Travaux pratiques d'introduction 8

Listes : adjonctions

Informatique tronc commun MPSI et PCSI

I Présentation

I.1 Les méthodes

Nous avons défini le type **list** de python et les fonctions de base. Jusqu'à présent nous utilisons les listes avec une longueur fixée : on pouvait changer les valeurs de chacune des positions mais on ne pouvait pas ajouter un élément supplémentaire.

On peut créer une nouvelle liste avec un élément de plus :

```
>>> l = [4, 3, 7, 6, 0]
>>> ll = l + [5]
>>> ll
[4, 3, 7, 6, 0, 5]
>>> l
[4, 3, 7, 6, 0]
```

Cette opération ne modifie pas la liste initiale mais crée une nouvelle liste : cela nécessite de copier tous les éléments et a donc un coût que l'on souhaite éviter. Python a prévu cette possibilité sous la forme d'une méthode.

Définition 1 : méthodes

Une méthode est une fonction d'un type spécial qui est invoquée en ajoutant son nom **après** l'objet auquel elle s'applique avec un point de séparation. Une méthode peut avoir des paramètres, ils seront placés classiquement entre les parenthèses qui suivent le nom de la méthode.

I.2 La méthode append

I.2.a Présentation

Définition 2: append

L'instruction `liste.append(x)` attache l'élément `x` au bout de la liste `liste`. C'est une instruction élémentaire qui procède sans affectation, elle ne fait que modifier sans créer.

```
>>> a = [1, 2, 3]
>>> a.append(5)
>>> a
[1, 2, 3, 5]
```

est équivalent à :

```
>>> a = [1, 2, 3]
>>> a = a + [5]
>>> a
[1, 2, 3, 5]
```

À retenir

Préférez la version avec `.append()` qui est plus compacte, plus facile à lire et est plus rapide à exécuter.

Il est toujours possible de mélanger dans une même liste des variables de type différent. On peut d'ailleurs mettre une liste dans une liste.

```
>>> liste = []
>>> liste
[]
>>> liste.append(1)
>>> liste
[1]
>>> liste.append("ok")
>>> liste
[1, 'ok']
```

À retenir : résultat de append

La fonction `append` modifie la liste mais ne renvoie rien, c'est-à-dire qu'elle ne renvoie pas d'objet récupérable dans une variable. Il s'agit d'un exemple d'utilisation de méthode (donc de fonction particulière) qui fait une action mais qui ne renvoie rien.

I.2.b Application d'une fonction aux éléments d'une liste

Nous avons vu qu'une liste de taille n pourra être construite selon le motif suivant :

```
def carre(liste):
    n = len(liste)
    l2 = [0]*n
    for i in range(n):
        l2[i] = liste[i]**2
    return l2
```

On peut construire une liste sans connaître à l'avance la taille de la liste. Une première conséquence est qu'il est possible de construire une liste de taille n en partant d'une liste vide et y adjoindre n éléments.

```
def carre(liste):
    l2 = []
    for x in liste:
        l2.append(x**2)
    return l2
```

Autre méthode :

```
def carre(liste):
    return [x**2 for x in liste]
```

On a maintenant deux méthodes possibles pour construire une liste de taille donnée, la construction avec `append()` étant légèrement plus lente.

I.3 La fonction pop

L'opération inverse de `append` est notée `pop`; l'instruction `liste.pop()` a deux effets.

Définition 3 : pop

L'instruction `liste.pop(i)` enlève de la liste l'élément situé à la position indiquée et le renvoie en valeur de retour. Si aucune position n'est spécifiée, `liste.pop()` enlève et renvoie le dernier élément de la liste.

```
>>> a = [1, 2, 3]
>>> a.pop()
3
>>> a
[1, 2]
```