Les listes

Vendredi 14 novembre

Option Informatique

Ecole Alsacienne

Plan

1. Avant de nous lancer…
2. Définitions
3. Caractéristiques
4. Les listes en Python
5. Premières fonctions
6. Début du TD

AVANT DE NOUS LANCER

L'ECOLE DANS 14 ANS

- Vendredi 21 novembre à 18h00
  - Cérémonie officielle d'inauguration

- Prises de parole :
  - Alain Grangé Cabane, président du Conseil d'administration de l'Ecole
  - Vincent Peillon, ancien ministre de l'Education nationale
  - Emmanuel Larroche, professeur d'histoire et de géographie à l'Ecole
  - Pierre de Panafieu, directeur de l'Ecole

- Cours banalisé : pas d'option informatique ce jour-là

Fin du premier trimestre

- Date limite de saisie des notes : lundi 1er décembre à 06h00

- Evaluation pour le premier trimestre :
  - TD *Bases de programmation – 2e partie*
  - TD *Les tableaux en pratique*
  - Comportement en classe
  - Investissement pendant et entre les cours

- Date limite de rendu des TD : Dimanche 23 novembre à 23h59
  - Tout retard impliquera une pénalité de points
  - Tout TD non rendu sera fortement pénalisé

- Rappel : Vous ne serez pas pénalisé si vous posez une question ou si vous envoyez d'abord une version incomplète.

DÉFINITIONS

POUR COMMENCER…

Qu'est-ce qu'une liste ?

En quelque sorte… une pile de crêpes !

PLUS FORMELLEMENT

Définition récursive

Une *liste chaînée* l d'éléments de type T est

- soit la liste vide : $l = [\,]$
- soit un élément t de type T suivi d'une liste l_2 d'éléments de type T : $l = t :: l_2$

