2:])
print(thislist[-4:-1])
```

Modifier une liste

- Modifier la valeur de l'élément : Pour modifier la valeur d'un élément spécifique,

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

- Modifier une plage de valeurs d'élément

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

- Modifiez la deuxième valeur en la remplaçant par deux nouvelles Valeurs

```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
thislist[1:3] = ["watermelon"]
print(thislist)
```

Insérer des éléments

- Pour insérer un nouvel élément de la liste, sans remplacer aucune des valeurs existantes, nous pouvons utiliser la méthode insert()

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

- Ajouter des éléments : pour ajouter un élément à la fin de la liste, utilisez la méthode append()

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Ajouter des éléments

- Pour ajouter des éléments d'une autre liste à la liste active, utilisez la méthode extend().

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

- vous pouvez ajouter n'importe quel objet itérable (tuples, ensembles, dictionnaires etc.)

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

Supprimer des éléments

- La méthode remove() supprime l'élément spécifié.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

- S'il existe plusieurs éléments avec la valeur spécifiée, la méthode remove() supprime la première occurrence :

```
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```

Supprimer des éléments

- La méthode pop() supprime l'élément d'indice i :

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

- Si vous ne spécifiez pas l'index, la méthode pop() supprime le dernier élément.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

- La méthode clear() vide la liste.

Parcourir une liste

- Vous pouvez parcourir en boucle les éléments de la liste à l'aide d'une boucle for.

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
    print(x)
```

- Avec les indices

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
    print(thislist[i])
```

count

- la fonction count est une méthode des objets de type liste et est utilisée pour compter le nombre d'occurrences d'un élément particulier dans la liste.

```
L = [1, 2, 3, 1, 4, 1, 5]
NbrOccur = L.count(1)
print("Nombre d'occurrences de 1 dans la liste :", NbrOccur)
```

- Dans cet exemple, NbrOccur sera égal à 3, car l'élément 1 apparaît trois fois dans la liste.

Liste de Compréhension

- La compréhension de liste offre une syntaxe plus courte lorsque vous souhaitez créer une nouvelle liste basée sur les valeurs d'un liste existante.

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x  for x in fruits    if "a" in x]
print(newlist)
```

- La syntaxe :

```
newlist = [expression for item in iterable if condition == True]
```

```
newlist1 = [x   for x in fruits    if x != "apple"]
newlist2 = [x   for x in fruits]
newlist3 = [x   for x in range(10)]
newlist4 = [x   for x in range(10)   if x < 5]
```

Trier les listes

- La méthode sort() :

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

- Tri décroissant :

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

- Ordre inverse

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

Copier une liste

- Vous ne pouvez pas copier une liste simplement en tapant list2 = list1, car list2 ne sera qu'une référence à list1, et les modifications apportées dans list1 seront automatiquement effectuées dans list2.
- Faites une copie d'une liste avec la méthode copy()

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

- Une autre façon de faire une copie est d'utiliser la méthode intégrée list()

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Fusionner deux listes

- L'un des moyens les plus simples consiste à utiliser l'opérateur +

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
```

- utilisez la méthode extend() pour ajouter list2 à la fin de list1 :

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
list1.extend(list2)
print(list1)
```

Les méthodes

- Python dispose d'un ensemble de méthodes intégrées que vous pouvez utiliser sur les listes.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Les dictionnaires

- Les dictionnaires sont utilisés pour stocker les valeurs de données dans des paires clé : valeur.
- Les dictionnaires sont des collections non ordonnées (avant Python 3.7), associant des clés à des valeurs, modifiable et qui ne Autoriser les doublons.
- Contrairement aux listes qui sont délimitées par des crochets, on utilise des accolades pour les dictionnaires.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Les dictionnaires

- Avantages :
  - Flexible : Les clés peuvent être de n'importe quel type immuable (chaînes, nombres, tuples...).
  - Performant : Les dictionnaires sont optimisés pour la recherche et la récupération rapide.

Création d'un dictionnaire

- Les dictionnaires sont des collections non ordonnées, modifiables et indexées par des clés. Ils permettent de stocker des paires clé-valeur, où chaque clé est unique.

```
# Méthode 1 : Avec des accolades
mon_dictionnaire = {"nom": "Sidi", "age": 33, "ville": "Nouakchott"}

# Méthode 2 : Avec le constructeur dict()
mon_dictionnaire = dict(nom="Sidi", age=33, ville="Nouakchott")

# Dictionnaire vide
dico_vide = {}
```

Les dictionnaires

- Accéder aux valeurs : Vous pouvez accéder à une valeur en utilisant sa clé

```
# Accéder à une valeur
print(mon_dictionnaire["nom"])  # Affiche : Sidi

# Utiliser la méthode get() pour éviter les erreurs si la clé n'existe pas
print(mon_dictionnaire.get("pays", "Clé non trouvée"))  # Affiche : Clé non trouvée
```

- Modifier ou ajouter des éléments

```
# Modifier une valeur
mon_dictionnaire["age"] = 34

# Ajouter une nouvelle clé-valeur
mon_dictionnaire["pays"] = "Mauritanie"
```

Les dictionnaires

- Supprimer des éléments Différentes méthodes permettent de supprimer des éléments

```
# Supprimer une clé spécifique
mon_dictionnaire.pop("ville")

# Supprimer la dernière paire clé-valeur (Python 3.7+)
mon_dictionnaire.popitem()

# Supprimer une clé avec del
del mon_dictionnaire["age"]

# Vider le dictionnaire
mon_dictionnaire.clear()
```

Les dictionnaires

- Vérifier la présence d'une clé que vous cherchez:

```
my_dict = {"a": 1, "b": 2, "c": 3}

# Vérifie si la clé "a" existe dans le dictionnaire
if "a" in my_dict:
    print("La clé 'a' existe dans le dictionnaire.")
else:
    print("La clé 'a' n'existe pas dans le dictionnaire.")
```

Itérer sur un dictionnaire

- Pour récupérer les clés on utilise la méthode keys.

```
# Parcourir les clés
for cle in mon_dictionnaire:
    print(cle)

# Parcourir les valeurs
for valeur in mon_dictionnaire.values():
    print(valeur)

# Parcourir les paires clé-valeur
for cle, valeur in mon_dictionnaire.items():
    print(f"{cle} : {valeur}")
```

Les dictionnaires

- Méthodes utiles :
  - keys() : Retourne les clés.
  - values() : Retourne les valeurs.
  - items() : Retourne les paires clé-valeur sous forme de tuples.
  - copy() : Crée une copie du dictionnaire.
  - update() : Met à jour le dictionnaire avec un autre dictionnaire ou des paires clé-valeur.

```
mon_dictionnaire = {"a": 1, "b": 2}
autre_dictionnaire = {"c": 3, "d": 4}
mon_dictionnaire.update(autre_dictionnaire)  # Combine les deux dictionnaires
print(mon_dictionnaire)  # Affiche : {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

Tuple

- En Python, un tuple est une structure de données similaire à une liste, mais immutable (on ne peut pas modifier son contenu après sa création). Les tuples sont souvent utilisés pour stocker des collections d'éléments qui ne doivent pas changer.
- Caractéristiques des tuples
  - Immutables : Une fois créés, leurs éléments ne peuvent pas être modifiés.
  - Indexés : Les éléments d'un tuple peuvent être accessibles via leur position (indice).
  - Hétérogènes : Un tuple peut contenir des éléments de types différents.

Tuple

❖ Pourquoi utiliser un tuple plutôt qu'une liste ?

- Sécurité : Puisqu'ils sont immuables, leur contenu ne peut pas être modifié accidentellement.
- Performance : Les tuples consomment moins de mémoire et sont plus rapides à manipuler.

