

# Un monde d'objets

---

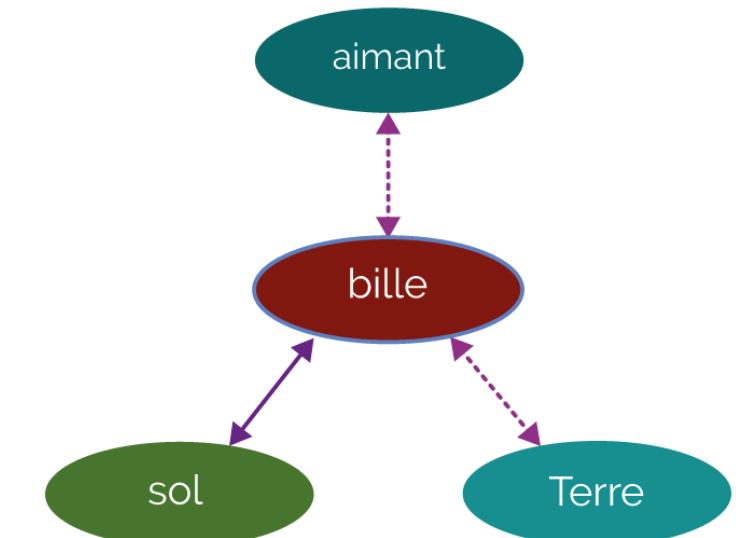
Outils Numériques / Semestre 6  
/ Institut d'Optique / ONIP-2

# Des objets qui interagissent



<https://www.lepoint.fr/dossiers/societe/velo-libre-service-velib/>

[https://masevaux.fr/objets\\_trouves/](https://masevaux.fr/objets_trouves/)



<https://www.maxicours.com/se/cours/les-diagrammes-objet-interaction/>

# Des objets qui interagissent

**Un objet** est caractérisé par :

ETAT

COMPORTEMENT



# Des objets qui interagissent

**Un objet** est caractérisé par :

**ETAT**

**COMPORTEMENT**



<https://www.lepoint.fr/dossiers/societe/velo-libre-service-velib/>

**CHIEN**

nom, couleur, race, poids...

manger, courir, aboyer...

**TRAIN**

marque, type, vitesse max...

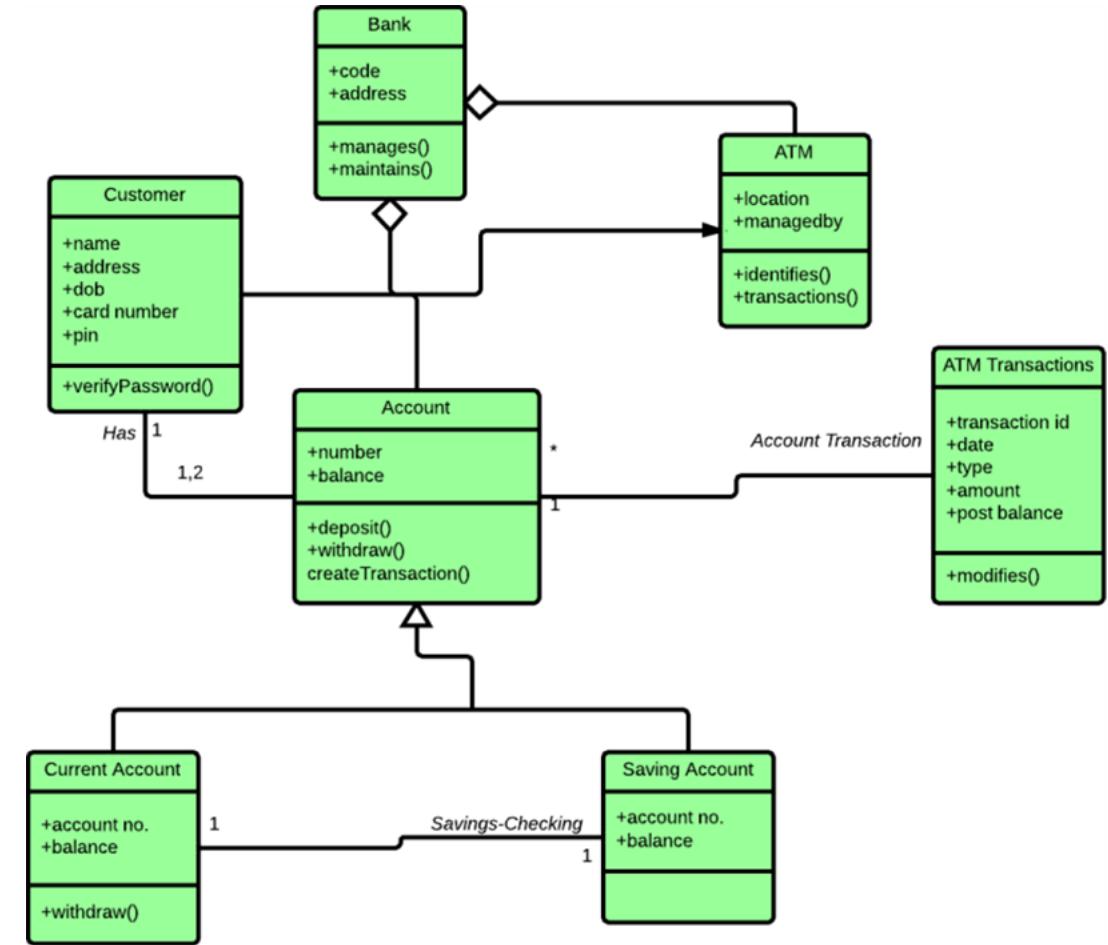
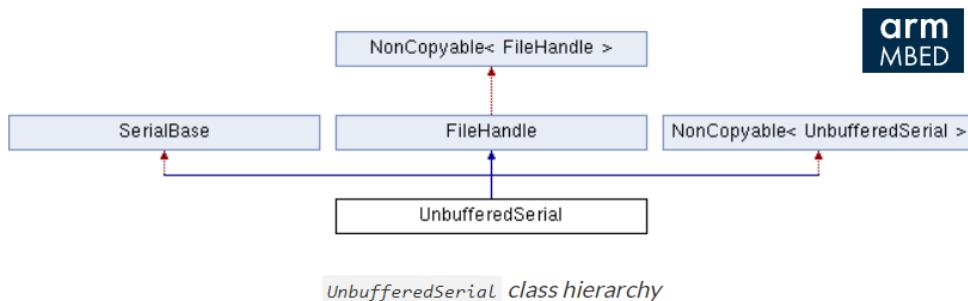
rouler, freiner, klaxonner...

