

Structures de données

I. Introduction

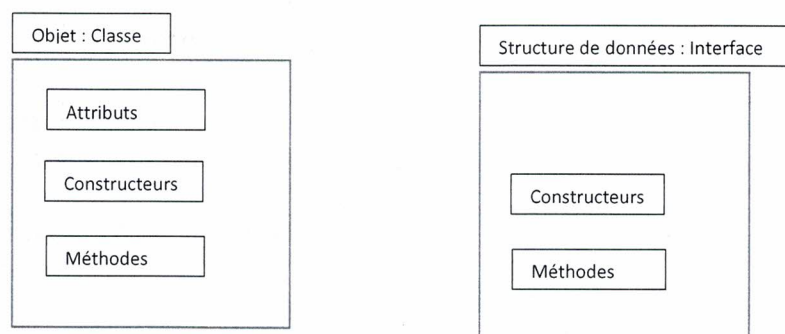
Une structure de données est une manière de stocker et d'organiser de l'information. Le choix d'une structure de données se fait donc en fonction du type de données (associées à de l'information), ce choix doit faciliter l'accès aux données et leur manipulation avec le moindre coût algorithmique.

Une structure de donnée est **un type abstrait**, elle manipule différents types de données (nombres, chaînes de caractères, objets, tableaux, etc.). Ici on détaillera les structures de données suivantes : **les dictionnaires, les listes, les piles et les files**.

II. Interface

Un objet a une classe pour l'instancier, **des méthodes** de la classe permettent d'interagir avec.

Une structure de données a une interface pour la créer, **des méthodes** de l'interface permettent d'interagir avec.



L'interface spécifie les procédures existantes pour manipuler les données (les méthodes) : Ajouter, retirer, compter, accéder, insérer, etc.

Comme pour un objet le constructeur d'une interface est une méthode particulière que l'on préfère différencier des autres méthodes.

Une structure de donnée est correctement implémentée si elle possède certaines méthodes indispensables (ces méthodes sont parfois appelées primitives).

Pour une même structure de données plusieurs interfaces sont possibles !

→ Comme en programmation orienté objet, changer l'interface d'une structure de données n'oblige pas à réécrire une application l'implémentant (à condition de ne pas changer les noms et arguments d'entrées des méthodes).

Structures de données

III. Le dictionnaire

Interface

Constructeur du dictionnaire: `lettres = dict()={ }`

Méthodes du dictionnaire:

- **opérateurs** ajouter une donnée `lettres['A']=5    lettres['E']=10`
 `lettres = { 'A':5, 'E':10 }`
 retirer une donnée `lettre=lettres.pop('A')`
 `lettres = { 'E':10 }`
- **accesseurs** accéder à la valeur `print(lettres['E'])`
 10
- **itérateurs** énumérer les éléments `lettres = { 'A':5, 'E':10 }`
 `for key,valeur in lettres.items() :`
 `print((key, valeur))`
 ('A':5)
 ('E':10)

Exemple d'utilisation: dictionnaire qui comptabilise le nombre d'apparition des lettres de l'alphabet dans un texte.

Structures de données

IV. La Pile

↓ ↑ Last In First Out

'E'
'D'
'C'
'B'
'A'

Les éléments d'une pile s'ajoutent les uns après les autres. Une pile se caractérise par la propriété « **LIFO** » qui veut dire **Last In First Out**.

➔ Pour accéder à un élément d'une pile, il faut dépiler la pile (comme la pile d'assiette).

Si on ne garde pas en mémoire les éléments dépilés, ils sont perdus !

Il existe plusieurs interfaces pour créer une pile, en voici une possible qui se base sur la programmation orientée objet.

Class Pile :

#Constructeur

```
def __init__(self):
    self.valeurs=[]
```

#Opérateur :ajouter une donnée

```
def empiler(self,valeur):
    self.valeurs.append(valeur)
```

#Opérateur :retirer une donnée

```
def depiler(self):
    if self.valeurs: len(self.valeurs) > 0 :
        return self.valeurs.pop(valeur)
```

#Accesseur :tester si la pile est vide

```
def estVide(self):
    return self.valeurs==[]
```

Exemples d'utilisation :

- Les algorithmes récursifs,
- le principe « undo et redo » utilisent les piles.

Structures de données

V. Le tableau dynamique en python

Interface

Constructeur de la liste: *entiers = liste()=[]*

Méthodes de la liste:

- | | | | |
|---|-------------------|-----------------------|---|
| - | opérateurs | ajouter une donnée | <i>entiers.append(1)</i>
<i>entiers.append(3)</i>
<i>entiers = [1, 3]</i> |
| | | retirer une donnée | <i>entier=entiers.pop(0)</i>
<i>entier = 1</i>
<i>entiers = [3]</i> |
| - | accesseurs | nombre d'éléments | <i>entiers = [1, 3]</i>
<i>print(len(entiers))</i>
<i>2</i> |
| - | itérateurs | énumérer les éléments | <i>entiers = [1, 3]</i>
<i>for entier in entiers :</i>
<i> print(entiers)</i>
<i>1</i>
<i>3</i> |

Exemple d'utilisation: tableau dynamique d'objets.