Tuple

- Création d'un tuple

```
# Tuple avec des éléments
mon_tuple = (1, 2, 3)

# Tuple vide
tuple_vide = ()

# Tuple avec un seul élément (notez la virgule)
tuple_unique = (42,)

#Avec le constructeur tuple()
mon_tuple = tuple([1, 2, 3])  # À partir d'une liste
```

Tuple : Accéder aux éléments

- Les tuples sont indexés de la même manière que les listes.

```
mon_tuple = (10, 20, 30, 40)

# Accéder à un élément
print(mon_tuple[1])  # Affiche : 20

# Accès en utilisant des indices négatifs
print(mon_tuple[-1])  # Affiche : 40

# Extraire une tranche (slice)
print(mon_tuple[1:3])  # Affiche : (20, 30)
```

Opérations sur les tuples

- Bien que les tuples soient immutables, vous pouvez effectuer certaines opérations :
  - Concaténation : Combine deux tuples.
  - Répétition : Répète un tuple un certain nombre de fois.

```
t1 = (1, 2)
t2 = (3, 4)

# Concaténation
resultat = t1 + t2  # (1, 2, 3, 4)

# Répétition
resultat = t1 * 3  # (1, 2, 1, 2, 1, 2)
```

Opérations sur les tuples

- Méthodes disponibles : Les tuples possèdent peu de méthodes en raison de leur immutabilité :
  - count(x) : Retourne le nombre de fois que l'élément x apparaît.
  - index(x) : Retourne l'indice de la première occurrence de x.

```
mon_tuple = (10, 20, 30, 20)

print(mon_tuple.count(20))  # Affiche : 2
print(mon_tuple.index(30))  # Affiche : 2
```

Utilisation des tuples

- Échanger des valeurs :

```
a, b = 5, 10
a, b = b, a  # Échange les valeurs
```

- Retourner plusieurs valeurs dans une fonction

```
def calculs(x, y):
    return x + y, x * y  # Retourne un tuple

resultat = calculs(3, 4)
print(resultat)  # (7, 12)
```

- Clés immuables dans un dictionnaire

```
mon_dictionnaire = {(1, 2): "point", (3, 4): "autre point"}
```

Fonctions

- Une fonction (ou function) est une suite d'instructions que l'on peut appeler avec un nom.
- La syntaxe Python pour la définition d'une fonction est la suivante :

```
def nom_fonction(liste de paramètres):
    bloc d'instructions
```

- Vous pouvez choisir n'importe quel nom pour la fonction que vous créez, à l'exception des mots-clés réservés du langage
- Comme les instructions **if**, **for** et **while**, l'instruction **def** est une instruction composée. La ligne contenant cette instruction se termine obligatoirement par un deux-points :, qui introduisent un bloc d'instructions qui est précisé grâce à l'indentation. Ce bloc d'instructions constitue le **corps de la fonction**.

Fonctions

- Fonction sans paramètre :

```
def compteur3():
    i = 0
    while i < 3:
        print(i)
        i = i + 1

print("bonjour")
compteur3()
compteur3()
```

```
bonjour
0
1
2
0
1
2
```

Fonctions

- Exemple de fonction qui appelle une autre fonction :

```
def compteur3():
    i = 0
    while i < 3:
        print(i)
        i = i + 1

def double_compteur3():
    compteur3()
    compteur3()

print("bonjour")
double_compteur3()
```

Fonctions

- Fonction avec paramètre :

```
def compteur(stop):
    i = 0
    while i < stop:
        print(i)
        i = i + 1

compteur(4)
compteur(2)
```

Fonction avec plusieurs paramètres

- La fonction suivante utilise trois paramètres : start qui contient la valeur de départ, stop la borne supérieure exclue comme dans l'exemple précédent et step le pas du compteur.

```
def compteur_complet(start, stop, step):
    i = start
    while i < stop:
        print(i)
        i = i + step

compteur_complet(1, 7, 2)
```

```
1
3
5
```

Fonction récursive

- Une fonction récursive est une fonction qui s'appelle elle-même jusqu'à ce qu'elle ne le fasse plus.
- La fonction fn() suivante est une fonction récursive car elle a un appel à elle-même.
- Pour pouvoir s'arrêter, une fonction récursive doit avoir une condition d'arrêt. Nous devons donc ajouter une instruction if comme celle-ci :

```
def fn():
    # ...
    if condition:
        # ne s'appelle pas elle-même
    else:
        fn()
    # ...
```

Fonction récursive

- En règle générale, on utilise une fonction récursive pour diviser un problème difficile à résoudre en problèmes plus petits qui sont plus faciles à résoudre.
- Exemple :

```
def somme(n):
    total = 0
    for index in range(n+1):
        total += index
    return total
result = somme(100)
print(result)
```

```
def somme(n):
    if n > 0:
        return n + somme(n-1)
    return 0
result = somme(100)
print(result)
```

Importation de Modules

- Python permet d'organiser le code en modules et en packages (Un package est un répertoire contenant plusieurs modules), ce qui facilite la réutilisation et la gestion.
  - Importation complète :

```
import math
print(math.sqrt(16))
```

  - Importation partielle :

```
from math import sqrt
print(sqrt(16))
```

  - Alias d'un module :

```
import math as m
print(m.sqrt(16))
```

  - Importation de tous les éléments d'un module (non recommandé) :

```
from math import *
print(sqrt(16))
```

Le module random

- Le module random en Python est une bibliothèque intégrée qui offre des fonctionnalités pour la génération de nombres aléatoires.
- Les fonctions couramment utilisées dans le module random :
  - ✓ random() : Retourne un nombre flottant aléatoire dans la plage [0, 1].

```
import random
random_number = random.random()
print("Nombre aléatoire entre 0 et 1 :", random_number)
```

  - ✓ randint(a, b) : Retourne un nombre entier aléatoire dans la plage [a, b]

```
import random
random_integer = random.randint(1, 10)
print("Nombre entier aléatoire entre 1 et 10 :", random_integer)
```

Le module OS

- Le module os en Python est un module standard qui fournit une manière d'interagir avec le système d'exploitation, comme la manipulation des fichiers et répertoires, la gestion des processus, l'accès aux variables d'environnement, etc.

Les fonctionnalités clés du module os

- os.getcwd() : Renvoie le répertoire de travail actuel.
- os.chdir(path) : Change le répertoire de travail actuel.
- os.listdir(path) : Liste tous les fichiers et répertoires dans le répertoire.
- os.mkdir(path) : Crée un répertoire.
- os.rename(src, dst) : Renomme un fichier ou répertoire.
- os.remove(path) : Supprime un fichier.
- os.path.isdir() vérifie si le chemin est un répertoire.
- os.path.isfile() vérifie si le chemin est un fichier.
- os.path.exists(path) Vérifie si un chemin spécifié existe
- os.path.abspath(path) Renvoie le chemin absolu

Les fonctionnalités clés du module os

```
import os
#  Obtenir le répertoire de travail actuel
current_dir = os.getcwd()
print(f"Répertoire de travail actuel : {current_dir}")

# Créer un nouveau répertoire
new_dir = "test_directory"
if not os.path.exists(new_dir):
    os.mkdir(new_dir)

# Changer de répertoire
os.chdir(new_dir)
print(f"Changé au répertoire : {os.getcwd()}")
```

Les fonctionnalités clés du module os

```
import os

# 5. Lister le contenu du répertoire
print(f"Contenu de {os.getcwd()} : {os.listdir('.')}")

# 8. Vérifier si le fichier renommé existe
print(os.path.exists(new_file_name))

# 9. Obtenir le chemin absolu du fichier renommé
print(f"Chemin absolu du fichier : {os.path.abspath(new_file_name)}")

# 10. Supprimer le fichier
os.remove(new_file_name)
print(f"Fichier '{new_file_name}' supprimé.")

# 11. Revenir au répertoire parent et supprimer le répertoire
os.chdir(current_dir)
os.rmdir(new_dir)
```

Installation de Packages avec pip

- pip est l'outil standard pour installer des packages externes en Python.
  - Installation d'un package : pip install nom_du_package
  - Installation d'une version spécifique :pip install nom_du_package == 1.2.3
  - Mise à jour d'un package : pip install --upgrade nom_du_package
  - Lister les packages installés : pip list
  - Désinstaller un package : pip uninstall nom_du_package

Numpy

👉 **NumPy** est une bibliothèque Python.

👉 **NumPy** est utilisé pour travailler avec des tableaux.

👉 **NumPy** est l'abréviation de « Numerical Python ».

👉 **NumPy** a également des fonctions pour travailler dans le domaine de l'algèbre linéaire, de la transformée de Fourier et des matrices.

