L3 Informatique - 2023-2024

Atelier 4 – Types Structurés Python

Université di Corsica Pasquale Paoli

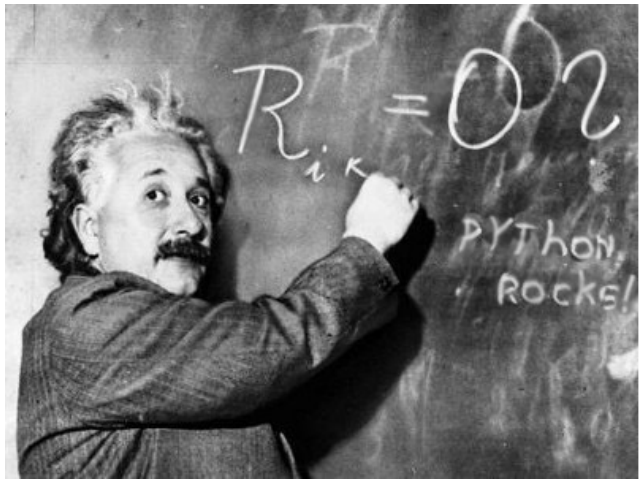
Paul-Antoine BISGAMBIGLIA – bisgambiglia_pa@univ-corse.fr
Marie-Laure NIVET – nivet_m@univ-corse.fr
Evelyne VITTORI - vittori_e@univ-corse.fr

Types structurés en python
Objectifs du cours

- Faire un bilan des différentes caractéristiques des conteneurs python et de leurs spécificités (listes, tuples, dictionnaires)
- Introduire la notion de graphe

Types structurés en python

1. Caractéristiques des conteneurs python
   - Indexable, itérable, immutable/mutable
   - transmission de paramètres
2. Listes Python
   - Cas particulier: Chaines de caractères
   - Listes de listes
3. Tuples
   - Listes de tuples
4. Dictionnaires
5. Tableaux numpy
6. Application à la représentation des graphes



POWERFUL AND OPEN TOOLS FOR SCIENTISTS.

1 - Types structurés en python : les CONTENEURS

- Notion de conteneur
- Conteneurs indexables
- Conteneurs itérables
- Conteneurs mutables/immutables
- Conteneurs et transmission de paramètres

Notion de conteneur en python

- Les conteneurs python permettent de créer des variables structurées de différents types.
- On distingue 4 principaux conteneurs:
  - Liste
    - Les chaines de caractères sont un cas particulier de liste
  - Tuple
  - Dictionnaire
  - *Ensemble (Set)*

Principaux conteneurs en python

- Listes : séries d'éléments (de types éventuellement différents) séparés par des virgules et encadrés de crochets

  l=[1,2,3,4]

  m=[1,2, 'fin']

  Les chaines de caractères (type str) sont des listes de caractères non modifiables (immutable)

- Tuples :suites d'éléments quelconques

  t=(1, (2,3), 4, (5,6,7), 8, 'fin')

Principaux conteneurs en python

- Dictionnaires : tableaux associatifs, ensembles de couples {clé:valeur, …}

  d={1:'a', 2:'b', 3:'abc', 'x': 'xyz'}

  Recherche d'un élément à partir de sa clé très rapide

- Ensembles (sets) : collections d'objets non ordonnés sans répétition

  e={1, 2, 8, 12, 3}

Conteneurs indexables

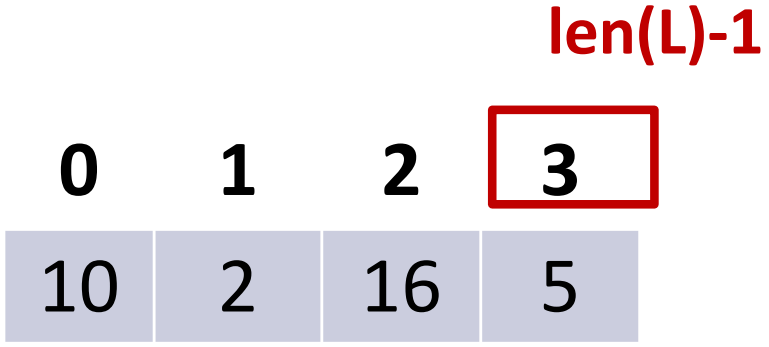
- **Listes**, Chaines, **Tuples**

Eléments numérotés

- len(L) # nombre d'éléments du conteneur

- Eléments numérotés de 0 à len(L)-1
  - L[n] # élément d'indice n
  - L[-1] # dernier élément

L[-1]=L[len(L)-1]



Manipulation des conteneurs indexables

- Sous-listes (tuple ou chaine) : slices

  L[n:m] #sous liste (tuple ou chaine) contenant les éléments de L d'indice n à m-1

  L[n:] # .. éléments de L d'indice n à la fin

  L[:m] # .. éléments de L d'indice 0 à m-1

Exemple

L=[0,1,2,3,4,5]

L[0:2]	[0, 1]
L[:2]	[0, 1]
L[2:]	[2, 3, 4, 5]
L[2:-1]	[2, 3, 4]
L[0:-1]	[0, 1, 2, 3, 4]
L[:-1]	[0,1, 2, 3, 4]
L[0:]	[0,1, 2, 3, 4, 5]

Liste entière

Manipulation des conteneurs indexables

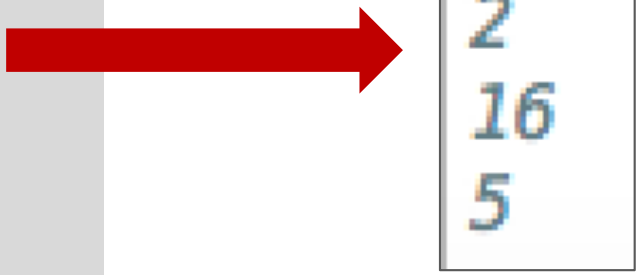
- Peuvent être parcourus par une boucle de la forme

```
for i in range(deb, fin, pas) :
        instructions
```

i est un indice

- itération de deb à fin-1 (si i>0)
- itération de fin à deb+1 (si i<0)

```
#Exemple d'une liste
L=[10,2,16, 5]
for i in range(0,len(L)):
        print(L[i])
```



```
10
2
16
5
```

Conteneurs itérables

- Tous les conteneurs sauf **les tuples** peuvent être parcourus par une boucle de la forme

v est un élément du conteneur

```
for v in conteneur :
        instructions
```

Exemple d'une liste

```
L=[10,2,16, 5]
for v in L :
        print(v)
```

```
10
2
16
5
```

Conteneurs Mutables/Immutables

- Mutables (Modifiables) : les valeurs stockées dans l'objet conteneur peuvent être modifiées
  - listes, dictionnaires, ensembles

