

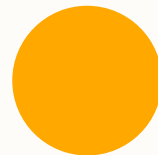


# Les piles, les files et la programmation orientée objet

---



Ilyan Mouas TNSI2





# 1

## L'objet Pile en Python





```
class Pile:
    def __init__(self, haut, reste=None):
        self.h=haut
        self.r=reste

    def estVide(self):
        if self.h==None:
            return True
        else:
            return False

    def empiler(self, valeur):
        self.r=Pile(self.h, self.r)
        self.h=valeur

    def depiler(self):
        if self.r==None:
            haut=self.h
            self.h=None
            return haut
        else:
            haut=self.h
            self.h=self.r.h
            self.r=self.r.r
            return haut

    def affiche(self):
        if self.r==None:
            print(self.h)
        else:
            Pile.affiche(self.r)
            print(self.h)
```





*Créer une pile vide, tester qu'elle est bien considérée comme vide.*

```
>>> P = Pile(None)
>>> P.estVide()
True
```

*Implémenter **P** sur votre machine en utilisant les méthodes (constructeurs et modificateurs) de la classe **Pile** afin de construire cette pile au fur et à mesure.*

```
>>> P.empiler('vacances')
>>> P.empiler(5)
>>> P.empiler('nsi')
>>> P.empiler(3)
>>> P.empiler('bob')
>>> P.affiche()
None
vacances
5
nsi
3
bob
```



*Mais que constatez-vous ?*

On peut voir que  
la pile est affichée  
à l'envers

"bob"
3
"nsi"
5
"vacances"
vide





Améliorer la méthode `affiche` pour quelle affiche la pile dans le bon sens.

```
def affiche(self):  
    if self.r==None:  
        print(self.h)  
    else:  
        Pile.affiche(self.r)  
        print(self.h)
```

```
>>> P.affiche()  
None  
vacances  
5  
nsi  
3  
bob
```



```
def affiche(self):  
    if self.r==None:  
        print(self.h)  
    else:  
        print(self.h)  
        Pile.affiche(self.r)
```

```
>>> P.affiche()  
bob  
3  
nsi  
5  
vacances  
None
```

"bob"
3
"nsi"
5
"vacances"
vide



Écrire une fonction `hauteur(P)` qui renvoie la hauteur de la pile.

```
def hauteur(self):  
    hauteur = 0  
    while self.h != None:  
        Pile.depiler(P)  
        hauteur += 1  
    return hauteur
```

```
P.empiler('vacances')  
P.empiler(5)  
P.empiler('nsi')  
P.empiler(3)  
P.empiler('bob')  
P.affiche()  
print(P.hauteur())
```

"bob"
3
"nsi"
5
"vacances"
vide

```
3  
>>>
```



Ecrire une fonction `insere(P,elt)` qui insère dans `P` l'élément `elt` à la position 2. On considère la position 1 correspond au haut de la pile.

```
def insere(self,P,elt):  
    z = self.h  
    Pile.depiler(P)  
    Pile.empiler(P,elt)  
    Pile.empiler(P,z)
```

```
elt = 'Banane'  
P.insere(P,elt)  
P.affiche()
```

```
bob  
Banane  
nsi  
5  
vacances  
None
```



```

def affiche(self):
    if self.r==None:
        print(self.h)
    else:
        print(self.h)
        Pile.affiche(self.r)
def hauteur(self):
    hauteur = 0
    while self.h != None:
        Pile.depiler(P)
        hauteur += 1
    return hauteur
def depil2(self):
    x = self.h
    Pile.depiler(P)
    y = self.h
    Pile.depiler(P)
    Pile.empiler(P,x)
    return y
def insere(self,P,elt):
    z = self.h
    Pile.depiler(P)
    Pile.empiler(P,elt)
    Pile.empiler(P,z)
def acces(P,pos):
    for i in range(pos):
        acces = P.h
        Pile.depiler(P)
    return acces

```

P = Pile()

```

P.empiler('vacances')
P.empiler(5)
P.empiler('nsi')
P.empiler(3)
P.empiler('bob')
P.affiche()

```

```

### 1
print(P.hauteur)

```

```

### 2
print(P.depil2())

```

```

### 3
elt = 'Banane'
P.insere(P,elt)
P.affiche()

```

```

### 4
print(P.acces(2))

