



## Présentation ADRAR Pôle Numérique

Suivez-nous... LinkedIn  et Twitter  @Adrar\_Numerique



# **ADRAR\o/** **PÔLE NUMERIQUE**

**PÔLE NUMERIQUE DU CENTRE DE FORMATION ADRAR**

- > SUPPORT, ADMINISTRATION SYSTEMES & RESEAUX
- > DEVELOPPEMENT D'APPLICATIONS WEB & MOBILES
- > WEBDESIGN & DEVELOPPEMENT INTERFACES
- > TRANSFORMATION NUMERIQUE DES ENTREPRISES

<http://www.adrar-numerique.com>

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)



Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

# LA PROGRAMMATION ORIENTEE

## OBJET : POO

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

La programmation orientée objet (POO) est une méthode de structuration de code permettant de rendre votre programme mieux organisé et plus lisible. Comme vous le savez en python TOUT est objet, c'est donc un langage propice à l'apprentissage de la POO. Quelques définitions :

- Classe : Ensemble de code regroupant des variables et des méthodes permettant de créer des objets.
- Objet : Instance d'une classe disposant d'attributs et de méthodes.
- Constructeur : Méthode qui va être utilisée pour instancier un objet.
- Assesseur (getter) : Méthode qui va être utilisé pour afficher un attribut.
- Mutateur (setter) : Méthode qui va être utilisé pour modifier un attribut.
- Héritage : Principe voulant qu'une classe hérite des propriétés de son parent. De base toutes les classes héritent de la classe Object.
- Composition : Principe voulant qu'un attribut d'une classe soit une instance d'une autre classe.

Suivez-nous sur LinkedIn 

Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

# Heritage et Composition

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Nous avons vu comment créer une classe avec ses attributs et ses méthodes privées et publiques. Nous avons également vu comment initialiser un objet de notre classe et manipuler nos éléments privés.

Nous allons maintenant aborder deux autres principes de la POO : l'héritage et la composition.

Ces deux principes illustrent la relation qu'il peut exister entre plusieurs classes. Il y a deux possibilités : une classe EST UNE particularité d'une autre classe (héritage) ou une classe A UNE entité d'une autre classe (composition).

Par exemple si nous avons trois classes *Rectangle*, *Carré* et *Point* alors une relation d'héritage existe entre *Carré* et *Rectangle* alors qu'une relation de composition existe entre *Point* et *Rectangle* (ou *Point* et *Carré*)

- Un carré EST UN rectangle
- Un rectangle A UN (ou plusieurs) point



Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Visualisons nos trois classes :

Classe : point

Attributs :

Position\_x : int

Position\_y : int

Méthodes:

Afficher()

Classe : Rectangle

Attributs :

Longueur : str

Largeur : int

Nom : str

Centre : point

Méthodes:

Calculer aire()

Calculer périmètre()

Afficher()

Classe : Carré (Rectangle)

Attributs :

Nom : str

Méthodes:

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

En python nous allons donc déclarer une classe Point(), une classe Rectangle() et une classe Carre(Rectangle).

```
class Point:
    def __init__(self, pos_x, pos_y):
        self.pos_x = pos_x
        self.pos_y = pos_y
```

```
class Rectangle:
    def __init__(self, longueur, largeur):
        self.longueur = longueur
        self.largeur = largeur
        self.nom = "rectangle"
        self.centre = Point(self.longueur/2, self.largeur/2)
```

```
class Carre(Rectangle):
    def __init__(self, cote):
        super().__init__(cote, cote)
        self.nom = "carre"
```



Suivez-nous sur LinkedIn Twitter @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Intéressons nous d'abord à la définition de notre classe Rectangle:

```
class Rectangle:
    def __init__(self, longueur, largeur):
        self.longueur = longueur
        self.largeur = largeur
        self.nom = "rectangle"
        self.centre = Point(self.longueur/2, self.largeur/2)

    def calculerAire(self):
        print("mon aire est de {}".format(self.longueur * self.largeur))

    def calculerPerimetre(self):
        print("mon périmètre est de {}".format((self.longueur * 2) + (self.largeur * 2)))
```

Notre rectangle est donc initialisé avec une longueur et une largeur. Le nom de rectangle lui est attribué et son centre est un objet de type Point dont les paramètres sont la moitié de sa longueur et la moitié de sa largeur. La méthode calculerAire() calcule l'aire du rectangle en multipliant sa longueur et sa longueur. La méthode calculerPerimetre() calcule le périmètre du rectangle en additionnant ses cotés.

```
monRectangle = Rectangle(8, 7)
monRectangle.calculerPerimetre()
monRectangle.calculerAire()
```

```
mon périmètre est de 30
mon aire est de 56
```

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Regardons maintenant la définition de la classe Carre :

```
class Carre(Rectangle):
    def __init__(self, cote):
        super().__init__(cote, cote)
        self.nom = "carre"
```

Notre carré est initialisé en définissant une longueur de coté. Dans son constructeur on commence par appelé le constructeur de sa classe mère avec comme paramètre le coté passé en paramètre de construction de l'objet Carre. *Super()* ici fait référence à la classe mère.

