Introduction à la Programmation Orientée Objet (POO)

Anne-Julie TINET
Ayoub BEL HACHMI
Alexis DERYCKE

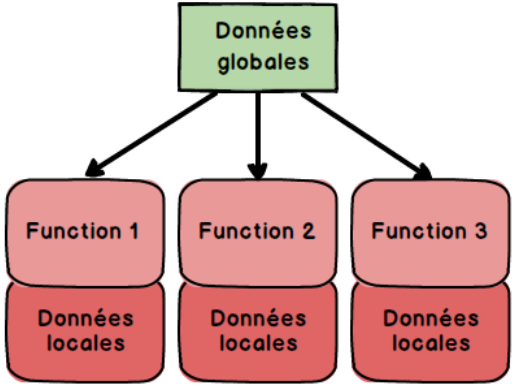
Pourquoi ?

Qu'est ce que la POO ?

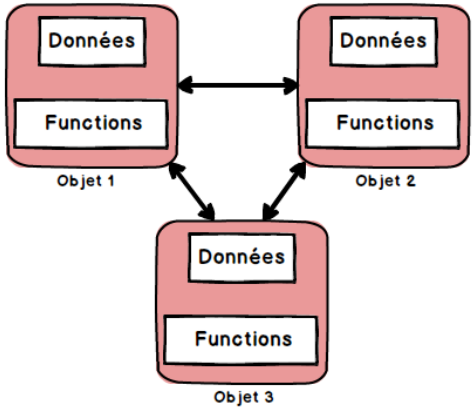
La programmation orientée objet est un **paradigme** de programmation.

Modèle de pensée. Représentation du monde.

Programmation Procédurale : La programmation est divisée en plusieurs procédures, ou l'on découpe le problème en plusieurs fonctions. Chaque fonction réalise une tâche/action précise.



Programmation orientée objet : Représentation des problèmes à l'aide d'objets réels et de leur comportement

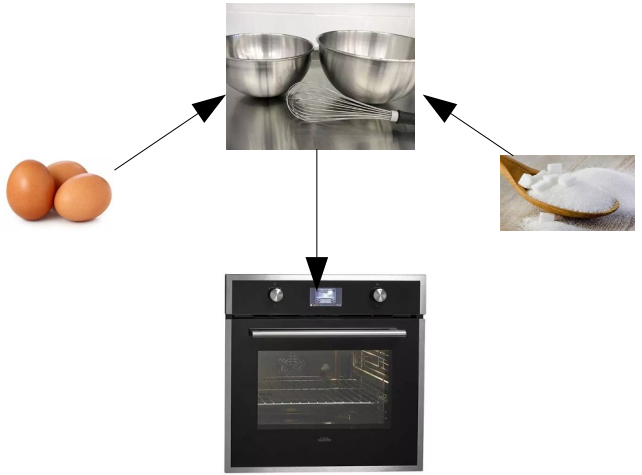


Qu'est-ce que la POO ?

Programmation procédurale

1) Séparer les blancs et les jaunes d'œufs
2) Battre les blancs en neige
3) Mélanger avec du sucre
4) Cuire au four

Programmation orientée objet



Pourquoi la POO ?

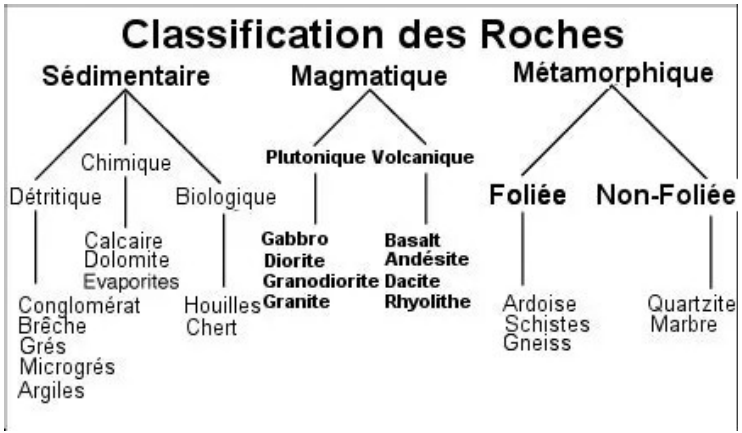
- ✔ Modélisation plus **naturelle** du problème.