```

Ecrire une fonction `acces(P,pos)` qui renvoie l'élément situé en position `pos`.

```

def acces(P,pos):
    for i in range(pos):
        acces = P.h
        Pile.depiler(P)
    return acces

```

```
print(P.acces(2))
```

acces

P	•
pos	2
i	1
acces	3
Return value	3

3







# 2

## L'objet File en Python



Voici une définition de l'objet **File** en PYTHON (implémentée avec une liste doublement chaînées), on commence par définir un élément d'une **liste doublement chaînées** :

```
class Element:
    #chaque élément a pour attribut : le précédent , le suivant et la valeur de l'élément

    def __init__(self,x):
        self.val=x
        self.precedent=None
        self.suivant=None

    def __str__(self): #methode qui permet de lancer un print sur un tel objet
        return str(self.val)+"-"+str(self.suivant)
```

F = (5, 11115, ("vacances", "Bob", 7, "nsi", 12))

On définit ensuite l'objet **File** :

```
class File:
    #ici une file est la donnée de deux attributs : la file complète de type Element et le
    #dernier élément de la file de type Element

    def __init__(self):
        self.tete=None
        self.queue=None

    def file_vide(self):
        return self.tete is None #renvoie True si None et False sinon

    def enfiler(self,x):
        e=Element(x) #on transforme l'élément à ajouter en un objet Element de listes
        #doublement chaînées

        if self.tete==None:
            self.tete=e #file vide la tête est remplacée par l'élément e
        else:
            e.precedent=self.queue #le précédent de l'élément pointe sur l'ancienne queue de
            #la file
            self.queue.suivant=e #l'ancienne queue de la file pointe sur e avec suivant
            self.queue=e #on redéfinit self.queue par e.

    def defiler(self):
        if not self.file_vide():
            e=self.tete #on stocke l'élément à defiler
            if e.suivant is None: #cas où il n'y a qu'un élément
                self.tete=None
                self.queue=None
            else:
                self.tete=e.suivant
                self.tete.precedent=None
            return e.val

    def __str__(self):
        return str(self.tete)
```

Identifier, dans l'exemple, la tête, la queue et le cœur de F :

F = (5, 11115, ("vacances", "Bob", 7, "nsi", 12))

tête

queue

cœur

1.

```
def file_vide(self):  
    return self.tete is None #renvoie True si None et False sinon
```

```
ff = File()
```

```
>>> ff.file_vide()  
True
```

2.

```
*** Console de processus distant Réinitialisée ***  
>>> ff.enfiler(5)  
>>> ff.enfiler('vacances')  
>>> ff.enfiler("Bob")  
>>> ff.enfiler(7)  
>>> ff.enfiler("nsi")  
>>> ff.enfiler(12)  
>>> ff.enfiler(11115)  
>>> print(ff)  
5-vacances-Bob-7-nsi-12-11115-None
```

3.

```
def longueur(L):  
    longueur = 0  
    while L.queue != None:  
        L.defiler()  
        longueur += 1  
    return longueur
```

```
print(ff.longueur())  
print(ff.file_vide())
```

```
7  
True
```



Créer une file vide et tester qu'elle est bien considérée comme vide.



Implémenter F sur votre machine en utilisant les méthodes (constructeurs et modification) de la classe File afin de construire cette file au fur et à mesure.

Ecrire une fonction longueur(F) qui renvoie la longueur de la file

```
def enfiler(self,x):  
    e=Element(x) #on transforme l'élément à ajouter en un objet Element de listes doublement chaînées  
    if self.tete==None:  
        self.tete=e #file vide la tête est remplacée par l'élément e  
    else:  
        e.precedent=self.queue #le précédent de l'élément pointe sur l'ancienne queue de la file  
        self.queue.suivant=e #l'ancienne queue de la file pointe sur e avec suivant  
        self.queue=e #on redéfinit self.queue par e.
```

Ecrire une fonction `defil2(F)` qui enlève de la file `F` l'élément situé en deuxième position de la file et renvoie cet élément.

```
def defil2(F):  
    x = F.tete.suivant  
    F.tete.suivant = F.tete.suivant.suivant  
    return x.val
```

```
print(ff.defil2())  
print(ff)
```

