

# Les tableaux en Python (Listes)

- Listes 1D
- Algorithmes de recherche

Radouan Dahbi  
[r.dahbi@caplogy.com](mailto:r.dahbi@caplogy.com)



01

## Les listes 1D



# Les listes 1D

## Définition

- Un tableau est représenté en Python par une liste.
- Une liste : Une variable structurée regroupant des données **les unes à la suite des autres**.
- Chaque valeur est identifiée par **sa position** dans la liste.
- **Exemples :**

```
moyennes = [15.3, 18.9, 12.5, 9.6, 17.1] # Liste de valeurs de type float représentant des moyennes
```

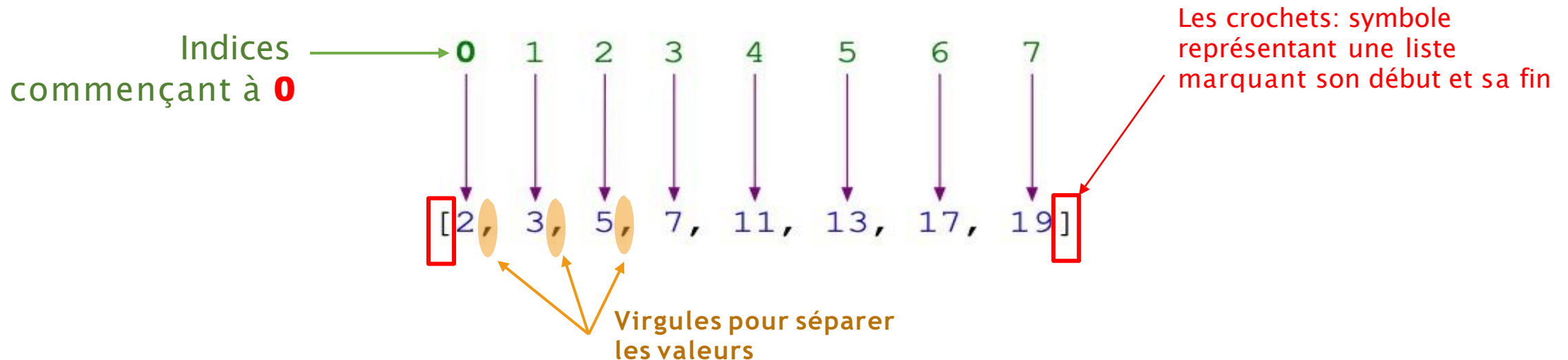
```
L = [ 12, -2, 34, 11, 897] # Liste de valeurs de type int
```

```
prenoms = ["Jean", "Marie", "Amine", "Pedro"] # Liste de valeurs de type str représentant des prénoms
```

# Les listes 1D

## Représentation d'une liste

- Python présente les listes comme une suite de valeurs séparées par des virgules, entourées par des crochets. Un indice (position) est associé à chaque valeur de la liste.
- Les indices sont attribués dans **un ordre séquentiel croissant** de gauche à droite et commencent par **ZERO (0)**

