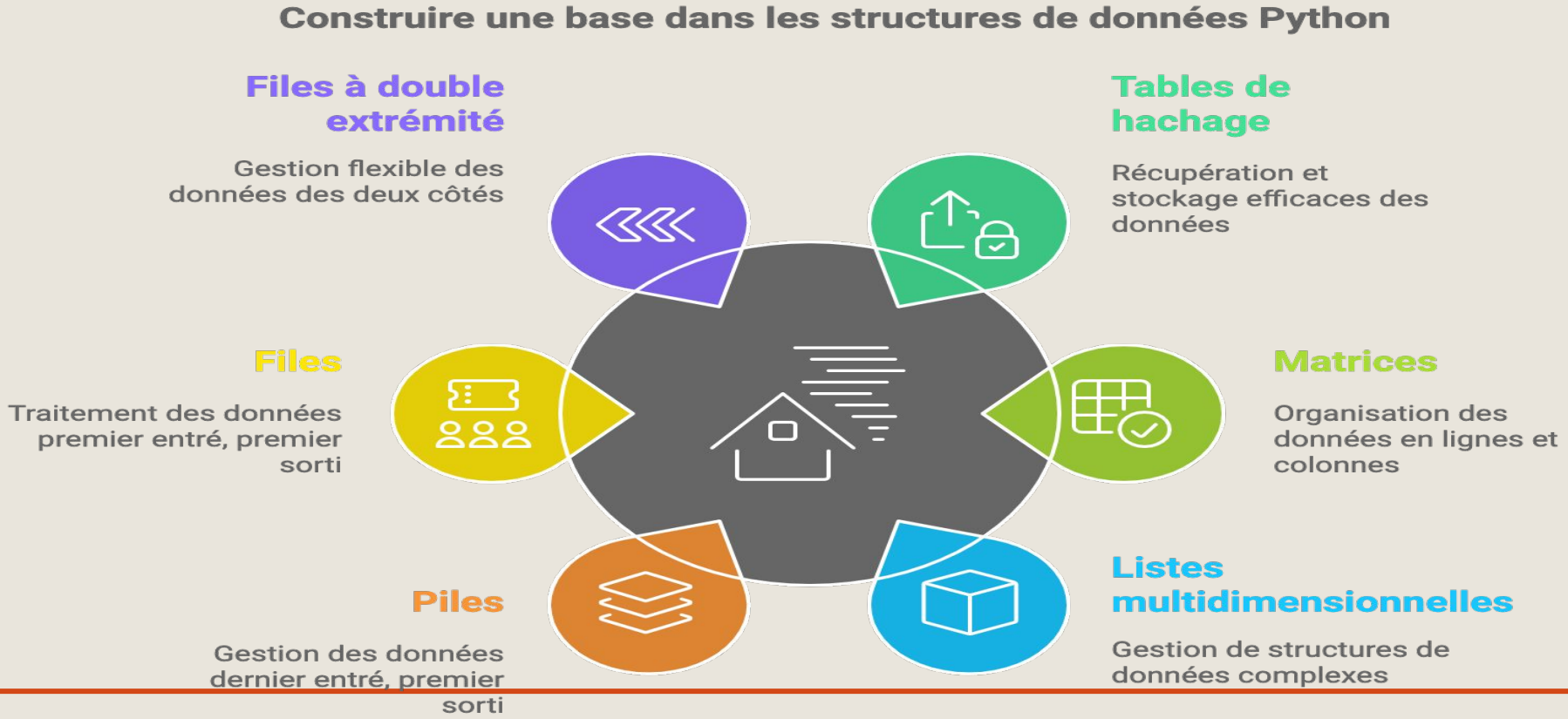
Algo-Python & Applications

Séance N°5

Table of contents

Les structures des donnés

Introduction

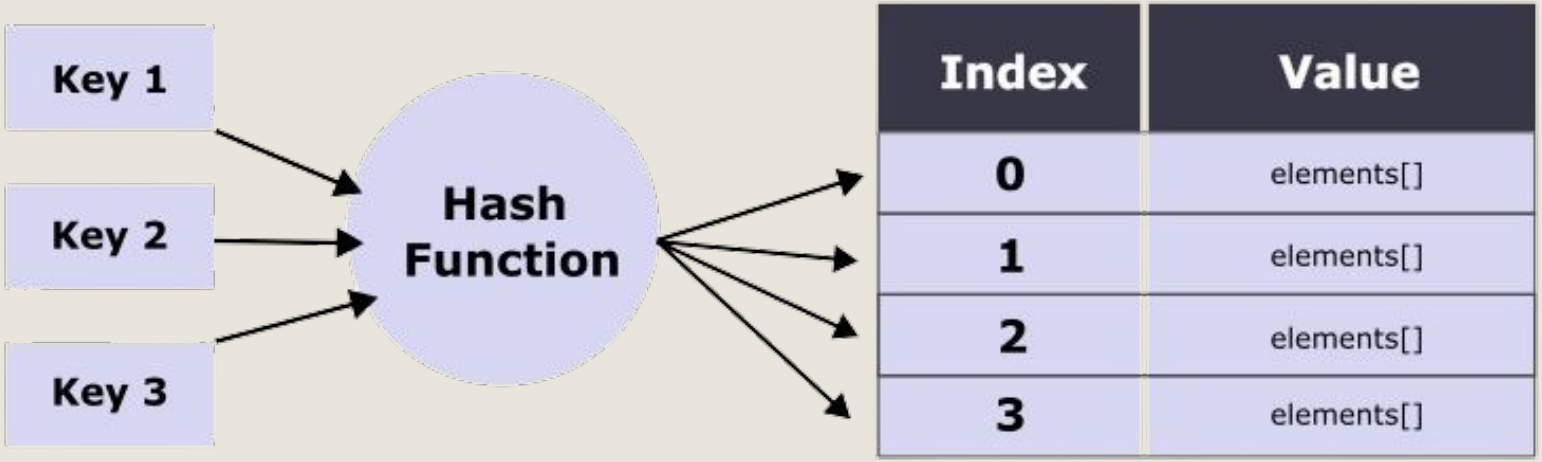


01

Les tables de hachage
Définition & Manipulation

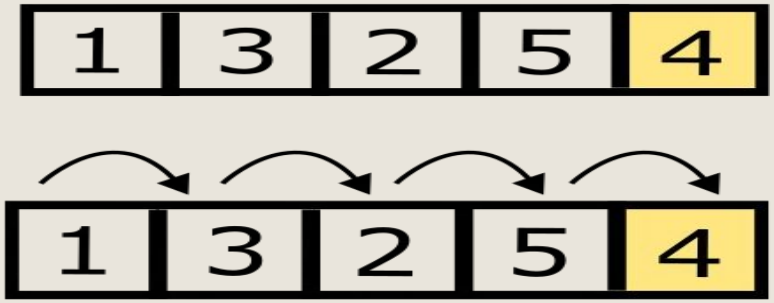
Définition

Une table de hachage (ou hash table en anglais) est une structure de données qui permet d'associer des clés à des valeurs. Contrairement aux listes ou aux tableaux, où l'accès à un élément se fait par son index, une table de hachage permet d'accéder directement à une valeur en connaissant sa clé.