Décrire l'image.

Reflet de la réalité et donc simplification de l'algorithme.

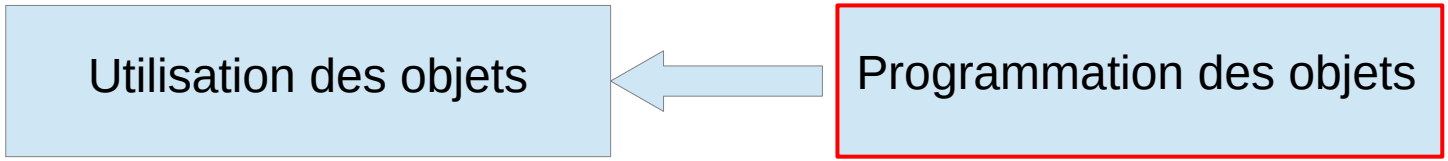
Pourquoi la POO ?

- ✔ Modélisation plus **naturelle** du problème.
- ✔ **Réutilisation** et spécialisation (=> **Héritage**)



Pourquoi la POO ?

- ✔ Modélisation plus **naturelle** du problème.
- ✔ **Réutilisation** et spécialisation (=> **Héritage**)
- ✔ **Simplification** des algorithmes complexes
- ✔ **Sécurité** et robustesse (=> **Encapsulation**)
- ✔ **Evolutivité**
- ✔ **Polymorphisme**



Comment ça marche ?

Les objets

Un objet (une instance) se décrit par :

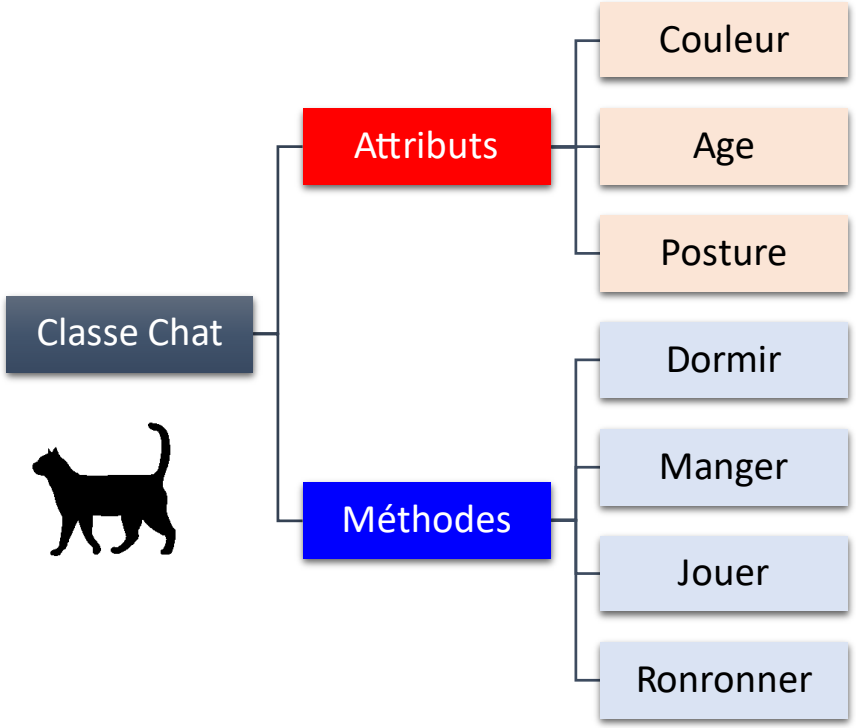
- Ce qu'il est / Son état : Les attributs.
- Ce qu'il fait / Ses actions & interactions : Les **méthodes**.
- Son identité : L'identifiant.

Les classes

Une classe = Un plan ou un moule pour fabriquer des objets

Une instance = Un objet créé à partir d'une classe.

Les classes - Exemple



Instance Duchesse
Couleur : Blanc
Age : Adulte
Posture : Assis

Instance Toulouse
Couleur : Roux
Age : Jeune
Posture : Debout
Joue

Les classes

```
class NomClasse:

    def __init__(self, ...):

        self.attribut_ = ...


    def methode(self, ...):

        ...
```

Définition d'une classe

Méthode constructeur

Bonnes pratiques :

Le nom de la classe commence par une majuscule

Les attributs se terminent par un underscore _.

