# Piles et Files

Yves Josse / Nicolas Markey

Lycée Chateaubriand – PCSI 3

Année 2024-2025

# Piles et Files : définitions et exemples

## Piles

- Une pile (*stack*) est une structure de données reposant sur le principe du « dernier arrivé, premier sorti » (*LIFO : Last In, First Out*). Les opérations autorisés sont l'insertion en haut de la pile et la suppression du dernier entré.
- Exemples dans la vie courante : pile, d'assiettes, historique des pages dans un navigateur, annulation des dernières opérations, exécution d'une fonction récursive...

## Files

- Une file (*queue*) est une structure de données reposant sur le principe du « premier entré, premier sorti » (*FIFO : First In, First Out*). Les opérations autorisés sont l'insertion au début de la file et la suppression du début de la file.
- Exemples dans la vie courante : files d'attente...

# Opérations de bases sur les piles

## Opérations disponibles sur les piles

Les seules opérations disponibles avec cette structure de données sont :

- Créer une pile vide ;
- Tester si une pile est vide ;
- Ajouter un élément au sommet (empiler, Push) ;
- Retirer l'élément du sommet (dépiler, Pop) et renvoyer cet élément.

On programme ici les 4 opérations à l'aide de listes sous python :

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

```
def TestePileVide(P):
    return P==[]
```

```
def Empile(P,x):
    return P.append(x)
```

```
def DepileRenvoie(P):
    return P.pop()
```

# Manipulations de piles

## Quelques manipulations sur les piles

Dans la suite on ne pourra utiliser que les 4 opérations précédentes (pas les fonctions habituelles sur les listes (`len(P)`, `L[i]`, slicing...).

1. Écrire la fonction `hauteur(P)` qui calcule le nombre d'éléments de la pile $P$ (on a le droit de détruire la pile).
2. Écrire la fonction `popsecond(P)` qui enlève le deuxième élément en partant du sommet en laissant le sommet en place.
3. Écrire la fonction `popfond(P)` qui enlève l'élément au fond de la pile, le reste de la pile restant inchangé.
4. Écrire une fonction `copier_coller(P)` qui admet comme argument une pile $P$. Cette fonction renvoie une copie sans modifier la pile initiale.
5. Écrire une fonction `inverser` qui admet comme argument une pile $P$. Cette fonction renvoie une pile $Q$ contenant les éléments de $P$ dans l'ordre inverse, sans modifier la pile initiale.

# Manipulations sur les piles

# Opérations de base sur les files

## Opérations disponibles sur les files

Les seules opérations disponibles avec cette structure de données sont :

- Créer une file vide ;
- Tester si une file est vide ;
- Enfiler : ajouter un élément à la queue de la file (arrière de la file d'attente) ;
- Défiler : supprimer l'élément situé à la tête de la file (au début de la file d'attente).

On programme ici les 4 opérations à l'aide de listes sous python :

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

```
def TesteFileVide(F):
    return F==[]
```

```
def Enfiler(F,x):
    return F.append(x)
```

```
def DepileRenvoie(F):
    return F.pop(0)
```

# Difficultés entre files et listes : utilisation de deques

## Problème du retrait d'un élément

L'utilisation de `P.pop(0)` entraîne un complexité linaire en $(n)$ avec $n$ la taille de la liste. Cette opération nécessite de modifier les indices de tous les éléments de la file. Le format liste n'est donc pas le plus adaptée à cette structure de données.

## Utilisation de deques

L'instruction `from collections import deque` permet de manipuler des deques.

- `D=deque()` permet de créer une deque vide ;
- `len(D)==0` retourne `True` si la deque est vide, `False` sinon ;
- `D.append(x)` ajoute un élément à l'extrémité droite ;
- `D.appendleft(x)` ajoute un élément à l'extrémité gauche ;
- `x=D.pop()` supprime et renvoie l'élément à l'extrémité droite ;
- `x=D.popleft()` supprime et renvoie l'élément à l'extrémité gauche.