Créez un tableau NumPy

❖ Exemple : en utilisant array()

- Créer un tableau unidimensionnel (vecteur) en utilisant une liste :

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

- Utilisez un tuple pour créer un tableau (vecteur) NumPy :

```
import numpy as np
arr = np.array((1, 2, 3, 4, 5))
print(arr)
print(type(arr))
```

Créez un tableau NumPy

❖ Exemple : Créer un tableau multidimensionnel (matrice)

- Créez un tableau 2D contenant deux tableaux avec les valeurs 1,2,3 et 4,5,6 :

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)

print(type(arr))
```

Numpy.arange

❖ arange(start, stop, step) :

- Renvoie des valeurs régulièrement espacées dans un intervalle donné.

```
>>> np.arange(3)
array([0, 1, 2])
>>> np.arange(3.0)
array([ 0.,  1.,  2.])
>>> np.arange(3,7)
array([3, 4, 5, 6])
>>> np.arange(3,7,2)
array([3, 5])
```

Générer des matrices

```
import numpy as np
# Créer une matrice 2x3 remplie de zéros
matrice_zeros = np.zeros((2, 3))
print("Matrice de zéros :\n", matrice_zeros)

# Créer une matrice 3x3 remplie de uns
matrice_uns = np.ones((3, 3))
print("\nMatrice de uns :\n", matrice_uns)

# Créer une matrice identité 4x4
matrice_identite = np.eye(4)
print("\nMatrice identité :\n", matrice_identite)

# Créer une matrice aléatoire 3x3 avec des valeurs entre 0 et 1
matrice_aleatoire = np.random.rand(3, 3)
print("\nMatrice aléatoire :\n", matrice_aleatoire)

# Créer une matrice aléatoire 3x3 avec des valeurs entre 5 et 15
matrice_al = np.random.randint(5,15,size=(3,3))
```

Opérations de base sur Numpy

```
result = arr + 2        # Addition
result = arr * 3        # Multiplication

mean = np.mean(arr)             # Calculer la moyenne
std_dev = np.std(arr)           # Calculer l'écart type

roduct = np.dot(matrix1, matrix2)         # Multiplication de matrices
solution = np.linalg.solve(coeff_matrix, constants)  # Résolution de systèmes linéaires

sub_array = arr[1:4]            # Extraire une tranche de `arr`

transposed = np.transpose(matrix)         # transposé
oncatenated = np.concatenate((arr1, arr2), axis=0)          # Concaténation verticale
sorted_arr = np.sort(arr)                 # Trier un tableau

determinant = np.linalg.det(matrix)       # determinant
trace = np.trace(matrix)                  # trace
inverse = np.linalg.inv(matrix)           # inverse

min_val = np.min(arr)                     # min
max_val = np.max(arr)                     # max
```

Matplotlib Pyplot

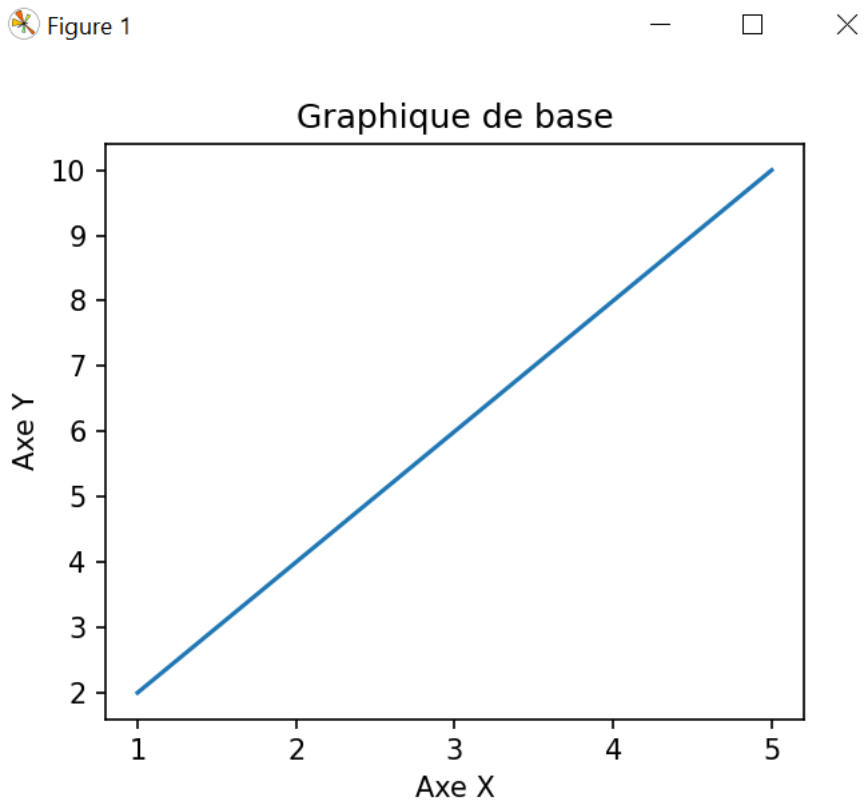
- Matplotlib est une bibliothèque Python populaire pour la création de graphiques et de visualisations de données.

- Elle offre un large éventail de fonctionnalités pour produire des graphiques de haute qualité pour l'exploration des données, la présentation de résultats, et bien plus encore.

Matplotlib Pyplot

- Création de graphiques de base : Matplotlib offre une grande flexibilité pour créer des graphiques. Voici comment vous pouvez créer un simple graphique à partir de données :

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.plot(x, y)
plt.xlabel('Axe X')
plt.ylabel('Axe Y')
plt.title('Graphique de base')
plt.show()
```



Matplotlib Pyplot

- Personnalisation des graphiques : Vous pouvez personnaliser l'apparence des graphiques en modifiant les couleurs, les styles de ligne, les étiquettes, les légendes, les titres, etc.

```
plt.plot(x, y, marker='o', linestyle='--', color='b', label='Données')
plt.legend()
```

- Création de sous-graphiques : Vous pouvez créer plusieurs sous-graphiques dans une même figure.

```
plt.subplot(2, 2, 1)  # Crée un sous-graphique en haut à gauche
plt.plot(x, y)
```

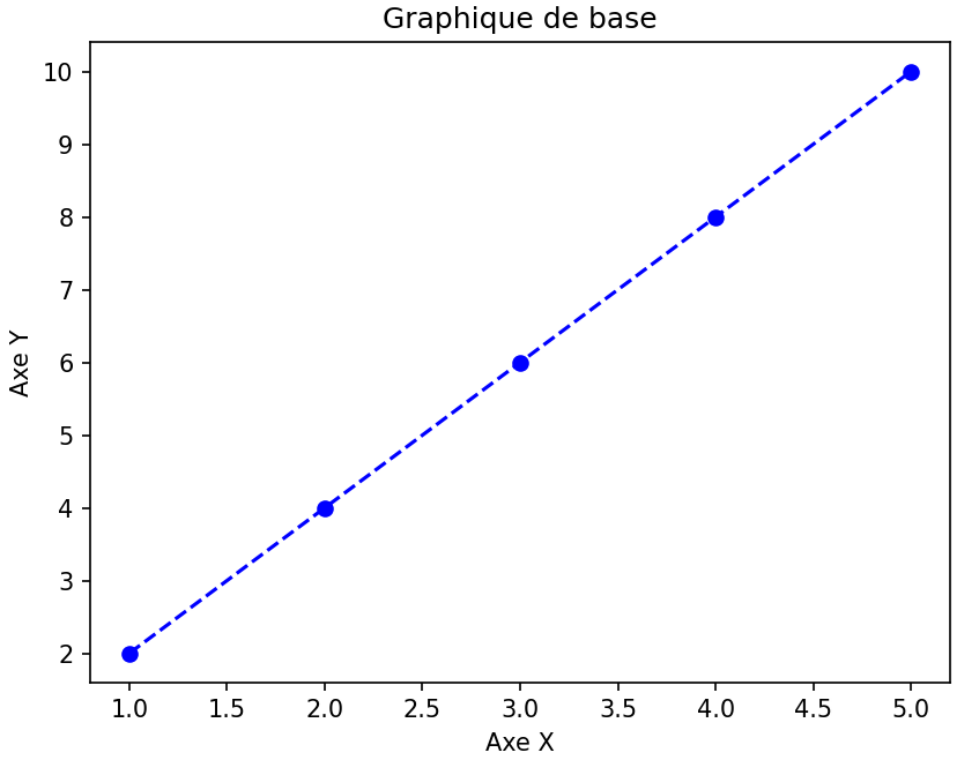
Matplotlib Pyplot

- Enregistrement des graphiques : Vous pouvez sauvegarder vos graphiques dans différents formats, tels que PNG, PDF, SVG, etc.

```
plt.savefig('mon_graphique.png')
```