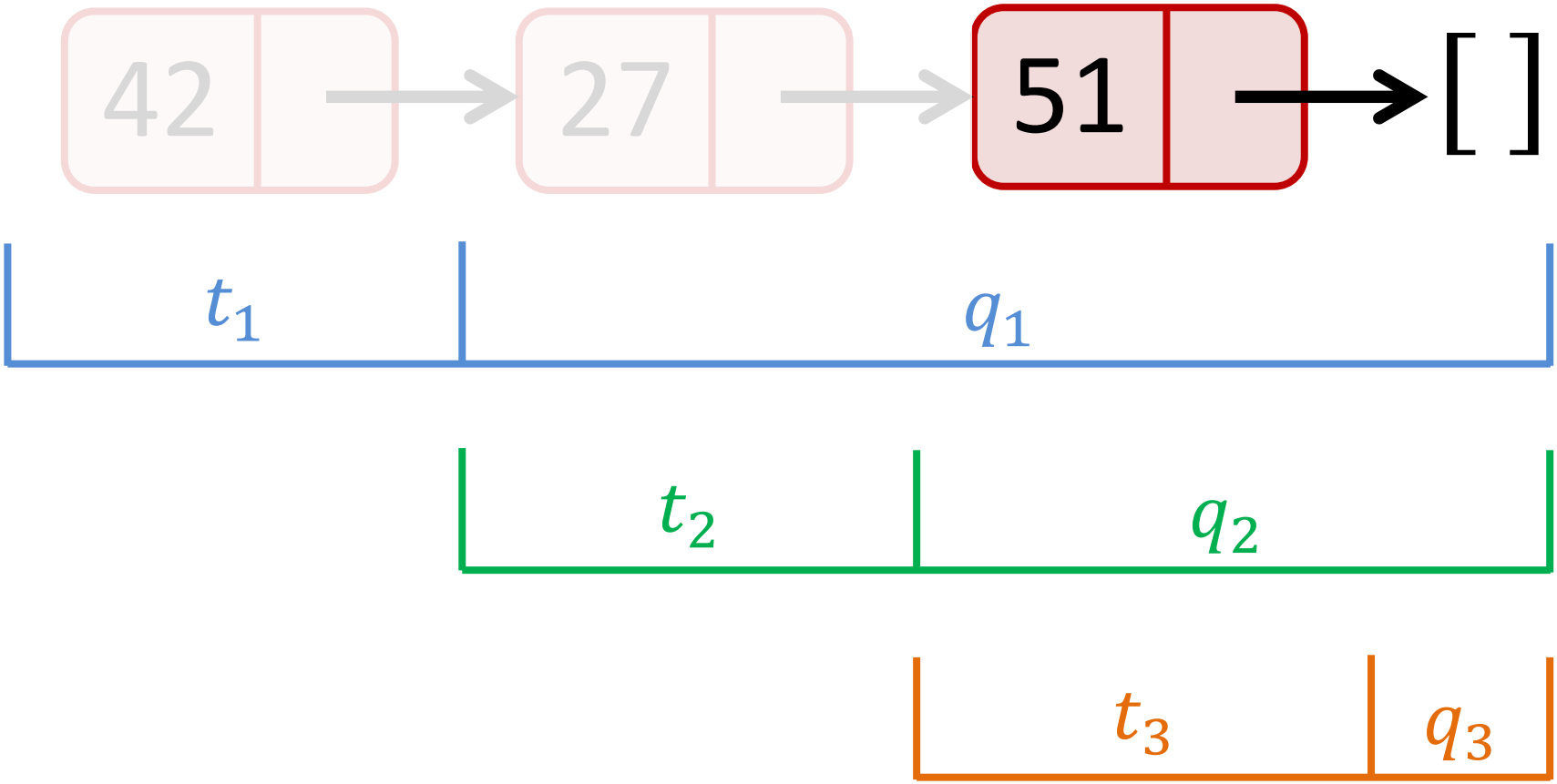
Caractérisation d'une liste non vide

- Si l est une liste non vide $(l \neq [\,])$, alors l s'écrit nécessairement sous la forme $l = t :: q$

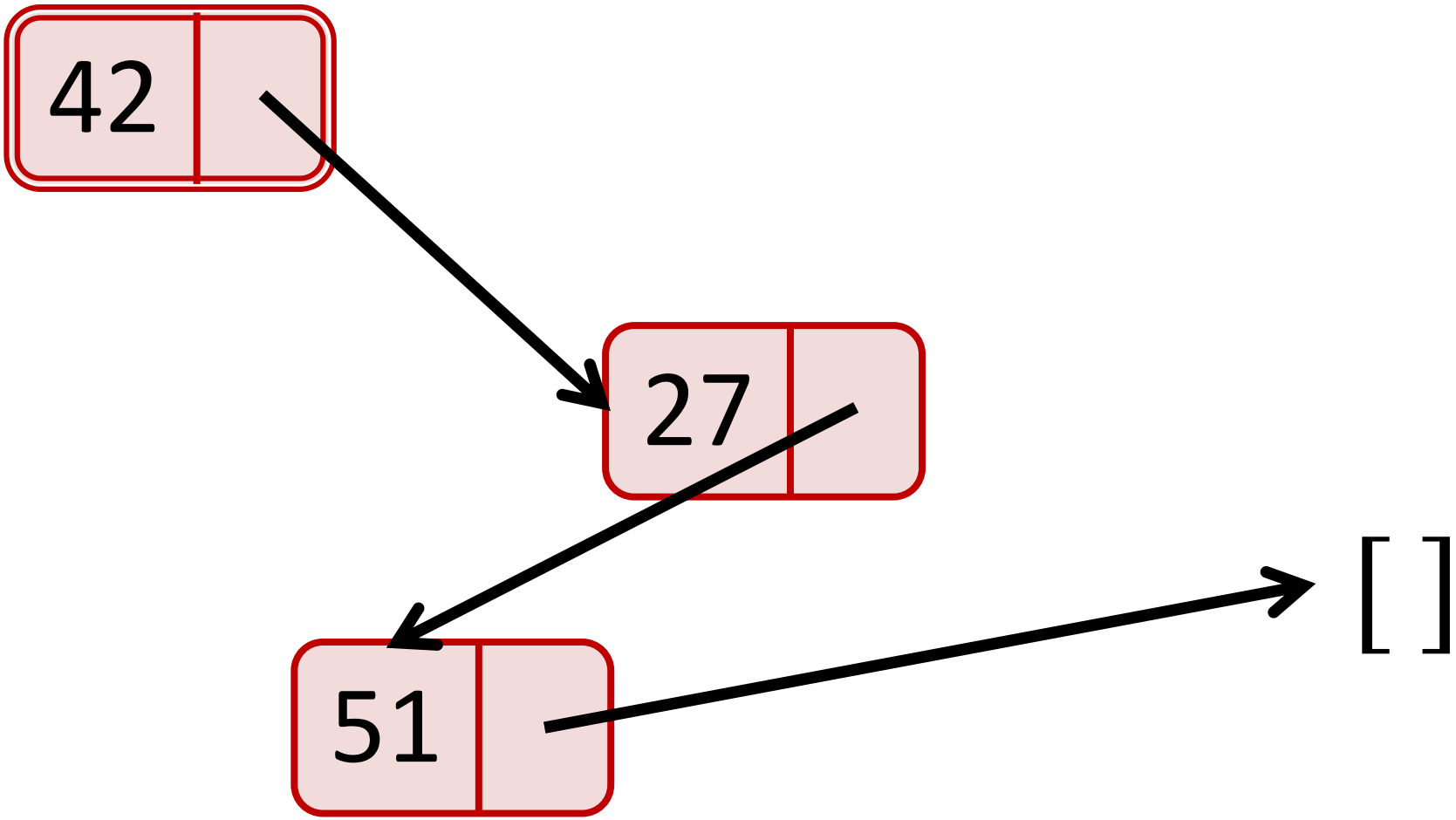
- Le premier élément t est alors appelé la tête de la liste l