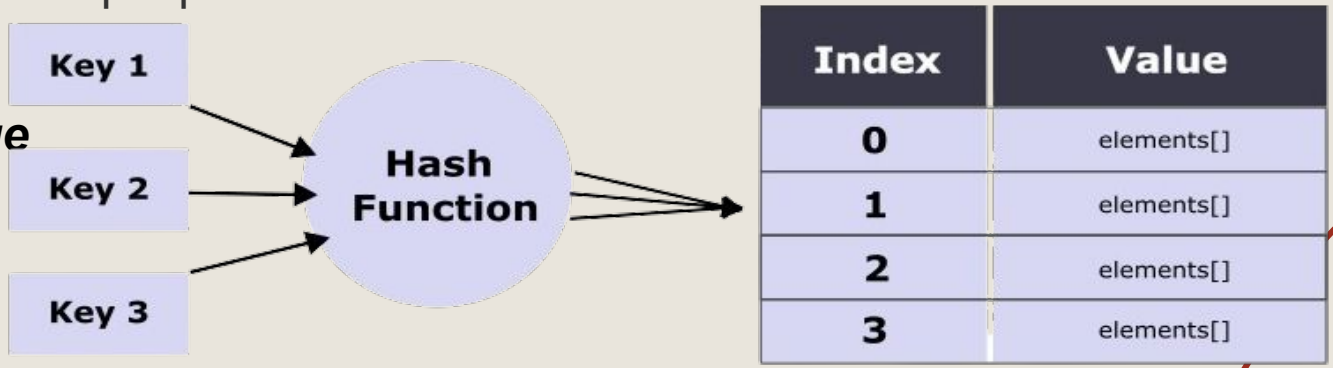
Exemple

Pour voir l'utilité d'une telle structure, prenons l'exemple du tableau suivant:



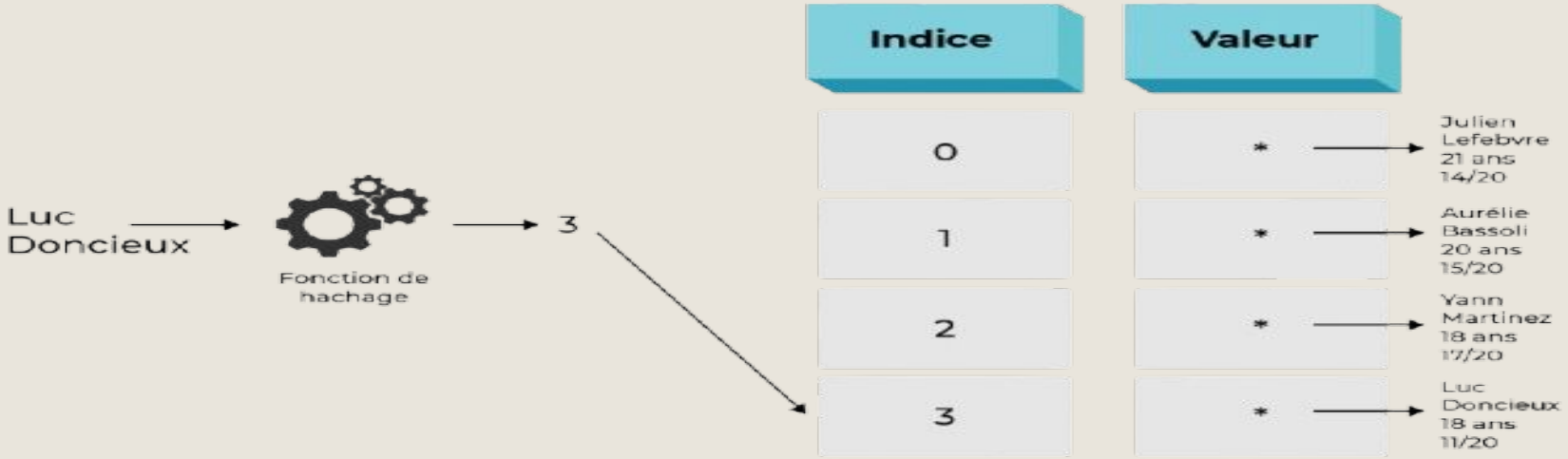
Cependant, il est possible que plusieurs clés soient hachées dans la même case.

Collisions de hachage



La fonction de hachage

La fonction de hachage est une fonction qui prend en tant qu'entrée une clé et qui nous renvoie un indice pour accéder à une valeur dans une structure dans laquelle nous pouvons indexer (un tableau).



Les algorithmes **MD5** et **SHA1** sont des fonctions de hachage célèbres.

Exemple simple de fonction de hachage

Additionner les valeurs ASCII de chaque lettre du nom, c'est-à-dire pour Cours Python, de faire la somme suivante :

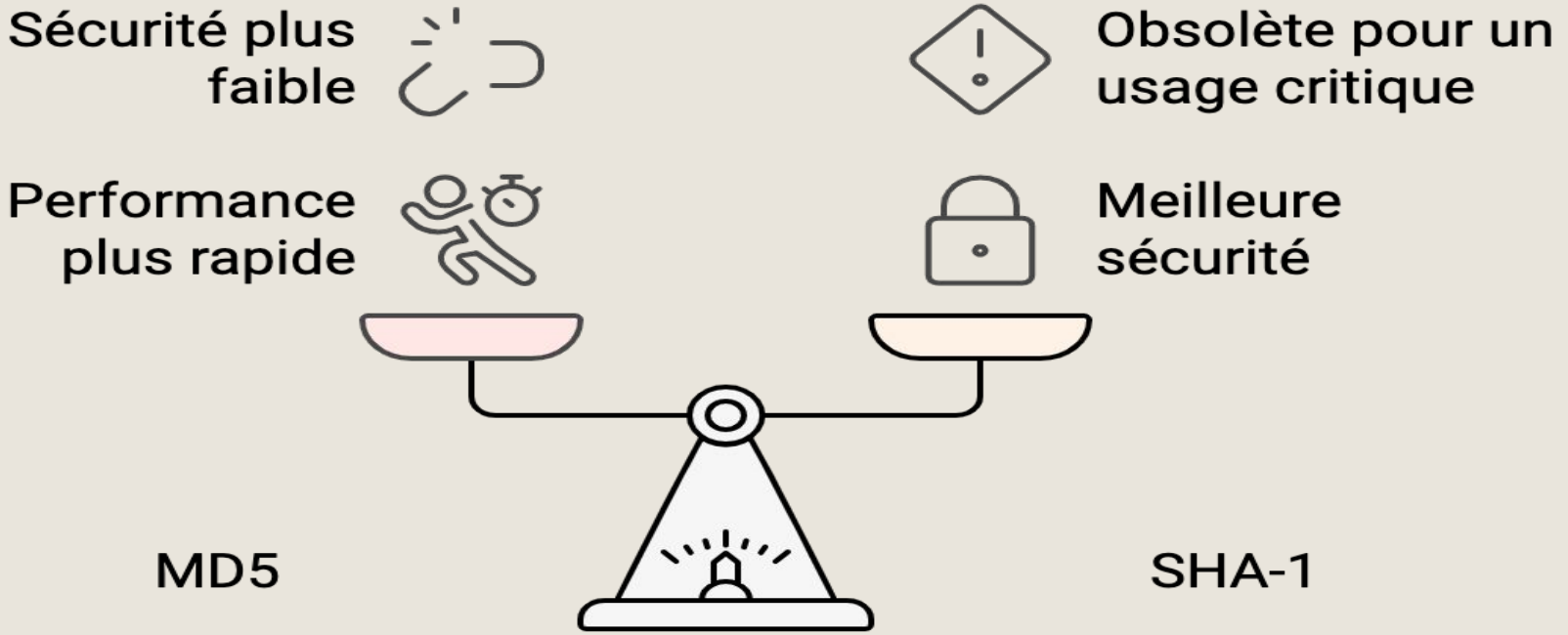
```
'C'+'o'+'u'+'r'+'s'+' '+'P'+'y'+'t'+'h'+'o'+'n
```

Toutefois, comme notre tableau est limité en nombre de cases (exemple 100 cases), cette somme dépasse 100. Pour régler le problème, on peut utiliser l'opérateur **modulo %**

```
def hachage(chaine):
    nombreHache = 0
    for char in chaine:
        nombreHache += ord(char)
    nombreHache %= 100
    return nombreHache
```

```
>>> hachage("Cours Python")
98
>>> hachage("ENSET-Media")
8
>>> hachage("F_I & Master")
60
```

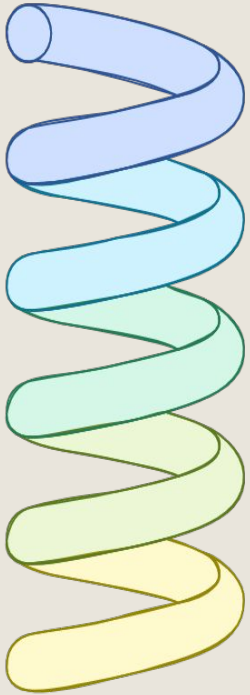
Algorithmes de hachage : MD5 et SHA-1



Comparaison de MD5 et SHA-1 en termes de performance et de sécurité.

