



# Formation NSI

Jeudi 15 décembre – Lycée DE BORDA - Dax

lundi 5 décembre - Lycée Jacques Monod – Lescar

Amilcar DOS-SANTOS / Luc VINCENT

V 12.14

lundi 5 décembre – Bergerac

lundi 5 décembre – Pau

lundi 5 décembre – Bordeaux

Xavier DUPIN – Olivier ELOI – Christophe VIROULAUD –Marc ELDIN

[luc.vincent@ac-bordeaux.fr](mailto:luc.vincent@ac-bordeaux.fr)



# Déroulement

- **Matin**

- **Données structurées**
  - réflexion et échange sur les notions (les listes, les tableaux, les dictionnaires ...)
- **Parcours de listes et dictionnaires**
  - index, élément, énumération
- **Muable ou immuable**

- **Après midi**

- **Projet**
  - Puissance4 Présentation
  - Light\_out Mise en Œuvre
  - Grille d'évaluation



## Ressources

[luc.vincent@ac-bordeaux.fr](mailto:luc.vincent@ac-bordeaux.fr)



# Les programmes

- En première

## 2. Représentation des données types construits

<b>Tableau indexé, tableau donné en compréhension</b>	<p>Lire et modifier les éléments d'un tableau grâce à leurs index.</p> <p>Construire un tableau par compréhension.</p> <p>Utiliser des tableaux de tableaux pour représenter des matrices : notation <code>a[i][j]</code>.</p> <p>Itérer sur les éléments d'un tableau.</p>	<p>Seuls les tableaux dont les éléments sont du même type sont présentés.</p> <p>Aucune connaissance des tranches (slices) n'est exigible.</p> <p>L'aspect dynamique des tableaux de Python n'est pas évoqué.</p> <p>Python identifie listes et tableaux.</p> <p>Il n'est pas fait référence aux tableaux de la bibliothèque NumPy.</p>
<b>Dictionnaires par clés et valeurs</b>	<p>Construire une entrée de dictionnaire.</p> <p>Itérer sur les éléments d'un dictionnaire.</p>	<p>Il est possible de présenter les données EXIF d'une image sous la forme d'un enregistrement.</p> <p>En Python, les p-uplets nommés sont implémentés par des dictionnaires.</p> <p>Utiliser les méthodes <code>keys()</code>, <code>values()</code> et <code>items()</code>.</p>



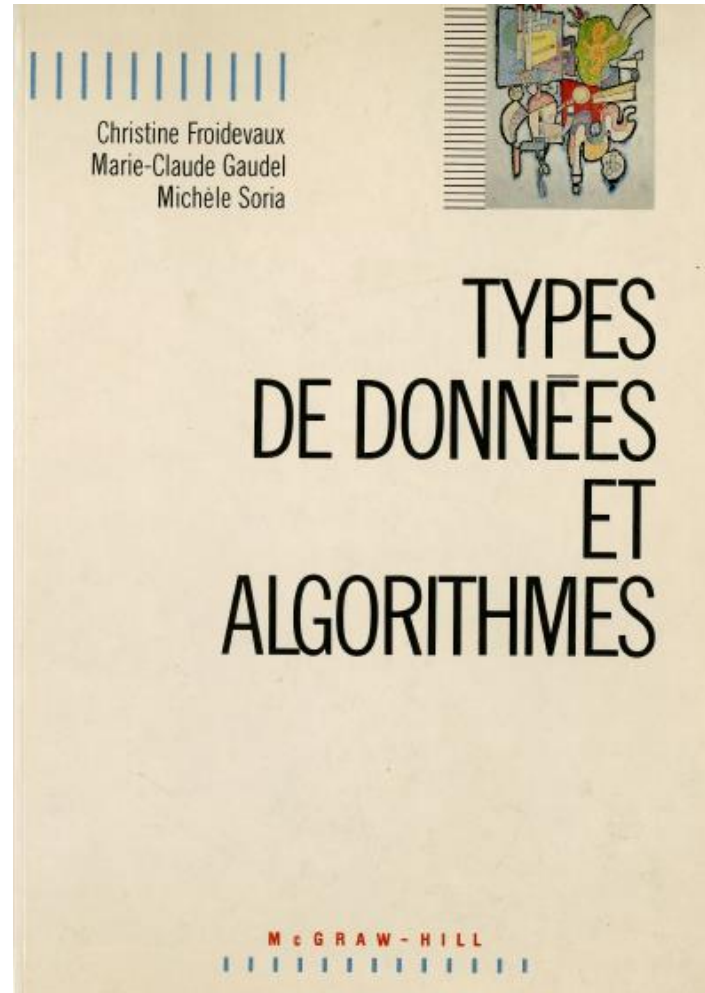
# Les programmes

- En terminale

## 1. Structures de données

<b>Listes, piles, files : structures linéaires.</b> <b>Dictionnaires, index et clé.</b>	Distinguer des structures par le jeu des méthodes qui les caractérisent. Choisir une structure de données adaptée à la situation à modéliser. Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.	On distingue les modes FIFO (first in first out) et LIFO (last in first out) des piles et des files.
--	--	--

# Les listes



luc.vincent@ac-bordeaux.fr

# Les listes

- Les listes linéaires sont la forme la plus commune d'organisation de données
- Une liste linéaire  $\lambda$  est une suite finie, éventuellement vide, d'éléments repérés selon leur rang dans la liste

$$\lambda = \langle e_1, e_2, \dots, e_n \rangle$$

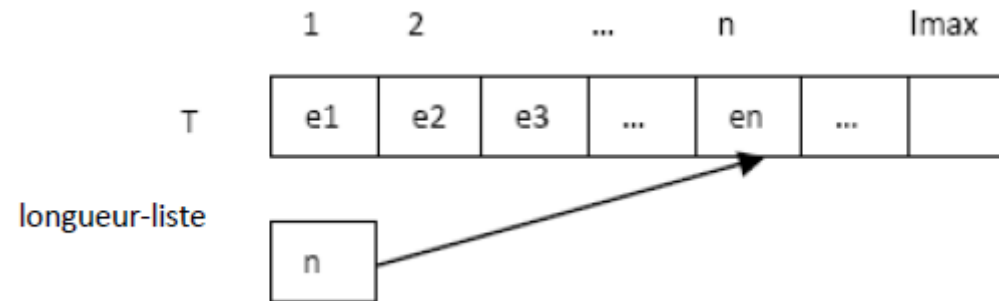
- L'ordre des éléments est déterminé par l'ordre des places des éléments.

