# Piles et Files

Ivan Noyer

Lycée Leconte de Lisle

1 Piles
- Généralités
- Implémentation

2 Files

# Crédits

- Informatique pour tous en classes préparatoires aux grande écoles (Eyrolles)

# Crédits

- Informatique pour tous en classes préparatoires aux grande écoles (Eyrolles)
- Wikipédia : http://fr.wikipedia.org/wiki/Pile_%28informatique%29 et (plus complet mais en anglais) http://en.wikipedia.org/wiki/Stack_%28abstract_data_type%29

# Crédits

- Informatique pour tous en classes préparatoires aux grande écoles (Eyrolles)
- Wikipédia : http://fr.wikipedia.org/wiki/Pile_%28informatique%29 et (plus complet mais en anglais) http://en.wikipedia.org/wiki/Stack_%28abstract_data_type%29
- Un chapitre de la documentation officielle de Python : http://www.afpy.org/doc/python/2.7/tutorial/datastructures.html.

# Crédits

- Informatique pour tous en classes préparatoires aux grande écoles (Eyrolles)
- Wikipédia : http://fr.wikipedia.org/wiki/Pile_%28informatique%29 et (plus complet mais en anglais) http://en.wikipedia.org/wiki/Stack_%28abstract_data_type%29
- Un chapitre de la documentation officielle de Python : http://www.afpy.org/doc/python/2.7/tutorial/datastructures.html.
- La doc sur la classe `deque` en Python : https://docs.python.org/2/library/collections.html#collections.deque

1 Piles
- Généralités
- Implémentation

2 Files

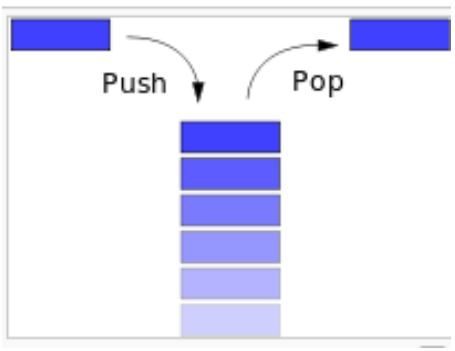
1 Piles
- Généralités
- Implémentation

2 Files

# Pile

- Une *pile* (en anglais stack) est une *structure de données* fondée sur le principe « dernier arrivé, premier sorti » (ou LIFO pour Last In, First Out), ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés.

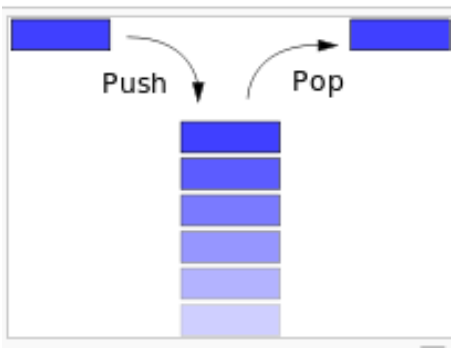
FIGURE – Empiler et dépiler



# Pile

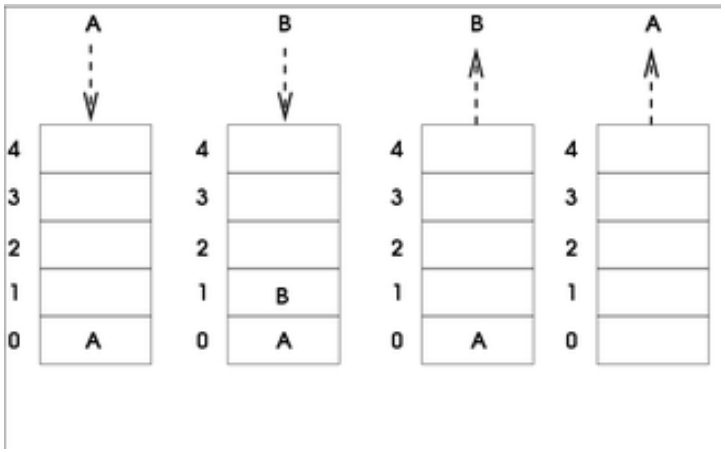
- Une *pile* (en anglais stack) est une *structure de données* fondée sur le principe « dernier arrivé, premier sorti » (ou LIFO pour Last In, First Out), ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés.
- Le fonctionnement est celui d'une pile d'assiettes : on ajoute des assiettes sur la pile, et on les récupère dans l'ordre inverse, en commençant par la dernière ajoutée.

