Université de AbdelHafid Boussouf - Mila

# CHAPITRE N° 4:

## LES STRUCTURES DE DONNÉES

## LISTES, TUPLES, DICTIONNAIRES ET ENSEMBLE.

**L1 Mathématiques appliquées**

# Introduction générale aux structures de données

➢En programmation, les données sont rarement manipulées de manière isolée. Dans de nombreuses situations, il est nécessaire de regrouper plusieurs valeurs dans une seule entité logique : liste de notes, coordonnées d'un vecteur, ensemble de résultats expérimentaux, association entre un nom et un numéro, etc.

➢Une structure de données permet d'organiser ces informations de manière cohérente afin de faciliter :

- ✓le stockage,
- ✓ l'accès,
- ✓ la modification,
- ✓le traitement algorithmique.

➢Python fournit plusieurs structures natives pour répondre à ces besoins : les listes, les tuples, les dictionnaires et les ensembles.

# Pourquoi plusieurs structures ?

Il n'existe pas une seule manière de stocker un groupe de données. Le choix dépend de la nature du problème.

Par exemple :

- si l'on veut une collection ordonnée et modifiable, on utilisera une liste ;
- si l'on veut des données ordonnées mais fixes, on utilisera un tuple ;
- si l'on veut associer une clé à une valeur, on utilisera un dictionnaire ;
- si l'on veut éliminer les doublons, on utilisera un ensemble.

Ainsi, comprendre les différences entre ces structures est essentiel pour écrire un programme clair, efficace et adapté.

# I- Partie Principale : Les listes

# Plan du Cours

1. Introduction aux listes en Python.
2. Definition.
3. Création des listes.
4. Accès aux éléments d'une liste.
5. Modification des éléments d'une liste.
6. Ajout d'éléments dans une liste.
7. Suppression des éléments dans une liste.
8. Fonctions utiles sur les listes.
9. Slicing des listes.
10. Parcourir une liste.

# 1- Introduction aux listes en Python

Dans la programmation, il est fréquent de manipuler plusieurs valeurs liées entre elles.

**Exemples :**

- notes d'un étudiant.
- coordonnées d'un vecteur.
- résultats d'une expérience.

## ❌ Problème sans liste

Sans utiliser une structure adaptée, on serait obligé de créer plusieurs variables :

```
a = 10
b = 12
c = 15
```

☞ Cela devient difficile à gérer lorsque le nombre de données augmente.

# 1-Introduction aux listes en Python(suite)

## Solution : utiliser une liste

Une liste permet de regrouper toutes ces valeurs dans une seule variable :

notes = [10, 12, 15]

☞ Cela permet de:

- ❑organiser les données,
- ❑simplifier le code,
- ❑faciliter les calculs.

## 2- Définition :

Une liste en Python est une structure de données :

- ✓ **Ordonnée**

Les éléments sont organisés dans un ordre précis.

- ✓ **Mutable**

On peut modifier, ajouter ou supprimer des éléments.

- ✓ **Dynamique**

La taille peut changer pendant l'exécution.

- ✓ **Hétérogène**

Elle peut contenir différents types.

**Exemple:**

L = [1, "hello", 3.5, True]

☞ En pratique, on utilise souvent des listes homogènes (ex : nombres).

# 3- Création de listes:

En Python, une liste peut être créée de plusieurs manières selon le besoin.

## 1) Méthode 1 : crochets

```
L = [1, 2, 3]   #C'est la méthode la plus simple et la plus  utilisée.
```

## 2) Méthode 2 : liste vide

```
L = []          #Utile lorsque l'on veut créer une liste et ajouter  les éléments plus tard.
```

## 3) Méthode 3 : constructeur

```
L = list((1, 2, 3))  #Permet de créer une liste à partir d'un autre objet (tuple, chaîne…)
```

## ✓ Exemple

```
L = [5, 10, 15]
print(L)
```

Résultat :

```
[5, 10, 15]
```

# 4-Accès aux éléments d'une liste

En Python, chaque élément d'une liste possède une position appelée **index**.

☞ Important **:** L'index commence toujours à **0.**

**Exemple:**

```
L = [10, 20, 30]
print(L[0])    # premier élément
print(L[1])    # deuxième élément
```

**Résultat :**

```
10
20
```

# 4-Accès aux éléments d'une liste (suite)

## Index négatif

Python permet aussi d'accéder aux éléments en partant de la fin de la liste.

☞ Très utile pour récupérer rapidement le dernier élément et éviter de calculer la taille.

**Exemple:**

```
L = [10, 20, 30]
print(L[-1]) # dernier élément
```

**Résultat :**

30

# 4-Accès aux éléments d'une liste (suite)

## ⚠ Attention (erreur fréquente)

- Si on dépasse la taille de la liste :

Exemple:

```
L = [10, 20, 30]
print(L[5])
```

**❌ Résultat :**

```
IndexError: list index out of range
```

☞ Cela signifie que l'index demandé n'existe pas.

# 5-Modification des éléments d'une liste

En Python, les listes sont **modifiables (mutable)**.

☞ Cela signifie que l'on peut :

- ✓changer un élément.
- ✓modifier une valeur existante.
- ✓mettre à jour une liste sans la recréer.

