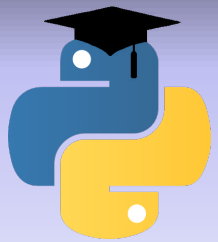
Langage et algorithmique
- Arbre -

A. Malek TOUMI

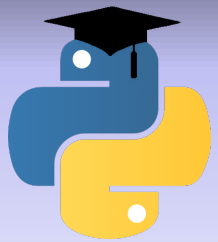
toumiab@ensta-bretagne.fr

2015/2016

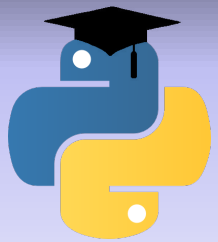
ENSTA Bretagne



Sommaire

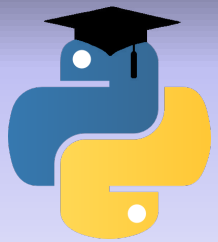


Sommaire



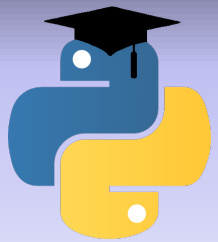
Pourquoi des arbres ?

- Structure dynamique : liste



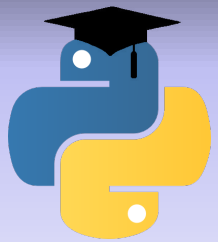
Pourquoi des arbres ?

- Structure dynamique : liste
- Constat : coût élevé des opérations de base



Pourquoi des arbres ?

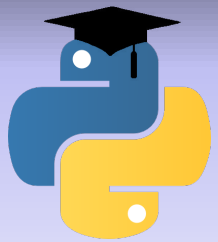
- Structure dynamique : liste
- Constat : coût élevé des opérations de base
- Objectif : trouver une structure de données plus efficace



Pourquoi des arbres ?

- Structure dynamique : liste
- Constat : coût élevé des opérations de base
- Objectif : trouver une structure de données plus efficace

Arbre :

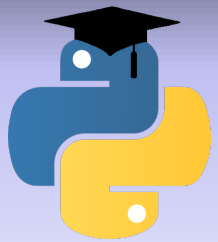


Pourquoi des arbres ?

- Structure dynamique : liste
- Constat : coût élevé des opérations de base
- Objectif : trouver une structure de données plus efficace

Arbre :

- Structure de données dynamique

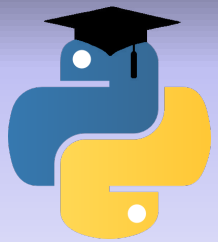


Pourquoi des arbres ?

- Structure dynamique : liste
- Constat : coût élevé des opérations de base
- Objectif : trouver une structure de données plus efficace

Arbre :

- Structure de données dynamique
- Bonne complexité pour les opérations de base

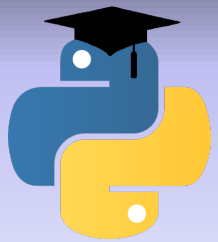


Pourquoi des arbres ?

- Structure dynamique : liste
- Constat : coût élevé des opérations de base
- Objectif : trouver une structure de données plus efficace

Arbre :

- Structure de données dynamique
- Bonne complexité pour les opérations de base
- Recherche, insertion, suppression : $\Theta(\log n)$

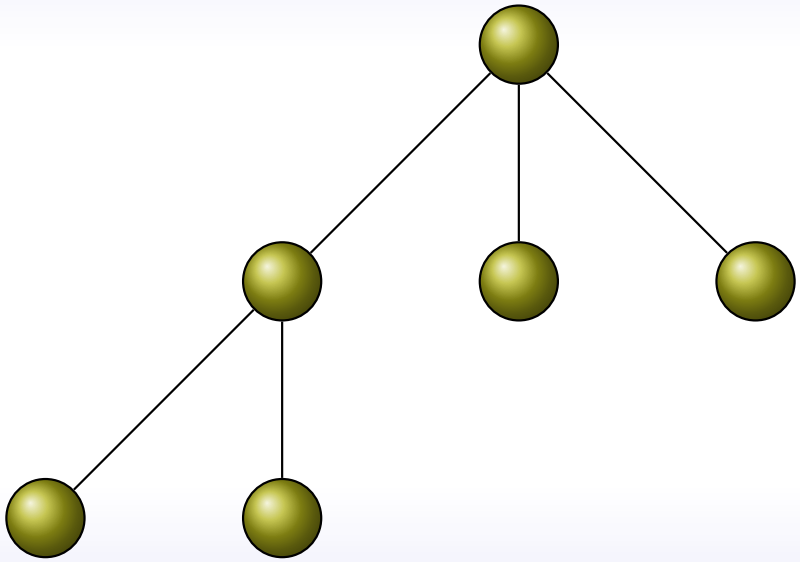


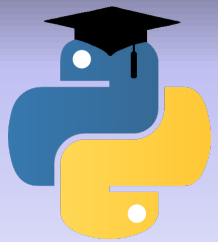
Définitions

Définition (Arbre)

Ensemble non vide de noeuds et d'arêtes tel que :

Définitions



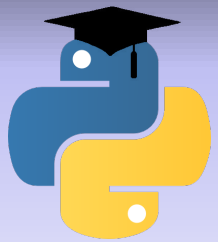


Définitions

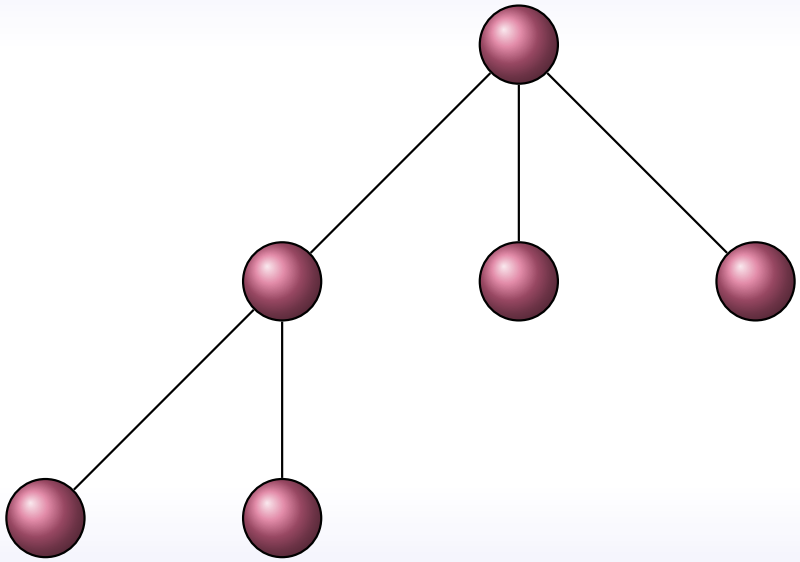
Définition (Arbre)

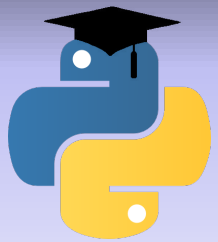
Ensemble non vide de noeuds et d'arêtes tel que :

- noeud : objet simple



Définitions



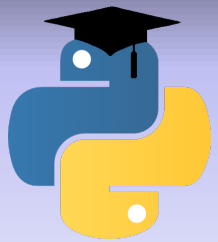


Définitions

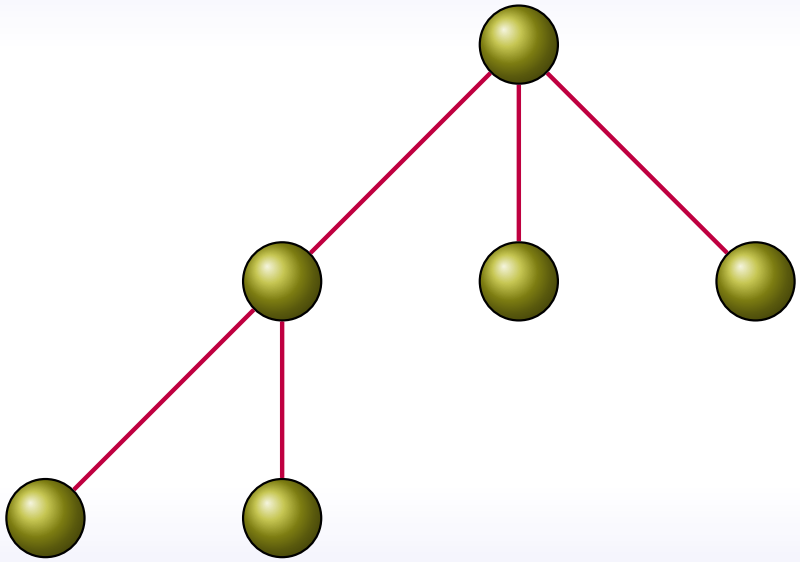
Définition (Arbre)

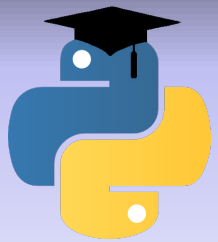
Ensemble non vide de noeuds et d'arêtes tel que :

- noeud : objet simple
- arête : relie deux noeuds (orienté)



Définitions



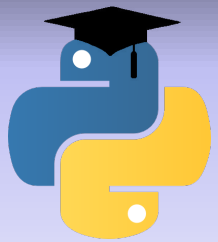


Définitions

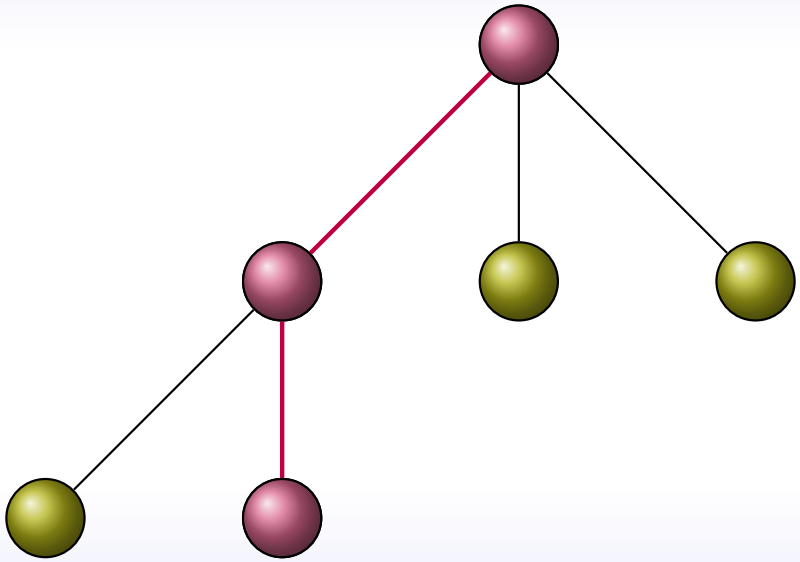
Définition (Arbre)

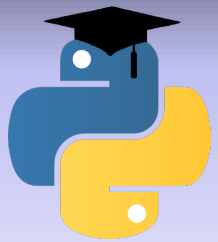
Ensemble non vide de noeuds et d'arêtes tel que :

- noeud : objet simple
- arête : relie deux noeuds (orienté)
- branche : suite de noeuds reliés par des arêtes



Définitions





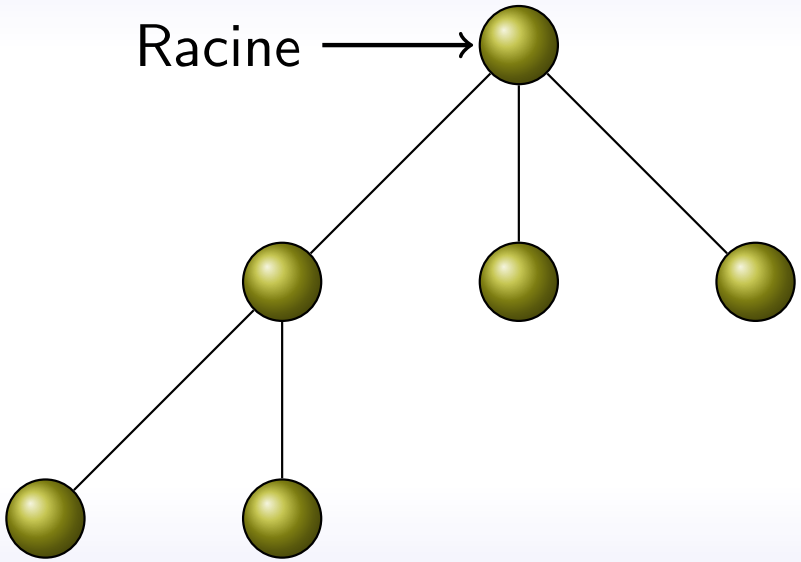
Définitions

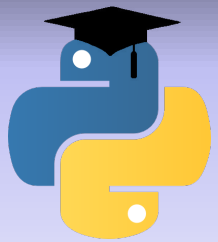
Définition (Arbre)

Ensemble non vide de noeuds et d'arêtes tel que :

- noeud : objet simple
- arête : relie deux noeuds (orienté)
- branche : suite de noeuds reliés par des arêtes
- racine : noeud sans prédécesseur

Définitions



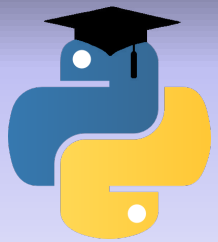


Définitions

Définition (Arbre)

Ensemble non vide de noeuds et d'arêtes tel que :

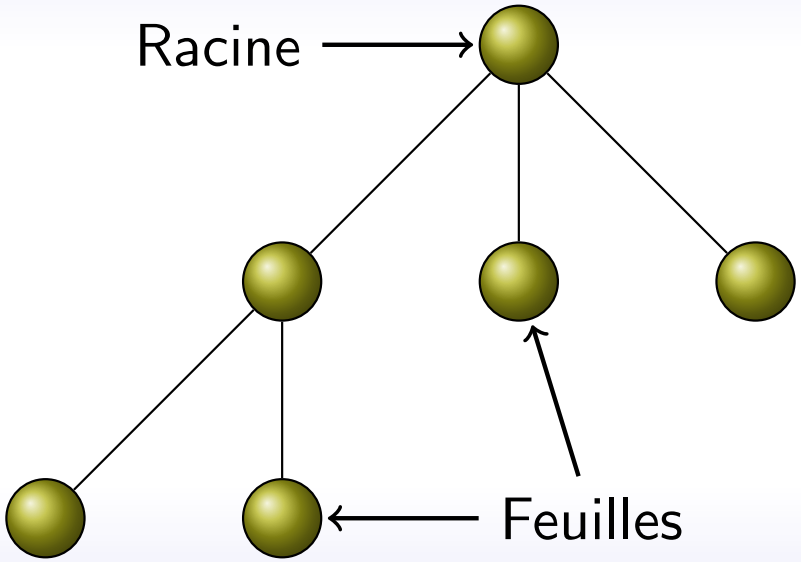
- noeud : objet simple
- arête : relie deux noeuds (orienté)
- branche : suite de noeuds reliés par des arêtes
- racine : noeud sans prédécesseur
- tous les noeuds sont accessibles depuis la racine par une seule branche