# Le type abstrait liste itérative

Opérations	Signature	Description
liste_vide	$\rightarrow \text{liste}$	<i>Définir la liste vide</i>
accéder	$\text{liste} \times \text{position} \rightarrow \text{élément}$	<i>Accéder à un élément de la liste; l'élément se trouve à une place ( position) donnée dans la liste</i>
modifier	$\text{liste} \times \text{position} \times \text{élément} \rightarrow \text{liste}$	<i>Modifier un élément de la liste; Le nouvel élément se trouve à une place ( position) donnée dans la liste</i>
longueur	$\text{liste} \rightarrow \text{entier}$	<i>Calculer la longueur d'une liste</i>
supprimer	$\text{liste} \times \text{position} \rightarrow \text{liste}$	<i>Supprimer un élément à la liste; l'élément à supprimer se trouve à une place donnée dans la liste</i>
insérer	$\text{liste} \times \text{position} \times \text{élément} \rightarrow \text{liste}$	<i>Ajouter un élément à la liste; l'élément ajouté se trouvera ( après ajout) à une place ( position) donnée dans la liste</i>
est-elle vide ?	$\text{liste} \rightarrow \text{booléen}$	<i>Tester si une liste est vide</i>

# Représentation des listes

- Représentation contiguë



- Représentation chaînée





# Tableau

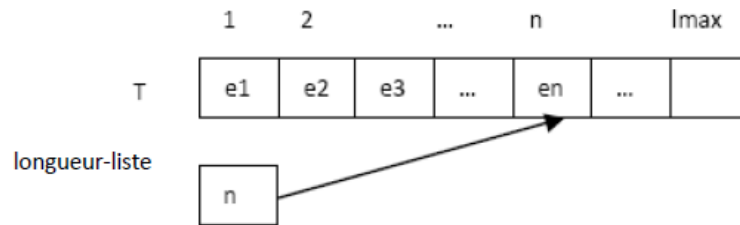
- Un tableau est une séquence ordonnée et contigüe d'éléments de même type.
- Le contenu d'un tableau est modifiable.

Un tableau en mémoire

h	e	l	l	o	!					
	3						9			
								6		
h	e	y	8	5	3	9	1	0	2	!
	3	4								

# Tableau

- La liste est représentée par un tableau dont la  $i_{\text{ème}}$  case est à la  $i_{\text{ème}}$  place de la liste
- La longueur de la liste ne peut pas dépasser la taille du tableau qui est surdimensionné
- La longueur de la liste est représentée par un entier
- La liste est donc représentée par un couple <tableau, entier>



Liste	Tableau	index	0	1	2	3	4	5	6	7	8
	T	valeur	8	5	3	9	1	0	2		
	Longueur	7									

# Tableau

- Il est simple d'accéder au  $k_{i\text{ème}}$  élément d'une liste
- On accède à un élément en temps constant grâce à son index .

5<sup>ième</sup> élément

tableau	index	0	1	2	3	4	5	6	7	8
	valeur	8	5	3	9	1	0	2		

# Tableau

- Il est simple d'insérer ou de supprimer le **dernier** élément de la liste
- Si l'élément à insérer ou supprimer est à la  $k^{\text{ième}}$  place de la liste il faut déplacer tous les éléments d'index  $k-1$  à l'index  $\text{Liste}(\text{long})-1$  dans le tableau.

index	0	1	2	3	4	5	6	7	8
valeur	8	5	3	9	1	0	2		



index	0	1	2	3	4	5	6	7	8
valeur	8	5	3	9	6	1	0	2	

# Tableau

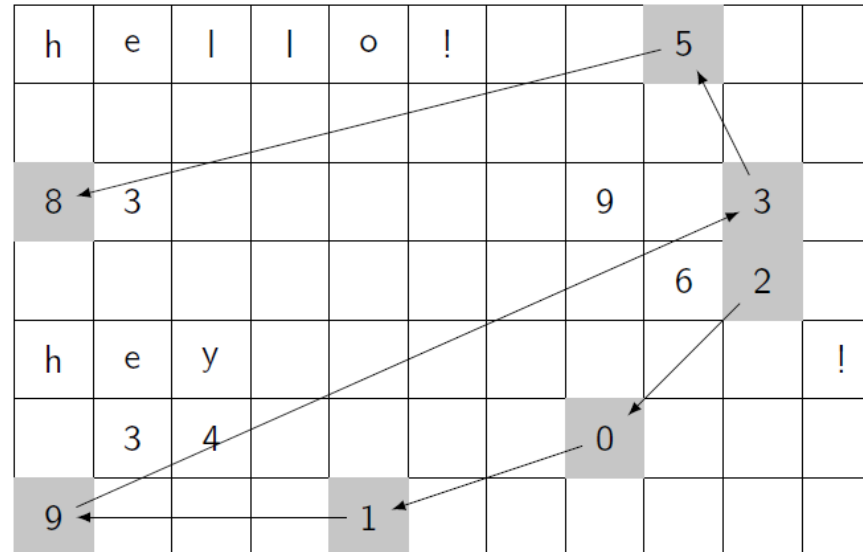
- Agrandir un tableau peut prendre un temps linéaire

h	e	l	l	o	!					
	3						9			
								6		
h	e	y	8	5	3	9	1	0	2	!
	3	4	↓	↓	↓	↓	↓	↓	↓	
			↓	↓	↓	↓	↓	↓	↓	7

# Liste chaînée

- Une liste chaînée est une séquence ordonnée où chaque élément possède une référence vers le suivant.

Une liste chaînée en mémoire





# Liste chaînée

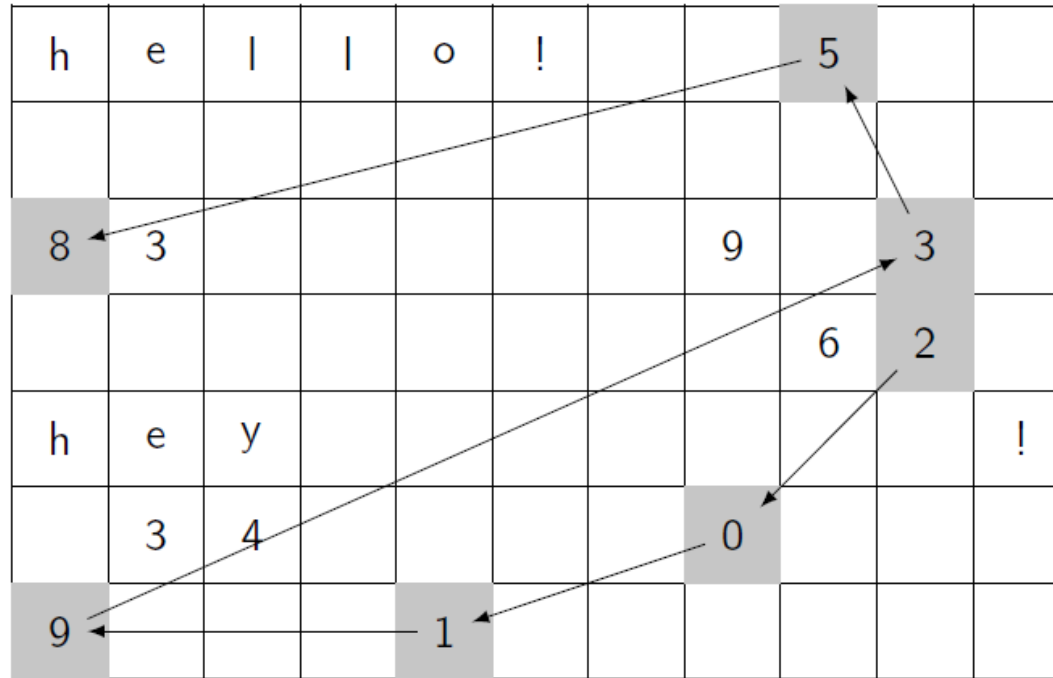
- On utilise des pointeurs pour chainer les éléments successifs
- La liste est déterminée par l'adresse de son premier élément



