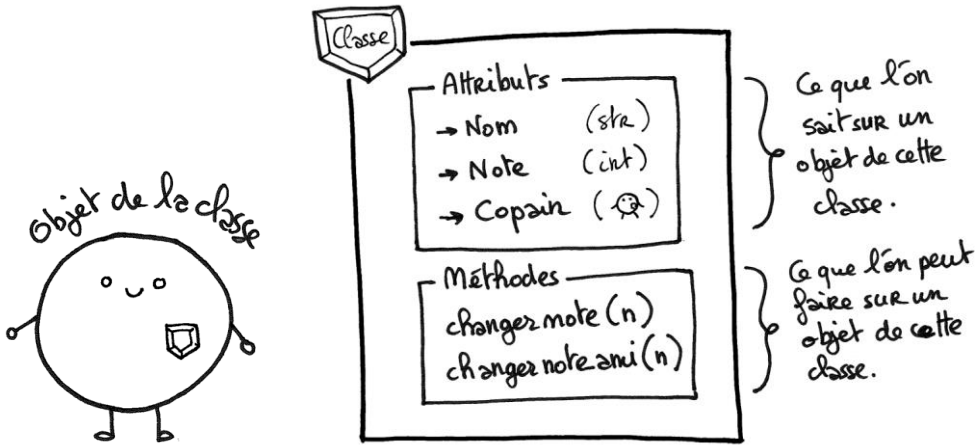
Mieux comprendre la POO

Une **classe** est un ensemble de variables et de fonctions permettant de créer des objets. Une classe peut contenir plusieurs objets.

Un **objet** est un bloc de code mêlant des attributs (des variables) et méthodes (des fonctions pour manipuler les attributs).

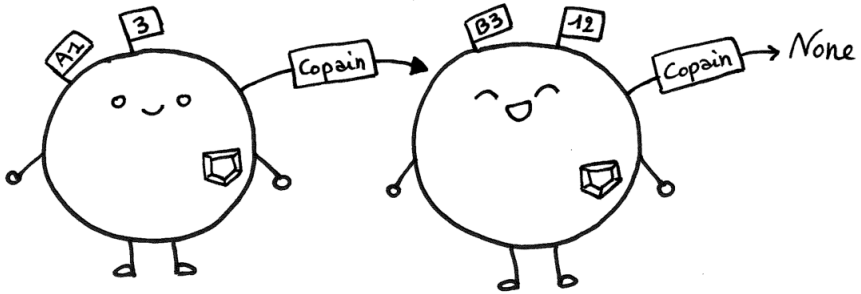


Codons notre classe d'exemple en python :

```
class MaClasse:

      def __init__(self, nom, n, c):
            self.nom = nom       # str
            self.note = n        # str
            self.copain = c      # MaClasse ou None

      def changernote(self, n):
            …

      def changernoteami(self, n) :
            …
```

Maintenant on peut créer deux objets de cette classe dans le programme principal :

```
objetB = MaClasse("B3", 12, None)
objetA = MaClasse("A1", 3, objetB)
```



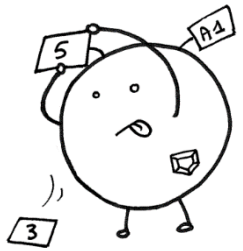
Maintenant, comment modifier la valeur d'un attribut d'un objet ?

Tout dépend où l'on code !

1. Dans la classe

La méthode `changernote` modifie l'attribut `note` par un entier `n`.

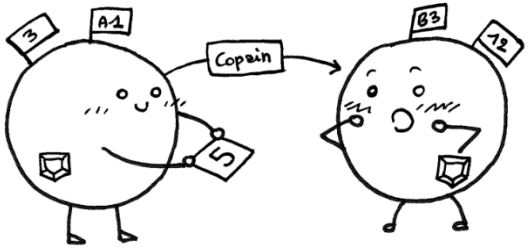
```
def changernote(self, n):
      self.note = n
```



L'objet va changer sa propre note, donc on utilise `self` devant l'attribut `note`.

La méthode `changernoteami` modifie l'attribut `note` d'un autre objet `MaClasse` par un entier `n`.

```
def changernoteami(self, n):
      ami = self.copain
      ami.note = n
```



L'objet va trouver son copain, donc on utilise `self` devant l'attribut `copain`.

Par contre, l'objet va changer la note d'un autre objet, donc on n'utilise pas `self` devant l'attribut `note`.

2. Hors de la classe

L'instruction suivante modifie l'attribut `note` de l'objet `objetA`..