- Le reste q est appelé queue de la liste

GRAPHIQUEMENT



Dans la mémoire



Caractéristiques

Mise en situation

- Que pouvez-vous faire face à cette structure ?
- Que ne pouvez-vous pas faire face à cette structure ?

L'ASSIETTE EST-ELLE VIDE ?

- La première chose qu'on vérifie en général avec une liste est sa **vacuité** : cette liste est-elle **vide** ou non ?

- On obtient ainsi un booléen qui permet un traitement au cas par cas :

```
Si EstVide(l)
    Alors
            ...
    Sinon
            ...
Fin Si
```

Ajouter une crèpe

- Que la liste soit vide ou non, on peut ajouter un élément
  - Dans une liste "classique", cet élément est ajouté "en dernier"

- En pseudo-code, on considère qu'on dispose d'une fonction `Empiler`, qui ajoute un élément à la fin d'une liste.
  - Exemple : `Empiler(x,l)`
  - Autre notation : `l.Empiler(x)`

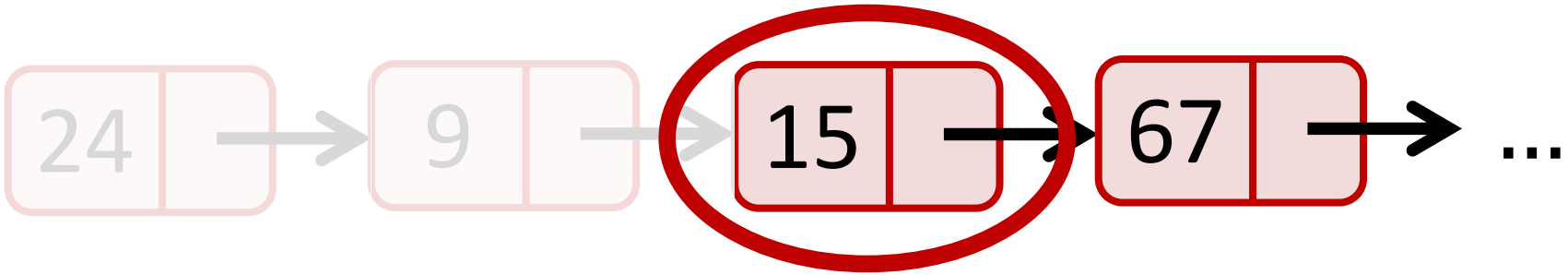
Retirer le dernier element

- Si une liste l est non vide, on peut récupérer le dernier élément de cette liste
  - Dans une liste "classique", il s'agit du dernier élément ajouté
  - On parle donc de "LIFO", de l'anglais *Last In, First Out*