# Des objets en informatique

Un **objet** est une **instance** de **classe**, possédant son propre état et son propre comportement

Docs › API references and tutorials › Drivers › Serial (UART) APIs › UnbufferedSerial

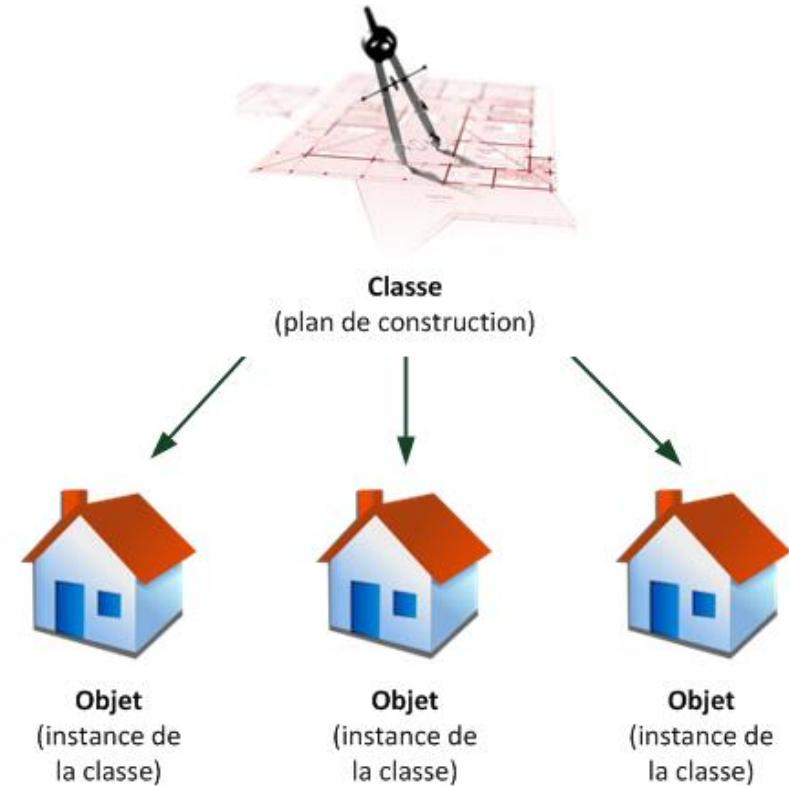
## UnbufferedSerial



# Programmation orientée objet

## Eléments de base

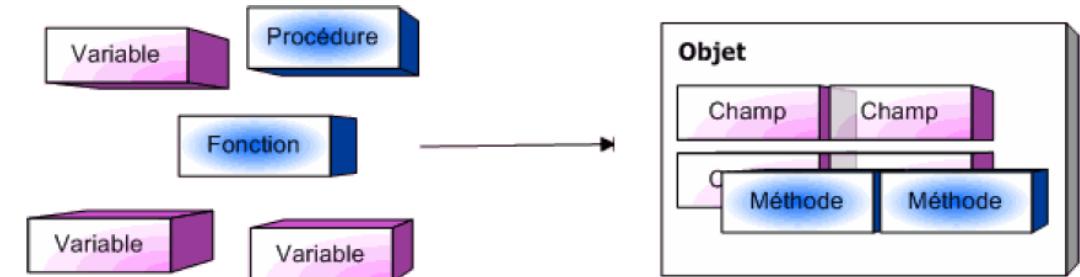
- **Classe** : rassemblement de différents **attributs** (état d'un objet) et **méthodes** (actions possibles d'un objet)
- **Objet** : instance d'une classe



# Programmation orientée objet

## Concept fondamental

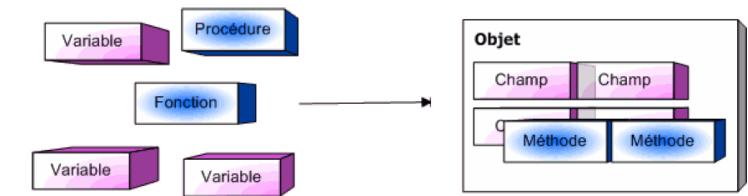
- **Encapsulation** : regroupement de différentes données et fonctions sous une même entité



# Programmation orientée objet

## Concepts fondamentaux

- **Encapsulation** : regroupement de différentes données et fonctions sous une même entité
- **Héritage** : arborescence de classes permettant la spécialisation  
*(notion non abordée dans ce module)*



classe ***numpy.ndarray***

Attributs

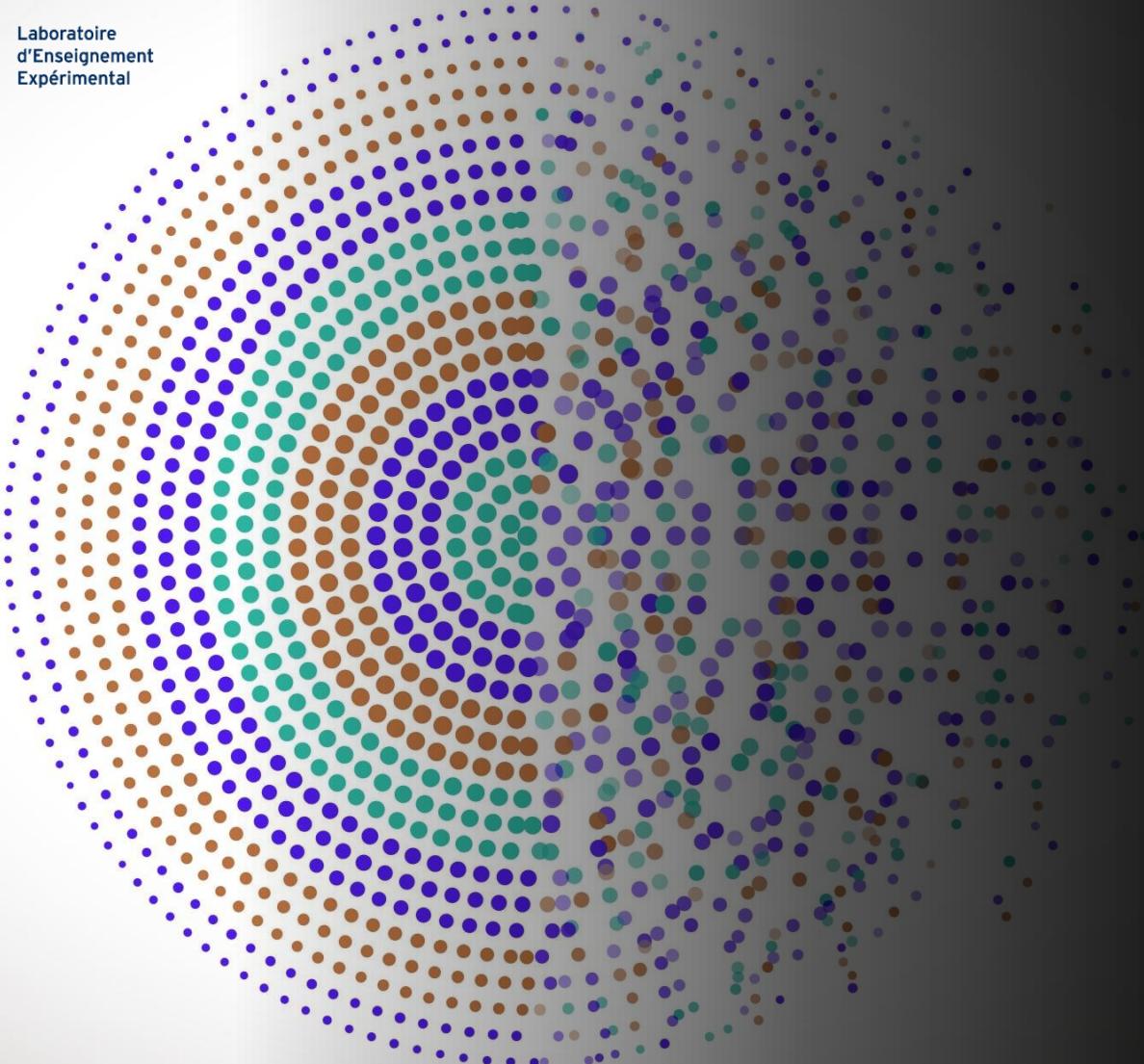
- *shape* (*Tuple d'entiers*)
- *data* (*buffer*)