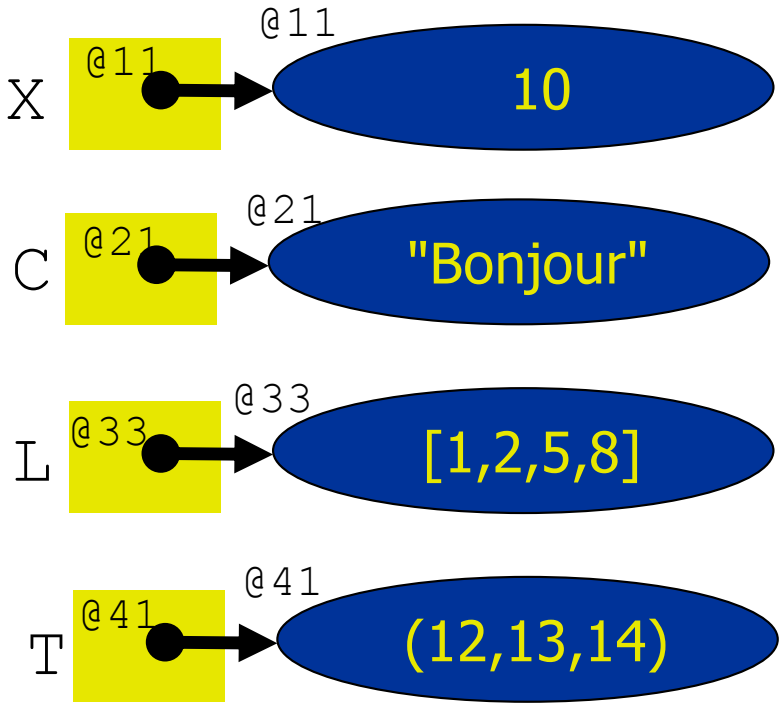
- Immutables (Non Modifiales) : l'objet conteneur ne peut pas être modifié
  - tuples
  - chaines de caractères

Remarque= en python les variables définies sur des types de base (entier, réel, booléen, ...) sont aussi des objets immutables

Notion d'objet

- Toutes les variables en python sont des objets
- Elle ne contiennent pas une valeur mais contiennent une référence (adresse mémoire) vers la valeur de l'objet

```
X=10 #variable de type entier
C = "Bonjour" #variable de type chaine
L=[1,2,5,8] #variable de type liste
T=(12,13,14) #variable de type tuple
```

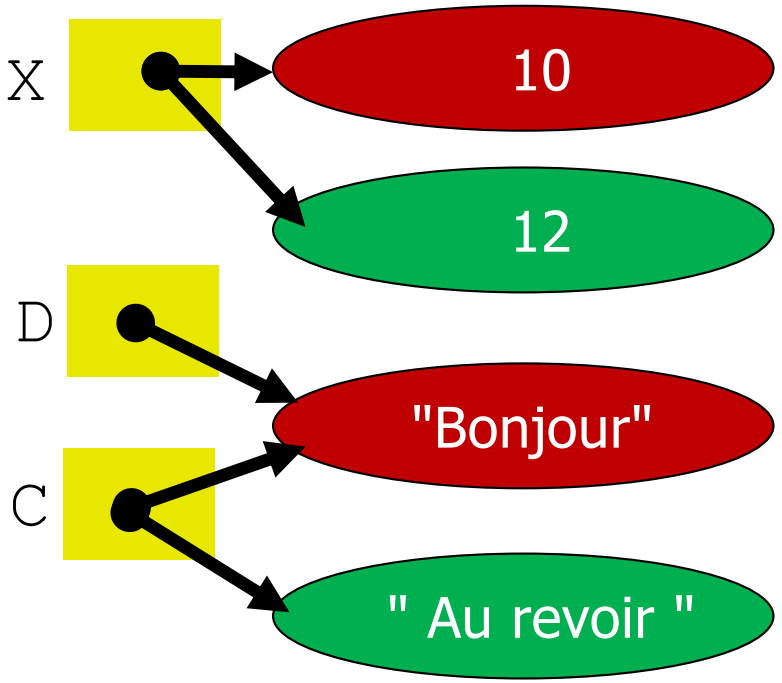


Notion d'objets immutables

- Certains objets sont immutables: leurs valeurs ne peuvent pas être modifiées
  - Types de base: entier, réels, booléens
  - Chaines
  - Tuples

```
X=10 #variable de type entier
X=12

C = "Bonjour" #variable de type chaine
D=C
C= "Au revoir"
```

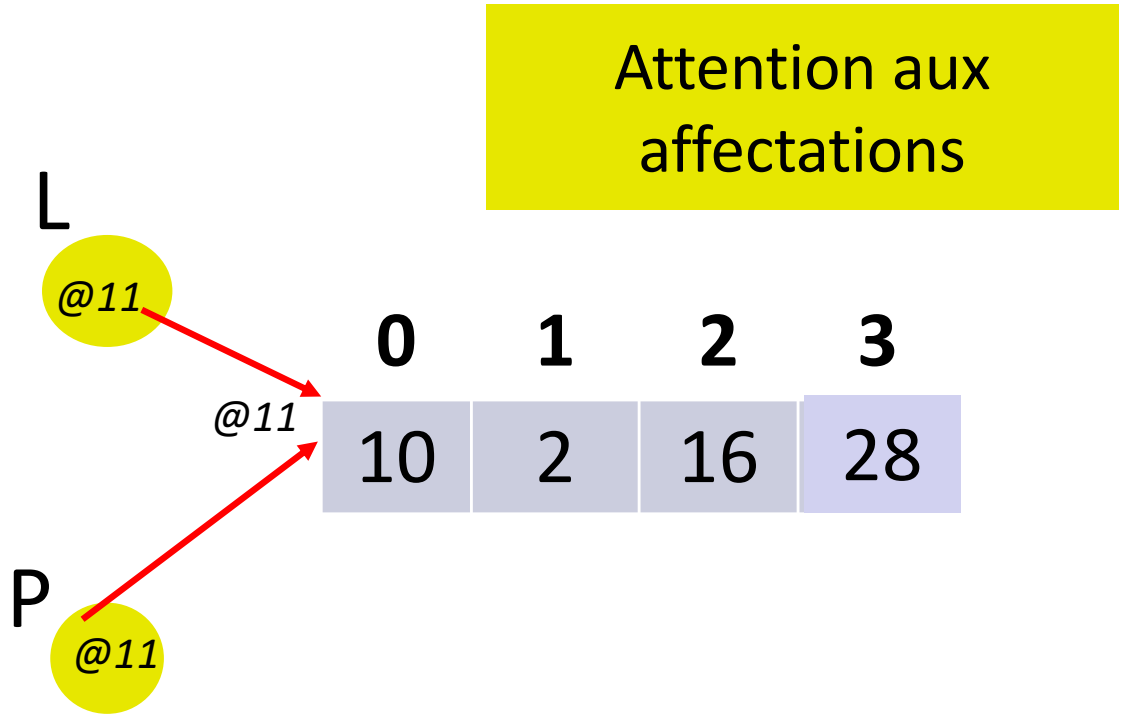


Un conteneur est un « objet »

- Une variable de type conteneur contient une **référence** vers l'objet conteneur.

