Python : Listes

1ère NSI

Plan du cours

1. Objet de type `list`
   - Notion de liste
   - Manipuler une liste
   - Caractère mutable d'une liste
   - Les *tuples* : des listes non mutables
2. Quelques fonctions utiles
3. Parcourir les éléments d'une liste

Notion de liste

- Les listes sont des objets Python qui peuvent contenir d'autres objets.
- Comme les chaînes de caractères, les listes sont des **séquences**, non pas de caractères, mais d'objets de types identiques ou différents.

Notion de liste

- Les listes sont des objets Python qui peuvent contenir d'autres objets.
- Comme les chaînes de caractères, les listes sont des **séquences**, non pas de caractères, mais d'objets de types identiques ou différents.
- Une liste se définit à l'aide de crochets :

```
1 liste1 = [1,2,3]                    # Liste d'entiers
2 liste2 = ["bonjour", "salut"]       # Chaînes de caractères
3 liste3 = ["Jean", "Dupont", 31] # Différents types
```

Notion de liste

- Les listes sont des objets Python qui peuvent contenir d'autres objets.
- Comme les chaînes de caractères, les listes sont des **séquences**, non pas de caractères, mais d'objets de types identiques ou différents.
- Une liste se définit à l'aide de crochets :

```
1 liste1 = [1,2,3]                    # Liste d'entiers
2 liste2 = ["bonjour", "salut"]       # Chaînes de caractères
3 liste3 = ["Jean", "Dupont", 31] # Différents types
```

- Pour accéder à un élément particulier, il suffit de renseigner l'**indice** de l'élément, comme avec les chaînes de caractères :

```
1 >>> liste = [10,26,3]
2 >>> liste[1]
3 26
```

Notion de liste

- On peut utiliser les *slices* pour extraire une partie de la liste :

```
1 >>> liste = [23, 46, 51, 37, 12, 80]
2 >>> liste[3:]
3 [37, 12, 80]
```

Notion de liste

- On peut utiliser les *slices* pour extraire une partie de la liste :

```
1 >>> liste = [23, 46, 51, 37, 12, 80]
2 >>> liste[3:]
3 [37, 12, 80]
```

- Le mot-clé `in` permet de savoir si un élément est dans la liste :

```
1 >>> liste = [13, 56, 24]
2 >>> 30 in liste
3 False
```

Manipuler une liste

- Pour **ajouter un élément** à une liste, on utilise la méthode `.append` :

```
1  >>> liste = [1,3,5,7]
2  >>> liste.append(9)
3  >>> liste
4  [1, 3, 5, 7, 9]
```

Manipuler une liste

- Pour **ajouter un élément** à une liste, on utilise la méthode `.append` :

```
1 >>> liste = [1,3,5,7]
2 >>> liste.append(9)
3 >>> liste
4 [1, 3, 5, 7, 9]
```

- On peut également *concaténer* deux listes comme on le ferait pour des chaînes de caractères :

```
1 >>> liste1 = [1,3,5,7]
2 >>> liste2 = [2,4,6,8]
3 >>> liste1 + liste2
4 [1, 3, 5, 7, 2, 4, 6, 8]
```

Manipuler une liste

- Contrairement aux chaînes de caractères, il est **possible de modifier** l'élément d'une liste de la façon suivante :

```
1 >>> prenoms = ["Alice", "Bertrand", "Charlie"]
2 >>> prenoms[1] = "Bob"
3 >>> prenoms
4 ['Alice', 'Bob', 'Charlie']
```

Manipuler une liste

- Contrairement aux chaînes de caractères, il est **possible de modifier** l'élément d'une liste de la façon suivante :

```
>>> prenoms = ["Alice", "Bertrand", "Charlie"]
>>> prenoms[1] = "Bob"
>>> prenoms
['Alice', 'Bob', 'Charlie']
```

- Pour **supprimer** un élément, on utilise le mot-clé `del` :

```
>>> premiers = [2,3,4,5,7,11,13]
>>> del premiers[2]   # On supprime l'élément d'indice 2
>>> premiers
[2, 3, 5, 7, 11, 13]
```

Caractère mutable d'une liste

- Un caractère très important des listes est le fait qu'elles soient **mutables** :

```
>>> liste = [1,2,3]
>>> copie = liste      # On 'copie' la liste
>>> copie.append(4)    # On ajoute 4 à la copie
>>> copie
[1, 2, 3, 4]           # copie contient maintenant 4
>>> liste
[1, 2, 3, 4]           # mais liste est aussi modifiée !
```