FIGURE – Empiler et dépiler



# Exemple

FIGURE – Empilement de $A$ puis $B$, dépilement de $B$ puis $A$ (LIFO)



# Historique

- Turing 1946. Description théorique d'appel et de retour de sous-routines.

# Historique

- Turing 1946. Description théorique d'appel et de retour de sous-routines.
- Klaus Samelson et Friedrich L. Bauer de l'université Technique de Münich généralisent l'idée en 1955 en présentant une modélisation du fonctionnement des ordinateurs à base de piles.

# Historique

- Turing 1946. Description théorique d'appel et de retour de sous-routines.
- Klaus Samelson et Friedrich L. Bauer de l'université Technique de Münich généralisent l'idée en 1955 en présentant une modélisation du fonctionnement des ordinateurs à base de piles.
- Même concept, indépendamment développé par l'australien Charles Leonard Hamblin en 1957.

# Les primitives indispensables

- « Empiler » : ajoute un élément sur la pile. Terme anglais correspondant : « Push ».

# Les primitives indispensables

- « Empiler » : ajoute un élément sur la pile. Terme anglais correspondant : « Push ».
- « Dépiler » : enlève un élément de la pile et le renvoie. Terme anglais correspondant : « Pop ».

# Les primitives indispensables

- « Empiler » : ajoute un élément sur la pile. Terme anglais correspondant : « Push ».
- « Dépiler » : enlève un élément de la pile et le renvoie. Terme anglais correspondant : « Pop ».
- « La pile est-elle vide ? » : renvoie vrai si la pile est vide, faux sinon. « isempty »

# Autres primitives

Elles peuvent être obtenues à partir des 3 premières :

- « Nombre d'éléments de la pile » : renvoie le nombre d'éléments dans la pile. Complexité : Selon les implémentations, peut être fait soit en temps constant soit en temps linéaire.

# Autres primitives

Elles peuvent être obtenues à partir des 3 premières :

- « Nombre d'éléments de la pile » : renvoie le nombre d'éléments dans la pile. Complexité : Selon les implémentations, peut être fait soit en temps constant soit en temps linéaire.
- « Quel est l'élément de tête ? » : renvoie l'élément de tête sans le désempiler. Terme anglais correspondant : « Peek ».

# Autres primitives

Elles peuvent être obtenues à partir des 3 premières :

- « Nombre d'éléments de la pile » : renvoie le nombre d'éléments dans la pile. Complexité : Selon les implémentations, peut être fait soit en temps constant soit en temps linéaire.
- « Quel est l'élément de tête ? » : renvoie l'élément de tête sans le désempiler. Terme anglais correspondant : « Peek ».
- « Vider la pile » : dépiler tous les éléments. Complexité : Selon l'implémentation, cela peut être fait en temps constant ou linéaire. Terme anglais correspondant : « Clear ».

# Autres primitives

Elles peuvent être obtenues à partir des 3 premières :

- « Nombre d'éléments de la pile » : renvoie le nombre d'éléments dans la pile. Complexité : Selon les implémentations, peut être fait soit en temps constant soit en temps linéaire.
- « Quel est l'élément de tête ? » : renvoie l'élément de tête sans le désempiler. Terme anglais correspondant : « Peek ».
- « Vider la pile » : dépiler tous les éléments. Complexité : Selon l'implémentation, cela peut être fait en temps constant ou linéaire. Terme anglais correspondant : « Clear ».
- « Dupliquer l'élément de tête » et « Échanger les deux premiers éléments » : existe sur les calculatrices fonctionnant en notation polonaise inverse (style HP). Termes anglais correspondants : « Dup » et « Swap » respectivement.

