

Un monde d'objets

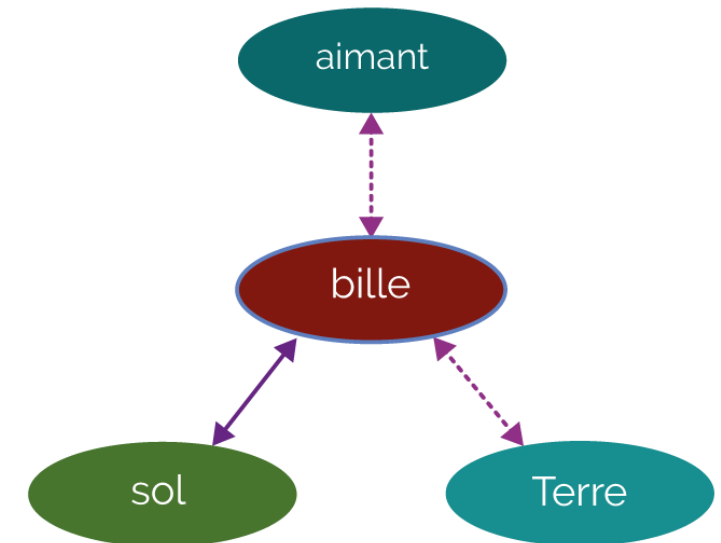
Outils Numériques / Semestre 6
/ Institut d'Optique / ONIP-2

Des objets qui interagissent

https://masevaux.fr/objets_trouves/



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



<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...

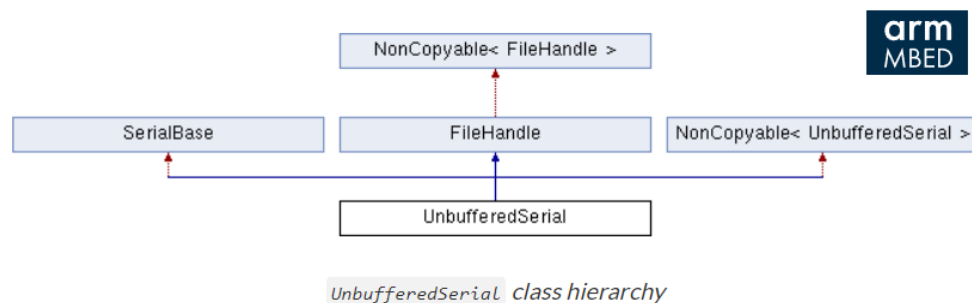
rouler, freiner, klaxonner...

Des objets en informatique

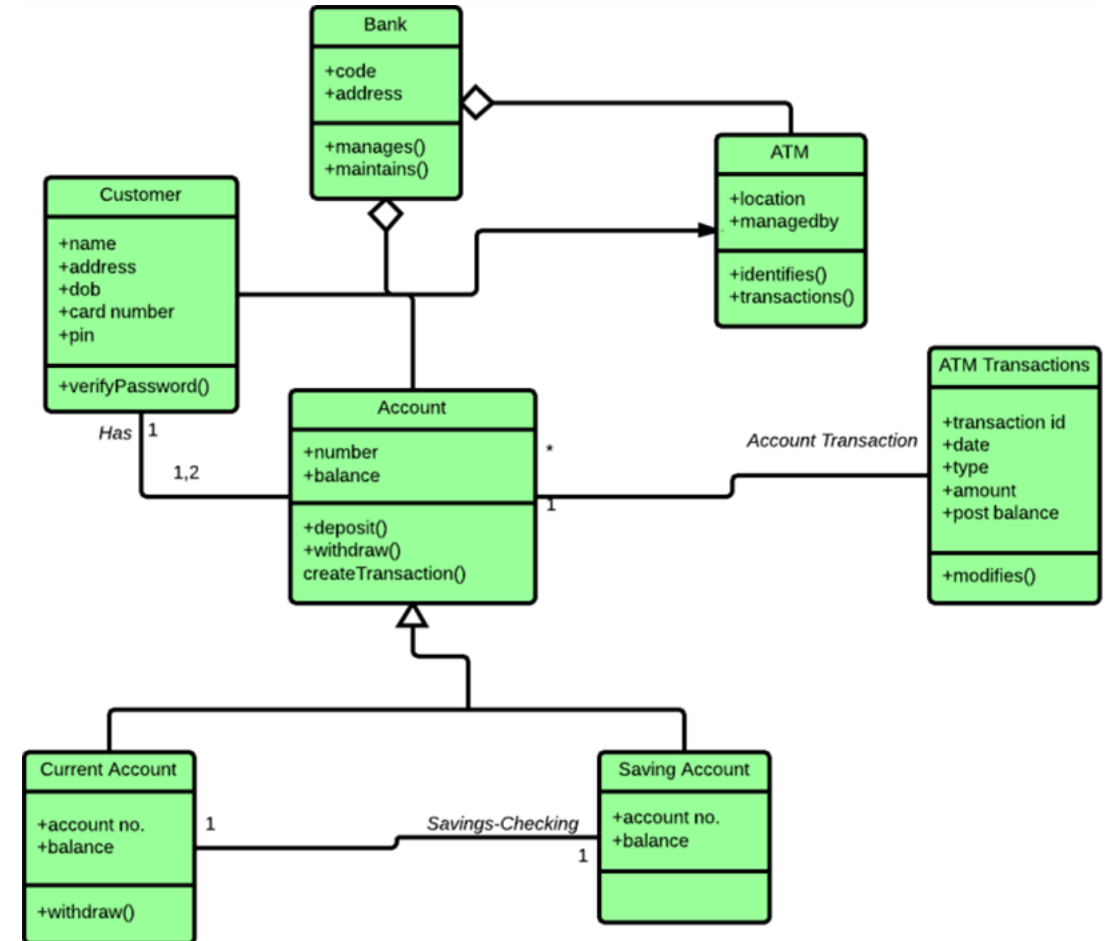
Un **objet** est une **instance** de **classe**, possédant son propre état et son propre comportement