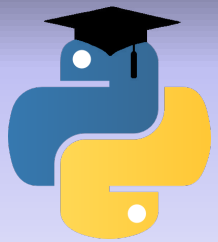


Définitions

- Représentation : racine au sommet de l'arbre
- Relation père-fils entre les noeuds
- Bas de l'arbre : feuilles

Définitions





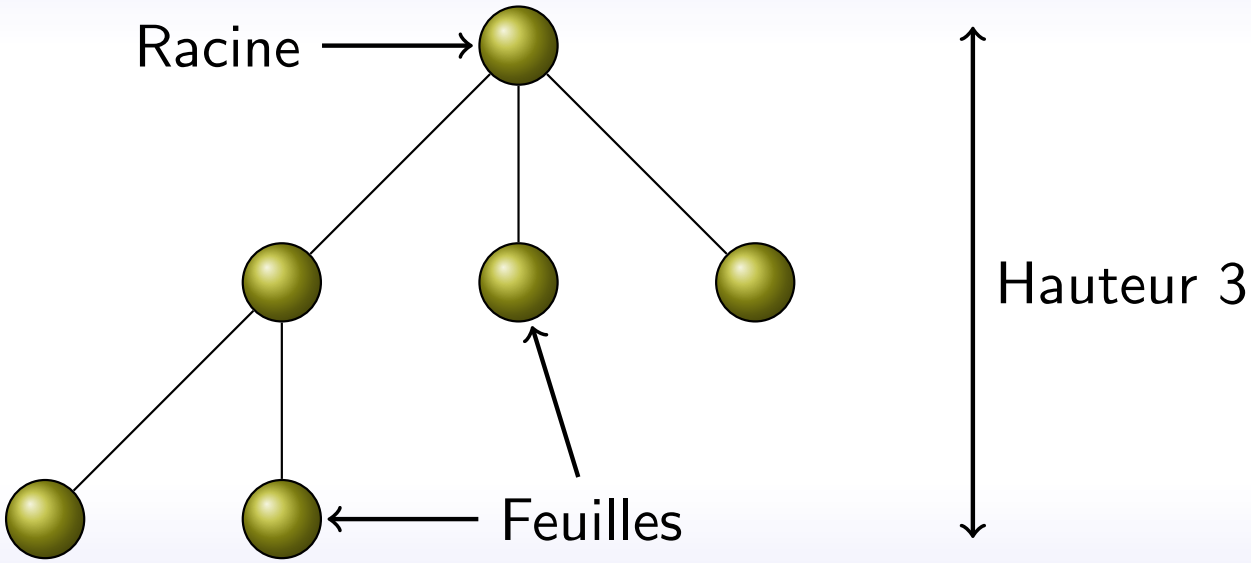
Définitions

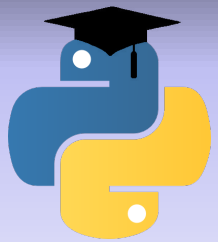
- Représentation : racine au sommet de l'arbre
- Relation père-fils entre les noeuds
- Bas de l'arbre : feuilles

Définition (Hauteur)

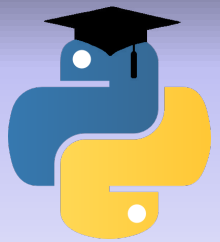
Nombre de noeuds du plus long chemin entre la racine et les feuilles.

Définitions





Sommaire



Définition

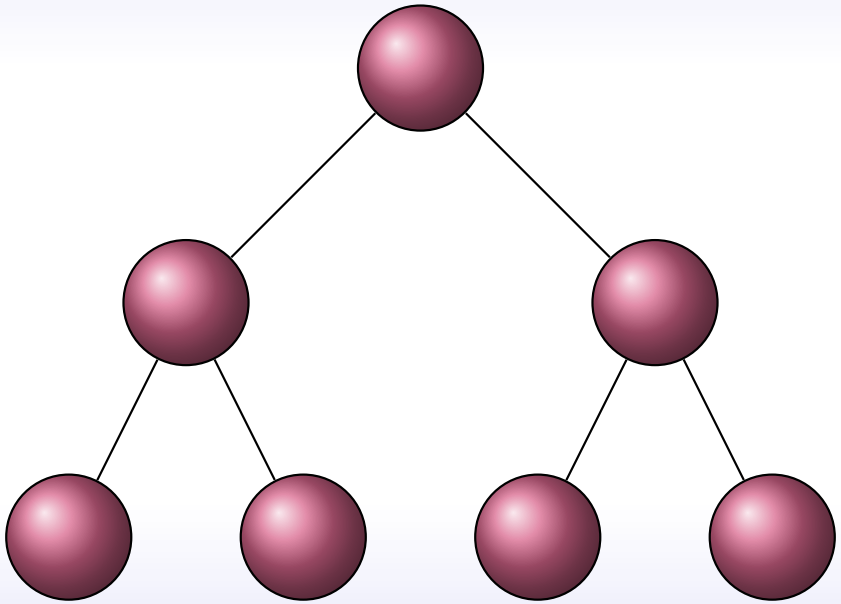
Définition (Arbre binaire)

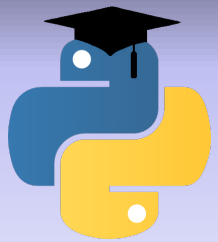
Un arbre binaire est un arbre dont chaque noeud possède au plus deux fils.

Définition (Arbre binaire complet)

Un arbre est complet si chaque noeud qui n'est pas une feuille possède exactement deux fils et où toutes les feuilles sont au même niveau.

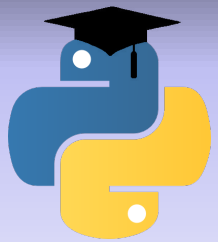
Définition





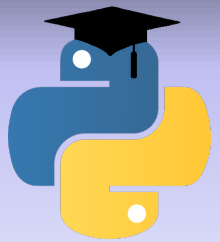
Propriétés

- Chaque noeud possède un fils gauche et un fils droit



Propriétés

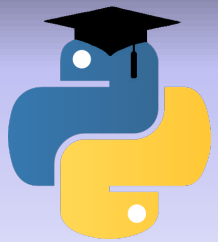
- Chaque noeud possède un fils gauche et un fils droit
- Arbre de hauteur h



Propriétés

- Chaque noeud possède un fils gauche et un fils droit
- Arbre de hauteur h

$\Rightarrow$ $2^h - 1$ noeuds (si complet)

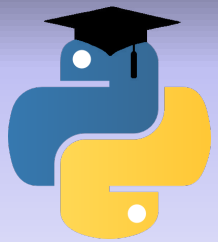


Propriétés

- Chaque noeud possède un fils gauche et un fils droit
- Arbre de hauteur h

$\Rightarrow$ $2^h - 1$ noeuds (si complet)

- n noeuds



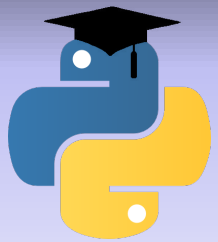
Propriétés

- Chaque noeud possède un fils gauche et un fils droit
- Arbre de hauteur h

$\Rightarrow$ $2^h - 1$ noeuds (si complet)

- n noeuds

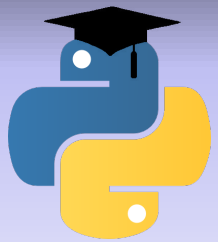
$\Rightarrow$ hauteur $\log n$



en Python

Code pour définir un noeud

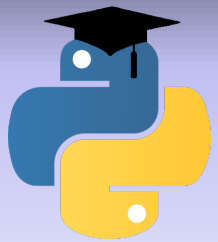
```
class Node(object):

    ...

    ...
```



en Python

Code pour définir un noeud

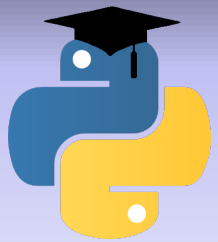
```
class Node(object):
    def __init__(self, valeur = 0):


        ...


        ...
```



en Python

Code pour définir un noeud

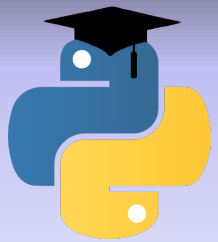
```
class Node(object):
    def __init__(self, valeur = 0):
        self.val = valeur



    ...



    ...
```



en Python

Code pour définir un noeud

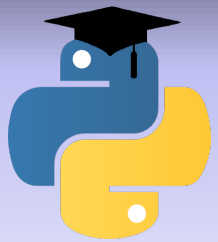
```
class Node(object):
    def __init__(self, valeur = 0):
        self.val = valeur
        self.right = None
        self.left = None
        # self.height = 0
    ...



    ...
```



en Python

Code pour définir un noeud

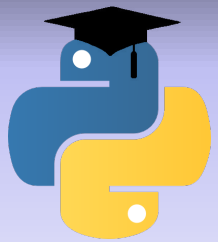
```
class Node(object):
    def __init__(self, valeur = 0):
        self.val = valeur
        self.right = None
        self.left = None
        # self.height = 0
    ...

class Arbre(object):
    def __init__(self):

    ...
```



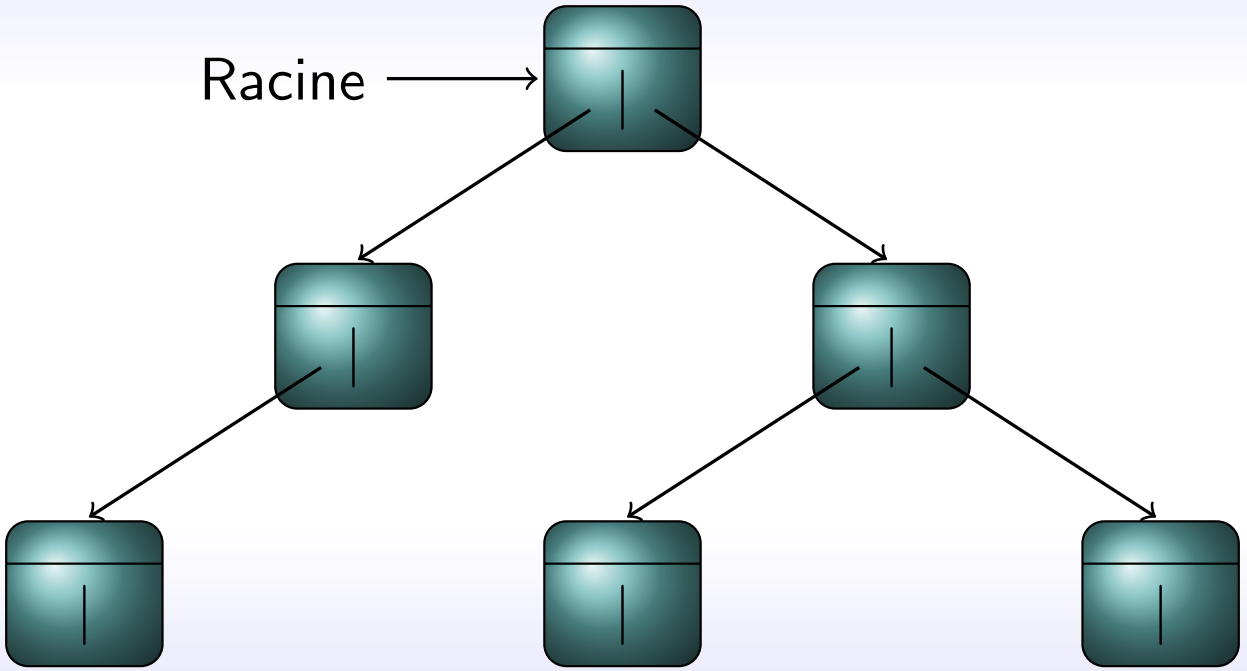
en Python

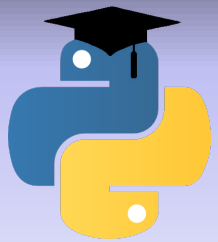
Code pour définir un noeud

```
class Node(object):
    def __init__(self, valeur = 0):
        self.val = valeur
        self.right = None
        self.left = None
        # self.height = 0
    ...

class Arbre(object):
    def __init__(self):
        self.__root = None
    ...
```



en Python

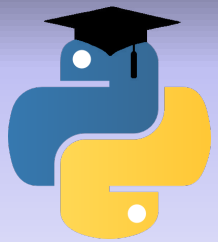




Généralisation

Arbres **n-aires** :

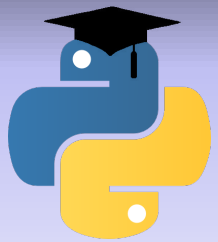
- chaque noeud possède au plus n fils.



Généralisation

Arbres **n-aires** :

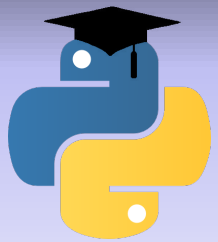
- chaque noeud possède au plus n fils.
- En **Python** : chaque noeud possède une liste de n fils



Généralisation

Arbres **n-aires** :

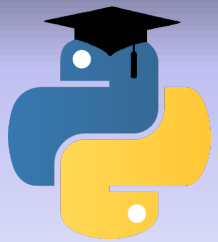
- chaque noeud possède au plus n fils.
- En **Python** : chaque noeud possède une liste de n fils
- Moins utile que les arbres binaires



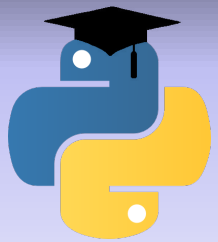
Généralisation

Arbres **n-aires** :

- chaque noeud possède au plus n fils.
- En **Python** : chaque noeud possède une liste de n fils
- Moins utile que les arbres binaires
- Exemple d'utilisation : **quadtree**



Sommaire



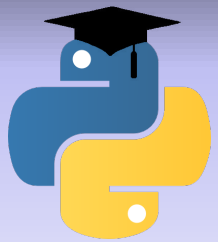
Définition

Définition (Arbre général)

Arbre pour lequel chaque noeud possède un nombre quelconque de fils (non borné **a priori**).