# Un peu de maths

En matière de structures abstraites, on peut considérer qu'une pile est un *monoïde libre*, c'est-à-dire un ensemble muni d'une loi de composition interne (la concaténation) associative et possédant un élément neutre (la pile vide).

# Applications

- La plupart des microprocesseurs gèrent nativement une pile. Elle correspond alors à une zone de la mémoire, et le processeur retient l'adresse du dernier élément.

# Applications

- La plupart des microprocesseurs gèrent nativement une pile. Elle correspond alors à une zone de la mémoire, et le processeur retient l'adresse du dernier élément.
- 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 désempile l'adresse de la page précédente en cliquant le bouton « Afficher la page précédente ».

# Applications

- La plupart des microprocesseurs gèrent nativement une pile. Elle correspond alors à une zone de la mémoire, et le processeur retient l'adresse du dernier élément.
- 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 désempile 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.

# Applications

- La fonction « Annuler la frappe » (en anglais Undo) d'un traitement de texte mémorise les modifications apportées au texte dans une pile.

# Applications

- La fonction ≪ Annuler la frappe ≫ (en anglais Undo) d'un traitement de texte mémorise les modifications apportées au texte dans une pile.
- Un algorithme de recherche *en profondeur d'abord* utilise une pile pour mémoriser les nœuds visités.

# Applications

- La fonction « Annuler la frappe » (en anglais Undo) d'un traitement de texte mémorise les modifications apportées au texte dans une pile.
- Un algorithme de recherche *en profondeur d'abord* utilise une pile pour mémoriser les nœuds visités.
- Inversion d'un tableau ou d'une chaîne de caractères.

# Applications

- La fonction « Annuler la frappe » (en anglais Undo) d'un traitement de texte mémorise les modifications apportées au texte dans une pile.
- Un algorithme de recherche *en profondeur d'abord* utilise une pile pour mémoriser les nœuds visités.
- Inversion d'un tableau ou d'une chaîne de caractères.
- Les algorithmes récursifs admis par certains langages (`LISP`, `C`, `Python`,etc... ) utilisent implicitement une pile d'appel.

1 Piles
- Généralités
- Implémentation

2 Files

# Implémentation en Python

## Avec des listes Python

- On utilise les *listes Python* qui sont en fait des *tableaux* pouvant grossir.

# Implémentation en Python

## Avec des listes Python

- On utilise les *listes Python* qui sont en fait des *tableaux* pouvant grossir.
- Pour ajouter un élément sur la pile, utiliser la méthode `append()`.

# Implémentation en Python

## Avec des listes Python

- On utilise les *listes Python* qui sont en fait des *tableaux* pouvant grossir.
- Pour ajouter un élément sur la pile, utiliser la méthode `append()`.
- Pour récupérer l'objet au sommet de la pile, utilisez la méthode `pop()`, sans indicateur de position.

# Implémentation en Python

## Avec des listes Python

- On utilise les *listes Python* qui sont en fait des *tableaux* pouvant grossir.
- Pour ajouter un élément sur la pile, utiliser la méthode `append()`.
- Pour récupérer l'objet au sommet de la pile, utilisez la méthode `pop()`, sans indicateur de position.
- Pour savoir si une liste est vide, comparer avec `[]` ou vérifier que sa longueur est nulle..