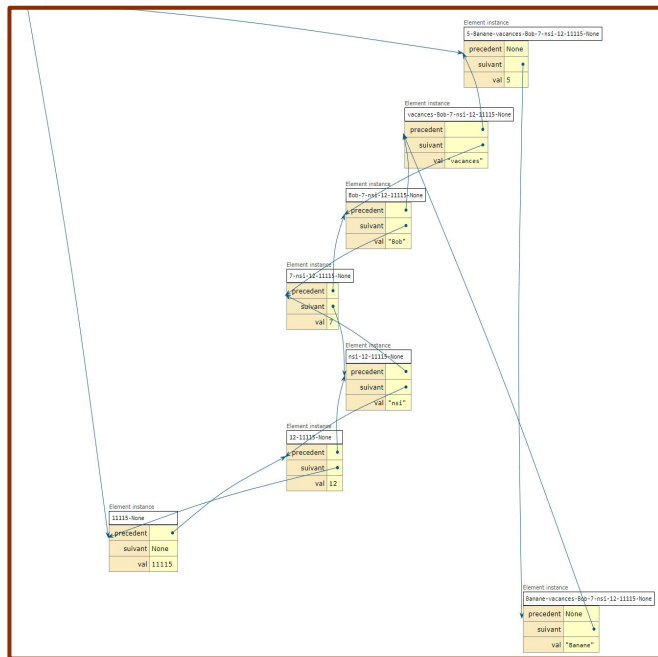
vacances  
5-Bob-7-nsi-12-11115-None

Ecrire une fonction `insere(F,elt)` qui insère dans `F` l'élément `elt` à la position 2. On considère la position 1 correspond au haut de la file.

```
def insere(F,elt):  
    y = F.tete.suivant  
    elt1 = Element(elt)  
    F.tete.suivant = elt1  
    F.tete.suivant.suivant = y  
    return F.tete
```

```
print(ff)  
print(ff.insere("Banane"))
```

5-Banane-vacances-Bob-7-nsi-12-11115-None



```

def longueur(L):
    longueur = 0
    while L.queue != None:
        L.defiler()
        longueur += 1
    return longueur

def defil2(F):
    x = F.tete.suivant
    F.tete.suivant = F.tete.suivant.suivant
    return x.val

def insere(F,elt):
    y = F.tete.suivant
    elt1 = Element(elt)
    F.tete.suivant = elt1
    F.tete.suivant.suivant = y
    return F.tete

def acces(F,pos):
    for i in range(pos):
        Element = F.tete.val
        F.defiler()
    return Element

```

```

ff = File()
ff.enfiler(5)
ff.enfiler('vacances')
ff.enfiler('Bob')
ff.enfiler(7)
ff.enfiler("nsi")
ff.enfiler(12)
ff.enfiler(11115)

print(ff)

### 2
print(ff.longueur())
print(ff.file_vide())

### 3
print(ff.defil2())
print(ff)

### 4
print(ff.insere("Banane"))

### 5
print(ff.acces(2))

```

Ecrire une fonction `acces(P,pos)` qui renvoie l'élément situé en position `pos`.

```

def acces(F,pos):
    for i in range(pos):
        Element = F.tete.val
        F.defiler()
    return Element

```

```

### 5
print(ff.acces(2))