Exemple d'une liste

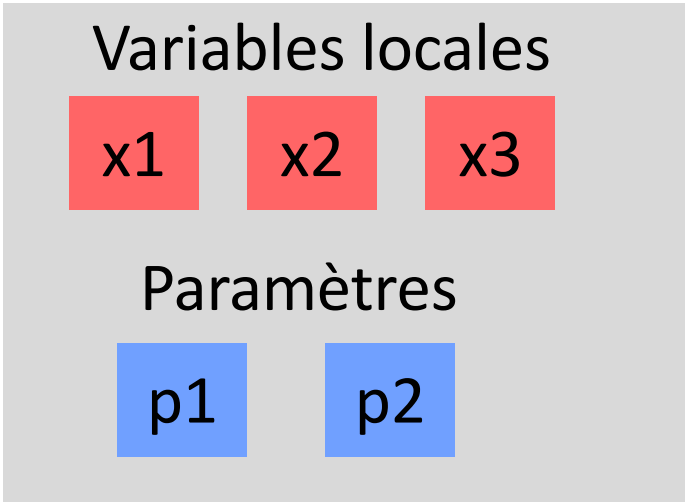
```
>>>L=[10,2,16, 5]
>>> P=L
>>> P[1]
2
>>> P[3]=28
>>> P
[10, 2, 16, 28]
>>> L
[10, 2, 16, 28]
```



Rappel: Principes de transmission des paramètres à une fonction

Lorsque une fonction est appelée:

1. Une zone mémoire est allouée (empilée) pour
   - Ses variables locales
   - Ses paramètres
2. Ses paramètres sont initialisés en fonction des paramètres effectifs utilisés dans l'appel
3. La fonction s'exécute

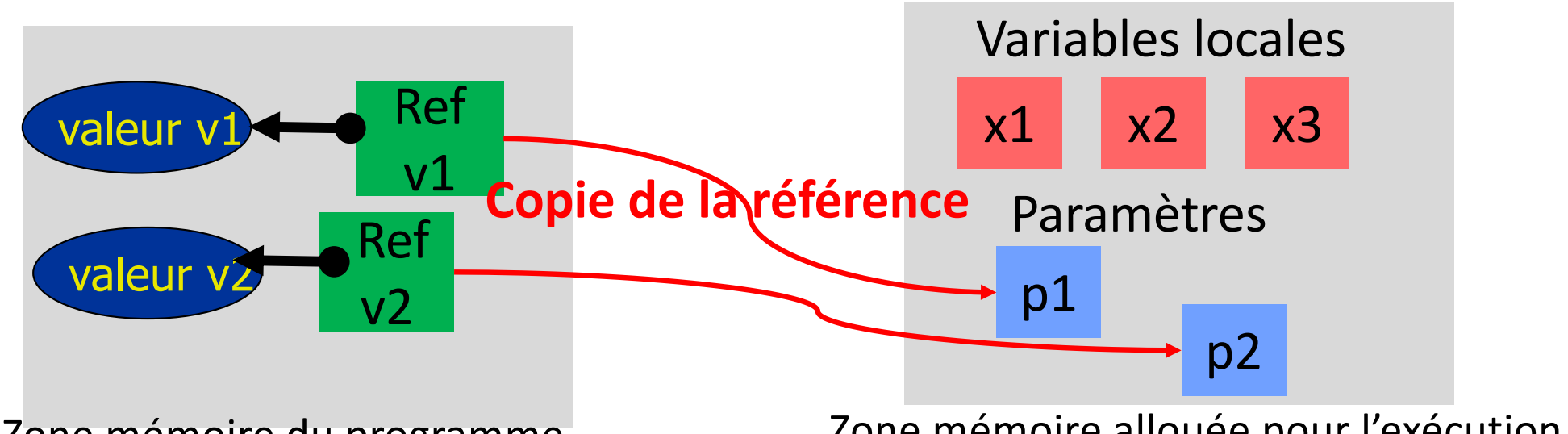


Zone mémoire allouée pour l'exécution d'une fonction f(v1,v2)

Mise en place de la Transmission des paramètres

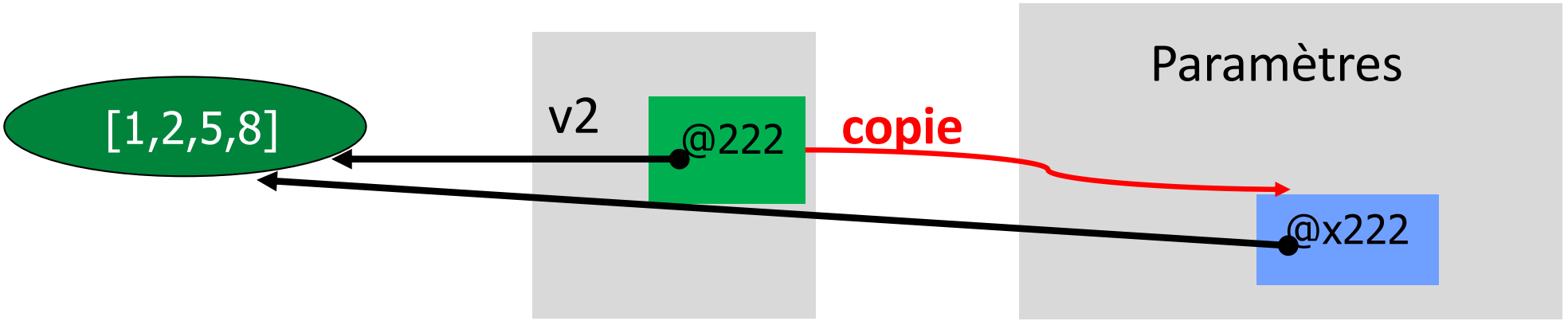
Rappels: Principes de transmission des paramètres à une fonction

- En python, la transmission se fait « **par valeur (copie)»** mais comme toutes toutes les variables contiennent des références cela revient à une copie de références :
  - Les références contenues dans les paramètres effectifs (utilisés dans l'appel) sont copiés dans les paramètres de la zone mémoire de la méthode



Rappels: Principes de transmission des paramètres à une fonction

- Une fonction est dite à « effet de bord » si elle effectue une modification de ses paramètres qui a des répercussions sur le programme appelant
  - Modification de la valeur d'un paramètre mutable



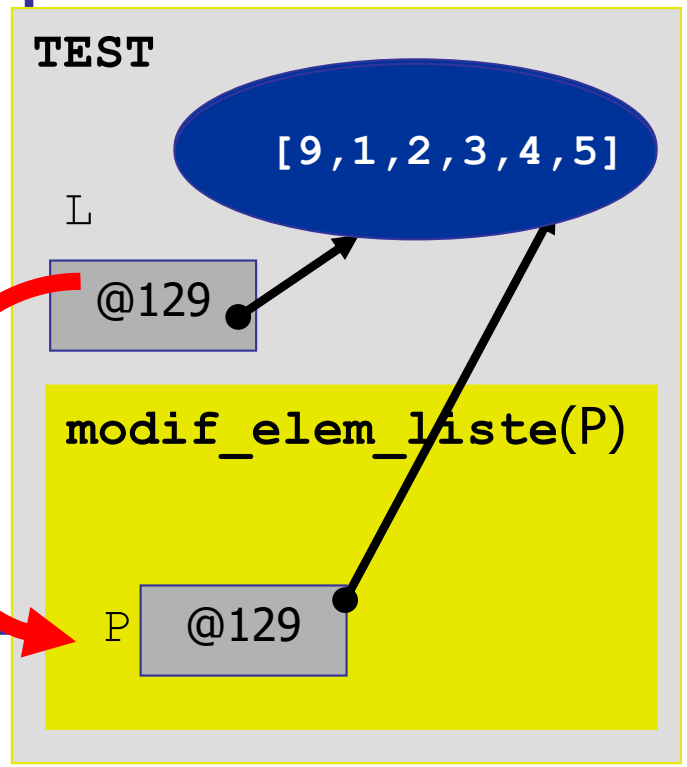
Fonctions en python

Passage d'un paramètre de type liste

Effet de bord

```
def modif_elem_liste(P) :
    #modifie le 1er élément d'une liste
    if len(P)!=0 :
        P[0] = 9

L=[0,1,2,3,4,5]
print("Avant appel L= " +str(L))
modif_elem_liste(L)
print("Apres appel L= " +str(L))
```