Exemple (Exemples d'utilisation)

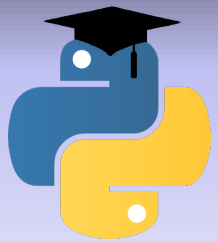
- Représentation d'une arborescence de répertoires
- Représentation des possibilités pour un jeu



Définition

Code pour définir un noeud

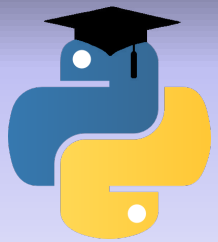
```
class Node(object):

    def __init__(self, valeur = 0):
```



Définition

Code pour définir un noeud

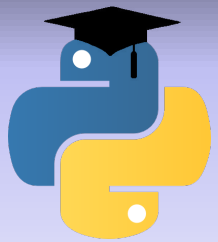
```
class Node(object):

    def __init__(self, valeur = 0):
        self.val = valeur
```



Définition

Code pour définir un noeud

```
class Node(object):

    def __init__(self, valeur = 0):
        self.val = valeur
        self.child = None
```

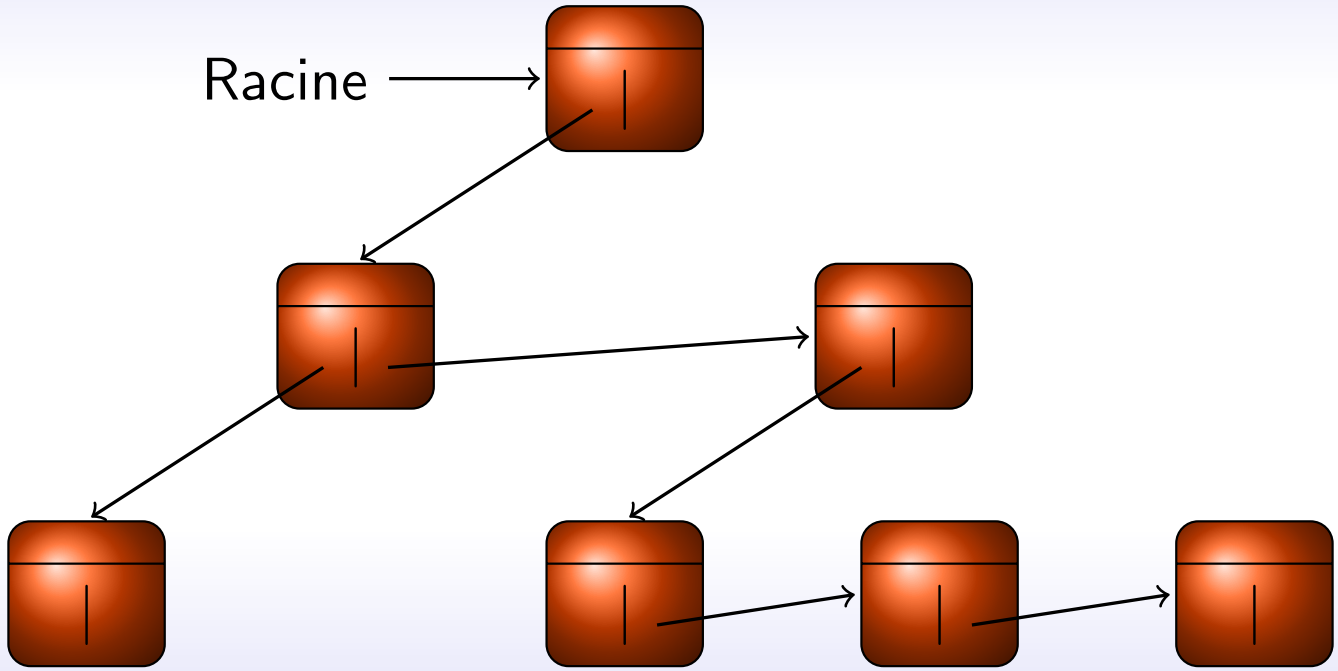


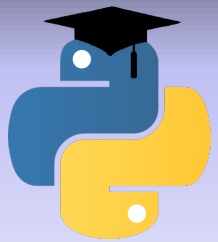
Définition

Code pour définir un noeud

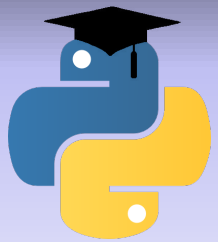
```
class Node(object):

    def __init__(self, valeur = 0):
        self.val = valeur
        self.child = None
        self.brothers = []
```

Définition

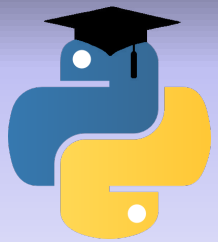




Sommaire



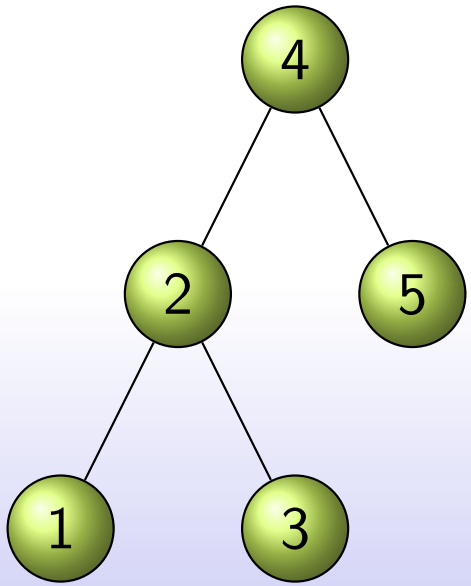
Sommaire

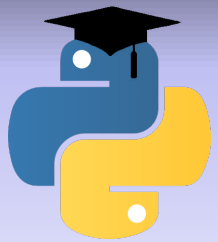


Définition

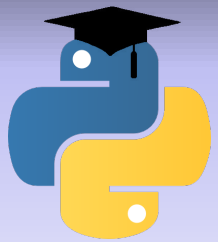
Définition (Arbre binaire de recherche)

Arbre binaire vérifiant la propriété : pour tout noeud n de l'arbre, tous les noeuds de son sous-arbre gauche sont inférieurs à n, tous les noeuds de son sous-arbre droit sont supérieurs à n.





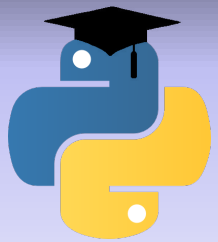
Sommaire



Recherche

Rechercher un élément :

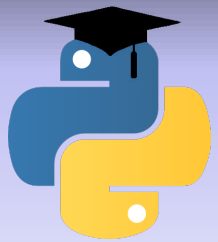
- partir de la racine



Recherche

Rechercher un élément :

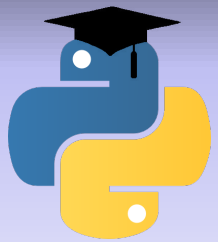
- partir de la racine
- comparer la valeur recherchée à l'élément courant



Recherche

Rechercher un élément :

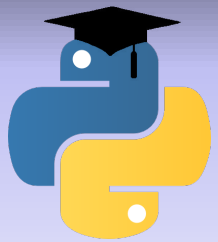
- partir de la racine
- comparer la valeur recherchée à l'élément courant
    - si elle est égale : fin



Recherche

Rechercher un élément :

- partir de la racine
- comparer la valeur recherchée à l'élément courant
  - si elle est égale : fin
  - si elle est inférieure : chercher dans le sous-arbre gauche

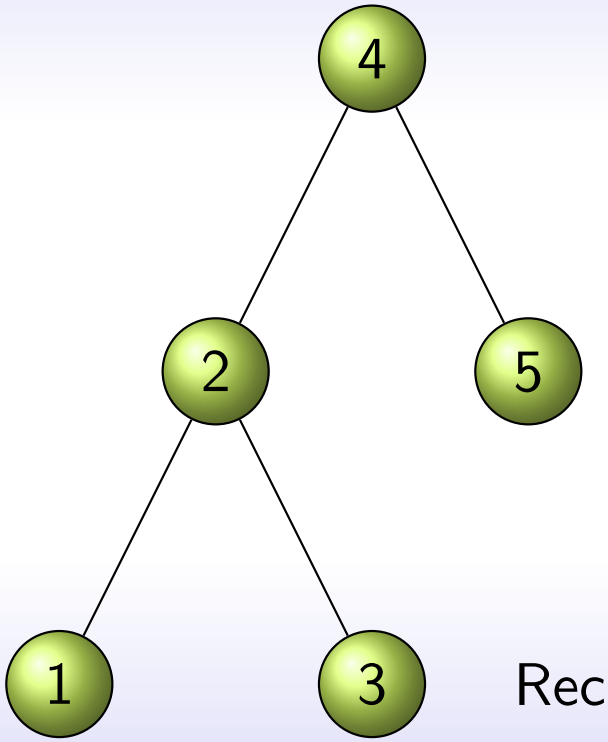


Recherche

Rechercher un élément :

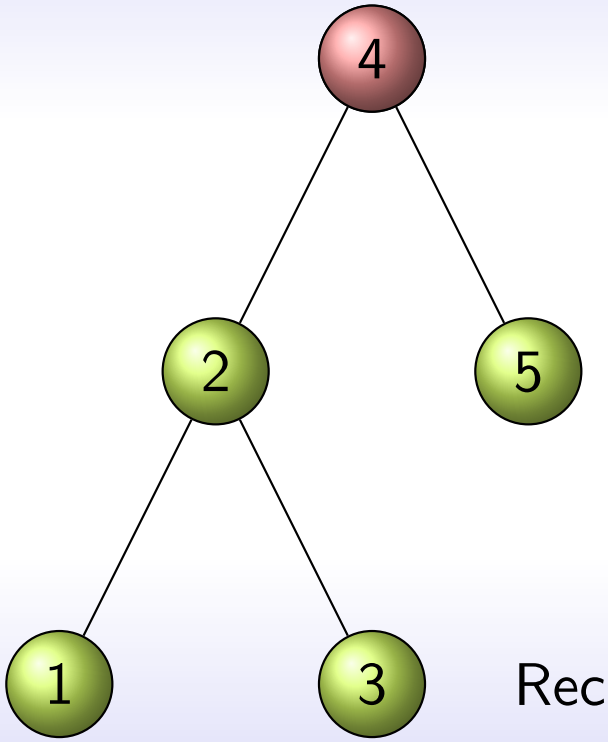
- partir de la racine
- comparer la valeur recherchée à l'élément courant
  - si elle est égale : fin
  - si elle est inférieure : chercher dans le sous-arbre gauche
  - sinon : chercher dans le sous-arbre droit

Exemple



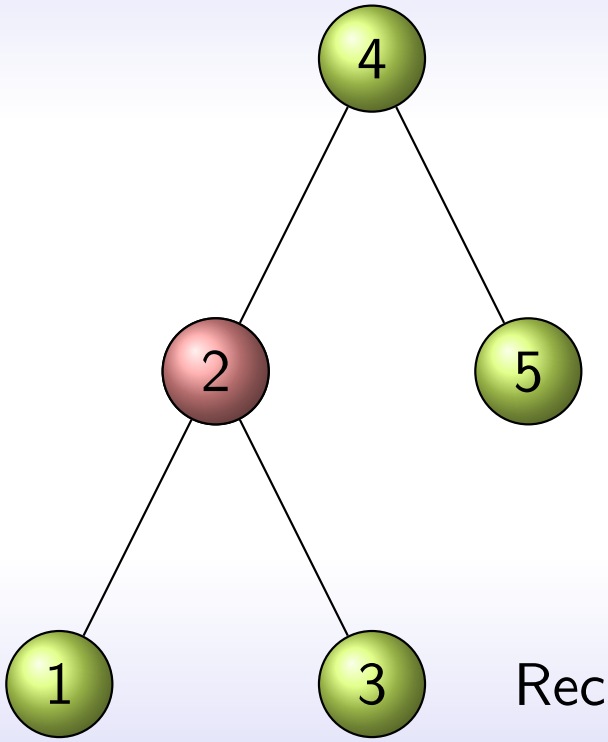
Recherche : 3

Exemple



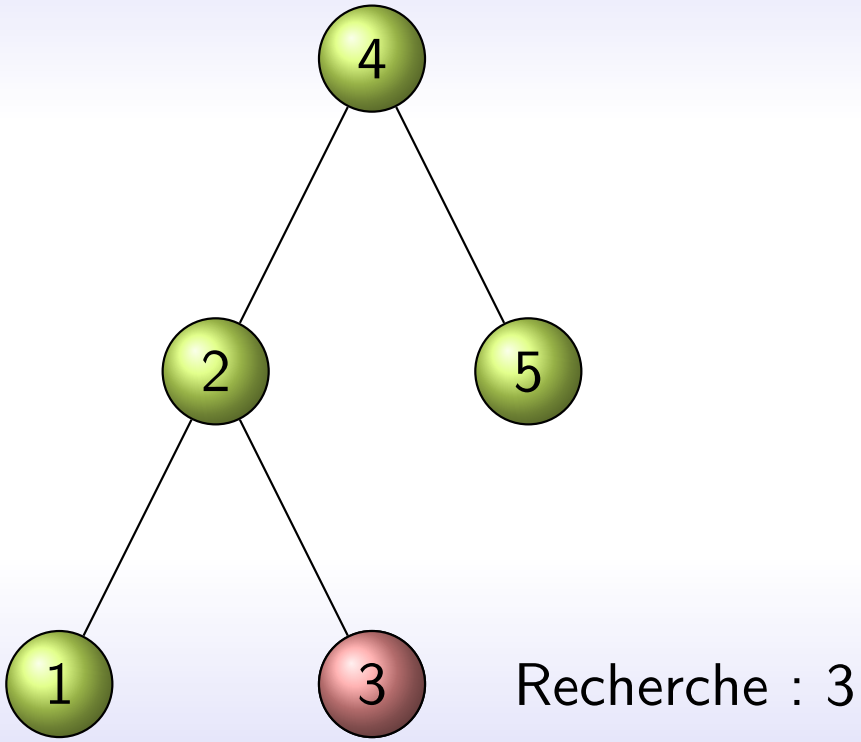
Recherche : 3

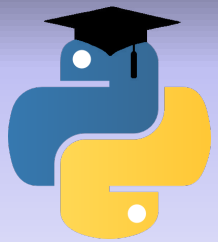
Exemple



Recherche : 3

Exemple

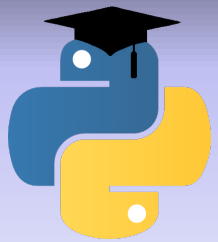




Complexité

Étude de l'algorithme :

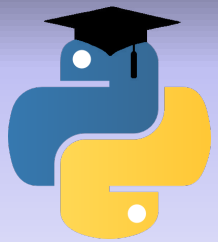
- pire des cas : aller jusqu'à une feuille