La méthode **constructeur** est obligatoire ! C'est elle qui permettra de créer des instances de la classe (i.e. des objets). **Tous les attributs doivent être créés / initialisés dans la méthode constructeur.**

Le paramètre **self** représente l'instance courante de la classe (i.e. l'objet sur lequel on travaille). Il est toujours indiqué en 1er paramètre lors de la création d'une méthode et est obligatoire.

Les classes - Exemple

Créer la classe **Elève** et sa méthode constructeur associée.

Un Elève a pour attributs :

- Son nom
- Sa promo
- Une liste de notes

```
class Eleve:

    def __init__(self, nom, promo, lnotes):

        self.nom_ = nom
        self.promo_ = promo
        self.lnotes_ = lnotes
```

Instanciation

L'instanciation correspond à la création d'une instance (i.e. d'un objet) d'une classe.

```
instance = NomClasse(...)
```

Lorsque l'on fait une instanciation, en fait, on appelle la méthode constructeur. Le paramètre `instance` correspond à `self`, les autres paramètres sont dans les parenthèses.

Instanciation - Exemple

Dans le programme principal, créer une instance de la classe Eleve.

```
class Eleve:

    def __init__(self, nom, promo, lnotes):

        self.nom_ = nom
        self.promo_ = promo
        self.lnotes_ = lnotes

# Programme principal
if __name__ == "__main__":

    padawan = Eleve("Luke", "1A", [])
```

Les méthodes

```
class NomClasse:

    def __init__(self, ...):

        self.attribut_ = ...


    def methode(self, ...):

        ...
```

Définition d'une méthode

```
instance.methode(...)
```

Appel d'une méthode pour une instance de la classe.

Le paramètre **self** est toujours indiqué en 1er paramètre lors de la création d'une méthode et est obligatoire. Lors de l'appel d'une méthode, l'**instance** sur laquelle elle est appliquée (qui correspond donc au paramètre **self**) est située devant la méthode. Les autres paramètres sont dans les parenthèses.
Les **méthodes** **utilisent les attributs**, elles peuvent aussi les **modifier**. Les paramètres correspondent aux données **extérieures** aux objets (information non présente dans les attributs).

Les méthodes - Exemple

Créer une méthode qui calcule la moyenne de l'élève.

Tester la méthode dans le programme principal.

```
class Eleve:

    def __init__(self, nom, promo, lnotes):

        self.nom_ = nom
        self.promo_ = promo
        self.lnotes_ = lnotes

    def calculer_moyenne(self):

        if self.lnotes_ != []:
            return sum(self.lnotes_)/len(self.lnotes_)

# Programme principal
if __name__ == "__main__":

    padawan = Eleve("Luke", "1A", [12, 14])
    print(padawan.calculer_moyenne())
```

Les méthodes - Exemple

Créer une méthode qui ajoute une note à l'élève.

```
class Eleve:

    def __init__(self, nom, promo, lnotes):

        self.nom_ = nom
        self.promo_ = promo
        self.lnotes_ = lnotes

    def calculer_moyenne(self):

        if self.lnotes_ != []:
            return sum(self.lnotes_)/len(self.lnotes_)

    def ajouter_note(self, note):

        self.lnotes_.append(note)
```

Accesseurs et mutateurs

```
class NomClasse:

    def __init__(self, ...):

        self.attribut_ = ...

    def getAttribut(self):

        return self.attribut_

    def setAttribut(self, valeur):

        self.attribut_ = valeur
```

Méthode accesseur

Méthode mutateur

On utilise les **méthodes accesseurs** (une méthode par attribut) pour accéder aux attributs hors de la classe (e.g. autre classe, programme principal).

