

Cinquième partie

Listes et fonctions d'ordre supérieur

1 Introduction

De nombreux calculs requièrent l'utilisation de collections d'objets.

On les groupe alors en *listes*, c'est-à-dire, en collections ordonnées d'objets de même type ou non.

Les listes constituent ce qu'on appelle une *structure de données*, au même titre que les *structures* ou les *tables* qui seront vues l'an prochain pour ceux qui choisiront l'option !

Le but de ce chapitre est de présenter la structure de données `liste` et les fonctions qui permettent d'effectuer des opérations sur les listes.

2 Qu'est-ce qu'une liste ?

C'est une collection **ordonnée** d'objets de type identique : ce type peut être quelconque. S'il s'agit d'entiers, on aura le type `liste entier`.

Le type `liste` est donc un constructeur de type : définition d'un nouvel ensemble d'objets, noté `liste type`

Une liste particulière : la liste vide, qui ne contient aucun élément.

On notera `liste+` le type formé des listes sauf la liste vide. Par exemple, `liste+ réel` est le type formé des listes de réels moins la liste vide.

2.1 Notation de liste en Python

Une liste composée des objets `a`, `b`, ..., `n` est notée `[a,b, ..., n]`.

Ainsi, `[2,3,-1]` désigne une liste d'entiers.

La liste vide est notée `[]`.

Une liste est une expression de Python, qui s'utilise comme toute autre expression.

On peut affecter une liste à un symbole.

```
uneListe = [1,2,5]
print (uneListe)
```

2.2 Longueur d'une liste

C'est son nombre d'éléments. Son type est :

$$\text{longueur} : \text{liste type} \rightarrow \text{naturel}$$

La longueur de la liste vide est 0.

En Python, la longueur d'une liste est donnée par la fonction `len`.

```
len([1,2,3])
3
len(["une", "liste"])
2
len([ ])
0
len [[1,2,3],[4],[5,6],[7,8,9,10]]
4
```

Attention : `[4]` n'est pas égale à 4.

2.3 Le type liste en Python

En Python, une liste est une **séquence** tout comme les chaînes de caractères.

Comme pour la longueur, certaines opérations sont communes aux séquences.

En Python, le type est **list**

Faire un exemple avec `type(uneListe)`.

En Python, comme dans d'autres langages, il est possible d'avoir des listes contenant des éléments de type différents (**liste indifférent**). Leur utilisation étant plus délicate pour la mise au point des programmes, nous ne le considérerons pas dans ce cours.

3 Fonctions et opérations sur les listes

Fonctions élémentaires permettant de :

- construire des listes,
- extraire des éléments d'une liste,
- ajouter, ôter ou remplacer les éléments d'une liste,
- combiner des listes.

Nous n'introduirons pas en L1 les méthodes offertes par la classe **list** : la présentation des objets sera au cœur du cours de L2.

Voir A.M. l'explication de ce que font ces fonctions

3.1 Construction de listes

3.1.1 La fonction range

la détailler pour `imin,imax,di`

`range` : entier \times entier \times entier \rightarrow séquence entier

`range(imin,imax,di)` construit une séquence de nombres entiers de `imin` à `imax` (valeur exclue), séparés par `di` (1 par défaut).

Pour construire une liste à l'aide `range`, il faut transformer le résultat à l'aide de la fonction `list`.

Par exemple, `list(range(1,10))`

`range` est très utilisée pour les itérations sur les séquences (voir section 4)

Exemple 3.1. `list(range(-4,7,3))`

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

`list(range(10,0,-2))`

`[10,8,6,4,2]`

`list(range(1,4))`

`[1,2,3]`

`list(range(10))` vaut `list(range(0,10,1))`

`[0,1,2,3,4,5,6,7,8,9]`

Mettre au tableau les trois versions de `range` et mettre un exemple avec `range(len(ch))`.