[Docs](#) › [API references and tutorials](#) › [Drivers](#) › [Serial \(UART\) APIs](#) › [UnbufferedSerial](#)

UnbufferedSerial



arm
MBED

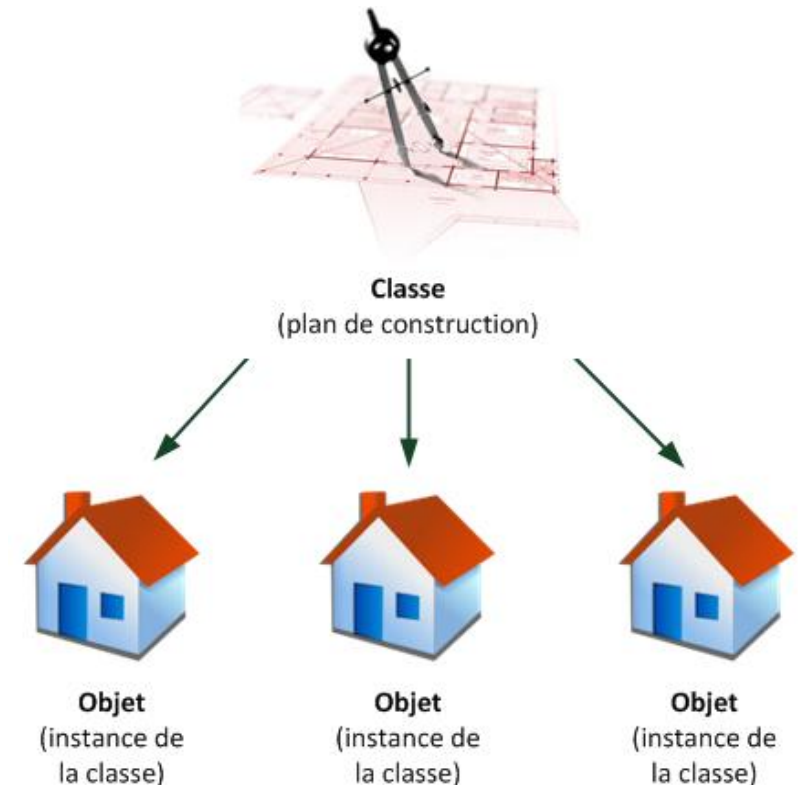


<https://python3.info/design-patterns/uml/class-diagram.html>

Programmation orientée objet

Éléments de base

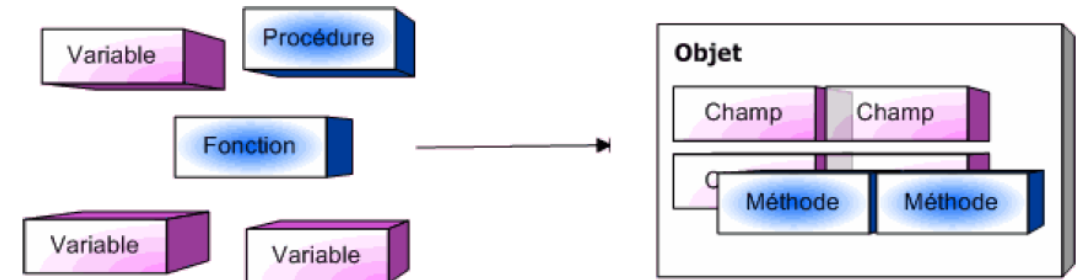
- **Classe** : rassemblement de différents **attributs** (état d'un objet) et **méthodes** (actions possibles d'un objet)
- **Objet** : instance d'une classe