Complexité

Étude de l'algorithme :

- pire des cas : aller jusqu'à une feuille
- complexité hauteur de l'arbre

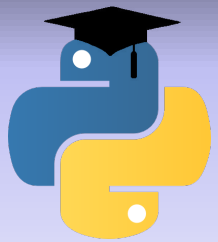


Complexité

Étude de l'algorithme :

- pire des cas : aller jusqu'à une feuille
- complexité hauteur de l'arbre

$\Rightarrow$ arbre complet : $\Theta(\log n)$



Complexité

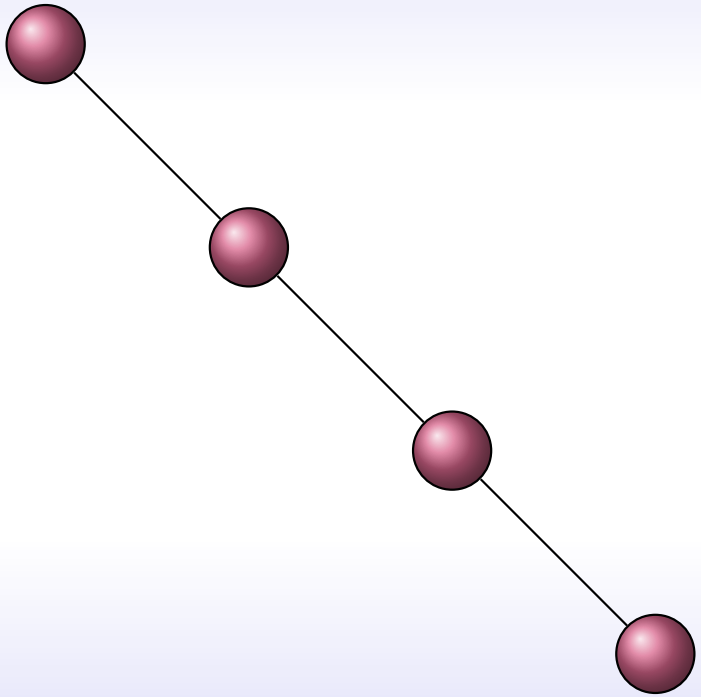
Étude de l'algorithme :

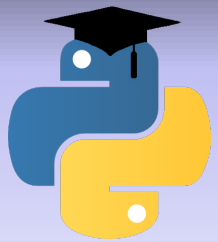
- pire des cas : aller jusqu'à une feuille
- complexité hauteur de l'arbre

$\Rightarrow$ arbre complet : $\Theta(\log n)$

- arbre le pire : peigne

Complexité





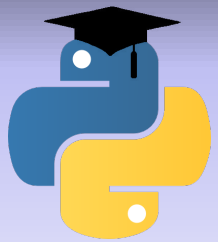
Complexité

Étude de l'algorithme :

- pire des cas : aller jusqu'à une feuille
- complexité hauteur de l'arbre

$\Rightarrow$ arbre complet : $\Theta(\log n)$

- arbre le pire : peigne
- complexité : $\Theta(n)$



Algorithme itératif

Algorithme 1: Recherche itérative dans un ABR

Entrées : entier val

$n \leftarrow racine$;

tant que $n \neq None$ **et** $n.val \neq val$ **faire**

    si $val < n.val$ **alors**

        $n \leftarrow n.filsGauche$;

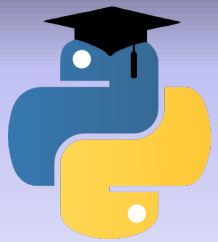
    sinon

        $n \leftarrow n.filsDroit$;

    fin

fin

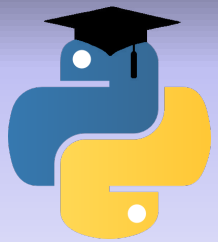
retourner n `// n a la valeur recherchée ou n vaut None`



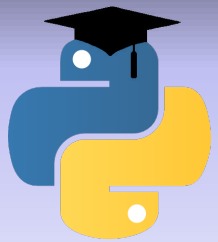
Algorithme récursif

Fonction rechRec(entier val, Noeud cour)

si *cour* = *None* **ou** *cour.val* = *val* **alors**
    retourner cour;
sinon si *val* < *cour.val* **alors**
    retourner rechRec(val, n.filsGauche);
sinon
    retourner rechRec(val, n.filsDroit);
fin

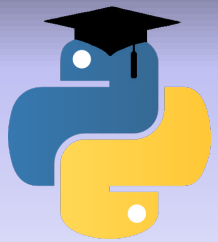


Sommaire



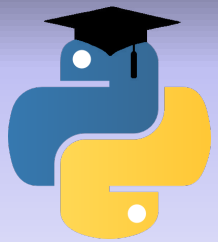
Principe

- parcourir un arbre : appliquer un traitement à tous les noeuds



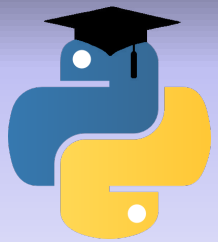
Principe

- parcourir un arbre : appliquer un traitement à tous les noeuds
- algorithme itératif : difficile à écrire



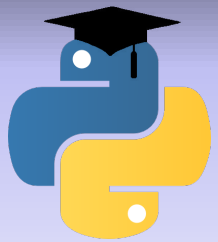
Principe

- parcourir un arbre : appliquer un traitement à tous les noeuds
- algorithme itératif : difficile à écrire
- algorithme récursif : très simple



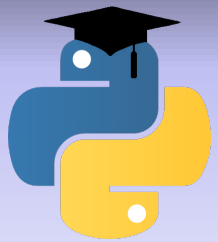
Principe

- parcourir un arbre : appliquer un traitement à tous les noeuds
- algorithme itératif : difficile à écrire
- algorithme récursif : très simple
  - principe : 3 traitements $\Rightarrow$ 2 sous-arbres et noeud courant



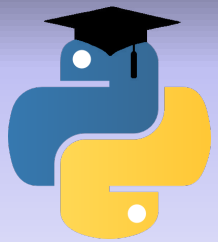
Principe

- parcourir un arbre : appliquer un traitement à tous les noeuds
- algorithme itératif : difficile à écrire
- algorithme récursif : très simple
  - principe : 3 traitements $\Rightarrow$ 2 sous-arbres et noeud courant
  - noeud courant avant le fils gauche $\Rightarrow$ préfixe



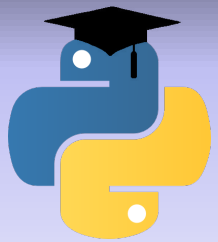
Principe

- parcourir un arbre : appliquer un traitement à tous les noeuds
- algorithme itératif : difficile à écrire
- algorithme récursif : très simple
  - principe : 3 traitements $\Rightarrow$ 2 sous-arbres et noeud courant
  - noeud courant avant le fils gauche $\Rightarrow$ préfixe
  - noeud courant entre les fils $\Rightarrow$ infixe

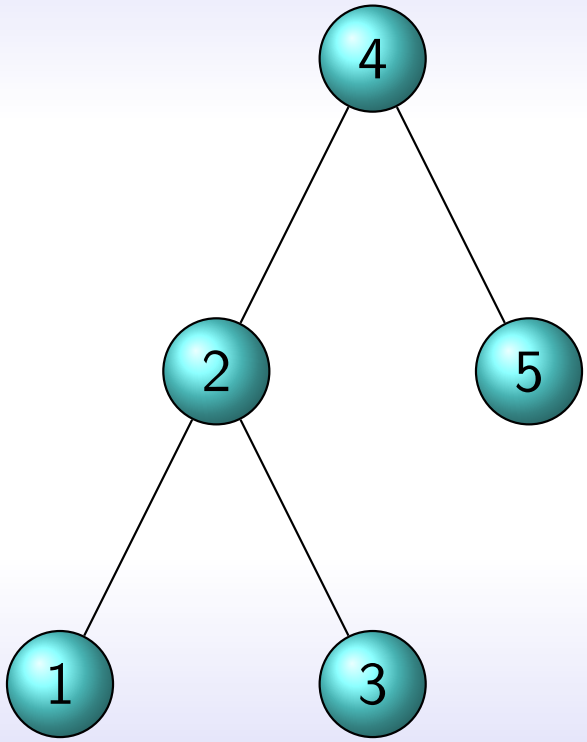


Principe

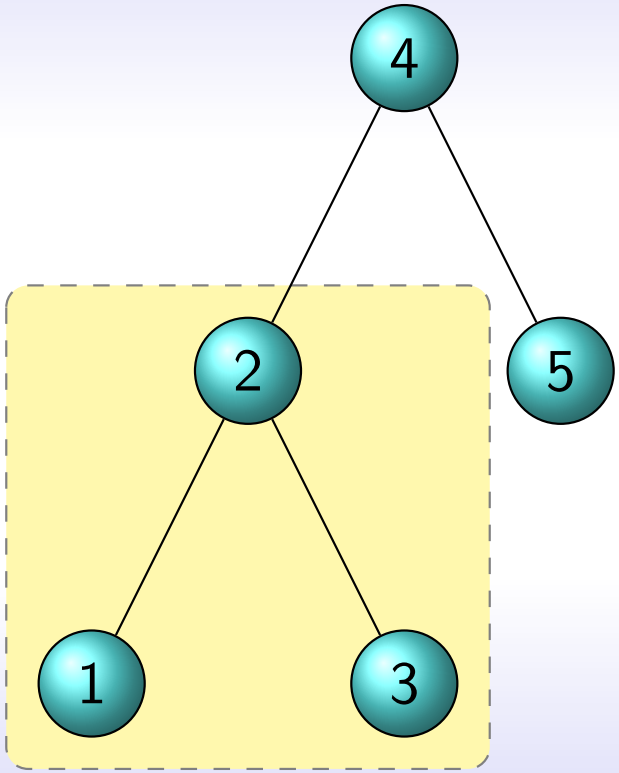
- parcourir un arbre : appliquer un traitement à tous les noeuds
- algorithme itératif : difficile à écrire
- algorithme récursif : très simple
  - principe : 3 traitements $\Rightarrow$ 2 sous-arbres et noeud courant
  - noeud courant avant le fils gauche $\Rightarrow$ préfixe
  - noeud courant entre les fils $\Rightarrow$ infixe
  - noeud courant après le fils droit $\Rightarrow$ postfixe