ETAT

Méthodes

- *max ([axis...])*
- *resize (new\_shape...)*

COMPORTEMENT



# POO en Python

---

Outils Numériques / Semestre 6  
/ Institut d'Optique / ONIP-2

# Exemple d'une classe

**Encapsulation** : regroupement de différentes données et fonctions sous une même entité

```
import datetime

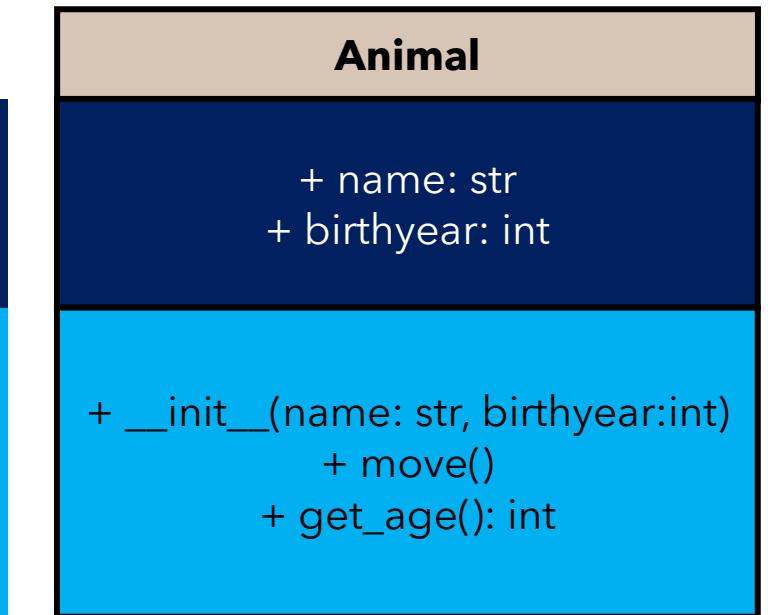
class Animal:
    """ object class Animal"""
    def __init__(self, name:str, birthyear:int):
        """ Animal class constructor
        :param name: name of the animal
        :param birthyear: year of birth of the animal
        """
        self.name = name
        self.birthyear = birthyear

    def move(self):
        print(f"\t{self.name} is moving")

    def get_age(self) -> int:
        return datetime.date.today().year - self.birthyear
```

COMPORTEMENT

ETAT





# Exemple d'une classe

**Encapsulation** : regroupement de différentes données et fonctions sous une même entité

ETAT

**variables**, propres à un objet (*instance d'une classe*), nommées **attributs**

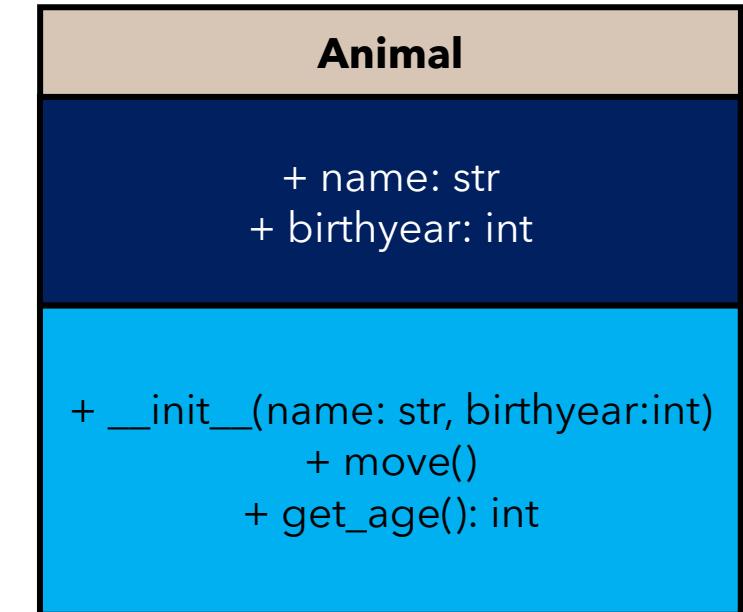
COMPORTEMENT

**Fonctions** associées à un objet (*instance d'une classe*), nommées **méthodes**

`__init__(self,...)` est le **constructeur** : méthode appelée à **l'instanciation d'un objet**      - **OBLIGATOIRE !**

**move()** et **get\_age()** sont des fonctions propres à cette classe

ETAT  
COMPORTEMENT



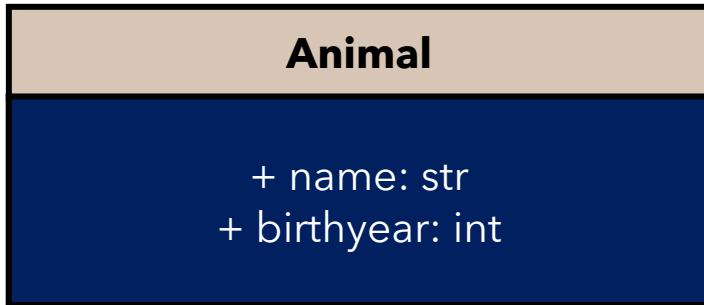
**self** est le mot clé utilisé pour accéder aux méthodes et attributs d'instance

# Utilisation d'une classe

## Instanciation d'un objet

```
def __init__(self, name:str, birthyear:int):  
    self.name = name  
    self.birthyear = birthyear
```

ETAT



```
animal1 = Animal("Felix", 2021)
```

```
animal2 = Animal("Garfield", 2015)
```

instances



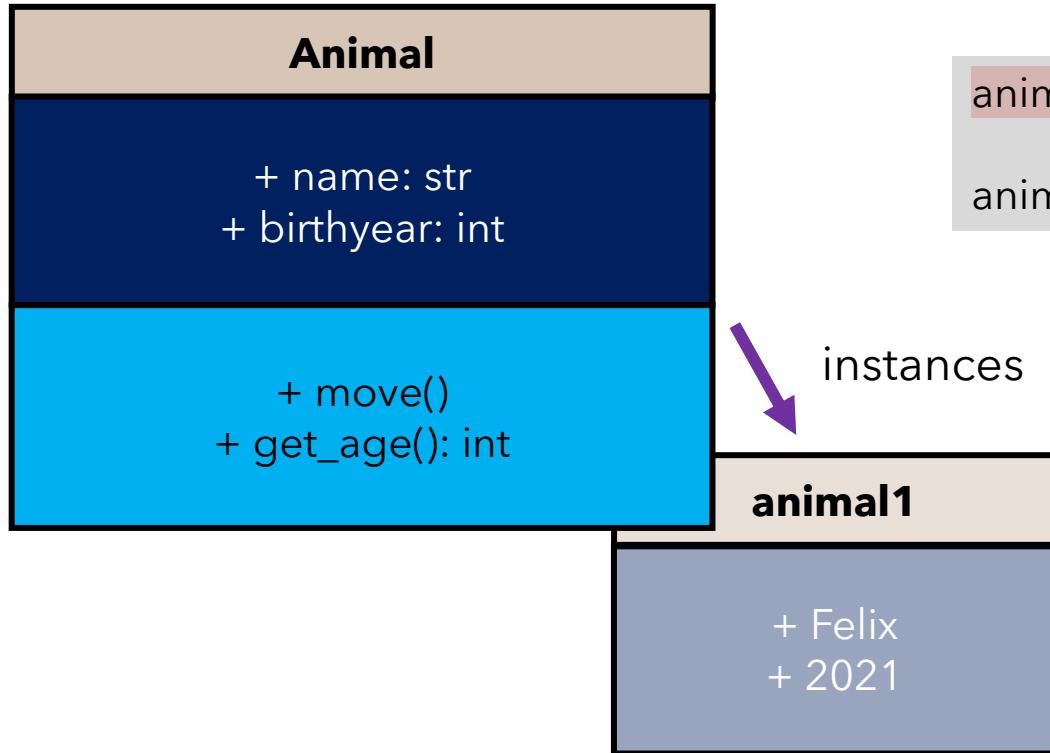
# Utilisation d'une classe

## Accès aux attributs d'un objet

```
def __init__(self, name:str, birthyear:int):
    self.name = name
    self.birthyear = birthyear
```

