Structure de données (1)

Ensemble d'objets organisés qui sont liés

Informations structurées

Interface : définition théorique d'une structure de données.

Implémentation : mise en œuvre concrète, pratique, de l'interface dans un langage de programmation en particulier.

Interface		Exemple d'implémentation
type abstrait **liste**	→	**liste chaînée** (python)
Classe en POO	→	Définition d'une classe (python)
Schéma d'une base de données relationnels	→	**Table SQL**
Le plan de votre maison de rêve	→	Le **plan de l'architecte** avec le **choix des matériaux**

Radiateur
Livre rouge
Tableau
bleu blanc

Informations bruts

Objet	Couleur
Radiateur	bleu
Livre	rouge
Tableau	blanc

Données structurées

Programmation orientée objet (POO) :

- Basée sur les concepts de **classe**, **attribut**, **méthode** et **objet**
- **Paradigme de programmation**

Autres paradigmes : programmation impérative, programmation fonctionnelle

	Définition	Métaphore culinaire	Métaphore architecturale
Classe	Définition des caractéristiques d'un ensemble d'objets et des fonctions associées	Le moule et la recette	Plan d'un studio
Objet	Représente un élément du monde physique ou un concept. Il est caractérisé par une structure interne (attributs) et un comportement (méthodes).	Un gâteau	Réalisation d'un studio
Attribut	Paramètres d'un objet de la classe	Poid, temps cuisson...	Taille, région...
Méthode	Fonction associée à une classe	Découper, ajouter une bougie...	Louer, estimer la valeur...

POO avec python

Docstring (aide utilisateur)

Attributs

Constructeur

Méthodes

Création d'un objet

Utilisation des méthodes

```python
class Pays:
    """
    La classe Pays définit des objets caractérisés par les attributs nom (string),
    drapeau (chemin d'un fichier image), continent (string) et roi (booléen) et
    par les méthodes __init__,  __repr__ et est_en_afrique.
    """
    def __init__(self, nom, drapeau, continent, roi):
        self.nom = nom
        self.drapeau = drapeau
        self.continent = continent
        self.roi = roi

    def __repr__(self):
        if self.roi:
            return self.nom + " est un pays (" + self.continent
            + ") qui est dirigé par un roi."
        else:
            return self.nom + " est un pays (" + self.continent
            + ") qui n'est pas dirigé par un roi."

    def est_en_afrique(self):
        return self.continent == "Afrique"

tokelau = Pays("Tokelau", "Océanie", "drapeau_tokelau.png", True)
print(tokelau)
print(tokelau.est_en_afrique())
```

objet.méthode()

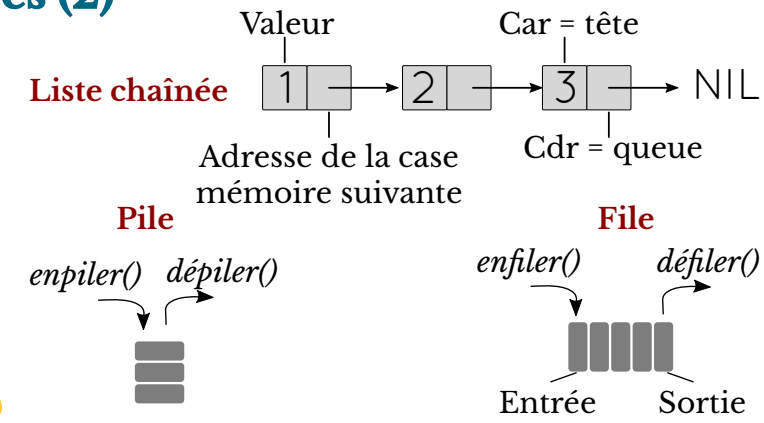
Adrien Taudière

Structure de données (2)

TYPE ABSTRAIT LISTE

Une implémentation du type abstrait liste doit permettre de :
- **Construire** (i) une liste vide, (ii) une liste à partir d'autres listes ;
- **Accéder** aux valeurs de la liste (*via* sa tête ou sa queue) ;
- **Tester** si une liste est vide.

Différentes implémentations du type abstrait liste :
- **Liste chaînée** (implémentation la moins gourmande en mémoire) ;
- **Tableau dynamique** (cas de l'implémentation sous python) ;
- **Pile** et **File**



TYPES CONSTRUITS (rappel première)

Tableaux Objet **mutable** (c.-à-d. modifiable) qui liste des objets de **tout type** y compris d'autres objets complexes (par ex. des tupples)

notes_Joe_Dalton = ["Dalton", "Joe", "01/01/1750", 12, 10, 15]

Daltons = [["Dalton", "Joe", "31 ans"], ["Dalton", "Jack", "31 ans"]]

Tableau 1 Tableau 2

Tupple *(p-upplet)* Objet **immutable** (c.-à-d. non modifiable) qui liste des objets de **tout type** y compris d'autres objets complexes

notes_Joe_Dalton = ("Dalton", "Joe", "01/01/1750", 12, 10, 15)

notes_Joe_Dalton = (["Dalton", "Joe", "01/01/1750"], [12, 10, 15])

Tableau 1 Tableau 2

Dictionnaires Objet **mutable** dont les **éléments sont indexés** par des couples clés (**keys**) - valeurs (**values**).