# En Python

## Avec des listes Python

```
1 stack=[10,20]
2 print(stack)
3 stack.append(6)#empiler
4 print(stack)
5 isempty = lambda t : t == [] #indicateur pile vide
6 n=len(stack)
7 for i in range(n):
8     print(stack.pop(), end = ' ; ')#dépiler
9 print(isempty(stack)) #a-t-on une pile vide ?
```

```
[10, 20]
[10, 20, 6]
6 ; 20 ; 10 ; True
```

# Implémentation en Python

## Avec la bibliothèque `collections`

- On utilise la fonction `deque` de `collections`.

# Implémentation en Python

## Avec la bibliothèque `collections`

- On utilise la fonction `deque` de `collections`.
- Pour ajouter un élément sur la pile, utiliser la méthode `append()` (comme pour les listes).

# Implémentation en Python

## Avec la bibliothèque `collections`

- On utilise la fonction `deque` de `collections`.
- Pour ajouter un élément sur la pile, utiliser la méthode `append()` (comme pour les listes).
- Pour récupérer l'objet au sommet de la pile, utilisez la méthode `pop()` (comme pour les listes).

# Implémentation en Python

## Avec la bibliothèque `collections`

- On utilise la fonction `deque` de `collections`.
- Pour ajouter un élément sur la pile, utiliser la méthode `append()` (comme pour les listes).
- Pour récupérer l'objet au sommet de la pile, utilisez la méthode `pop()` (comme pour les listes).
- Pour savoir si une liste est vide, dépiler. Si une exception est soulevée, c'est que la pile est vide. Sinon, elle n'est pas vide.

# Implémentation en Python

## Avec la bibliothèque `collections`

```
1 from collections import deque
2 myStack = deque()#création d'une pile vide
3
4 myStack.append('a')#ajout d'un élément
5 myStack.append('b')#ajout d'un élément
6 myStack.append('c')#ajout d'un elt
7
8 print(myStack)
```

On obtient `deque(['a', 'b', 'c'])`

# Implémentation en Python

## Avec la bibliothèque `collections`

```
1 myStack.pop()#dépiler
2 myStack.pop()#dépiler
3 myStack.pop()#dépiler
4 print(myStack)
```

On obtient `deque([])`

```
1 myStack.pop() # IndexError: pop from an empty deque
```

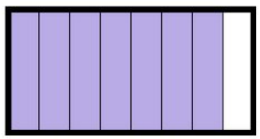
`len(q)` donne la longueur de `q` en $O(1)$.

# Deux philosophies différentes

## Implémentation par des listes PYTHON

- Créer une liste PYTHON consiste à choisir un bloc de cases mémoires contiguës. Un item de la liste est stockés juste à côté de son prédecesseur.

FIGURE – Exemple de zone mémoire non pleine allouée à une liste

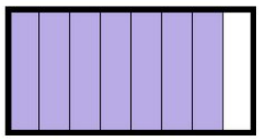


# Deux philosophies différentes

## Implémentation par des listes PYTHON

- Créer une liste PYTHON consiste à choisir un bloc de cases mémoires contiguës. Un item de la liste est stockés juste à côté de son prédecesseur.

FIGURE – Exemple de zone mémoire non pleine allouée à une liste



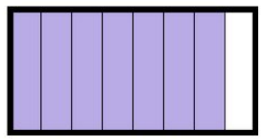
- C'est un avantage pour beaucoup d'opérations comme récupérer un élément par son index.

# Deux philosophies différentes

## Implémentation par des listes PYTHON

- Créer une liste PYTHON consiste à choisir un bloc de cases mémoires contiguës. Un item de la liste est stockés juste à côté de son prédecesseur.

FIGURE – Exemple de zone mémoire non pleine allouée à une liste



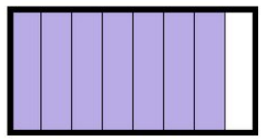
- C'est un avantage pour beaucoup d'opérations comme récupérer un élément par son index.
- Avec `myList[3]`, Python se rend à l'adresse de la liste et se déplace vers la droite de 3 cases : c'est quasiment à coût constant.