ETAT

COMPORTEMENT



```
animal1 = Animal("Felix", 2021)
```

```
animal2 = Animal("Garfield", 2015)
```

```
print("Animal 1 Name = ", animal1.name)
```

```
>>> Animal 1 Name = Felix
```

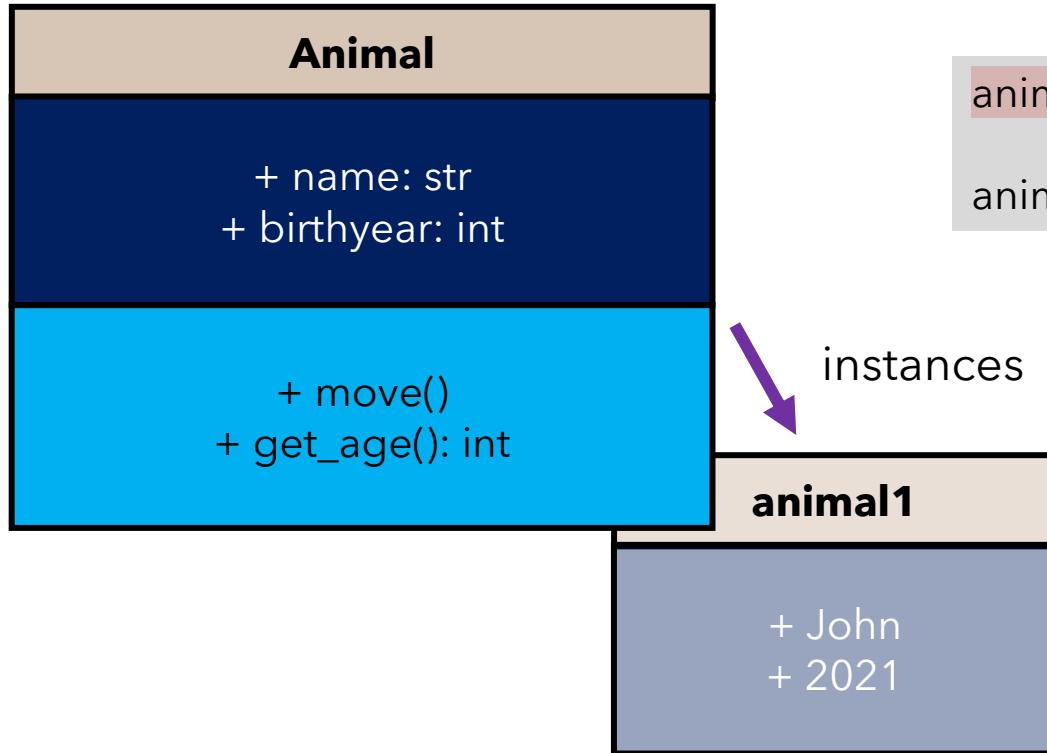
# Utilisation d'une classe

## Accès aux attributs d'un objet

```
def __init__(self, name:str, birthyear:int):
    self.name = name
    self.birthyear = birthyear
```

ETAT

COMPORTEMENT



```
animal1 = Animal("Felix", 2021)
```

```
animal2 = Animal("Garfield", 2015)
```

```
print("Animal 1 Name = ", animal1.name)
```

```
>>> Animal 1 Name = Felix
```

```
animal1.name = "John"
```

```
print("Animal 1 Name = ", animal1.name)
```

```
>>> Animal 1 Name = John
```

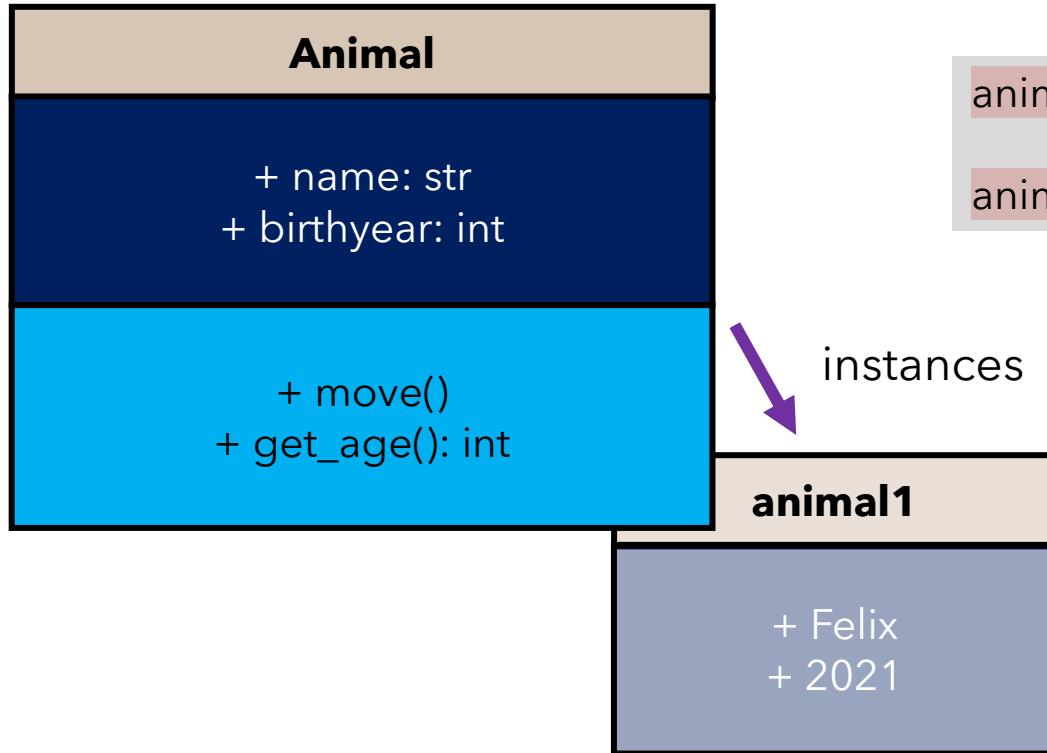
# Utilisation d'une classe

## Appel à une méthode à l'extérieur d'une classe

```
def move(self):
    print(f"\t[ {self.name} ] is moving")
```

ETAT

COMPORTEMENT



```
animal1 = Animal("Felix", 2021)
```

```
animal2 = Animal("Garfield", 2015)
```

```
animal1.move()
```

```
>>> [ Felix ] is moving
```

```
animal2.move()
```

