Création d'un arbre (avec la classe nœud)

Objectif : savoir créer sa propre classe pour manipuler un objet type Nœud (d'un arbre).

```
#
# Définition de notre classe Arbre
#

class noeud:
    def __init__(self,numero,fils_gauche,fils_droit):
        self.etiquette = numero
        self.pointeur_gauche = fils_gauche
        self.pointeur_droit = fils_droit

# Creation noeud fils gauche
noeud2 = noeud(2,None,None)
print ("noeud2 =",noeud2)
print (noeud2.etiquette)

# Creation noeud fils droit
noeud3 = noeud(3,None,None)
print ("noeud3 =",noeud3)
print (noeud3.etiquette)

#creation noeud père qui pointe vers
#ses fils gauche et droit
noeud1 = noeud(1,noeud2,noeud3)
print ("noeud1 =",noeud1)
print (noeud1.etiquette)

print ("fils gauche de noeud 1:",  noeud1.pointeur_gauche)
print ("fils droit de noeud1 :",  noeud1.pointeur_droit )

```

Le nœud doit être initialisé avec une étiquette (son numéro de nœud), un fils gauche et un fils droit.

Je crée une instance nommée noeud2, numéro =2 et pas de fils gauche ni droit (on met **None** pour indiquer que ca pointe sur rien)

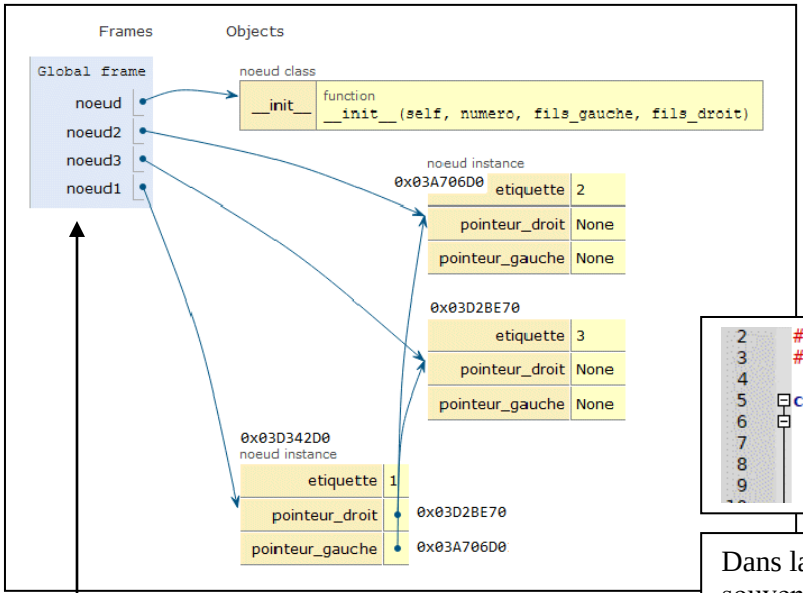
Je crée une instance nommée noeud3 toujours sans fils gauche ni droit

Je crée le nœud père avec cette fois comme fils gauche le nœud2 et fils droit le nœud 3

```
noeud2 = <__main__.noeud object at 0x03A706D0>
2
noeud3 = <__main__.noeud object at 0x03D2BE70>
3
noeud1 = <__main__.noeud object at 0x03D342D0>
1
fils gauche de noeud 1: <__main__.noeud object at 0x03A706D0>
fils droit de noeud1 : <__main__.noeud object at 0x03D2BE70>
```

Noeud2, Noeud3 et Nœud sont des adresses réservées en mémoire

Le pointeur_gauche de noeud1 contient **l'adresse** de noeud2.
le pointeur_droit de noeud1 contient **l'adresse** de noeud3



Au final, en créant des instances de ma classe, je réserve des zones en mémoire avec la structuration initialisée par la classe.
A l'adresse de (l'instance) nœud 2, on a une case mémoire pour l'étiquette, et deux cases mémoires pour des pointeurs.

