Découvrir les listes Python

Informatique, 1re année de licence, Univ. Lille

novembre 2020

Coup d'œil

On découvre

- comment manipuler des collections de valeurs
- comment écrire des listes Python
- comment parcourir une liste Python ou accéder à un de ses éléments

Liste

Il est parfois nécessaire de manipuler une collection de valeurs.
Par exemple le nombre de jours de chacun des mois de l'année, ou les notes de l'ensemble des étudiants de 1re année de licence…

En Python, on utilisera par exemple les *listes*.

Ces listes Python sont des structures capables de contenir un nombre quelconque de valeurs.

Comme les chaînes de caractères ou les intervalles `range`, les listes sont des *itérables* : on va donc pouvoir réaliser une même instruction sur chacun des éléments d'une liste.

Comme les chaînes de caractères ou les intervalles `range`, les éléments des listes sont *indicés* : on va donc pouvoir accéder à un élément via son *indice* dans la liste.

Nous découvrirons ensuite d'autres caractéristiques des listes Python.

Liste — en Python

En Python, une liste de valeurs écrites entre crochets `[` et `]`, et séparées par des virgules `,` forme une liste.
On écrit par exemple

```
>>> [31, 28, 31, 30]
[31, 28, 31, 30]
>>> [12.0, 10.5, 9.5, 19., 14.5]
[12.0, 10.5, 9.5, 19.0, 14.5]
```

Le type de ces valeurs est `list`, indépendamment du type des éléments :

```
>>> type([31, 28, 31, 30])
<class 'list'>
```

```
>>> type([12.0, 10.5, 9.5, 19., 14.5])
<class 'list'>
```

Complément — Tableau et liste en informatique

En informatique, un *tableau* est une structure de données constituée d'une séquence d'éléments auxquels on peut accéder par leur indice.

En informatique, une *liste* est une structure de données qui permet de regrouper des *éléments*, et qui autorise :

- d'ajouter un élément à la liste,
- retirer un élément à la liste,
- savoir si la liste est vide.

Ces deux structures de données classiques se retrouvent dans beaucoup de langages de programmation. Les tableaux ou listes de certains langages peuvent avoir des caractéristiques propres.

Le cas de Python est particulier. La structure de données *liste Python* est plus proche de ce que l'on nomme habituellement *tableau*.

Nous nous en tiendrons pour le moment à l'étude des listes Python.

Construire des listes

Séquence d'éléments

Une liste peut donc être écrite comme une suite entre crochets d'éléments séparés par des virgules.

Tous les types sont acceptés :

```
>>> [31, 28, 31, 30]
[31, 28, 31, 30]
>>> [12.0, 10.5, 9.5, 19., 14.5]
[12.0, 10.5, 9.5, 19.0, 14.5]
>>> ["une", "liste", "Python"]
['une', 'liste', 'Python']
>>> [True, False, False, True]
[True, False, False, True]
```

De manière plus générale que ces premiers exemples, les éléments sont donnés sous forme d'expressions qui sont évaluées :

```
>>> l = [3 * 14, 42 // 7]
>>> l
[42, 6]
>>> [3 < 14, '3' < '14', 3.14 < pi, type(pi) == float]
[True, False, True, True]
```

Une liste peut ne contenir qu'un unique élément. On écrit logiquement

```
>>> singleton = [42]
>>> singleton
[42]
```

Une liste peut ne contenir aucun élément. C'est la *liste vide*.

On écrit alors :

```
>>> []
[]
```

Concaténation et répétition

Les opérateurs maintenant habituels de concaténation – `+` –, et de répétition – `*` – peuvent être utilisés sur les listes :

```
>>> [31, 30] * 2
[31, 30, 31, 30]
>>> seq5 = ([31, 30] * 2) + [31]
>>> annee = [31, 28] + seq5 * 2
>>> annee
[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

Transtypage d'itérables

Il est possible de convertir un itérable en liste en utilisant la fonction prédéfinie `list()`.
Cette fonction crée une liste dont les éléments sont ceux de l'itérable.

Nous pouvons l'utiliser sur les itérables que nous connaissons, intervalles `range` et chaînes de caractères :