- Remarques :
  - Cela nécessite de la place supplémentaire en mémoire
  - Il n'y a pas de taille maximum pour la liste
  - La suppression ou l'insertion sont aisées car ces opérations ne nécessitent pas d'autres déplacements en mémoire.
- Un site assez complet :
  - <https://www.codeheroes.fr/2021/01/04/reussir-son-entretien-technique-les-listes-chainees/>

# Liste chaînée

- On accède à l'élément d'indice  $n$  en temps linéaire.

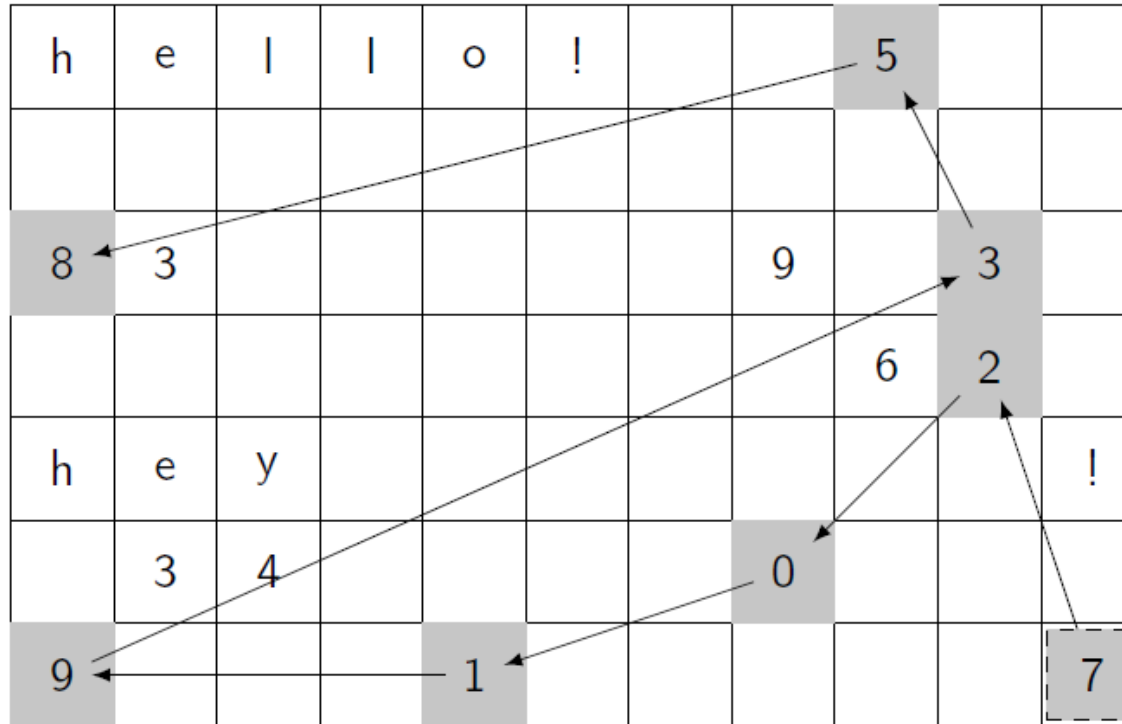


Python Tutor



# Liste chaînée

- On ajoute un élément en tête de liste en temps constant.



Python Tutor



# Le type list python

- Python est un langage de haut-niveau.
- Le type list essaie de prendre les avantages des deux structures précédentes.
- Le type list va plus loin : les éléments peuvent être de types différents.



# Les piles et les files

- Pour beaucoup d'applications les seules opérations à effectuer sur les listes sont des insertions et des suppressions aux extrémités.
- Piles (LIFO): les insertions et les suppressions se font à une seule extrémité de la pile appelée sommet.
- Files (FIFO): On fait les insertions à une extrémité appelée « queue » les accès et les suppressions à l'autre extrémité, appelée « tête »

# Le type abstrait Pile

Opérations	Signature	Description
pile_vide	$\rightarrow \text{pile}$	<i>Définir la pile vide</i>
empiler	$\text{pile} \times \text{élément} \rightarrow \text{pile}$	<i>Ajouter un élément à la pile; l'élément ajouté se trouvera (après ajout) au sommet de la pile</i>
depiler	$\text{pile} \rightarrow \text{pile}$	<i>Supprimer l'élément au sommet de la pile</i>
sommet	$\text{pile} \rightarrow \text{élément}$	<i>Accéder au sommet de la pile</i>
est-elle vide ?	$\text{pile} \rightarrow \text{booléen}$	<i>Tester si une pile est vide</i>

# Le type abstrait File

Opérations	Signature	Description
file_vide	$\rightarrow \text{file}$	<i>Définir la file vide</i>
enfiler	$\text{file} \times \text{élément} \rightarrow \text{file}$	<i>Ajouter un élément à la file; l'élément ajouté se trouvera (après ajout) en queue de la file</i>
defiler	$\text{file} \rightarrow \text{file}$	<i>Supprimer l'élément en tête de la file</i>
premier	$\text{file} \rightarrow \text{élément}$	<i>Accéder à l'élément en tête de la file (présent depuis le plus longtemps)</i>
est-elle vide ?	$\text{file} \rightarrow \text{booléen}$	<i>Tester si une file est vide</i>



# Les dictionnaires

- Dans les listes, il y a un ordre sur les places et les éléments sont donc ordonnés.
- Dans un dictionnaire, la place d'un élément est calculée uniquement à partir de sa clé.
- Ce calcul est réalisé grâce à une fonction dite fonction de hachage (hash function)
- Cette méthode a la particularité de rendre la complexité des opérations de recherche, d'ajout ou de suppression en moyenne constante.