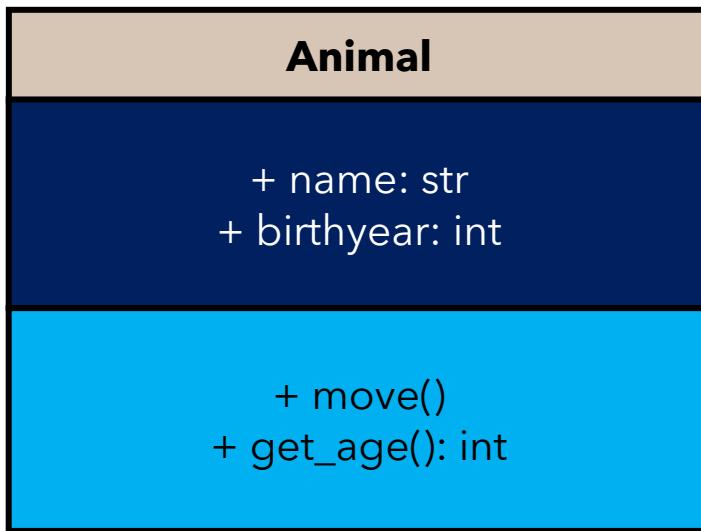
```
>>> [ Garfield ] is moving
```

# Utilisation d'une classe

## Liste d'objets

ETAT

COMPORTEMENT



```
animal1 = Animal("Felix", 2021)
```

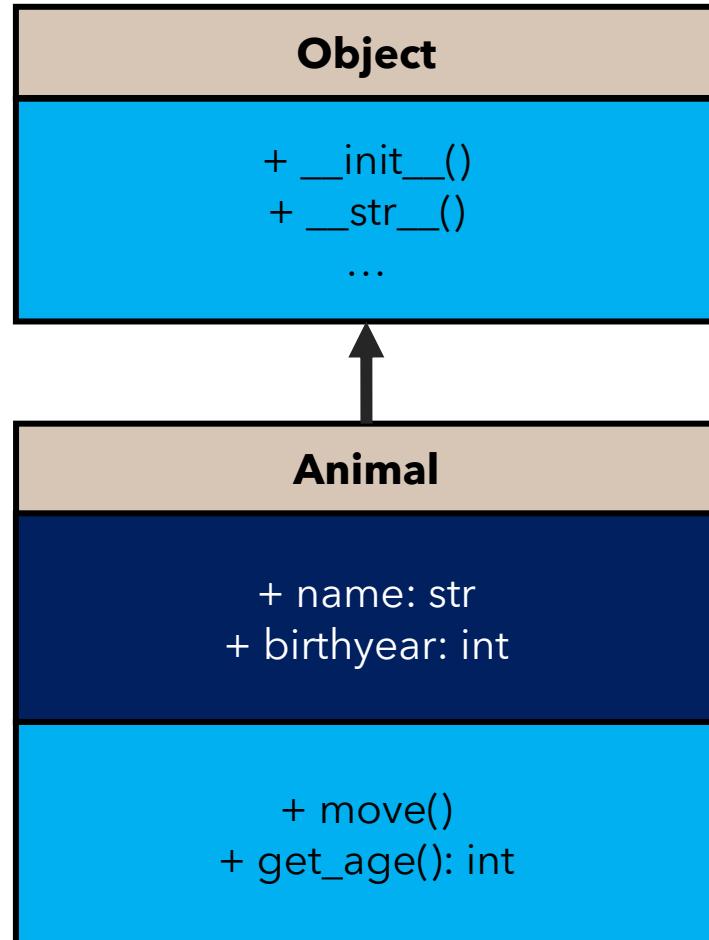
```
animal2 = Animal("Garfield", 2015)
```

```
animaux = []
animaux.append(animal1)
animaux.append(animal2)
animaux[0].move()
```

```
>>> [ Felix ] is moving
```

# Objets en Python

## Gestion des objets



ETAT

COMPORTEMENT

```
animal1 = Animal("Felix", 2021)
```

```
print(animal1)
```

```
>>> <__main__.Animal object at 0x000001E4FA066750>
```

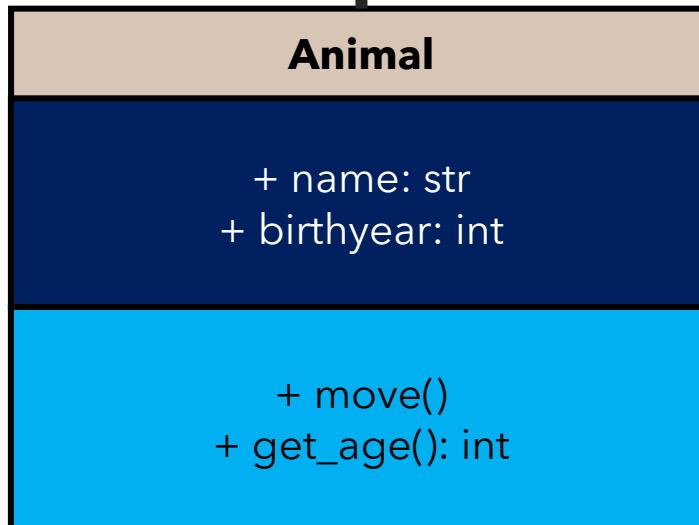
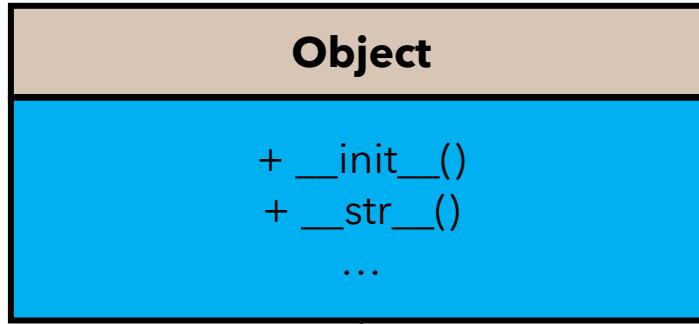
# Objets en Python

## Gestion des objets / Redéfinition de fonctions

Super classe !

ETAT

COMPORTEMENT



```

def __str__(self):
    str = f"Animal [ {self.name} ] born in {self.birthyear}"
    return str
  
```

```
animal1 = Animal("Felix", 2021)
```

```
print(animal1)
```

```
>>> <__main__.Animal object at 0x000001E4FA066750>
```

```
>>> Animal [ Felix ] born in 2021
```

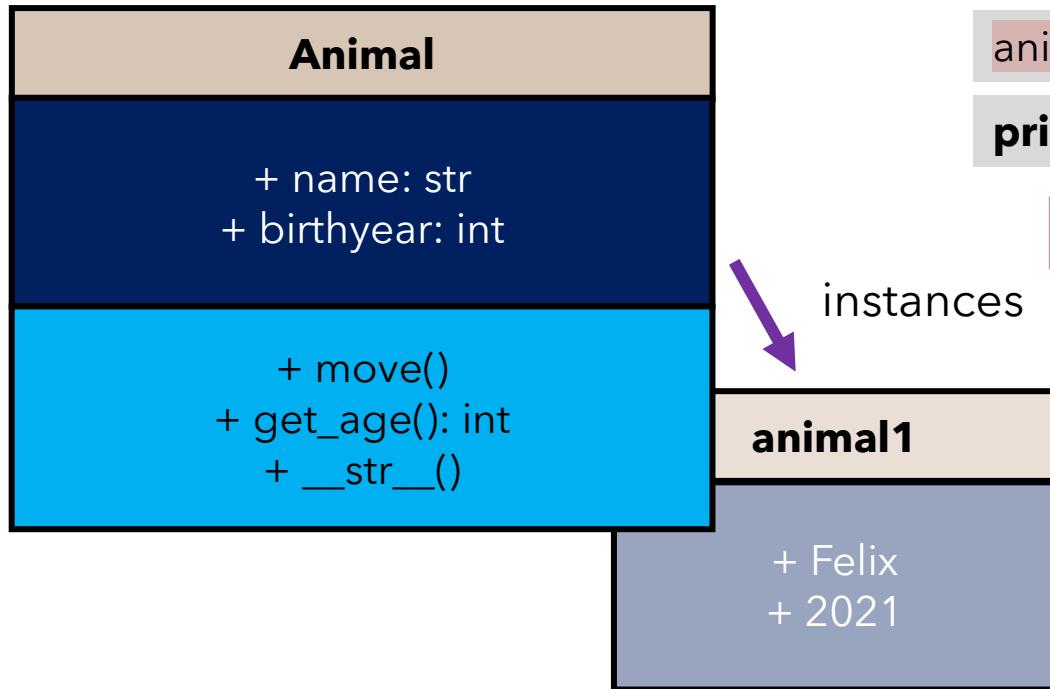
# Utilisation d'une classe

## Appel à une méthode à l'intérieur d'une classe

```
def __str__(self):
    str = f"Animal [ {self.name} ] born in {self.birthyear}"
    str += f" ({self.get_age()} yo)"
    return str
```