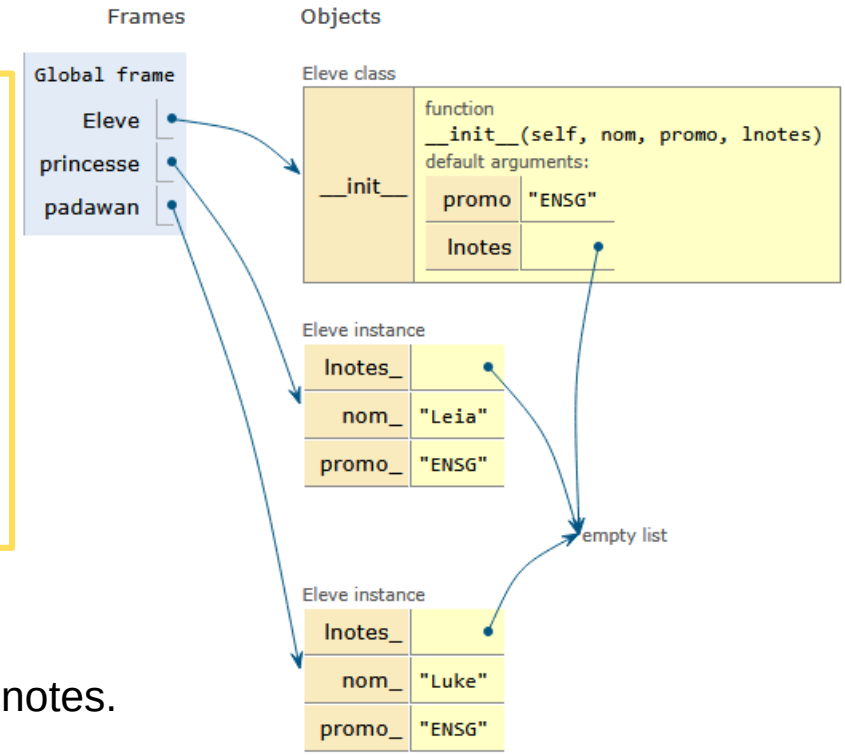
On utilise les **méthodes mutateurs** (une méthode par attribut) pour modifier les attributs hors de la classe (e.g. autre classe, programme principal).

Les pièges à éviter - Exemples

Modifier la méthode constructeur en utilisant les valeurs par défaut ‘ENSG’ pour la promo et une liste vide pour la liste de notes.

Les pièges à éviter - Exemples

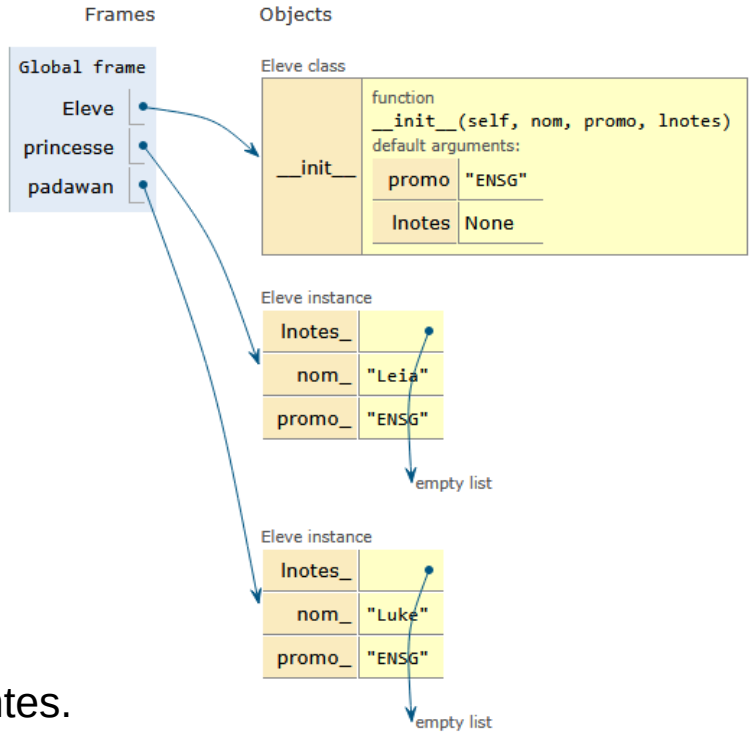
```
class Eleve:

    def __init__(self, nom, promo = 'ENSG', lnotes = []):

        self.nom_ = nom
        self.promo_ = promo
        self.lnotes_ = lnotes

# Programme Principal
princesse = Eleve('Leia')
padawan = Eleve('Luke')
```



Les deux élèves partagent la même liste de notes.