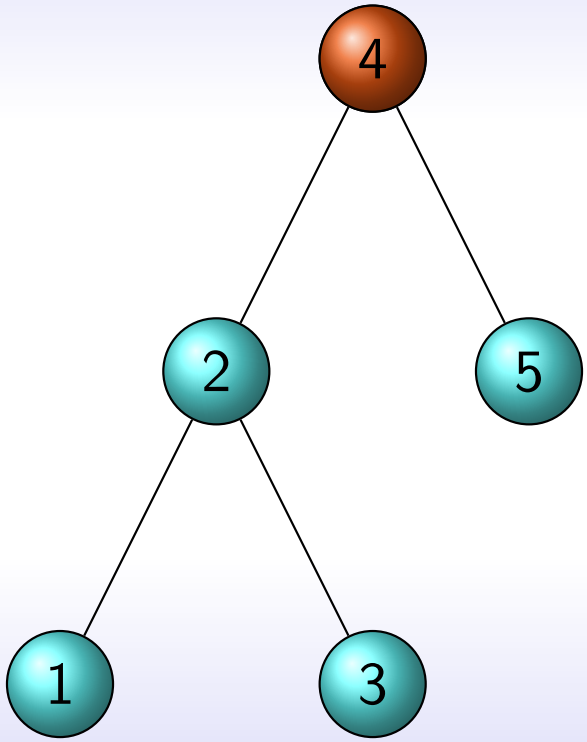
Exemple : parcours infixe



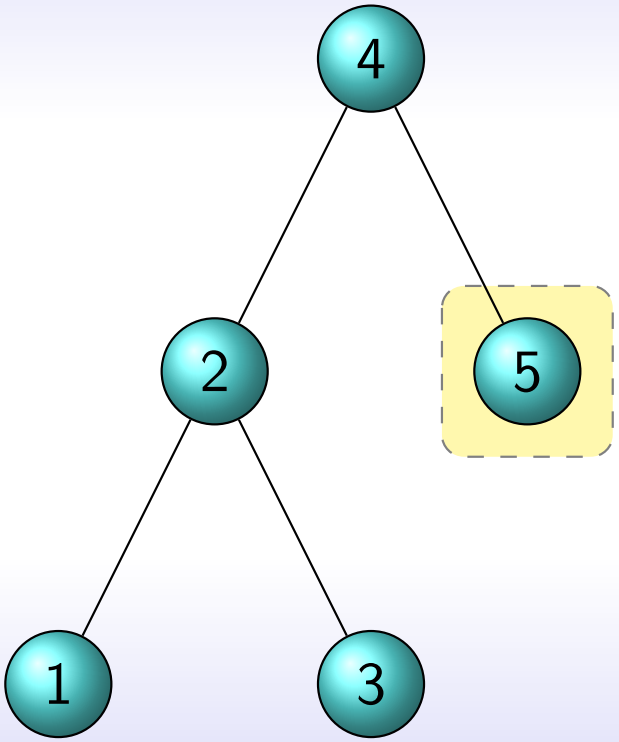
Exemple : parcours infixe



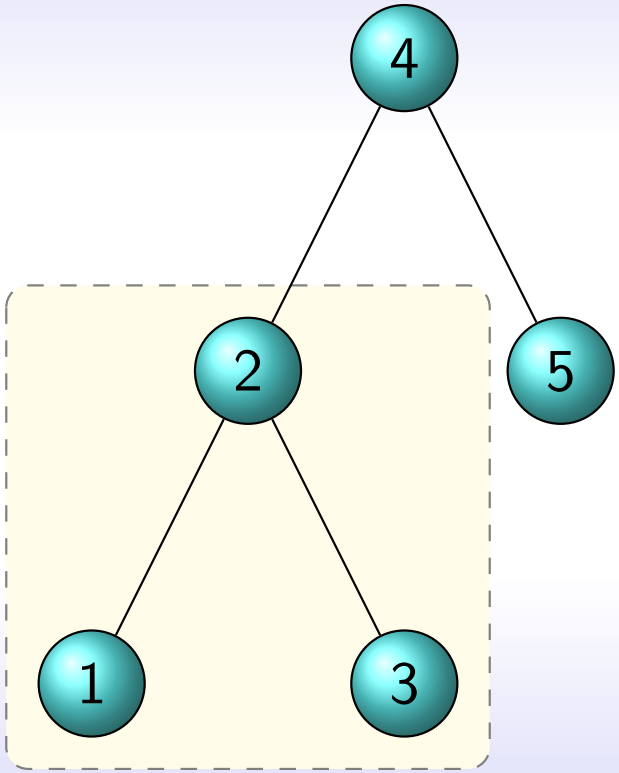
Exemple : parcours infixe



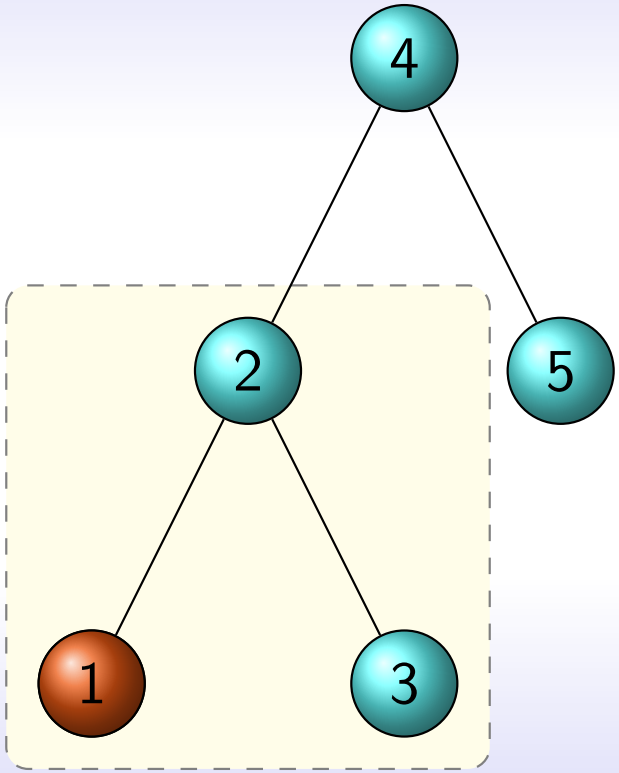
Exemple : parcours infixe

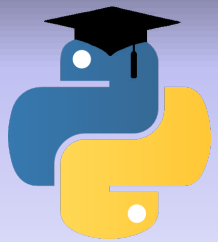


Exemple : parcours infixe

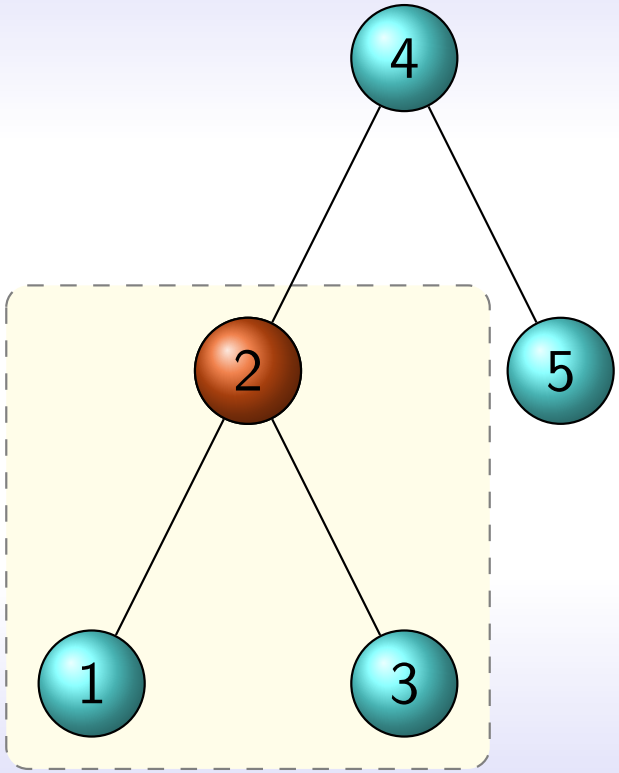


Exemple : parcours infixe

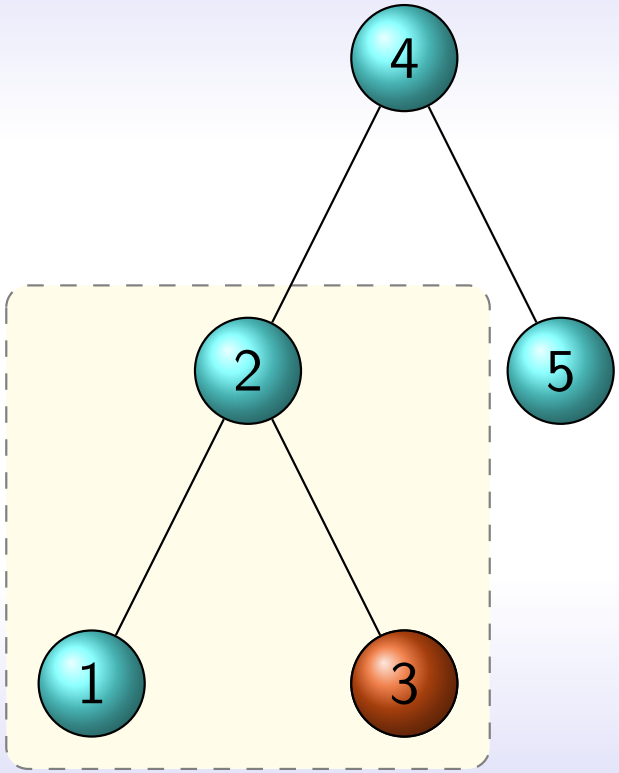


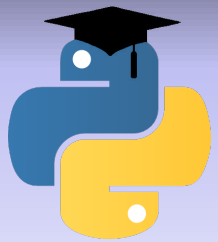


Exemple : parcours infixe



Exemple : parcours infixe

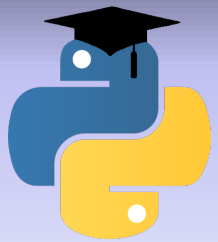




Algorithmes

Procédure infixe(Noeud cour)

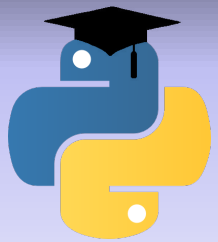
si *cour* $\neq$ *None* **alors**
    infixe(cour.filsGauche);
    traiter **cour** ;
    infixe(cour.filsDroit);
fin



Algorithmes

Procédure prefixe(Noeud cour)

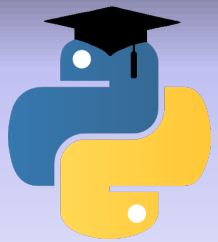
si *cour* $\neq$ *None* **alors**
    traiter **cour** ;
    prefixe(cour.filsGauche);
    prefixe(cour.filsDroit);
fin



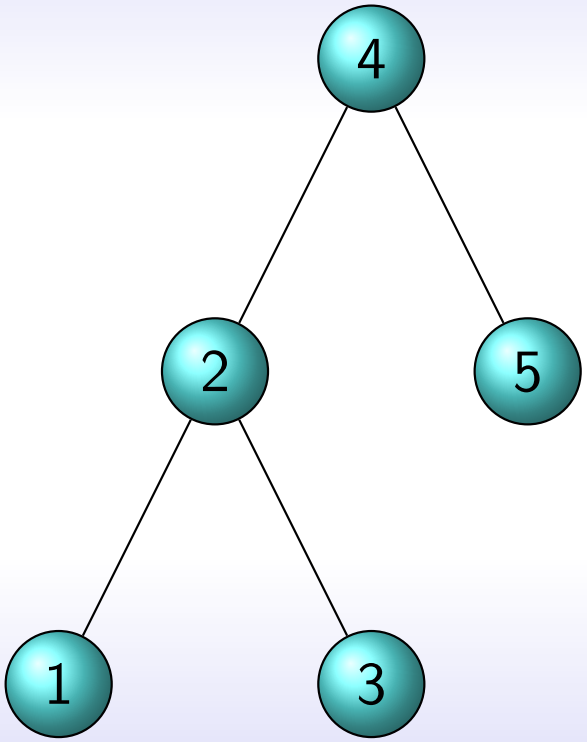
Algorithmes

Procédure postfixe(Noeud cour)

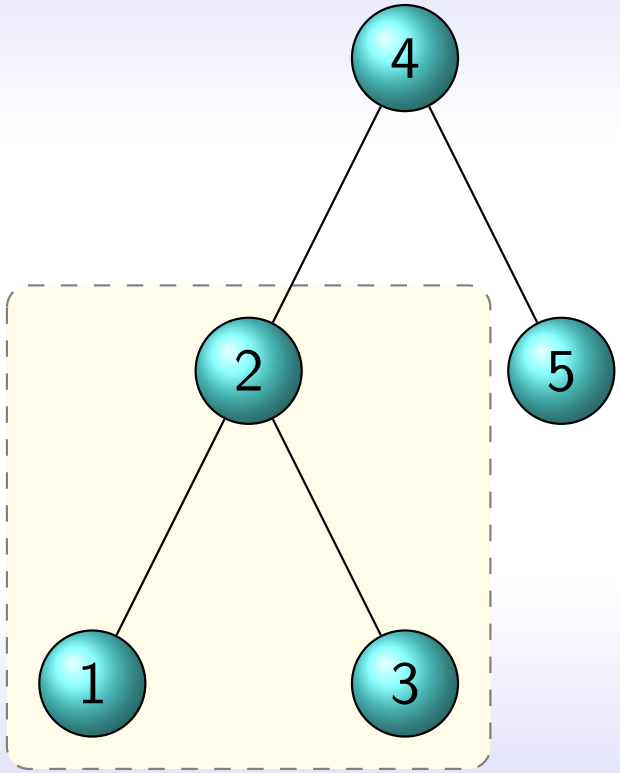
si *cour* $\neq$ *None* **alors**
    postfixe(cour.filsGauche);
    postfixe(cour.filsDroit);
    traiter **cour** ;
fin

