Listes-Tuples-Dictionnaires
Compléments

Sommaire

1. Listes
   - Les bases
   - Le «découpage» (slicing) de liste
   - Les listes en compréhension
   - Les listes de listes
2. Les tuples
3. Les dictionnaires

Les bases

```
>>> l = []  # liste vide
>>> l1 = [1, 2, 3, 4, 5, 6]  # liste contenant 6 entiers
>>> l1[0]  # accès au premier élément
1
>>> len(l1)  # longueur de l1
6
>>> l1[len(l1)-1]  # dernier élément
6
>>> l1[-1]  # autre solution pour accéder au dernier élément
6
>>> l1[0] = 7  # une liste est mutable
>>> l1
[7, 2, 3, 4, 5, 6]
>>> 3 in l1  # appartenance à une liste
True
```

Les bases (suite)

```
>>> l1 = [1, 2, 3]
>>> l1 + [4, 5, 6]  # concaténation de listes
[1, 2, 3, 4, 5, 6]
>>> l1 * 2  # réplication (ici duplication) d'une liste
[1, 2, 3, 1, 2, 3]
>>> l1.append(4)  # ajout d'un élément à la fin
>>> l1
[1, 2, 3, 4]
>>> x = l1.pop()  # suppression du dernier élt. et récup. de l'élt. supprimé
>>> x
1
>>> l1
[1, 2, 3]
```

Les bases (suite)

```
>>> l1 = [1, 2, 3, 4, 5, 6]
>>> l2 = l1
>>> l2
[1, 2, 3, 4, 5, 6]
>>> l1[0] = 7
>>> l1
[7, 2, 3, 4, 5, 6]
>>> l2  # ATTENTION: l2 est un pointeur sur le contenu de l1
[7, 2, 3, 4, 5, 6]
>>> l3 = list(l1)  # solution: faire une copie profonde de l1
>>> l3
[7, 2, 3, 4, 5, 6]
>>> l1[0] = 8
>>> l1
[8, 2, 3, 4, 5, 6]
>>> l3
[7, 2, 3, 4, 5, 6]
```

Les bases (suite)

```
>>> l1 = [1, 2, 3]
>>> for i in range(len(l1)):  # type list: itérable
...   print(l1[i])
...
1
2
3
>>> for elt in l1:  # autre solution
...   print(elt)
...
1
2
3
>>> l1 = [1, 2, 3, 4, 5, 6]
>>> for i in range(2, 5):  # de l[2] à l[5-1]
...   print(l1[i])
3
4
5
```

Les bases (suite)

```
>>> l1 = [1, 2, 3, 4, 5, 6]
>>> for i in range(0, len(l1), 2):  # de l[0] à l[5] par pas de 2
...   print(l1[i])
...
1
3
5
>>> for i in range(len(l1)-1, 1, -2):  # de l[5] à l[1+1] par pas de -2
...   print(l1[i])
...
6
4
>>> for i in range(len(l1)-1, 0, -2):  # de l[5] à l[0+1] par pas de -2
...   print(l1[i])
...
6
4
2
```

Le découpage de liste

```
>>> l1 = [1, 2, 3, 4, 5, 6]
>>> l1[0:6]  # de l1[0] à l1[6-1] (renvoie une liste)
[1, 2, 3, 4, 5, 6]
>>> l1[1:4]  # de l1[1] à l1[3]
[2, 3, 4]
>>> l1[1:]  # tous les éléments, sauf le premier
[2, 3, 4, 5, 6]
>>> l1[:-1]  # tous les éléments, sauf le dernier
[1, 2, 3, 4, 5]
>>> l1[0:len(l1):2]  # un élément sur deux, en partant du premier
[1, 3, 5]
>>> l1[::2]  # idem, simplifié
[1, 3, 5]
>>> l1[::-1]  # tous les éléments, dans l'ordre inverse
[6, 5, 4, 3, 2, 1]
>>> l1[::-2]  # un élément sur deux, dans l'ordre inverse, en partant du dernier
[6, 4, 2]
>>> l1[4:1:-2] # de l1[4] à l1[1+1] par pas de -2
[5, 3]
```

Le tranchage de liste (suite)

```
>>> l1 = [1, 2, 3, 4, 5, 6]
>>> l1[3:6] = [7, 8, 9]  # remplct d'une tranche par une autre de même longueur
>>> l1
[1, 2, 3, 7, 8, 9]
>>> l1[5:6] = [9, 10]  # remplct d'une tranche par une plus longue
>>> l1
[1, 2, 3, 7, 8, 9, 10]
>>> l1[0:3] = [0]  # remplct d'une tranche par une plus courte
>>> l1
[0, 7, 8, 9, 10]
>>> l1[0:1] = []  # suppression d'une tranche (ici de 1 élément)
>>> l1
[7, 8, 9, 10]
>>> l1 = [1, 2, 3, 4, 5, 6]
>>> l1[0:4] = l1[3:5]  # avec des tranches ayant des éléments en commun
>>> l1
[4, 5, 5, 6]
```

Le tranchage de liste (suite)

```
>>> l1 = [1, 2, 3]
>>> l2 = l[:]  # autre solution pour une copie
>>> l1[0] = 4  # modif de l1
>>> l2  # non répercutée sur l2
[1, 2, 3]
```

