

THÈME 7: LISTES

Notions du thème:

- Listes:
 - Structure de données
 - Opérateurs sur les listes
 - Effets de bord sur les listes

Types simples et types complexes

```
>>> a = 6
>>> type(a)
<class 'int'>
>>> type(3.5)
<class 'float'>
>>> type(True)
<class 'bool'>
```

- Types simples
- Une seule valeur

- Besoin de manipuler des « structures de données » plus complexes
 - Liste
 - Ensemble
 - Tableaux multi-dimensions
 - ...

Type Python « Liste »

- Ensemble **ordonné** d'éléments (objets)
- Associée à un identificateur
- Peut grandir (ou se réduire) dynamiquement
- Les éléments peuvent être de types différents
- Exemples:

```
Weekend=["Samedi", "Dimanche"]
```

```
Multiple3 = [3, 6, 9, 12]
```

```
Romain = [[1, 'I'], [2, 'II'], [3, 'III'], [4, 'IV']]
```

```
iv = 4
```

```
FourreTout = ["Un", 2, 3.0, iv]
```

```
Vide = []
```

Type Python « Liste » (suite)

```
>>> Weekend
```

```
['Samedi', 'Dimanche']
```

```
>>> Multiple3
```

```
[3, 6, 9, 12]
```

```
>>> FourreTout
```

```
['Un', 2, 3.0, 4]
```

```
>>> Romain
```

```
[[1, 'I'], [2, 'II'], [3, 'III'], [4, 'IV']]
```

Des opérateurs sur les listes

- `Liste[index]`
obtenir l'élément à l'index `index`
Les éléments sont indexés à partir de 0
- `Liste.append(element)`
ajout d'un seul élément à la fin de la liste

```
>>> Multiple3 = [3, 6, 9]
>>> Multiple3[0]
3
>>> Multiple3.append(21)
>>> Multiple3
[3, 6, 9, 21]
```

Exercice : remplir les trous :

```
>> Multiple3 = [3, 6, 9, 15, 21]
```

```
>> Multiple3[2]
```

```
>> Multiple3[_____]
```

```
21
```

```
>> Multiple3._____
```

```
>> Multiple3
```

```
[3, 6, 9, 15, 21, 24]
```

Des opérateurs sur les listes

- **len(Liste)**

Pour avoir la taille d'une liste

```
>> Multiple3 = [3, 6, 9, 15, 21, 24, 27]
```

```
>> len(Multiple3)
```

```
7
```

```
>> Rom = [[1, "I"], [2, "II"], [3, "III"], [4, "IV"]]
```

```
>> len(Rom)
```

```
4
```


Afficher une liste

```
>>> l1=[1,2,3] # crée une liste
>>> print(l1)
[1, 2, 3]
```

Tester si un élément est dans une liste: mot-clé **in**

```
def cherche(elem, l):  
    if elem in l:  
        return True  
    else:  
        return False  
  
# prog. principal  
ma_liste=[2,5,8,12,17,25]  
trouve=cherche(12,ma_liste)  
print(trouve) # affiche True  
trouve=cherche(6, ma_liste)  
print(trouve) # affiche False
```

Des opérateurs sur les listes

- `Liste.insert(index, element)`

Ajout d'un seul élément à l'index `index`

- `Liste.extend(liste2)`

Ajout d'une liste à la fin de la liste

```
>> Multiple3 = [3, 6, 9, 21]
```

```
>> Multiple3.insert(3, 15)
```

```
>> Multiple3
```

```
[3, 6, 9, 15, 21]
```

```
>> Multiple3[3]
```

```
15
```

```
>> Multiple3.extend([24, 27])
```

```
>> Multiple3
```

```
[3, 6, 9, 15, 21, 24, 27]
```

Exercice : remplir les trous :

```
>>> Multiple3 = [3, 6, 9, 15, 21, 24, 27]
```

```
>>> Multiple3.insert( ____, 12)
```

```
>>> Multiple3.insert( ____, 18)
```

```
>>> Multiple3
```

```
[3, 6, 9, 12, 15, 18, 21, 24, 27]
```

```
>>> Multiple3. _____ ([30, 33])
```

```
>>> Multiple3
```

```
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33]
```

```
>>> Multiple3. _____
```

```
>>> Multiple3
```

```
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36]
```