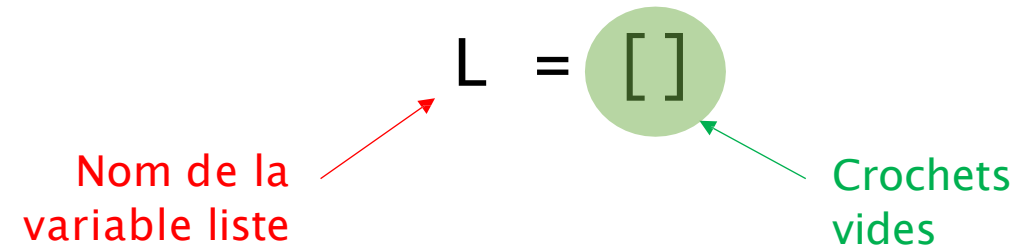


# Les listes 1D

## Création d'une liste

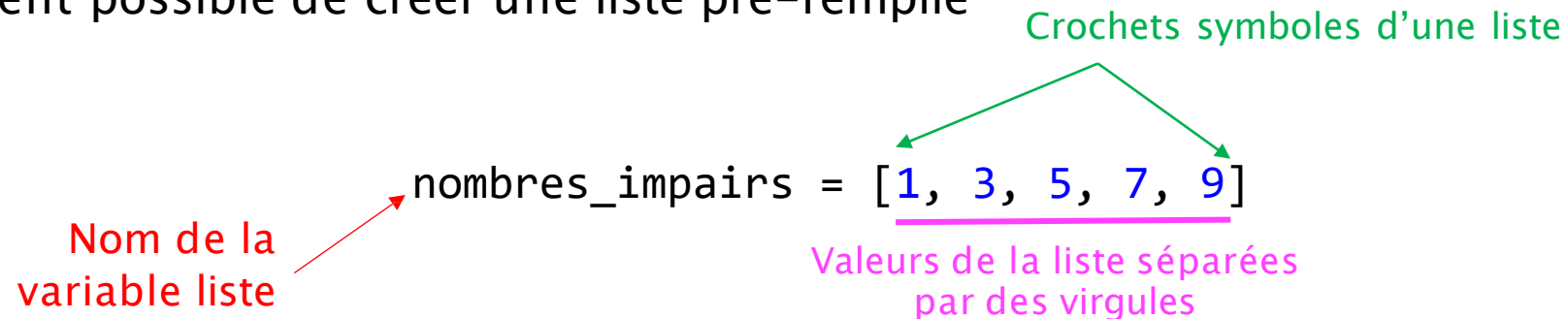
### Création d'une liste vide :

Il est possible de créer une liste vide et lui ajouter des valeurs plus tard

 `L = []`

### Création d'une liste avec des valeurs :

Il est également possible de créer une liste pré-remplie

 `nombres_impairs = [1, 3, 5, 7, 9]`

# Les listes 1D

## Accès aux éléments d'une liste

On accède à un élément de la liste en utilisant le nom de la liste et la position de cet élément entre crochets.

```
nom_liste[indice_element]
```

### Exemple :

```
premiers = [2, 3, 5, 7, 11, 13, 17, 19]
print("La valeur à l'indice 3 est :", premiers[3]) # À noter que c'est la 4ème valeur de la liste
print("La 5ème valeur de cette liste est :", premiers[4]) # Car la numérotation commence à partir de 0
```

```
La valeur à l'indice 3 est : 7
La 5ème valeur de cette liste est : 11
```

# Les listes 1D

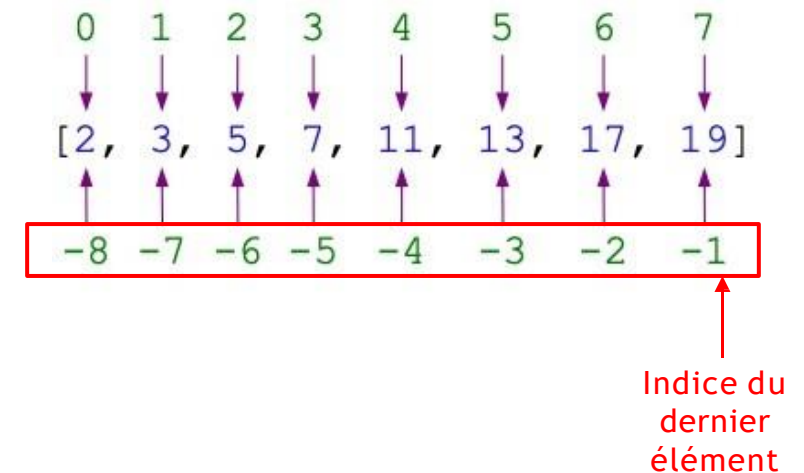
## Les indices négatifs

En Python, les valeurs d'une liste sont également accessibles de la fin (droite) vers le début (gauche) en utilisant des indices négatifs. Ainsi:

- -1 est l'indice du dernier élément
- -2 est l'indice de l'avant dernier élément
- ...

### Exemple :

```
fruits = ["banane", "poire", "framboise", "pomme"]  
print("Le dernier élément de cette liste est :", fruits[-1])  
print("Le 3ème élément en partant de la fin de la liste est :", fruits[-3])
```



```
Le dernier élément de cette liste est : pomme  
Le 3ème élément en partant de la fin de la liste est : poire
```

# Les listes 1D

## La taille d'une liste

- En Python, il est possible d'avoir à tout moment le nombre de valeurs d'une liste
- Le nombre de valeurs représente la taille logique de la liste
- La fonction qui donne la taille est : **len(nom\_liste)**

### Exemple :

```
fruits = ["banane", "poire", "framboise", "pomme"]
print("la taille de la liste fruits est :", len(fruits))
moyennes = [15.3, 18.9, 12.5, 9.6, 17.1]
print("la taille de la liste moyennes est :", len(moyennes))
```

```
la taille de la liste fruits est : 4
la taille de la liste moyennes est : 5
```