Algorithmes de hachage : MD5 et SHA-1

Processus de Hachage MD5



- Entrée du Message
- Segmentation du Message
- Transformation des Blocs
- Génération du Hachage Final
- Utilisation du Hachage

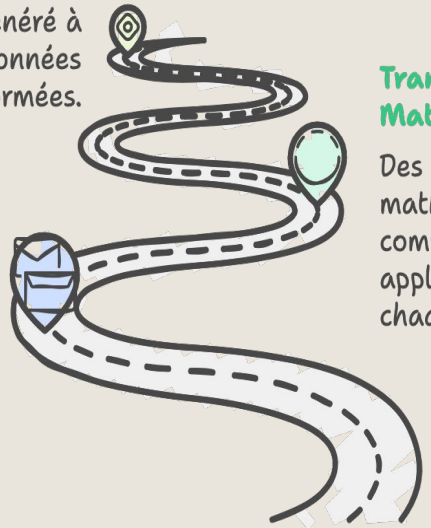
Processus de Hachage SHA-1

Génération du Hachage Final

Un hachage de 160 bits est généré à partir des données transformées.

Transformations Mathématiques

Des opérations mathématiques complexes sont appliquées à chaque bloc.



Division du Message

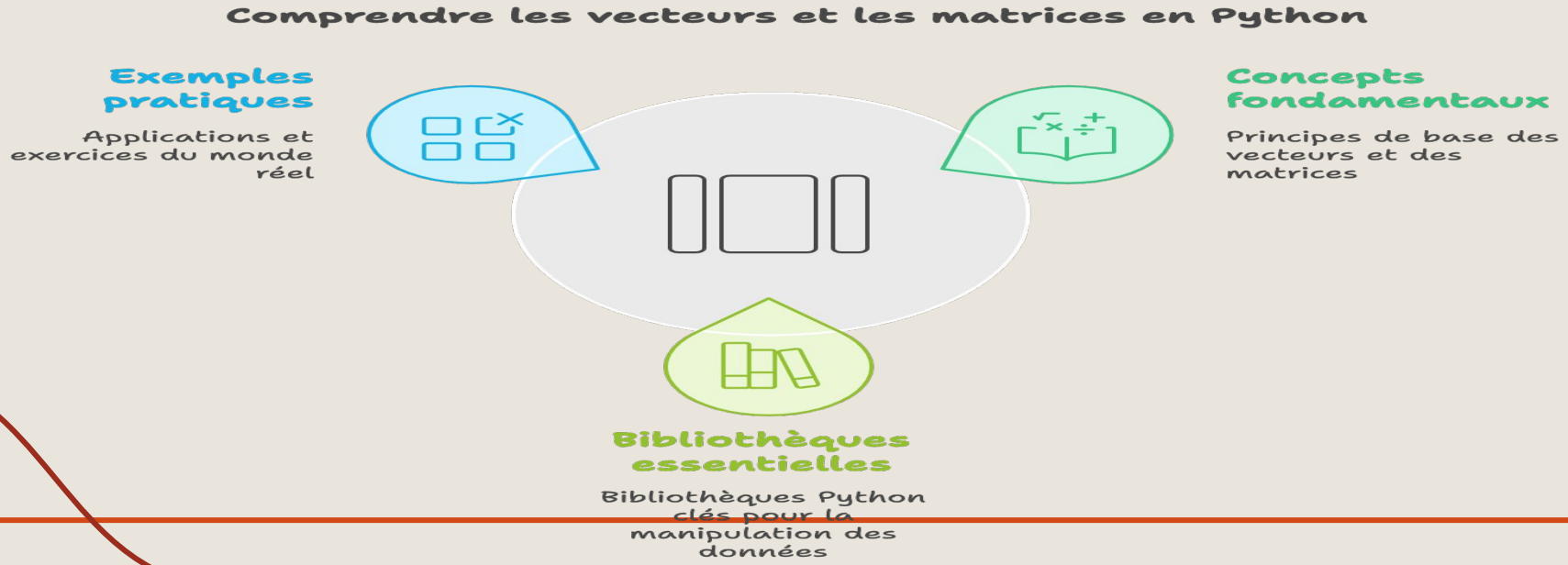
Le message d'entrée est divisé en blocs de 512 bits.

02

Vecteurs & Matrices

Vecteurs & Matrices

Les vecteurs et les matrices sont des structures de données fondamentales en mathématiques et en informatique, souvent utilisées dans des domaines tels que l'algèbre linéaire, le traitement d'images, et l'apprentissage automatique. En Python, nous avons plusieurs bibliothèques qui facilitent la manipulation de ces structures, notamment **NumPy.**



Vecteurs

Un vecteur peut être considéré comme une liste de nombres. Avec NumPy, vous pouvez créer un vecteur comme suivants :

A la main v = np .array ([1. ,2. ,3.])

Pas constant : arange

```
np.arange(0,10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

np.arange(3,0,-0.5) # le pas peut etre negatif
array([ 3. ,  2.5,  2. ,  1.5,  1. ,  0.5])

np.arange(2,3,0.2) #(3-2)/0.2 est entier
array([ 2. ,  2.2,  2.4,  2.6,  2.8])

np.arange(2,3.2,0.2) #(3.2-2)/0.2 n'est pas entier
array([ 2. ,  2.2,  2.4,  2.6,  2.8,  3. ,  3.2])
```

Pas constant : linspace

```
np.linspace(2.0, 3.0, num=5)
array([ 2.  ,  2.25,  2.5 ,  2.75,  3.  ])

np.linspace(2.0, 3.0, num=5, endpoint=False)
array([ 2. ,  2.2,  2.4,  2.6,  2.8])

x,pas=np.linspace(2.0, 3.0, num=5, retstep=True)
x
array([ 2.  ,  2.25,  2.5 ,  2.75,  3.  ])

pas
0.25
```

Vecteurs -EXTRACTION DES VALEURS

L'accès aux éléments d'un vecteur est identique à ceux d'une liste

```
x  = np.arange(1,5);
x[0]                    # Premier element
1

x[:]                    # Tous les elements
array([1, 2, 3, 4])

x[-1]                   # Dernier element
4

x[-2]                   # Avant-dernier element
3
```

```
x[1:3]                  # L'increment est 1 par defaut
array([2, 3])

x[-3:-1]                # L'increment est 1 par defaut
array([2, 3])

x[2:]                   # On prend jusqu'au dernier
array([3, 4])

x[1::2]                 # L'incr\'ement vaut 2
array([2, 4])

x1[::-1]                # Lecture de x a l'envers
```

Vecteurs -Taille et Opérations sur les vecteurs

```
a=np.zeros(5)
a
array([0., 0., 0., 0., 0.])

np.alen(a)
5

a.size
5

a.shape
(5,)
```

np.sum(v)	additionne tous les éléments du vecteur v.
np.prod(v)	multiplie tous les éléments du vecteur v.
np.max(v)	calcule le plus grand élément du vecteur v.
np.min(v)	calcule le plus petit élément du vecteur v.

Matrices

Une matrice est une collection de vecteurs, qu'on peut la définir comme suivants :

A la main M = np .array ([[1 ,2 ,3] ,[4 ,5 ,6]])

Matrices particulières

`np.eye(n)`	matrice identité de taille n.
`np.zeros((n,m))`	matrice de taille n×n ou n×m remplie de zeros.
`np.ones((n,m))`	matrice de taille n×n ou n×m remplie de uns.
`np.random.rand(n,m)`	matrice de taille n×n ou n×m remplie de nombres aléatoires.
`np.diag(v)`	matrice dont la diagonale est le vecteur v.

