Les collections

Le terme « collections » est un terme générique englobant tout objet contenant plusieurs éléments ou objets de type simple. Elles sont aussi appelés « itérables ». Python propose de nombreux types de collection différents, chacun adapté à un usage.

1 Listes

Définition

Une **liste** est une collection de valeurs qui sont modifiables. Elle est notée entre crochets. Deux valeurs sont séparées par une virgules.

Par exemple : `[2,2.,1+i,"deux"]` est une liste de 4 éléments en Python. Comme cet exemple le montre, les listes ne sont pas forcément de types homogènes.

Il existe aussi la liste vide : `[]`

1.1 Manipulation de listes

1.1.1 Création explicite

La façon la plus simple de définir une liste est d'utiliser une affectation explicitement. Par exemple :

```
L1=[2,4,8,16,32,64,128]
```

1.1.2 Opérateurs sur les listes

Quatre opérateurs sont à retenir pour la manipulation de liste : deux opérateurs de concaténation et deux opérateurs de test d'appartenance.

La concatenation `+` consiste en l'assemblage d'une liste par morceaux. Dans la même logique, l'operation `*` assemble plusieurs fois le même morceau.

Deux tests d'appartenance peuvent être utilisés :

- `x in liste` : renvoie `True` si l'élément `x` est présent dans `liste`, `False` sinon
- `x not in liste` : renvoie `True` si l'élément `x` n'est pas présent dans `liste`, `False` sinon

```
>>> print([1,2]+[3,4])
[1,2,3,4]
>>> print(3*[1,2])
[1,2,1,2,1,2]
```

```
>>> 2 in [1,2]
True
>>> 2 not in [1,2]
False
```

1.1.3 Comparaison des listes & arrays

	`list`	`array`
type	natif	importé du module `numpy`
contenu	hétérogène	homogène
opérations	« `+` » concaténation « `*` » répétition	« `+` » addition terme à terme « `*` » multiplication terme à terme

1.1.4 Fonctions

1. `len(liste)` : renvoie la longueur, c'est à dire le nombre d'éléments, d'une liste
2. `min(liste)` : renvoie le plus petit élément d'une liste (dans la mesure où ils sont comparables)
3. `max(liste)` : renvoie le plus grand élément d'une liste (dans la mesure où ils sont comparables)
4. `sum(liste)` : renvoie la somme de tous les éléments d'une liste (dans la mesure où ils sont additionnables)
5. `sorted(liste)` : renvoie une nouvelle liste des éléments de `liste` triés par ordre croissant
6. `del(liste[i])` : supprime le i^{eme} élément de la `liste`
7. `del(liste)` : supprime `liste` de la mémoire

1.2 Accès aux éléments d'une liste

1.2.1 Principe

Le traitement de données nécessite souvent le travail sur certains éléments d'un liste uniquement. Il est donc important de maîtriser comment accéder à un élément ou groupe d'éléments d'une liste. Les éléments d'une liste de longueur n sont numérotés de 0 **à** $n-1$. On appelle **indice** ce numéro (*index* en anglais). Il est aussi possible d'utiliser un **indice négatif** qui correspond à une numérotation des éléments en partant de la fin ($-n$ pour le premier à -1 pour le dernier).

Soit `L` une liste quelconque.

1. `L[p]` renvoie l'élément d'indice p de `L`
2. `L[p:n]` renvoie une nouvelle liste constituée des éléments de `L` d'indice p inclus à n **exclu**
3. `L[p:n:pas]` renvoie une nouvelle liste constituée des éléments de `L` d'indice p inclus à n exclu, tous les pas
4. `L[:]` renvoie une nouvelle liste constituée de tous les éléments de `L`
5. `L[p:]` renvoie une nouvelle liste constituée de tous les éléments de `L` à partir de l'élément d'indice p inclus jusqu'à la fin
6. `L[:n]` renvoie une nouvelle liste constituée de tous les éléments de `L` depuis le premier jusqu'à l'élément d'indice n exclu
7. `L[::pas]` renvoie une nouvelle liste constituée des éléments de `L`, tous les pas

1.2.2 Exemples

Soit la liste :

```
L=["A","B","C","D","E","F","G","H","I","J","K"]
```

1.2.3 Affectation d'un élément de liste

L'affectation d'une liste est possible. L'affectation d'un élément ou groupe d'éléments de liste est possible aussi. La syntaxe est la suivante :

```
L1[3]=14
```

Attention, cette instruction pourrait être remplacée par l'instruction complète :

```
L1=L1[:3]+[14]+L1[4:]
```

Evidemment, si `L1` n'a pas au moins 4 éléments, cette instruction renvoie une erreur.

