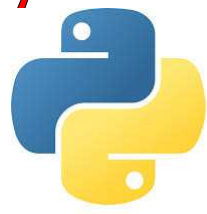


Faculté des Sciences  
كلية العلوم

Département d'Informatique  
Filière MIP - Semestre 2  
2024-2025

# Informatique II

## Algorithmique II / Python



Pr. Khadija Louzaoui

1

### Programmation Python

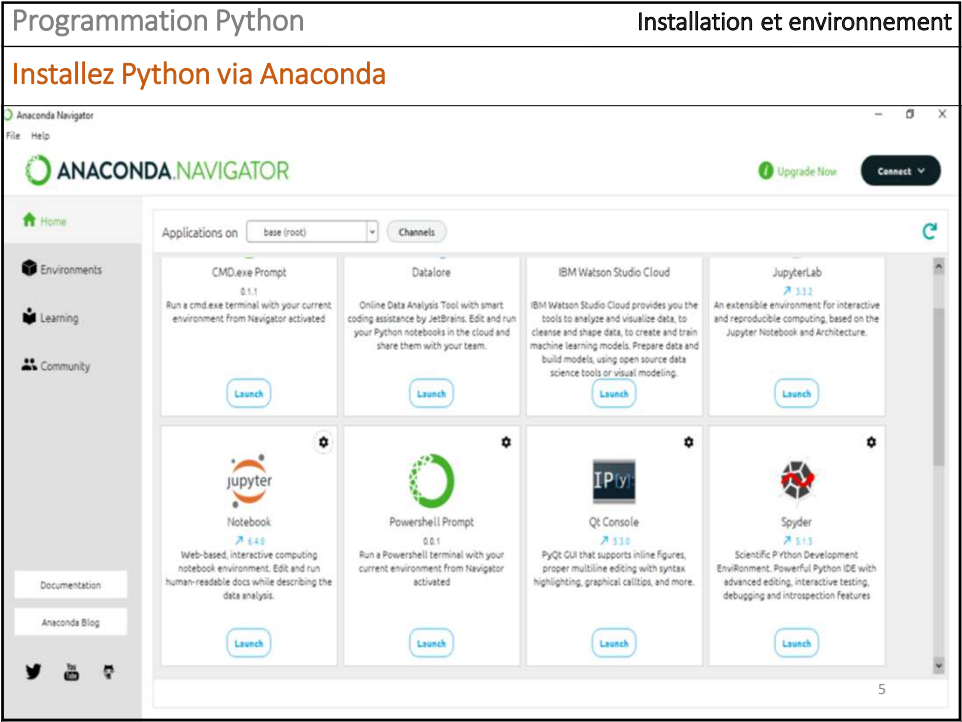
#### Introduction à Python

- Python est un langage de programmation **interprété, haut niveau et polyvalent**, conçu pour être facile à lire et à écrire.
- Créé par **Guido van Rossum** en 1991, Python est connu pour sa syntaxe simple, claire et concise.
- Principales caractéristiques :
  - **Lisibilité** : La syntaxe est proche du langage naturel, ce qui le rend accessible aux débutants.
  - **Communauté large** : Des milliers de bibliothèques et de ressources disponibles.
  - **Multi-paradigme**: Supporte la programmation procédurale, orientée objet et fonctionnelle.
- Utilisé dans divers domaines :
  - Développement web (Django, Flask)
  - Science des données et IA (NumPy, Pandas, TensorFlow)
  - Automatisation de tâches
  - Jeux vidéo, applications mobiles, et plus encore.

2

Programmation Python	Installation et environnement
<b>Installation et configuration</b>	
<b>Installation de Python</b>	
<ol style="list-style-type: none"><li><b>1. Téléchargement</b> : Rendez-vous sur le site officiel <a href="https://python.org">python.org</a>.</li><li><b>2. Choisissez la version Python 3.x</b> (recommandée) et téléchargez l'installateur correspondant à votre système d'exploitation (Windows, macOS, Linux).</li><li><b>3. Installation</b></li><li><b>4. Installation d'un éditeur de code</b><ul style="list-style-type: none"><li>• IDLE : Inclus avec Python, simple pour débiter.</li><li>• Visual Studio Code : Léger, extensible, et idéal pour Python.</li><li>• PyCharm : Un IDE puissant avec des fonctionnalités avancées.</li><li>• Jupyter Notebook : Excellent pour les analyses interactives.</li></ul></li><li><b>5. Installer des bibliothèques supplémentaires</b><ul style="list-style-type: none"><li>• Utilisez pip (le gestionnaire de paquets intégré à Python) pour installer des bibliothèques.</li></ul></li></ol>	
3	

Programmation Python	Installation et environnement
<b>Installez Python via Anaconda</b>	
<ul style="list-style-type: none"><li>▪ Anaconda est une distribution scientifique de Python : c'est-à-dire qu'en installant Anaconda, vous installerez Python, Jupyter Notebook et des dizaines de packages scientifiques, dont certains indispensables à l'analyse de données.</li><li>▪ Pour commencer, téléchargez la distribution Anaconda correspondant à votre système d'exploitation</li></ul> <p><a href="https://www.anaconda.com/products/distribution">https://www.anaconda.com/products/distribution</a></p>	
4	