Les listes en compréhension

```
>>> l1 = [i for i in range(1,7)]  # premier exemple
>>> l1
[1, 2, 3, 4, 5, 6]
>>> l2 = []  # construction équivalente avec une boucle for
>>> for i in range(1,7):
...   l2.append(i)
...
>>> l2
[1, 2, 3, 4, 5, 6]
>>> l1 = [x+1 for x in l1]  # avec une opération sur la variable de boucle
>>> l1
[2, 3, 4, 5, 6, 7]
>>> l2 = [car*2 for car in 'toto']  # fonctionne avec un itérable (ici 'toto')
>>> l2
['tt', 'oo', 'tt', 'oo']
>>> l2 = [x for x in l1 if x % 2 == 0]  # avec un filtre
>>> l2
[2, 4, 6]
```

Les listes en compréhension (suite)

```
>>> l1 = [i for i in range(1,7)]
>>> l2 = [x**2 for x in l1 if x % 2 == 0]  # avec une opération et un filtre
>>> l2
[4, 16, 36]
>>> [3*i+j for i in range(3) for j in range(3)]  # avec deux boucles imbriquées
[0, 1, 2, 3, 4, 5, 6, 7, 8]

>>> [[3*j+i for i in range(3)] for j in range(3)]  # exemple: matrice 3x3
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
>>>
>>> # une opération, une boucle et deux if
>>> [i+1 for i in range(10) if i%2 == 0 if i>=5]
[7, 9]
```

Les listes de listes

```
>>> l = [[1, 2, 3, 4],  # une liste 2D (régulière) à 3 lignes et 4 colonnnes
...      [5, 6, 7, 8],
...      [9, 10, 11, 12]]
>>> l
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
>>> len(l)  # le nombre de lignes
3
>>> len(l[0])  # le nombre de colonnes
4
>>> l[1][2]  # élément sur la ligne d'indice 1 et la colonne d'indice 2
7
>>> l[1]  # la ligne d'indice 1 (deuxième ligne)
[5, 6, 7, 8]
>>> [l[i][2] for i in range(len(l))]  # la colonne d'indice 2
[3, 7, 11]
```

Sommaire

1. Listes
   - Les bases
   - Le «découpage» (slicing) de liste
   - Les listes en compréhension
   - Les listes de listes
2. Les tuples
3. Les dictionnaires

Les tuples

```
>>> point = (1, 2, 3)  # un tuple à trois éléments
>>> point
(1, 2, 3)
>>> point1 = ()  # tuple vide
>>> point1
()
>>> t1 = (1)
>>> t1
1  # surprise!
>>> type(t1)
<class 'int'>  # explication: (1) est l'entier 1 (ce qui est logique)
>>> t1 = (1,)  # solution: rajouter une virgule
>>> t1
(1,)
>>> len(point)  # nombre d'éléments
2
>>> point[1]  # élément d'indice 1
2
```

Les tuples (suite)

```
>>> point[0] = 3  # un tuples est immutable (non modifiable)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
>>> for i in range(len(point)):  # un tuple est iterable
...   print(point[i])
...
1
2
3
>>> 4 in point  # appartenance à un tuple
False
>>>
>>> point + (4, 5) # concaténation de 2 tuples
(1, 2, 3, 4, 5)
>>> point * 2  # réplication (ici duplication) d'un tuple
(1, 2, 3, 1, 2, 3)
```

Sommaire

1. Listes
   - Les bases
   - Le «découpage» (slicing) de liste
   - Les listes en compréhension
   - Les listes de listes
2. Les tuples
3. Les dictionnaires

Les dictionnaires

```
>>> dico = {'a': 1, 'b': 2}  # dictionnaire à deux clefs
>>> dico
{'a': 1, 'b': 2}
>>> dico1 = {}  # dictionnaire vide
>>> len(dico), len(dico1)  # longueur d'un dictionnaire
(2, 0)
>>> dico['a']  # accès à une valeur par sa clef
1
>>> [clef for clef in dico]  # liste des clefs (qui peuvent être inconnues)
['a', 'b']
>>> list(dico.keys())  # autre façon
['a', 'b']
>>> for clef in dico:  # un dictionnaire est itérable
...   print(dico[clef])
...
1
2
>>> dico['a'] = 3  # un dictionnaire est mutable
>>> dico
{'a': 3, 'b': 2}
>>> 'b' in dico  # appartenance d'une clef à un dictionnaire
True
```

Les dictionnaires (suite)

```
>>> dico1 = {'a': 1, 'c': 3, 'b': 2}
>>> dico2 = {'c': 3, 'b': 2, 'a': 1}
>>> dico1 == dico2  # test d'égalité de deux dictionnaires
True
>>> dico3 = {'a': 1, 'a': 1}  # que se passe-t-il ?
>>> dico3
{'a': 1}  # réponse: un seul couple (clef, valeur) enregistré
>>>
>>> dico3 = {'a': 1, 'a': 2}  # que se passe-t-il ?
>>> dico3['a']
2
>>> dico3
{'a': 2}  # réponse: le dernier couple ayant la clef répétée prévaut
>>>
>>> dico = {'a': 1, 'b': 2}
>>> dico['c'] = 3  # ajout d'un nouveau couple (clef, valeur)
>>> dico
{'a': 1, 'b': 2, 'c': 3}
>>> dico.pop('b')  # suppression d'une clef (et de sa valeur)
2
>>> dico
{'a': 3, 'c': 4}
```