# Le type abstrait dictionnaire

Opérations	Signature	Description
dictionnaire_vide	$\rightarrow$ dictionnaire	<i>Définir le dictionnaire vide</i>
accéder	dictionnaire x clé $\rightarrow$ élément	<i>Accéder à un élément du dictionnaire; l'élément se trouve à une position définie par sa clé</i>
modifier	dictionnaire x clé x élément $\rightarrow$ dictionnaire	<i>Modifier un couple clé:valeur en remplaçant la valeur courante par une autre valeur (la clé restant identique)</i>
supprimer	dictionnaire x clé $\rightarrow$ dictionnaire	Supprimer une clé (et donc la valeur qui lui est associée)
ajouter	dictionnaire x clé x élément $\rightarrow$ dictionnaire	<i>Associer une nouvelle valeur à une nouvelle clé</i>



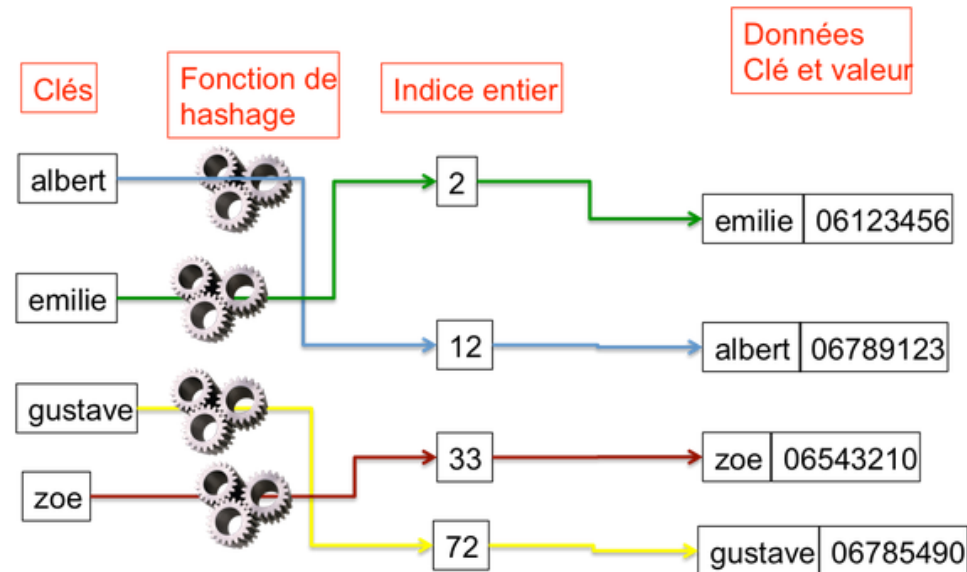
# Fonction de hachage

- Nous cherchons à indexer un ensemble de valeurs (e.g les numéros de téléphones) par un ensemble de clés (e.g les noms).
- Ceci n'est bien sûr pas possible avec un tableau dont les indices sont obligatoirement des entiers (afin d'être capable de calculer l'adresse d'un élément).



# Fonction de hachage

- Le premier principe essentiel des tables de hachage est de transformer une clé en un entier, qui sera alors l'indice du tableau contenant les données. C'est le rôle de la fonction de hachage, qui assure une correspondance entre notre clé et l'indice entier du tableau.



# Table d'association

- Ajouter un couple clé, valeur c'est
  - Calculer l'indice que ce couple occupera dans la table, c'est à dire exécuter la fonction de hachage pour récupérer la valeur entière qu'elle retourne ;
  - Mettre ce couple [clé, valeur] dans la table à l'indice qui vient d'être calculé

La table initiale est vide

	Clé	Valeur
0		
1		
...		
6		
...		
31		
...		
99		



# Table d'association

- Rechercher un couple clé, valeur c'est
  - Calculer l'indice que ce couple occupe s'il existe dans la table, à l'aide de la fonction de hachage ;
  - S'ils existent, la clé et la valeur sont dans la table à l'indice qui vient d'être calculé.

on calcule l'indice de hashage

$$\text{hash}(\text{"bac"}) = (2+1+3) \% 100 = 6$$

	Clé	Valeur
0		
...		
6	bac	examen passé en terminale
...		
31	lame	partie coupante d'un couteau
...		
42		
...		
99		

La case d'index **6** de la table n'est pas vide et contient la clé *bac* et sa *définition*

on calcule l'indice de hashage

$$\text{hash}(\text{"trac"}) = (20+18+1+3) \% 100 = 42$$

	Clé	Valeur
0		
...		
6	bac	examen passé en terminale
...		
31	lame	partie coupante d'un couteau
...		
42		
...		
99		

La case d'index **42** de la table est vide et la table ne contient aucune information sur le mot *trac*



WIKIPÉDIA  
L'encyclopédie libre

[https://fr.wikipedia.org/wiki/Fonction\\_de\\_hachage](https://fr.wikipedia.org/wiki/Fonction_de_hachage)



<https://interstices.info/le-hachage/>

# Parcours de listes et dictionnaires

- Difficultés rencontrées par les élèves

```
1 # confusion indice / élément
2 tab = [3, 5, 7]
3 for i in tab:
4     print(tab[i])
5
```



Python Tutor

IndexError: list index out of range

```
1 # indice dans un dictionnaire
2 dico = {"a":0, "b":1, "c":2}
3 for i in range(len(dico)):
4     print(dico[i])
5
```



Python Tutor

KeyError: 0

# Tableau parcours par index

Pour variable entière variant de valeur de début à valeur de fin par pas de taille faire ...

```
1 # Parcours par index
2 tab = [3, 5, 7]
3 for i in range(0, len(tab), 1):
4     print(tab[i])
5
```



Python Tutor

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         int[] tab = {3, 5, 7};
5         for (int i=0; i<tab.length; i=i+1){
6             System.out.println(tab[i]);
7         }
8     }
9 }
10
```



Java Tutor

# Tableau parcours par éléments

Pour chaque élément dans le tableau ...