Matplotlib Pyplot

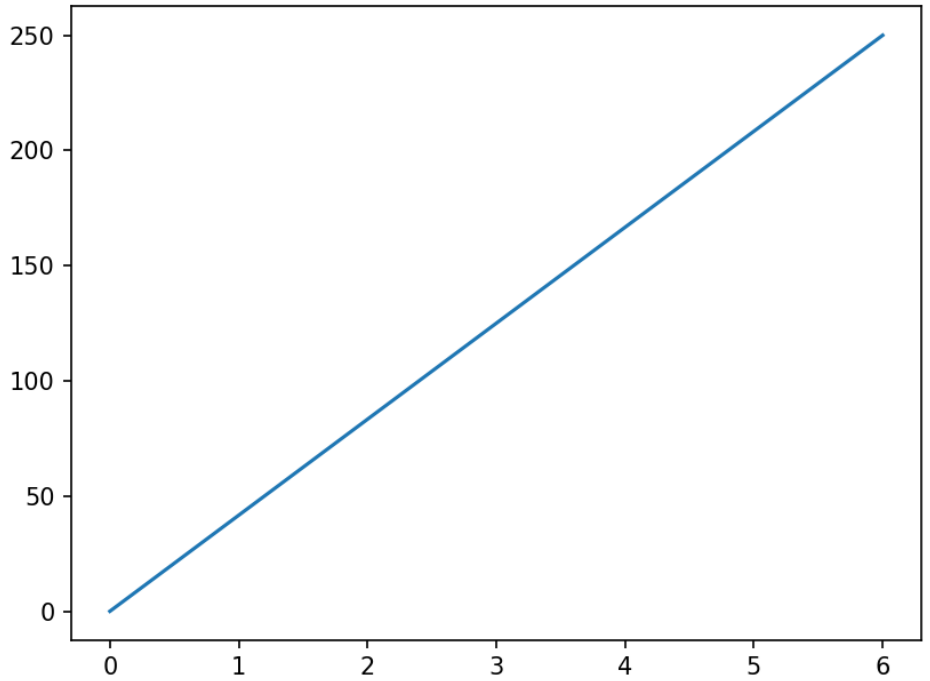
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([1, 2, 3, 4, 5])
y = x*2
plt.plot(x, y,marker='o', linestyle='--', color='b', label='Données' )

plt.xlabel('Axe X')
plt.ylabel('Axe Y')

plt.title('Graphique de base')
plt.savefig('mon_graphique.pdf')

plt.show()
```



Matplotlib Pyplot

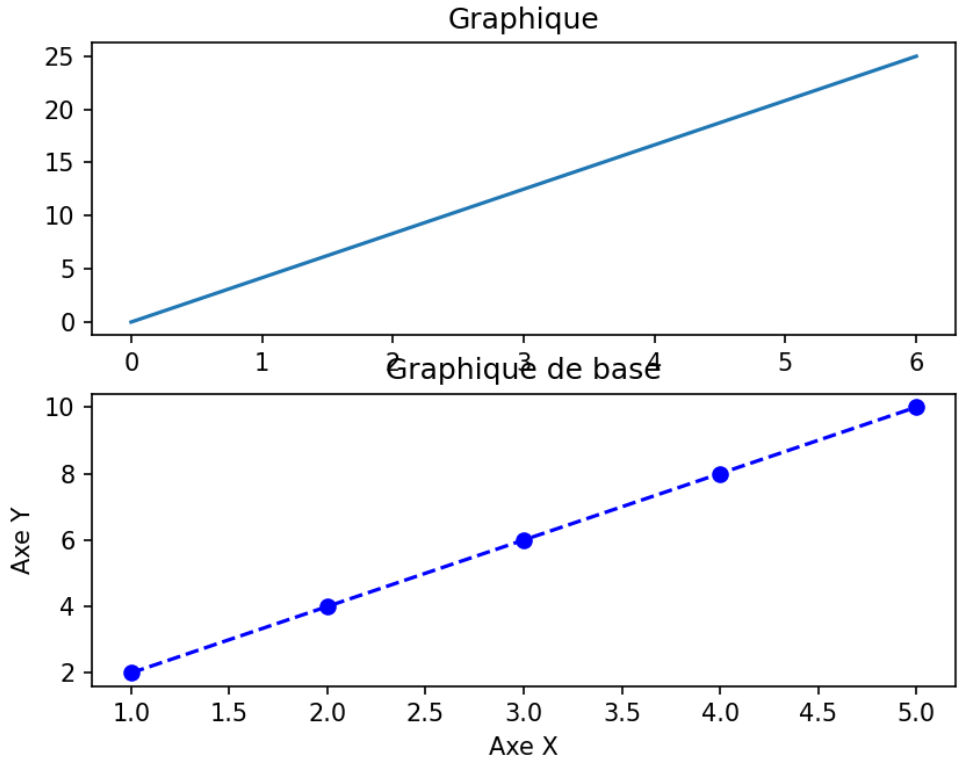
- Exemple : Tracez une ligne dans un diagramme de la position (0,0) à la position (6 250) :

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```



Matplotlib Pyplot

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([1, 2, 3, 4, 5])
y = x*2

plt.subplot(2, 1, 1)
xpoints = np.array([0, 6])
ypoints = np.array([0, 25])
plt.plot(xpoints, ypoints)
plt.title('Graphique')

plt.subplot(2, 1, 2)
plt.plot(x, y,marker='o', linestyle='--', color='b', label='Données' )
plt.xlabel('Axe X')
plt.ylabel('Axe Y')
plt.title('Graphique de base')
plt.savefig('mon_graphique.pdf')
plt.show()
```



Traitement du fichier

- Une manière de stocker des données de manière pérenne est de les stocker dans des fichiers.
- Editer un fichier :

  Pour éditer un fichier en python on utilise la fonction **open,** Cette fonction prend en premier paramètre le chemin du fichier (relatif ou absolu) et en second paramètre le type d'ouverture
- Chemin relatif et chemin absolu :
  - Un chemin relatif en informatique est un chemin qui prend en compte l'emplacement de lecture.
  - Un chemin absolu est un chemin complet qui peut être lu quelque soit l'emplacement de lecture.

La fonction open

- Voici la syntaxe pour lire un fichier : open(nom_fichier, mode, [options])

```
>>> fichier = open("data.txt", "r")
>>> print fichier
<open file 'data.txt', mode 'r' at 0x7ff6cf3fe4b0>
```

On remarque que le deuxième paramètre est renseigné par un r , ce paramètre indique une ouverture de fichier en lecture.

La fonction open

- Les types d'ouverture : Il existe plusieurs modes d'ouverture:
  - r : pour une ouverture en lecture (READ).
  - w : pour une ouverture en écriture (WRITE), à chaque ouverture le contenu du fichier est écrasé. Si le fichier n'existe pas python le crée.
  - a : pour une ouverture en mode ajout à la fin du fichier (APPEND). Si le fichier n'existe pas python le crée.
  - b : pour une ouverture en mode binaire.
  - t : pour une ouverture en mode texte.
  - x : crée un nouveau fichier et l'ouvre pour écriture

Fermeture d'un fichier

- Comme tout élément ouvert, il faut le refermer une fois les instructions terminées, Pour cela on utilise la méthode close() .

```
fichier = open("data.txt", "r")

fichier.close()
```

Lire le contenu d'un fichier

- Pour afficher tout le contenu d'un fichier, vous pouvez utiliser la méthode **read** sur l'objet-fichier.

```
fichier = open("data.txt", "r", encoding = 'utf_8')
print (fichier.read())
fichier.close()
```

Ecrire dans un fichier

- Voici la syntaxe pour écrire dans un fichier :

```
fichier = open("A.txt", "a")
fichier.write("Bonjour monde")
fichier.close()
```

- A noter que pour le monde d'ouverture a , si vous voulez écrire à la ligne, vous pouvez utiliser le saut de ligne \n

```
fichier = open("A.txt", "a")
fichier.write("\nBonjour monde")
fichier.close()
```

Le mot clé with

- le mot clé **with** est utilisée pour créer un contexte de gestion de fichier. Elle garantit que le fichier sera correctement fermé une fois que le bloc d'instructions est terminé.
- Voici la syntaxe:

```
with open("data.txt", "r" encoding = 'utf_8') as fichier:
    print fichier.read()
```

La fonction readline()

- La méthode **readline()** est utilisée pour lire une seule ligne à la fois à partir d'un fichier en Python. Voici comment vous pouvez l'utiliser pour lire un fichier ligne par ligne :

```
with open("data.txt", "r", encoding = 'utf_8') as file:
    line = file.readline()
    while line:
        print(line)
        line = file.readline()
```

- Lorsque readline() atteint la fin du fichier, il renvoie une chaîne vide, ce qui met fin à la boucle.

