	**Structures de données**	**NSI T^{ale}**
	Programmation Orientée Objet	Cours/TD

1. Classe et objet

Un **objet** est une variable (presque) comme les autres

Le **type d'un objet** est un type complexe qu'on appelle **classe**

Une **classe** regroupe un ensemble d'**attributs** (données) et un ensemble de **méthodes** de traitement.

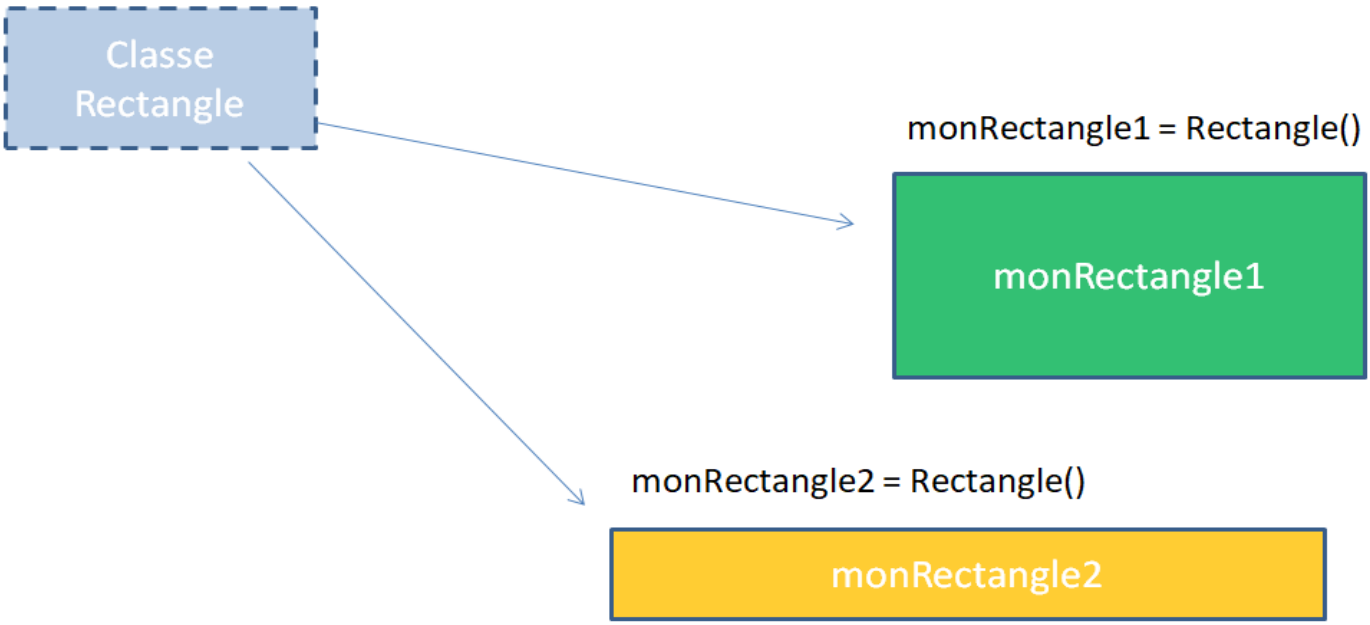
Exemple simple :

```
class Eleve:
    nom = None
    classe = None

    def change_classe(self, nouvelle_classe):
        self.classe = nouvelle_classe
```

Exemple plus complexe :

```
class Rectangle:

    nom = "rectangle"
    # créerRectangle: → Rectangle[E,E,E,E]
    def __init__(self, longueur = None, largeur = None, origine_x = None, origine_y = None):
        self._longueur = longueur      # Entier
        self._largeur = largeur        # Entier
        self.__origine_x = origine_x    # Entier
        self.__origine_y = origine_y    # Entier
        self.couleur = "rouge"

    def change_nom(self, nouveau_nom):
        self.__class__.nom = nouveau_nom

    def change_couleur(self, nouvelle_couleur):
        self.couleur = nouvelle_couleur

    def set_dimensions(self, longueur, largeur):
        self._longueur = longueur
        self._largeur = largeur

    def set_position(self, origine_x, origine_y):
        self.__origine_x = origine_x
        self.__origine_y = origine_y

    # estVide : Rectangle[E,E,E,E] → Booléen
    def est_vide(self):
        return self._longueur is None or self._largeur is None

    def deplace(self, x, y):
        self._origine_x += x
        self._origine_y += y

    def get_dimensions(self):
        return self._longueur, self._largeur

    def get_position(self):
        return self._origine_x, self._origine_y

    def surface(self):
        if self.estVide():
            return 0
        else:
            return self._longueur * self._largeur
```

2. Classe Rectangle et objets



On parle aussi d'**instances**.

