Listes, tuples et dictionnaires

Partie 2

PCSI - Lycées Condorcet / Jean Jaurès - 2024

1.1.7 Copie d'une liste, copie en profondeur

Question : Que contiennent les variables `s` et `t` à la fin des instructions ci-dessous ?

[ ]:
```
s=[2,[2,3]]
t=s
t[0]=4

print(s)
print(t)
```

Réponse :

On a vu qu'on peut copier une liste `L1` à l'aide de l'instruction `L2=L1[:]`.

On peut aussi utiliser la *méthode* `copy()` qui a exactement le même effet :

[ ]:
```
s=[2,[2,3]]
t=s.copy()
t[0]=4

print(s)
print(t)
```

```
[2, [2, 3]]
[4, [2, 3]]
```

Question : Que contiennent les variables `s` et `t` à la fin des instructions ci-dessous ?

[ ]:
```
s=[2,[2,3]]
t=s.copy()
t[1][0]=4

print(s)
print(t)
```

Réponse :

Pour palier ce problème, on peut utiliser une *copie profonde* du module `copy` :

[ ]:
```
import copy
s=[2,[2,3]]
t=copy.deepcopy(s)
t[1][0]=4

print(s)
print(t)
```

```
[2, [2, 3]]
[2, [4, 3]]
```

1.1.8 Opérations sur les listes et tuples

Les différentes opérations au programme de PCSI/PSI/PC sur les listes sont résumées dans le tableau ci-dessous, dans lequel les variables `s` **et** `t` **sont soit deux listes, soit deux tuples**.

Instruction	Effet	Remarque
`len(s)`	Renvoie le nombre d'éléments de la liste/tuples	
`s[i]`	Renvoie l'élément d'indice `i` de la chaîne. Les indices vont de `0` à `len(L)-1`	Renvoie une erreur si `i>len(L)-1`, mais on peut utiliser des indices négatifs
`s[i:j]`	Renvoie une liste (ou un tuple) constitués des éléments de `s` allant de `s[i]` à `s[j-1]`	Ne renvoie pas d'erreur si `j` dépasse la longueur de `s`.
`s[i:]`	Renvoie une liste (ou un tuple) contenant tous les éléments de `s` à partir de `s[i]` (inclus)	
`s[:j]`	Renvoie une liste (ou un tuple) contenant tous les éléments de `s` jusqu'à `s[j-1]`	Ne renvoie pas d'erreur si `j` dépasse la longueur de `s`.
`s[i:j:p]`	Renvoie une liste (ou un tuple) constitués des éléments de `s` allant de `s[i]` à `s[j-1]`, en avançant par pas de `p`.	
`s+t`	Renvoie une liste (ou un tuple) qui est la concaténation de `s` et `t`	
`C*4`	Renvoie une liste (ou un tuple) qui une concaténation de 4 copies de la chaine `s`.	
`x in s`	Renvoie `True` si un élement de `s` est égal à `x` et `False` sinon.	
`x not in s`	Renvoie `False` si un élement de `s` est égal à `x` et `True` sinon.	

Exemples :

[5]:
```
L=[5]*10
print(L)
print(4 in L)
print(5 in L)
print(4 not in L)
```

```
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
False
True
True
```

Ajout ou suppression d'un élément à la fin d'une liste

Ces deux opérations sont spécifiques aux listes (et aux dictionnaires) sont à connaître et ne fonctionnent pas avec des tuples.

Instruction	Effet
`L.append(x)`	Ajoute `x` à la fin de `L`.
`L.pop()`	Supprime le dernier élément de `L`, et le renvoie

Tri d'une liste, tri d'un tuple.

On peut trier une **liste** (pas un tuple) directement à l'aide de la méthode `sort()`, hors programme, mais bien utile parfois.

Instruction	Effet
`L.sort()`	Tri la liste `L` par ordre croissant.

L'instruction `sorted(L)` permet de renvoyer une **liste** des valeurs de L (que L soit une liste ou un tuple), triées par ordre croissant.

Instruction	Effet
`sorted(L)`	Renvoie une liste des valeurs de `L` triées par ordre croissant.

Conversion d'une liste en tuple et réciproquement

Si `L` est une liste et `T` un tuple :

Instruction	Effet
`tuple(L)`	Renvoie un tuple comprenant les valeurs de la liste `L`.
`list(T)`	Renvoie une liste comprenant les valeurs du tuple `T`.

On peut bien entendu combiner ces fonctions pour trier un tuple par exemple.

[ ]:
```
L=[3,2,1,4]
L.sort()
print(L)
T=(3,2,1,4)
L2=sorted(T)
print(L2)
T2=tuple(L2)
print(T2)
print(list(T2))
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
(1, 2, 3, 4)
[1, 2, 3, 4]
```

Exercice 9 Ecrire une fonction `communs(L1,L2)` qui renvoie une liste des éléments communs aux listes `L1` et `L2`. Les éléments de la liste renvoyée doivent être 2 à 2 distincts.

[ ]:
```
```

[ ]:
```
#Test :
L=[1,2,3,4,5,7,9]
S=[2,4,6,7,8,10,11]
print(communs(L,S))
```

Création par compréhension

Cette technique consiste à utiliser la structure suivante : `[e for x in s]` où :

- `e` est une expression dépendant ou non de `x`
- `s` est un *itérable* c'est-à-dire que ça peut-être :
  - une chaine de caractère,
  - un tuple,
  - une expression de la forme range(n) ou range(n,m),
  - une liste,
  - un dictionnaire.

Voici quelques exemples que vous pouvez exécuter pour observer leur effet :

[ ]:
```
L=[x**2 for x in range(-10,11)]
print(L)
L=[4 for x in range(5)]
print(L)
L=[c for c in "bibliothèque"]
print(L)
L=["_"+c+"_" for c in "bibliothèque"]
print(L)
L=[2*a for a in [1,2,3,4]]
print(L)
```

```
[100, 81, 64, 49, 36, 25, 16, 9, 4, 1, 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[4, 4, 4, 4, 4]
['b', 'i', 'b', 'l', 'i', 'o', 't', 'h', 'è', 'q', 'u', 'e']
['_b_', '_i_', '_b_', '_l_', '_i_', '_o_', '_t_', '_h_', '_è_', '_q_', '_u_',
'_e_']
[2, 4, 6, 8]
```