Programmation Python

Installation et environnement

Installez Python via Anaconda

Il existe plusieurs applications dans Anaconda

- Navigator tel que :
- JupyterLab
- JupyterNotebook
- Spyder
- Pycharm
- VSCode
- Orane 3 APP
- RStudio
- Anaconda powerShell

6

Programmation Python

Installation et environnement

Jupyter Notebook

- Jupyter Notebook est une plateforme Web open source qui facilite la création et le partage des documents qui contiennent du code.
- Jupyter Notebook permet de gérer des fichiers, des modules et aussi la présentation du travail. C'est un produit phare open-source de Projet Jupyter et est largement utilisé en science des données.
- Il y a deux manières de lancer Jupiter Notebook.

**Avec la console Anaconda prompt** : sur la barre de recherche, taper “Anaconda prompt” puis taper «jupyter notebook” :

**Méthode directe** : taper directement dans la barre de recherche : “Anacondanavigator”. Une fois que l’interface est affichée, cliquez sur launch de l'application Jupyter.

7

Programmation Python

Installation et environnement

Jupyter Notebook

Jupyter Test

Last Checkpoint: il y a une heure (unsaved changes)

Logout

File

Edit

View

Insert

Cell

Kernel

Widgets

Help

Trusted

Python 3 (ipykernel)

+

%

↑

↓

▶Run

⌂

↻

Code

▼

In [ ]:

8

Programmation Python

Installation et environnement

Version de python

In [1]:

```
from platform import python_version
python_version()
```

Out[1]:

'3.10.9'

Les Calculs

Une des premières fonctionnalités d'un interpréteur est de faire des calculs:

In [1]:

```
1+2+8
```

Out[1]:

11

Tous les opérateurs sont utilisables

9

Programmation Python

Installation et environnement

Les commentaires

Les commentaires peuvent être utilisés pour expliquer le code pour plus de lisibilité.

In [2]:

```
# Commentaire sur une seule ligne
val = 50
```

In [3]:

```
# Commentaire
# sur plusieurs
# lignes
val = 50
```

In [4]:

```
'''
Commentaire
sur plusieurs
lignes
'''
val = 50
```

In [5]:

```
"""
Commentaire
sur plusieurs
lignes
"""
val = 50
```

10

Pr. Louzaoui Khadija

5

Programmation Python

Installation et environnement

Identifiant

Un **identifiant** est un nom donné à des entités telles que des classes, des fonctions, des variables, etc. Il permet de différencier une entité d'une autre.

L'identifiant :

- Ne peut pas commencer par un chiffre
- Ne peut pas utiliser de symboles spéciaux
- Les mots clés ne peuvent pas être utilisés comme identifiants

In [6]:

1var = 10

Cell In[6], line 1

1var = 10

^

SyntaxError: invalid decimal literal

In [7]:

val@ = 50

Cell In[7], line 1

val@ = 50

^

SyntaxError: invalid syntax

In [8]:

import = 17

Cell In[8], line 1

import = 17

^

SyntaxError: invalid syntax

11

Programmation Python

Installation et environnement

Identifiant

Python 3.10 contient des mots clés réservés. Ces mots ne peuvent pas être utilisés comme identifiant:

In [9]:

import keyword

print(keyword.kwlist)

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

12

Programmation Python

Syntaxe de base

Les Variables

- Une **variable** est une zone de la mémoire de l'ordinateur dans laquelle une valeur est stockée. Coté programmeur, cette variable est définie par un nom (identificateur), alors que pour l'ordinateur, il s'agit en fait d'une adresse, c'est-à-dire d'une zone particulière de la mémoire.
- Une variable python a les caractéristiques suivantes :
  - Un pointeur vers une zone mémoire.
  - Permettant de stocker une ou plusieurs données.
  - Pouvant avoir plusieurs valeurs différentes dans un programme.
  - Pas de type à préciser pour les variables.
  - Une variable peut changer de type.
  - Impossible de concaténer une chaîne et un nombre sans conversion.

13

Programmation Python

Syntaxe de base

Les Variables

- 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.
- Testez les instructions suivantes après avoir lancé l'interpréteur :

In [10]:

#Pour déclarer une variable  
x = 2  
#Pour afficher une variable  
x

Out[10]:

2

In [11]:

#Pour utiliser une variable  
y = x + 10  
y

Out[11]:

12

In [12]:

#Pour déclarer plusieurs variables et leur affecter une même valeur  
a = b = c = 5  
#Pour afficher les valeurs respectives des variables précédentes  
a,b,c

Out[12]:

(5, 5, 5)

In [13]:

# Pour déclarer plusieurs variables et leur affecter des valeurs différentes  
d, e, f = 1, 2, "trois"  
d,e,f

Out[13]:

(1, 2, 'trois')

14

Pr. Louzaoui Khadija

7

Programmation Python

Syntaxe de base

Typage des variables

En Python, le typage des variables peut être dynamique ou explicite grâce à l'utilisation d'annotations de type.

1. Typage Dynamique

Python est un langage à typage dynamique, ce qui signifie que le type d'une variable est déterminé automatiquement au moment de l'exécution.

