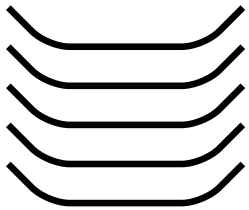
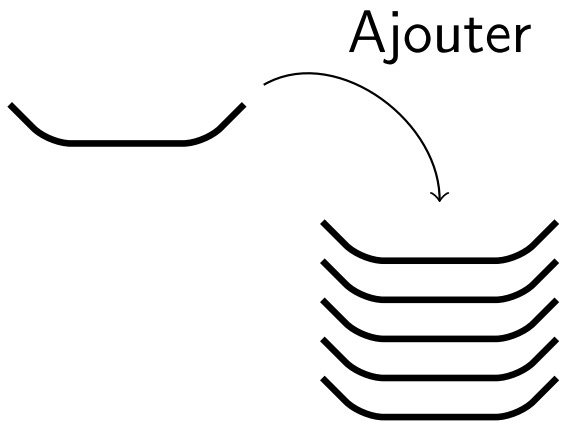
Piles et files

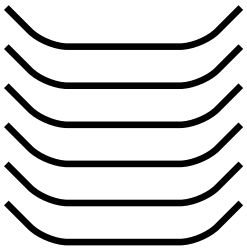
Pile d'assiettes



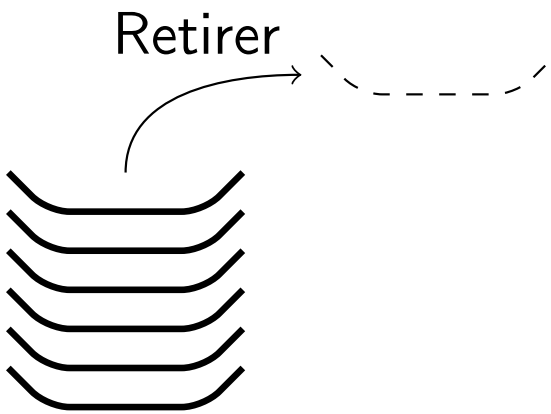
Pile d'assiettes



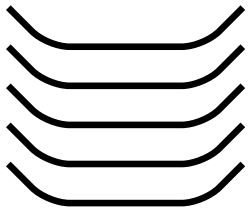
Pile d'assiettes



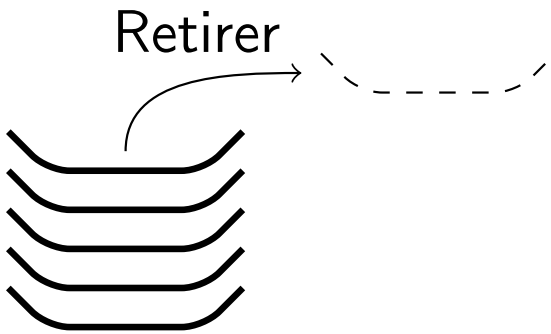
Pile d'assiettes



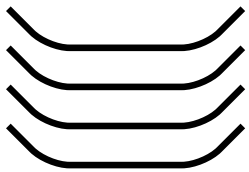
Pile d'assiettes



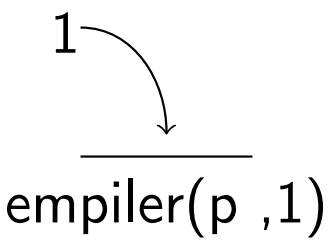
Pile d'assiettes

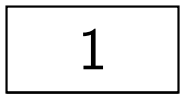


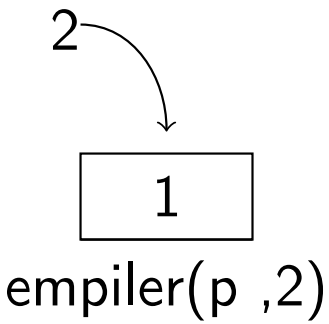
Pile d'assiettes

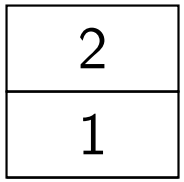


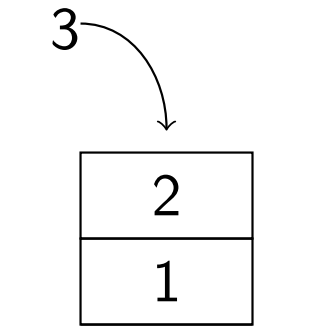
$$p = \overline{\text{creer\_pile}()}$$


1
empiler(p ,1)