3.2 Extraire des éléments d'une liste

Utilisation de l'indexation. Il existe aussi une méthode `pop` (<http://docs.python.org/2/tutorial/datastructures.html#more-on-lists>)

3.2.1 Prendre un élément quelconque

Les éléments de la liste étant ordonnés, on peut y accéder en donnant l'index de l'élément cherché (index compris entre **0** et **len(l)-1** : identique au fonctionnement sur les chaînes de caractères.)

`l = [8,5,1,2,12]` (à mettre au tableau et à conserver : `exemple.py`)

`l[0]` rend 8.

Lorsque l'index i est négatif, on extrait le i -ème élément à partir de la fin

```
l[-2] rend 2
```

Si $i \geq \text{len}(liste)$ ou $i < -\text{len}(liste)$, une erreur est rendue (`list index out of range`)

3.2.2 Extraire une sous-liste d'éléments consécutifs

Utilisation du « **tranchage** » : indication des index de début et de fin de tranche entre crochets.

Attention : l'index de fin n'est pas compris dans la tranche!

Faire un schéma au tableau avec les index entre les éléments pour expliquer.

```
l[1:3] rend [5, 1]
l[1:2] rend [5] (!= de l[1] !)
```

Si l'index de début est omis, il est considéré comme 0, si l'index de fin est omis, il est considéré comme `len(l)`.

N.B. : cela fonctionne aussi sur les chaînes de caractères.

```
l[:3] rend [8,5, 1]
l[3:] rend [2,12]
```

3.3 Ôter, remplacer et ajouter des éléments dans une liste

3.3.1 Ôter des éléments

L'instruction `del` permet de supprimer un élément ou plusieurs éléments dans une liste.

faire une copie de `l` de deux façons:

```
l1=l
lc=l[:]
del l[0]
del l[2:]
```

et montrer la différence de comportement entre une copie et un alias (schéma au tableau).

Cela peut aussi se faire en affectant une liste vide dans une « tranche » de la liste

```
l[0:1] = []
l[2:] = []
```

3.3.2 Remplacer un élément

Il suffit de faire une affectation avec le bon index!

Ce remplacement peut aussi être réalisé avec un « tranchage » : permet de modifier plusieurs éléments d'un coup. Attention : il faut que l'expression en partie droite de l'affectation soit une **liste** (pas obligatoirement de la même taille que la tranche).

```
l[2]= 100
l[3: ]= [8, -1]
l[:]=[3] !!!!
```

3.3.3 Ajouter un élément à une liste

Il est possible d'insérer un ou plusieurs éléments dans une liste en donnant une tranche « vide » en partie gauche d'une affectation.

Rappel : la partie droite doit être une liste

```
l[2:2]=[2000] utiliser le schéma avec les indices pour expliquer
```

```
l[-1:-1]=[-1,-2,-3]
```

```

l[len(l):]=[34] -> ajout en fin
l[2:2]=[100]
l[0:0]=[-12] -> ajout en tête
l[3:3]=[12,0]

```

3.4 Concaténation de listes

L'opérateur « + » appliqué à deux listes les concatène en une seule et nouvelle liste.

L'opérateur « * » appliqué à une liste et un entier, concatène **n** fois la liste donnée.

```

[1,2] + [8,5]
[1,2] * 3

```

3.5 Autres instructions offertes

Instructions

x in l teste si **x** est présent dans **l**

x not in l teste si **x** n'est pas présent dans **l**

4 Itération sur les séquences

Il est souvent nécessaire de pouvoir parcourir le contenu d'une liste (ou d'une séquence).

Possibilité de le faire avec une itération « while ».