Avant appel L= [0, 1, 2, 3, 4, 5]
Apres appel L= [9, 1, 2, 3, 4, 5]

Fonctions en python

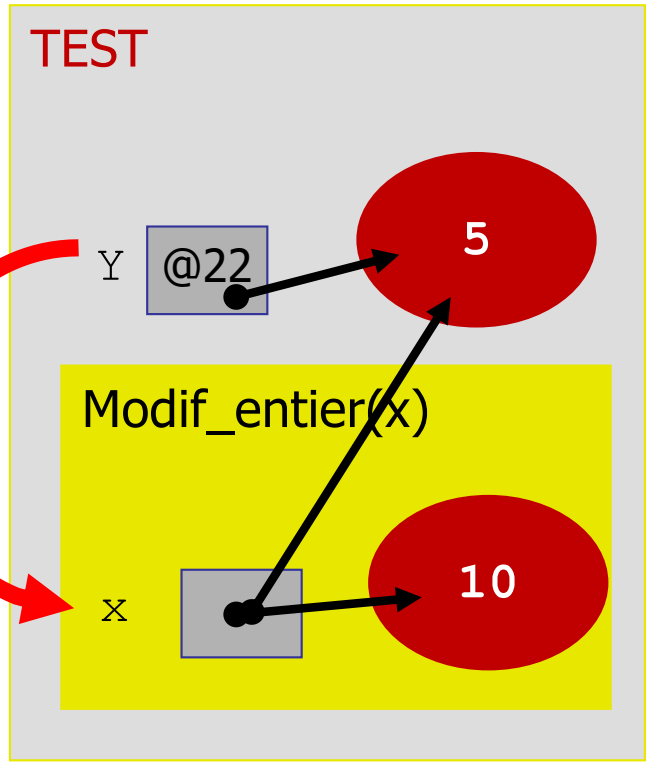
Passage d'un paramètre de type entier

Absence d'effet de bord

```
def modif_entier(X) :
    X=10

#TEST
Y=5
print("Avant appel Y= " +str(Y))
modif_entier(Y)
print("Aprés appel Y= " +str(Y))
}
```

Copie



Aucun effet de bord possible pour les objets immutables

Avant appel Y= 5
Apres appel Y= 5

Paramètres de type mutables et effets de bord

- Si un nouvel objet est affecté au paramètre, il n'y aura pas d'effet de bord.

```
def modif_liste(L) :
    #creation d'une nouvelle liste
    L=[1000,2000]
#TEST
L=[0,1,2,3,4,5]
print("Avant appel L= " +str(L))
modif_liste(L)
print("Aprés appel L= " +str(L))
```

Aucun effet de bord car la nouvelle liste est créée dans l'environnement de la fonction

```
Avant appel L= [0, 1, 2, 3, 4, 5]
Apres appel L= [0, 1, 2, 3, 4, 5]
```

Conseils

- Attention aux effets de bord non maitrisés:
  - Ex: on modifie une liste sans le vouloir!
  - Si l'on ne veut pas qu'un paramètre de type liste soit modifié, travailler sur une copie locale.

Exercice

- Quel affichage provoque l'exécution de ce programme?

```
def modif_listes(L1,L2) :
    res=[]
    if len(L1)!=0 :
        L1[0] = 1000
    L2=L1
    res=L1
    L1=[]
    return res
#TEST
L=[0,1,2,3,4,5]
P=[2,3]
print("Avant appel L= " +str(L))
print("Avant appel P= " +str(P))
R=modif_listes(L,P)
print("Aprés appel L= " +str(L))
print("Aprés appel P= "+str(P))
print("R = "+str(R))
```

Avant appel L= [0, 1, 2, 3, 4, 5]
Avant appel P= [2, 3]
Aprés appel L= [1000, 1, 2, 3, 4, 5]
Aprés appel P= [2, 3]
R = [1000, 1, 2, 3, 4, 5]

Exercice

- Quel affichage provoque l'exécution de ce programme?

```
def modif_var(X,C) :
    X=X+10
    C=C+" suite"
    return X,C

#TEST
E=5
M="bonjour"
print("Avant appel E= " +str(E))
print("Avant appel M= "+str(M))
ES,MS =modif_var(E,M)
Print("Aprés appel E= " +str(E))
print("Aprés appel M= "+str(M))
print("ES= " +str(ES))
print("MS= " +str(MS))
```

Avant appel E= 5
Avant appel M= bonjour
Aprés appel E= 5
Aprés appel M= bonjour
ES= 15
MS= bonjour suite