Matrice -Informations sur les matrices

Taille des matrices:

```
E= =np.array([[1,2],[3,4]])
n,m=E.shape
```

Type des éléments de la matrice:

```
a = np.array([1, 2, 3], int)
a.dtype
dtype('int64')
```

Matrice - Informations et Accès aux éléments d'une matrices

Taille :

```
E= =np.array([[1,2],[3,4]])
n,m=E.shape
```

Type des éléments :

```
a = np.array([1, 2, 3], int)
a.dtype
dtype('int64')
```

Accès aux éléments

```
A=np.array([np.arange(0,3),np.arange(0,3)*2])
array([[0, 1, 2],
       [0, 2, 4]])

A[1,:] # Acces \'a la 2eme ligne
array([0, 2, 4])
```

Matrice – Opérations sur les matrices

Transposition:

```
A=np.array([[1,2,3],[4,5,6]])
B=np.transpose(A);
```

Opérations élémentaires

- Les opérations élémentaires comme les additions, multiplications, divisions et puissances, se font terme à terme.

```
A=np.array([[1,2,3],[1,1,1]])
B=np.array([[1,1,1],[3,2,1]])
A*B
array([[1, 2, 3],
       [3, 2, 1]])
```

Algèbre linéaire

`np.dot(A,B)`	Produit des matrices A et B
`np.vdot(x,y)`	Produit scalaire des vecteurs x et y
`np.linalg.inv(A)`	Inverse de A
`np.linalg.solve(A,b)`	Résolution de $AX = b$
`np.linalg.det(A,b)`	déterminant de A
`np.trace(A)`	trace de A
`np.rank(A)`	rang de A

Autres opérations

```
A=np.array([[1,10,3],[5,2,7]])
np.amax(A,axis=0)  # Maximum sur chaque colonne
array([ 5, 10,  7])
np.amax(A,axis=1)  # Maximum sur chaque ligne
array([10,  7])
```

Les fonctions sum et prod fonctionnent de la même manière.

Matrice – Manipulation des lignes et colonnes des matrices

Suppression de ligne ou colonne:

```
A=np.array([[1,2,3],[1,1,1],[2,2,2]]);
B=np.delete(A,0,axis=0);B  # supprime la 1ere ligne
array([[1, 1, 1],
       [2, 2, 2]])
```

Ajout de ligne ou

```
np.insert(A,1,np.array([5, 5, 5]),axis=0)
array([[1, 2, 3],
       [5, 5, 5],
       [1, 1, 1],
       [2, 2, 2]])
```

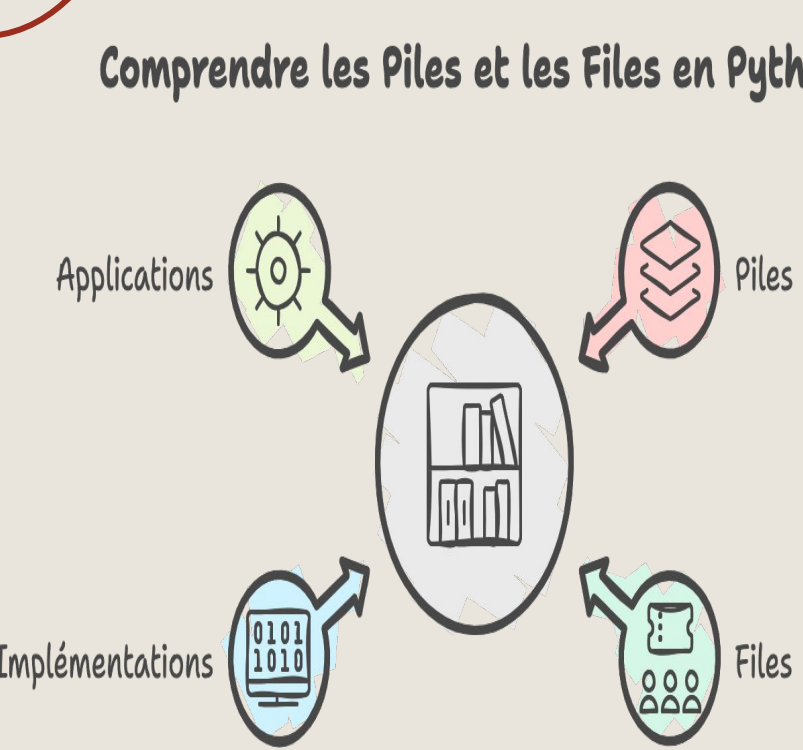
Redimensionnement

```
a = np.array(range(6)); a
array([0, 1, 2, 3, 4, 5])

a.reshape(2,3)
array([[0, 1, 2],
       [3, 4, 5]])
```

répéter une

```
A=np.array([[1,2],[3,4]])
np.tile(A,[2,1])
array([[1, 2],
       [3, 4],
       [1, 2],
       [3, 4]])
```



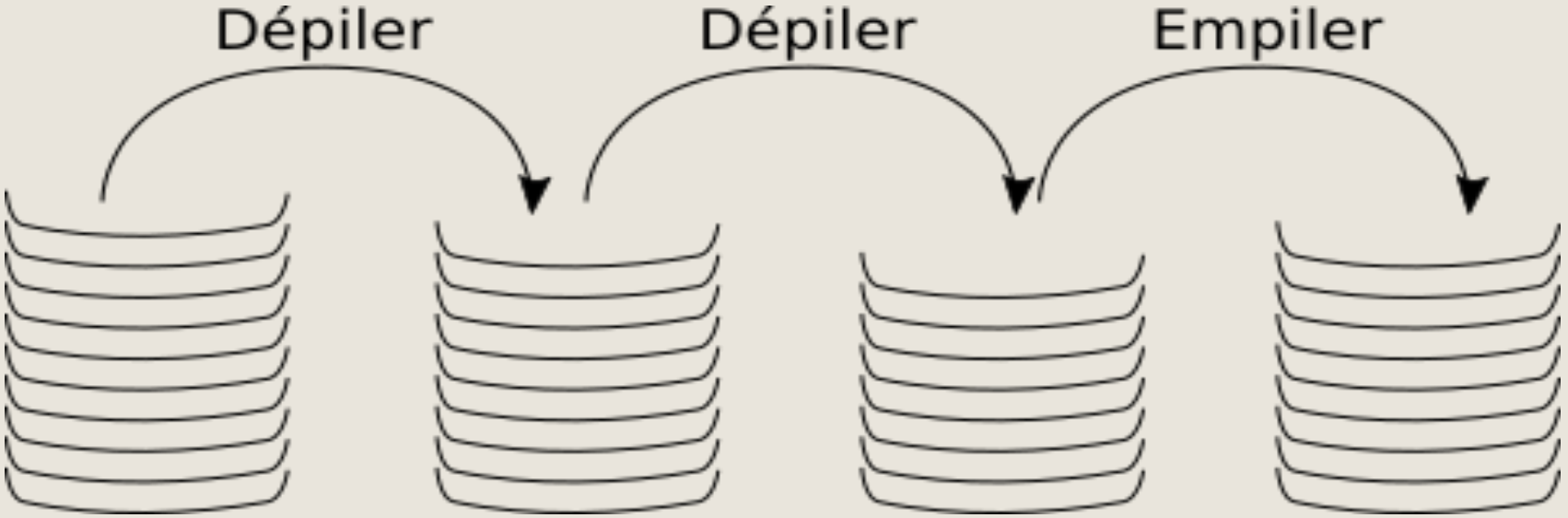
03

Piles et files

Deux structures de données essentielles en informatique. Les piles (ou "stacks") fonctionnent selon le principe LIFO (Last In, First Out), tandis que les files (ou "queues") suivent le principe FIFO (First In, First Out)

Les Piles

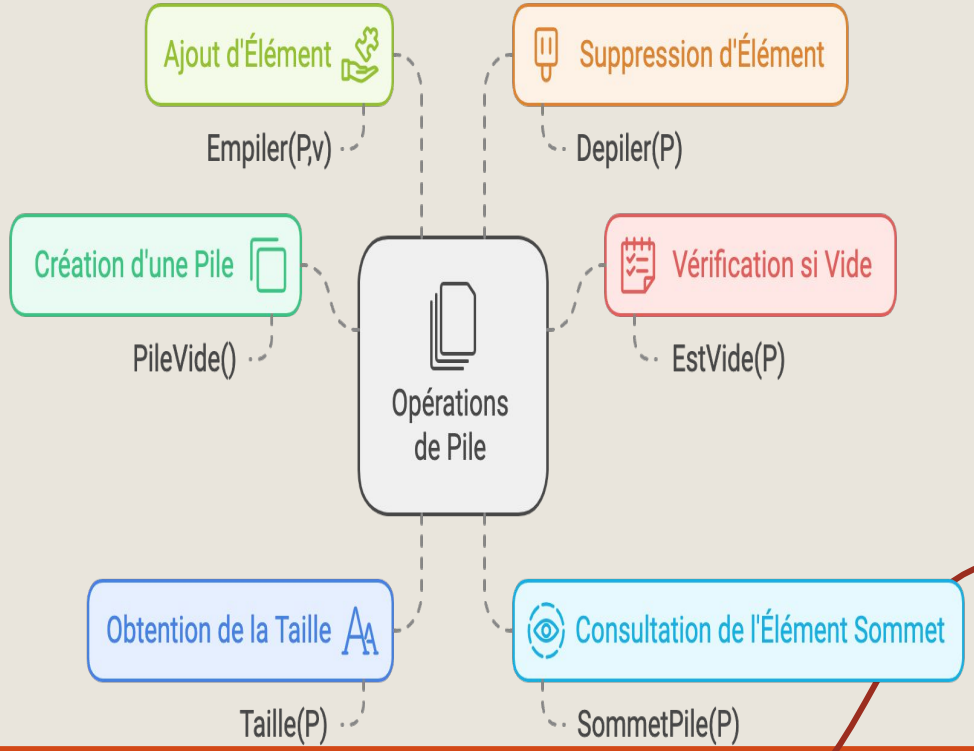
Une pile est une collection d'éléments où l'ajout et la suppression d'éléments se font uniquement à partir du sommet de la pile. Cela signifie que le dernier élément ajouté est le premier à être retiré.