```

vacances



**3**

# **Les parcours d'arbres**



## Implémenter les différents exemples d'arbres vus dans le "M5-TD6 : Parcourir un arbre" et tester sur eux les différentes méthodes ci-dessous :

```
class Arbre:
    def __init__(self, valeur):
        self.v=valeur
        self.fg=None
        self.fd=None
    def ajout_gauche(self, val):
        self.fg=Arbre(val)
    def ajout_droit(self, val):
        self.fd=Arbre(val)

    def affiche(self):
        """permet d'afficher un arbre"""
        if self==None:
            return None
        else :
            return [self.v, Arbre.affiche(self.fg), Arbre.affiche(self.fd)]

    def taille(self):
        if self==None:
            return 0
        else :
            return 1+Arbre.taille(self.fg)+Arbre.taille(self.fd)

    def hauteur(self):
        if self==None:
            return 0
        elif self.fg==None and self.fd==None:
            return 0
        else :
            return 1+max(Arbre.hauteur(self.fg), Arbre.hauteur(self.fd))

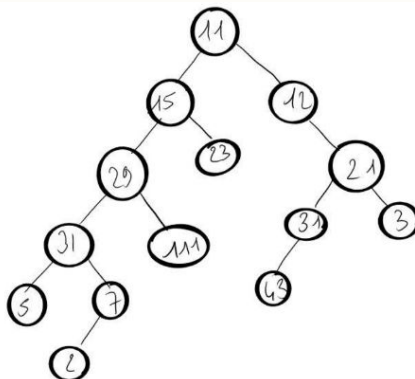
    def get_valeur(self):
        if self==None:
            return None
        else:
            return print(self.v)
```

```
# H1
ArbreA = Arbre(11)
# H2
ArbreA.ajout_gauche(15)
ArbreA.ajout_droit(12)
# H3
ArbreA.fg.ajout_gauche(29)
ArbreA.fg.ajout_droit(23)
ArbreA.fd.ajout_droit(21)
# H4
ArbreA.fg.fg.ajout_gauche(31)
ArbreA.fg.fg.ajout_droit(111)
ArbreA.fd.fd.ajout_gauche(31)
ArbreA.fd.fd.ajout_droit(3)
# H5
ArbreA.fg.fg.fg.ajout_gauche(5)
ArbreA.fg.fg.fg.ajout_droit(7)
ArbreA.fd.fd.fg.ajout_gauche(43)
# H6
ArbreA.fg.fg.fg.fd.ajout_gauche(2)
```

```
print(ArbreA.hauteur())
print(ArbreA.taille())
print(ArbreA.get_valeur())
```

Print output (

```
5
14
11
None
```





## Implémenter les différents exemples d'arbres vus dans le "M5-TD6 : Parcourir un arbre" et tester sur eux les différentes méthodes ci-dessous :

```
class Arbre:
    def __init__(self, valeur):
        self.v=valeur
        self.fg=None
        self.fd=None
    def ajout_gauche(self, val):
        self.fg=Arbre(val)
    def ajout_droit(self, val):
        self.fd=Arbre(val)

    def affiche(self):
        """permet d'afficher un arbre"""
        if self==None:
            return None
        else :
            return [self.v, Arbre.affiche(self.fg), Arbre.affiche(self.fd)]

    def taille(self):
        if self==None:
            return 0
        else :
            return 1+Arbre.taille(self.fg)+Arbre.taille(self.fd)

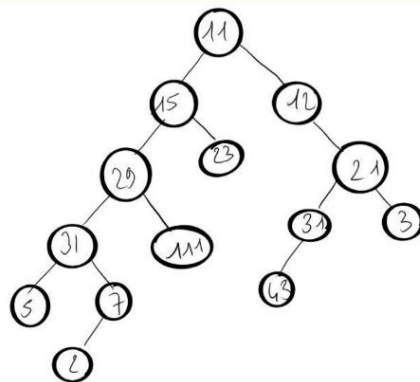
    def hauteur(self):
        if self==None:
            return 0
        elif self.fg==None and self.fd==None:
            return 0
        else :
            return 1+max(Arbre.hauteur(self.fg), Arbre.hauteur(self.fd))

    def get_valeur(self):
        if self==None:
            return None
        else:
            return print(self.v)
```

```
# H1
A2 = Arbre('A')
# H2
A2.ajout_gauche('B')
A2.ajout_droit('F')
# H3
A2.fg.ajout_gauche('C')
A2.fg.ajout_droit('D')
A2.fg.ajout_gauche('G')
A2.fg.ajout_droit('H')
# H4
A2.fg.fg.ajout_droit('E')
A2.fg.fg.ajout_gauche('I')
A2.fg.fg.ajout_droit('J')
```

```
print(A2.hauteur())
print(A2.taille())
print(A2.get_valeur())
```

```
3
10
A
None
```





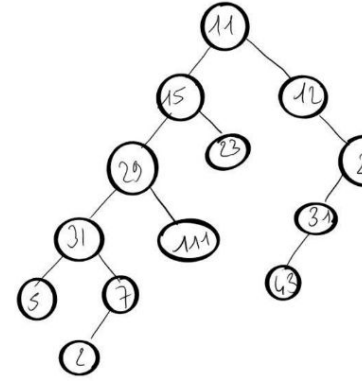
*Pour chacun, réaliser les 3 parcours d'arbres en profondeur (préfixe, infixe et suffixe) grâce à la programmation Orientée Objets (vue avec les files).*

