RÉPUBLIQUE FRANÇAISE
*Liberté*
*Égalité*
*Fraternité*

UNIVERSITÉ TOULOUSE III PAUL SABATIER
FACULTÉ SCIENCES ET INGÉNIERIE

# Base de la POO en Python

Mathieu RAYNAL
*mathieu.raynal@irit.fr*
*http://www.irit.fr/~Mathieu.Raynal*

# Déclarer une classe

- Déclarer une classe

```
class MaClasse:
  …
```

- Constructeur

```
def __init__(self [,params]):
  …
```

- Il ne peut y avoir qu'un seul constructeur dans la classe
  - Possibilité de mettre des valeurs par défaut aux paramètres

```
def __init__(self, x=0, y=0):
  …
```

# Attributs et méthodes d'une classe

- Attribut (variable d'instance)
  - Généralement déclaré et initialisé dans le constructeur

```
self.monAttribut = saValeur
```

- Les méthodes se déclarent comme les fonctions
  - Obligatoirement un premier paramètre nommé ***self***

```
def maMethode(self [,params]):
```

# Visibilité des membres

- Les membres peuvent être
  - publics
    - Accès possible depuis une instance de la classe
  - protégés : nom débute par _
    - Accès qu'à l'intérieur de la classe ou depuis une classe fille
  - privés : nom débute par __
    - Accès qu'à l'intérieur de la classe

# Les getters (accesseurs) et setters (mutateur)

- Dans les deux cas, le nom de la méthode est le nom de l'attribut à modifier
  - Pour l'accesseur, faire précéder la méthode de *@property*

```
@property
def x(self):
  return self.__x
```

  - Pour le mutateur, faire précéder la méthode de *@**nomAttribut**.setter*

```
@x.setter
def x(self, x):
  self.__x = x
```

# Les méthodes __str__ et __repr__

- Pour afficher la représentation d'un objet, il faut définir une des méthodes suivantes

```
def __str__(self):
```

- Ou

```
def __repr__(self):
```

# Surcharge d'opérateur

## Opérateurs mathématiques

| Méthode | Opérateur |
|---|---|
| __add__ | + |
| __sub__ | - |
| __mul__ | * |
| __truediv__ | / |
| __mod__ | % |
| __power__ | ** |

## Opérateurs de comparaison

| Méthode | Opérateur |
|---|---|
| __eq__ | == |
| __ne__ | != |
| __gt__ | > |
| __ge__ | >= |
| __lt__ | < |
| __le__ | <= |

```
def __add__(self, p):
    return Point(self.x+p.x, self.y+p.y)
```

```
p1 = Point(3,2)
p2 = Point(4,5)
p = p1 + p2
```

# Héritage

- Déclaration de la classe dont on hérite

```
class MaClasse(ClasseMere):
  …
```

- Utilisation de méthodes de la classe mère

```
super().laMethodeClasseMere()
```