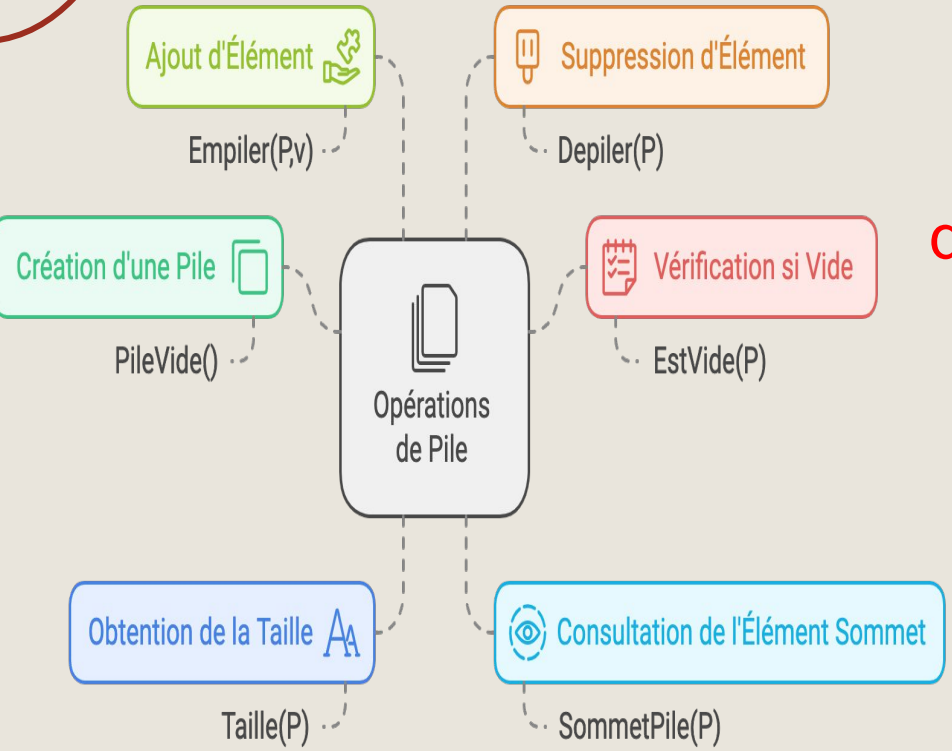
Les Piles- Les primitives

Une pile est une collection d'éléments manipuler par une liste réduite de fonctions ou méthodes appelées communément **"les primitives".**

- PileVide() : crée et renvoie une pile vide ;
- EstVide(P) : renvoie vrai si la pile est vide, faux sinon ;
- Taille(P) : renvoie le nombre d'éléments dans la pile ;
- SommetPile(P) : renvoie l'élément sommet de la pile P ;
- Empiler(P,v) : ajoute au sommet de la pile P l'´élément v ;
- Depiler(P) : supprime de la pile le sommet.



Les Piles- Les primitives

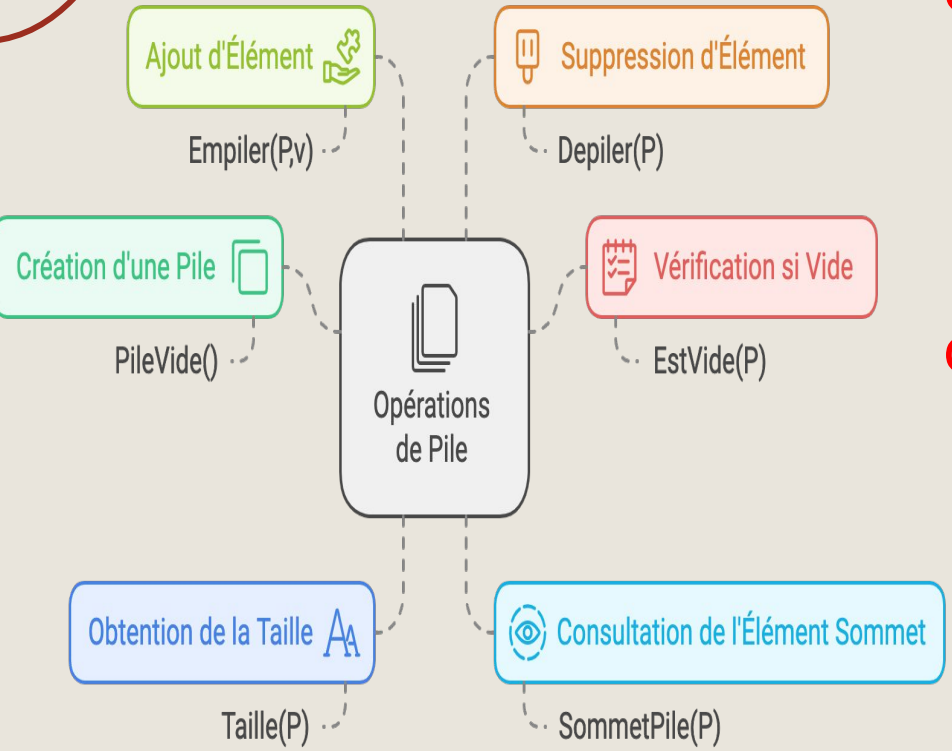



```
def PileVide() : return []
```

```
def EstVide(P) :
    if len(P)==0 : return True
    else :         return False
```

```
def Taille(P) :    return len(P)
```

Les Piles- Les primitives



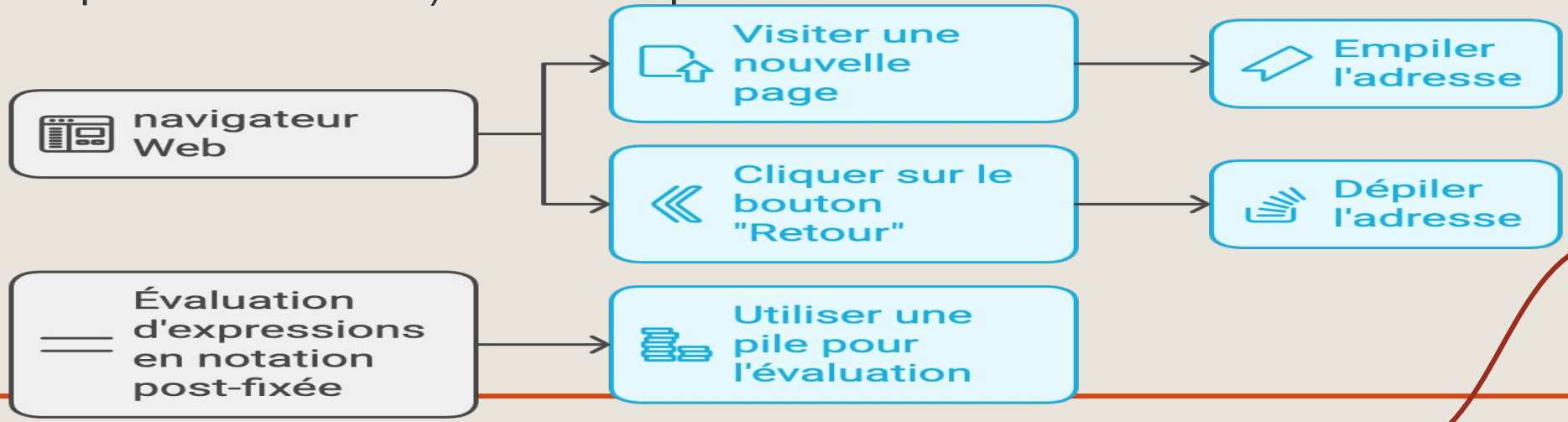

```
def Empiler(P,v) :
    P.append(v)
    return P
```

```
def Depiler(P) :
    if len(P)==0 :
        print(”Erreur : pile vide”)
    else :
        P.pop()
        return P
```

Les Piles- Les applications

De nombreuse applications utilisent une pile, on peut citer :

- ✔ Dans un navigateur web, une pile sert à mémoriser les pages Web visitées. L'adresse de chaque nouvelle page visitée est empilée et l'utilisateur dépile l'adresse de la page précédente en cliquant le bouton Afficher la page précédente .
- ✔ L'évaluation des expressions mathématiques en notation post-fixée (ou polonaise inverse) utilise une pile.