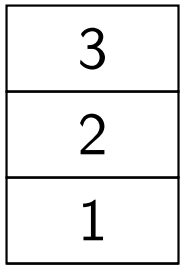

1

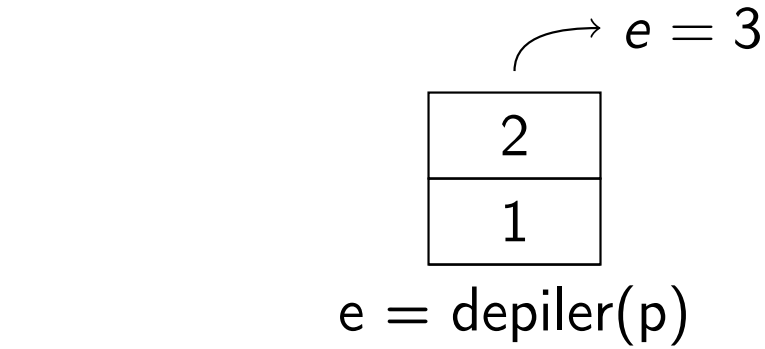

2
1
empiler(p ,2)


2
1



empiler(p ,3)


3
2
1


e = 3
2
1
e = depiler(p)

Interface d'une pile

fonction	description
`creer_pile()`	crée et renvoie une pile vide
`est_vide(p)`	renvoie `True` si la pile est vide et `False` sinon
`empiler(p, e)`	ajoute `e` au sommet de `p`
`depiler(p)`	retire et renvoie l'élément au sommet de `p` si `p` n'est pas vide, et lève une exception sinon

Réalisation avec une liste chaînée

Class Pile réaliser avec pour unique attribut une liste chaînée `contenu`.

```
1 class Pile:
2     '''structure de pile'''
3     def __init__(self):
4         self.contenu = None
```

L'attribut `contenu` est alors initialiser à la liste vide.

Réalisation avec une liste chaînée

La pile est vide si son contenu est vide.

```
1     def est_vide(self):
2         return self.contenu is None
```

Réalisation avec une liste chaînée

La pile est vide si son contenu est vide.

```
1     def est_vide(self):
2         return self.contenu is None
```

Empiler un élement est ajouter un élément en tête de la liste chaînée `contenu`.

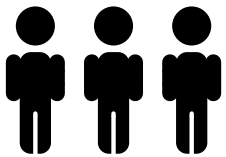
```
1     def empiler(self, e):
2         self.contenu = Cellule(e,self.contenu)
```

Réalisation avec une liste chaînée

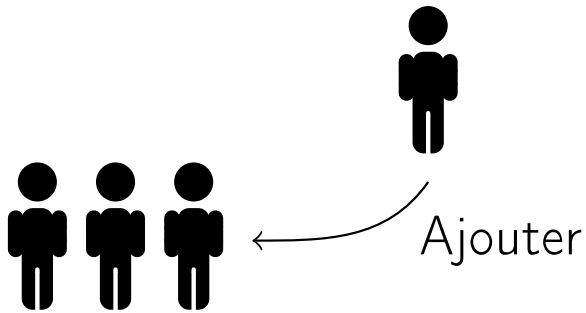
Dépiler revient à supprimer l'élément en tête de la liste chaînée contenu si celle-ci n'est pas vide.

```
1  def depiler(self):
2      if self.est_vide():
3          raise IndexError('depiler sur une pile vide')
4      else:
5          v = self.contenu.valeur
6          self.contenu = self.contenu.suivante
7          return v
```