- En pseudo-code, on considère qu'on dispose d'une fonction `Depiler` qui retire le dernier élément d'une liste et le renvoie
  - Exemple : `x = Depiler(l)`
  - Autre notation : : `x = l.Depiler()`

Accès au iÈme élément d'une liste

- **Question** : *comment accéder au 3e élément d'une liste ?*
- **Réponse :** on retire le dernier élément… trois fois !



- **Conséquence :** On est obligé de parcourir une liste dans l'ordre, élément par élément

La taille d'une liste

- La taille d'une liste n'est **pas fixée** (ni même bornée)

- Pour connaître la taille d'une liste, il est généralement nécessaire de la **parcourir en entier**

- Cette opération ne peut donc se faire en temps constant. La durée de ce calcul est proportionnelle à la longueur de la liste : on parle de complexité **linéaire**.

Les listes en Python

Le type `list`

- En Python, le type `list` est utilisé pour représenter deux structures de données classiques en informatique :
  - Les tableaux ("meubles de taille fixe")
  - Les listes ("piles de crêpes")

- Dans la plupart des langages, les tableaux et les listes sont représentés par des types différents

- Selon les circonstances, on manipulera les objets de type `list` plutôt comme des tableaux, ou plutôt comme des listes.

La liste vide

- Deux syntaxes sont possibles pour déclarer une liste vide
  - `nom = []`
  - `nom = list()`

- Exemple

```
l0 = []
print(l0)
```

```
[]
```

- Il arrive fréquemment qu'on crée une liste vide, et qu'on la remplisse progressivement en y ajoutant des éléments.

Déclaration élément par élément

- Il est également possible de déclarer une liste élément par élément, en utilisant des crochets.

- Exemple

```
l1 = ["bleu", "rouge", "vert"]
print(l1)
```

```
['bleu', 'rouge', 'vert']
```

- Remarque :
  - La syntaxe est identique à celle utilisée pour les tableaux
  - C'est normal : c'est le même type `list` qui est utilisé dans les deux cas

LONGUEUR D'UNE LISTE

- La longueur d'une liste est le nombre d'éléments contenus dans cette liste

- Pour obtenir cette longueur, on utilise la fonction `len`

- Exemple :

```
l2 = [2, 4, 5]
longueur = len(l2)
print(longueur)

3
```

- Remarque : La liste vide a pour longueur 0

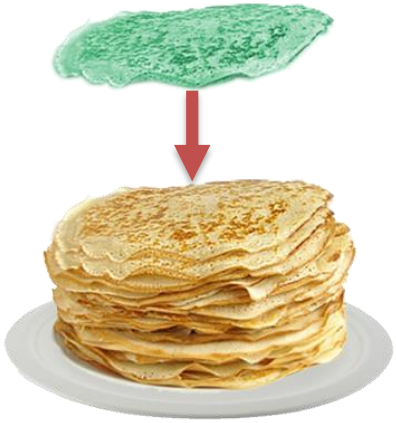
Ajouter un élément à la fin d'une liste

- Pour ajouter un élément à la fin d'une liste, on utilise la fonction `append`

- Exemple :

```
l3 = [2, 4, 5]
l3.append(3)
print(l3)

[2, 4, 5, 3]
```



- Le nouvel élément est ajouté en dernière position

- La liste est modifiée sur place (on ne crée pas une nouvelle liste avec un élément en plus)
  - En effet, on utilise la "méthode" `append` de l'objet `l3`
  - Ce sera le cas de la majorité des fonctions sur les listes
  - Si on ne veut pas modifier la liste initiale, il faut créer une copie