Par exemple :

```
In [ ]: x = 10      # entier (int)
        y = 3.14    # nombre à virgule flottante (float)
        Module = "Algorithmique" # chaîne de caractères (str)
        est_valide = True # booléen (bool)
```

Dans ce cas, vous n'avez pas besoin de déclarer explicitement le type d'une variable.

15

Programmation Python

Syntaxe de base

Typage des variables

2. Typage Statique avec les Annotations de Type

Depuis Python 3.5, vous pouvez utiliser des annotations de type pour rendre le code plus clair et détecter les erreurs à l'aide d'outils comme MyPy.

Voici des exemples :

a. Annotations simples

```
python

age: int = 25      # `age` doit être un entier
pi: float = 3.14159 # `pi` est un float
name: str = "Alice" # `name` est une chaîne
is_valid: bool = True # `is_valid` est un booléen
```

16

Pr. Louzaoui Khadija

8



## Typage des variables

**2. Typage Statique avec les Annotations de Type**

Depuis Python 3.5, vous pouvez utiliser des annotations de type pour rendre le code plus clair et détecter les erreurs à l'aide d'outils comme MyPy.

Voici des exemples :

**b. Listes, Dictionnaires, et Autres Types Composés**

Pour annoter les structures de données, utilisez le module `typing` :

```
python

from typing import List, Dict

numbers: List[int] = [1, 2, 3, 4]
person: Dict[str, int] = {"age": 30, "height": 170}
```

17

## Typage des variables

**2. Typage Statique avec les Annotations de Type**

Depuis Python 3.5, vous pouvez utiliser des annotations de type pour rendre le code plus clair et détecter les erreurs à l'aide d'outils comme MyPy.

Voici des exemples :

**c. Fonctions avec annotations**

Vous pouvez également annoter les arguments et la valeur de retour des fonctions :

```
python

def greet(name: str) -> str:
    return f"Hello, {name}!"
```

18

Programmation Python

Syntaxe de base

Typage des variables

3. Vérification des Types

Python ne vérifie pas les types à l'exécution. Les annotations servent principalement à :

- Documenter le code.
- Permettre à des outils externes (comme MyPy ou Pyright) de détecter les incohérences.

Le typage dynamique de Python offre une grande flexibilité, tandis que les annotations de type rendent le code plus lisible et robuste. Cela permet de bénéficier du meilleur des deux mondes.

19

Programmation Python

Syntaxe de base

Typage des variables

▪ Les types de base:

Type	Description	Exemple
int	Nombres entiers	x = 42
float	Nombres à virgule flottante	y = 3.14
str	Chaînes de caractères	name = "Khadija"
bool	Valeurs booléennes (Vrai/Faux)	is_valid = True
complex	Nombres complexes	z = 2 + 3j

20

Programmation Python	Syntaxe de base
<p><b>Typage des variables</b></p> <ul style="list-style-type: none"> <li>Les types de base:</li> </ul> <pre style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> In [17]: x=2.5           type(x)  Out[17]: float  In [18]: x = 'bonjour'           type(x)  Out[18]: str  In [19]: x = True           type(x)  Out[19]: bool  In [20]: x= 5 + 3j           type(x)  Out[20]: complex </pre>	

21

Programmation Python	Syntaxe de base
<p><b>Lecture/Ecriture</b></p> <p>En Python, nous utilisons pour la lecture et l'écriture les fonctions :</p> <p><b>1. La fonction print</b></p> <p>La fonction <b>print()</b> permet d'afficher du texte ou des valeurs dans la console. Elle est très flexible et peut formater les données de diverses façons.</p> <pre style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> In [4]: print("Bonjour, les amis!")            Bonjour, les amis!  In [5]: age = 25           print("Ahmed a", age, "ans.")            Ahmed a 25 ans. </pre>	

22

## Lecture/Ecriture

Utilisation de **sep** et **end**

- **sep** : Spécifie le séparateur entre les valeurs.
- **end**: Spécifie ce qui est ajouté à la fin de l'affichage (par défaut, un saut de ligne).

```
In [9]: print("Art", "Bricolage", "Musique", sep=", ") # Utilisation de virgules comme séparateur
        print("Bonjour", end="!") # Pas de saut de ligne
        print(" Comment ça va ?")
```

```
Art, Bricolage, Musique
Bonjour! Comment ça va ?
```

Formatage avec **f-strings**

Les **f-strings** permettent d'insérer des variables ou des expressions dans une chaîne.

```
In [17]: nom = "Ahmed"
        age = 25
        print(f"Bonjour,{nom}! Tu as {age} ans.")
        Bonjour,Ahmed! Tu as 25 ans.
```

23

## Lecture/Ecriture

**2. La fonction input**

La fonction `input()` permet de recevoir une entrée de l'utilisateur sous forme de chaîne de caractères.

```
In [23]: print("Comment t'appelles-tu ? ")
        nom = input()
        print(f"Bonjour, {nom}!")
        Comment t'appelles-tu ?
        Ahmed
        Bonjour, Ahmed!
```