La fonction readline()

- Vous pouvez également utiliser une boucle for pour itérer à travers les lignes du fichier de manière plus concise :

```
with open("data.txt", "r", encoding = 'utf_8') as file:
    for line in file:
        print(line)
```

- Cela simplifie le processus en utilisant automatiquement readline() pour parcourir les lignes du fichier.

La fonction readline()

- La fonction readline() renverra des lignes qui incluent le caractère de nouvelle ligne ('\n') à la fin de chaque ligne.
- Vous pouvez utiliser la méthode **strip()** pour supprimer ces caractères de nouvelle ligne si nécessaire :

```
with open("mon_fichier.txt", "r") as file:
    for line in file:
        cleaned_line = line.strip()  # Supprimer les caractères de nouvelle ligne
        print(cleaned_line)
```

- Cela vous permettra de traiter chaque ligne du fichier sans les caractères de nouvelle ligne.

La fonction readlines()

- La fonction readlines() est une méthode couramment utilisée en programmation pour lire le contenu d'un fichier texte en Python. Cette fonction est spécifiquement associée aux objets de fichier ouverts en mode lecture.
- Voici comment elle fonctionne :

```
with open("data.txt", "r", encoding = 'utf_8') as file:
    lignes = file.readlines()
    for ligne in lignes:
        print(ligne)
```

La fonction split

- La fonction split() en Python est utilisée pour diviser une chaîne de caractères en plusieurs sous-chaînes en fonction d'un séparateur spécifié. Cette fonction est couramment utilisée pour traiter du texte, analyser des données ou extraire des informations d'une chaîne.
- La syntaxe de base de la fonction split() est la suivante :

```
phrase = "Ceci est un exemple de chaîne de caractères"
mots = phrase.split()  # Par défaut, sépare la chaîne par les espaces
print(mots)
```

```
['Ceci', 'est', 'un', 'exemple', 'de', 'chaîne', 'de', 'caractères']
```

Openpyxl

- **openpyxl** permet de lire, écrire et modifier des fichiers .xlsx
- vous devez d'abord l'installer via pip : pip install openpyxl
- Faire import openpyxl pour l'utiliser
- Manipulation d'un workbook :
  - ✓ **workbook = openpyxl.Workbook()** : création d'un nouveau workbook, qui par défaut a toujours un onglet.
  - ✓ **workbook.save('output.xlsx')** : pour sauvegarder le workbook.
  - ✓ **workbook = openpyxl.load_workbook('input.xlsx')** : pour lire un workbook.
  - ✓ **workbook.close()** : ferme le workbook.

lecture et écriture xlsx

- Un classeur est toujours créé avec au moins une feuille de calcul. Vous pouvez l'obtenir en utilisant la propriété : **Workbook.active**
  - ✓ **ws = wb.active**
- Vous pouvez créer de nouvelles feuilles de calcul à l'aide de la méthode suivante : Workbook.create_sheet()
  - ✓ ws1 = wb.create_sheet("Mysheet")
- Vous pouvez modifier ce nom à tout moment avec la propriété : Worksheet.title
  - ✓ ws.title = "New Title"
- Une fois que vous avez donné un nom à une feuille de calcul, vous pouvez l'obtenir en tant que clé du classeur : ws3 = wb["New Title"]

Accès à une cellule

- Maintenant que nous savons comment obtenir une feuille de calcul, nous pouvons commencer à modifier le contenu des cellules. Les cellules sont accessibles directement en tant que clés de la feuille de calcul : ws['A4']

- Il y a aussi la méthode ***Worksheet.cell()*** Cela permet d'accéder aux cellules à l'aide de la notation de ligne et de colonne : ws.cell(i,j)

Exemple

```
import openpyxl

# Charger un fichier Excel existant
workbook = openpyxl.load_workbook('mon_fichier.xlsx')

# Accéder à une feuille de calcul spécifique
sheet = workbook['Nom de la feuille']

# Lire la valeur d'une cellule
valeur = sheet['A1'].value

# Afficher la valeur
print(valeur)
```

Exemple

```
import openpyxl
# Créer un nouveau classeur (fichier Excel)

workbook = openpyxl.Workbook()

# Sélectionner la première feuille de calcul par défaut
sheet = workbook.active

# Ajouter des données dans quelques cellules
sheet['A1'] = 'Nom'
sheet['B1'] = 'Âge'
sheet['A2'] = 'Al'
sheet['B2'] = 28
sheet['A3'] = ‘Ba'
sheet['B3'] = 32
sheet['A4'] = ‘Cheikh'
sheet['B4'] = 24

# Sauvegarder le fichier Excel sous un nom spécifié
workbook.save('mon_nouveau_fichier.xlsx')

# Fermer le classeur
workbook.close()
```

Exemple

```
import openpyxl

# Charger un fichier Excel existant
workbook = openpyxl.load_workbook('mon_fichier.xlsx')

# Accéder à une feuille de calcul spécifique
sheet = workbook['Nom de la feuille']

# Parcourir les cellules de la feuille de calcul en utilisant des indices i et j
for i in range(1, sheet.max_row + 1):
    for j in range(1, sheet.max_column + 1):
        # Accéder à la valeur de la cellule à la position (i, j)
        valeur = sheet.cell(row=i, column=j).value
        # Faites quelque chose avec la valeur, par exemple l'afficher
        print(valeur)

# Fermer le classeur
workbook.close()
```

Traitement d'images en Python

- Les images publiées sur un site internet, les photographies prises avec un téléphone portable, sont des exemples d'images numériques. Il est possible de représenter ce type d'image par une matrice. Par exemple, L'image ci-dessous peut être représentée par une matrice 35 × 35 dont les éléments, des 0 ou des 1, indiquent la couleur du pixel : 0 pour le noir et 1 pour le blanc.

Traitement d'images en Python

- les images en couleur peuvent être représentées par trois matrices, chacune déterminant la quantité respective de rouge, de vert et de bleu qui constitue l'image. Ce modèle de couleur est appelé RGB (red-green-blue)

- Les éléments de ces matrices sont des nombres entiers compris entre 0 et 255 (des entiers non signés sur 8 bits) qui déterminent l'intensité de la couleur de la matrice pour le pixel correspondant. Le modèle RGB permet donc de représenter 2563 = 224 = 16 777 216 couleurs différentes en théorie.

Traitement d'images en Python

- les images en couleur peuvent être représentées par trois matrices, chacune déterminant la quantité respective de rouge, de vert et de bleu qui constitue l'image. Ce modèle de couleur est appelé RGB (red-green-blue)

- Les éléments de ces matrices sont des nombres entiers compris entre 0 et 255 (des entiers non signés sur 8 bits) qui déterminent l'intensité de la couleur de la matrice pour le pixel correspondant. Le modèle RGB permet donc de représenter 2563 = 224 = 16 777 216 couleurs différentes en théorie.

Traitement d'images en Python

- Une image en couleur. Les trois autres images ont été obtenues en ne conservant qu'une des trois composantes RGB qui la composent.

Traitement d'images en Python

- Le module matplotlib.image qui propose quelques fonctions basiques pour importer une image, essentiellement :
  - imread('fichier.png') qui prend en argument le nom d'un fichier image au format png et retourne un tableau numpy
  - imsave('fichier.png', tableau) qui sauvegarde un tableau numpy représentant une image sous la forme d'un fichier png.