Les Piles- Les applications

De nombreuse applications utilisent une pile, on peut citer :

- ✔ La fonction Annuler la frappe (en anglais Undo) d'un traitement de texte mémorise les modifications apportées au texte dans une pile.
- ✔ Vérification de parenthésage d'une chaine de caractères ;
- ✔ La récursivité (une fonction qui fait appel à elle-même) ;
- ✔ etc.

Les Files

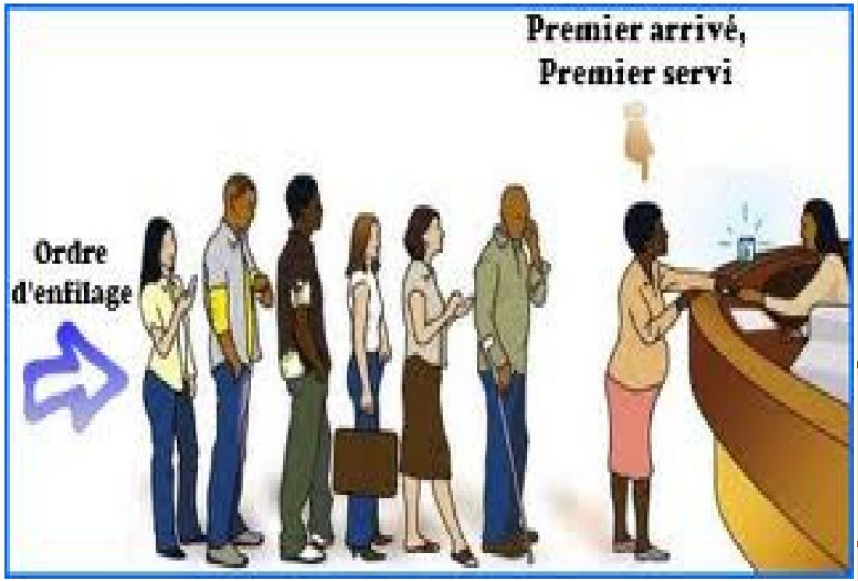
Définition :

- Une file (queue en anglais) est une structure de données dans laquelle l'insertion se fait à la fin et la suppression d'un élément s'effectue à partir de début de cette structure.
- Le fonctionnement ressemble à une file d'attente : les premières personnes arrivées sont les premières personnes à sortir de la file.

Le mécanisme FIFO (first in, first out)

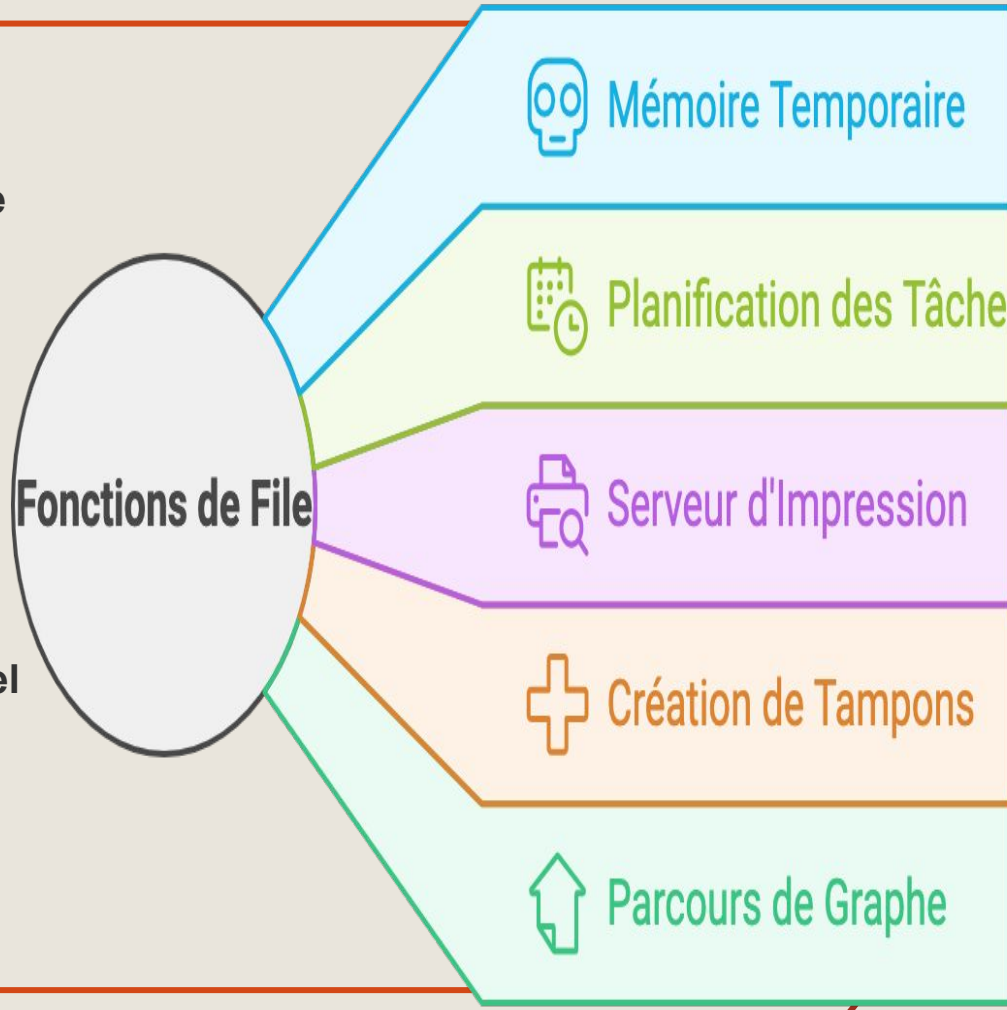
Une file permet de modéliser un système régi par le m´mécanisme "premier arrivé premier sorti" ; appelé souvent FIFO (first in, first out)

- L'opération enfiler ajoute un nouvel élément à la fin de la file ;
- défiler retire l'´élément situé au début de la file.



Les Files - Les applications

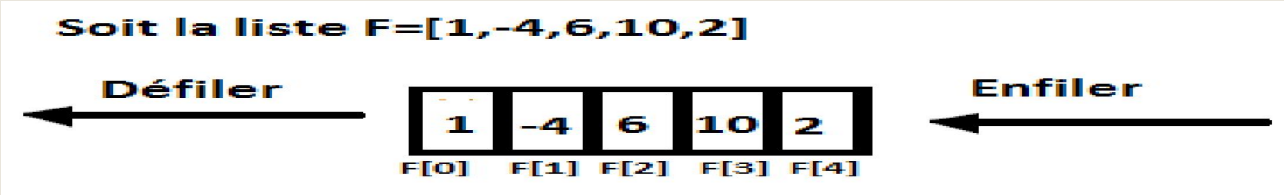
- En général, on utilise des files pour m´mémoriser temporairement des transactions qui doivent attendre pour être traitées ;
- Certains moteurs multitâches, dans un système d'exploitation, qui doivent accorder du temps-machine à chaque tâche, sans en privilégier aucune ;
- On utilise aussi des files pour créer toutes sortes de mémoires tampons (en anglais buffers).
- Les serveurs d'impression, qui doivent traiter les requetés dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue) ;
- Un algorithme de parcours en largeur utilise une file pour mémoriser les nœuds visités ;
- etc.