```

1 # Parcours par éléments
2 tab = [3, 5, 7]
3 for e in tab:
4     print(e)
5

```



Python Tutor

```

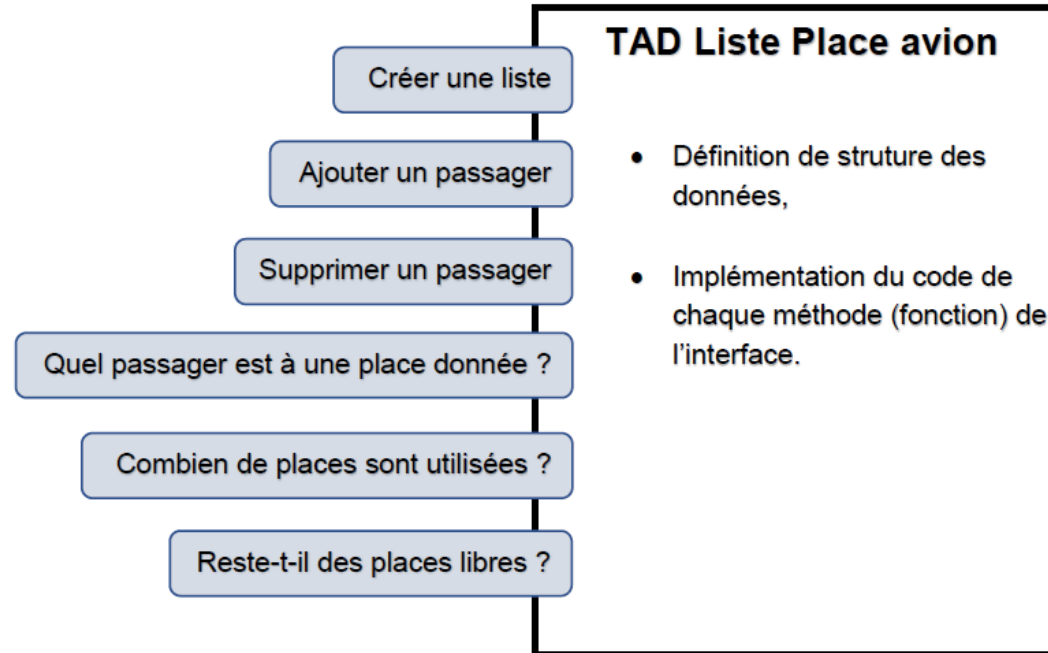
1 public class Main
2 {
3     public static void main(String[] args) {
4         int[] tab = {3, 5, 7};
5         for (int e:tab){
6             System.out.println(e);
7         }
8     }
9 }
10

```



Java Tutor

# Exemple de fiche élève



• Fiche de travail [marc.eldin@ac-bordeaux.fr](mailto:marc.eldin@ac-bordeaux.fr)

# Dictionnaire parcours par clé

Pour chaque clé du dictionnaire faire ...

```
1 # clé dans un dictionnaire
2 dico = {"a":0, "b":1, "c":2}
3 for k in dico.keys():
4     print(dico[k])
5
```



Python Tutor

```
1 import java.util.*;
2
3 public class Main
4 {
5     public static void main(String[] args) {
6         Hashtable dic=new Hashtable();
7         dic.put("a",0);
8         dic.put("b",1);
9         dic.put("c",2);
10        Set<String> setOfKeys = dic.keySet();
11        for (String key: setOfKeys){
12            System.out.println(dic.get(key));
13        }
14    }
15 }
16
```



Java Tutor





# Dictionnaire parcours par éléments

Pour chaque élément du dictionnaire faire ...

```
1 # éléments dans un dictionnaire
2 dico = {"a":0, "b":1, "c":2}
3 for e in dico.values():
4     print(e)
5
```



Python Tutor

```
1 import java.util.*;
2 import java.util.Map.Entry;
3
4 public class Main
5 {
6     public static void main(String[] args) {
7         Hashtable dic=new Hashtable();
8         dic.put("a",0);
9         dic.put("b",1);
10        dic.put("c",2);
11        Set<Entry<String, Integer> > entrySet = dic.entrySet();
12        for (Entry<String, Integer> entry : entrySet){
13            System.out.println(entry.getValue());
14        }
15    }
16 }
17
```



Java Tutor

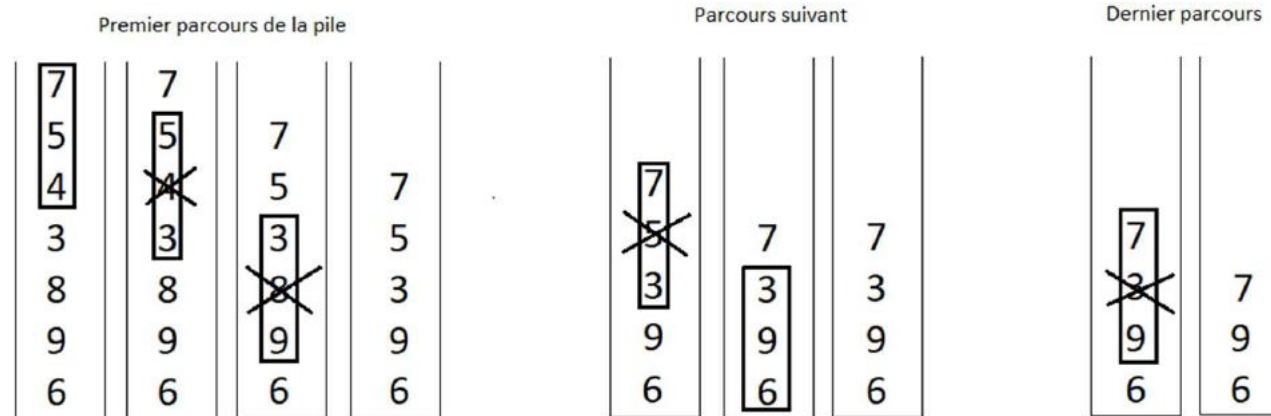


# Muable ou immuable

- Baccalauréat 22\_NSIJ2ME1

La poussette est un jeu de cartes en solitaire. Cet exercice propose une version simplifiée de ce jeu basée sur des nombres.

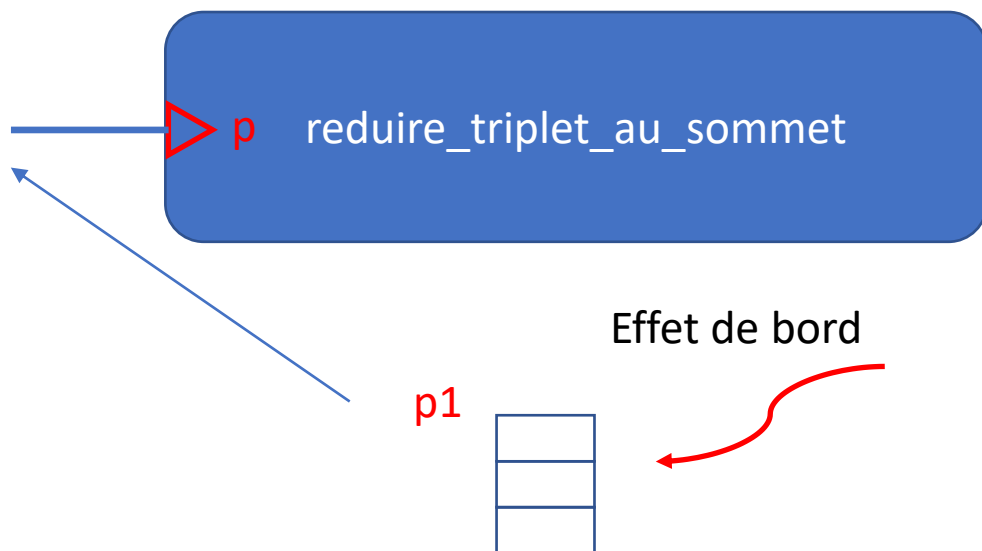
On considère une pile constituée de nombres entiers tirés aléatoirement. Le jeu consiste à réduire la pile suivant la règle suivante : quand la pile contient du haut vers le bas un triplet dont les termes du haut et du bas sont de même parité, on supprime l'élément central.



# Muable ou immuable

- Baccalauréat

- « Recopier et compléter sur la copie le code de la fonction `reduire_triplet_au_sommet` prenant une pile `p` en paramètre et qui la modifie en place. Cette fonction ne renvoie donc rien. »