**Convertir une entrée en un autre type**

Toutes les entrées via `input` sont des chaînes de caractères. Si vous attendez un entier ou un autre type, vous devez convertir l'entrée.

```
In [24]: print("Quel âge as-tu ? ")
        age = input()
        age = int(age) # Convertir la chaîne en entier
        print(f"Dans 5 ans, tu auras {age + 5} ans.")
        Quel âge as-tu ?
        26
        Dans 5 ans, tu auras 31 ans.
```

24

## Lecture/Ecriture

**Combinaison de print et input**

Un exemple classique qui combine print et input :

Exemple :

```
In [22]: nom = input("Quel est ton nom ? ")
age = int(input("Quel âge as-tu ? "))
print(f"Enchanté, {nom}! Tu as {age} ans.")
```

```
Quel est ton nom ? Ahmed
Quel âge as-tu ? 26
Enchanté, Ahmed! Tu as 26 ans.
```

25

## Lecture/Ecriture

Utilisez un bloc try pour gérer les erreurs de conversion.

Exemple :

```
In [21]: try:
age = int(input("Quel âge as-tu ? "))
print(f"Dans 10 ans, tu auras {age + 10} ans.")
except ValueError:
print("Veuillez entrer un nombre valide.")
```

```
Quel âge as-tu ? m
Veuillez entrer un nombre valide.
```

26

Programmation Python

Syntaxe de base

Opérateurs de base

Les opérateurs de base en Python permettent de manipuler des valeurs et des variables dans des expressions.

1. Opérateurs arithmétiques

Utilisés pour effectuer des calculs mathématiques.

Opérateur	Description	Exemple	Résultat
+	Addition	5 + 3	8
-	Soustraction	10 - 4	6
*	Multiplication	7 * 2	14
/	Division	10 / 2	5.0
//	Division entière	10 // 3	3
%	Modulo (reste de division)	10 % 3	1
**	Puissance	2 ** 3	8

27

Programmation Python

Syntaxe de base

Opérateurs de base

2. Opérateurs de comparaison

Permettent de comparer des valeurs. Ils retournent un booléen ( True ou False ).

Opérateur	Description	Exemple	Résultat
==	Égalité	5 == 5	True
!=	Différent	5 != 3	True
>	Supérieur	5 > 3	True
<	Inférieur	3 < 5	True
>=	Supérieur ou égal	5 >= 5	True
<=	Inférieur ou égal	3 <= 5	True

28

Programmation Python

Syntaxe de base

Opérateurs de base

3. Opérateurs logiques

Utilisés pour combiner des expressions conditionnelles.

Opérateur	Description	Exemple	Résultat
and	Retourne <code>True</code> si les deux expressions sont vraies	<code>(5 &gt; 3) and (8 &gt; 6)</code>	<code>True</code>
or	Retourne <code>True</code> si au moins une expression est vraie	<code>(5 &gt; 3) or (8 &lt; 6)</code>	<code>True</code>
not	Inverse la valeur booléenne	<code>not(5 &gt; 3)</code>	<code>False</code>

4. Opérateurs d'affectation

Utilisés pour assigner des valeurs aux variables.

Opérateur	Description	Exemple	Résultat
=	Affectation simple	<code>x = 5</code>	<code>x = 5</code>
+=	Ajout et assignation	<code>x += 3 (x = x + 3)</code>	<code>x = 8</code>
-=	Soustraction et assignation	<code>x -= 2 (x = x - 2)</code>	<code>x = 6</code>
*=	Multiplication et assignation	<code>x *= 4 (x = x * 4)</code>	<code>x = 24</code>
/=	Division et assignation	<code>x /= 2 (x = x / 2)</code>	<code>x = 12</code>
//=	Division entière et assignation	<code>x //= 3</code>	<code>x = 4</code>
%=	Modulo et assignation	<code>x %= 3 (x = x % 3)</code>	<code>x = 1</code>
**=	Puissance et assignation	<code>x **= 2 (x = x ** 2)</code>	<code>x = 1</code>

29

Programmation Python

Syntaxe de base

Opérateurs de base

5. Opérateurs sur les bits

Permettent de manipuler les bits des entiers.

Opérateur	Description	Exemple	Résultat
&	ET bit à bit	<code>5 &amp; 3</code>	<code>1</code>
	OU bit à bit	<code>5   3</code>	<code>7</code>
^	OU exclusif (XOR)	<code>5 ^ 3</code>	<code>6</code>
~	Négation bit à bit	<code>~5</code>	<code>-6</code>
<<	Décalage à gauche	<code>5 &lt;&lt; 1</code>	<code>10</code>
>>	Décalage à droite	<code>5 &gt;&gt; 1</code>	<code>2</code>

6. Opérateurs d'appartenance

Permettent de vérifier si un élément est présent dans une séquence (liste, chaîne, etc.).

Opérateur	Description	Exemple	Résultat
in	Vrai si l'élément est présent	<code>'a' in 'apple'</code>	<code>True</code>
not in	Vrai si l'élément est absent	<code>'b' not in 'apple'</code>	<code>True</code>

30

Programmation Python

Syntaxe de base

Opérateurs de base

7. Opérateurs d'identité

Permettent de vérifier si deux objets pointent vers la même zone mémoire.