## Parcours Préfixe

```
def affiche_prefixe(self):  
    if self==None:  
        return None  
    else :  
        return [self.v,Arbre.affiche_prefixe(self.fg),Arbre.affiche_prefixe(self.fd)]  
  
print(ArbreA.affiche_prefixe())
```

```
[11, [15, [29, [31, [5, None, None], [7, [2, None, None], None]], [111, None, None]], [23, None, None]], [12, None, [21, [31, [43, None, None], None], [3, None, None]]]]
```

11, 15, 29, 31, 5, 7, 2, 111, 23, 12, 21, 31, 43, 3



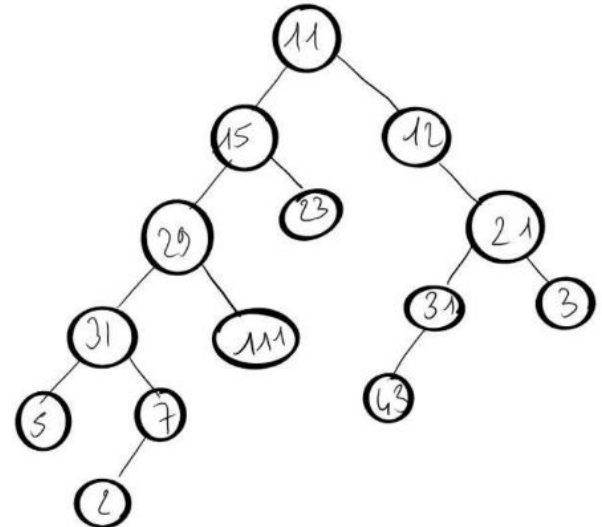
## Parcours Infixe



```
def affiche_infixe(self):  
    if self==None:  
        return None  
    else :  
        return [Arbre.affiche_prefixe(self.fg),self.v,Arbre.affiche_prefixe(self.fd)]  
  
print(ArbreA.affiche_infixe())
```

```
[[[[[None, 5, None], 31, [[None, 2, None], 7, None]], 29, [None, 111, None]], 15, [None, 23, None]], 11, [None, 12, [[[None, 43, None], 31, None], 21, [None, 3, None]]]]]
```

5, 31, 2, 7, 29, 111, 15, 23, 11, 12, 43, 31, 21, 3



## Parcours Suffixe



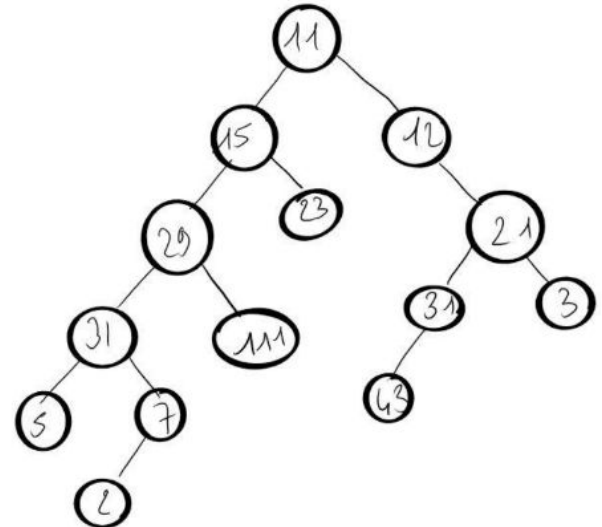
```
def affiche_suffixe(self):  
    if self==None:  
        return None  
    else :  
        return [Arbre.affiche_prefixe(self.fg),Arbre.affiche_prefixe(self.fd), self.v]
```

```
print(ArbreA.affiche_suffixe())
```

```
[[[[[None, None, 5], [[None, None, 2], None, 7], 31], [None, None, 111], 29], [None, None, 23], 15], [None, [[[None, None, 43], None, 31], [None, None, 3], 21], 12], 11]
```

5, 2, 7, 31, 111, 29, 23, 15, 43, 31, 3, 21, 12, 11

Le programme note chaque sommet la dernière fois qu'il le rencontre



Pour chacun, réaliser les 3 parcours d'arbres en profondeur (préfixe, infixe et suffixe) grâce à la programmation Orientée Objets (vue avec les files).