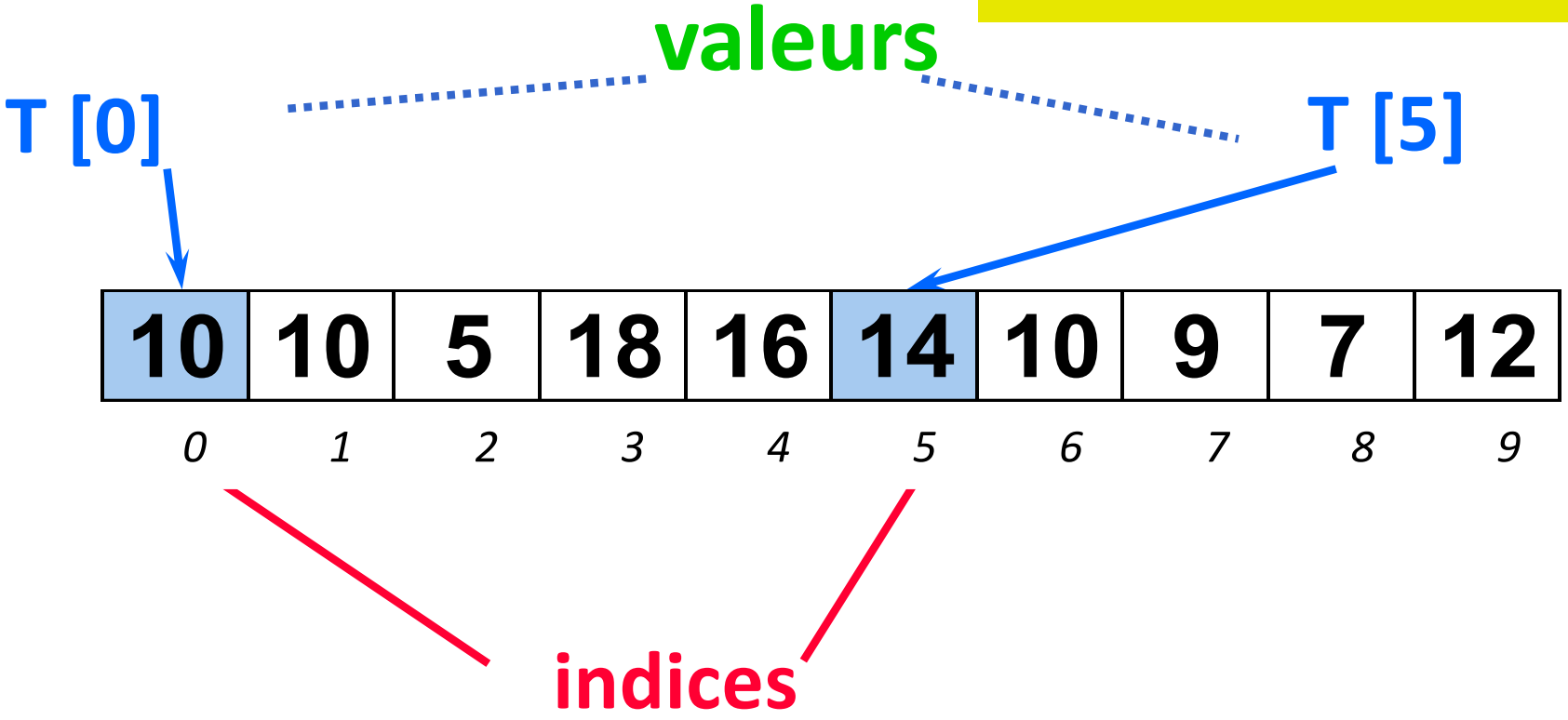
2. Listes en python

- Listes et tableaux
- Affectation et copies
- Manipulation
- Listes en compréhension

Notion de tableau

- Une variable de type tableau est une variable structurée composée de plusieurs cases numérotées.

La numérotation commence en général à 0



Tableaux statiques et dynamiques

- Un tableau peut être « statique »

Cas général dans la plupart des langages

  - Sa taille est fixée au moment de sa création et ne peut plus être modifiée.

Ex: déclaration d'un tableau en langage C

```
int T[10]; // déclare un tableau de 10 entiers
```

- Un tableau peut-être « dynamique », on parle alors de « liste »:
  - Sa taille est variable et peut évoluer pendant l'exécution.

Allocation dynamique

Les listes en python

- Les listes sont des tableaux dynamiques.
- Objets de la classe list
- Une liste contient une série de valeurs qui peuvent être de types différents (ex: chaines et entier)
- Créer une liste vide L=[ ]

```
noms = [‘pierre',‘paul',‘marie']
notes= [10.5, 15, 8]
etudiants= [‘pierre‘, 10.5, ‘paul', 15, ‘marie‘, 8]
```

Affectations et Copies de listes

- M=L

  # les listes M et L pointent sur le même objet

Copie d'une référence

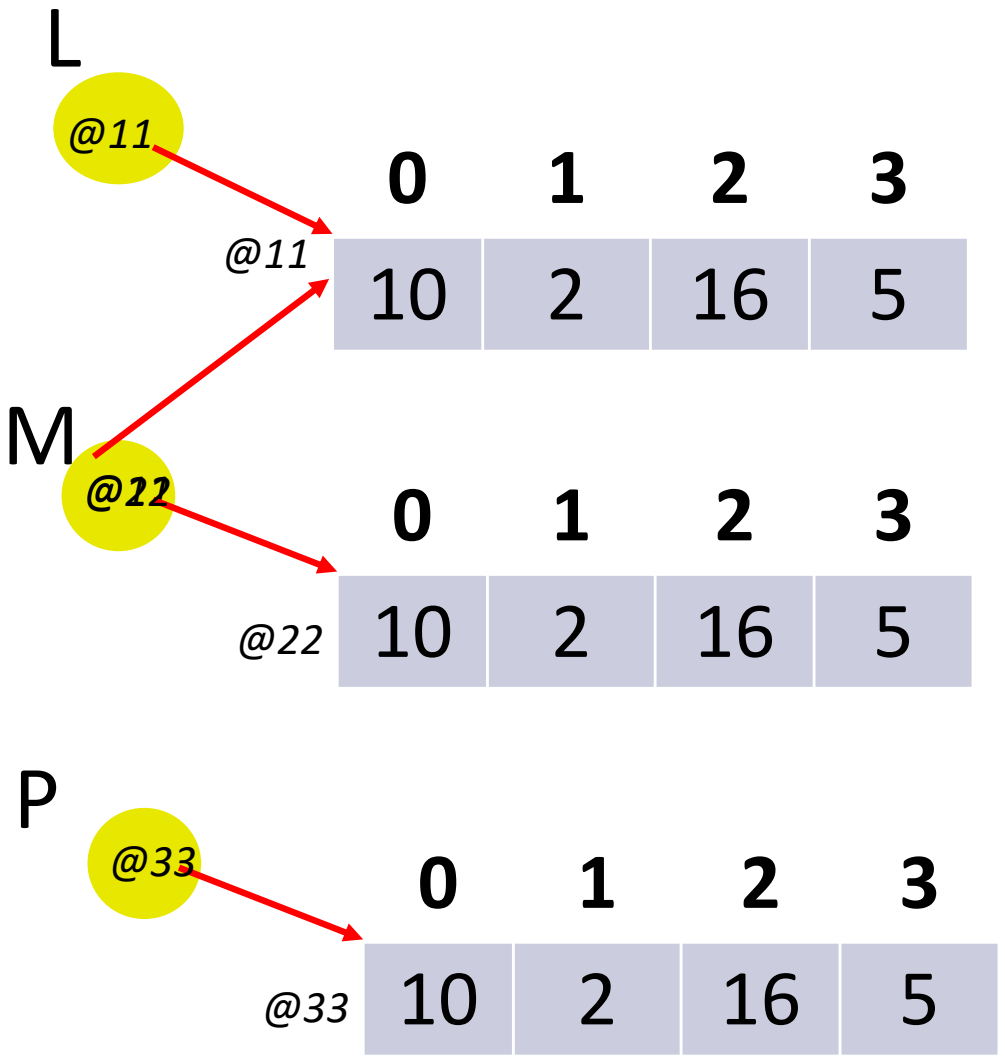
Copie véritable de la liste

- M=L[0:len(L)]

  #Création d'un autre objet liste L en mémoire

- P=list(M)

  #Création d'un autre objet liste L en mémoire



Manipulation des listes

- Concaténation: opérateur +

```
>>>noms1 = ['pierre','paul’]
>>>noms2 = ['jules','marie', 'lucie’]
>>>noms1+noms2
['pierre', 'paul', 'jules', 'marie', 'lucie']
```

Attention! création d’une autre liste en mémoire

- Duplication: opérateur *

```
>>>noms1*3
['pierre', 'paul', 'pierre', 'paul', 'pierre', 'paul‘]
```

Méthodes de la classe list

Méthode	Résultat
a.append(*x*)	ajoute l'élément x à la fin de la liste a
a.extend(*L*)	ajoute les éléments de la liste L à la fin de la liste a
a.insert(*i*, *x*)	insère l'élement x à la position i
a.remove(*x*)	supprime la première occurrence de l'élément x dans la liste a
a.count(x)	retourne le nombre d'occurrences de l'élément x dans la liste a

Equivalent à a[len(a):] = [x]

Equivalent à a[len(a):] = L

del L[i]
Supprime l'élément d'incide i

Méthodes de la classe list

Méthode	Résultat
a.index(*x*)	▪retourne l'indice de la première occurrence de l'élément x dans la liste a ▪génère une erreur si x n'est pas trouvé
a.pop([*i*])	▪Supprime l'élément d'indice i de la liste a et retourne cet élément ▪a.pop() : supprime et retourne le **dernier élément**
a.reverse()	inverse les éléments de la liste a
a.sort([key[,reverse]])	trie les éléments de la liste a selon la clé (la liste a est modifiée)

Comment trier une liste?