Opérateur	Description	Exemple	Résultat
is	Vrai si les objets sont identiques	x is y	True ou False
is not	Vrai si les objets sont différents	x is not y	True ou False

31

Programmation Python

Syntaxe de base

Indentation

- Une indentation représente un ou plusieurs espaces au début d'une ligne de code.
- Il est recommandé avec Python d'indenter son code avec une tabulation (équivalent à 4 espaces).
- L'indentation est primordiale avec Python car elle sert à déterminer les blocs qui constituent votre code là où d'autres langages privilégient par exemple les accolades ({} ) pour spécifier ces blocs.

Block 1

Block 2

Block 3

Block 2, continuation

Block 1, continuation

32



Programmation Python

Syntaxe de base

Indentation

- Si l'indentation n'est pas correcte, nous nous retrouverons avec **IndentationError** Erreur.

Exemple :

```
if(5>2):  
    print("5 est supérieur que 2")  
  
5 est supérieur que 2
```

```
if(5>2):  
print("5 est supérieur que 2")
```

```
Input In [43]  
    print("5 est supérieur que 2")  
    ^  
IndentationError: expected an indented block
```

33

Programmation Python

Syntaxe de base

les instructions conditionnelles

Instruction if :

```
if condition:  
    # Bloc de code exécuté si La condition est vraie  
    print("Condition remplie !")
```

Exemple :

```
In [26]: age = 20  
         if age >= 18:  
             print("Vous êtes majeur.")  
  
         Vous êtes majeur.
```

34

Programmation Python	Syntaxe de base
<p style="color: #D2691E; margin: 0;">les instructions conditionnelles</p> <p style="color: #DC143C; margin: 0;"><b>Instruction if-else :</b></p> <div style="background-color: #F0F0F0; padding: 10px; margin: 10px 0;"> <pre> if condition:     # Bloc de code exécuté si la condition est vraie     print("Condition remplie !") else:     # Bloc de code exécuté si la condition est fausse     print("Condition non remplie.") </pre> </div> <p style="margin: 0;"><b>Exemple :</b></p> <div style="display: flex; align-items: flex-start;"> <div style="margin-right: 10px; color: #4169E1;">In [27]:</div> <div style="background-color: #F0F0F0; padding: 10px; flex-grow: 1;"> <pre> age = 16 if age &gt;= 18:     print("Vous êtes majeur.") else:     print("Vous êtes mineur.") </pre> <p style="margin-top: 10px;">Vous êtes mineur.</p> </div> </div>	
35	

Programmation Python	Syntaxe de base
<p style="color: #D2691E; margin: 0;">les instructions conditionnelles</p> <p style="color: #DC143C; margin: 0;"><b>Instruction if-elif-else :</b></p> <div style="background-color: #F0F0F0; padding: 10px; margin: 10px 0;"> <pre> if condition1:     # Bloc de code si condition1 est vraie     print("Condition 1 remplie.") elif condition2:     # Bloc de code si condition1 est fausse et condition2 est vraie     print("Condition 2 remplie.") else:     # Bloc de code si aucune condition n'est remplie     print("Aucune condition remplie.") </pre> </div> <p style="margin: 0;"><b>Exemple :</b></p> <div style="display: flex; align-items: flex-start;"> <div style="margin-right: 10px; color: #4169E1;">In [28]:</div> <div style="background-color: #F0F0F0; padding: 10px; flex-grow: 1;"> <pre> note = 14 if note &gt;= 16:     print("Excellent") elif note &gt;= 12:     print("Bien") elif note &gt;= 10:     print("Passable") else:     print("Échec") </pre> <p style="margin-top: 10px;">Bien</p> </div> </div>	
36	

Programmation Python	Syntaxe de base
<p><b>les instructions conditionnelles</b></p> <p><b>Les conditions imbriquées</b></p> <p>Vous pouvez imbriquer des blocs if pour des conditions plus complexes.</p> <p><b>Exemple :</b></p> <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <pre style="margin: 0;">In [29]: age = 20           citoyen = True            if age &gt;= 18:               if citoyen:                   print("Vous pouvez voter.")               else:                   print("Vous devez être citoyen pour voter.")           else:               print("Vous êtes trop jeune pour voter.")</pre> <p style="margin-top: 10px;">Vous pouvez voter.</p> </div>	
37	

Programmation Python	Syntaxe de base
<p><b>les instructions conditionnelles</b></p> <p><b>La syntaxe conditionnelle courte (opérateur ternaire)</b></p> <p>Pour une condition simple, vous pouvez utiliser une expression if-else sur une seule ligne.</p> <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc; margin: 10px 0;"> <pre style="margin: 0;">variable = valeur1 if condition else valeur2</pre> </div> <p><b>Exemple :</b></p> <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <pre style="margin: 0;">In [30]: age = 18           statut = "majeur" if age &gt;= 18 else "mineur"           print(statut) # Affiche "majeur"</pre> <p style="margin-top: 10px;">majeur</p> </div>	
38	

Programmation Python

Syntaxe de base

les instructions itératives

**La boucle for**

La boucle for est utilisée pour itérer sur une séquence (liste, chaîne, range, etc.).

```
for variable in sequence:  
    # Bloc de code exécuté pour chaque élément de la séquence
```

Exemple avec liste:

```
In [31]: nombres = [1, 2, 3, 4, 5]  
for nombre in nombres:  
    print(nombre)
```

1

2

3

4

5

39

Programmation Python

Syntaxe de base

les instructions itératives

**La boucle for**

La boucle for est utilisée pour itérer sur une séquence (liste, chaîne, range, etc.).