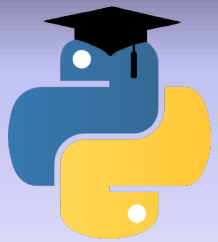


Exemple : parcours infixe

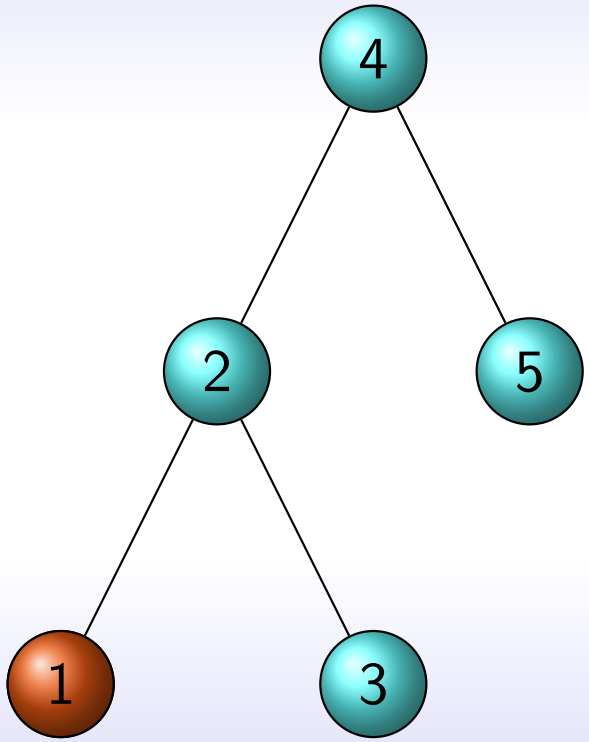


Exemple : parcours infixe



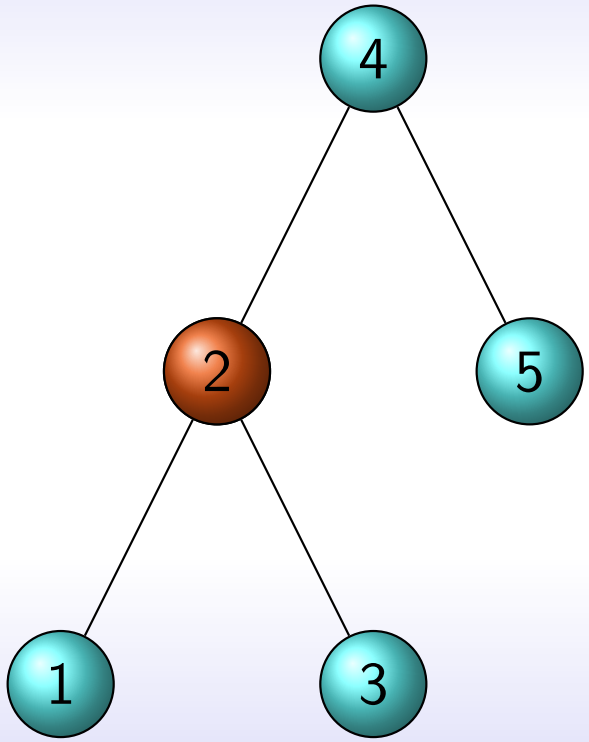


Exemple : parcours infixe



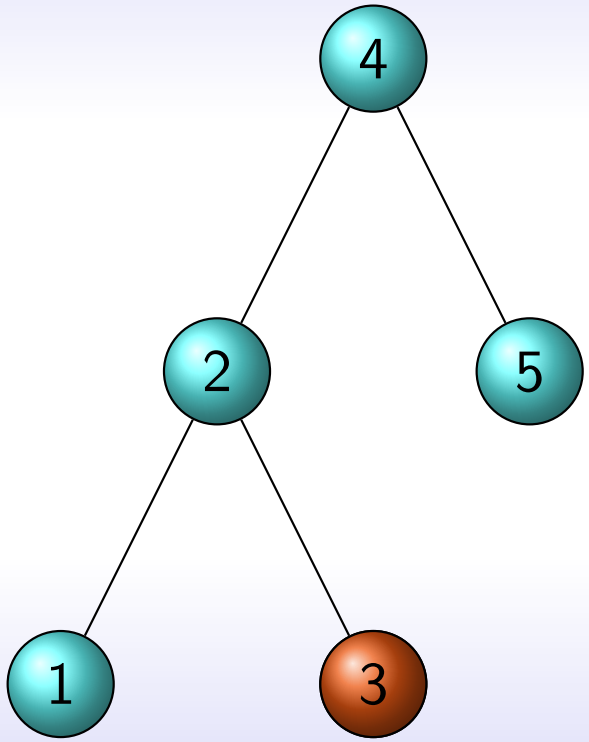
1

Exemple : parcours infixe



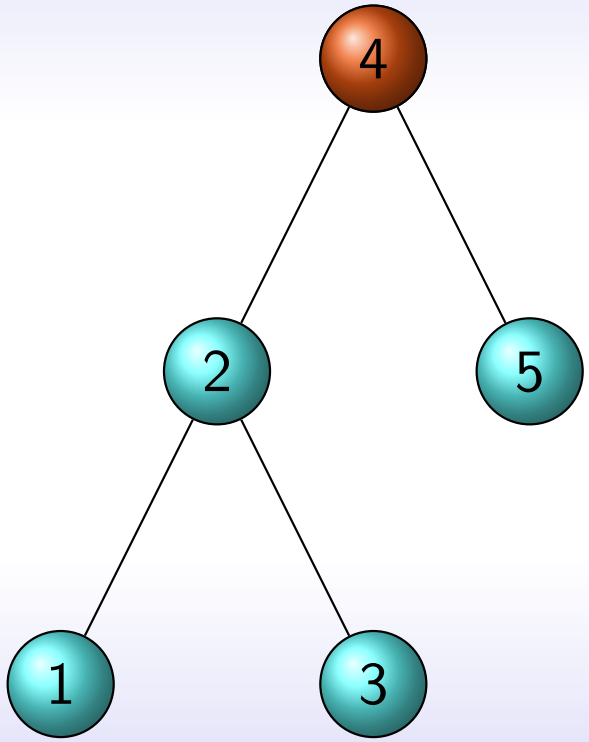
1 2

Exemple : parcours infixe



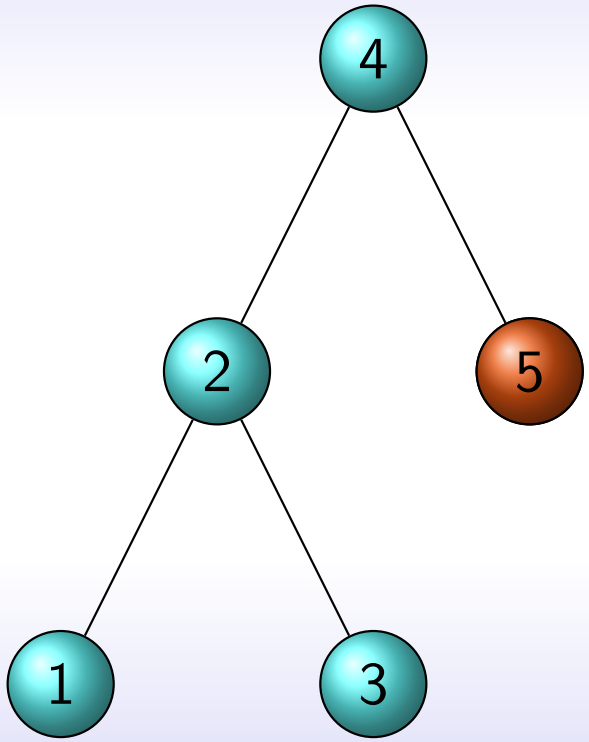
1 2 3

Exemple : parcours infixe



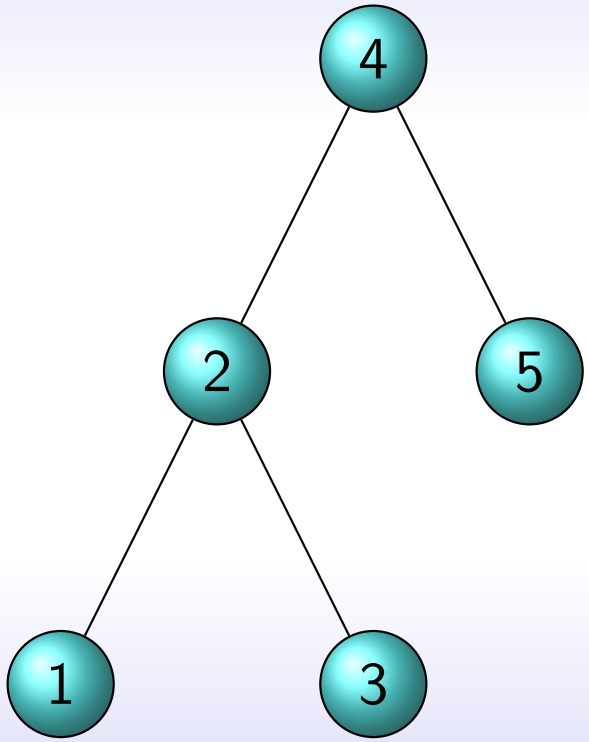
1 2 3 4

Exemple : parcours infixe

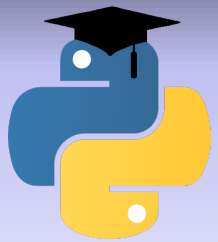


1 2 3 4 5

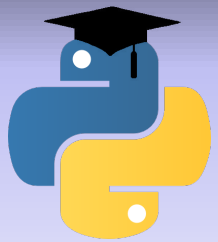
Exemple : parcours infixe



1 2 3 4 5



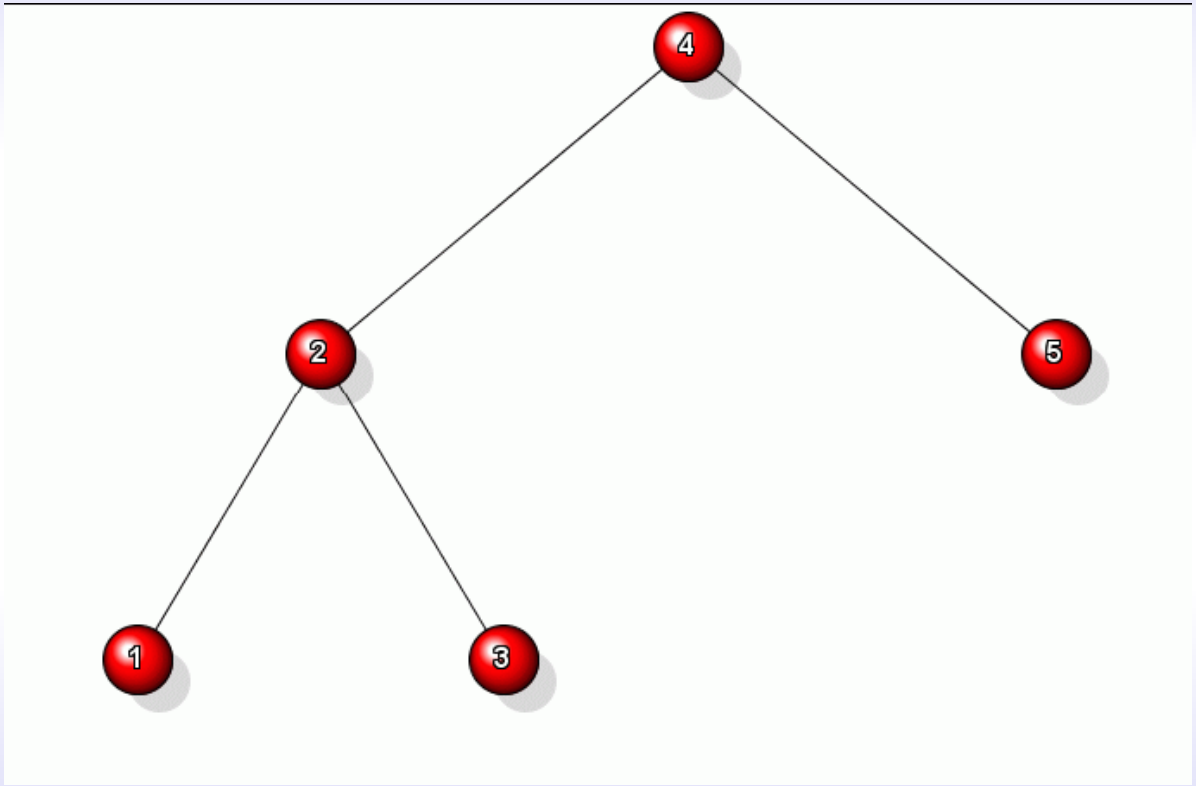
Sommaire

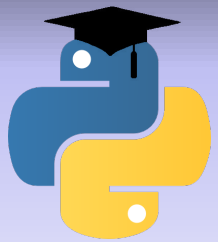


Principe

- Insertion uniquement sur les feuilles
- Principe :
  - se placer sur la bonne feuille (déplacement itératif ou récursif)
  - ajouter le fils
- principe similaire à l'insertion dans une liste

Demo

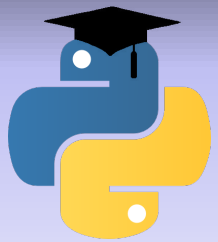




Algorithme

Algorithme 2: Insertion dans un arbre binaire de recherche

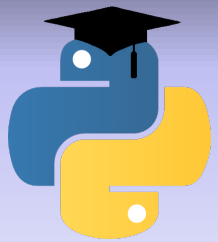
Entrées : noeud val
si *arbre* $= \emptyset$ **alors**
  racine $\leftarrow$ *val*;
sinon
  insert(val, racine) `// insère val sous la racine`
fin



Algorithme

Procédure insert(noeud val,noeud pere)

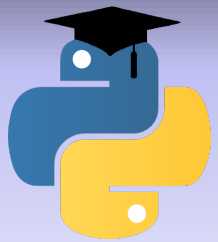
```
n ← pere;
tant que n ≠ None faire
    pere ← n;
    si val < n alors
        n ← n.filsGauche;
    sinon
        n ← n.filsDroit;
    fin
fin
// maintenant le fils de pere est None
```



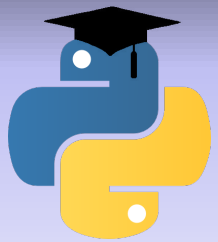
Algorithme

Procédure insert(noeud val,noeud pere)

```
// ... maintenant le fils de pere est None
si val < pere alors
    pere.filsGauche ← val;
sinon
    pere.filsDroit ← val;
fin
```



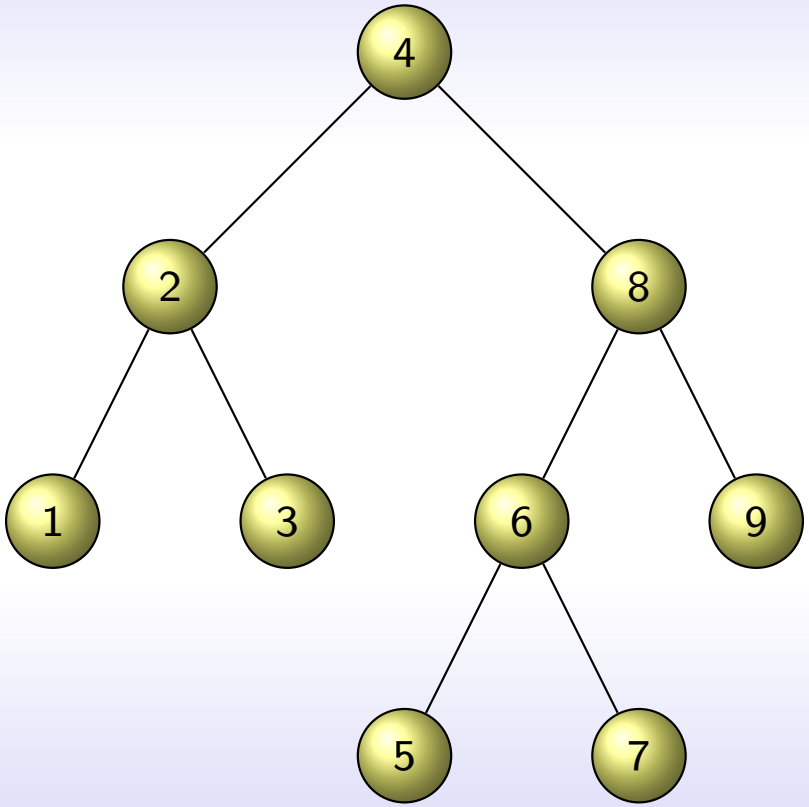
Sommaire

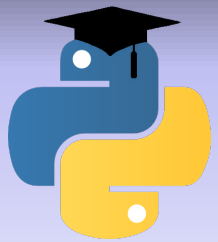


Principe

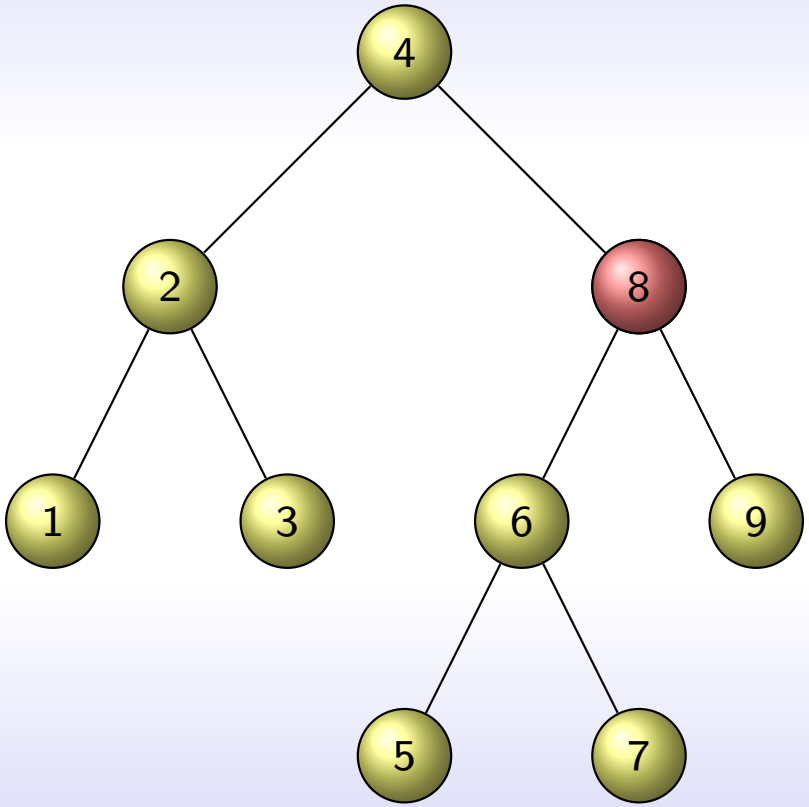
- se placer au dessus de l'élément à supprimer
- remplacer le noeud par le sous-arbre gauche
- réinsérer le sous-arbre droit