Les pièges à éviter - Exemples

```
class Eleve:

    def __init__(self, nom, promo = 'ENSG', lnotes =
None):

        self.nom_ = nom
        self.promo_ = promo
        if lnotes is None:
            self.lnotes_ = []
        else:
            self.lnotes_ = lnotes

# Programme Principal
princesse = Eleve('Leia')
padawan = Eleve('Luke')
```



Les deux élèves ont des listes de notes différentes.

Il faut faire attention dès que la valeur par défaut est de type **mutable** (*e.g.* liste, dictionnaire, objet issu d'une classe 'fait-maison').

Les pièges à éviter - Exemples

Modifier l'écriture de la classe de façon à ce la méthode de calcul de la moyenne créé un attribut.

Créer la méthode accesseur pour la moyenne.

Dans le programme principal :

- Créer une instance de la classe Eleve, par défaut.
- Afficher sa moyenne (à l'aide de la méthode accesseur).
- Ajouter une note à l'élève.
- Afficher sa moyenne (à l'aide de la méthode accesseur).

Les pièges à éviter - Exemples

```
class Eleve:

    def __init__(self, nom, promo = 'ENSG', lnotes = None):

        self.nom_ = nom
        self.promo_ = promo
        if lnotes is None:
            self.lnotes_ = []
        else:
            self.lnotes_ = lnotes

    def calculer_moyenne(self):

        if self.lnotes_ != []:
            self.moyenne_ = sum(self.lnotes_)/len(self.lnotes_)
        else:
            self.moyenne_ = None

    def ajouter_note(self, note):

        self.lnotes_.append(note)

    def getMoyenne(self):

        return self.moyenne_
```

```
# Programme Principal
padawan = Eleve('Luke')
print(padawan.getMoyenne())

padawan.ajouter_note(14)
print(padawan.getMoyenne())
```

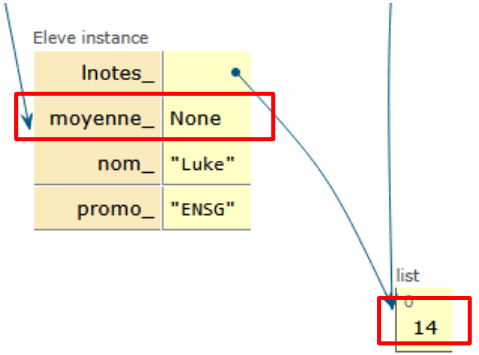
```
AttributeError: 'Eleve' object has no
attribute 'moyenne_'
```

Les pièges à éviter - Exemples

```
class Eleve:

    def __init__(self, nom, promo = 'ENSG', lnotes = None):

        self.nom_ = nom
        self.promo_ = promo
        if lnotes is None:
            self.lnotes_ = []
        else:
            self.lnotes_ = lnotes

        self.calculer_moyenne()

    def calculer_moyenne(self):

        if self.lnotes_ != []:
            self.moyenne_ = sum(self.lnotes_)/len(self.lnotes_)
        else:
            self.moyenne_ = None

    def ajouter_note(self, note):

        self.lnotes_.append(note)

    def getMoyenne(self):

        return self.moyenne_
```

```
# Programme Principal
padawan = Eleve('Luke')
print(padawan.getMoyenne())

padawan.ajouter_note(14)
print(padawan.getMoyenne())
```

Tous les attributs doivent être créés dans la méthode constructeur. Il est possible de le faire en **appelant une autre méthode** de la classe.



Les pièges à éviter - Exemples

```
class Eleve:

    def __init__(self, nom, promo = 'ENSG', lnotes = None):

        self.nom_ = nom
        self.promo_ = promo
        if lnotes is None:
            self.lnotes_ = []
        else:
            self.lnotes_ = lnotes

        self.calculer_moyenne()

    def calculer_moyenne(self):

        if self.lnotes_ != []:
            self.moyenne_ = sum(self.lnotes_)/len(self.lnotes_)
        else:
            self.moyenne_ = None

    def ajouter_note(self, note):

        self.lnotes_.append(note)
        self.calculer_moyenne()

    def getMoyenne(self):

        return self.moyenne_
```

```
# Programme Principal
padawan = Eleve('Luke')
print(padawan.getMoyenne())

padawan.ajouter_note(14)
print(padawan.getMoyenne())
```

Attention à bien **mettre à jour** l'ensemble des attributs concernés par l'application d'une méthode donnée.