**ATTENTION
ZONE
PIÉGÉE**

Des opérateurs sur les listes

- **Liste.pop(index)**
retire l'élément présent à la position `index` et le renvoie
- **Liste.remove(element)**
retire l'élément donné (le premier trouvé)

```
>>> Multiple3 = [3, 6, 9, 15, 21, 24, 27, 24, 24]
>>> a = Multiple3.pop(0)
>>> a
3
>>> Multiple3
[6, 9, 15, 21, 24, 27, 24, 24]
>>> Multiple3.remove(24)
>>> Multiple3
[6, 9, 15, 21, 27, 24, 24]
```

Exercice : remplir les trous :

```
>> EhEh = ["tra", "la", "la", "la", "lère"])
```

```
>> EhEh._____
```

```
>> EhEh
```

```
["tra", "la", "la", "la"]
```

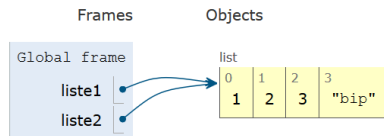
```
>> EhEh._____
```

```
>> EhEh
```

```
["tra", "la", "la"]
```

L'opérateur = sur les listes

```
liste1=[1,2,3]
liste2=liste1
liste2.append("bip")
```



ATTENTION
ZONE
PIÉGÉE

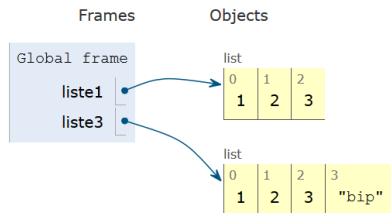
L'égalité permet de donner 2 noms
à la même liste !

Les modifications apportées à une des listes après la copie **s'appliquent également** à l'autre liste.

Copie de listes

- Pour copier une liste, on peut utiliser
 - la fonction `list()`
 - l'opération générique `copy.deepcopy()` (`import copy`)

```
liste1=[1, 2, 3]
liste3=list(liste1)
liste3.append("bip")
```

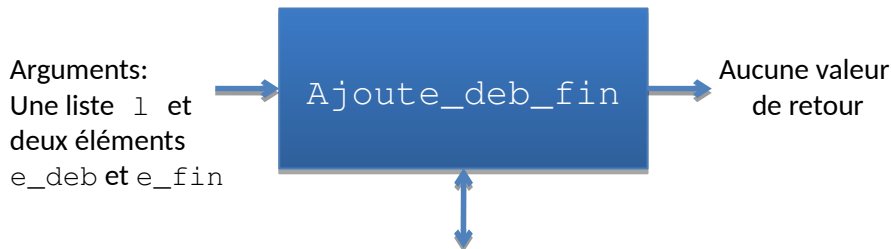


Les modifications apportées à une des listes après la copie **n'affectent pas** l'autre liste.

Effets de bord qui modifient une liste

Une fonction peut **modifier** une liste passée en argument, **indépendamment de sa valeur de retour**.

C'est une nouvelle forme **d'effet de bord** (jusqu'ici: `print`, `input`, `turtle`).



Effets de bords:

Insère `e_deb` au début de `l` et `e_fin` à la fin de `l`

Effets de bord qui modifient une liste

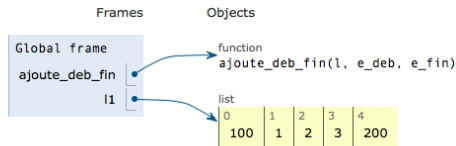
Python 3.6

```
1 def ajoute_deb_fin(l, e_deb, e_fin):
2     l.append(e_fin)
3     l.insert(0, e_deb)
4
5
6 l1=[1,2,3]
7 ajoute_deb_fin(l1, 100, 200)
8 print(l1)
```

[Edit code](#) | [Live programming](#)

Print output (drag lower right corner to resize)

```
[100, 1, 2, 3, 200]
```



Les modifications apportées à la liste dans la fonction sont conservées après la sortie de la fonction.

Attention aux effets de bord non désirés!

Il faut bien **penser à copier** la liste passée en argument si **on ne souhaite pas qu'elle soit modifiée** par la fonction.

Exemple **avec oubli** de la copie:

```
import random

def ajoute_random(l) :
    """ Renvoie une liste obtenue a partir de l en
    ajoutant un entier aleatoire entre 5 et 10 """
    x=random.randint(5,10)
    l.append(x)
    return l
```

Exemple avec oubli de la copie

La liste argument `l` est modifiée dans la fonction.