File d'attente

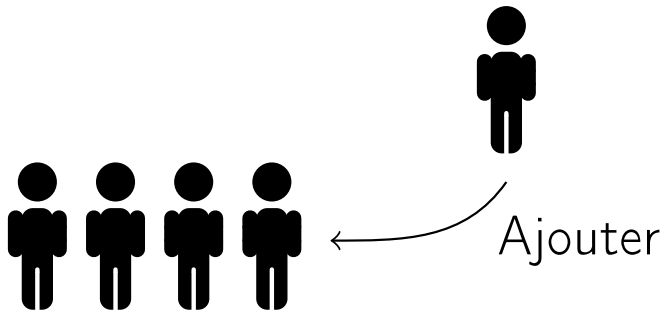


File d'attente

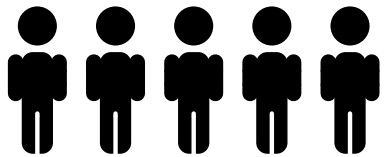


File d'attente

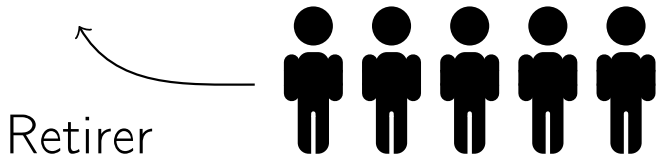
File d'attente



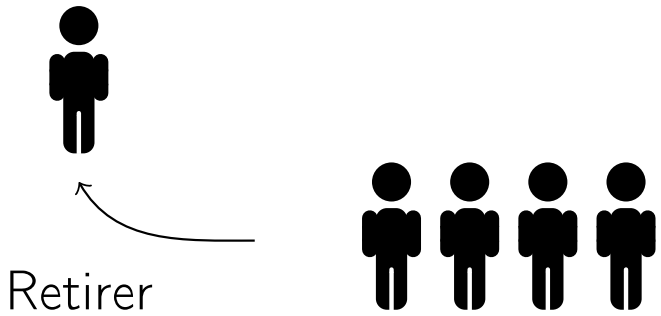
File d'attente



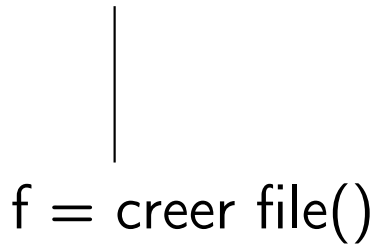
File d'attente

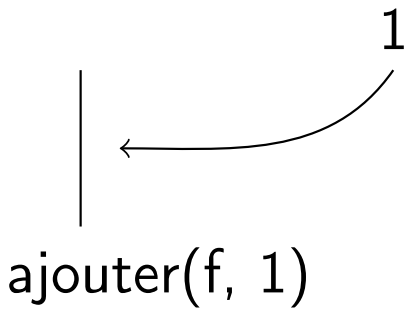


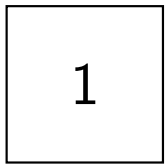
File d'attente

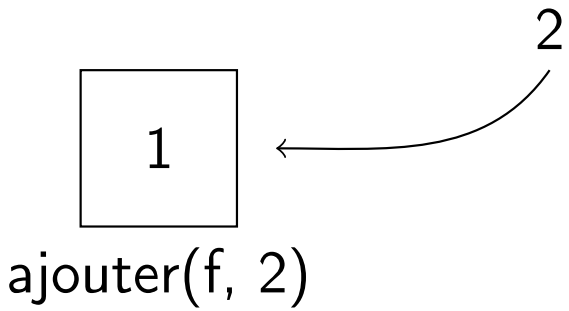


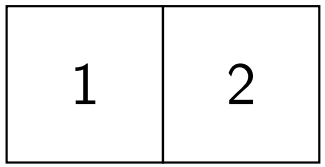
File d'attente

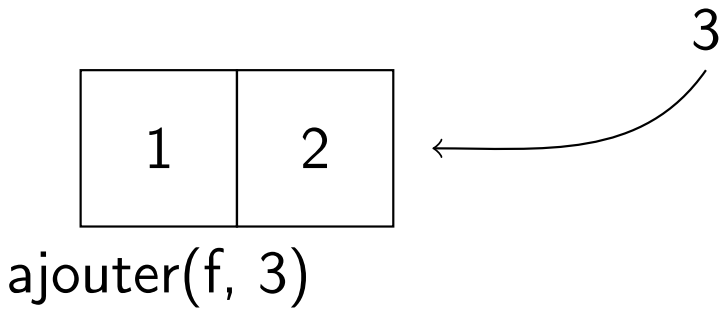

f = creer file()