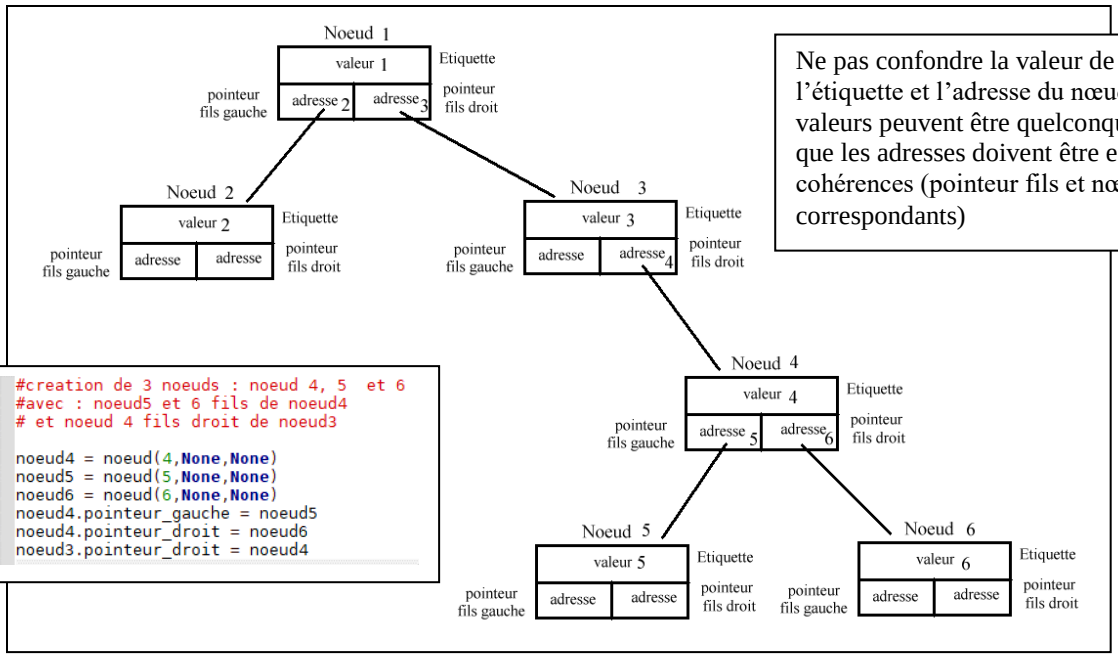
```
# Ce qu'on voit souvent
# dans la littérature

class noeud:
    def __init__(self,etiquette,gauche,droit):
        self.etiquette = etiquette
        self.gauche = gauche
        self.droit = droit
```

Dans la littérature sur les classes vous verrez souvent l'écriture ci-dessus qui est courante mais attention aux confusions : Il faut bien distinguer ce qui est passé en paramètre et les attributs de l'objet.
« self.etiquette » correspond à une case mémoire qui va être initialisé avec « etiquette » qui est une donnée passée en paramètre.

nœud, noeud1, noeud2, noeud3 correspondent aussi à une adresse en mémoire (on y a pas accès) . Lorsqu'on fait print (noeud1), on affiche le contenu de noeud1 c'est-à-dire ici 0x03D342D0. Au final on assimile le pointeur à l'objet (zone mémoire) pointé…

Compléter avec le code indiqué ci-dessous pour obtenir la structure ci-dessous :