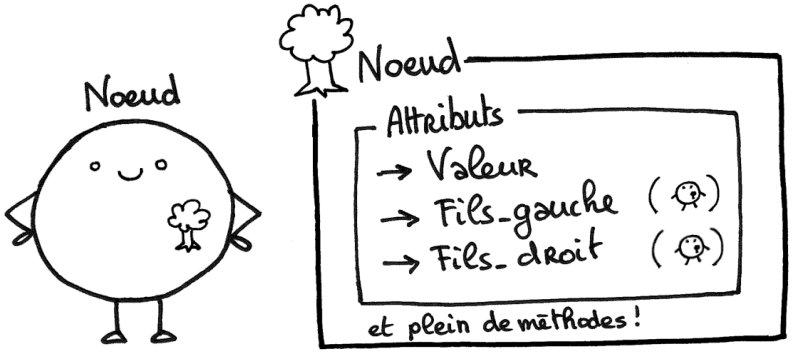
```
objetA.note = 5
```

Puisque l'on se trouve hors de la classe, on n'utilise pas `self` devant l'attribut `note`. On utilise la variable qui référence l'objet (ici `objetA`).

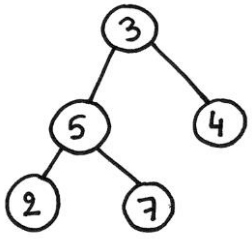
Voyons maintenant un exemple concret : les **arbres** !

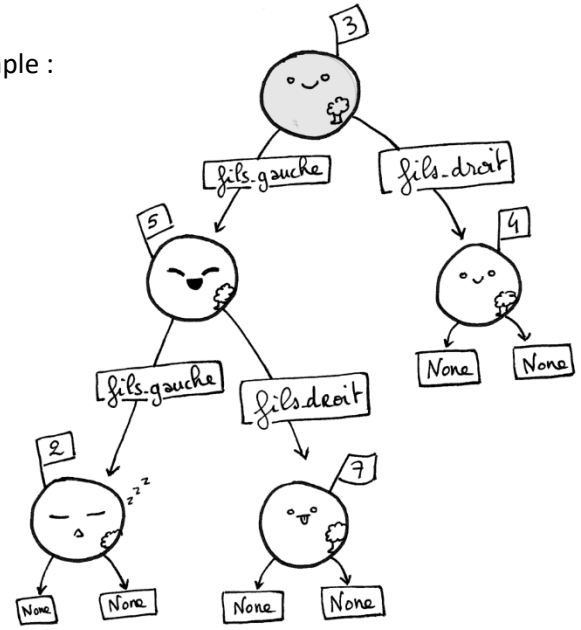
Codons notre classe `Noeud` en python :

```
class Noeud:

      def __init__(self, v, g, d):
            self.valeur = v        # str ou int
            self.fils_gauche = g   # Noeud ou None
            self.fils_droit = d    # Noeud ou None

      # On suppose qu'il y a des méthodes :)
```