# Deux philosophies différentes

## Implémentation par des listes Python

- Créer une liste Python consiste à choisir un bloc de cases mémoires contiguës. Un item de la liste est stockés juste à côté de son prédecesseur.

Figure – Exemple de zone mémoire non pleine allouée à une liste



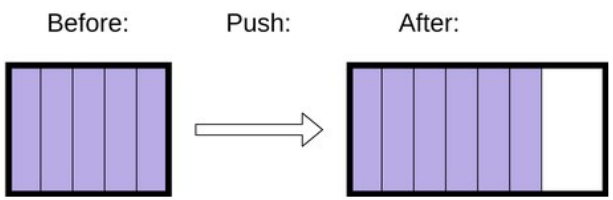
- C'est un avantage pour beaucoup d'opérations comme récupérer un élément par son index.
- Avec `myList[3]`, Python se rend à l'adresse de la liste et se déplace vers la droite de 3 cases : c'est quasiment à coût constant.
- Ajouter un élément quand le bloc n'est pas plein est également très rapide.

# Deux philosophies différentes

## Implémentation par des listes PYTHON

- Cependant, lorsque le bloc est plein et qu'on ajoute un élément, il faut réserver une zone mémoire assez grande pour stocker la liste et tout recopier dans la nouvelle zone. Cette opération est coûteuse.

FIGURE – Ajout d'un élément lorsque le bloc alloué est plein

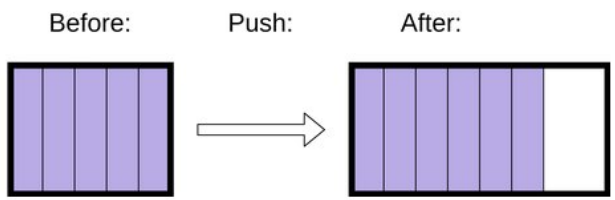


# Deux philosophies différentes

## Implémentation par des listes PYTHON

- Cependant, lorsque le bloc est plein et qu'on ajoute un élément, il faut réserver une zone mémoire assez grande pour stocker la liste et tout recopier dans la nouvelle zone. Cette opération est coûteuse.

FIGURE – Ajout d'un élément lorsque le bloc alloué est plein



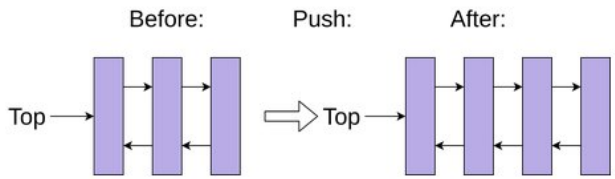

- C'est la raison pour laquelle, l'ajout d'un élément en fin de liste n'est de complexité constante qu'en moyenne. Dans le pire des cas (lorsque la zone allouée est pleine), il est de complexité linéaire.

# Deux philosophies différentes

## Implémentation par la fonction `deque`

- On utilise des *listes doublement chaînées*, une structure de donnée dans laquelle chaque élément, outre sa valeur, contient l'adresse de son prédecesseur et de son successeur dans la liste chaînée.

FIGURE – Exemple d'ajout au sommet d'une liste doublement chaînée.

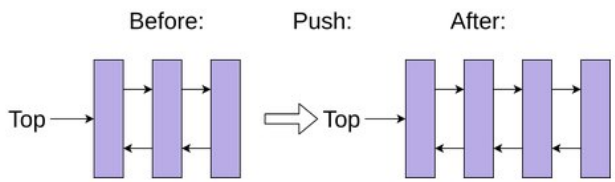


# Deux philosophies différentes

## Implémentation par la fonction `deque`

- On utilise des *listes doublement chaînées*, une structure de donnée dans laquelle chaque élément, outre sa valeur, contient l'adresse de son prédecesseur et de son successeur dans la liste chaînée.

FIGURE – Exemple d'ajout au sommet d'une liste doublement chaînée.