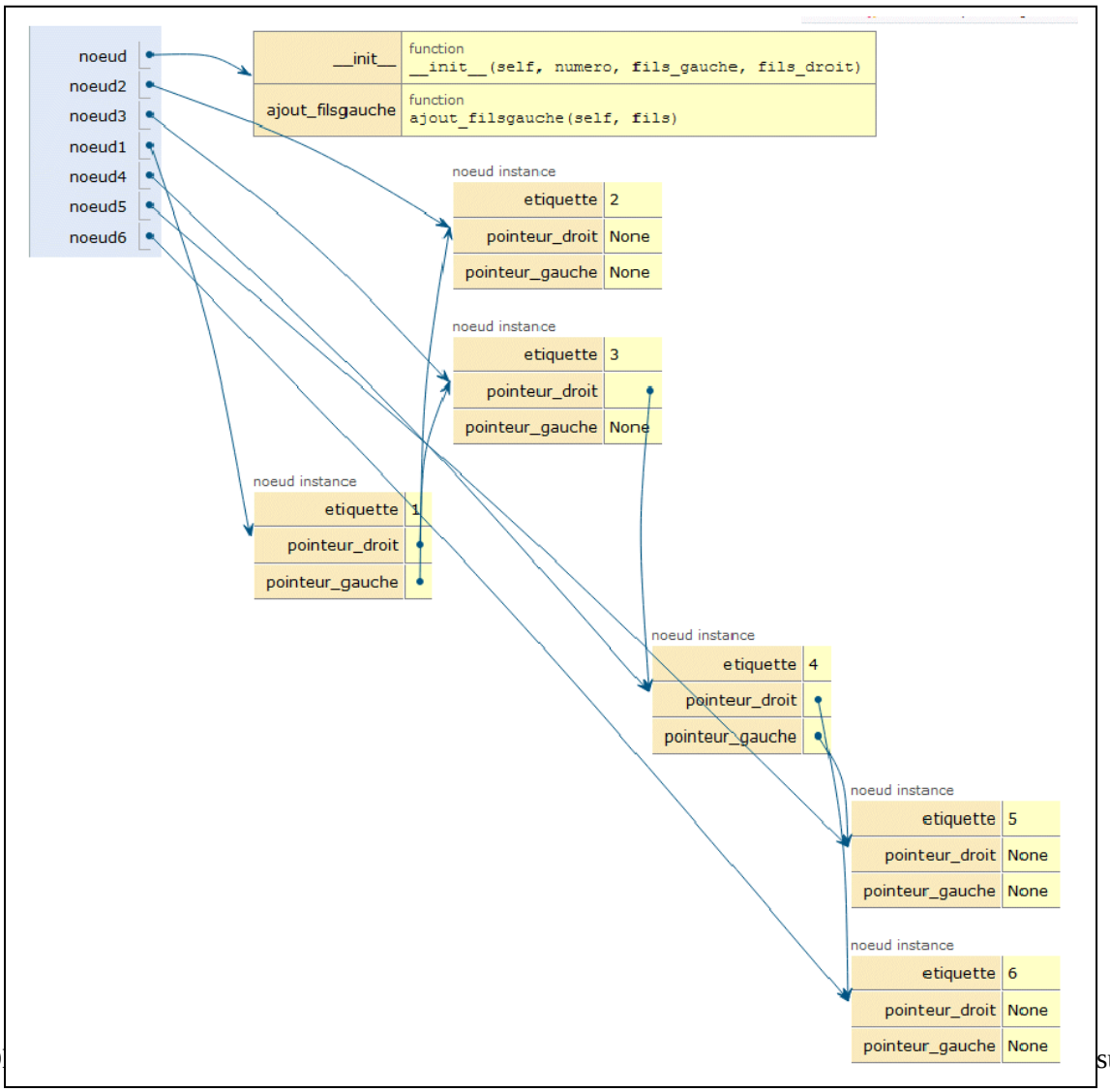


Ne pas confondre la valeur de l'étiquette et l'adresse du nœud. Les valeurs peuvent être quelconques alors que les adresses doivent être en cohérences (pointeur fils et nœuds correspondants)

```
#creation de 3 noeuds : noeud 4, 5  et 6
#avec : noeud5 et 6 fils de noeud4
# et noeud 4 fils droit de noeud3

noeud4 = noeud(4,None,None)
noeud5 = noeud(5,None,None)
noeud6 = noeud(6,None,None)
noeud4.pointeur_gauche = noeud5
noeud4.pointeur_droit = noeud6
noeud3.pointeur_droit = noeud4
```

Vérifier avec python tutor que vous obtenez bien le schéma ci-dessous :



```python
class noeud:
    def __init__(self,numero,fils_gauche,fils_droit):
        self.etiquette = numero
        self.pointeur_gauche = fils_gauche
        self.pointeur_droit = fils_droit

    def ajout_filsgauche(self,fils):
        self.pointeur_gauche = fils

    def ajout_filsdroit(self,fils):
        self.pointeur_droit = fils
```

Accéder aux attributs directement n'est pas recommandé, il est préférable d'utiliser des méthodes pour modifier les attributs.
Compléter votre classe « nœud » avec les méthodes indiquées ci-contre (ajout_filsgauche et ajout_filsdroit)
Ne pas oublier les « self »

```python
#creation de 3 noeuds : noeud 4, 5  et 6
#avec : noeud5 et 6 fils de noeud4
# et noeud 4 fils droit de noeud3

noeud4 = noeud(4,None,None)
noeud5 = noeud(5,None,None)
noeud6 = noeud(6,None,None)
#noeud4.pointeur_gauche = noeud5
#noeud4.pointeur_droit = noeud6
#noeud3.pointeur_droit = noeud4

noeud4.ajout_filsgauche(noeud5)
noeud4.ajout_filsdroit(noeud6)
noeud3.ajout_filsdroit(noeud4)
```

Dans le programme principal, désactiver les lignes de codes comme indiqué

et rajouter les appels aux méthodes.
Tester et vérifier que vous obtenez les mêmes résultats.
Validation prof :

*** Compléter en choisissant les termes : « instance, attribut, méthode, paramètre »)**

Etiquette , pointeur_gauche, pointeur_droit sont appelés des …… de l'objet créé.

ajout_filsgauche et ajout_filsdroit sont des …… de la classe noeud

noeud1, noeud2, noeud3 sont des …… de la classe noeud

numero, fils_gauche, fils_droit sont des ……. de la fonction __init__

« Self » dans la classe nœud représente l'…… auquel on fait appel

*** L'instruction** « noeud4.ajout_filsgauche(noeud5) » permet de faire appel à la méthode : Ajout_filsgauche(self, fils) . Que valent alors les paramètres « self » et « fils » ?
Rep :
Self = ……. fils= …….