Les pièges à éviter

- ✗ Utilisation de valeurs par défaut

Il faut faire attention dès que la valeur par défaut est de type **mutable** (*e.g.* liste, dictionnaire, objet issu d'une classe 'fait-maison').

- ✗ Création de nouveaux attributs

Tous les attributs doivent être créés dans la méthode constructeur. Il est possible de le faire en **appelant une autre méthode** de la classe.

- ✗ Modification d'attributs et mises à jour

Attention à bien **mettre à jour** l'ensemble des attributs concernés par l'application d'une méthode donnée.

Le polymorphisme

Le polymorphisme ad hoc consiste à utiliser le même nom pour des méthodes qui ont le même usage dans des classes différentes.

Le polymorphisme - Exemple

Classe **Temperature**.

Créer une classe qui représente des températures. Elle a comme attributs le lieu de la mesure, l'unité de mesure et une liste de valeurs de températures.

Créer la méthode de calcul de moyenne des températures(créé un attribut moyenne).

Bonnes pratiques :

On créé un fichier par classe !
Il faut donc penser à importer le fichier définissant la classe Elève.

Le polymorphisme - Exemple

Fichier temperature.py

```
class Temperature:

    def __init__(self, lieu, unite, valeurs):

        self.lieu_ = lieu
        self.unite_ = unite
        self.valeurs_ = valeurs
        self.calculer_moyenne()

    def calculer_moyenne(self):

        if self.valeurs_ == []:
            self.moyenne_ = None

        else:
            self.moyenne_ =
sum(self.valeurs_)/len(self.valeurs_)

    def getMoyenne(self):

        return self.moyenne_
```

Fichier main.py

```
# Script principal (fichier main.py)

from eleve import *
from temperature import *

if __name__ == "__main__":

    padawan = Eleve("Luke", "1A", [12, 14])
    salle = Temperature("G007", "°C", [18, 17, 19])

    print(padawan.getMoyenne())
    print(salle.getMoyenne())
```

Le polymorphisme - Exemple

Classe **Groupe**.

Créer une classe qui représente un groupe d'élèves. Elle a comme attribut une liste d'élèves (i.e. liste d'instances de la classe Eleve).

Créer une méthode calculer_moyenne qui calcule la moyenne du groupe (moyenne des moyennes de chaque étudiant du groupe).

Créer une méthode ajouter_note qui rajoute la même note à chaque élève du groupe.

Le polymorphisme - Exemple

```
class Groupe:

    def __init__(self, leleves):

        self.leleves_ = leleves
        self.calculer_moyenne()

    def calculer_moyenne(self):

        if self.leleves_ == []:
            self.moyenne_ = None

        else:
            self.moyenne_ = 0
            nb = 0
            for eleve in self.leleves_:
                m = eleve.getMoyenne()
                if m is not None:
                    nb += 1
                    self.moyenne_ += m

            self.moyenne_ /= nb
```

On peut **utiliser** des objets dans d'autres objets. On peut alors utiliser leurs méthodes.

```
# Programme Principal
padawan = Eleve("Luke", "1A", [12, 10, 14])
princesse = Eleve("Leia", "1A", [17, 15, 8])

apprentis = Groupe([padawan, princesse])
```

Le polymorphisme - Exemple

```
class Groupe:

    def __init__(self, leleves):

        self.leleves_ = leleves
        self.calculer_moyenne()

    def calculer_moyenne(self):

        ...

    def ajouter_note(self, note):

        for eleve in self.eleves_:
            eleve.ajouter_note(note)

        self.calculer_moyenne()
```

Méthode ajouter_note de la classe Groupe.

Instance de la classe Groupe.

Méthode ajouter_note de la classe Elève.

Instance de la classe Elève.

Exemple

```
class Eleve:

    def __init__(self, nom, promo = 'ENSG', lnotes = None):

        self.nom_ = nom
        self.promo_ = promo
        if lnotes is None:
            self.lnotes_ = []
        else:
            self.lnotes_ = lnotes

        self.calculer_moyenne()
```

```
# Programme Principal
padawan = Eleve('Luke')
print(padawan)
```

```
<__main__.Eleve object at 0x7f2a315db2e8>
```

Ce n'est pas un message d'erreur !