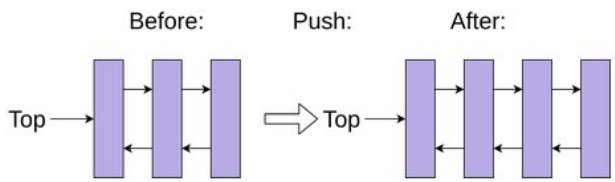

- Ceci permet d'ajouter *à coût constant* un élément au sommet (Top) de la liste. Il n'y a jamais de problème de recopiage.

# Deux philosophies différentes

## Implémentation par la fonction `deque`

- On utilise des *listes doublement chaînées*, une structure de donnée dans laquelle chaque élément, outre sa valeur, contient l'adresse de son prédecesseur et de son successeur dans la liste chaînée.

FIGURE – Exemple d'ajout au sommet d'une liste doublement chaînée.




- Ceci permet d'ajouter *à coût constant* un élément au sommet (Top) de la liste. Il n'y a jamais de problème de recopiage.
- Cependant, l'accès à un item par son index est assez long car `Python` doit partir du sommet et parcourir tous les éléments jusqu'à trouver le bon index. Complexité linéaire.

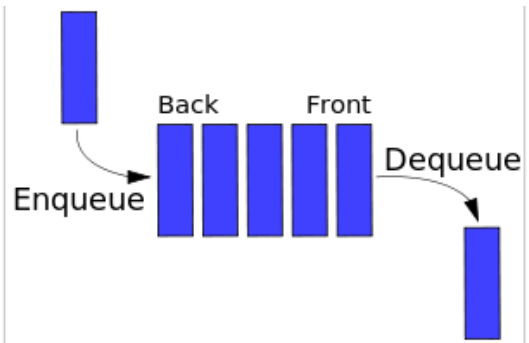
1 Piles
- Généralités
- Implémentation

2 Files

# Définition

- En informatique, une *file* (*queue* en anglais) est une structure de données basée sur le principe du Premier entré, premier sorti, en anglais FIFO (First In, First Out), ce qui veut dire que les premiers éléments ajoutés à la file seront les premiers à être récupérés.

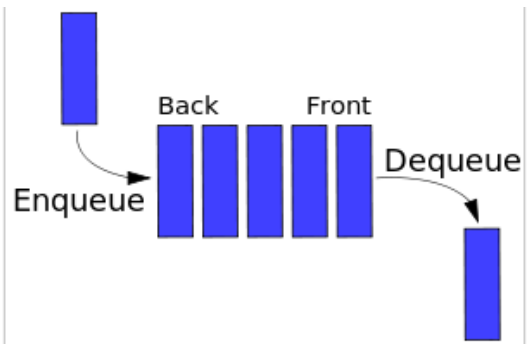
FIGURE – Une pile (FIFO)



# Définition

- En informatique, une *file* (*queue* en anglais) est une structure de données basée sur le principe du Premier entré, premier sorti, en anglais FIFO (First In, First Out), ce qui veut dire que les premiers éléments ajoutés à la file seront les premiers à être récupérés.
- Le fonctionnement ressemble à une file d'attente à la poste : les premières personnes à arriver sont les premières personnes à sortir de la file.

FIGURE – Une pile (FIFO)



# Applications

- Usage : Mémoriser temporairement des transactions qui doivent attendre pour être traitées *dans l'ordre d'arrivée*.

# Applications

- Usage : Mémoriser temporairement des transactions qui doivent attendre pour être traitées *dans l'ordre d'arrivée*.
- Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue).

# Applications

- Usage : Mémoriser temporairement des transactions qui doivent attendre pour être traitées *dans l'ordre d'arrivée*.
- Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue).
- Certains moteurs multitâches, dans un système d'exploitation, qui doivent accorder du temps-machine à chaque tâche, sans en privilégier aucune.