# Les listes 1D

## Parcourir une liste

### Parcourir une liste via les indices

- Grâce à une boucle **for**
- Utilisation de l'opérateur « **in** » sur les valeurs des indices des éléments d'une liste.

indice : prend les valeurs de  
0 à `len(nom_liste) - 1`



**for** indice **in** range (`len(nom_liste)`) :

### Exemples :

```
fruits = ["banane", "poire", "framboise", "pomme"]  
for indice in range (len(fruits)):  
    print(fruits[indice], end=" ")
```

```
banane poire framboise pomme
```

# Les listes 1D

## Parcourir une liste

### Parcourir une liste via les indices



**ATTENTION !!!** Lors du parcours d'une liste, il ne faut **JAMAIS** dépasser sa taille.

Les indices d'une liste *nom\_liste* de taille *len(nom\_liste)* sont entre **0** et **len(nom\_liste) – 1**  
Toute tentative d'accès à une case se trouvant en dehors de cet intervalle engendre une **erreur**

### Exemples :

```
fruits = ["banane", "poire", "framboise", "pomme"]  
print(fruits[len(fruits)])
```

La case d'indice len(fruits)  
n'existe pas

```
Traceback (most recent call last):  
  File "/.....", line 20, in <module>  
    print(fruits[len(fruits)])  
IndexError: list index out of range
```

# Les listes 1D

## Parcourir une liste

### Parcourir une liste via un itérateur

- Grâce à une boucle **for**
- En utilisant l'opérateur d'appartenance « **in** » à une liste

### Exemple :

```
fruits = ["banane", "poire", "framboise", "pomme"]  
for valeur in fruits:  
    print(valeur, end=" ")
```

**for** valeur **in** nom\_liste:

↑  
Représente une valeur  
(contenu de la case) de la  
liste

banane poire framboise pomme

# Les listes 1D

## Affichage des éléments d'une liste

En Python, il est possible d'afficher une liste en bloc en utilisant la fonction **print()** comme pour une simple variable.

```
print(nom_liste)
```

Mettre directement le nom  
de la liste sans indices

### Exemple :

```
fruits = ["banane", "poire", "framboise", "pomme"]  
print(fruits)
```



```
['banane', 'poire', 'framboise', 'pomme']
```

Style d'affichage unique: sous forme d'une liste

- Crochets en début et fin de la liste
- Virgules pour séparer les valeurs
- Ici quotes car valeurs de type str

# Les listes 1D

## Ajout d'un élément à la liste

- La notion de taille physique n'existe pas sur les listes
- Il est possible d'ajouter à tout moment des valeurs dans la liste
- La taille de la liste est automatiquement mise à jour suite à un ajout d'une valeur
- La méthode **append()** permet d'ajouter une valeur mais **toujours à la fin** de la liste

**nom\_liste.append(valeur\_a\_ajouter)**

### Exemple :

```
fruits = ["banane", "poire", "framboise", "pomme"]
print("la taille de la liste fruits est :", len(fruits))
fruits.append("melon")
print("la nouvelle taille de la liste fruits est :", len(fruits)) # la taille de la liste est mise à jour
```

```
la taille de la liste fruits est : 4
la nouvelle taille de la liste fruits est : 5
```

# Les listes 1D

## Suppression d'un élément de la liste

- Il s'agit d'une suppression d'un élément par son indice
- La fonction permettant la suppression est **del** `del nom_liste[indice_valeur_a_supprimer]`
- Après la suppression les éléments sont automatiquement décalés vers la gauche
- La taille de la liste est mise à jour automatiquement

### Exemple :

```
fruits = ["banane", "poire", "framboise", "pomme"]
print("la taille de la liste fruits est :", len(fruits))
fruits.append("melon")
print("la taille de la liste fruits après l'ajout d'un fruit est :", len(fruits))
del fruits[2]
print("la taille de la liste fruits après la suppression du fruit à l'indice 2 :", len(fruits))
```

```
la taille de la liste fruits est : 4
la taille de la liste fruits après l'ajout d'un fruit est : 5
la taille de la liste fruits après la suppression du fruit à l'indice 2 : 4
```

# Les listes 1D

## Modification d'un élément de la liste

Pour modifier une valeur dans une liste se trouvant à une position donnée, il suffit d'affecter la nouvelle valeur à la case de la même position.

```
nom_liste[indice_valeur_a_modifier] = nouvelle_valeur
```