Exemple

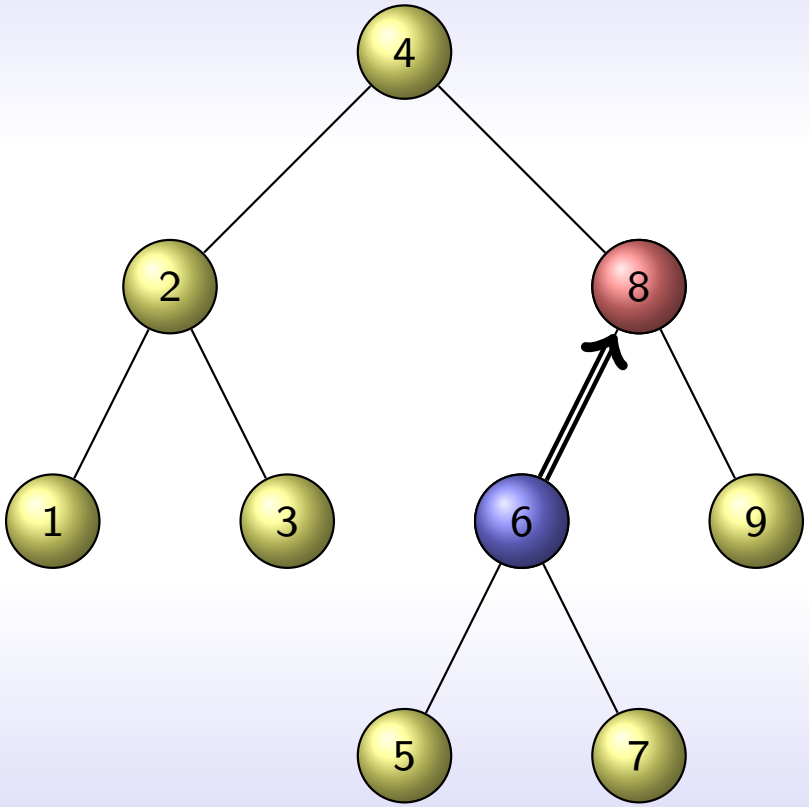


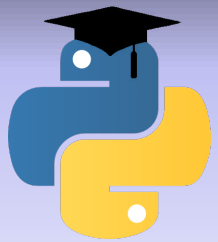


Exemple

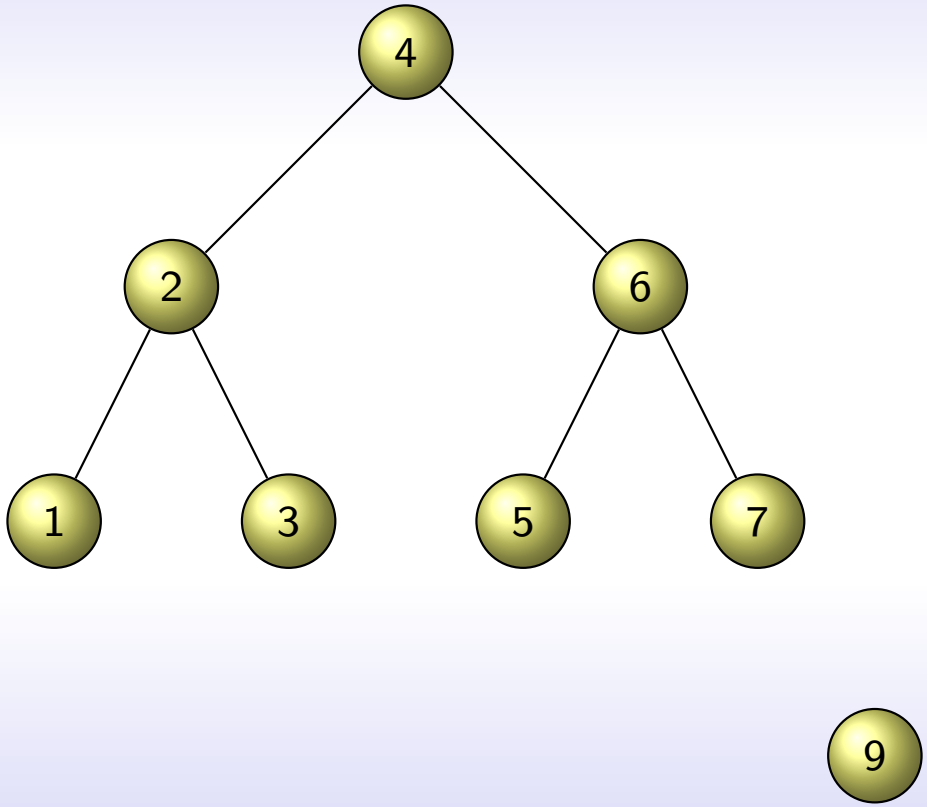


Exemple

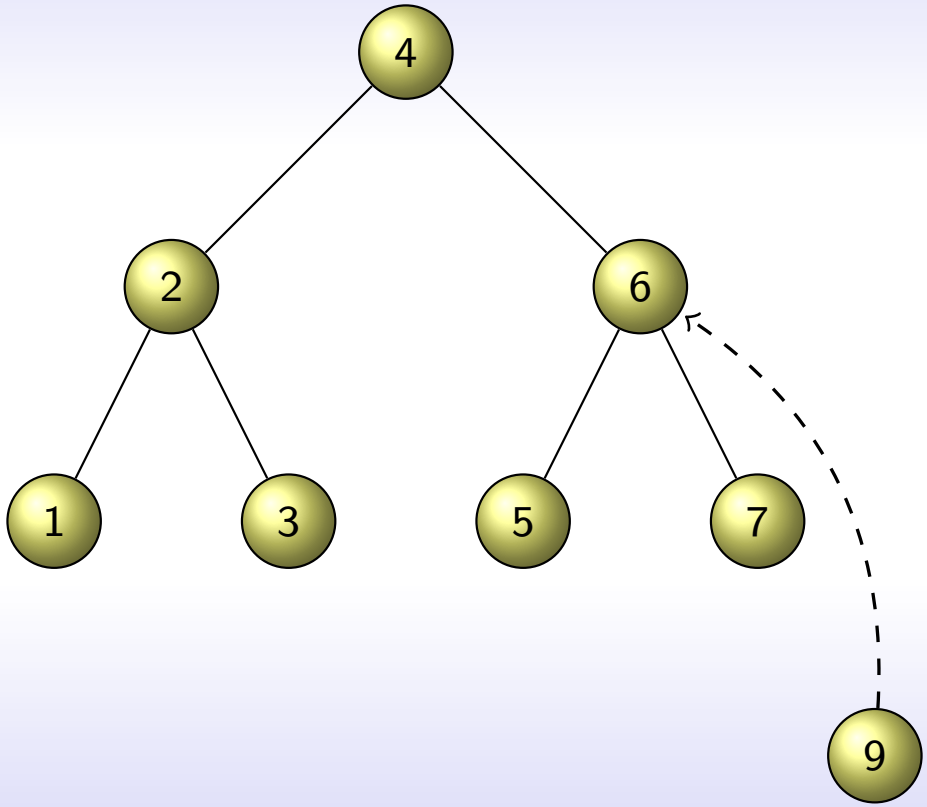


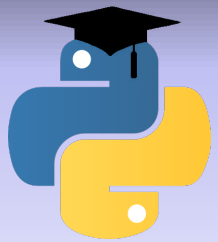


Exemple

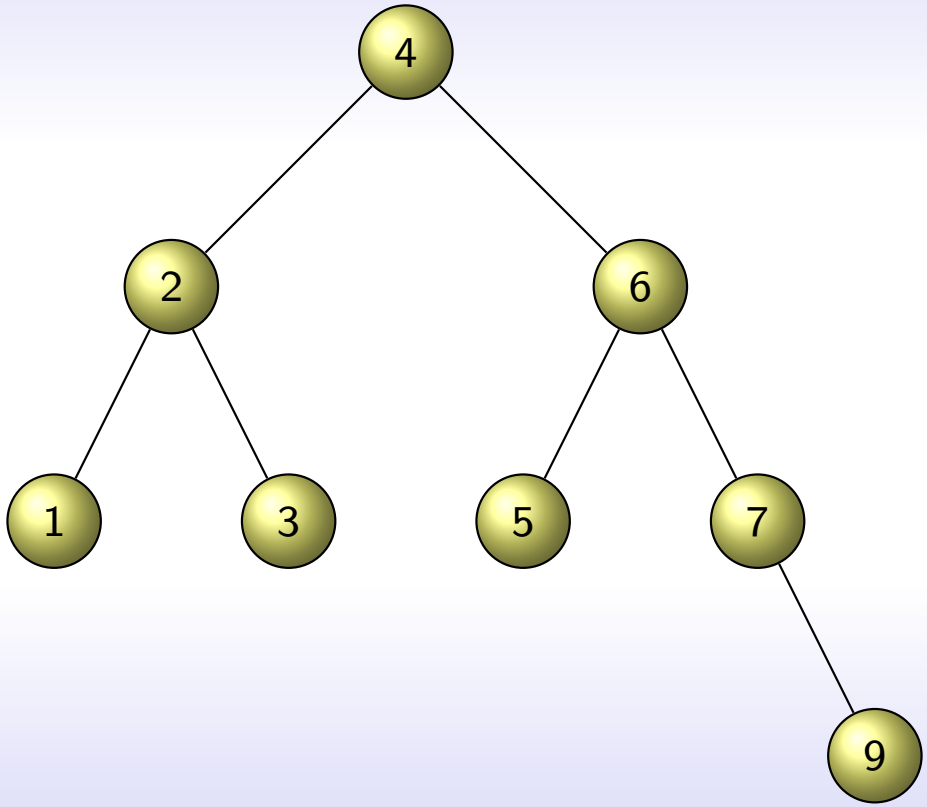


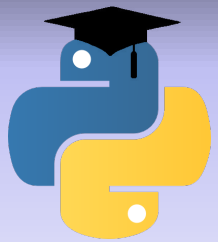
Exemple



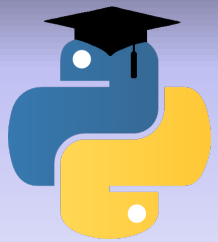


Exemple

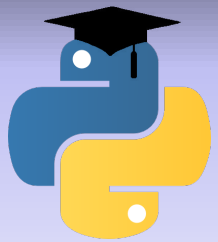




Sommaire



Sommaire



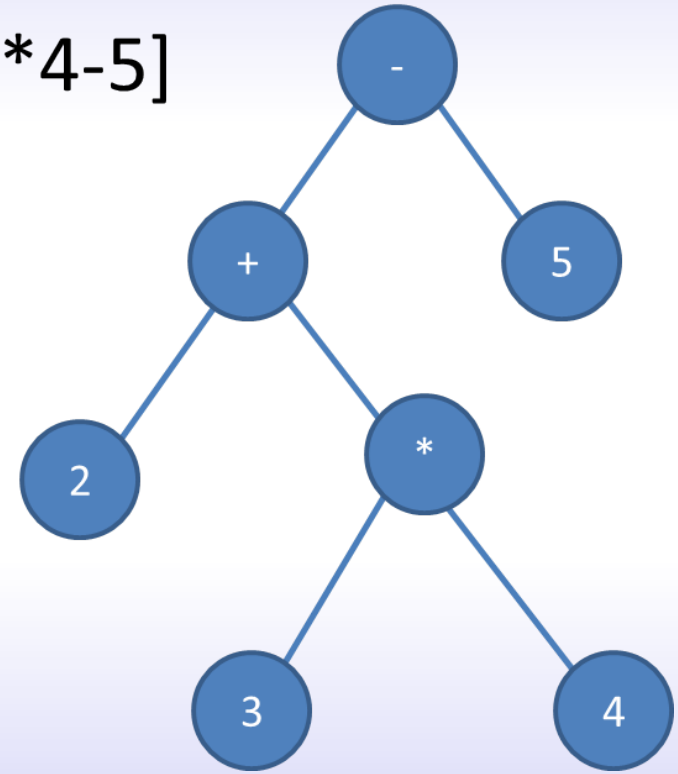
Arbre syntaxique

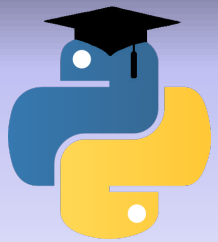
- Arbre représentant un phrase
- ou un programme ou une partie d'un programme
- exemple : expression arithmétique
- Sert à analyser la phrase/le programme

Exemple : $2 + 3 * 4 - 5$

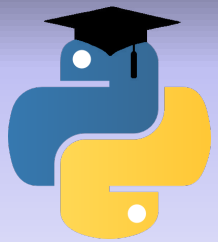
Arbre syntaxique

[2+3*4-5]





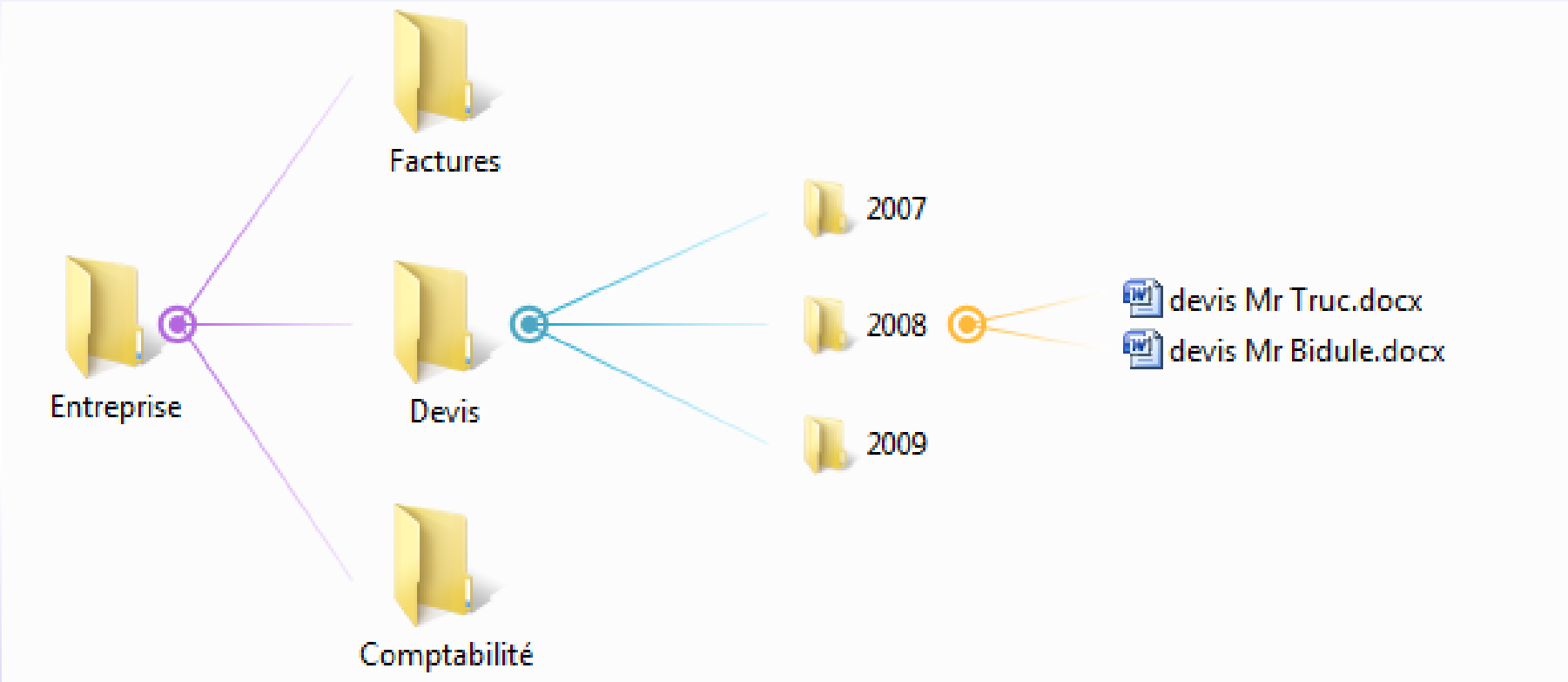
Sommaire

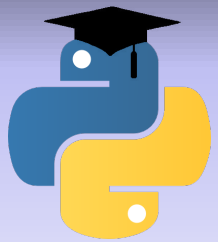


Fichiers

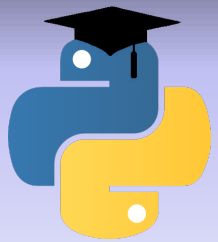
- Représentation sous forme d'arbre général
- Fichiers : feuilles
- Répertoires : noeuds intermédiaires