Traitement d'images en Python

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('picasso.png')
rouge = np.zeros_like(img)
vert = np.zeros_like(img)
bleu = np.zeros_like(img)
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        rouge[i, j, 0] = img[i, j, 0]
        vert[i, j, 1] = img[i, j, 1]
        bleu[i, j, 2] = img[i, j, 2]
fig = plt.figure(figsize=(6, 6), frameon=False)
plt.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=.05, wspace=.05)
plt.subplot(2, 2, 1)
plt.axis('off')
plt.imshow(img)
plt.subplot(2, 2, 2)
plt.axis('off')
plt.imshow(rouge)
plt.subplot(2, 2, 3)
plt.axis('off')
plt.imshow(vert)
plt.subplot(2, 2, 4)
plt.axis('off')
plt.imshow(bleu)
plt.show()
```

Les algorithmes de tri

- Qu'est-ce qu'un algorithme de tri ?

👉 Un algorithme de tri est une suite finie d'opérations permettant d'organiser des éléments dans un ordre déterminé préalablement. Le plus souvent, les informaticiens trient des tableaux d'entiers du plus petit au plus grand.

👉 Problème : étant donné un tableau d'entiers T, trier T dans l'ordre croissant.

👉 Des algorithmes bases sur des idées et des structures de données très différentes.

👉 Des complexités différentes.

👉 Des algorithmes optimaux.

Les algorithmes de tri

- Les différents algorithmes de tri à connaître :

  - 👉 Le tri à bulle
  - 👉 Le tri par sélection
  - 👉 Le tri par insertion
  - 👉 Le tri par fusion
  - 👉 Le tri rapide

Le tri à bulle

- Cette méthode de tri consiste à inverser tous les couples de nombres non ordonnés.

- Par exemple, avec les 3 éléments : 6|5|4 : le tri à bulle va comparer 6 et 5, comme ils ne sont pas ordonnés, ils vont être inversés (on aura 5|6|4), puis va comparer 6 et 4 et les échanger (5|4|6).

- À la fin de la première boucle, le plus grand élément du tableau sera forcément bien placé. Puis, l'algorithme recommence jusqu'à la fin du tri.

Le tri à bulle

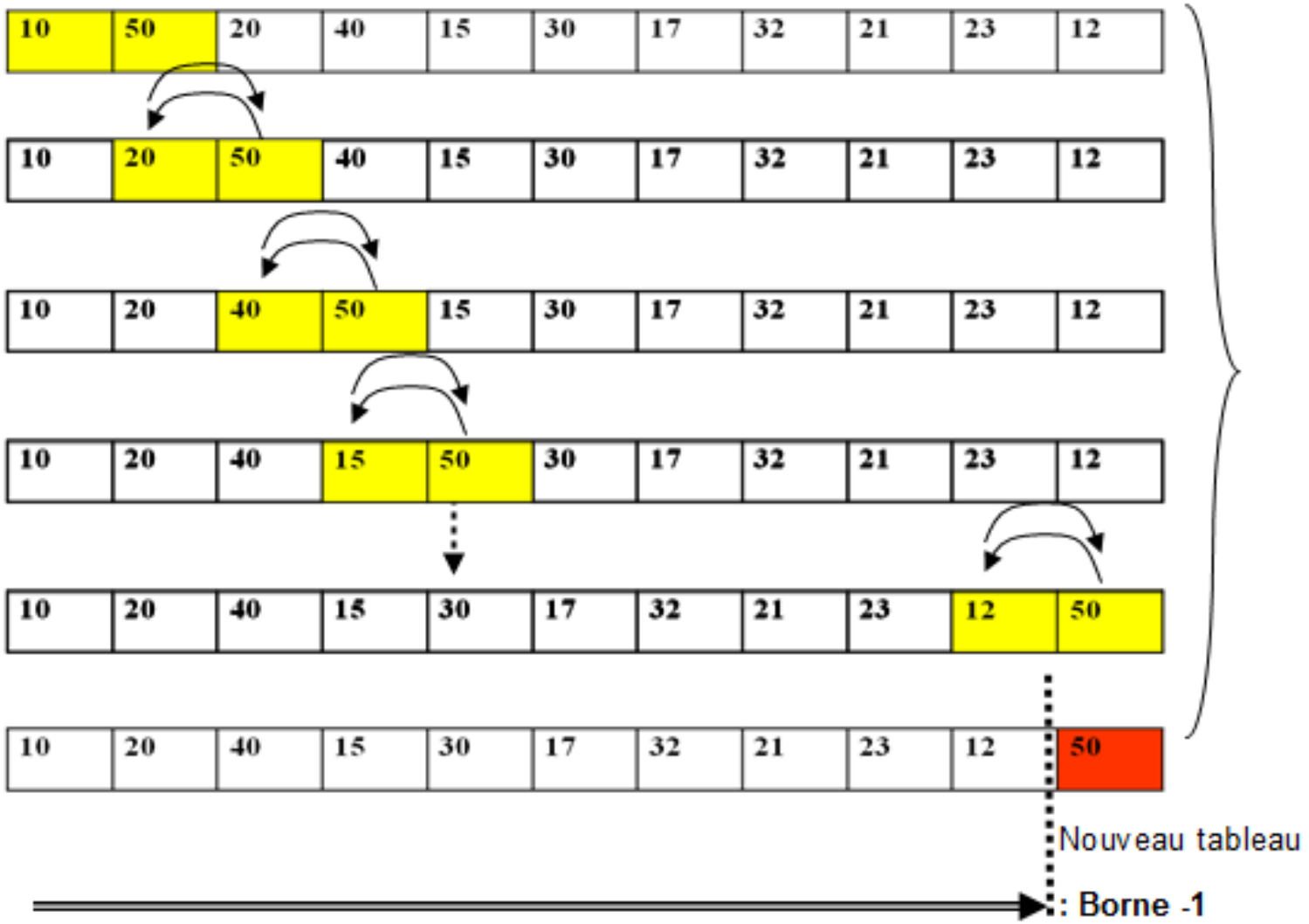
- Le tri à bulle consiste à parcourir le tableau, par exemple de gauche à droite, en comparant les éléments côte à côte et en les permutant s'ils ne sont pas dans le bon ordre.

- Au cours d'une passe du tableau, les plus grands éléments remontent de proche en proche vers la droite comme des bulles vers la surface.

- On s'arrête dès que l'on détecte que le tableau est trié : si aucune permutation n'a été faite au cours d'une passe.

Le tri à bulle

- Exemple :



Le tri à bulle

- Le pseudo-code du tri à bulle :

```
tri_à_bulles(Tableau T)
        pour i allant de (taille de T)-1 à 1
                pour j allant de 0 à i-1
                        si T[j+1] < T[j]
                                (T[j+1], T[j]) = (T[j], T[j+1])
                        fin_si
                fin_pour
        fin_pour
fin_tri_à_bulles
```

- Cet algorithme a aussi une complexité quadratique en $O(n^2)$. Cependant, plus le tableau est préalablement trié, plus il est efficace.

Le tri à bulle

- Implémentation avec python :

```
def tri_a_bulles(T):
    n = len(T)

    for i in range(n - 1, 0, -1):
        for j in range(0, i):
            if T[j + 1] < T[j]:
                T[j], T[j + 1] = T[j + 1], T[j]