### Exemple :

```
fruits = ["banane", "poire", "framboise", "pomme"]  
print(fruits)
```

```
fruits[2] = "fraise"  
print(fruits)
```

```
fruits[-1] = "abricot"  
print(fruits)
```

```
['banane', 'poire', 'framboise', 'pomme']  
['banane', 'poire', 'fraise', 'pomme']  
['banane', 'poire', 'fraise', 'abricot']
```

# Les listes 1D

## Les slices en Python

- Permettent le découpage d'une liste afin d'en extraire une partie.
- Slices possibles :
  - `nom_liste [n : p]` retourne la liste des éléments numérotés de **n** à **p - 1**.
  - `nom_liste [ : p]` retourne la liste des éléments numérotés de **0** à **p - 1** (tous les éléments jusqu'au (p-1) ième.)
  - `nom_liste [p : ]` retourne la liste de tous les éléments allant de l'indice **p** à **len(nom\_liste)-1**.



# Les listes 1D

## Les slices en Python

### Exemple :

```
fruits = ["banane", "poire", "framboise", "pomme"]  
  
print(fruits[1:3]) # valeurs des positions 1 et 2  
print(fruits[:3]) # valeurs des positions 0, 1 et 2  
print(fruits[2:]) # valeurs des positions 2 et 3  
print(fruits[:]) # toutes les valeurs de la liste
```

```
['poire', 'framboise']  
['banane', 'poire', 'framboise']  
['framboise', 'pomme']  
['banane', 'poire', 'framboise', 'pomme']
```

# Les listes 1D

## Autres opérations sur les listes

### Concaténation :



### Exemple :

```
liste1 = [1, 2, 3, 4]
liste2 = [5, 6, 7]
liste = liste1 + liste2
print(liste)
```

[1, 2, 3, 4, 5, 6, 7]

# Les listes 1D

## Autres opérations sur les listes

### Répétition :



Liste plus longue avec comme contenu les valeurs de **Liste** répétées **Nombre** fois

### Exemple :

```
liste1 = [1, 2]
liste = 3 * liste1
print(liste)
```

```
# Pour initialiser une liste, on peut utiliser ceci
L = [0] * 10
print(L)
```

[1, 2, 1, 2, 1, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

# Les listes 1D

## Autres opérations sur les listes

### Tester l'existence d'une valeur dans une liste :

Il est possible de chercher une valeur dans une liste en utilisant l'opérateur d'appartenance « **in** »

```
if valeur_recherchee in nom_liste :
```

### Exemple :

```
pre noms = ["Jean", "Marie", "Amine",  
"Pedro"] print("Donner votre prénom :")  
prenom = input()  
if prenom in noms :  
    print("Votre prénom existe déjà dans la  
liste") else:  
    print("Votre prénom n'existe pas dans la liste")
```

Donner votre prénom :

*Pedro*

Votre prénom existe déjà dans la liste

Donner votre prénom :

*Julie*

Votre prénom n'existe pas dans la liste

## 02

### **Algorithmes de recherche :**

- **Recherche linéaire (séquentielle)**
- **Recherche binaire (dichotomique)**

# Recherche linéaire (séquentielle)

## Recherche linéaire 1

- **tab** est un tableau d'entier **non trié**
- Quel est l'algorithme nécessaire pour déterminer si **x** se trouve dans **tab** ?
- Exemple de valeur pour **x** :
  - 72 se trouve-t-il dans **tab** ?
  - 12 se trouve-t-il dans **tab** ?
  - 6 se trouve-t-il dans **tab** ?

	0	1	2	3	4	5	6	7	8	9	10
tab	10	6	15	149	25	35	40	99	12	10	60



= BUROTIX ()

# Recherche linéaire (séquentielle)

## Objectif

- Rechercher une valeur dans un tableau (liste).
- Si la valeur est trouvée on retourne **son indice** sinon on retourne  $-1$ .

# Recherche linéaire (séquentielle)

## Principe

- Parcourir le tableau de la première à la dernière case en comparant la valeur recherchée à chacun des éléments présents dans le tableau.
- Quand faut-il arrêter de parcourir le tableau ?
  - Dès que l'élément est trouvé
  - Dès qu'on arrive à la fin



# Recherche linéaire (séquentielle)

## Recherche linéaire 1

```
x ← arg[1]
i ← 0
trouve ← FAUX
n ← NB(tab)
TANT QUE (i < n) FAIRE
    SI (tab[i]=x) ALORS
        trouve ← VRAI
        BREAK TANT QUE
    FIN SI
    i ← i + 1
FIN TANT QUE
```

	0	1	2	3	4	5	6	7	8	9	10
tab	10	6	15	149	25	35	40	99	12	10	60



= BUROTIX ()

# Recherche linéaire (séquentielle)

## Recherche linéaire 2

- **tab** est un tableau d'entier **trié**
- Quel est l'algorithme nécessaire pour déterminer si x se trouve dans **tab**?
- Exemple de valeur pour x :
  - 14 se trouve-t-il dans **tab**?
  - 12 se trouve-t-il dans **tab**?
  - 6 se trouve-t-il dans **tab**?