Python 3.6

```

1 import random
2
3 def ajoute_random(l):
4     x=random.randint(5,9)
5     l.append(x)
6     return l
7
8 l1=[1,2,3]
9 l2=ajoute_random(l1)
10 print(l2)

```

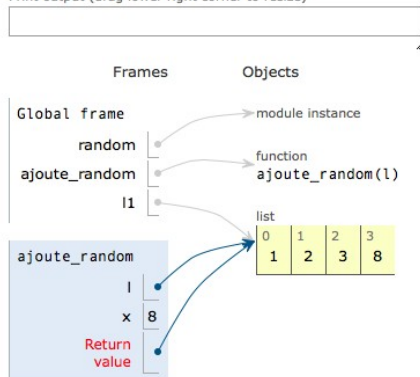
[Edit code](#) | [Live programming](#)

not executed

execute

set a breakpoint; use the Back and Forward buttons to jump there.

Print output (drag lower right corner to resize)



Exemple (suite)

Python 3.6

```

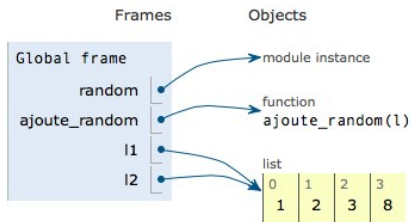
1 import random
2
3 def ajoute_random(l):
4     x=random.randint(5,9)
5     l.append(x)
6     return l
7
8 l1=[1,2,3]
9 l2=ajoute_random(l1)
10 print(l2)

```

[Edit code](#) | [Live programming](#)

Print output (drag lower right corner to resize)

[1, 2, 3, 8]



La liste `l1` a été modifiée alors que l'on ne le souhaitait pas.

Sans effets de bord non désirés

Exemple corrigé: copie de la liste avec `list (...)`

```
import random

def ajoute_random(l):
    """ Renvoie une liste obtenue a partir de l en
    ajoutant un entier aleatoire entre 5 et 10"""
    x=random.randint(5,10)
    ma_liste=list(l)
    ma_liste.append(x)
    return ma_liste
```

Sans effets de bord non désirés

Python 3.6

```

1 import random
2
3 def ajoute_random(l):
4     x=random.randint(5,10)
5     ma_liste=list(l)
6     ma_liste.append(x)
7     return ma_liste
8
9 l1=[1,2,3]
10 l2=ajoute_random(l1)
11 print(l1, l2)

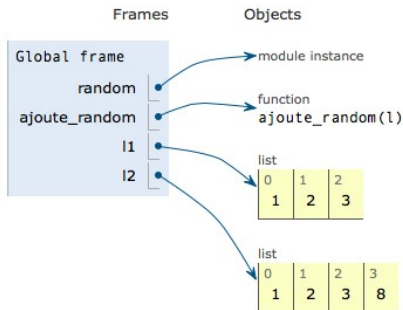
```

[Edit code](#) | [Live programming](#)

t executed
:ute

Print output (drag lower right corner to resize)

```
[1, 2, 3] [1, 2, 3, 8]
```



l2 est créée correctement et l1 n'est pas modifiée.

Erreur classique

Erreur à ne pas commettre:

```
import random

def ajoute_random(l):
    """ Renvoie une liste obtenue a partir de l en
    ajoutant un entier aleatoire entre 5 et 10"""
    x=random.randint(5,10)
    ma_liste=l
    ma_liste.append(x)
    return ma_liste
```


Avec effets de bord non désirés

Python 3.6

```

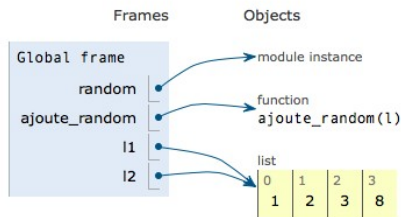
1 import random
2
3 def ajoute_random(l):
4     x=random.randint(5,10)
5     ma_liste=l
6     ma_liste.append(x)
7     return ma_liste
8
9 l1=[1,2,3]
10 l2=ajoute_random(l1)
11 print(l1, l2)

```

[Edit code](#) | [Live programming](#)

Print output (drag lower right corner to resize)

```
[1, 2, 3, 8] [1, 2, 3, 8]
```



Le symbole = n'a pas créé une copie de l'argument `l`
 donc la liste `l1` est **modifiée involontairement**.

- Ces slides ont été réalisés par:
 - Amir Charif
 - Lydie Du Bousquet
 - Aurélie Lagoutte
 - Julie Peyre
 - Florence Thiard
- Leur contenu est placé sous les termes de la licence **Creative Commons CC BY-NC-SA**