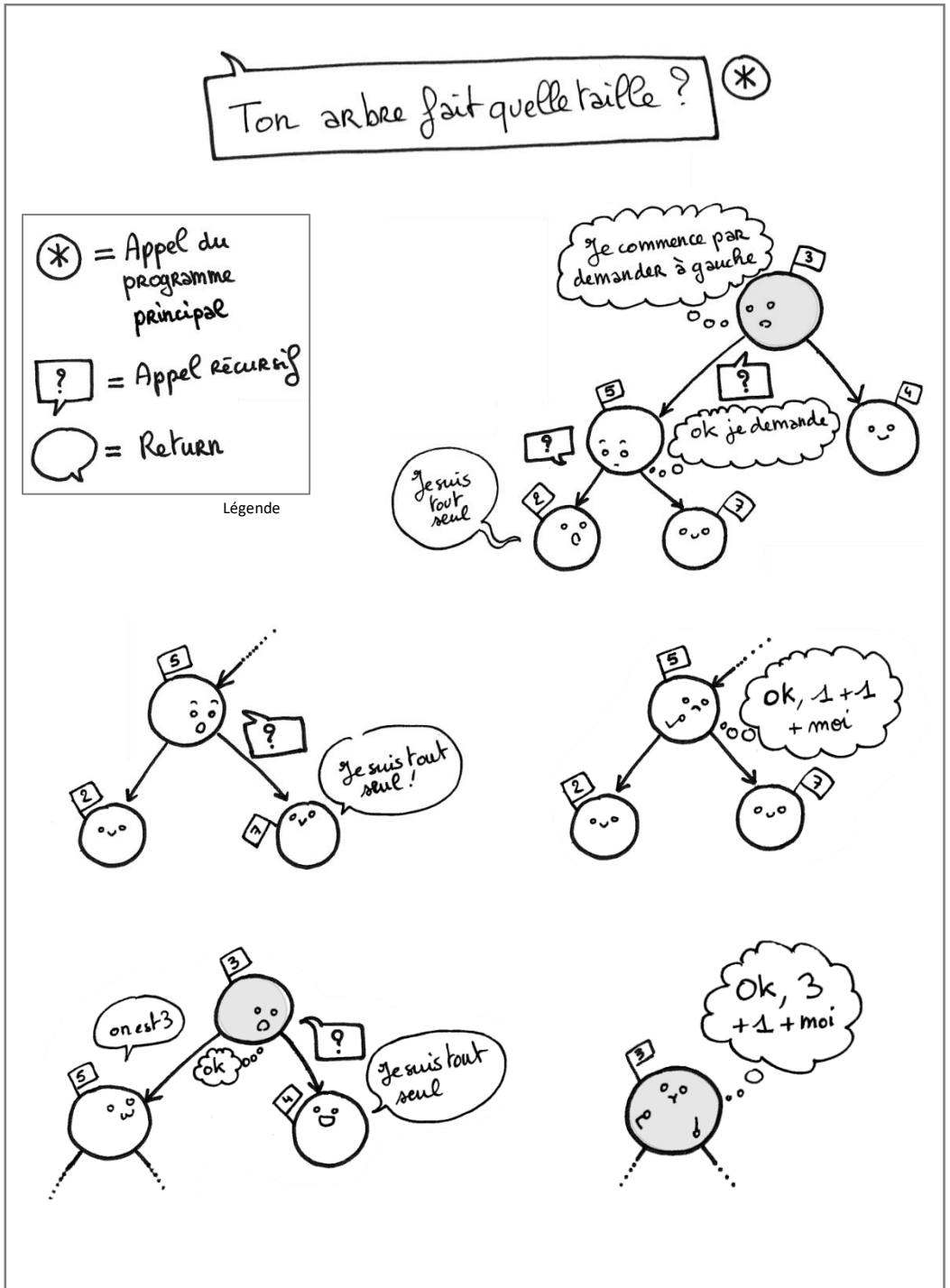


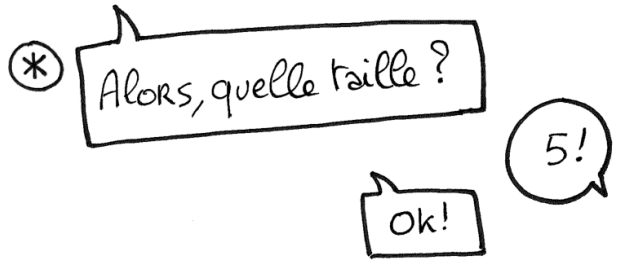
Prenons cet arbre comme exemple :



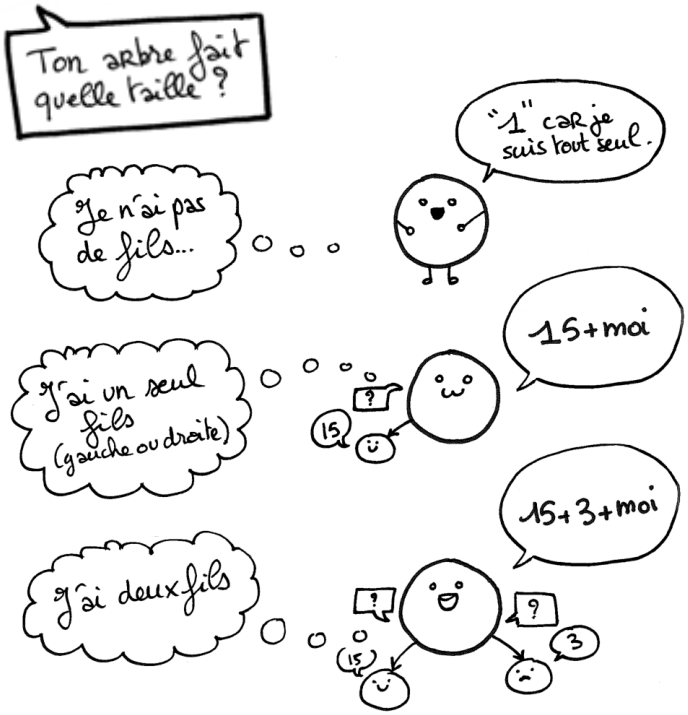


Prenons en exemple la méthode `taille` qui retourne la taille d'un arbre :





Récapitulatif :



Si le nœud n'a pas de fils (condition d'arrêt), il retourne 1 : il se compte lui-même.

Si le nœud a un/des fils, il leur pose la même question (appels récursifs), puis répond (retourne) leurs résultats + 1 (car le nœud se compte lui-même).