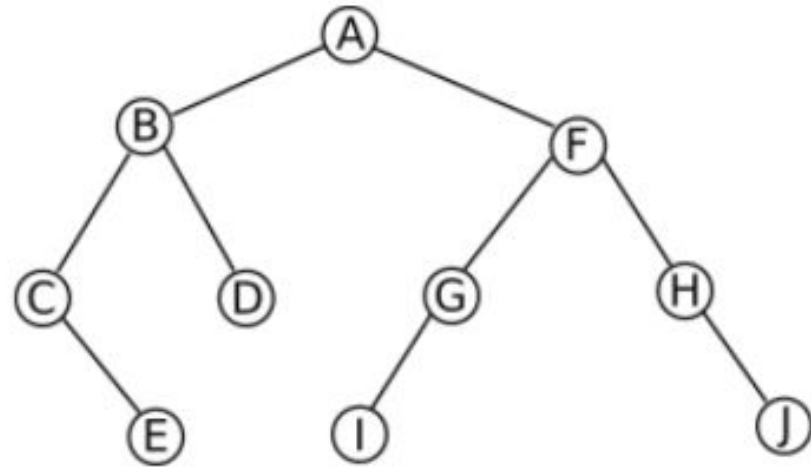


## Parcours Préfixe

```
def affiche_prefixe(self):  
    if self==None:  
        return None  
    else :  
        return [self.v,Arbre.affiche_prefixe(self.fg),Arbre.affiche_prefixe(self.fd)]
```

```
['A', ['B', ['C', None, ['E', None, None]], ['D', None, None]], ['F', ['G', ['I', None, None], None], ['H', None, ['J', None, None]]]
```

A, B, C, E, D, F, G, I, H, J

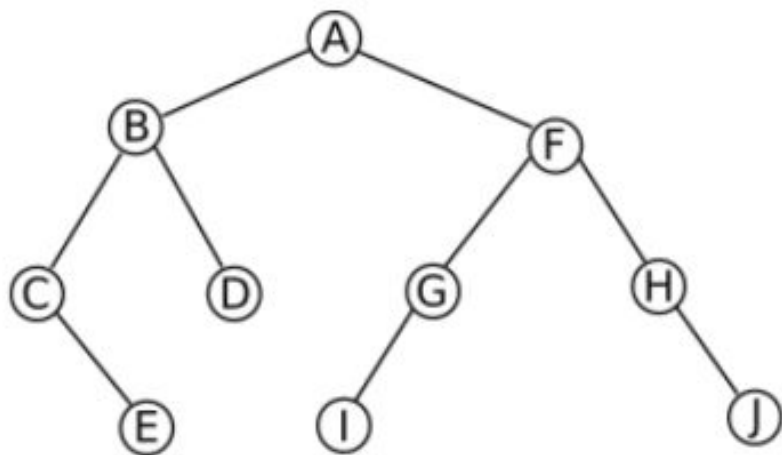




```
def affiche_infixe(self):  
    if self==None:  
        return None  
    else :  
        return [Arbre.affiche_prefixe(self.fg),self.v,Arbre.affiche_prefixe(self.fd)]
```

```
|[[[None, 'C', [None, 'E', None]], 'B', [None, 'D', None]], 'A', [[[None, 'I', None], 'G', None], 'F', [None, 'H', [None, 'J', None]]]]
```