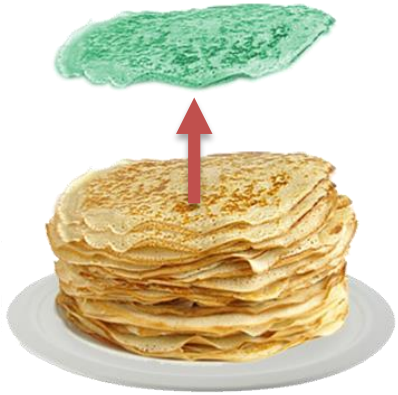
Retirer et renvoyer le dernier élément d'une liste

- La fonction `pop` permet de
  - retirer le dernier élément d'une liste
  - renvoyer cet élément

- Exemple :

```
l4 = [2, 4, 5]
x = l4.pop()
print(l5)
print(x)
```

```
[2, 4]
5
```



Concaténation de listes

- **Concaténer** deux listes, c'est les juxtaposer, les mettre l'une derrière l'autre dans une nouvelle grande liste.

- En Python, on dispose pour cela de la fonction `extend`

- Exemple :

```
l5 = [2, 4, 5]
l6 = [0, 7]
l5.extend(l6)
print(l5)
```

```
[2, 4, 5, 0, 7]
```

Listes et références

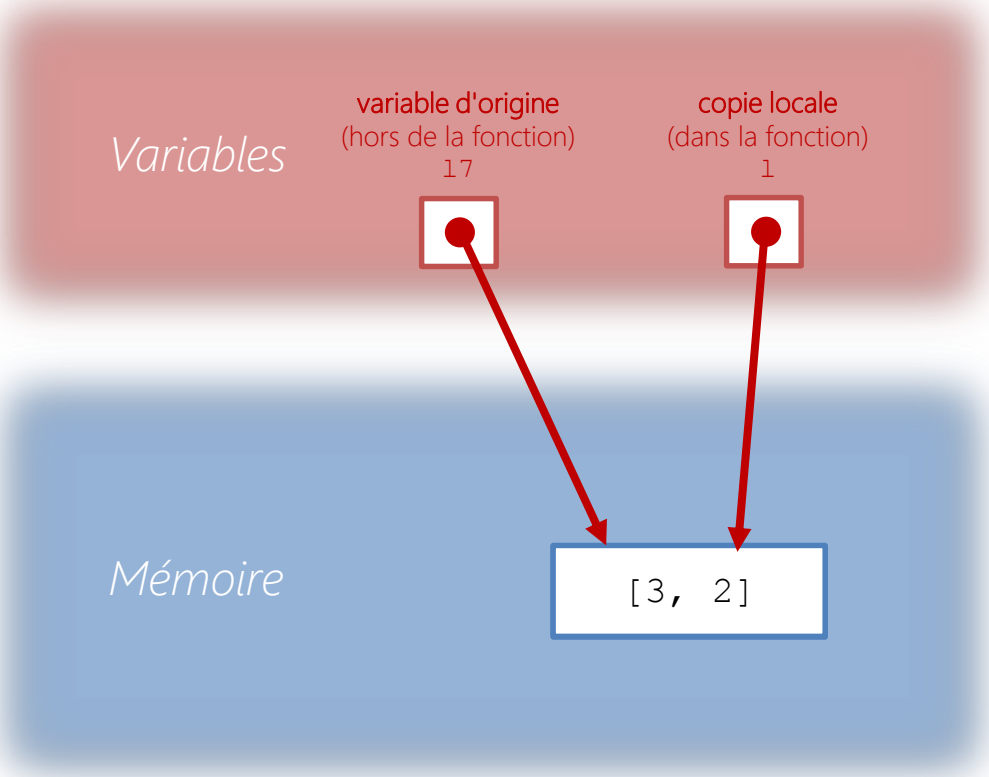
- **Rappel** : En général, les arguments des fonctions sont des références (on parle de "passage par référence")