Exemple avec chaîne de caractères :

```
In [32]: mot = "Python"  
for lettre in mot:  
    print(lettre)
```

p

y

t

h

o

n

40

Pr. Louzaoui Khadija

20

Programmation Python

Syntaxe de base

les instructions itératives

La boucle for

La boucle for est utilisée pour itérer sur une séquence (liste, chaîne, range, etc.).

Exemple avec range:

In [33]:

```
for i in range(5): # Génère Les nombres de 0 à 4
    print(i)

# Exemple avec une plage personnalisée
for i in range(1, 10, 2): # De 1 à 9 avec un pas de 2
    print(i)
```

0

1

2

3

4

1

3

5

7

9

41

Programmation Python

Syntaxe de base

les instructions itératives

La boucle while

La boucle while répète un bloc de code tant qu'une condition est vraie.

Exemple :

In [34]:

```
compteur = 0
while compteur < 5:
    print(compteur)
    compteur += 1 # Incrémentation
```

0

1

2

3

4

42

Programmation Python

Syntaxe de base

les instructions itératives

Les mots-clés break et continue

**Break:** Permet de sortir immédiatement de la boucle, même si la condition ou la séquence n'est pas terminée.

```
In [35]: for i in range(10):
         if i == 5:
             break # Quitte la boucle
         print(i)
```

0  
1  
2  
3  
4

**Continue:** Permet de passer directement à l'itération suivante, en sautant le reste du bloc de code.

```
In [36]: for i in range(10):
         if i % 2 == 0:
             continue # Passe à l'itération suivante
         print(i) # Affiche seulement les nombres impairs
```

1  
3  
5  
7  
9

43

Programmation Python

Syntaxe de base

les instructions itératives

Boucles imbriquées

Une boucle peut être placée à l'intérieur d'une autre boucle.

```
In [37]: for i in range(3):
         for j in range(2):
             print(f"i={i}, j={j}")
```

i=0, j=0  
i=0, j=1  
i=1, j=0  
i=1, j=1  
i=2, j=0  
i=2, j=1

44

Programmation Python

Syntaxe de base

les instructions itératives

La clause else avec les boucles

En Python, les boucles peuvent avoir une clause **else**, qui s'exécute si la boucle termine sans interruption **(break)**.

In [46]:

```
for i in range(5):  
    if i == 10:  
        break  
else:  
    print("Boucle terminée sans interruption.")
```

Boucle terminée sans interruption.

In [47]:

```
for i in range(5):  
    if i == 3:  
        break  
else:  
    print("Boucle terminée sans interruption.")
```

In [48]:

```
compteur = 0  
while compteur < 5:  
    compteur += 1  
    if compteur == 6:  
        break  
else:  
    print("Boucle terminée sans interruption.")
```

Boucle terminée sans interruption.

In [49]:

```
compteur = 0  
while compteur < 5:  
    compteur += 1  
    if compteur == 3:  
        break  
else:  
    print("Boucle terminée sans interruption.")
```

45

Les tableaux en Python

❑

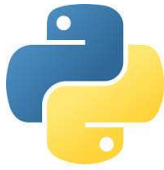
En Python, les tableaux sont des **structures de données** utilisées pour stocker une collection d'éléments.

❑

Python ne dispose pas de **tableaux natifs** comme certains autres langages (C, Java).

❑

Python propose plusieurs structures semblables, avec les **listes** et les **modules spécialisés (array, Numpy)**



46

Pr. Louzaoui Khadija

23

### Tableau avec le module array

Le module **array** fournit une structure de données similaire à un tableau, mais elle est plus restrictive. Tous les éléments doivent être du même type.

#### Création d'un tableau :

```
In [1]: import array

# Syntaxe : array(typecode, [éléments])
tab = array.array('i', [1, 2, 3, 4, 5]) # 'i' signifie entier signé
print(tab)

array('i', [1, 2, 3, 4, 5])
```

**Note** : La première lettre (comme 'i' dans l'exemple) représente le type des éléments. Par exemple, 'i' est pour les entiers.