C, E, B, D, A, I, G, F, H, J



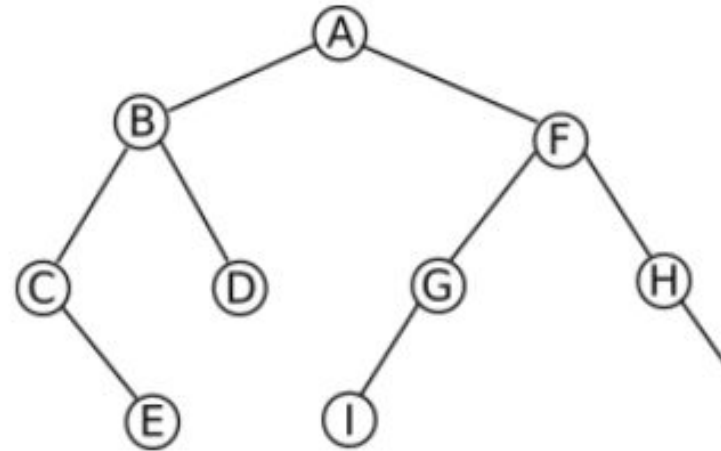
## Parcours Suffixe

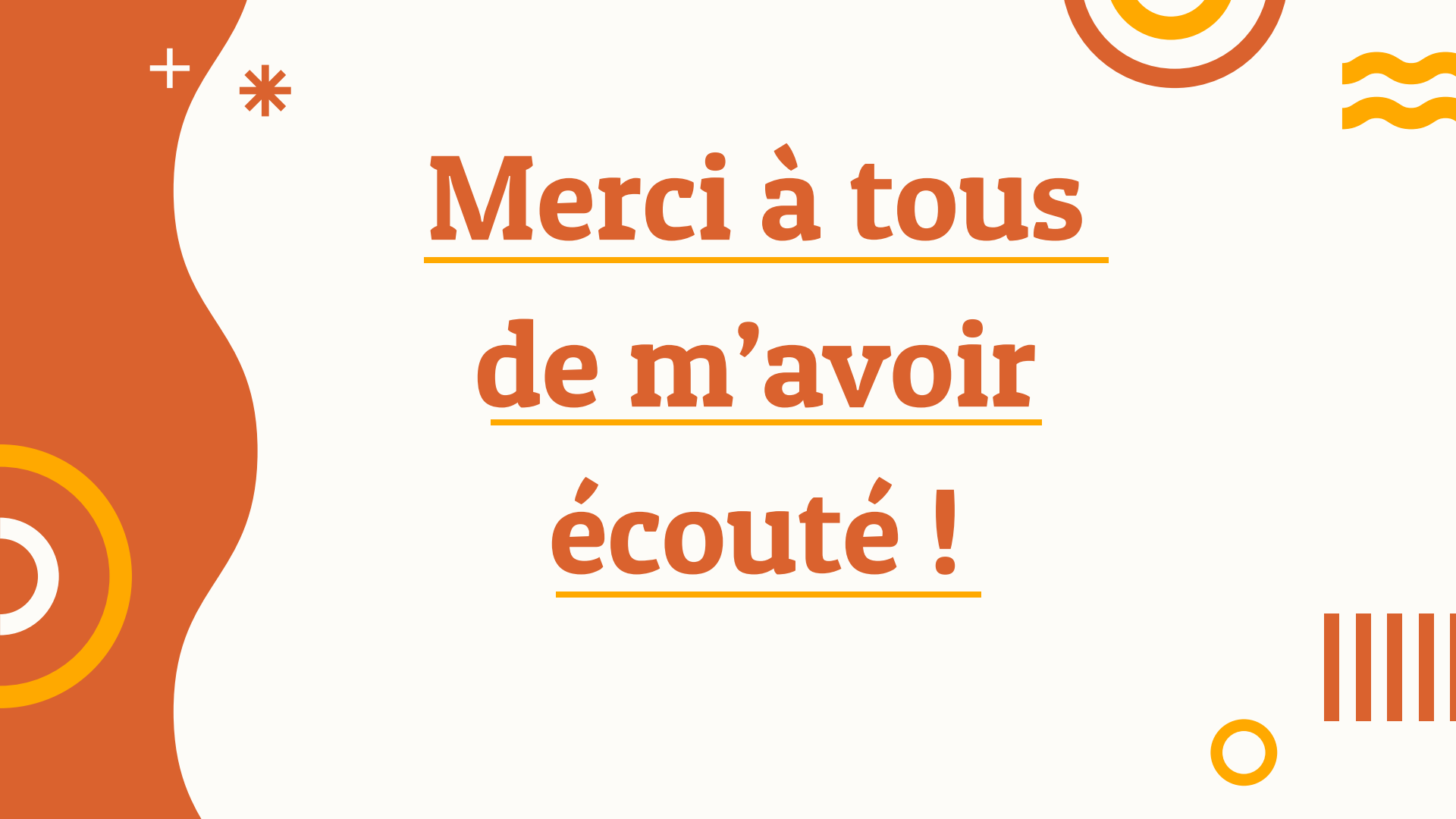
```
def affiche_suffixe(self):  
    if self==None:  
        return None  
    else :  
        return [Arbre.affiche_prefixe(self.fg),Arbre.affiche_prefixe(self.fd), self.v]
```

```
[[[None, [None, None, 'E'], 'C'], [None, None, 'D'], 'B'], [[[None, None, 'I'], None, 'G'], [None, [None, None, 'J'], 'H'], 'F'], 'A']
```

E, C, D, B, I, G, J, H, F, A

Le programme note chaque sommet la dernière fois qu'il le rencontre





**Merci à tous**  
**de m'avoir**  
**écouté !**