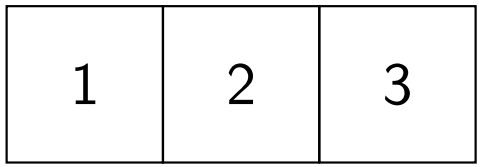

1
ajouter(f, 1)

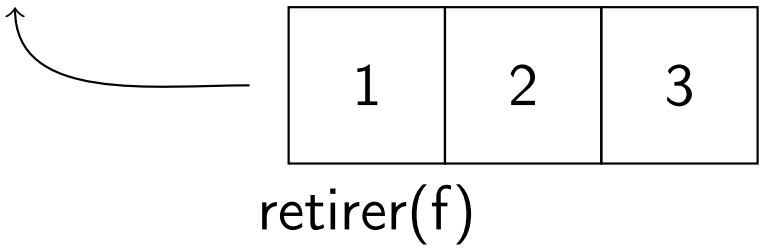

1

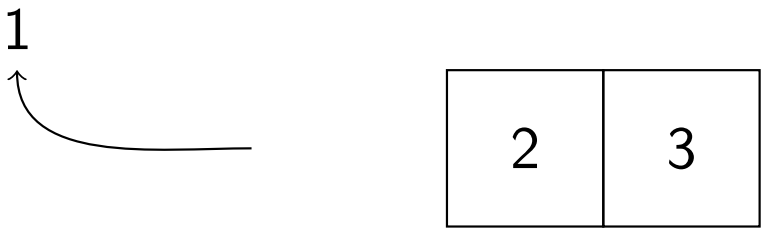

2
1
ajouter(f, 2)

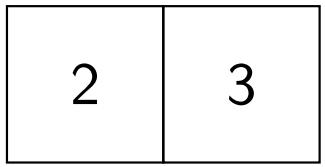

1
2


3
1
2
ajouter(f, 3)


1
2
3


1
2
3
retirer(f)


1
2
3


2
3

Interface d'une file

fonction	description
`creer_file()`	crée et renvoie une file vide
`est_vide(f)`	renvoie `True` si la file est vide et `False` sinon
`ajouter(f, e)`	ajoute `e` à l'arrière de `f`
`retirer(p)`	retire et renvoie l'élément à l'avant de `f` si `f` n'est pas vide, et lève une exception sinon

Réalisation avec une liste chaînée

Classe `File` avec deux attributs, `tete` et queue, et désignant respectivement la première cellule et la dernière cellule de la liste chaînée.

```
1 class File:
2     '''structure de file'''
3     def __init__(self):
4         self.tete = None
5         self.queue = None
```

Réalisation avec une liste chaînée

Pour tester si la file est vide, il suffit de tester si un de ces deux attributs est `None`.

```
1     def est_vide(self):
2         return self.tete is None
```

Réalisation avec une liste chaînée

Pour ajouter un élément, on créer une nouvelle cellule.

```
1       c = Cellule(e, None)
```

Réalisation avec une liste chaînée

Pour ajouter un élément, on créer une nouvelle cellule.

```
1       c = Cellule(e, None)
```

On l'ajoute en fin de file.

```
1       self.queue.suivante = c
2       self.queue = c
```

Réalisation avec une liste chaînée

Pour ajouter un élément, on créer une nouvelle cellule.

```
1       c = Cellule(e, None)
```

On l'ajoute en fin de file.

```
1       self.queue.suivante = c
2       self.queue = c
```

On doit cependant traîter le cas particulier où la file est vide.

```
1   def ajouter(self, e):
2       c = Cellule(e, None)
3       if self.est_vide():
4           self.tete = c
5       else:
6           self.queue.suivante = c
7       self.queue = c
```

Réalisation avec une liste chaînée

Pour retirer un élément, on procède de la même manière que pour une pile.

```
1  def retirer(self):
2      if self.est_vide():
3          raise IndexError('retirer sur une file vide')
4      else:
5          v = self.tete.valeur
6          self.tete = self.tete.suivante
7          if self.tete is None:
8              self.queue = None
9          return v
```

Il faut redéfinir l'attribut queue à `None` lorsque l'opération vide la file.