ETAT

COMPORTEMENT



```
animal1 = Animal("Felix", 2021)
```

```
print(animal1)
```

```
>>> Animal [ Felix ] born in 2021 (4 yo)
```

# Programmation orientée objet

## Quelques règles

- Une classe possède **obligatoirement** un **constructeur `__init__`**

- Le **nom des méthodes ne doit pas commencer par `__`** (double underscore)

*(signification très particulière en Python - utilisation réservée à certaines méthodes ou attributs)*

The [Google Python Style Guide](#) has the following convention:

ClassName

method\_name

function\_name

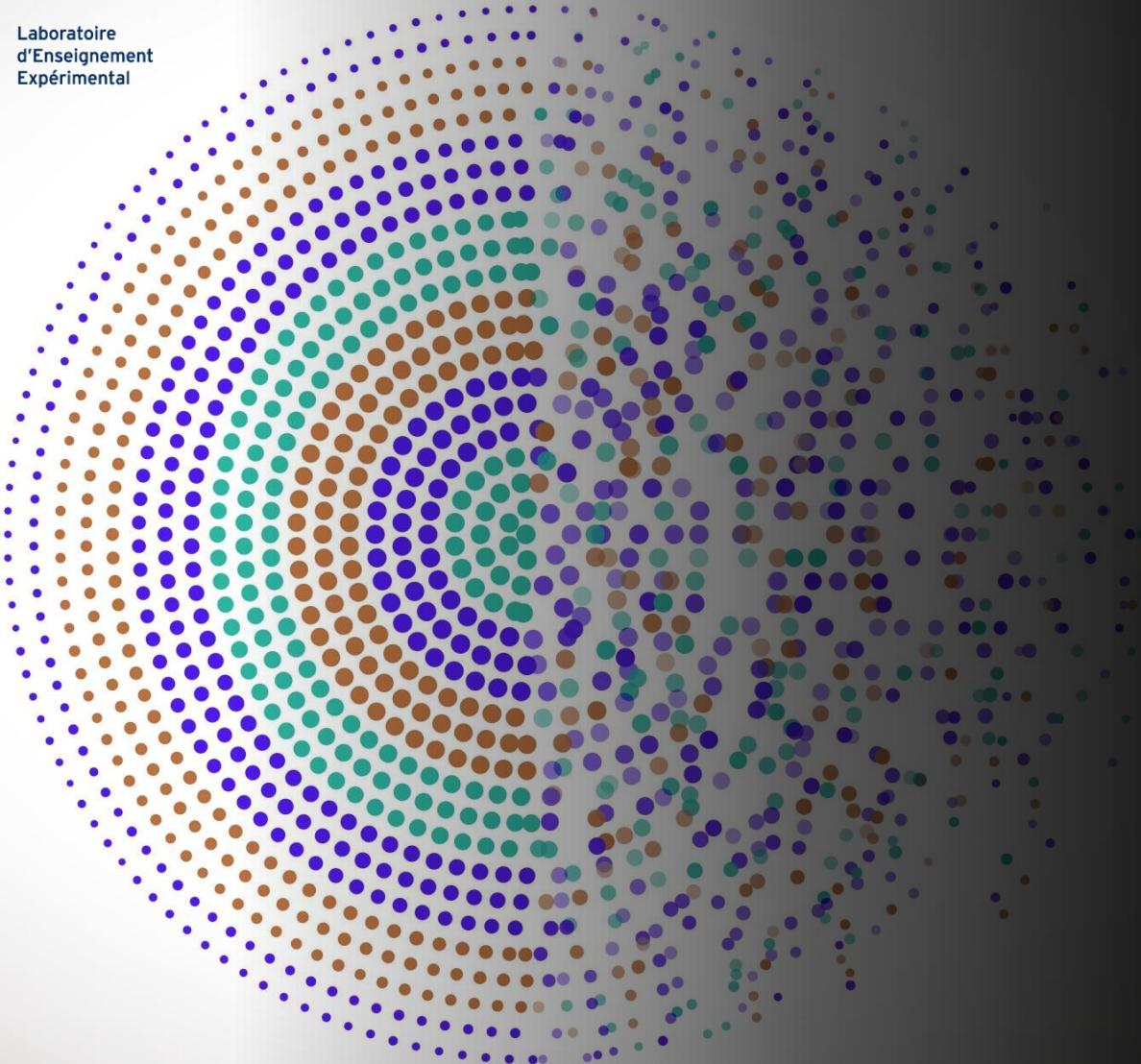
GLOBAL\_CONSTANT\_NAME

global\_var\_name

instance\_var\_name

function\_parameter\_name

local\_var\_name



# POO S'entraîner

---

Outils Numériques / Semestre 6  
/ Institut d'Optique / ONIP-2

# S'entraîner à la POO

A travers les exemples proposés, vous serez capables de :

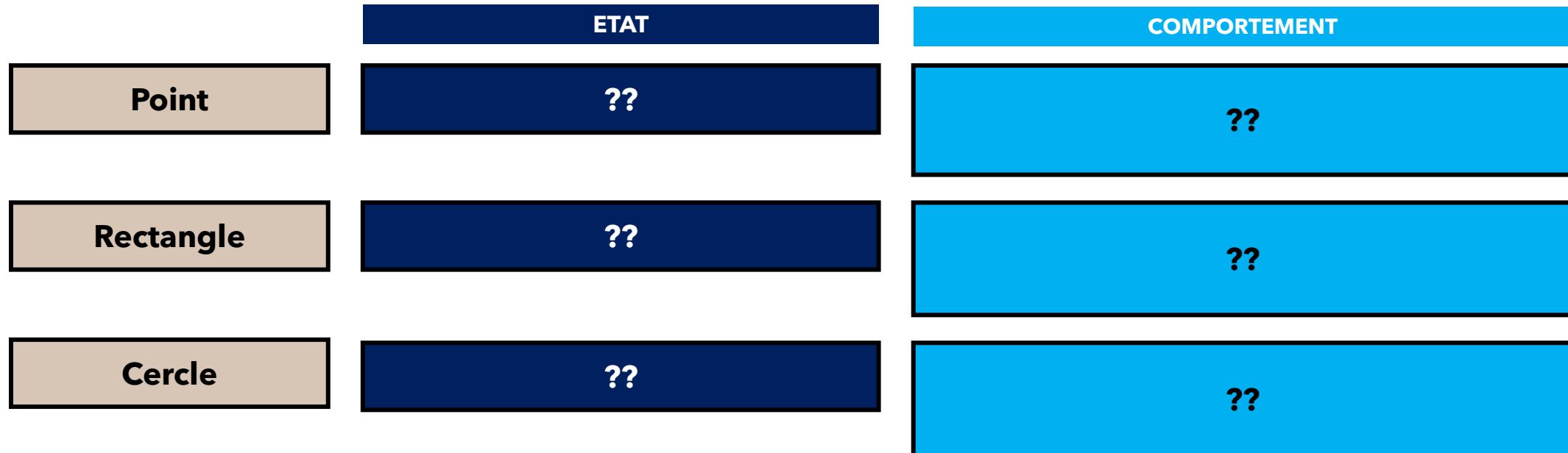
- Créer des **classes** incluant des **méthodes** et des **attributs**
- **Instancier des objets** et les faire interagir
- Définir et **documenter** les méthodes et attributs de chaque classe