```
>>> conversion_range = list(range(5, 14))
>>> conversion_range
[5, 6, 7, 8, 9, 10, 11, 12, 13]
>>> conversion_str = list("bonjour")
>>> conversion_str
['b', 'o', 'n', 'j', 'o', 'u', 'r']
```

Accéder aux éléments d'une liste

Les éléments d'une liste sont indicés. Il est donc possible d'accéder à un élément à partir de son indice.

Comme pour les chaînes de caractères, les indices débutent à 0.
De même, la fonction prédéfinie `len()` renvoie le nombre d'éléments d'une liste, et l'indice du dernier élément d'une liste `l` est `len(l)-1`.

On accède à un élément avec la même syntaxe que pour les chaînes.

On écrit donc par exemple :

```
>>> notes = [12.0, 10.5, 9.5, 19., 14.5]
>>> notes[1]
10.5
>>> len(notes)
5
>>> notes[4]
14.5
```

Parcourir des listes

Itérer sur les éléments d'une liste

Les listes sont des *itérables*. On peut donc les parcourir à l'aide boucle `for`.

À chaque tour de boucle, la variable d'itération sera associée à l'élément courant de la liste.

Il est par exemple possible de calculer la somme des éléments d'une liste comme ceci :

```
def somme(l):
    s = 0
    for x in l:
        s += x
    return s
```

Lors d'un appel `somme([6, 14, 22])`,

- lors du premier tour de boucle, `x` prend la valeur `6`,
  et le calcul de `s` produit `6`
- lors du deuxième tour de boucle, `x` prend la valeur `14`,
  et le calcul de `s` produit `20`
- lors du troisième et dernier tour de boucle, `x` prend la valeur `22`,
  et le calcul de `s` produit `42`

```
>>> des_nombres = [6, 14, 22]
>>> somme(des_nombres)
42
```

La version suivante travaille sur des chaînes de caractères :

```
def concatenation(l):
    s = ""
    for x in l:
        s += x + ' '
    return s
```

et par exemple :

```
>>> des_mots = ["une", "liste", "Python"]
>>> concatenation(des_mots)
'une liste Python '
```

Itérer sur les indices des éléments d'une liste

Les éléments des listes étant accessibles par leur indice, il est également possible de parcourir les éléments d'une liste via ces indices.
Pour parcourir l'ensemble des éléments, on fera varier les indices de `0` – inclus –, à la longue de la liste – non incluse –.

Une autre fonction pour le calcul de la somme des éléments d'une liste peut être :

```
def somme_via_indice(l):
    s = 0
    for i in range(0, len(l)):
        s += l[i]
    return s
```

On peut alors ne parcourir que certains éléments, par exemple ceux de rang pair

```
def somme_rang_pair(l):
    s = 0
    for i in range(0, len(l), 2):
        s += l[i]
    return s
```

et écrire

```
>>> des_nombres = [6, 14, 22, 1, 3, 5]
>>> 6 + 22 + 3
31
>>> somme_rang_pair(des_nombres)
31
```

ou parcourir les éléments dans l'ordre inverse :

```
def concatenation_inverse(l):
    s = ""
    for i in range(len(l)-1, -1, -1):
        s += l[i] + ' '
    return s
```

et écrire

```
>>> des_mots = ["une", "liste", "Python"]
>>> concatenation_inverse(des_mots)
'Python liste une '
```

Il est bien entendu possible de parcourir les éléments via leur indice à l'aide d'une boucle `while`. Cela fera l'objet d'exercices.

Memento

- les listes sont des séquences d'éléments de types quelconques
- l'écriture littérale d'une liste est une séquence d'éléments séparés par une virgule `,` encadrée de crochets `[` et `]`
- on concatène des listes avec l'opérateur `+`
- on peut répéter une liste avec l'opérateur `*`.
- les listes sont des itérables, on les parcourt avec une boucle `for`
- on accède à l'élément d'indice `i` d'une liste `l` avec la syntaxe `l[i]`.