Fichiers


Factures
Entreprise
Devis
2007
2008
2009
devis Mr Truc.docx
devis Mr Bidule.docx
Comptabilité



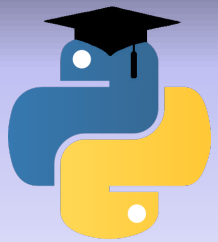
Sommaire



Intérêt

- Arbre binaire : efficace si complet
- Arbre binaire complet : pas toujours possible

$\Rightarrow$ notion d'arbre équilibré



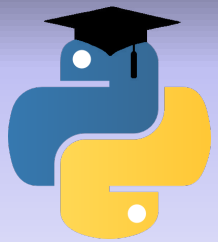
Intérêt

- Arbre binaire : efficace si complet
- Arbre binaire complet : pas toujours possible

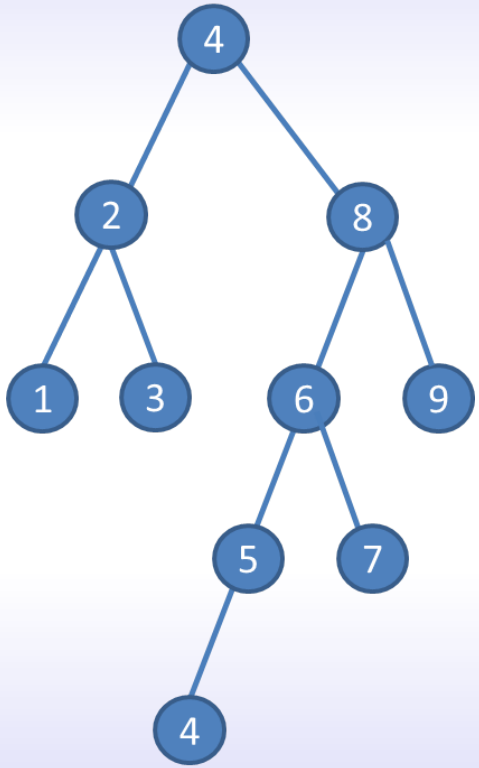
$\Rightarrow$ notion d'arbre équilibré

Définition (Arbre équilibré)

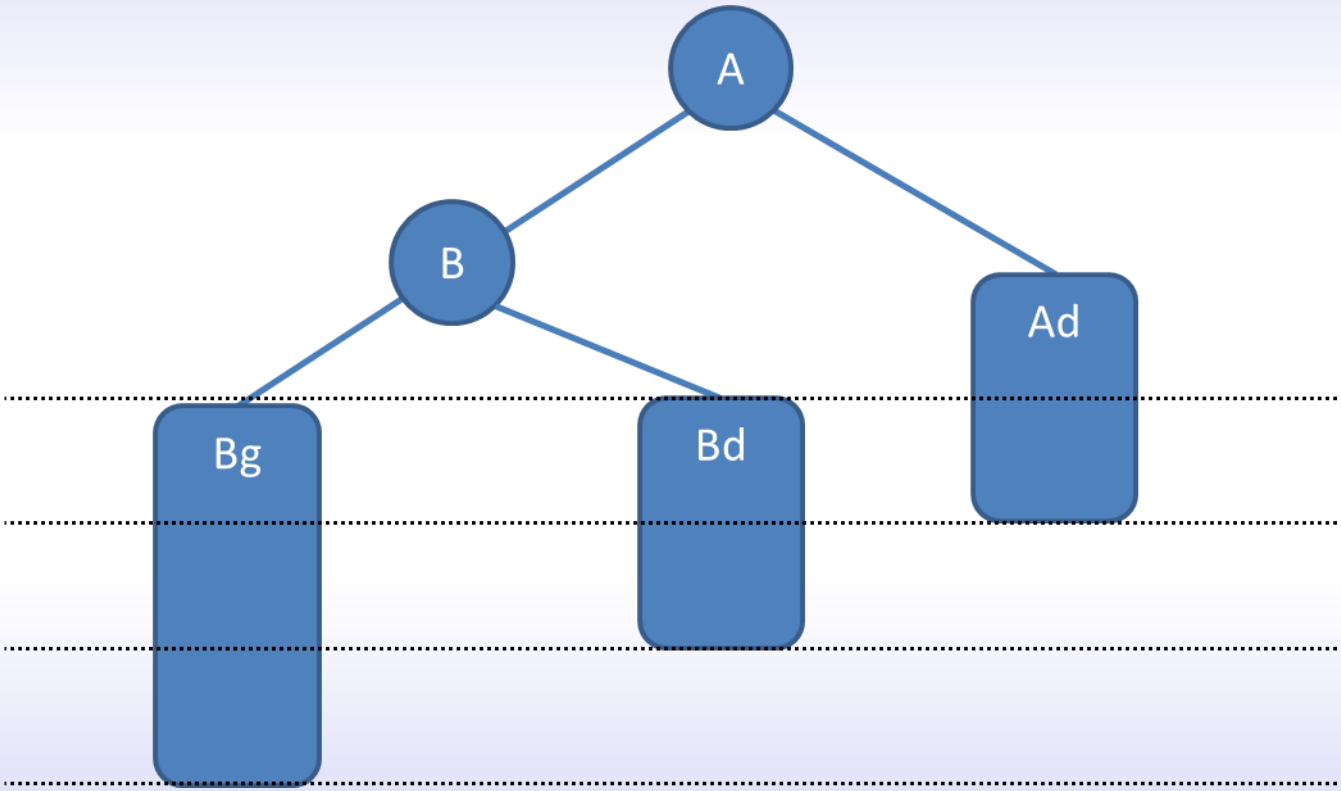
Arbre binaire est équilibré : la différence entre les hauteurs des fils gauche et droit de tout noeud ne peut excéder 1.



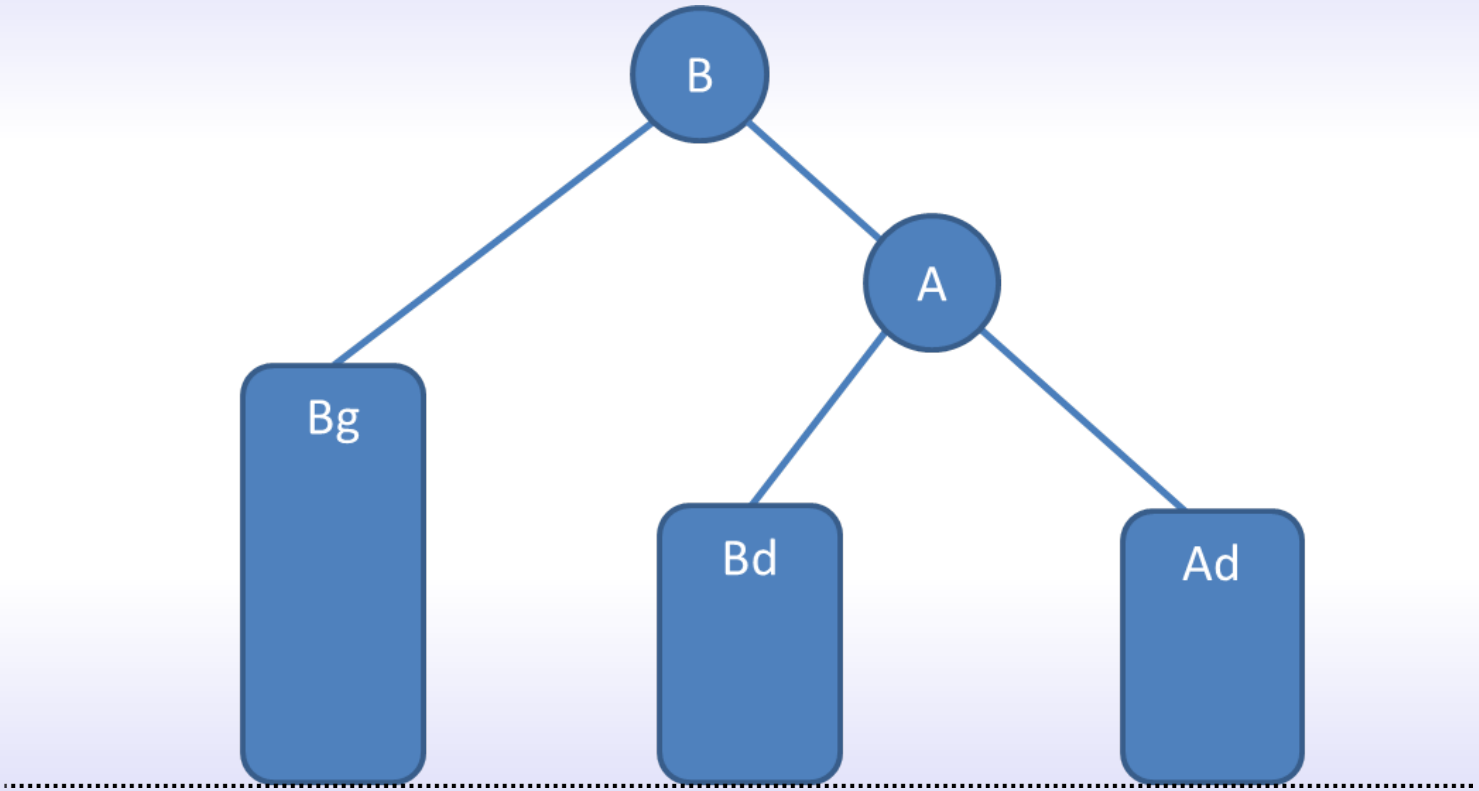
Exemple : arbre déséquilibré



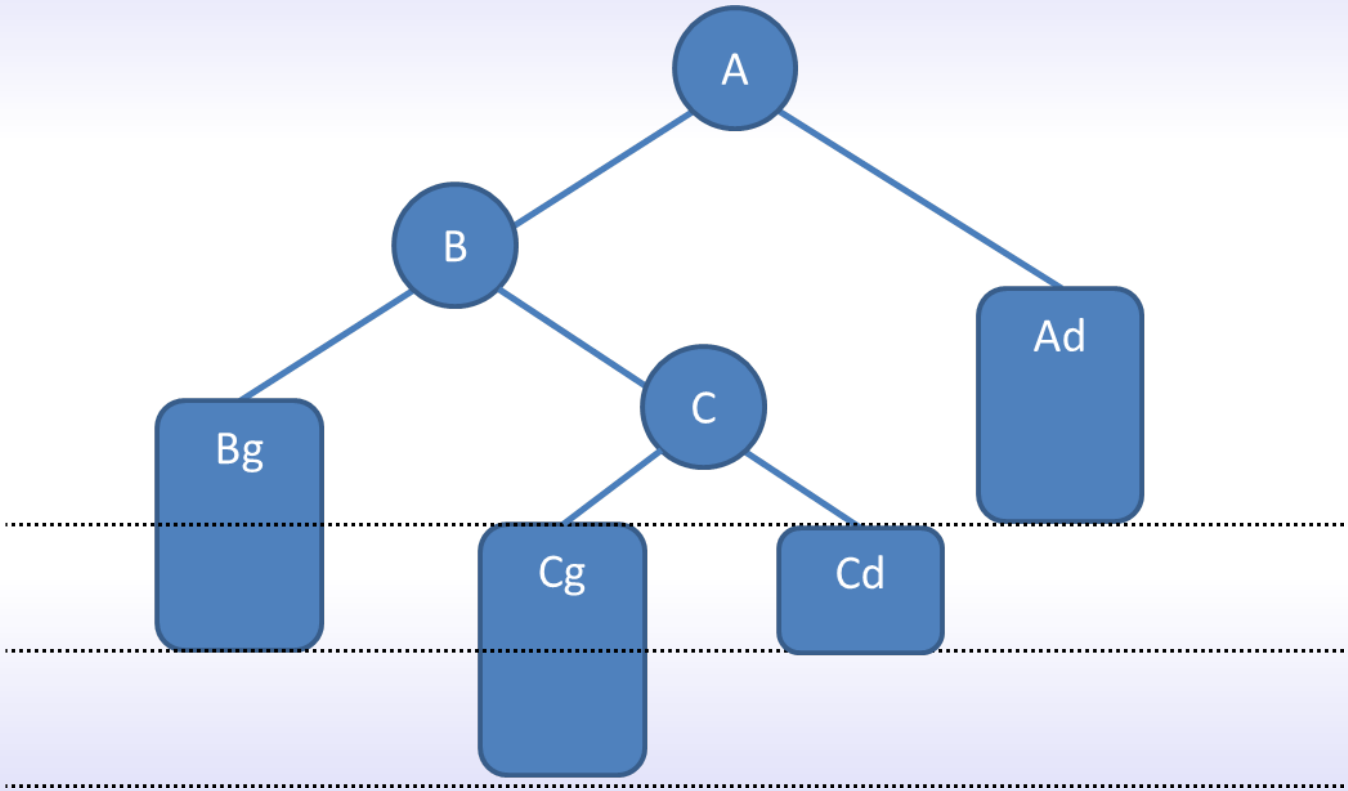
Équilibrage d'un arbre



Équilibrage d'un arbre



Équilibrage d'un arbre



Équilibrage d'un arbre