Mais, **en général, ce n'est pas ce qu'on a envie d'afficher**…

```
# Programme Principal
padawan = Eleve('Luke',promo ='1A', lnotes = [12, 10, 14])
skywalker = Eleve('Luke',promo ='1A', lnotes = [12, 10, 14])
print(padawan == skywalker)
```

```
False
```

Il ne s'agit pas du même objet !

Mais si on voulait **comparer le contenu** ?

→ **Les méthodes spéciales**

Les méthodes spéciales

Les **méthodes spéciales** permettent aux « d'expliquer » à certaines **fonctions connues** (e.g. print) ou **opérateurs** (e.g. +, *) comment s'appliquer aux objets que l'on a créé. On parle de **surcharge** de fonctions ou d'opérateurs.

Une méthode spéciale a un nom qui commence et qui termine par 2 underscores (__). Par exemple __init__ est une méthode spéciale. **Elle ne s'appelle pas comme une méthode normale**, mais utilise les fonctions ou opérateurs qui lui sont associées. Les noms des méthodes spéciales sont déjà identifiées dans Python, donc n'importe quelle méthode ne peut pas être rendue spéciale !

Méthode spéciale	Définition	Renvoi	Surcharge	Appel
__repr__	def __repr__(self) :	1 chaîne de caractères	print( )	print(a)
__add__	def __add__(self, other) :	Instance de la classe	+	a + b
__sub__	def __sub__(self, other) :	Instance de la classe	-	a - b
__eq__	def __eq__(self, other) :	Booléen	==	a == b
__lt__	def __lt__(self, other) :	Booléen	<	a < b
__ge__	def __ge__(self, other) :	Booléen	≥	a >= b

Mais aussi __mul__ (*), __truediv__ (/), __le__ (≤), __gt__ (>), __pow__(**), __str__ (str), __floordiv__(//), __ne__(!=), etc.

Les méthodes spéciales - Exemple

Dans la classe Elève :

Créer une **méthode qui surcharge la fonction print**. On affiche alors le nom de l'élève, sa promo et sa moyenne.

Créer une **méthode qui surcharge l'opérateur d'égalité** (==). Deux élèves sont considérés égaux s'ils ont la même moyenne et qu'ils sont de la même promo.

Dans la classe Groupe :

Créer une méthode qui **surcharge l'opérateur d'addition**. Lors de la somme de deux groupes, les liste d'élèves sont concaténées.

Penser à tester vos méthodes dans les programmes principaux !

Les méthodes spéciales - Exemple

```
class Eleve:

    def __init__(self, nom, promo = 'ENSG', lnotes = None):
        ...

    def calculer_moyenne(self):
        ...

    def ajouter_note(self, note):
        ...

    def getMoyenne(self):
        ...

    def __repr__(self):

        return  "Eleve " + self.nom_ + " promo " + self.promo_ +\
                " moyenne " + str(self.moyenne_) + "\n"

    def __eq__(self, other):

        if self.moyenne_ == other.getMoyenne():
            if self.promo_ == other.promo_:
                return True

        return False
```

```
# Programme Principal
padawan = Eleve("Luke", "1A", [10, 12, 14])
print(padawan)

princess = Eleve("Leia", "1A", [10, 12, 13, 13])
jedi = Eleve("Kenobi", "2A", [18, 15, 17])

print(padawan == princess)
print(padawan == jedi)
```

Les méthodes spéciales - Exemple

```
class Groupe:

    def __init__(self, leleves):

        ...

    def calculer_moyenne(self):

        ...

    def ajouter_note(self, note):

        ...

    def __add__(self, other):

        return Groupe(self.leleves_ + other.leleves_)
```

```
# Programme Principal
padawan = Eleve("Luke", "1A", [12, 10, 14])
princesse = Eleve("Leia", "1A", [17, 15, 18])

apprentis = Groupe([padawan, princesse])

robot = Eleve("R2D2", "2A", [4, 16, 17])
androide = Eleve("C3PO", "2A", [15, 5, 10])

assistants = Groupe([robot, androide])

resistance = apprentis + assistants
```