1 Les listes

Une *liste* en Python est une suite finie d'élements quelconques. Le type des listes est appelé `list`.

L'ensemble des listes est constitué des expressions de la forme

$$[e_1, \ldots, e_n]$$

où les e_i sont des données absolument quelconques (elles peuvent même être aussi elles-mêmes des listes).

```
>>> [2+3,"toto",2.5]
[5, 'toto', 2.5]
>>> type([5,"toto",2.5])
<type 'list'>
>>> [5,[9,5.5],"toto"]
[5, [9, 5.5], 'toto']
>>> type([5,[9,5.5],"toto"])
<type 'list'>
```

Il se peut qu'il n'y ait aucun élément dans la liste ; il s'agit alors de la *liste vide*, notée `[]`.

Beaucoup d'opérations travaillent sur les listes, à commencer par les opérations d'accès direct que nous venons de voir sur les chaînes de caractères. Ces dernières se transcrivent sur les listes de manière naturelle :

```
>>> ["toto",5,"tutu",5.5][1]
5
>>> ["toto",5,"tutu",5.5][1:3]
[5, 'tutu']
```

Le test d'appartenance ne fonctionne que pour un unique élément dans une liste, et non pas pour une sous-liste contrairement aux chaînes de caractères.

```
>>> "glop" in ["toto",5,"tutu",5.5]
False
>>> "tutu" in ["toto",5,"tutu",5.5]
True
>>> "tu" in ["toto",5,"tutu",5.5]
False
>>> [5,"tutu"] in ["toto",5,"tutu",5.5]
False
>>> [5,"tutu"] in ["toto",[5,"tutu"],5.5]
True
```

Comme pour les chaînes de caractères, on trouve également les opérations : `len` pour la longueur (nombre d'éléments dans la liste) et `+` pour la concaténation de deux listes.

```
>>> def homogene (l) :
...     if len(l) == 0 :
...         return True
...     else :
...         t = type(l[0])
...         i = 1
...         vu = True
...         while vu and i < len(l) :
...             vu = vu and t == type(l[i])
...             i = i + 1
```

```
...         return vu
...
>>> homogene (["toto",5,"tutu",5.5])
False
>>> homogene (["toto","glop","titi"])
True
>>> homogene ([])
True
```

Lorsqu'une liste `l` est homogène, les opérations `min(l)` et `max(l)` fournissent respectivement le plus petit et le plus grand élément de la liste.

On peut également énumérer les éléments d'une liste homogène avec `for` :

```
>>> def somme (l) :
...     s = 0
...     for n in l :
...         s = s + n
...     return s
...
>>> somme ([])
0
>>> somme ([2,2,5,3])
12
>>> min ([2,2,5,3])
2
>>> max ([2,2,5,3])
5
```

2 Instructions de modification de liste

Contrairement aux chaînes de caractères, une variable de type liste peut être partiellement modifiée sans réaffecter toute la variable. Par exemple :

```
>>> coursesAfaire=["beurre","pain","artichaut","vin rouge"]
>>> coursesAfaire
['beurre', 'pain', 'artichaut', 'vin rouge']
>>> coursesAfaire[2]="avocat"
>>> coursesAfaire
['beurre', 'pain', 'avocat', 'vin rouge']
```

Ainsi, si v est une variable de type liste, alors l'instruction « v`[`i`]=`*expr* » remplace l'élément d'indice i dans la liste par la valeur du calcul de l'expression *expr*.

En revanche, avec des chaînes de caractères c'est impossible :

```
>>> nom="Samon"
>>> nom[1]
'a'
>>> nom[1]='i'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

On peut aussi remplacer une portion entière de liste par une autre liste :

```
>>> coursesAfaire[1:3]=["baguette","chou","vinaigre"]
>>> coursesAfaire
['beurre', 'baguette', 'chou', 'vinaigre', 'vin rouge']
```

Remarquez que dans l'instruction « v`[`i`:`j`]=`*expr* » on remplace les indices de i à $(j-1)$...

3 Particularités des modifications de liste en Python

Par défaut, l'expression doit être de type liste mais Python « fait ce qu'il peut » pour transformer l'expression en liste. Ici : une chaîne en liste de caractères...

```
>>> coursesAfaire = ['beurre', 'baguette', 'chou', 'vinaigre', 'vin rouge']
>>> coursesAfaire[1:3]="toto"
>>> coursesAfaire
['beurre', 't', 'o', 't', 'o', 'vinaigre', 'vin rouge']
>>> coursesAfaire[1:3]=8
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only assign an iterable
```

Plus généralement, une structure de données qui peut être transformée en liste de manière naturelle est dite « itérable » en Python.

Il est également possible de supprimer un élément ou une tranche d'éléments d'une variable de type liste :

```
>>> coursesAfaire
['beurre', 't', 'o', 't', 'o', 'vinaigre', 'vin rouge']
>>> del coursesAfaire[2]
>>> coursesAfaire
['beurre', 't', 't', 'o', 'vinaigre', 'vin rouge']
>>> del coursesAfaire[1:4]
>>> coursesAfaire
['beurre', 'vinaigre', 'vin rouge']
```

Évitez à tout prix de prendre des indices hors des bornes, le comportement n'est pas garanti ; cela ne produit pas forcément une erreur !

```
>>> del coursesAfaire[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
>>> del coursesAfaire[1:6]
>>> coursesAfaire
['beurre']
```

En revanche, il est possible de donner des indices négatifs : cela signifie « en partant de la fin ». Ainsi « *v*`[-`*i*`]=`*expr* » est équivalent à « *v*`[len(`*v*`)-`*i*`]=`*expr* » :

```
>>> coursesAfaire=["beurre","pain","artichaut","vin rouge"]
>>> coursesAfaire[-1]
'vin rouge'
>>> coursesAfaire[1:-1]
['pain', 'artichaut']
>>> del coursesAfaire[1:-1]
>>> coursesAfaire
['beurre', 'vin rouge']
```

Enfin pour les tranches d'éléments (avec le « : »), un indice manquant va jusqu'au bout de la liste :

```
>>> coursesAfaire=["beurre","pain","artichaut","vin rouge"]
>>> coursesAfaire[1:]
['pain', 'artichaut', 'vin rouge']
>>> coursesAfaire[2:]
['artichaut', 'vin rouge']
>>> coursesAfaire[:2]
```

```
['beurre', 'pain']
>>> coursesAfaire[:]
['beurre', 'pain', 'artichaut', 'vin rouge']
```

4 Exemple de programme sur les listes

Ecrire une fonction `combien` qui prend en entrée deux arguments : le premier est une valeur `e` de type quelconque et le second est une liste `l`. Cette fonction doit retourner en résultat le nombre de fois où l'élément `e` apparaît dans la liste `l` (en particulier 0 fois si l'élément n'apparaît jamais dans la liste).

```
>>> def combien (e,l) :
...     compteur = 0
...     for x in l :
...         if x == e :
...             compteur = compteur + 1
...     return(compteur)
```