47

### Tableau avec le module array

#### Typecodes disponibles

Chaque tableau doit être initialisé avec un typecode qui définit le type des éléments :

- 'b': entier signé (1 octet)
- 'B': entier non signé (1 octet)
- 'u': caractère Unicode
- 'h': entier signé (2 octets)
- 'H': entier non signé (2 octets)
- 'i': entier signé (4 octets)
- 'I': entier non signé (4 octets)
- 'f': nombre à virgule flottante (4 octets)
- 'd': nombre à virgule flottante (8 octets)

48



Les tableaux en Python

### Tableau avec le module array

Opérations de base:

Ajout d'éléments :

```
In [2]: tab.append(6) # Ajoute un élément à la fin
print(tab)

array('i', [1, 2, 3, 4, 5, 6])
```

Insertion :

```
In [3]: tab.insert(2, 10) # Insère 10 à l'indice 2
print(tab)

array('i', [1, 2, 10, 3, 4, 5, 6])
```

Suppression :

```
In [4]: tab.remove(3) # Supprime la première occurrence de 3
print(tab)

array('i', [1, 2, 10, 4, 5, 6])
```

49

Les tableaux en Python

### Tableau avec le module array

Opérations de base:

Accès et modification :

```
In [5]: print(tab[1]) # Accède à l'élément à l'indice 1
tab[1] = 20 # Modifie l'élément à l'indice 1
print(tab)

2
array('i', [1, 20, 10, 4, 5, 6])
```

Parcours :

```
In [6]: for elem in tab:
print(elem)

1
20
10
4
5
6
```

50

## Listes

Les **listes** :

- ❑ sont le type de tableau le plus utilisé en Python.
- ❑ peuvent contenir des éléments de différents types
- ❑ sont dynamiques (leur taille peut changer)
- ❑ offrent de nombreuses fonctionnalités.

**Création d'une liste :**

```
In [1]: # Liste vide
ma_liste = []

# Liste avec des éléments
ma_liste = [1, 2, 3, 4, 5]

# Liste avec des types différents
ma_liste = [1, "Python", 3.14, True]
```

51

**Opérations courantes sur les listes :**

### Append

Permet d'ajouter un élément à la liste

```
# Liste avec des types différents
ma_liste = [1, "Python", 3.14, True]

In [2]: # Ajouter un élément
ma_liste.append(6)

In [3]: ma_liste

Out[3]: [1, 'Python', 3.14, True, 6]
```

52

Les tableaux en Python

Opérations courantes sur les listes :

Remove

Permet de supprimer un élément à la liste

```
In [3]: ma_liste
Out[3]: [1, 'Python', 3.14, True, 6]

In [5]: # Supprimer un élément
ma_liste.remove(3.14)

In [6]: ma_liste
Out[6]: [1, 'Python', True, 6]
```

```
In [3]: ma_liste
Out[3]: [1, 'Python', 3.14, True, 6]

In [4]: # Supprimer un élément
ma_liste.remove(3)

-----
ValueError                                Traceback (most recent call last)
Cell In[4], line 2
      1 # Supprimer un élément
----> 2 ma_liste.remove(3)

ValueError: list.remove(x): x not in list
```

53

Les tableaux en Python

Opérations courantes sur les listes :

Accéder à un élément

```
In [6]: ma_liste
Out[6]: [1, 'Python', True, 6]

In [7]: # Accéder à un élément (index commence à 0)
print(ma_liste[1]) # Affiche le deuxième élément

Python
```

54

Les tableaux en Python

Opérations courantes sur les listes :  
Extraction d'une sous liste

```
In [9]: print(ma_liste)

[1, 'Python', True, 6]
```

```
In [10]: # Slicing (extraction d'une sous-liste)
print(ma_liste[1:4]) # Affiche les éléments de l'index 1 à 3

['Python', True, 6]
```

55

Les tableaux en Python

Opérations courantes sur les listes :  
**len**

Permet de donner la longueur de la liste

```
In [12]: print(ma_liste)
# Longueur de la liste
print(len(ma_liste))

[1, 'Python', True, 6]
4
```

56

## Opérations courantes sur les listes :

**sort**

Permet de trier les éléments de la liste

```
In [17]: ma_liste = [15, -2, 53, 14, -5]
          print(ma_liste)
          # Trier la liste
          ma_liste.sort()
          print(ma_liste)

          [15, -2, 53, 14, -5]
          [-5, -2, 14, 15, 53]
```

```
In [20]: # Liste avec des types différents
          ma_liste = [1, "Python", 3.14, True]
          print(ma_liste)
          # Trier la liste
          ma_liste.sort()
          print(ma_liste)

          [1, 'Python', 3.14, True]

-----
TypeError                                 Traceback (most recent call last)
Cell In[20], line 5
      3 print(ma_liste)
      4 # Trier la liste
----> 5 ma_liste.sort()
      6 print(ma_liste)

TypeError: '<' not supported between instances of 'str' and 'int'
```

57

## Tableau avec la bibliothèque NumPy

**NumPy** (Numerical Python) est une bibliothèque puissante pour effectuer des calculs numériques en Python, avec une efficacité bien supérieure à celle des listes natives. Elle est particulièrement utile pour manipuler des tableaux multidimensionnels (ou matrices) et effectuer des opérations mathématiques.