**Exemple :**

```
L = [1, 2, 3]
L[1] = 100
print(L)
```

**Résultat**

```
[1, 100, 3] # L'élément à l'index 1 a été remplacé.
```

# 6-Ajout d'éléments dans une liste

En Python, il existe plusieurs méthodes pour ajouter des éléments à une liste.
Le choix de la méthode dépend de ce que l'on veut ajouter (un seul élément ou plusieurs).

1. **Méthode append():** Ajoute un élément **à la fin de la liste.**

Exemple:

```
L = [1, 2]
L.append(3)
print(L)
L.append([4, 5])
print(L)
```

Résultat :

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

# 6-Ajout d'éléments dans une liste (suite)

**2. Méthode insert() :**Ajoute un élément à une **position précise.**

Exemple:

```
L = [1, 2]
L.insert(1, 10)
print(L)
```

Résultat :

```
[1, 10, 2]
```

# 6-Ajout d'éléments dans une liste (suite)

**3. Méthode extend():** Ajoute **plusieurs éléments** à la liste.

Exemple:

```
L = [1, 2]
L.extend([4, 5])
print(L)
```

Résultat :

```
[1, 2, 4, 5]
```

# 7-Suppression des éléments dans une liste

En Python, on peut supprimer des éléments d'une liste de plusieurs manières selon le besoin.

1. **remove(valeur):** Supprime un élément **par sa valeur.**

Exemple:

```
L = [10, 20, 30]
L.remove(10)
print(L)
```

Résultat :

```
[20, 30]
```

✓ Supprime **la première occurrence.**

✓ Utilisé quand on connaît la valeur.

**⚠Erreurs fréquentes**

```
L = [1, 2, 3]
L.remove(5) # ✗ erreur
```

☞ Erreur :

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

## 7-Suppression des éléments dans une liste (suite)

**2. pop():** Supprime le **dernier élément.**

Exemple:

```
L = [10, 20, 30]
L.pop()
print(L)
```

Résultat :

```
[10, 20]
```

## 7-Suppression des éléments dans une liste (suite)

**3. pop(index):** Supprime un élément par sa **position (index).**

Exemple:

```
L = [10, 20, 30]
L.pop(1)
print(L)
```

Résultat :

```
[10, 30]
```

# 8-Fonctions utiles sur les listes

Python fournit plusieurs fonctions de base intégrées pour manipuler facilement les listes.

Exemple:

```
L = [4, 2, 8, 2, 10]

print(len(L))
print(max(L))
print(min(L))
print(sum(L))
print(sorted(L))
print(L.count(2))
print(L.index(8))
print(5 in L)
L.reverse()
print(L)
```

Résultat

```
5
10
2
26
[2, 2, 4, 8, 10]
2
2
False

[10, 2, 8, 2, 4]
```

# 9-Slicing des listes

Le **slicing** permet d'extraire une partie d'une liste.

## Syntaxe générale :

L[start : end : step]

start → début (inclus)
end → fin (exclus)
step → pas (intervalle)

## Exemple

```
L = [1, 2, 3, 4, 5]
print(L[1:4])   # éléments de l'index 1 à 3
print(L[:3])    # du début jusqu'à index 2
print(L[::2])   #prendre un élément puis sauter un élément
```

## Résultats

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

# 10-Parcours d'une liste

Le parcours d'une liste se fait généralement avec une boucle for.

Exemple

```
L = [30,45,80,10]
for x in L:
print(x)
```

Résultats

```
30
45
80
10
```

# II- Partie SECONDAIRE: Apercu Des autres Structures

# 1-Les Tuples

Un tuple est une structure proche de la liste, mais non modifiable.

Exemple

```
T = (11, 23, 13)
print(T)
```

Résultats

```
(11, 23, 13)
```

⇨ Les tuples sont utilisés lorsque les données doivent rester fixes.

# 2-Les dictionnaires

Un dictionnaire associe une clé à une valeur.

Exemple

```
D = {"nom": "Ali", "age": 20}
print(D["nom"])
```

Résultats

```
Ali
```

⇨ Cette structure est utile pour représenter des informations Décrites par des attributs nommés.

# 3-Les ensembles

Un ensemble est une collection sans doublons.

Exemple

```
S = {1, 2, 3, 3,1}
print(S)
```

Résultats

```
{1, 2, 3}
```

⇨ Les ensembles sont utiles pour éliminer les répétitions et effectuer des opérations ensemblistes.

# Vue d'ensemble des structures Python

| Structure | Syntaxe | Ordre | Modifiable | Doublons | Accès |
|---|---|---|---|---|---|
| Liste | [ ] | Oui | Oui | Oui | par index |
| Tuple | ( ) | Oui | Non | Oui | par index |
| Dictionnaire | {clé: valeur} | par clés | Oui | clés uniques | par clé |
| Ensemble | { } | Non pertinent | Oui | Non | pas d'index |

# Conclusion

- Les structures de données permettent d'organiser et manipuler efficacement les informations en programmation.
- Python propose plusieurs structures : listes, tuples, dictionnaires et ensembles.
- Les **listes** sont la structure la plus utilisée grâce à leur flexibilité et leur simplicité.
- Elles permettent d'ajouter, modifier, supprimer et parcourir des données facilement.
- Les **tuples** sont utilisés pour des données fixes.
- Les **dictionnaires** permettent d'associer des clés à des valeurs.
- Les **ensembles** sont utiles pour éliminer les doublons.
- Une bonne maîtrise des listes facilite la compréhension des autres structures.
- Le choix de la structure dépend toujours du problème à résoudre.