Python propose une instruction pour le parcours d'une séquence : **for in**

```

for x in l :
    print (x, " ", end="")

```

cette instruction composée va afficher tous les éléments de la liste **l**

Construction d'une liste en compréhension

[**expr** for **x** in **seq**] → construit une liste des valeurs de **exp** pour toutes les valeurs de **x** dans la séquence **seq**

Par exemple : [2**x for x in range(0,11)] → liste de puissance de 2 de 2⁰ à 2¹⁰

[**expr** for **x** in **seq** if **cond**] → construit une liste des valeurs de **exp** pour toutes les valeurs de **x** dans la séquence **seq** si la condition **cond** est vérifiée

Par exemple : [x**2 for x in range(0,11) if pairQ(2**x)] → liste des carrés pairs 0² à 10²

Exemple 4.1. *Faire des constructions avec des chaînes, liste et range.*

```

ch = "azerty"
[c for c in ch] (équivalent à list(ch))
[10**x for x in range (5,-1,-1)]
[x**2 for x in l if pairQ(x)]
[[x**i for x in range(1,11)] for i in range(2,5)]

```

5 Opérations classiques sur les listes et fonctions d'ordre supérieur

Certaines opérations sur les listes sont très fréquentes :

- application d'une opération à chaque élément d'une liste;
- réduction d'une liste (par addition par exemple) ;
- sélectionner les éléments d'une liste.

Nous allons voir plusieurs mises en œuvre, dont une utilisant des fonctions d'ordre supérieur : c'est une fonction qui accepte comme paramètre une fonction.

5.1 Application à tous les éléments d'une liste : la fonction map

On souhaite, à partir d'une liste `[e1,e2, ..., en]`, obtenir la liste `[f(e1),f(e2) ..., f(n)]` (`f` étant une fonction unaire).

- utilisation d'une itération « for »
- construction d'une liste en compréhension
- fonction d'ordre supérieur « map »

`map(f, liste)` rend la séquence obtenue en appliquant la fonction `f` à chacun des éléments de `liste`.

Il faut transformer la séquence pour obtenir une liste (`list(...)`).

`list(map(f,[e1,e2, ..., en]))` rend `[f(e1),f(e2) ..., f(n)]`

Cette fonction peut être utilisée pour combiner plusieurs listes de même longueur :

`list(map(mul,[2,4,5],[6,4,3]))` rend `[mul(2,6),mul(4,4),mul(5,3)]`

Exemple 5.1. *Faire l'équivalence avec une boucle for*

```
lRes=[] # liste résultat
for e in l : # l -> liste d'origine
    lRes[len(lRes):] = [math.sin(e)]

list(map( math.sin, [math.pi/4,math.pi/2,3*math.pi/2,math.pi]))
[1/Sqrt[2], 1, -1, 0]
(+ exemple sur portable)
```

5.2 Réduction d'une liste : la fonction reduce

On souhaite, à partir d'une liste `[e1,e2,e3, ..., en]`, obtenir la liste `[f(...f(f(e1,e2),e3),...,n)]` (`f` étant une fonction binaire).

Par exemple, cela permet de faire la somme des entiers d'une liste.

- utilisation d'une itération « for »
- utilisation de la fonction d'ordre supérieur `reduce` (elle fait partie de la bibliothèque `functools` `functools.reduce(f,[a,b,c,d])` a pour résultat l'évaluation de l'expression `f(f(f(a,b),c),d)`.

Exemple 5.2. `som=0` # résultat

```
for e in l : # l -> liste d'origine
    som = som + e

import functools
functools.reduce(add,[1,2,3,4])
10
(+ exemple sur le portable avec mul)
functools.reduce(add,(map(math.sqrt,range(1,11))))
```

5.3 Sélection conditionnelle : la fonction filter

On souhaite sélectionner dans une liste les éléments vérifiant une condition.

- utilisation d'une itération « for »
- construction d'une liste en compréhension
- fonction d'ordre supérieur « filter »

`filter(pred,liste)` permet de sélectionner dans une liste tous les éléments qui satisfont un prédicat unaire.

Elle rend une séquence, qu'il faut transformer pour obtenir une liste.

Par exemple, `list(filter(pairQ,[8,5,1,2,12]))`

Exemple 5.3. `lRes=[]` # liste résultat

```
for e in l : # l -> liste d'origine
    if pairQ(e) : lRes[len(lRes):] = [e] #ajout en fin
[e for e in l if pairQ(e)]
```

```
list(filter(pairQ,[8,5,1,2,12]))  
[8,2,12]  
(+ exemple sur le portable)
```