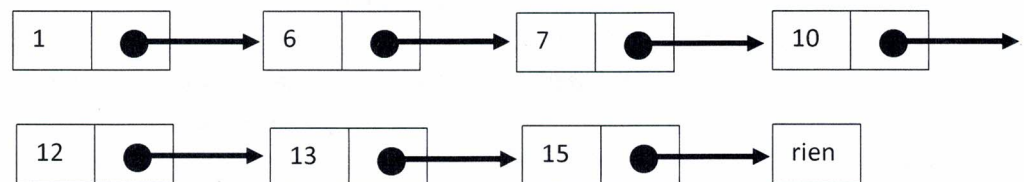
Structures de données

VI. La liste (ou liste chaînée)

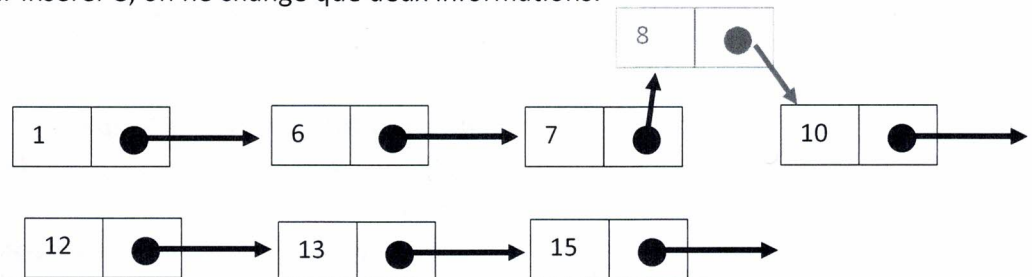
La liste en python qui est en réalité un tableau dynamique n'est pas la vraie liste des informaticiens.

Mise en situation :

Pour insérer l'entier 8 dans le tableau d'entiers triés [1,6,7,10,12,13,15], il faut décaler les 4 derniers éléments du tableau. Soit 4 opérations à réaliser. Sur des tableaux de très grande taille, il est inconcevable de réaliser cette opération. On préférera utiliser l'organisation suivante, à partir de cellules :



Pour insérer 8, on ne change que deux informations:



- La cellule contenant 7 pointe vers la nouvelle cellule contenant 8
- La nouvelle cellule pointe vers la cellule contenant 10.

➔ Cette organisation est appelée **liste chaînée**.

Une liste chaînée est constituée **de cellules**, une cellule contient :

- une donnée (ici un nombre entier)
- un pointeur qui pointe vers la cellule suivante (soit l'adresse où se trouve la donnée suivante)

Structures de données

```
class Cellule:
    def __init__(self, tete, queue;
                self.car = tete ):# tete → donnée
                self.cdr = queue # queue → la cellule pointée
```

On peut utiliser par exemple l'interface suivante ([source : eduscol.education.fr](http://source.eduscol.education.fr)):

```
class Liste:
    def __init__(self, c): #Constructeur
        self.cellule = c

    def estVide(self): # Accesseur teste si la liste est vide
        return self.cellule is None

    def car(self): # Opérateur renvoyer la tête
        assert not(self.cellule is None), 'Liste vide'
        return self.cellule.car

    def cdr(self): # Opérateur renvoyer la queue
        assert not(self.cellule is None), 'Liste vide'
        return self.cellule.cdr
```

Utilisation de l'interface pour construire une liste

```
def cons(tete, queue): ajouter une donnée à la liste
    return Liste(Cellule(tete, queue))
```

En s'appuyant sur cette interface, on peut définir les fonctions suivantes :

```
def longueurListe(L):
    #la fonction renvoie la longueur de la liste
    n = 0
    while not(L.estVide()):
        n += 1
        L = L.cdr()
    return n

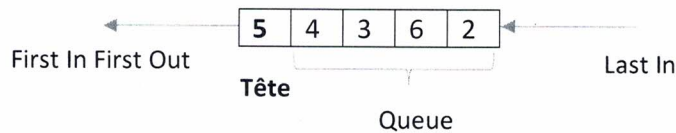
def listeElements(L):
    # La fonction renvoie la liste (au sens de Python) des éléments d'une
    liste chaînée.
    t = []
    while not(L.estVide()):
        t.append(L.car())
        L = L.cdr()

    return t
```

Structures de données

VII. La file

Une file se caractérise par la propriété « **FIFO** » qui veut dire **First In First Out**.



La **tête** est le premier élément ajouté à la file.

La **queue** sont les éléments suivants.

→ Pour accéder à un élément d'une file, il faut défiler chaque élément.

Si on ne garde pas en mémoire les éléments défilés, ils sont perdus !

Il existe plusieurs interfaces pour créer une file, en voici une possible qui se base sur la programmation orientée objet.

Class File :

#Constructeur

```
def __init__(self):  
    self.valeurs=[]
```

#Opérateur :ajouter une donnée en tête

```
def enfiler(self,valeur) :  
    self.valeurs.append(valeur)
```

#Opérateur :retirer une donnée en queue

```
def defiler(self) :  
    if self.valeurs :  
        return self.valeurs.pop(0)
```

#Accesseur :tester si la file est vide

```
def estVide(self) :  
    return self.valeurs==[]
```

Exemples d'utilisation :

- Les buffers (mémoires tampon), un serveur d'impression utilisent les files.
- Les réservations de billets