```
def reduire_triplet_au_sommet(p):
    """Reduction de la pile p
```

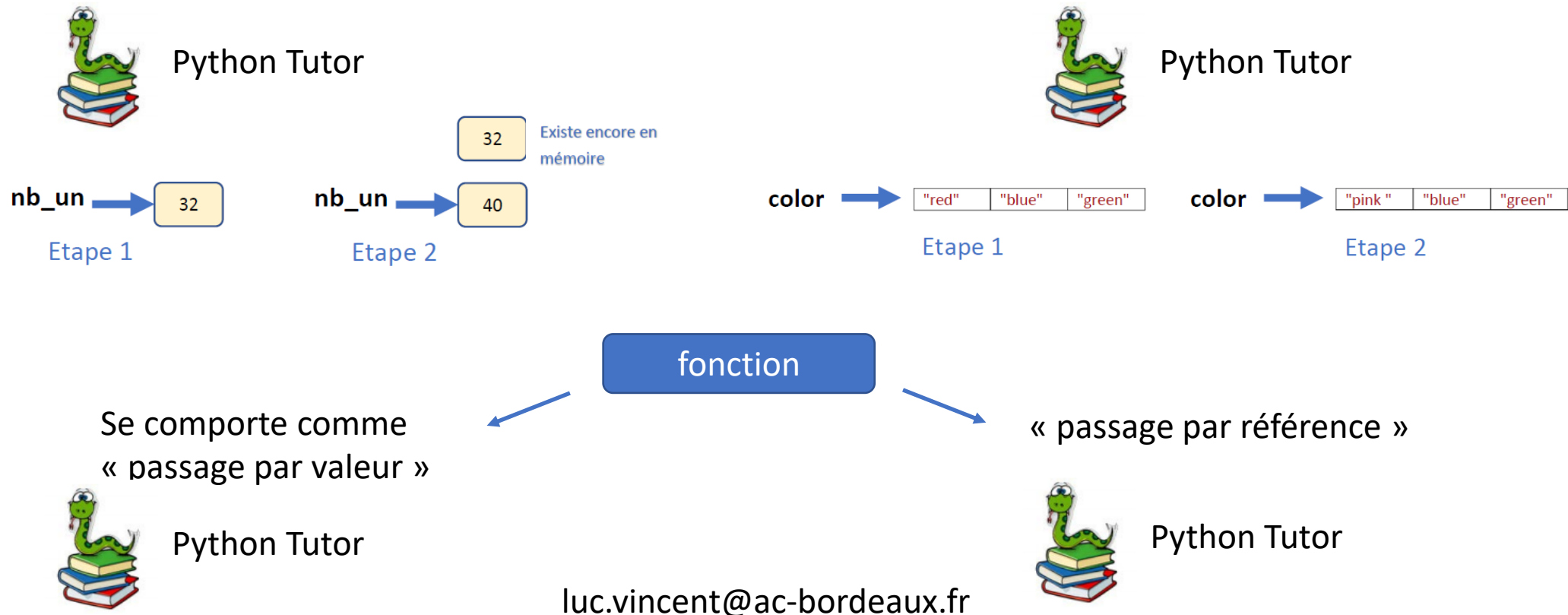
```
    permet de supprimer l'élément central des trois premiers éléments
    en partant du haut de la pile,
    si l'élément du bas et du haut sont de même parité.
    Les éléments dépilés et non supprimés sont remplacés
    dans le bon ordre dans la pile.
    """
```