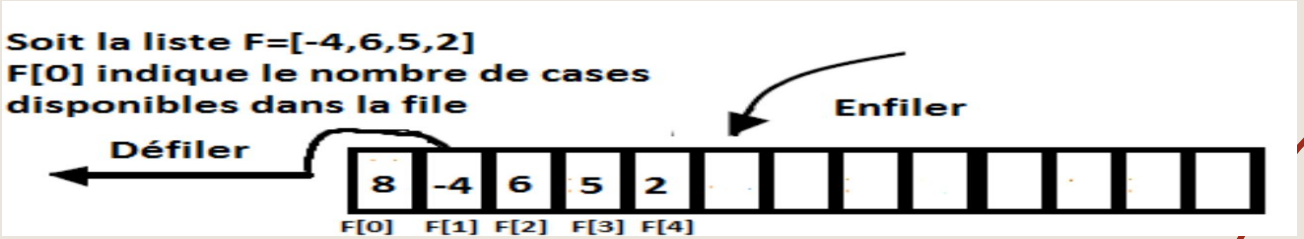
Les Files - Implémentation

En python, il existe deux façons pour implémenter une file avec une liste :

- **Soit on utilise une liste de taille finie pour réaliser une file ;**



- **Soit on utilise une liste de taille non finie pour réaliser une file.**

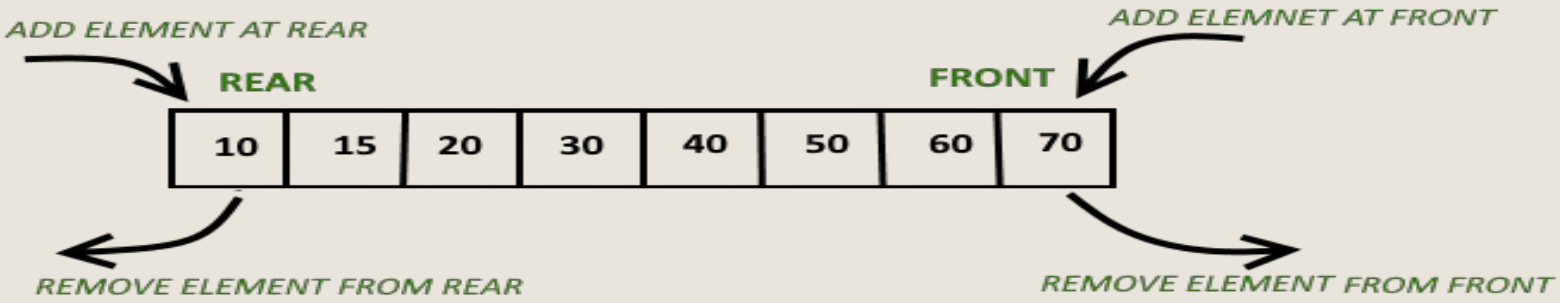


Les Files - Remarque

La liste n'est pas une structure très forte pour représenter une file ;
Pour cela, python possède une structure spécifique au File appelée deque.

```
>>>from collections import deque
>>>F=deque([10,15,20,30,40,50,60,70])
```

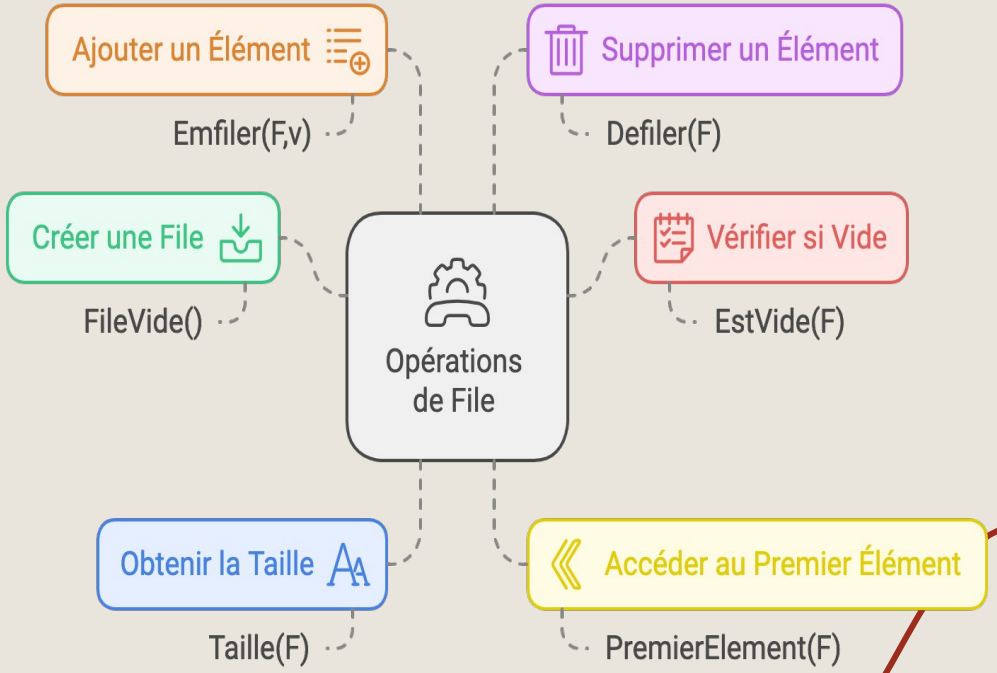
- **La fonction popleft permet de défiler et**
- **la fonction append permet d'enfiler.**



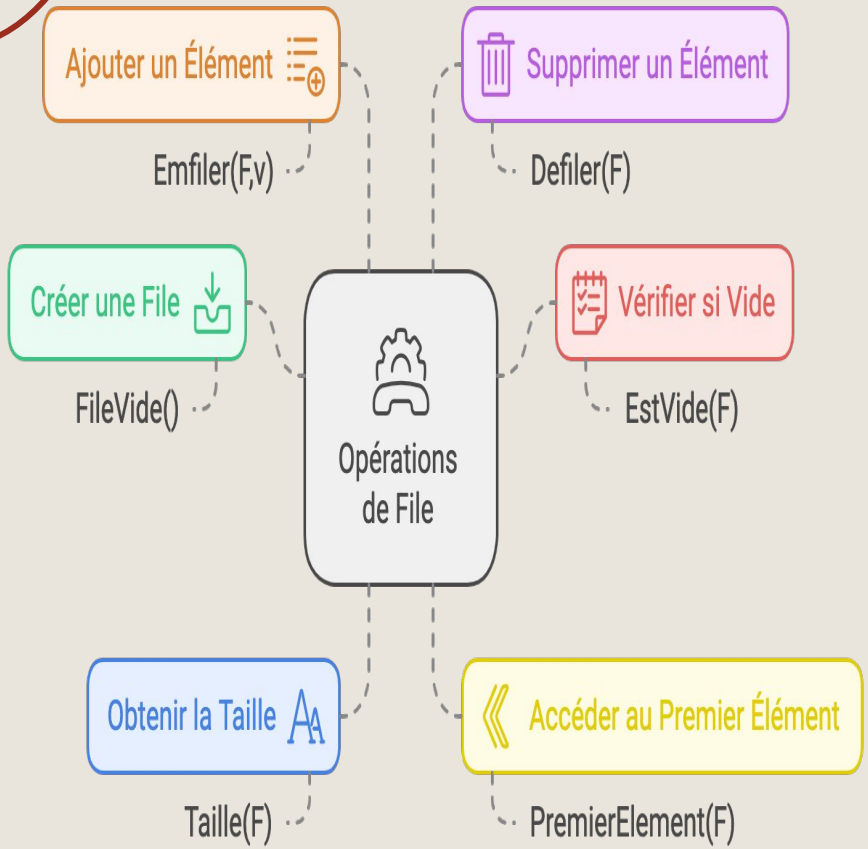
Les Files- Les primitives

Pour résoudre un problème donné qui repose sur la structure file, il est n´nécessaire de programmer un ensemble des primitives pour la gestion d'une file.

- FileVide() : crée et renvoie une file vide ;
- EstVide(F) : renvoie vrai si la file est vide, faux sinon ;
- Taille(F) : renvoie le nombre d'éléments dans la file ;
- PremierElement(F) : renvoie l'élément sommet de la file F ;
- Emfiler(F,v) : ajoute à la fin de la file F l' élément v ;
- Defiler(F) : supprime de la file le premier élément.