**Création d'un tableau**

```
import numpy as np

# Tableau 1D
tab = np.array([1, 2, 3, 4, 5])
print(tab)

# Tableau 2D
tab_2d = np.array([[1, 2], [3, 4], [5, 6]])
print(tab_2d)
```

58

## Tableau avec la bibliothèque NumPy

### Types et Propriétés des tableaux

```
tab = np.array([1, 2, 3], dtype='float64') # Tableau de flottants
print(tab)
```

```
[1. 2. 3.]
```

```
print(tab.shape) # Dimensions du tableau
print(tab.ndim)  # Nombre de dimensions
print(tab.size)  # Nombre total d'éléments
print(tab.dtype) # Type des éléments
```

```
(3,)
1
3
float64
```

## Tableau avec la bibliothèque NumPy

### Opérations mathématiques

NumPy permet d'effectuer des opérations mathématiques directement sur les tableaux :

```
tab = np.array([1, 2, 3, 4])

# Addition
print(tab + 2)

# Multiplication
print(tab * 3)

# Opérations élément par élément
tab2 = np.array([5, 6, 7, 8])
print(tab + tab2)
```

```
[3 4 5 6]
[ 3  6  9 12]
[ 6  8 10 12]
```

Les tableaux en Python

Tableau avec la bibliothèque NumPy

Fonctions utiles

```
print(np.zeros((2, 3))) # Tableau de zéros
print(np.ones((3, 2))) # Tableau de uns
print(np.arange(1, 10, 2)) # Équivalent à range()
print(np.linspace(0, 1, 5)) # Divise l'intervalle [0, 1] en 5 parties
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1.]
 [1. 1.]
 [1. 1.]]
[1 3 5 7 9]
[0. 0.25 0.5 0.75 1. ]
```

61

Les tableaux en Python

Tableau avec la bibliothèque NumPy

Fonctions utiles

```
tab = np.array([1, 2, 3, 4, 5])
print(tab.mean()) # Moyenne
print(tab.sum()) # Somme
print(tab.min()) # Minimum
print(tab.max()) # Maximum
```

```
3.0
15
1
5
```

```
tab = np.array([1, 2, 3, 4, 5, 6])
print(tab.reshape((2, 3))) # Transforme en tableau 2x3
```

```
[[1 2 3]
 [4 5 6]]
```

62

Les tableaux en Python

Tableau avec la bibliothèque NumPy

Différences entre array et NumPy

Caractéristique	Module array	NumPy
Type des éléments	Limité (types de base)	Très varié
Dimensions	Unidimensionnel	Multi-dimensionnel
Performance	Moins optimisé	Très performant (C optimisé)
Fonctionnalités	Basique	Avancé (opérations mathématiques, algèbre linéaire, etc.)

63

Les tableaux en Python

Comparaison entre Listes Python et NumPy

Caractéristique	Listes Python	NumPy
Performance	Moins rapide	Très performant
Fonctions avancées	Limitées	Riches (mathématiques, etc.)
Facilité d'usage	Simple	API dédiée (besoin d'apprentissage)
Dimension	Dynamique	Fixée après création

64



**Tableaux à deux dimensions:**

En Python, un tableau à deux dimensions peut être représenté de différentes manières.

**Listes imbriquées**

Une liste à deux dimensions est simplement une liste contenant d'autres listes.

```
# Création d'un tableau 2D
tableau = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Accès à un élément
print(tableau[1][2]) # 6 (ligne 1, colonne 2)

# Modification d'un élément
tableau[0][0] = 10
print(tableau)
```

65

**Tableaux à deux dimensions:****Itération sur un tableau 2D :**

```
# Parcourir le tableau ligne par ligne
for ligne in tableau:
    print(ligne)

# Parcourir chaque élément
for i in range(len(tableau)):
    for j in range(len(tableau[i])):
        print(f"Élément [{i}][{j}] = {tableau[i][j]}")
```

66

**Tableaux à deux dimensions:****NumPy**

La bibliothèque NumPy est optimisée pour les opérations sur des tableaux (matrices) et est plus performante que les listes imbriquées.

```
import numpy as np

# Création d'un tableau 2D
tableau = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

# Accès à un élément
print(tableau[1, 2]) # 6 (ligne 1, colonne 2)

# Modification d'un élément
tableau[0, 0] = 10
print(tableau)
```

67

**Tableaux à deux dimensions:****Opérations avec NumPy :**

```
# Ajouter un scalaire à tous les éléments
tableau += 5
print(tableau)

# Somme de tous les éléments
print(np.sum(tableau))

# Transposer le tableau
print(tableau.T)

# Extraction d'une colonne
print(tableau[:, 1]) # Colonne 1
```

68

**Tableaux à deux dimensions:****Pandas**

Si votre tableau représente des données tabulaires avec des noms pour les lignes et colonnes, Pandas est une excellente option.

```
import pandas as pd

# Création d'un tableau 2D avec Pandas
tableau = pd.DataFrame({
    'Colonne1': [1, 4, 7],
    'Colonne2': [2, 5, 8],
    'Colonne3': [3, 6, 9]
})

# Afficher le tableau
print(tableau)

# Accès à une cellule
print(tableau.loc[1, 'Colonne3']) # 6 (ligne 1, colonne "Colonne3")

# Modifier une cellule
tableau.loc[0, 'Colonne1'] = 10
print(tableau)
```

69

**Tableaux à deux dimensions:****Opérations avec Pandas**

```
# Accéder à une colonne
print(tableau['Colonne2'])

# Filtrer les données
print(tableau[tableau['Colonne1'] > 5])
```

70

### Applications des tableaux

- ☐ Manipulation de données numériques
- ☐ Traitement d'images
- ☐ Analyse de données avec **Pandas**
- ☐ Calculs scientifiques avec **SciPy**