```
    # la pile p étant une liste c'est un objet mutable
    # Cette fonction réduit le sommet de la pile en place
    a = depiler(p)
    b = depiler(p)
    c = sommet(p)
    if a % 2 != c % 2:
        empiler(p, b)
        empiler(p, a)
```

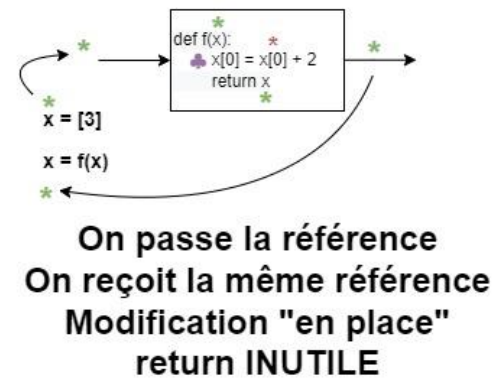
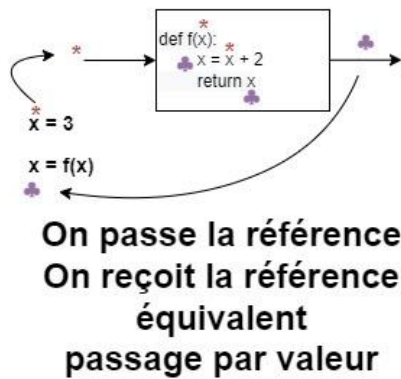
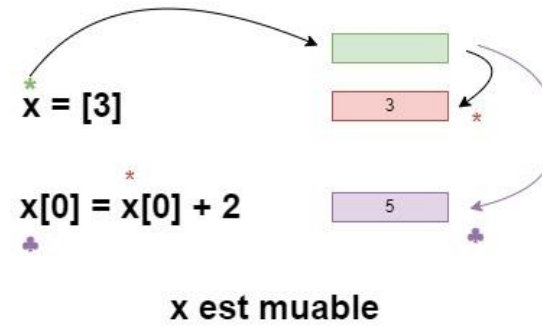
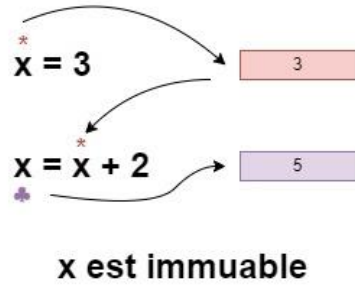


# Muable ou immuable

- « An immutable object is an object whose value cannot change. »
- « In Python, 'mutable' is the ability of objects to change their values »



# Muable ou immuable



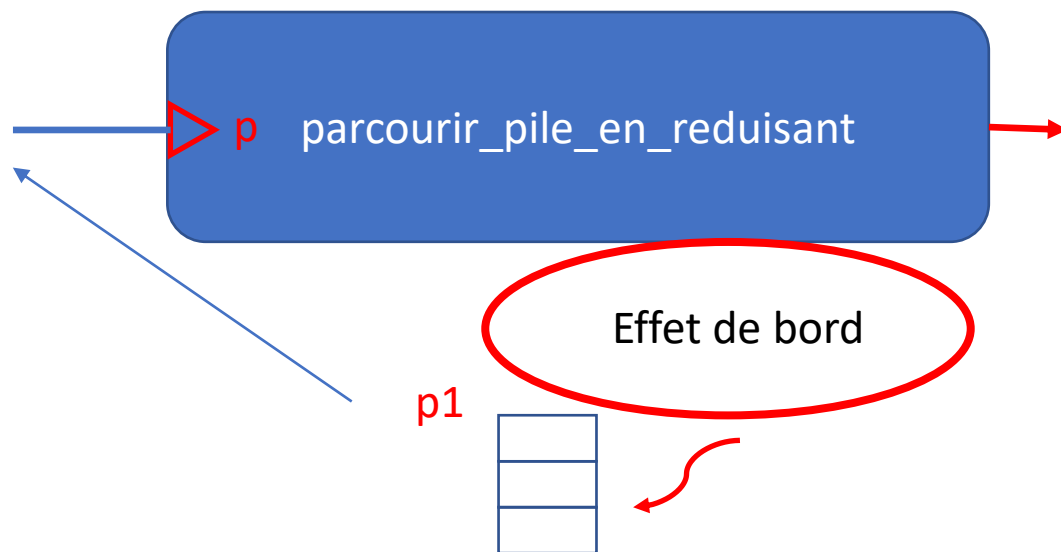
- Fiche résumée



# Muable ou immuable

- Baccalauréat

- «On se propose maintenant d'écrire une fonction parcourir\_pile\_en\_reduisant qui parcourt la pile du haut vers le bas en procédant aux réductions pour chaque triplet rencontré quand cela est possible.»



```

1  def parcourir_pile_en_reduisant(p):
2      q = creer_pile_vide()
3      while taille(p) >= ....:
4          reduire_triplet_au_sommet(p)
5          e = depiler(p)
6          empiler(q, e)
7      while not est_vide(q):
8          .....
9          .....
10     return p

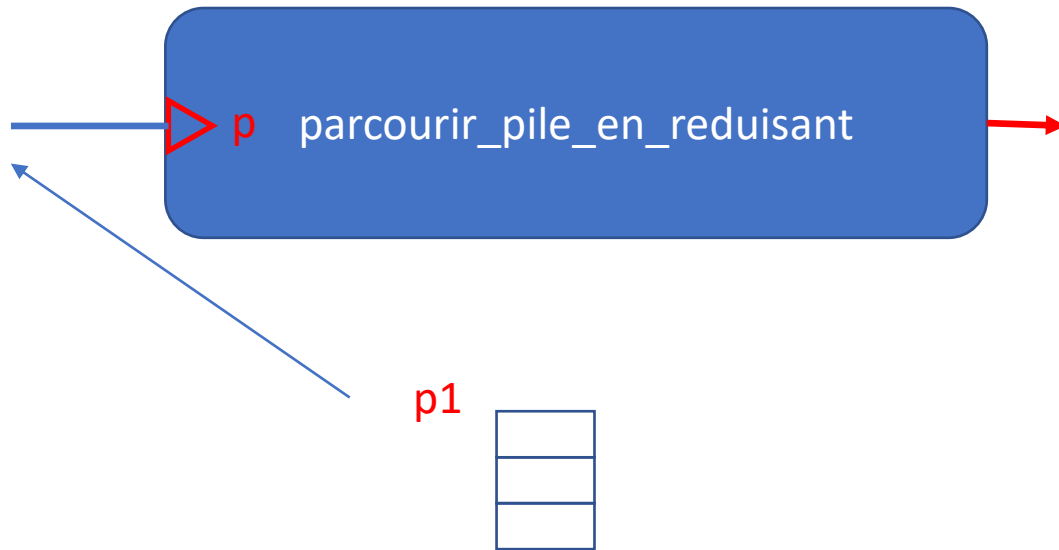
```



# Muable ou immuable

- Bonne méthode

- Dans un projet ne JAMAIS mélanger les modifications en place et les fonctions pures (sans effet de bord)
- Dans le cas d'objet mutable faire une copie de l'objet pour éviter les effets de bord !



```
def reduire_triplet_au_sommet(p):
    """Reduction de la pile p
    permet de supprimer l'élément central des trois premiers éléments
    en partant du haut de la pile,
    si l'élément du bas et du haut sont de même parité.
    Les éléments dépilés et non supprimés sont
    replacés dans le bon ordre dans la pile.
    """

    # la pile p étant une liste c'est un objet mutable
    # On commence par en faire une copie (superficielle)
    # p = [p[i] for i in range(len(p))]
    # p = list(p)
    p = p.copy()
    a = depiler(p)
    b = depiler(p)
    c = sommet(p)
    if a % 2 != c % 2:
        empiler(p, b)
        empiler(p, a)
    return p
```

# Muable ou immuable

- Procédure uniquement
  - Variable globale connue de chaque procédure



une\_pile



reduire\_triplet\_au\_sommet()

parcourir\_pile\_en\_reduisant()

jouer()

affichage()

```
# une_pile est une variable globale connue de toutes les fonctions
une_pile = creer_pile_vide()
poussette = (6, 9, 8, 3, 4, 5, 7)
for value in poussette:
    empiler(une_pile, value)
jouer()
affichage()
```