- Exemple :

```
def empiler_2(l):
    l.append(2)

l7 = [3]
print(l7)
empiler_2(l7)
print(l7)
```

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



Listes et références

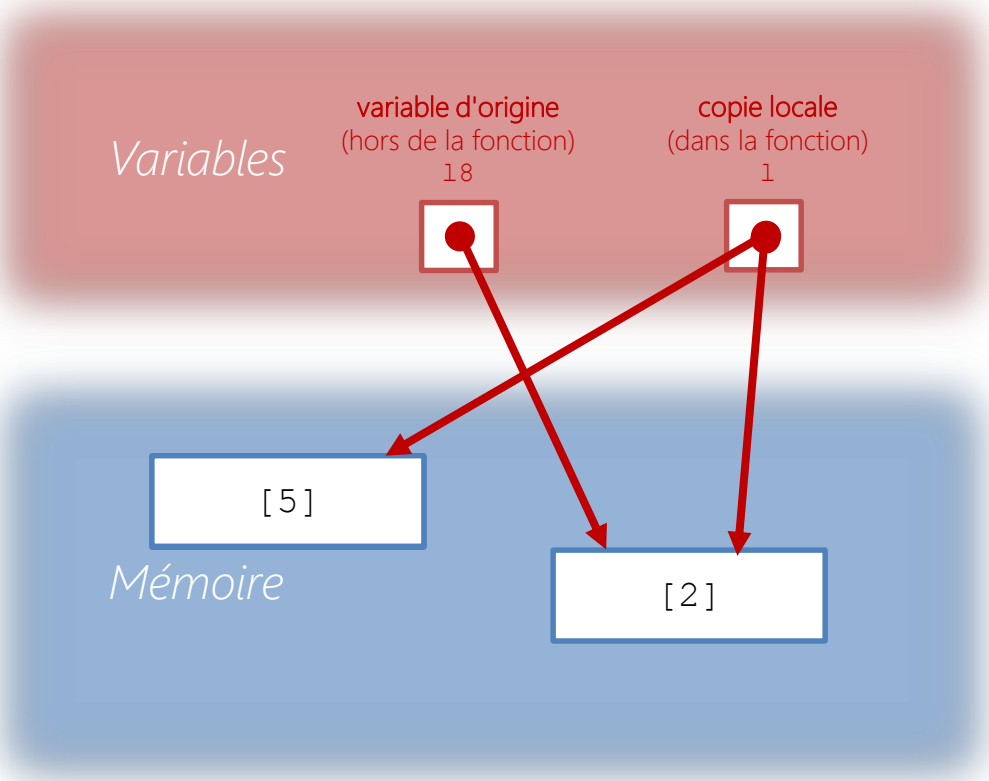
- Attention, si vous essayez de "redéfinir" la liste, vous perdrez la référence
  - Vos modifications ne seront valables que dans la fonction

- Exemple :

```
def nettoyer(l):
    l = []
    l.append(5)
    print(l)

l8 = [2]
print(l8)
nettoyer(l8)
print(l8)
```

```
[2]
[5]
[2]
```