notes_Joe_Dalton = {"Nom" = "Dalton", Prénom" = "Joe", "Date de naissance" = "01/01/1750", "Note NSI" = 12, "Note" = 10}

KEY VALUE

Implémentation d'une pile en python

```python
class Pile:

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

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

    def depiler(self):
        if self.valeurs:
            return self.valeurs.pop()

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

    def lireSommet(self):
        return self.valeurs[-1]


    def __str__(self):
        ch = ''
        for x in self.valeurs:
            ch = "|\t" + str(x) + "\t|" + "\n" + ch
        ch = "\nEtat de la pile:\n" + ch
        return ch
```

TYPES DE BASE (rappel première)

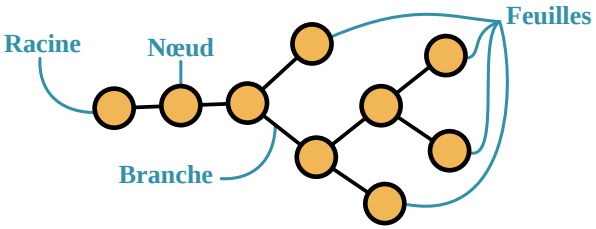
- **int** (integer) pour les variables constituées de nombres entiers,
- **float** pour les variables constituées de nombres décimaux (à virgule),
- **str** pour les variables constituées de chaînes de caractères
- **bool** pour les variables constituées de booléens (True/False)

Adrien Taudière

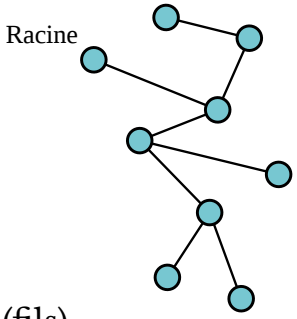
Structure de données (3)

STRUCTURE EN ARBRE

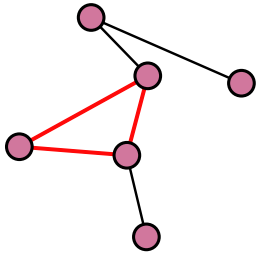
Un **arbre** est constitué de :
- **Nœuds** dont un nœud est la **racine** et les nœuds à l'opposé sont des **feuilles** ;
- **Branches** qui relient les nœuds.



Il ne peut pas y avoir plusieurs chemins entre des nœuds dans un arbre.



Arbre de taille 9, de hauteur 4 et de degré 3



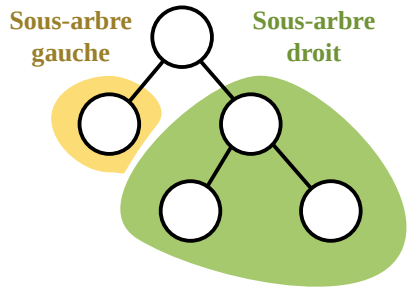
Pas un arbre (présence d'un cycle)

Un nœud est caractérisé par :
- Son statut de **racine** ou de **feuille** ;
- Son **degré** : nombre de descendants (fils).

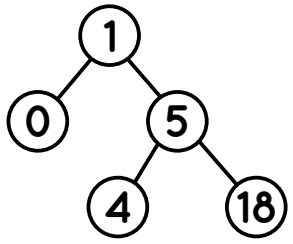
```python
class Noeud:
    def __init__(self, valeur, gauche, droit):
        self.n = valeur
        self.g = gauche
        self.d = droit

class ArbreBinaire:
    def __init__(self, c):
        self.r = c
    def creeVide():
        return ArbreBinaire(None)
    def creeNGD(valeur, gauche = None, droit = None):
        return ArbreBinaire(Noeud(valeur, gauche, droit))
    def estVide(self):
        return self.r is None
    def racine(self):
        assert not(self.r is None), 'Arbre vide'
        return self.r.n
    def filsGauche(self):
        assert not(self.r is None), 'Arbre vide'
        return self.r.g
    def filsDroit(self):
        assert not(self.r is None), 'Arbre vide'
        return self.r.d
```

Un **arbre** est caractérisé par :
- Sa **hauteur** : nombre de nœuds qui constituent la branche contenant le plus de nœuds ;
- Sa **taille** : nombre de nœuds ;
- Son **degré** : plus grand des degrés de ses noeuds.

Un **arbre binaire** est un arbre de **degré 2**. Un arbre binaire peut être défini comme une récursion d'arbres soit vide, soit ayant pour descendant un sous-arbre gauche et un sous arbre droit.



Un **arbre binaire de recherche** est un arbre binaire dont les nœuds possédent une valeur (*int* ou *str*). Ces valeurs sont triées afin de faire des recherches avec la valeur du fils gauche toujours inférieure ou égale à celle du nœud père et la valeur du fils droit toujours supérieure ou égale.



Une implémentation d'arbres binaires doit permettre un certain nombre d'opérations :
- Construire un arbre vide ;
- Tester si un arbre est vide ;
- Construire un arbre à partir d'un entier et de deux sous-arbres gauche et droit ;
- Accèder à la racine d'un arbre ;
- Accèder au sous-arbre gauche et droit.

Adrien Taudière