Si par la suite j'écris monRectangle1 = monRectangle2, ils représentent alors le même objet et sont donc modifiés ensemble.

3. Construire une Classe

```
class Rectangle:

    # créerRectangle: → Rectangle[E,E,E,E]
    def __init__(self, longueur = None, largeur = None, origine_x = None, origine_y = None):
        self._longueur = longueur      # Entier
        self._largeur = largeur        # Entier
        self._origine_x = origine_x    # Entier
        self._origine_y = origine_y    # Entier
```

Cette méthode est appelée le **constructeur** de la classe

En Python, il ne peut y avoir qu'un constructeur mais on peut lui donner des **valeurs par défaut** comme dans l'exemple.
Dans d'autres langages (ex: Java) une classe peut avoir de nombreux constructeurs qui proposent des options différentes.

4. Portée des variables

En fonction de la position de la variable, il s'agit d'une **variable de classe ou d'instance** :

```
class Rectangle:

    nom = "rectangle"
    # créerRectangle: → Rectangle[E,E,E,E]
    def __init__(self, longueur = None, largeur = Nor
        self._longueur = longueur       # Entier
        self._largeur = largeur         # Entier
        self.__origine_x = origine_x    # Entier
        self.__origine_y = origine_y    # Entier
        self.couleur = "rouge"

    def change_nom(self, nouveau_nom):
        self.__class__.nom = nouveau_nom

    def change_couleur(self, nouvelle_couleur):
        self.couleur = nouvelle_couleur
```

```
In [6]: a = Rectangle()

In [7]: a.couleur
Out[7]: 'rouge'

In [8]: a.nom
Out[8]: 'rectangle'

In [9]: b = Rectangle()

In [10]: b.change_couleur("bleu")

In [11]: b.change_nom("carré")

In [12]: b.couleur
Out[12]: 'bleu'

In [13]: b.nom
Out[13]: 'carré'

In [14]: a.couleur
Out[14]: 'rouge'

In [15]: a.nom
Out[15]: 'carré'

In [16]: Rectangle.nom
Out[16]: 'carré'

In [17]: Rectangle.couleur
Traceback (most recent call last):

  File "<ipython-input-17-4086649b1f65>", line 1,
in <module>
    Rectangle.couleur

AttributeError: type object 'Rectangle' has no
attribute 'couleur'
```

La variable **nom** est **commune à la classe et toutes ses instances** et peut-être manipulée indifféremment par le nom de la classe ou de l'une de ses instances. Ce n'est pas le cas de la variable **couleur** qui **dépend de l'instance**.

5. Encapsulation : public / privé

L'**encapsulation** consiste à protéger les données et méthodes d'une classe.
C'est une **notion fondamentale de la POO**.

En règle générale, tous les **attributs** (données) d'une classe ne devraient être accessibles qu'à l'intérieur de cette classe : ils doivent être **privés**.
De même, certaines méthodes ne devraient pas être accessible en dehors de la classe.

En revanche, certaines **méthodes** doivent permettre des **échanges** avec l'extérieur : elles sont **publiques**.

Python, contrairement à d'autres langages (comme Java), ne bloque jamais l'accès aux attributs et méthodes d'une classe. La philosophie du langage est plus axée sur la confiance : si nous voulons accéder à un élément protégé ou privé c'est que nous avons une bonne raison de le faire.

D'ailleurs, une phrase du fondateur de Python, Guido van Rossum, est fréquemment citée à ce propos : "We are all consenting adults here". Elle signifie : "Nous sommes tous des adultes consentants". Sous entendu : si vous voulez vous tirer une balle dans le pied, allez-y, vous êtes adulte après tout.

Il existe cependant deux mécanismes pour indiquer qu'un attribut ou qu'une méthode est privée :

Par convention on utilise **l'underscore «_»** pour indiquer que l'attribut (ou la méthode) doit être considéré comme privé.

Si on double **l'underscore «__»** on protège un peu plus car le nom de l'attribut est alors précédé du nom de la classe à l'extérieur de la classe.

```
class Rectangle:

    nom = "rectangle"
    # créerRectangle: → Rectangle[E,E,E,E]
    def __init__(self, longueur = None, large
        self._longueur = longueur       # Ent
        self._largeur = largeur         # Ent
        self.__origine_x = origine_x    # En
        self.__origine_y = origine_y    # En
        self.couleur = "rouge"
```

```
In [84]: a = Rectangle(10,10,100,50)

In [85]: print(a._longueur)
    ...:
10

In [86]: print(a.__origine_x)
Traceback (most recent call last):

  File "<ipython-input-86-78498b177085>", line 1,
in <module>
    print(a.__origine_x)

AttributeError: 'Rectangle' object has no attribute
'__origine_x'


In [87]:

In [87]: print(a._Rectangle__origine_x)
100
```