# Exemple d'utilisation
ma_liste = [64, 34, 25, 12, 22, 11, 90]
tri_a_bulles(ma_liste)
print("Liste triée:", ma_liste)
```

Le tri par insertion

- Cette stratégie consiste à trier élément par élément le tableau.

- D'abord, l'algorithme trie les 2 premières valeurs. Puis, il place la troisième à sa place vis-à-vis des 2 premières.

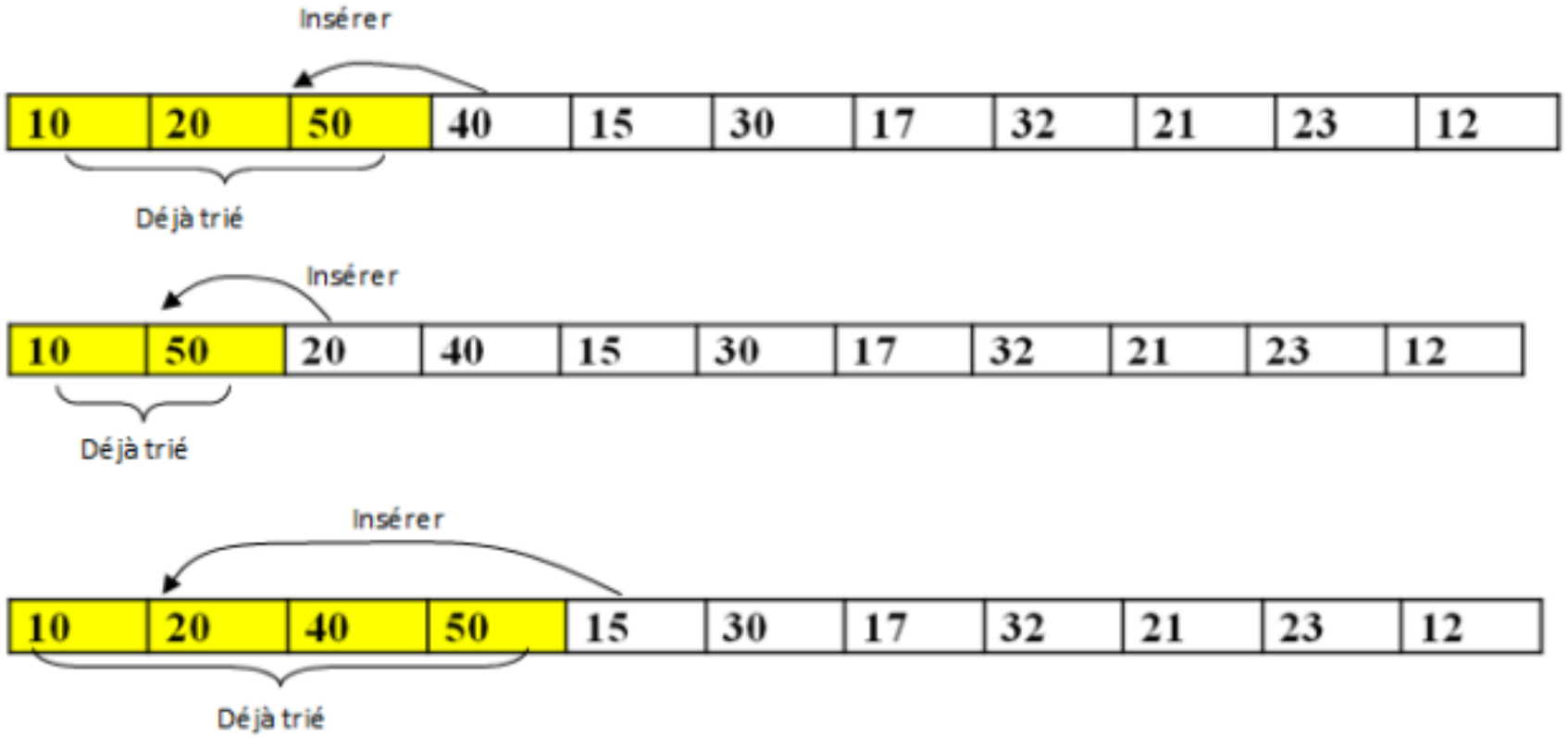
- Au global, il s'agit de placer la valeur i dans la partie du tableau déjà triée allant de la première valeur à la i-1ème.

- Méthode: Trier la liste de gauche à droite en insérant à chaque fois l'élément i+1 dans le tableau (déjà trié) des premiers éléments.

Le tri par insertion

- Exemple :

Trier la liste de gauche à droite en insérant à chaque fois l'élément i+1 dans le tableau (déjà trié) des premiers éléments.



Le tri par insertion

- le pseudo-code

```
tri_insertion(Tableau T)
        pour i de 1 à taille(T) - 1
                x ← T[i]
                j ← i
                        tant que j > 0 et T[j - 1] > x
                                T[j] ← T[j - 1]
                                j ← j - 1
                        fin_tant_que
                T[j] ← x
        fin_pour
fin_tri_insertion
```

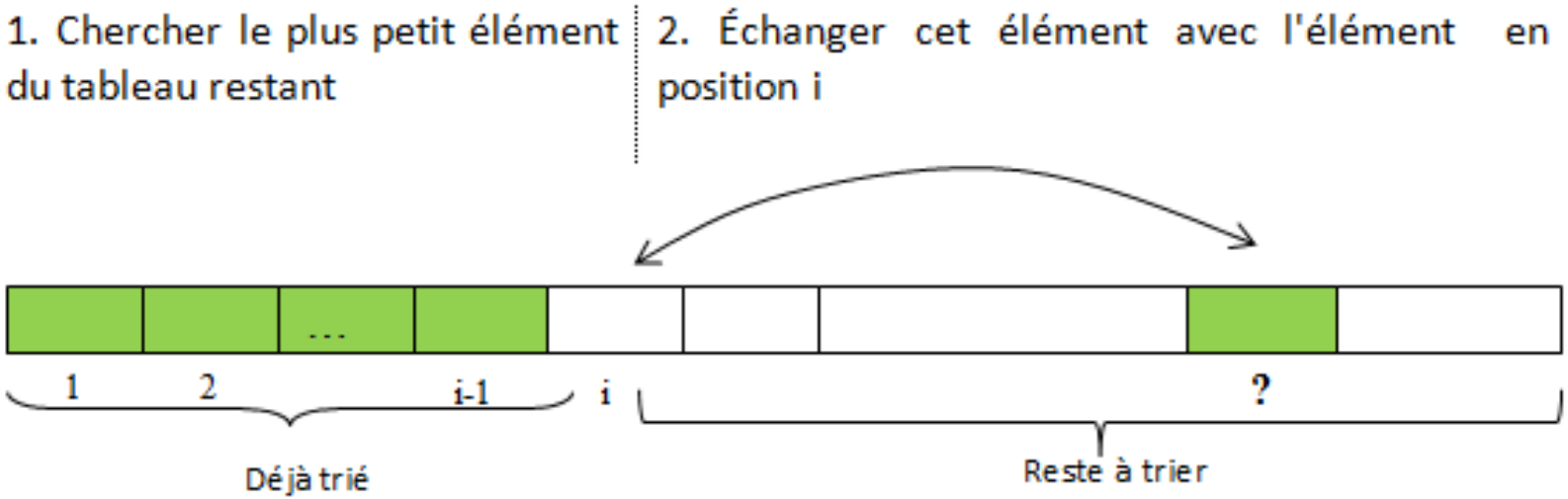
- En moyenne, le tri par insertion a une complexité quadratique ($O(n^2)$), mais si le tableau est déjà ordonné, sa complexité est linéaire ($O(n)$).

Le tri par insertion

```
def tri_insertion(T):
    n = len(T)

    for i in range(1, n):
        x = T[i]
        j = i

        while j > 0 and T[j - 1] > x:
            T[j] = T[j - 1]
            j -= 1

        T[j] = x

# Exemple d'utilisation
ma_liste = [64, 34, 25, 12, 22, 11, 90]
tri_insertion(ma_liste)
print("Liste triée:", ma_liste)
```

Le tri par sélection

- Cet algorithme est intuitif. D'abord, il cherche l'élément le plus petit du tableau. Puis, il échange cet élément avec l'élément en première place du tableau. Enfin, il réitère ces actions jusqu'à ce que le tableau soit entièrement trié.
- L'idée du tri du consiste à chaque étape à rechercher le plus petit élément non encore trié et à le placer à la suite des éléments déjà triés.



Le tri par sélection

- Le pseudo code correspondant :

```
tri_selection(Tableau T)
        n ← longueur(T)
        pour i de 0 à n - 2
                min ← i
                pour j de i + 1 à n - 1
                        si T[j] < T[min] alors
                                min ← j
                fin pour
                si min ≠ i, alors
                        échanger T[i] et T[min]
        fin pour
fin tri_selection
```

- La complexité du tri par sélection est en $O(n^2)$. Il n'est donc pas très efficace.

Le tri par sélection

```
def tri_selection(T):
    n = len(T)

    for i in range(n - 1):
        min_index = i

        for j in range(i + 1, n):
            if T[j] < T[min_index]:
                min_index = j

        if min_index != i:
            T[i], T[min_index] = T[min_index], T[i]