Notre classe Carre ne dispose pas de méthodes propre à elle. Par contre, héritant de la classe Rectangle nous pouvons appeler les méthodes `calculerAire()` et `calculerPerimetre` sur un objet de type Carre.

```
monCarre = Carre(9)
monCarre.calculerPerimetre()
monCarre.calculerAire()
```

```
mon périmètre est de 36
mon aire est de 81
```

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Regardons maintenant la définition de la classe Point :

```
class Point:
    def __init__(self, pos_x, pos_y):
        self.pos_x = pos_x
        self.pos_y = pos_y

    def affiche(self):
        print("abscice : {}, ordonnée : {}".format(self.pos_x, self.pos_y))
```

Notre point est initialisé avec deux valeurs *pos\_x* et *pos\_y* et possède une méthode *affiche()* affichant les attributs. Nous pouvons créer manuellement un objet Point et l'afficher :

```
monPoint = Point(8, 9)
monPoint.affiche()
```

```
abscice : 8, ordonnée : 9
```

Suivez-nous sur LinkedIn Twitter @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Dans le constructeur de notre classe Rectangle nous avons défini un attribut *centre* comme étant un objet de la classe Point :

```
class Rectangle:
    def __init__(self, longueur, largeur):
        self.longueur = longueur
        self.largeur = largeur
        self.nom = "rectangle"
        self.centre = Point(self.longueur/2, self.largeur/2)
```

*Centre* étant une instance de la classe point nous pouvons donc utiliser les méthodes de sa classe :

```
monRectangle = Rectangle(8, 7)
monRectangle.centre.affiche()
```

```
abscisse : 4.0, ordonnée : 3.5
```

Nous pouvons même utiliser les méthodes de la classe point dans la classe Rectangle :

```
def affiche(self):
    print("Je suis un {} de longueur {} et de largeur {}".format(self.nom, self.longueur, self.largeur))
    print("Mon centre :", end="\n\t")
    self.centre.affiche()
```

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Nous pouvons également effectuer la même chose avec un objet de la classe Carre vue qu'elle hérite de la classe Rectangle :

```
monCarre = Carre(9)
monCarre.centre.affiche()
monCarre.affiche()
```

```
abscice : 4.5, ordonnée : 4.5
Je suis un carre de longueur 9 et de largeur 9
Mon centre :
    abscice : 4.5, ordonnée : 4.5
```

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

# EXERCICE

Définissez une classe User qui aura comme attributs un nom, un mot de passe et un compte.  
Définissez une classe Premium qui héritera de la classe User et qui aura comme attribut un montant d'emprunt.  
Récupérez le fichier de classe Compte de l'exercice du même nom pour pouvoir générer le compte d'un User.  
Un utilisateur pourra afficher ses informations et créer un compte. Un utilisateur premium peut effectuer les actions d'un utilisateur et également emprunter de l'argent

Réalisez un programme qui demande à l'utilisateur si il est nouveau, si c'est le cas il lui propose de créer un compte utilisateur classique ou premium. Le programme doit ensuite créer un compte bancaire à l'utilisateur en lui demandant le montant qu'il veut mettre sur son compte en premier crédit. Le compte utilisateur est ensuite enregistré dans un tableau.

Si l'utilisateur a déjà un compte il lui propose de se connecter (vérification de mot de passe) puis lui propose les différentes actions possibles relative à son compte (utilisation de la fonction builtin *isinstance(objet, class)*)

Bonus : implémentation de la méthode privée *rembourserEmprunt* qui prélève à intervalle régulier un pourcentage du montant emprunté.



Suivez-nous sur LinkedIn 

Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Classe : Compte

Attributs :

Date création : int

Solde : int

Propriétaire: str

Banque: str

Méthodes:

Créditer(montant)

Débiter(montant)

Afficher(solde)

Classe : User

Attributs :

Nom : str

Mdp : str

CompteBanquaire : compte

Méthodes:

AfficherInformations()

CreerCompte(nom, montant)

Classe : Prenium(User)

Attributs :

Emprunt : int

Méthodes:

Emprunter(montant)