Point

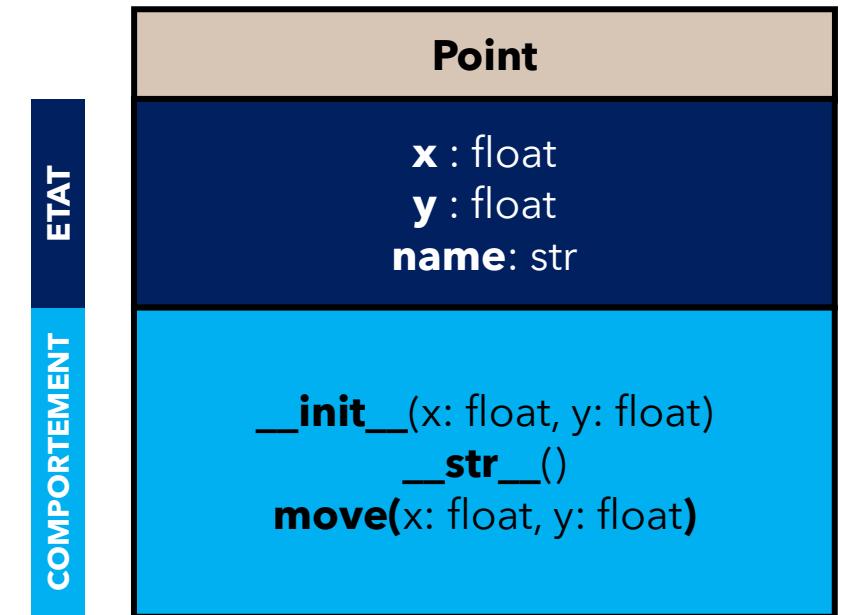
Rectangle

Cercle

# Définir les classes

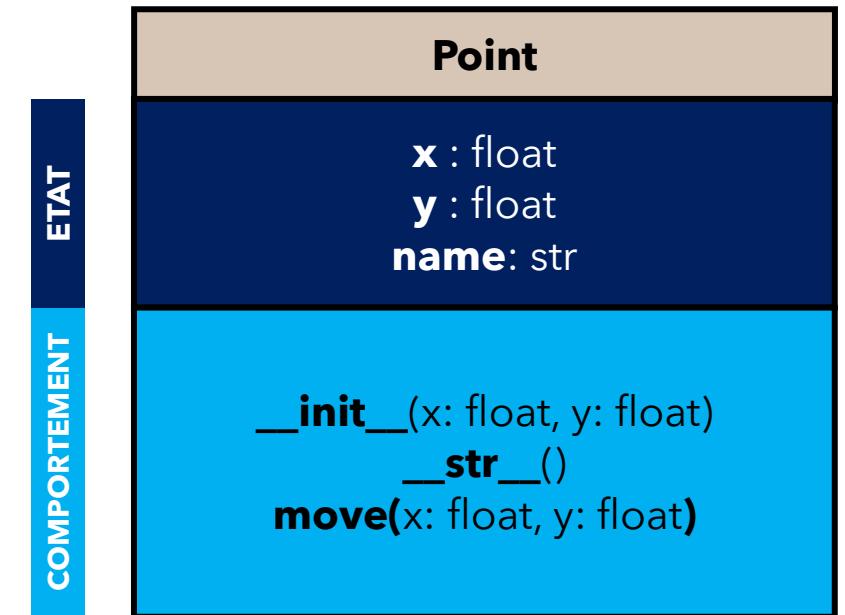


# S'entraîner à la POO



# S'entraîner à la POO

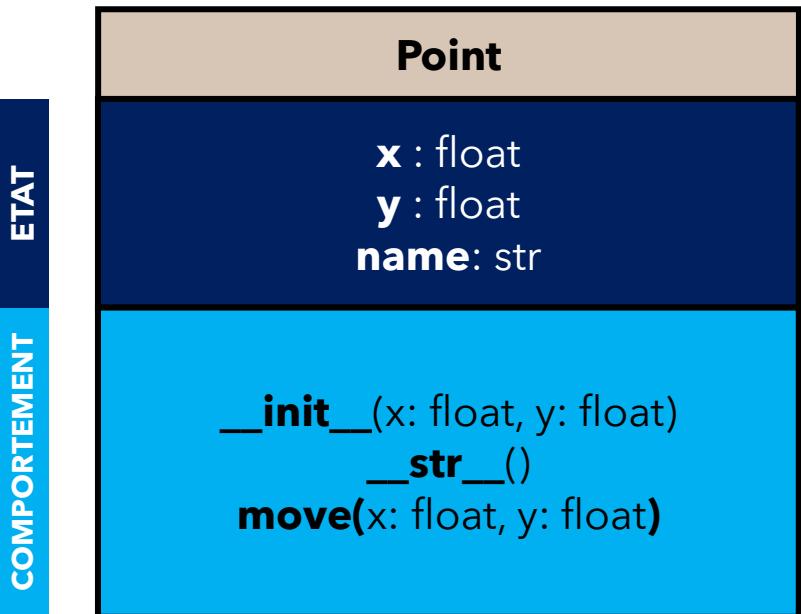
```
class Point:  
    def __init__(self, x:float, y:float, name:str):  
        self.x = x  
        self.y = y  
        self.name = name
```



# S'entraîner à la POO

```
class Point:  
    def __init__(self, x:float, y:float, name:str):  
        self.x = x  
        self.y = y  
        self.name = name
```

```
pointA = Point(-0.5, 5.5, 'A')
```



# S'entraîner à la POO

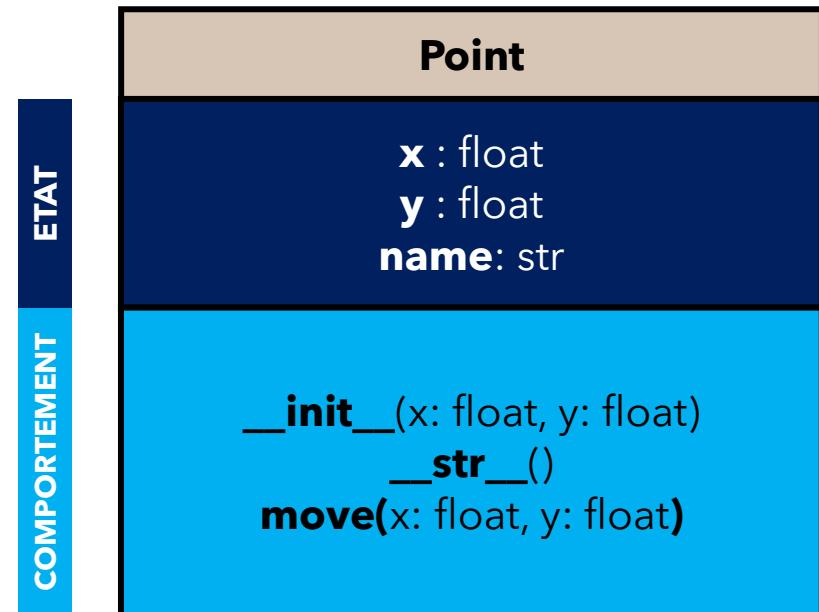
```
class Point:  
    def __init__(self, x:float, y:float, name:str):  
        self.x = x  
        self.y = y  
        self.name = name
```

```
pointA = Point(-0.5, 5.5, 'A')
```

```
def __str__():  
    str = f'p_{self.name}({self.x}, {self.y})'  
    return str
```

```
print(pointA)
```

```
>>> p_A( -0.5, 5.5 )
```