- Fonction sorted(L): renvoie une nouvelle liste triée

```
>>> L=[14,10,8,25,10,11]
>>> Res=sorted(L)
>>> Res
[8, 10, 10, 11, 14, 25]
```

- Méthode sort à appliquer sur une liste: modifie la liste

```
>>> L=[14,10,8,25,10,11]
>>> L.sort()
>>> L
[8, 10, 10, 11, 14, 25]
```

Attention invocation de méthode sur L
L.sort ()

Listes en compréhension

- Les listes en compréhension permettent de créer de façon concise des listes à partir de listes existantes ou d'itérables (ex: range(…))
- Syntaxe

L=[expression(*x*) for *x* in *listeExistante* if *condition(x)*]

- Exemples

```
A= [x for x in range(5)] #[0, 1, 2, 3, 4]
B=[x for x in range(0,21) if x%3==0]#[0, 3, 6, 9, 12, 15, 18]
C=[x for x in B if 10<= x <=15] #[12,15]
D=[2*x for x in F] # [24, 30]
```

Nombres entre 0 et 20 divisibles par 3

Eléments de B compris entre 10 et 15

Listes en compréhension imbriquées

- Exemple

Construire une liste contenant successivement les tables de multiplication de 2 à 3

```
m = [i*n for i in range(2,4) for n in range(1,11)]
#[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```

Pour plus de détails voir https://code.tutsplus.com/fr/tutorials/list-comprehensions-in-python--cms-26836

Listes de listes

- Les tableaux à plusieurs dimensions peuvent être définis en python par des listes de listes.
- Une matrice par exemple:

```
M= [[1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12]]
M[1][0]=0
#M= [[1, 2, 3, 4],[0, 6, 7, 8], [9, 10, 11, 12]]
```

Initialisation de Listes de listes

- Affectation directe

  L = [[1, 2], [11, 22], [111, 222]]

- Création de listes de listes avec valeurs répétées

```
M=[0]*3  #création d'une liste d'entiers
M[0]=[1]*3 #remplacement de la valeur d'indice 0 par une liste de 1
M[1]=[1]*3 #remplacement de la valeur d'indice 1 par une liste de 1
M[2]=[1]*3 #remplacement de la valeur d'indice 1 par une liste de 1
#M= [[1, 1, 1], [1, 1, 1], [1, 1, 1]]
```

Initialisation de Listes de listes

- Initialisation d'une matrice carrée de
- dimension 3 avec des zéros

```
>>>MC=[[O]*3] *3
>>>MC
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>>MC[0][1]=10
>>>MC
[[0, 10, 0], [0, 10, 0], [0, 10, 0]]
```

Il y a création d'une première liste [0,0,0] puis copie de sa référence (et non copie réelle)
Une solution : utiliser des listes en compréhension

Listes de listes en compréhension

- Création d'une liste de listes

```
M=[[0 for j in range(3)] for i in range(3)]
#M=[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
L=[[x, x**2] for x in range(1,5)]
#[[1, 1], [2, 4], [3, 9], [4, 16]]
```

- Création d'une liste de tuples

```
LT=[(x, x**2) for x in range(1,5)]
#LT=[(1, 1), (2, 4), (3, 9), (4, 16)]
```

Attention
LT[0][0]=2
INTERDIT
Car les tuples sont immutables

- Aplatissement d'une liste de listes

```
A = [n for ligne in L for n in ligne]
#A=[1, 1, 2, 4, 3, 9, 4, 16]
```

3. Tuples en python

Les tuples

- Un tuple est une liste non modifiable
  - Objet immutable
- Création d'un tuple :
  - t=() #tuple vide
  - t= 1, 2, 3 ou t=(1,2,3)
  - tNoms=(1, "pierre",   "paoli")

Manipulation de tuples

- t=(1,2,3)
- print(t[1])
- t[1]=0

```
2

t[1]=0
TypeError: 'tuple' object
does not support item
assignment
```

Les tuples sont immutables!!

- t=(12,13,14,15)

Accepté!! POURQUOI?

L'objet tuple initial n'est pas modifié.
Il y a création d'un nouvel objet tuple affecté à la variable t

Listes de tuples

```
>>> etudiants = [( "Pierre", 10),( "Paul", 4),
( "Marie", 18),( "Jean", 15),( "Laura", 9)]
>>> noteMarie=etudiants[2][1]
```

Attention:

- Structure de données à choisir si l'on ne souhaite pas modifier les valeurs
  - etudiants[2][1]=19 provoque une erreur
- Si besoin de modification, préférer une liste de listes.

4. Dictionnaires en python

Dictionnaires python

- Création d'un dictionnaire vide:
  - D=dict()
  - D={}
- Exemples :
  - D=dict()
  - D["nom"] = "Pierre"
  - D["age"] = 10
  - Param = {'Police': 12, 'Style': 'Gras', 'Couleur': [255, 0, 255]}

Représentation des donnés EXIF d'une image par un dictionnaire

clés	valeurs
Make	= Canon
Model	= Canon EOS 7D
Orientation	= top/leftX
Resolution	= 72Y
Resolution	= 72
Resolution Unit	= inch
Date Time	= 2016-09-05 10 :40 :35
Artist	=YCbCr
Positioning	= co-sited
Copyright	=Exif IFD

Dictionnaires python

Soit D = {"nom": "Olivier", "age": 30}

- Accéder à la valeur d'une clé
  - print(D ["age"]) #affiche 30
  - *Ou* print(D.get("age"))

x=D.get ("age", 0)
x est la valeur de la clé age si elle existe et 0 sinon

Attention si la clé n'existe pas:
- D[cle] provoque une erreur d'exécution
- D.get(cle) renvoie None

- Supprimer la valeur d'une clé
  - del(D ["age"]))

Initialiser un dictionnaire en compréhension

- Dictionnaire dont les clés sont les entiers de 0 à 5 et les valeurs 0
  - dico = { cle:0 for cle in range(0,5)}

```
{0: 0, 1: 0, 2: 0, 3: 0, 4: 0}
```

- Création d'un dictionnaire dont les clés sont les lettres de l'alphabet et les valeurs leur numéro dans l'alphabet
  - Dico={chr(i+96):i for i in range(1,27)}

```
{'z': 26, 'a': 1, 'l': 12, 'e': 5, 'm': 13, 'o': 15, 'p': 16, 'u': 21, 'y': 25, 't': 20, 'k': 11, 'f': 6, 'i': 9, 'b': 2, 's': 19, 'x': 24, 'v': 22, 'd': 4, 'n': 14, 'h': 8, 'c': 3, 'q': 17, 'r': 18, 'g': 7, 'j': 10, 'w': 23}
```

Boucle de parcours de dictionnaires

- car = {" a":21, "b":3, "c":31}
- for c in car.keys():

  print(c) #affiche les clés

```
c
b
a
```

- for v in car.values():

  print(v) #affiche les valeurs

```
31
3
21
```

- for c,v in car.items():

  print(c,v) #affiche les clés et les valeurs

```
c 31
b 3
a 21
```

Utilisation d'un dictionnaire pour des calculs de fréquence

