Introduction aux listes en python

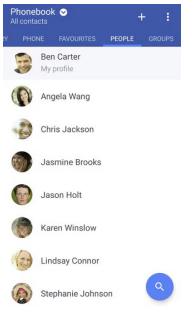
Lionel Chatelain – Gymnase de Chamblandes

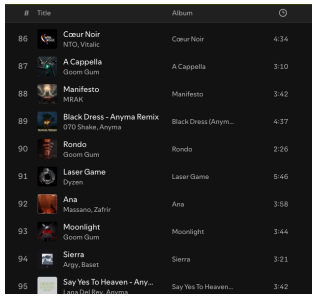
2024-2025

1

Organiser et gérer efficacement plusieurs données

Imaginez que vous devez gérer votre playlist musicale, une liste de courses, ou encore les devoirs que vous devez faire cette semaine. Comment organiser tout cela efficacement en Python ?





2

Pourquoi avons-nous besoin de listes ?

- Gérer une playlist musicale
- Organiser une liste de courses
- Gérer l'annuaire des élèves du gymnase
- Gérer les listes d'absences
- Organiser les livres à la bibliothèque
- Organiser les contacts téléphoniques / réseaux sociaux
- Recenser les habitants d'une commune
- Trier des contacts par ordre alphabétique

→ Comment stocker plusieurs données dans une seule **variable** ?

3

Les listes en Python

- **Définition:** Une liste est une collection ordonnée d'éléments, qui peuvent être différents
- **Exemple simple:** ma_liste = [1, 2, 3, 4]
- **Point clé:** Les éléments d'une liste sont accessibles via leur position (index).

4

Créer une liste

- **Syntaxe de base:**
  - ma_liste = [élément1, élément2, élément3]
- **Exemples:**
  - Liste de nombres: [1, 2, 3, 4]
  - Liste de chaînes: ["chanson1", "chanson2", "chanson3"]
  - Liste mixte: [1, "chanson2", 3.5]

5

Accéder aux éléments d'une liste

- **Rappel point clé:** Les éléments d'une liste sont accessibles via leur position (index).
- **Accès par index:**
  - ma_liste[0] donne le premier élément.
  - ma_liste[-1] donne le dernier élément.
- **Remarque:** L'indexation commence à 0 !
- **Exemple pratique**
  - Soit la liste fruits = ["orange", "banane", "pomme", "citron", "fraise"]
  - Que vaut fruits[2] ?
  - Que vaut fruits[-3] ?

6

Modifier les éléments d'une liste

- **Modifier un élément:**
  - ma_liste[1] = "nouveau_valeur"
- **Exemple pratique:**
  - Modifier un élément de notre liste de fruits

  fruits = ["orange", "banane", "pomme", "citron", "fraise"]

  →Comment modifier "orange" par "poire" ?
  →Comment modifier "citron" par "mangue" ?

7

Ajouter et supprimer des éléments

- **Ajouter un élément:**
  - ma_liste.append(nouvel_élément)
- **Supprimer un élément:**
  - ma_liste.remove(élément)
- **Exemple sur Thonny:**
  - Ajouter un nouvel élément à une liste de fruits
  - Supprimer un nouvel élément à une liste de fruits

8

Parcourir une liste (boucler sur une liste)

- **Utilisation de for i in range:**
  - Boucler sur les éléments de la liste.
- **Exemple:**
  - Afficher chaque élément d'une liste de fruits.

```
fruits = ["orange", "banane", "pomme", "citron", "fraise"]

for i in fruits:
    print(i)
```

9

Quelques fonctions utiles avec les listes

- **Fonctions communes:**
  - len(ma_liste) : Taille de la liste.
  - sorted(ma_liste) :Tri de la liste.
  - min(ma_liste), max(ma_liste) : Valeurs minimale et maximale.
  - Index(element) : Donne la position d'un élément dans la liste.
- **Exemple pratique:**
  - Tri d'une liste de notes et affichage de la note minimale et maximale

→ Calculons ensemble la moyenne de cet élève !

10

Les listes : un outil essentiel en algorithmique

- **Stockage de données multiples :**
  - Les listes permettent de gérer et de manipuler plusieurs données de manière structurée, facilitant la résolution de problèmes complexes.
- **Organisation et tri :**
  - Les algorithmes de tri (comme le tri par insertion, le tri à bulles, ou le tri rapide) reposent souvent sur l'utilisation de listes pour organiser les données.
- **Recherche efficace :**
  - Les listes sont utilisées pour implémenter des algorithmes de recherche, tels que la recherche linéaire ou la recherche dichotomique, permettant de trouver rapidement un élément dans une grande collection.
- **Parcours et analyse :**
  - Grâce à la possibilité de parcourir les listes, on peut appliquer des algorithmes pour analyser les données, comme calculer des moyennes, trouver des maximums ou minimums, et détecter des patterns.

11

Conclusion et questions

- **Résumé des points clés:**
  - Le type «list» en Python est une structure de données plus complexe et flexible comparée aux types de base tels que int, str, bool, et float.
  - Création, accès, modification, ajout, suppression et boucles.
  - Le type «list» est un conteneur, capable de stocker plusieurs valeurs. Ces valeurs peuvent être de n'importe quel type (y compris d'autres listes), et la liste peut être modifiée après sa création (ajout, suppression d'éléments, etc.).

Des questions ?

12

Complément listes – suppression avec del

- En plus de pouvoir supprimer un élément par valeur (remove()) python permet aussi de supprimer un élément par index avec del(index):

```
fruits = ["pomme", "orange", "banane"]
del fruits[1]
print(fruits) # Affiche ["pomme", "banane"]
```

13

Complèments listes – problèmes avec la suppression

Lorsque vous supprimez des éléments d'une liste en la parcourant avec une boucle, les indices des éléments restants sont modifiés, ce qui peut provoquer des erreurs inattendues.

```
nombres = [1, 2, 3, 4, 5]
for i in range(len(nombres)):
        del nombres[i] # Cette ligne provoque une erreur logique
                                -> Pourquoi ?
```

14

Complément listes – Slicing I

- **Définition du slicing :**

Le slicing permet de récupérer une partie d'une liste, c'est-à-dire une sous-liste, en spécifiant un intervalle d'indices.

- **Syntaxe du slicing :**

```
ma_liste[start:end]
```

- **Où :**
  - start est l'indice de départ (inclus).
  - end est l'indice de fin (exclu).

→ Le slicing est une manière puissante de manipuler des listes sans avoir à écrire de boucles supplémentaires, surtout pour extraire ou copier des parties de listes."

15

Complèment listes – slicing II

- Exemple classique :

```
nombres = [10, 20, 30, 40, 50]
print(nombres[1:4]) # Affiche [20, 30, 40]
```

- Exemple avec omission d'indices

```
print(nombres[:3]) # Affiche [10, 20, 30]
print(nombres[2:]) # Affiche [30, 40, 50]
print(nombres[:]) # Affiche toute la liste [10, 20, 30, 40, 50]
```

- Slicing avec index négatifs

```
print(nombres[-3:]) # Affiche [30, 40, 50]
```

16