# Applications

- Usage : Mémoriser temporairement des transactions qui doivent attendre pour être traitées *dans l'ordre d'arrivée*.
- Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue).
- Certains moteurs multitâches, dans un système d'exploitation, qui doivent accorder du temps-machine à chaque tâche, sans en privilégier aucune.
- Un algorithme de *parcours en largeur d'abord* dans un graphe utilise une file pour mémoriser les nœuds visités.

# Applications

- Usage : Mémoriser temporairement des transactions qui doivent attendre pour être traitées *dans l'ordre d'arrivée*.
- Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue).
- Certains moteurs multitâches, dans un système d'exploitation, qui doivent accorder du temps-machine à chaque tâche, sans en privilégier aucune.
- Un algorithme de *parcours en largeur d'abord* dans un graphe utilise une file pour mémoriser les nœuds visités.
- Pour créer toutes sortes de mémoires tampons (en anglais *buffers*).

# Primitives

Voici les primitives communément utilisées pour manipuler des files. Il n'existe pas de normalisation pour les primitives de manipulation de file. Leurs noms sont donc indiqués de manière informelle.

- « Enfiler » : ajoute un élément dans la file. Terme anglais correspondant : « Enqueue ».

# Primitives

Voici les primitives communément utilisées pour manipuler des files. Il n'existe pas de normalisation pour les primitives de manipulation de file. Leurs noms sont donc indiqués de manière informelle.

- « Enfiler » : ajoute un élément dans la file. Terme anglais correspondant : « Enqueue ».
- « Défiler » : renvoie le prochain élément de la file, et le retire de la file. Terme anglais correspondant : « Dequeue ».

# Primitives

Voici les primitives communément utilisées pour manipuler des files. Il n'existe pas de normalisation pour les primitives de manipulation de file. Leurs noms sont donc indiqués de manière informelle.

- « Enfiler » : ajoute un élément dans la file. Terme anglais correspondant : « Enqueue ».
- « Défiler » : renvoie le prochain élément de la file, et le retire de la file. Terme anglais correspondant : « Dequeue ».
- « La file est-elle vide ? » : renvoie « vrai » si la file est vide, « faux » sinon.

# Primitives

Voici les primitives communément utilisées pour manipuler des files. Il n'existe pas de normalisation pour les primitives de manipulation de file. Leurs noms sont donc indiqués de manière informelle.

- « Enfiler » : ajoute un élément dans la file. Terme anglais correspondant : « Enqueue ».
- « Défiler » : renvoie le prochain élément de la file, et le retire de la file. Terme anglais correspondant : « Dequeue ».
- « La file est-elle vide ? » : renvoie « vrai » si la file est vide, « faux » sinon.
- « Nombre d'éléments dans la file » : renvoie le nombre d'éléments dans la file.

# En Python

- On peut utiliser une liste comme une file FIFO.

# En Python

- On peut utiliser une liste comme une file FIFO.
- Pas efficace : alors que les ajouts et suppressions en fin de liste sont rapides, les opérations d'insertions ou de retraits en début de liste sont lentes (car tous les autres éléments doivent être décalés d'une position).

# En Python

- On peut utiliser une liste comme une file FIFO.
- Pas efficace : alors que les ajouts et suppressions en fin de liste sont rapides, les opérations d'insertions ou de retraits en début de liste sont lentes (car tous les autres éléments doivent être décalés d'une position).
- Pour implémenter une file, utiliser la classe `collections.deque` qui a été conçue pour fournir des opérations d'ajouts et de retraits rapides aux deux extrémités.

# En Python

```
1 from collections import deque
2 from collections import deque
3 queue = deque(["Eric", "John", "Michael"])
4 queue.append("Terry") # Terry arrive
5 queue.append("Graham") # Graham arrive
6 #le 1er arrivé s'en va:
7 print(queue.popleft())
8 #le snd arrivé s'en va:
9 print(queue.popleft())
10 print(queue)
```

Affichage :

```
Eric
John
deque(['Michael', 'Terry', 'Graham'])
```