# Exemple d'utilisation
ma_liste = [64, 34, 25, 12, 22, 11, 90]
tri_selection(ma_liste)
print("Liste triée:", ma_liste)
```

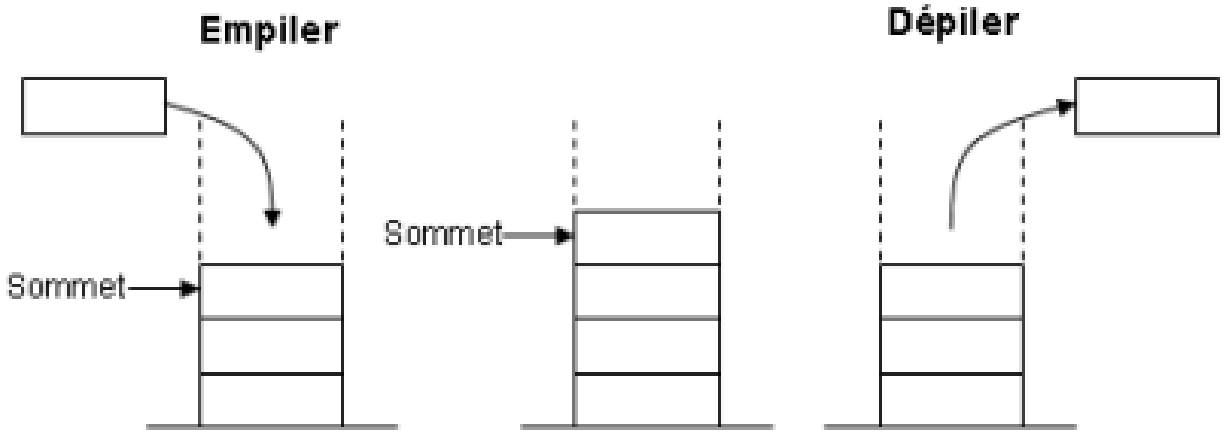
Pile et File

- les piles (stacks) et les files (queues) sont des structures de données couramment utilisées pour gérer des collections d'éléments, mais elles fonctionnent de manière différente.

- Les Piles peuvent être considérées comme des listes avec des conditions d'utilisation restreintes : Il est possible d'insérer ou de supprimer un élément uniquement à la fin, (en anglais « LIFO », acronyme de « Last In First Out » dernier entre, premier sorti).

Pile

- Une pile (stack en anglais) est une structure de données fondamentale en informatique qui suit le principe de Last-In, First-Out (LIFO). Cela signifie que le dernier élément ajouté à la pile est le premier à en être retiré.



Pile

- Vous pouvez implémenter une pile en Python en utilisant une liste (ou array) et en utilisant certaines méthodes pour simuler ce comportement.
  - Créer une pile (vide) : Dans python, les piles sont réalisées avec le type List :

```
def creer_pile():
    return []
```

  - Ajouter un élément au sommet : La pile permet d'ajouter des éléments (empiler). En Principe, la pile n'a pas de taille maximale.

```
def empiler(pile, element):
    pile.append(element)
```

Pile

- Retirer le dernier élément ajouté : Pour dépiler un élément, on utilise la fonction pop, qui enlève le dernier élément ajouté et le renvoie

```
def depiler(pile):
    if not est_vide(pile):
        return pile.pop()
```

- La pile est-elle vide :

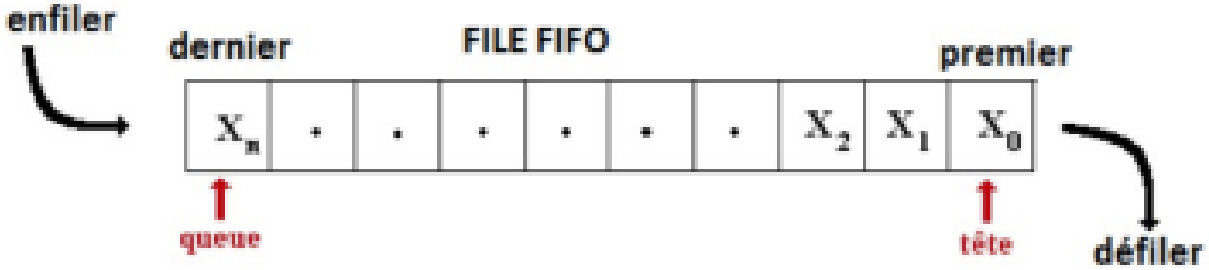
```
def est_vide(pile):
    return pile == []
```

- Sommet de la pile (dernier élément ajouté) : On a un message d'erreur si la pile est vide.

```
def sommet(pile):
    if not est_vide(pile):
        return pile[-1]
```

File

- Une file est une structure de données de type FIFO (first in first out) le premier entré est le premier sorti.
- Une file est utilisée à fin de mémoriser des données en attente de traitement.
- Une file est définie par deux opérations :
  - Entrer une valeur (Enfiler)
  - Sortir une valeur (Défiler)



File

- Créer file :

```
def creer_file():
    return []
```

- La file est-elle vide

```
def est_vide(file):
    return file == []
```

- Placer un élément en file d'attente :

```
def enfiler(file, element):
    file.append(element)
```

- Faire sortir le premier élément de la file :

```
def defiler(file):
    if not est_vide(file):
        return file.pop(0)
```

La Programmation Orientée Objet (POO)

- Python est un langage de programmation orienté objet.
- Presque tout en Python est un objet, avec ses propriétés et ses méthodes.
- Une classe est comme un constructeur d'objet ou un « plan » pour créer des objets.
- une classe est un ensemble de code contenant des variables et des fonctions permettant de créer des objets.

Créer une classe

- Une classe est un modèle pour créer des objets. Vous pouvez penser à une classe comme un plan ou un modèle pour créer des objets.
- Pour créer une classe, utilisez le mot-clé : **class**
- Exemple : Voici comment vous définissez une classe en Python

  Créez une classe nommée MyClass, avec une propriété nommée x :

```
class MyClass:
  x = 5
```

Créer une classe

- Nous pouvons maintenant utiliser le nom de la classe pour créer des objets.
- Exemple
  - ✓ Créez un objet nommé p1 et affichez la valeur de x :

```
p1 = MyClass()
print(p1.x)
```

La fonction __init__()

- Toutes les classes ont une fonction appelée __init__(), qui est toujours exécutée pour créer un objet.
- Utilisez la fonction __init__() pour affecter des valeurs à des propriétés d'objet.

```
class MaClasse:
    def __init__(self, attribut1, attribut2):
        self.attribut1 = attribut1
        self.attribut2 = attribut2

    def methode1(self):
        # Définition de la méthode pass
```

La fonction __init__()

- Objet : Un objet est une instance d'une classe. Vous créez un objet en appelant le constructeur de la classe, qui est généralement la méthode __init__.

```
mon_objet = MaClasse(valeur1, valeur2)
```

- Attributs : Les attributs sont des variables qui appartiennent à un objet. Vous pouvez accéder à ces attributs en utilisant la notation point (.)

Par exemple : `print(mon_objet.attribut1)`

- Méthodes : Les méthodes sont des fonctions définies dans une classe qui agissent sur les données de l'objet. Vous appelez une méthode en utilisant la notation point. Par exemple `mon_objet.methode1()`

La fonction __init__()

❖ Exemple

✓ Créez une classe nommée Person, utilisez la fonction __init__() pour affecter les valeurs des propriétés nom et âge.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person('Ali', 22)

print(p.name)
print(p.age)
```

✓ La fonction __init__() est appelée automatiquement chaque fois que la classe est utilisée pour créer un nouvel objet.

La fonction __str__()

- La fonction __str__() permet d'afficher l'objet sous forme d'une chaine de caractére.
- Exemple :

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __str__(self):
        return self.name + ' ' + str(self.age)

p = Person('Ali', 22)

print(p)
```

La fonction __str__()

- Les objets peuvent également contenir des méthodes. Les méthodes dans les objets sont des fonctions qui appartiennent à l'objet.

- Exemple :

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

- Le paramètre **self** est une référence à l'instance actuelle de la classe et est utilisée pour accéder aux variables qui appartiennent à la classe.

Encapsulation

- La POO favorise l'encapsulation, ce qui signifie que les données et les méthodes associées à un objet sont regroupées en une seule entité. En Python, vous pouvez définir des méthodes et des attributs privés en préfixant leur nom avec un double trait de soulignement (_ _).
- Par exemple :

```
class MaClasse:
    def __init__(self, attribut1, attribut2):
        self.__attribut1 = attribut1
```