Caractère mutable d'une liste

- Un caractère très important des listes est le fait qu'elles soient **mutables** :

```
>>> liste = [1,2,3]
>>> copie = liste      # On 'copie' la liste
>>> copie.append(4)    # On ajoute 4 à la copie
>>> copie
[1, 2, 3, 4]           # copie contient maintenant 4
>>> liste
[1, 2, 3, 4]           # mais liste est aussi modifiée !
```

- Pour créer une copie distincte de la liste, on utilise la fonction `list` :

```
>>> liste = [1,2,3]
>>> copie = list(liste)  # On crée une 'vraie' copie
>>> copie.append(4)
>>> copie, liste
[1, 2, 3, 4], [1, 2, 3]
```

Les *tuples* : des listes non mutables

- Les **tuples** (ou *n-uplets* en français) sont des listes non mutables. Ils sont définis avec des parenthèses :

```
1  >>> T = (1,2,3)
2  >>> T[0]
3  1
```

- On ne peut **ni ajouter, ni modifier, ni supprimer** d'éléments comme dans une liste.

Les *tuples* : des listes non mutables

- Les **tuples** (ou *n-uplets* en français) sont des listes non mutables. Ils sont définis avec des parenthèses :

```
1  >>> T = (1,2,3)
2  >>> T[0]
3  1
```

- On ne peut **ni ajouter, ni modifier, ni supprimer** d'éléments comme dans une liste.
- Lorsqu'une fonction renvoie plusieurs éléments, elle renvoie implicitement un *tuple*.

```
1  def f(x):
2      return x, x+1
```

```
1  >>> f(1)
2  (1, 2)
```

Quelques fonctions utiles

- Comme tout objet en Python, les listes possèdent des méthodes qui leur sont propres :
  - `.append(a)` : ajoute l'objet `a` à la fin de la liste
  - `.count(a)` : renvoie le nombre de `a` dans la liste
  - `.index(a)` : renvoie l'indice du premier `a` dans la liste
  - `.pop()` : supprime et renvoie le dernier élément de la liste
  - `.remove(a)` : supprime le premier `a` dans la liste
  - `.sort()` : trie dans la liste dans l'ordre croissant

Quelques fonctions utiles

- Comme tout objet en Python, les listes possèdent des méthodes qui leur sont propres :
  - `.append(a)` : ajoute l'objet `a` à la fin de la liste
  - `.count(a)` : renvoie le nombre de `a` dans la liste
  - `.index(a)` : renvoie l'indice du premier `a` dans la liste
  - `.pop()` : supprime et renvoie le dernier élément de la liste
  - `.remove(a)` : supprime le premier `a` dans la liste
  - `.sort()` : trie dans la liste dans l'ordre croissant
- On peut également leur appliquer des fonctions communes à d'autres objets :
  - `max, min` : renvoie le maximum / le minimum d'une liste
  - `len` : renvoie le nombre d'éléments de la liste
  - `sum` : renvoie la somme des éléments de la liste
  - `sorted` : renvoie une copie de la liste triée dans l'ordre croissante

Parcourir les éléments d'une liste

Pour parcourir les éléments d'une liste, on peut utiliser une boucle `for` de plusieurs manières. Par exemple, pour afficher chaque élément de la liste :

Parcourir les éléments d'une liste

Pour parcourir les éléments d'une liste, on peut utiliser une boucle `for` de plusieurs manières. Par exemple, pour afficher chaque élément de la liste :

- le **parcours par éléments**

```
1  for element in liste:
2      print(element)
```

Parcourir les éléments d'une liste

Pour parcourir les éléments d'une liste, on peut utiliser une boucle `for` de plusieurs manières. Par exemple, pour afficher chaque élément de la liste :

- le **parcours par éléments**

```
1  for element in liste:
2      print(element)
```

- le **parcours par indices**

```
1  for i in range(len(liste)):
2      print(liste[i])
```

Parcourir les éléments d'une liste

Pour parcourir les éléments d'une liste, on peut utiliser une boucle `for` de plusieurs manières. Par exemple, pour afficher chaque élément de la liste :

- le **parcours par éléments**

```
1 for element in liste:
2     print(element)
```

- le **parcours par indices**

```
1 for i in range(len(liste)):
2     print(liste[i])
```

- l'**énumeration** : on dispose de l'indice ET de l'élément

```
1 for i, element in enumerate(liste):
2     print(element)
```

Exemple

Avec les listes, la méthode `.find(a)` (première occurrence de `a` dans la liste) n'existe pas. On peut coder une fonction analogue de la manière suivante :

```
"""
find(liste, a) : retourne l'indice de la première
occurrence de a dans la liste, -1 si a n'y est pas
"""

def find(liste, a):

    for indice, element in enumerate(liste):
        if element == a:
            return indice

    return -1
```