Il est tout à fait possible (et en général c'est ce qu'on fait) de construire un arbre sans pour autant créer de liens vers l'instance (ou frame dans « python tutor).
Rajouter à la suite le code ci-dessous et tester le pour vérifier pas d'erreur.

```python
noeud2.ajout_filsgauche(noeud(7,None,None))
```

Visualisez ensuite le résultat dans python tutor : que constatez vous ? (regardez au niveau des « frames » en haut à gauche) .

Rep :

Faire de même pour ajouter à noeud2 un fils droit (avec numéro 8 pour le fils).

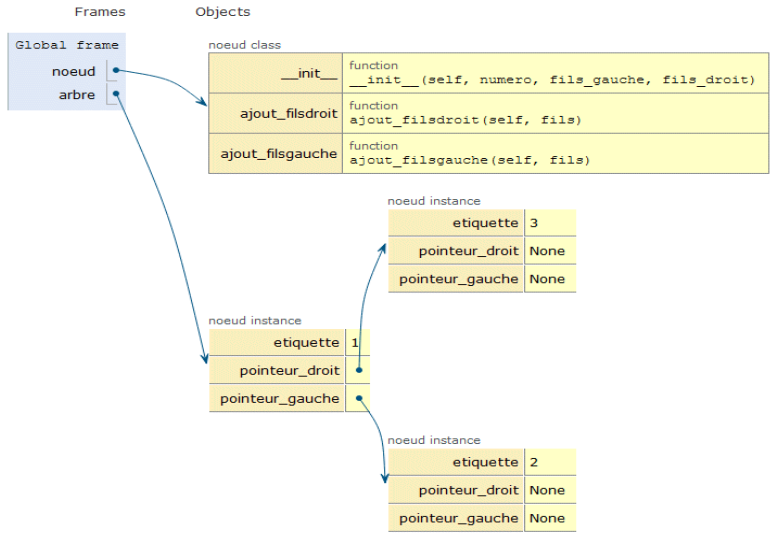
Pour la suite, vous devez créer un nouveau fichier python , par exemple td_arbre2.py :

```
1   #
2   # Construisons notre arbre
3   #
4
5   class noeud:
6       def __init__(self,numero,fils_gauche,fils_droit):
7           self.etiquette = numero
8           self.pointeur_gauche = fils_gauche
9           self.pointeur_droit = fils_droit
10
11      def ajout_filsgauche(self,fils):
12          self.pointeur_gauche = fils
13
14      def ajout_filsdroit(self,fils):
15          self.pointeur_droit = fils
16
17
18  arbre = noeud(1, noeud(2,None,None), noeud(3,None,None) )
19
```

Etiquette 1, fils gauche, fils droit

L'objectif ici est de créer l'arbre sans donner de nom aux instances (des nœuds) qui sont créées :
Pour cela il est possible d'appeler la classe nœud récursivement (voir ci-contre)
Tester le code ci-dessous et visualisez son exécution sous python tutor.

Vérifier que vous obtenez bien ceci :



Sachant qu'il est donc possible de définir des nœuds à l'intérieur d'un nœud, refaire l'arbre de la page 2 **en rajoutant** les nœuds nécessaires dans l'instance arbre (ligne 18 du programme ci-dessus)

Visualisez l'arbre ainsi créé et **faire valider par le professeur**.

Essayez maintenant ceci :

```
arbre = noeud(1, noeud(2,None,None), noeud(3,None,None) )

arbre2 = noeud(0,arbre,noeud(10,None,None ))
```

Il est possible de mettre comme nœud un arbre (qui n'est qu'un pointeur vers un nœud au final)

```
22  def visite(arbre):
23
24      print (arbre.etiquette)
25
26      if arbre.pointeur_gauche !=None:
27          visite(arbre.pointeur_gauche)
28
29      if arbre.pointeur_droit !=None:
30          visite(arbre.pointeur_droit)
31
32  visite(arbre2)
```

Tester la fonction visite (que vous avez vu déjà en cours) .
Afficher le parcours préfixé, infixé et postfixé (suffixe)

Comment faire pour compter le nombre de nœud de notre arbre ? (Expérimentez des solutions)