	0	1	2	3	4	5	6	7	8	9	10
tab	6	10	10	12	15	25	35	40	60	99	149



= BUROTIX ()

# Recherche linéaire (séquentielle)

## Objectif

- Rechercher une valeur dans un tableau (liste).
- Si la valeur est trouvée on retourne **son indice** sinon on retourne  $-1$ .

# Recherche linéaire (séquentielle)

## Principe

- Parcourir le tableau de la première à la dernière case en comparant la valeur recherchée à chacun des éléments présents dans le tableau.
- Quand faut-il arrêter de parcourir le tableau ?
  - Dès que l'élément est trouvé
  - Dès qu'on arrive à la fin
  - Dès que le l'élément du tableau est plus grand que le nombre recherché

# Recherche linéaire (séquentielle)

## Recherche linéaire 2

```
x ← arg[1]
i ← 0
trouve ← FAUX
n ← NB(tab)
TANT QUE (i < n ET tab[i] ≤ x) FAIRE
    SI (tab[i]=x) ALORS
        trouve ← VRAI
        BREAK TANT QUE
    SINON
        i ← i + 1
FIN TANT QUE
```

	0	1	2	3	4	5	6	7	8	9	10
tab	6	10	10	12	15	25	35	40	60	99	149



= BUROTIX 0

# Recherche binaire (dichotomique)

- **tab** est un tableau d'entier trié
- Comment recherche-t-on un mot dans le dictionnaire?
  - Au lieu de rechercher page après page en démarrant du début, on ouvre le dictionnaire au milieu
  - On compare le mot à chercher avec un mot de la page du milieu du dictionnaire
  - Si c'est le même, on l'a trouvé
  - Sinon on recommence sur la première moitié ou la deuxième selon que le mot à chercher est avant ou après

	0	1	2	3	4	5	6	7	8	9	10
tab	6	10	10	12	15	25	35	40	60	99	149



= BUROTIX ()

# Recherche binaire (dichotomique)

## Objectif

- Rechercher une valeur dans un tableau (liste).
- Si la valeur est trouvée on retourne **son indice** sinon on retourne  $-1$ .

# Recherche binaire (dichotomique)

## Principe

- Pour chercher un élément **x** dans un tableau **T trié** en ordre croissant :
  - On calcule l'indice du milieu du tableau
  - On compare **x** avec  $T[\text{milieu}]$  :
    - Si  $x = T[\text{milieu}]$ , on retourne l'indice et on quitte le programme
    - Si  $x > T[\text{milieu}]$ , on cherche **x** dans la partie supérieure du tableau
    - Si  $x < T[\text{milieu}]$ , on cherche **x** dans la partie inférieure du tableau



# Recherche binaire (dichotomique)

- Dichotomie = couper en deux
- Principe
  - Au lieu de rechercher séquentiellement, on compare l'élément à chercher à l'élément qui se trouve au **milieu** du tableau
  - Si c'est le même, trouve=true
  - Sinon on recommence sur la **première moitié** ou la **deuxième** selon que l'élément à chercher est plus petit ou plus grand
- Exemple: 41

Milieu?  
Comment le  
calculer?

tab

0	1	2	3	4	5	6	7	8	9	10
6	10	10	12	15	25	35	40	60	99	149



# Recherche binaire (dichotomique)

```

x ← arg[1]
b_inf ← 0
b_sup ← NB(tab)-1
trouve ← FAUX
TANT QUE trouve = FAUX ET b_inf ≤ b_sup FAIRE
    m ← (b_sup + b_inf) DIV 2 // Division entière
    SI (tab[m] = x) ALORS
        trouve ← VRAI
    SINON
        SI (x < tab[m]) ALORS
            b_sup ← m-1
        SINON
            b_inf ← m + 1
        FIN SI
    FIN SI
FIN TANT QUE

```

	0	1	2	3	4	5	6	7	8	9	10
tab	6	10	10	12	15	25	35	40	60	99	149