Les Piles- Les primitives

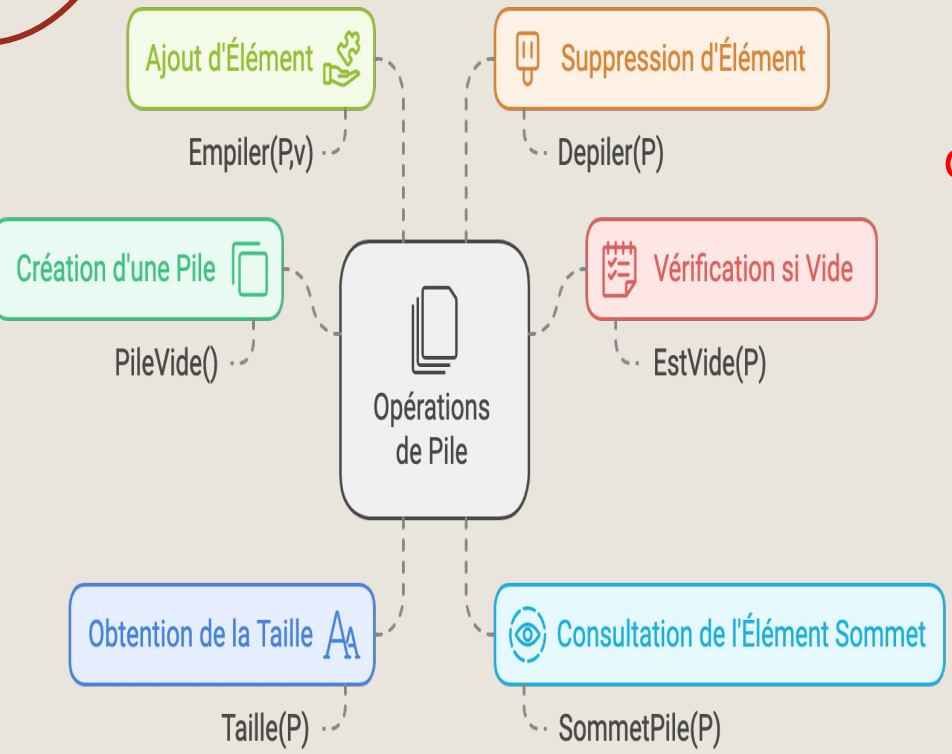



```
def FileVide() : return []
```

```
def EstVide(F) :
    if len(F)==0 : return True
    else :         return False
```

```
def Taille(F) :    return len(F)
```

Les Files- Les primitives




```
def Enfiler(F,v) :
        F.append(v)
        return F
def Defiler(F) :
      if len(F)==0 :
          print(”Erreur : pile vide”)
      else :
          F.pop(0) #F.remove(F[0])
          return F
def PremierElement(F) :
  if len(F)==0 :
       print(”file vide”)
       return None
  else :  return F[0]
```

04

Les fichiers

Non structurés & Structurés

Les fichiers

Une collection d'informations stockées sur une mémoire de masse (non volatile, capacité plus importante que la mémoire vive).

Types de fichiers

1. Organisation des données :
   - ✔ Structuré, les informations sont organisées d'une manière particulière qu'il faut respecter (ex. CSV) ;
   - ✔ Non-Structuré, les informations sont alignées à la suite (ex. fichier texte)
   - ✔ Structure non prédéfinie, un fichier XML par ex. est un fichier texte (manipulable avec un éditeur de texte) mais qui obéit à une organisation structurée
2. Mode d'accès aux données : accès indicé, utilisable comme un tableau ; accès séquentiel, lecture ou écriture pas à pas (ex. ligne par ligne dans un fichier texte)

Les fichiers - Lecture en bloc avec read()

```
#ouverture en lecture
f = open("voitures.txt","r") #lecture
s = f.read() #affichage
print("** contenu de s **")
print(s)
print("** fin contenu **")
#information sur s
print("type de s : ",type(s))
print("longueur de s : ", len(s))
#fermeture
f.close()
```

Fichier texte à lire : « voiture.txt »

megane
clio
twingo
safrane
laguna

```
** contenu de s **
megane
clio
twingo
safrane
laguna
vel satis
** fin contenu **
type de s :  <class 'str'>
longueur de s :  43
```

- open() permet d'ouvrir un fichier, en lecture ici avec l'option « r ». La fonction renvoie un objet de type fichier stocké dans f
- le curseur de fichier est placé sur la première ligne
- read() lit tout le contenu du fichier en bloc
- close() ferme le fichier (et donc le déverrouille)

Les fichiers - Lecture en bloc avec read()

```
#ouverture en lecture
f = open("voitures.txt","r") #lecture
s = f.read() #affichage
print("** contenu de s **")
print(s)
print("** fin contenu **")
#information sur s
print("type de s : ",type(s))
print("longueur de s : ", len(s))
#fermeture
f.close()
```

Fichier texte à lire : « voiture.txt »

megane
clio
twingo
safrane
laguna

```
** contenu de s **
megane
clio
twingo
safrane
laguna
vel satis
** fin contenu **
type de s :  <class 'str'>
longueur de s :  43
```

- open() permet d'ouvrir un fichier, en lecture ici avec l'option « r ». La fonction renvoie un objet de type fichier stocké dans f
- le curseur de fichier est placé sur la première ligne
- read() lit tout le contenu du fichier en bloc
- close() ferme le fichier (et donc le déverrouille)

Les fichiers - Lecture en bloc avec readlines()

```
filin = open("voiture.txt", "r")
lignes = filin.readlines()
print(lignes)
['megane\n', 'clio\n', 'twingo\n', 'safrane\n','laguna\n']
```

```
for ligne in lignes:
    print(ligne)
megane

clio

twingo

safrane

laguna
```

```
filin.close()
```

Les fichiers -Lecture en bloc avec readlines() & with

```
with open("voiture.txt", 'r') as filin:
    lignes = filin.readlines()
    for ligne in lignes:
        print(ligne)
```

```
megane

clio

twingo

safrane

laguna
```

Les fichiers -Lecture en bloc avec readline()

```
with open("voiture.txt", "r") as filin:
    ligne = filin.readline()
    while ligne != "":
        print(ligne)
        ligne = filin.readline()
```

```
megane

clio

twingo

safrane

laguna
```

Les fichiers - Itérations directes sur le fichier

```
with open("voiture.txt", "r") as filin:
    for ligne infilin
        print(ligne)
```

```
megane

clio

twingo

safrane

laguna
```

Les fichiers - Écriture dans un fichier

```
animaux2 = ["poisson", "abeille", "chat"]
with open("animaux2.txt", "w") as filout:
    for animal in animaux2:
        filout.write(animal)
7
7
4
```

```
animaux2 = ["poisson", "abeille", "chat"]
with open("animaux2.txt", "w") as filout:
    for animal in animaux2:
        filout.write(f"{animal}\n")
8
8
5
```

Les fichiers CSV- Extraction des données

```
import csv
with open('Tableur1.csv',newline='') as f: #Ouverture du fichier CSV
 tableau=[]
 lire=csv.reader(f) #chargement des lignes du fichier csv
 print('',end='\n')
 print('Affichage des lignes du tableau',end='\n')
 for ligne in lire: #Pour chaque ligne...
    print(ligne, end='\n') #...affichage de la ligne dans la console
    tableau.append(ligne) #...on ajoute la ligne dans la liste ...
 #...de liste nommée tableau
print(tableau) # Affichage du tableau
print(tableau[1]) # Affichage de la deuxième ligne
print(type(tableau[1])) # Type de la variable tableau[1]
print(tableau[1][2]) # Affichage de la variable tableau[1][2]
print(type(tableau[1][2])) # Type de la variable tableau[1][2]
```

Les fichiers CSV- Extraction des données en dictionnaire

```
#ouverture en lecture
f = open("personnes.csv","r")

#importation du module csv
import csv

#lecture
lecteur = csv.DictReader(f,delimiter=";")

#affichage
for ligne in lecteur:
    print(ligne)

#fermeture
f.close()
```

Chaque ligne est un **Dict**

```
{'poids': '65', 'age': '45', 'taille': '178'}
{'poids': '89', 'age': '58', 'taille': '176'}
{'poids': '85', 'age': '35', 'taille': '194'}
```