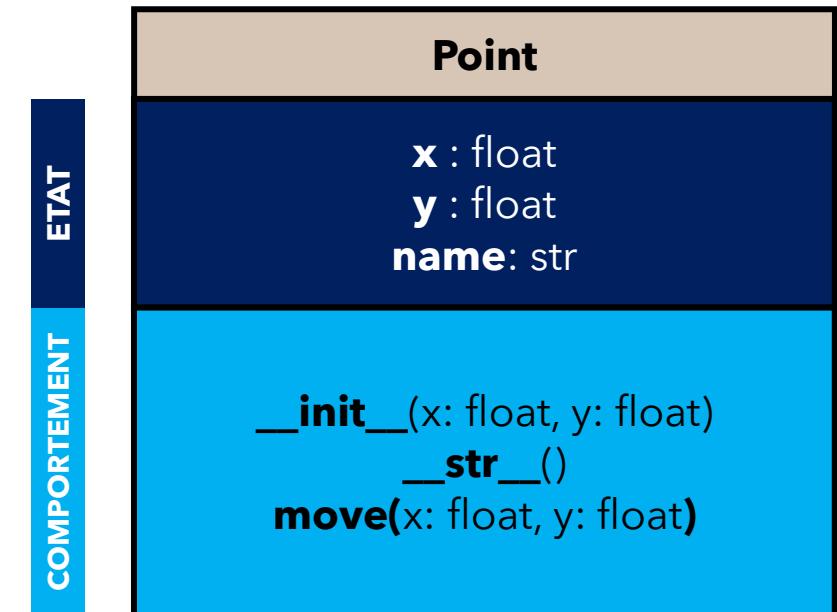
# S'entraîner à la POO

```
class Point:  
    def __init__(self, x:float, y:float, name:str):  
        self.x = x  
        self.y = y  
        self.name = name
```

```
pointA = Point(-0.5, 5.5, 'A')
```

```
def move(self, x:float, y:float):  
    self.x = x  
    self.y = y
```

```
pointA.move(1.0, -2.3)
```

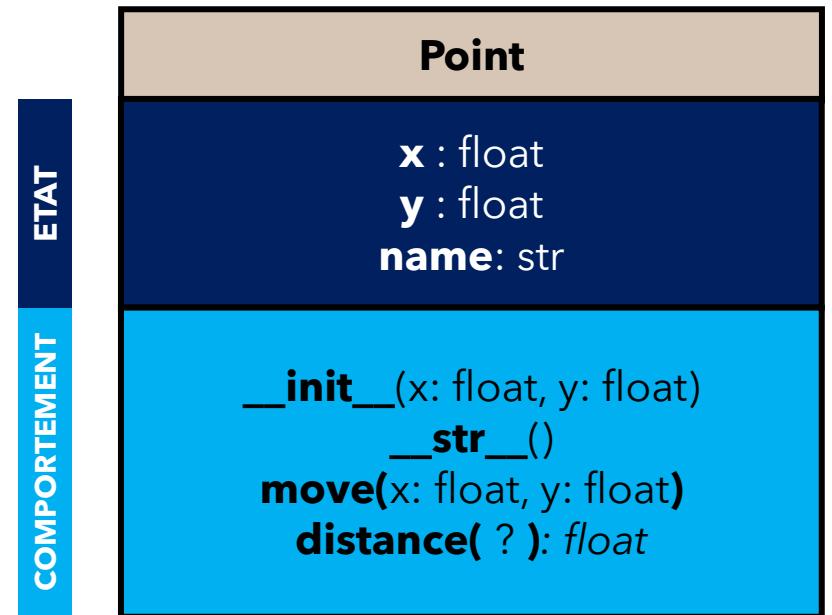


# S'entraîner à la POO

```
class Point:  
    def __init__(self, x:float, y:float, name:str):  
        self.x = x  
        self.y = y  
        self.name = name
```

```
pointA = Point(-0.5, 5.5, 'A')
```

```
def distance(self, ??):  
    ?
```



# S'entraîner à la POO

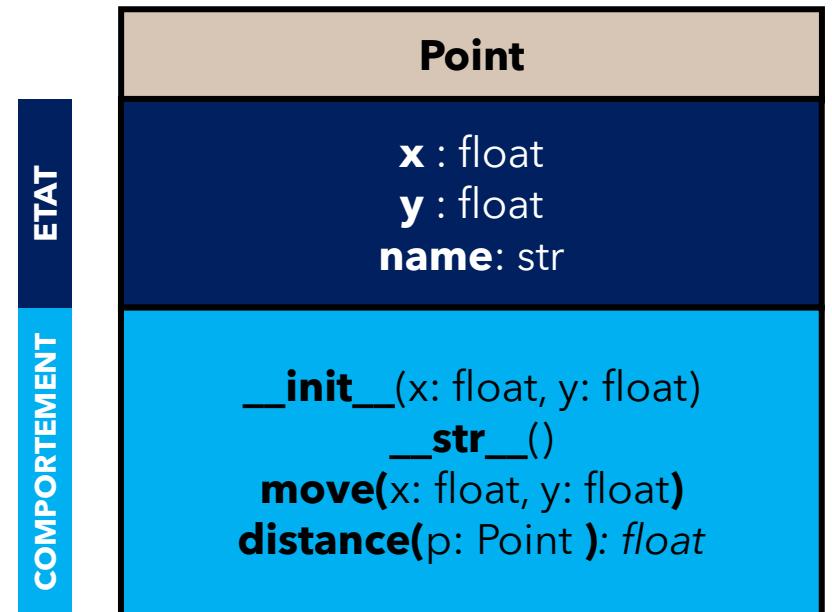
```
class Point:  
    def __init__(self, x:float, y:float, name:str):  
        self.x = x  
        self.y = y  
        self.name = name
```

```
pointA = Point(3, 6, 'A')
```

```
def distance(self, p: Point):  
    dx = self.x - p.x  
    dy = self.y - p.y  
    return np.sqrt(dx**2 + dy**2)
```

```
pointB = Point(0, 10, 'B')  
print( pointA.distance(pointB) )
```

```
>>> 5.0
```



# S'entraîner à la POO

	ETAT	COMPORTEMENT
Point	<b>x</b> : float, <b>y</b> : float, <b>name</b> : str	<b>__init__(x, y)</b> , <b>__str__()</b> <b>move(x, y)</b> , <b>distance(Point p)</b> : float
Rectangle	??	??
Cercle	??	??



# S'entraîner à la POO

	ETAT	COMPORTEMENT
Point	<b>x</b> : float, <b>y</b> : float, <b>name</b> : str	<b>__init__(x, y)</b> , <b>__str__()</b> <b>move(x, y)</b> , <b>distance(Point p)</b> : float
Rectangle	<b>p1</b> : Point, <b>p2</b> : Point, <b>name</b> : str	<b>__init__(x, y)</b> , <b>__str__()</b> <b>perimetre()</b> : float, <b>surface()</b> : float
Cercle	<b>p1</b> : Point, <b>radius</b> : float	<b>__init__(x, y)</b> , <b>__str__()</b> <b>perimetre()</b> : float, <b>surface()</b> : float