1.3 Parcours d'une liste

A partir des syntaxes d'accès à un élément de liste, il est possible de parcourir un après l'autre les élément d'une liste. Deux approches sont possibles :

```python
L=[2,4,8,16,32,64,128]
for i in range(len(L)):
    print(L[i])
```

```python
L=[2,4,8,16,32,64,128]
for x in L:
    print(x)
```

1.3.1 Parcours de liste et definition par compréhension

Sur cette base, il est possible de construire une liste « par compréhension » . C'est une méthode qui consiste à définir une liste à l'aide d'une expression dépendante d'une variable qui parcourt un itérable.

La syntaxe générale est à adapter suivant le contexte :

```
[expression(i) for i in range(n)]
```

ou

```
[expression(x) for x in L]
```

Exemple : quelle est la liste cosntruite par : `L3=[i%3 for i in range(9)]`

Application 1

Qu'est ce qui est stocké dans les listes L007 et L008 suivantes :

```
L007=[i**2 for i in range(6)]
L008=[2*i+1 for i in L007]
```

Ecrire le programme permettant de créer ces listes en utilisant deux boucles `for` explicites.

1.4 Algorithmes sur les listes

Application 2 : Recherche d'un élément dans une liste

Une première problématique consiste à savoir si un élément `a` est présent dans une liste `L` déjà en mémoire. Il n'est pas nécessaire de parcourir tous les éléments de la liste, car on souhaite interrompre le parcours dès qu'un élément est trouvé. Deux solutions sont possibles, en utilisant une boucle `for` ou une boucle `while`.

```python
# L et a sont déjà en mémoire
i=0
while i<len(L)and L[i]!=a :
    i=i+1
print(i<len(L)) # Ceci est un booléen
```

Application 3 : Recherche d'un maximum

Deuxième problématique : trouver le maximum d'une liste de nombres déjà en mémoire.

```python
# liste est déjà en mémoire
maxi=liste[0]
for i in range(len(liste)):
    if liste[i]>maxi:
        maxi=liste[i]
print(maxi)
```

2 Autres collections

2.1 les « $n-$uplets » ou « tuplets »

Définition

Les « $n-$uplets » ou « tuplets » sont des similaires à des listes, mais non modifiables. Ils sont définis entre parenthèses ; le séparateur est la virgule : `(1.,1,'1',True)`.

2.2 Chaîne de caractères

Une chaîne de caractères est aussi similaire à une liste de caractères, et n'est pas modifiable.

Elles sont définies entre apostrophe `'blablabla'` ou entre guillements `"blablabla"` sans séparateur entre deux caractères.

Enfin, l'accès par indexation aux éléments d'une chaîne de caractères respectent la même syntaxe que celle pour les listes.

Petite différence avec les listes.

```
>>> L=[2]

>>> L[0]==L
False
```

```
>>> S='b'

>>> S[0]==S
True
```

L'affectation d'un élément de chaîne n'est pas autorisée.

2.2.1 Caractères spéciaux

Parmi les caractères spéciaux de la table `ASCII`, nous n'en retiendrons que 2 :

`\n`	retour à la ligne	*line feed*
`\t`	tabulation	*tabulation*
`\\`	anti-slash	\

On remarque que les caractères spéciaux sont introduits par `\`; `\\` devient lui-même un caractère spécial.

Application 4 : compter les occurences d'une lettre

Proposer un programme qui compte le nombre de fois où la lettre `'s'` apparait dans une chaine de caractère `S` déjà en mémoire.

2.3 Dictionnaires

Un dictionnaire est une collection aux propriétés atypiques dont nous ferons un usage dans certains cas très particuliers. A la différence d'une liste où les éléments sont accessibles via leur index, les éléments d'un dictionnaire sont accessibles via leur « clé ».

0	2001
1	'odyssée'
2	'espace'
3	'Thomas'
4	'Pesquet'

Jean	0612345678
Josephine	0623456789
Jamel	0634567890
Jacques	0645678901
Jimmy	0656789012

Les clés doivent être constantes (types élémentaires, `str`, `tuple`, mais pas une liste). Les valeurs peuvent être tout type d'objet.
Création d'un dictionnaire

```
d = {}
d = {'a' : 12, 'droit' : 'au but', 42 : [1,2,3]}
```

lire une valeur sur la base de sa clé

```
d['a']
```

Ajout ou mise à jour d'un élément :

```
d[True]=888
```

Supprimer un élément

```
del(d[42])
```

Application 5

Deux listes sont en mémoires :

- une liste de noms, notée `Nom` ;
- une liste de numéros de téléphone, notée `Tel`.

Ecrire un programme qui crée un dictionnaire dont les clés sont les différents noms, et dont les valeurs sont les numéros de téléphone.

3 A retenir

Parcours d'une liste `L=[a0,...,an]` :	`for i in range(len(L)):` `  print(L[i])`	`for x in L:` `  print(x)`

Construction d'une liste `[a0,...,an]`
où `ak`$=f(k)$
	`L=[]`
	`for k in range(0,n+1):`
	`  L.append(f(k))`
		`L=[f(k) for k in range(0,n+1)]`