```
ens = ["a", "b",  "c", "e", "b",  "c"]
freq = {}
for e in ens:
    if e in freq:
        freq[e] += 1
    else:
        freq[e] = 1
hist
```

```
freq[e] = freq.get(e, 0) + 1
```

```
hist
{'e': 1, 'a': 1, 'c': 2, 'b': 2}
```

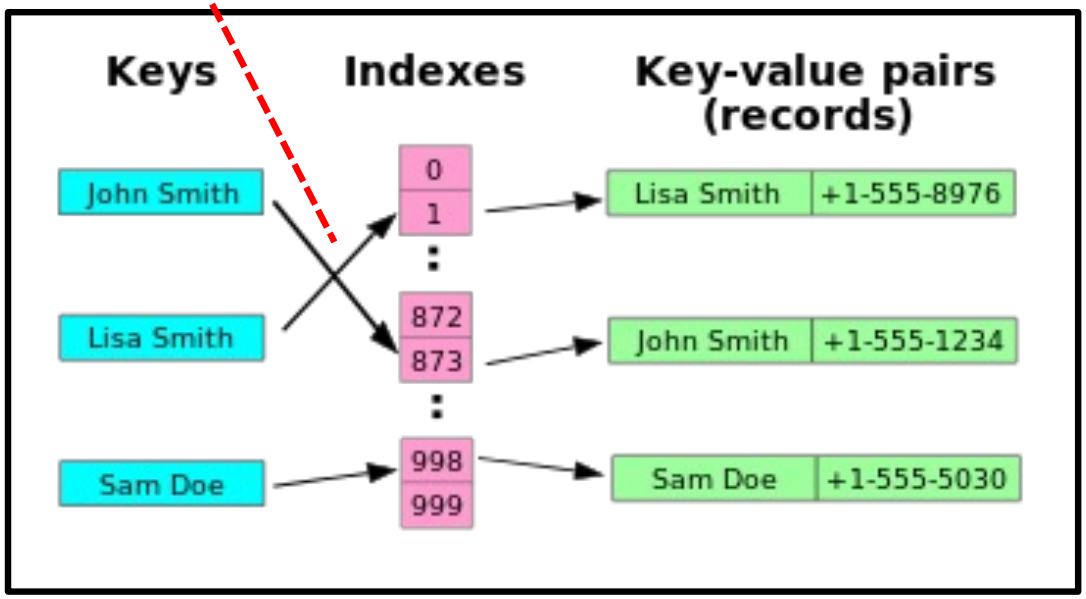
Propriétés des clés

- Les valeurs des clés sont uniques
  - Si l'on essaye d'ajouter une clé déjà présente, la clé existante sera modifiée.
- En revanche, deux clés différentes peuvent être associées à la même valeur
- Une clé peut être définie sur un type structuré:
    - Par exemple une matrice peut être représentée par un dictionnaire dont les clés sont les coordonnées (tuple(ligne,colonne))

Pourquoi utiliser un dictionnaire au lieu d'une liste de tuples?

- Les valeurs sont modifiables (comme dans une liste de listes)
- Les recherches sont beaucoup plus rapides.

Calcul d'un indice par une fonction de hachage



L'accès à une valeur à partir de sa clé est directe (table de hachage)

Source image: https://fr.wikipedia.org/wiki/Table_de_hachage

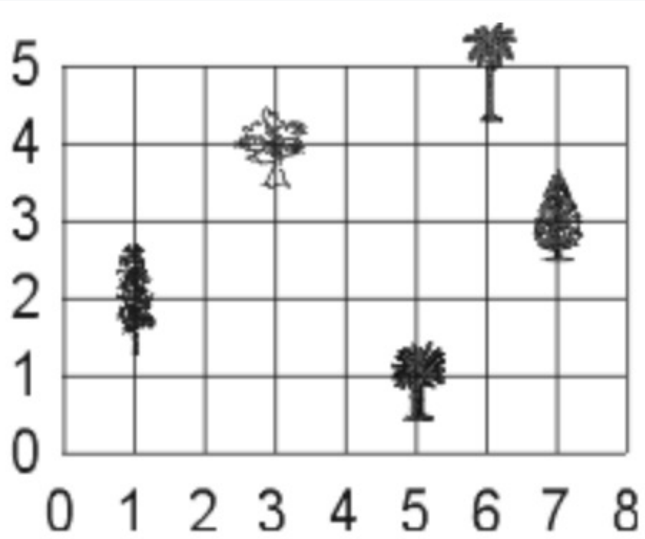
Exemple de dictionnaire

Représentation de la position d'arbres sur un quadrillage

```
>>> arb = {}
>>> arb[(1,2)] = 'Peuplier'
>>> arb[(3,4)] = 'Platane'
>>> arb[6,5] = 'Palmier'
>>> arb[5,1] = 'Cycas'
>>> arb[7,3] = 'Sapin'