```
86 def jouer():
87     """Simplifier entièrement la pile"""
88     # la pile p étant une liste c'est un objet mutable
89     # Cette fonction simplifie en place la pile une_pile
90     # la variable est globale "une_pile"
91     taille_p_ar = taille(une_pile) # taille_p_ar avant reduction
92     parcourir_pile_en_reduisant()
93     if taille(une_pile) != taille_p_ar:
94         jouer()
95
```

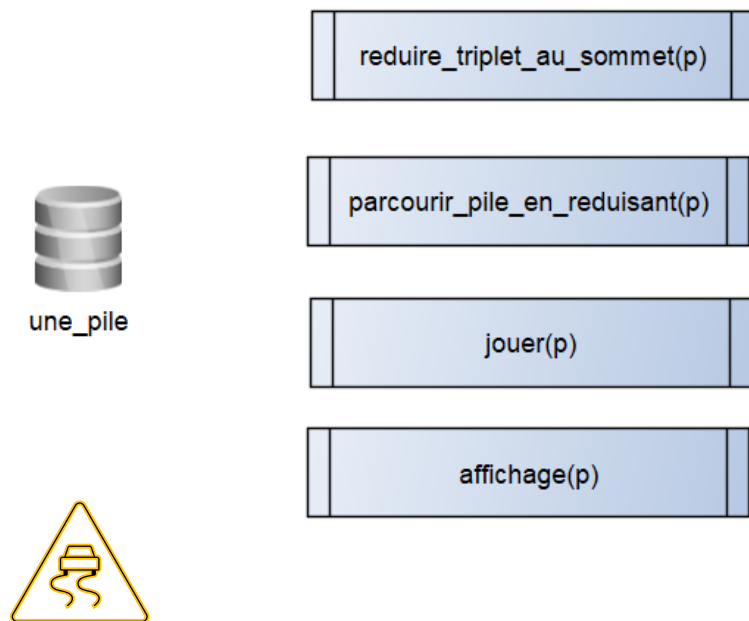


Exercice\_2\_E\_pile\_avec\_effetDeBord\_varGlobale



# Muable ou immuable

- Procédure avec effet de bord
  - Référence passée comme argument
  - Renvoi par effet de bord et sans « return »



```
# une_pile est passée par référence
# à chaque fonction
une_pile = creer_pile_vide()
poussette = (6, 9, 8, 3, 4, 5, 7)
for value in poussette:
    empiler(une_pile, value)
jouer(une_pile)
affichage(une_pile)
```

```
def jouer(p):
    """Simplifier entièrement la pile"""
    # la pile p étant une liste c'est un objet mutable
    # Cette fonction simplifie la pile en place
    taille_p_ar = taille(p) # taille_p_ar avant reduction
    parcourir_pile_en_reduisant(p)
    if taille(p) != taille_p_ar:
        jouer(p)
```

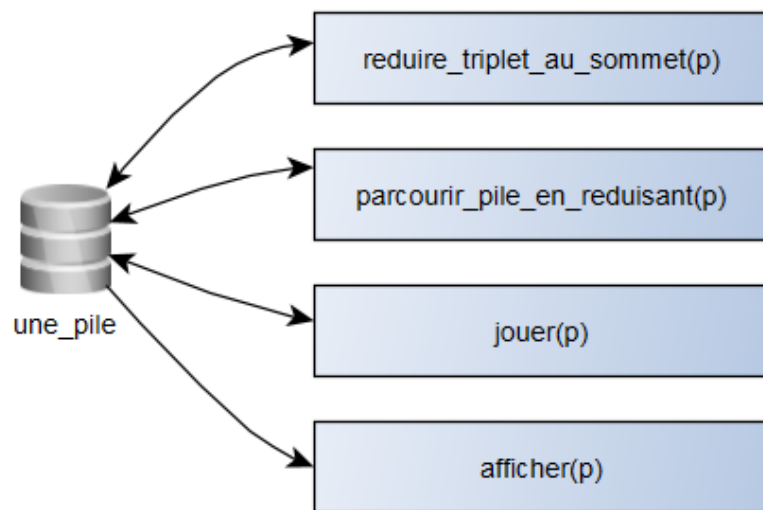


Exercice\_2\_C\_avec\_EffetDeBord\_AvecArgument

# Muable ou immuable

- Fonction avec effet de bord

- Référence passée comme argument
- Renvoi par effet de bord et avec « return »

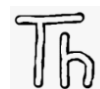


```

poussette = (1, 6, 7, 8, 4, 3)
une_pile = creer_pile_vide()
for value in poussette:
    empiler(une_pile, value)
une_pile_j = jouer(une_pile) # fonction
affichage(une_pile_j)
  
```

```

def jouer(p):
    """Renvoyer une pile entièrement simplifiée"""
    q = parcourir_pile_en_reduisant(p) # fonction
    if q == p :
        return q
    else:
        return jouer(q)
  
```

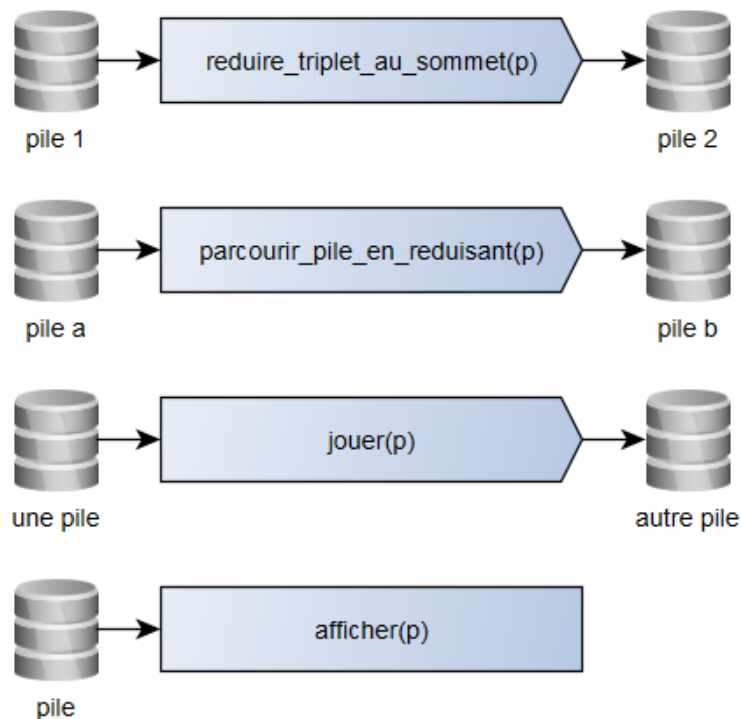


Exercice\_2\_F\_pile\_avec\_effetDeBord\_AvecArgument

# Muable ou immuable

- Fonction pure

- Référence passée comme argument
- Renvoi d'un nouvel objet sans modifier l'objet initial



```

poussette = (6, 9, 8, 3, 4, 5, 7)
une_pile = creer_pile_vide()
for value in poussette:
    empiler(une_pile, value)
une_pile_j = jouer(une_pile) # fonction
affichage(une_pile_j)
  
```

```

def jouer(p):
    """Renvoyer une pile entièrement simplifiée"""
    q = parcourir_pile_en_reduisant(p) # fonction
    if q == p :
        return q
    else:
        return jouer(q)
  
```



*Exercice\_2\_D\_pile\_avec\_effetDeBord\_AvecArgument*