Passage par valeur

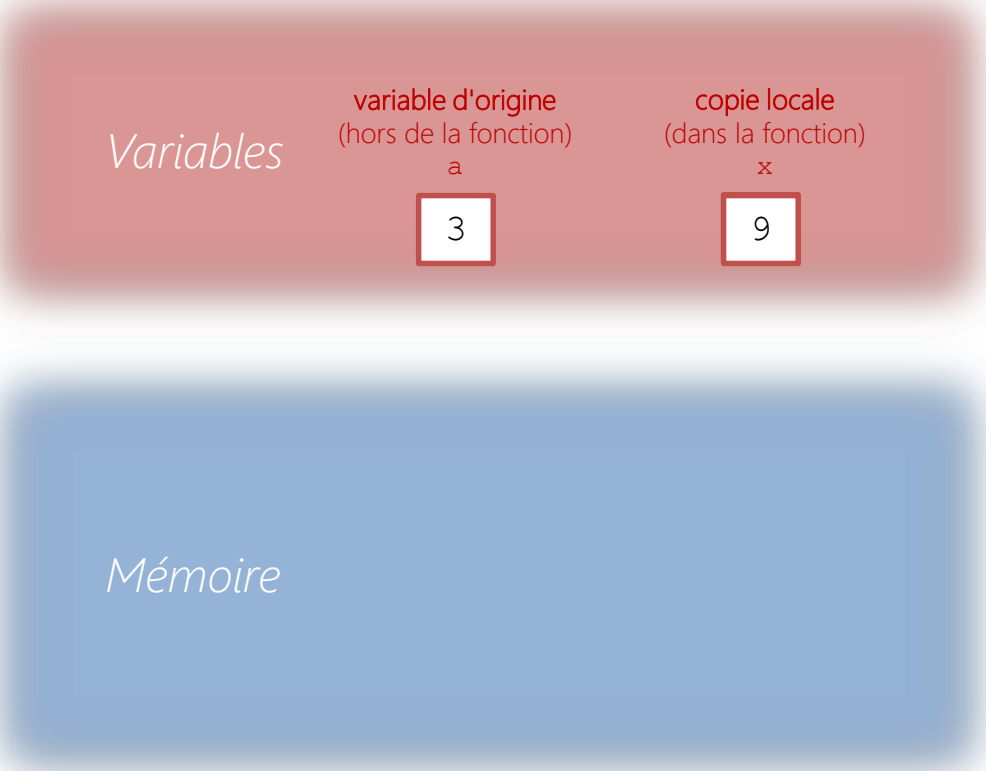
- Pour les types de base (par exemple int), il n'y a pas de référence : on passe directement une copie de la valeur
  - On parle de "passage par valeur"

- Exemple :

```
def carre(x):
    x = x*x
    print(x)

a = 3
print(a)
carre(a)
print(a)
```

```
[3]
[9]
[3]
```



Premières fonctions

Calculer la taille d'une liste

- **Question** : *Comment calculer la taille d'une liste ?*

- **Solution** (à un détail près) :

```
LongueurListe(l) =
    Si EstVide(l)
          Alors
                Renvoyer 0
          Sinon
                x = l.Depiler()
                Renvoyer 1 + LongueurListe(l)
    Fin Si
```

- **Problème** : On a vidé entièrement la liste
  - Pour éviter ce problème, on commence par créer une copie de la liste

TESTER L'APPARTENANCE

- **Question** : *Comment savoir si un élément* `x` *appartient à une liste* `l` *?*

- **Solution** (à un détail près) :

```
Appartient(x,l) =
    Si EstVide(l)
            Alors
                    Renvoyer Faux
            Sinon
                    t = l.Depiler()
                    Si (t=x)
                            Alors
                                    Renvoyer Vrai
                            Sinon
                                    Renvoyer Appartient(x,l)
                    Fin Si
    Fin si
```

- **Problème** : On a retiré certain éléments de la liste
  - Pour éviter ce problème, on commence par créer une copie de la liste

Renverser une liste

- Question : *Comment renverser une liste ?*

- Solution :

```
Miroir(l) =
    a_traiter = Copier(l)
    resultat = []
    Tant que Non(EstVide(a_traiter))
    Faire
        x = a_traiter.Depiler()
        resultat.Empiler(x)
    Fin Faire
    Retourner resultat
```

Fonction avancée sur les listes

- Python met à votre disposition un certain nombre de fonctions plus avancées sur les listes
  - `in` : détermine si un élément appartient à une liste
  - `index` : détermine la position de la première occurrence d'un élément dans une liste
  - `count` : détermine le nombre d'occurrence d'un élément dans une liste
  - `remove` : retire la première occurrence d'un élément dans une liste
  - `insert` : insère un élément à une position donnée dans une liste
  - `reverse` : "renverse" une liste
  - `sort` : trie une liste

Début du TD

Prochaine séance

Vendredi 28 novembre

[TD] Les listes en pratique (suite et fin)