PYTHON AU LYCÉE (5) : LES LISTES

COURS

Python au lycée (5) : Les listes

DURÉE ESTIMÉE

15 minutes

OBJECTIFS DU CHAPITRE

- Créer une liste en compréhension
- Construire une liste pas à pas avec append()
- Parcourir une liste pour calculer une somme, une moyenne ou un extremum
- Tester l'appartenance d'un élément à une liste

1. Le type liste

Définition d'une liste en Python

Une *liste* est un type de valeur qui contient une suite ordonnée de valeurs.

On dit qu'une liste est définie *en extension* lorsque l'on énumère toutes les valeurs de la liste. Une liste est encadrée par des **crochets** et les différentes valeurs de la liste sont séparées par des virgules ; par exemple :

```
liste_de_nombres = [ 7, 12, 1.5, 9, 43]
liste_de_couleurs = [ "rouge", "vert", "bleu", "jaune", "orange"
```

Il est possible, en Python, de définir des listes comportant des valeurs de types différents (nombres, chaîne de caractères ou même des sous-listes...) :

```
liste_de_types_differents = [ 3, "Bonjour", 2.5, ["a", "b"], Tru
```

Il est également courant de définir une liste vide que l'on remplira par la suite (grâce à l'instruction *append* que l'on détaillera ultérieurement) :

```
liste_vide = []
```

Accès à un élément

Il est possible d'accéder à un élément d'une liste grâce à sa position appelée *indice*.

Attention : Le premier élément d'une liste correspond à l'indice 0 et si la liste contient *n* éléments, l'indice du dernier élément est *n-1* :

```
liste = [ "un", "deux", "trois", "quatre"]
# indices : 0       1        2         3
```

Pour accéder à un élément d'une liste on utilise la syntaxe suivante : *nom_de_la_liste[indice]*, par exemple :

```
liste = [ "rouge", "vert", "bleu", "jaune", "orange" ]
print(liste[0]) # affiche rouge
print(liste[4]) # affiche orange
```

```
liste[2] = "violet" # modifie le 3ème élément de la liste
print(liste) # affiche ['rouge', 'vert', 'violet', 'jaune', 'ora
```

Liste définie en compréhension

En mathématiques, il est possible de définir un ensemble en *compréhension*. Par exemple, l'ensemble :

$$E = \{n \in \mathbb{N} \mid 5 \leqslant n < 10\}$$

représente l'ensemble des entiers naturels supérieurs ou égaux à 5 et strictement inférieurs à 10, c'est à dire l'ensemble :

$$E = \{5\ ;\ 6\ ;\ 7\ ;\ 8\ ;\ 9\}$$

En Python, il est également possible de définir une liste en *compréhension* en utilisant une boucle *for* à l'intérieur de la définition de la liste :

```
liste = [ n for n in range(5, 10) ]
# équivaut à liste = [ 5, 6, 7, 8, 9 ]
```

Voici deux exemples plus avancés de listes définies en compréhension :

```
liste1 = [ n**2 for n in range(6) ]
# liste1 contient [ 0, 1, 4, 9, 16, 25 ]
liste2 = [ n for n in range(2, 7) if n!=4 ]
# liste2 contient [2, 3, 5, 6]
```

2. Opérations sur les listes

La méthode .append()

En programmation, une *méthode* est une fonction qui agit sur un certain objet.

La méthode *.append()* permet d'ajouter un élément à la fin d'une liste, par exemple :

```
mes_notes = [12, 14, 9, 16]
mes_notes.append(13)
print(mes_notes) # affiche [12, 14, 9, 16, 13]
```

Il est fréquent, lorsque l'on souhaite créer une liste pas à pas, de partir d'une liste vide et d'ajouter les éléments un par un. Par exemple, le programme suivant crée une liste en comptant de 3 en 3 en partant de 1 jusqu'à 16 :

```
ma_liste = []
n = 1
while n <= 16 :
   ma_liste.append(n)
   n = n+3
print(ma_liste) # affiche [1, 4, 7, 10, 13, 16]
```

Concaténation de listes

Comme pour les chaînes de caractères, l'opérateur « + » permet de concaténer des listes :

```
couleurs1 = [ "rouge", "jaune", "vert"]
couleurs2 = ["orange", "bleu"]
```

```
couleurs = couleurs1 + couleurs2
print(couleurs) # affiche ['rouge', 'jaune', 'vert', 'orange',
```

Longueur d'une liste

La fonction *len()* retourne la longueur de la liste passée en paramètre.

Cette fonction peut être utilisée lorsque l'on souhaite parcourir les éléments d'une liste pour déterminer la fin de la boucle. Par exemple, le programme ci-dessous affiche les éléments de la liste séparés par des points-virgules :

```
liste = ["a", "b", "c", "d"]
for i in range(len(liste)) :
   print (liste[i], end=";") # affiche a;b;c;d;
```

Parcourir une liste élément par élément

Lorsque l'on n'a pas besoin de l'indice, on peut parcourir directement les éléments d'une liste avec la syntaxe *for élément in liste* :

```
notes = [12, 14, 9, 16, 13]
somme = 0
for note in notes :
   somme = somme + note
print(somme) # affiche 64
```

Cette écriture est plus concise que *for i in range(len(liste))* et s'utilise chaque fois que la position de l'élément n'intervient pas dans le calcul.

Tester l'appartenance à une liste

L'opérateur *in* permet de tester si un élément appartient à une liste. Il retourne *True* si l'élément est dans la liste et *False* sinon :

```
couleurs = ["rouge", "vert", "bleu"]
print("vert" in couleurs)   # affiche True
print("jaune" in couleurs)  # affiche False
```

Les questions essentielles

1. Comment créer une liste à partir d'une formule ou d'une condition ?

On utilise la syntaxe de *liste en compréhension* : *[expression for variable in iterable if condition]*. Cette écriture condense en une ligne ce qui nécessiterait autrement une boucle *for* avec *append*.

Voir la fiche méthode : Créer une liste en compréhension

2. Comment construire une liste pas à pas avec une boucle ?

On initialise une liste vide *ma_liste = []* avant la boucle, puis on ajoute chaque élément avec *ma_liste.append(valeur)*. Cette approche convient aussi bien à *for* (nombre de termes connu) qu'à *while* (arrêt sur condition).

Voir la fiche méthode : Construire une liste pas à pas avec append()

3. Comment calculer une somme, une moyenne ou un extremum sur une liste ?

On applique le schéma d'*accumulation* : initialiser une variable adaptée (0 pour une somme, *liste[0]* pour un min/max), parcourir la

liste avec *for element in liste* et mettre à jour la variable à chaque tour.

Voir la fiche méthode : Parcourir une liste pour calculer une somme, une moyenne ou un extremum

4. Comment vérifier qu'une valeur figure dans une liste ?

On utilise l'opérateur *in* dans une condition : *if element in liste :* exécute une action quand la valeur est présente, *not in* teste l'absence (utile par exemple pour éviter les doublons avant un *append*).

Voir la fiche méthode : Tester l'appartenance d'un élément à une liste

Télécharger en PDF