>>> print arb
{(3, 4): 'Platane', (6, 5): 'Palmier', (5, 1): 'Cycas', (1, 2): 'Peuplier', (7, 3): 'Sapin'}
>>> print arb[(6,5)]
palmier
```

Les clés sont les coordonnées (tuples)



Source : https://fr.wikibooks.org/wiki/Programmation_Python/Dictionnaires

5. Tableaux numpy

Tableaux Numpy

- Numpy est une bibliothèque numérique permettant de créer des tableaux multi dimensionnels et de les manipuler de manière efficace
- Importation du module :
  - **import** numpy as np

Très utiles pour la manipulation de matrices

- Contrairement aux listes, les tableaux numpy ne regroupent que des éléments du même type
- Comme les listes, les tableaux sont des objets mutables (modifiables) dont la taille peut évoluer dynamiquement par ajout de nouveaux éléments (méthode append)

Tableaux numpy

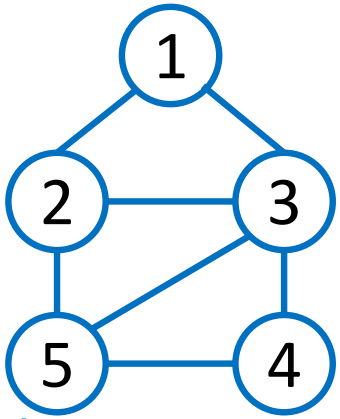
- Création d'un tableau à partir d'une liste
  - v = np.array([1, 3, 2, 4])
- Initialisation avec des 0 ou des 1 : zeros(dim1,dim2, …) et ones(dim1,dim2, …)
  - v=np.zeros(4)
- Initialisation avec des nombres aléatoires
  - v=np.random.randint(0, 10, 5)

  Tableau de 5 nombres aléatoires entre 0 et 10
- Taille d'un tableau: v.size
- Possibilité de calculs groupés sur l'ensemble des valeurs d'un tableau (sans faire de boucle):
  - Exemple :
  - TCarre=T**2 *#TCarre contient les éléments du tableau T élévés au carré*

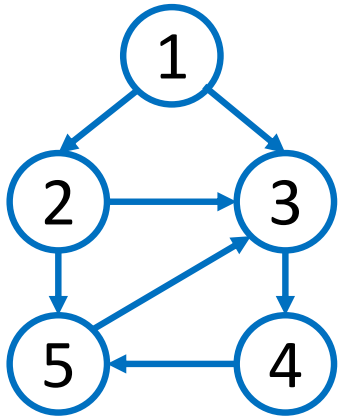
6. Représentation des graphes

Graphes: Définitions de base

- Un graphe G c'est un couple (S,A) avec :
  - S= ensemble fini non vide de sommets
  - A= *ensemble de paires de sommets appelées arêtes(relation binaire sur S)*
- Le graphe est dit orienté si les paires sont ordonnées. Elles sont alors appelées arcs.



Graphe non orienté



Graphe orienté

Représentation d'un graphe en mémoire

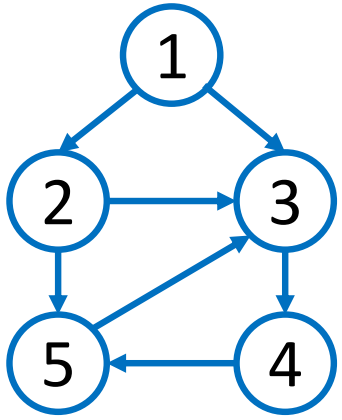
- Trois représentations possibles:
  - Matrice d'adjacence
  - Listes d'adjacence
  - Matrice d'incidence
- Le choix de la structure de données utilisée aura un impact sur la place mémoire nécessaire, l'implémentation des algorithmes et leur complexité.

Matrice d'adjacence
Graphe orienté

- Matrice carrée d'ordre n (n=nombre de sommets)
  - mat[i,j] = 1 si les sommets *i* et *j* sont adjacents

    ((i,j) ∈ A)
  - mat[i,j]=0 sinon

Graphe G1

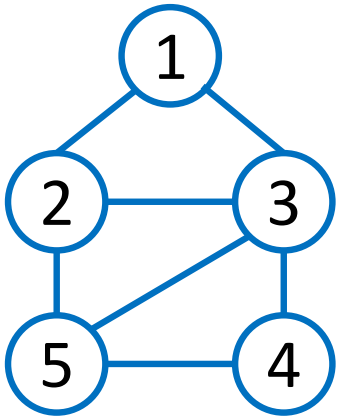


	1	*2*	*3*	*4*	*5*
1	0	1	1	0	0
2	0	0	1	0	1
3	0	0	0	1	0
4	0	0	0	0	1
5	0	0	1	0	0

Matrice d'adjacence graphe non orienté

matrice symétrique

Graphe G2



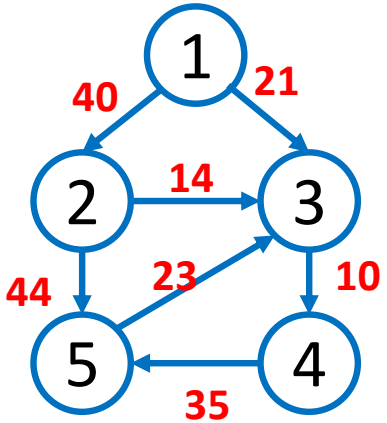
	1	2	3	4	5
1	0	1	1	0	0
2	1	0	1	0	1
3	1	1	0	1	1
4	0	0	1	0	1
5	0	1	1	1	0

Avantages	Inconvénients
▪simplicité d'accès aux sommets ▪adapté aux graphes simples	▪Place mémoire ▪ Redondance d'information pour les graphes non orientés ▪stockage et examen inutile de zéros

Matrice d'adjacence
Graphe pondéré

- Un graphe est dit pondéré ou valué si des poids sont associés aux arêtes (arcs):
  - Dans la matrice d'adjacence, les 1 sont remplacés par les poids

Graphe G3

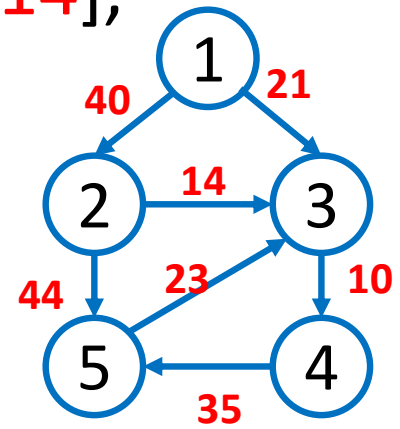


	1	2	3	4	5
1	0	40	21	0	0
2	0	0	14	0	44
3	0	0	0	10	0
4	0	0	0	0	35
5	0	0	23	0	0

Implémentation en python (sol1)

- Liste de sommets + liste d'arcs (ou arêtes)
  - Sommets=[0,1,2,3,4]
  - Arcs= [[0,1], [0,2], [1,4], [1,2],[2,3], [3,4], [4,2]]
- Si le graphe est pondéré, la liste d'arcs est composée de triplets:
  - Arcs= [[0,1,40], [0,2,21], [1,4,44], [1,2,14], [2,3,10], [3,4,35], [4,2,23]]

sommets numérotés à partir de 0



Graphe G3

Implémentation en python (sol2)

- Matrice d'adjacence =Array de numpy
  - initialisation avec des zéros

    mat=np.zeros([nbSommets, nbSommets])
  - méthode *shape*: renvoie un tuple (nombre lignes, nombre colonnes)
  - Boucle de parcours

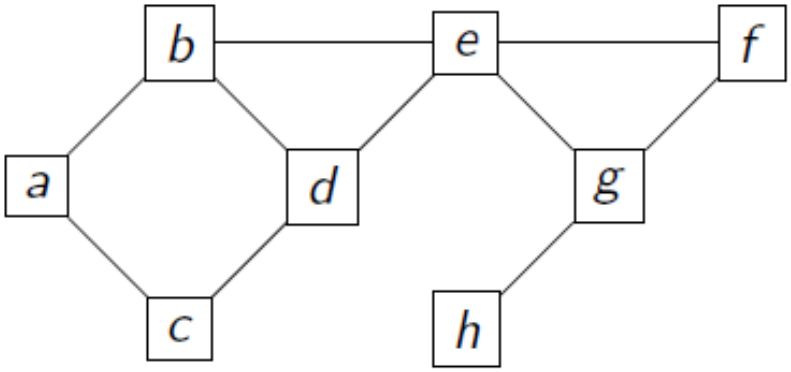
```
for iLigne in range(0,mat.shape[0]):
    for iCol in range(0,mat.shape[1]):
```

Implémentation Python (sol3)

- Utilisation d'un dictionnaire

```
G=dict()
G['a']=['b','c']
G['b']=['a','d','e']
G['c']=['a','d']
G['d']=['b','c','e']
G['e']=['b','d','f','g']
G['f']=['e','g']
G['g']=['e','f','h']
G['h']=['g']
```

Liste d'adjacence



Liens

- Listes et tableaux en python
  - http://python.physique.free.fr/listes_et_tableaux.html
- Des complements sur les listes
  - https://openclassrooms.com/fr/courses/1206331-utilisation-avancee-des-listes-en-python
- Librairie numpy :
  - http://informatique-python.readthedocs.io/fr/latest/Cours/science.html
  - Documentation officielle:
    - http://docs.scipy.org/doc/numpy/reference
    - https://docs.scipy.org/doc/numpy/user/
    - https://docs.scipy.org/doc/numpy/genindex.html#R (index des fonctions)

Liens

- Des exercices qui peuvent vous être utiles:
  - https://www.irif.fr/~sangnier/enseignement/IP1-Python/IP1-Python-cours-td-5-7.pdf
  - http://hebergement.u-psud.fr/iut-orsay/Pedagogie/MPHY/Python/exercices-python3.pdf