Programmation orientée objet

Concepts fondamentaux

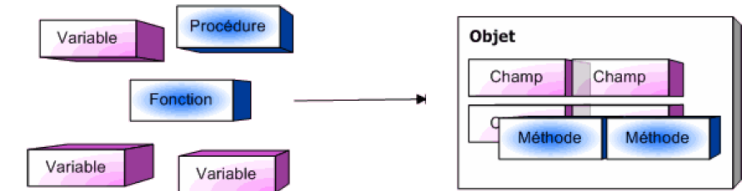
- **Encapsulation** : regroupement de différentes données et fonctions sous une même entité
- **Héritage** : arborescence de classes permettant la spécialisation
(notion non abordée dans ce module)



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)

Méthodes

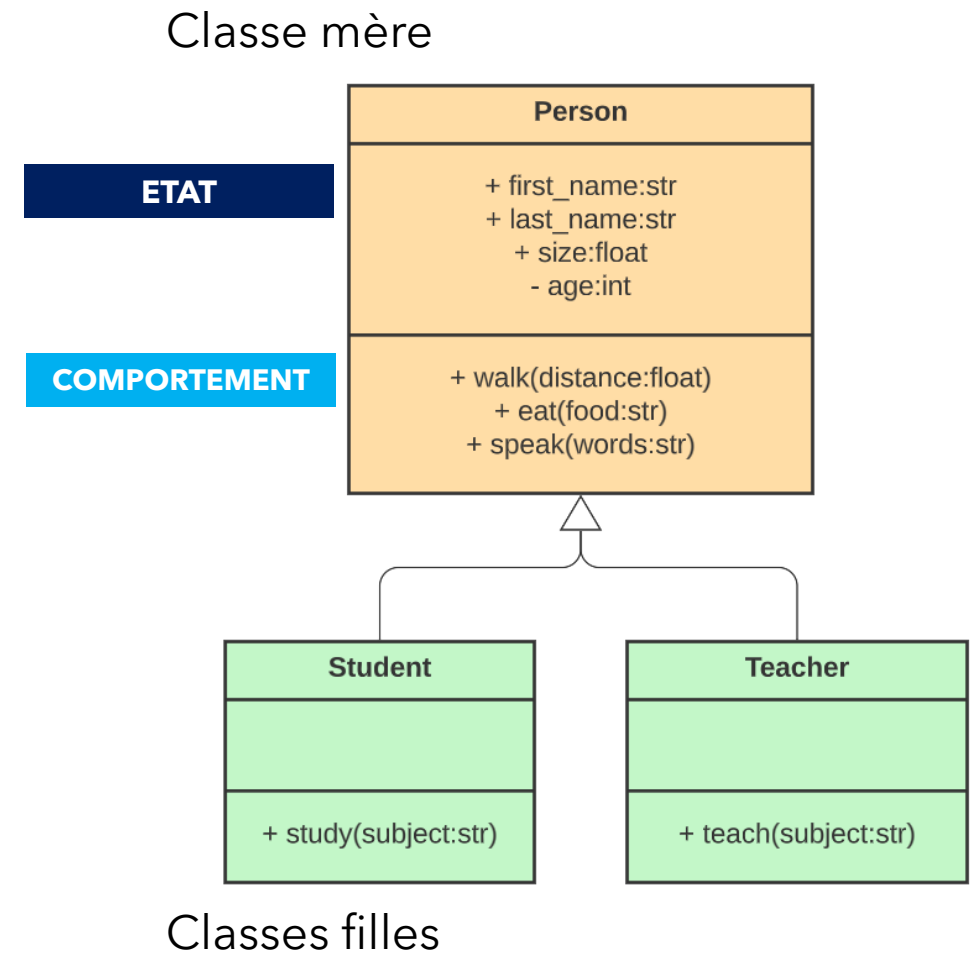
- *max* ([*axis...*])
- *resize* (*new_shape...*)

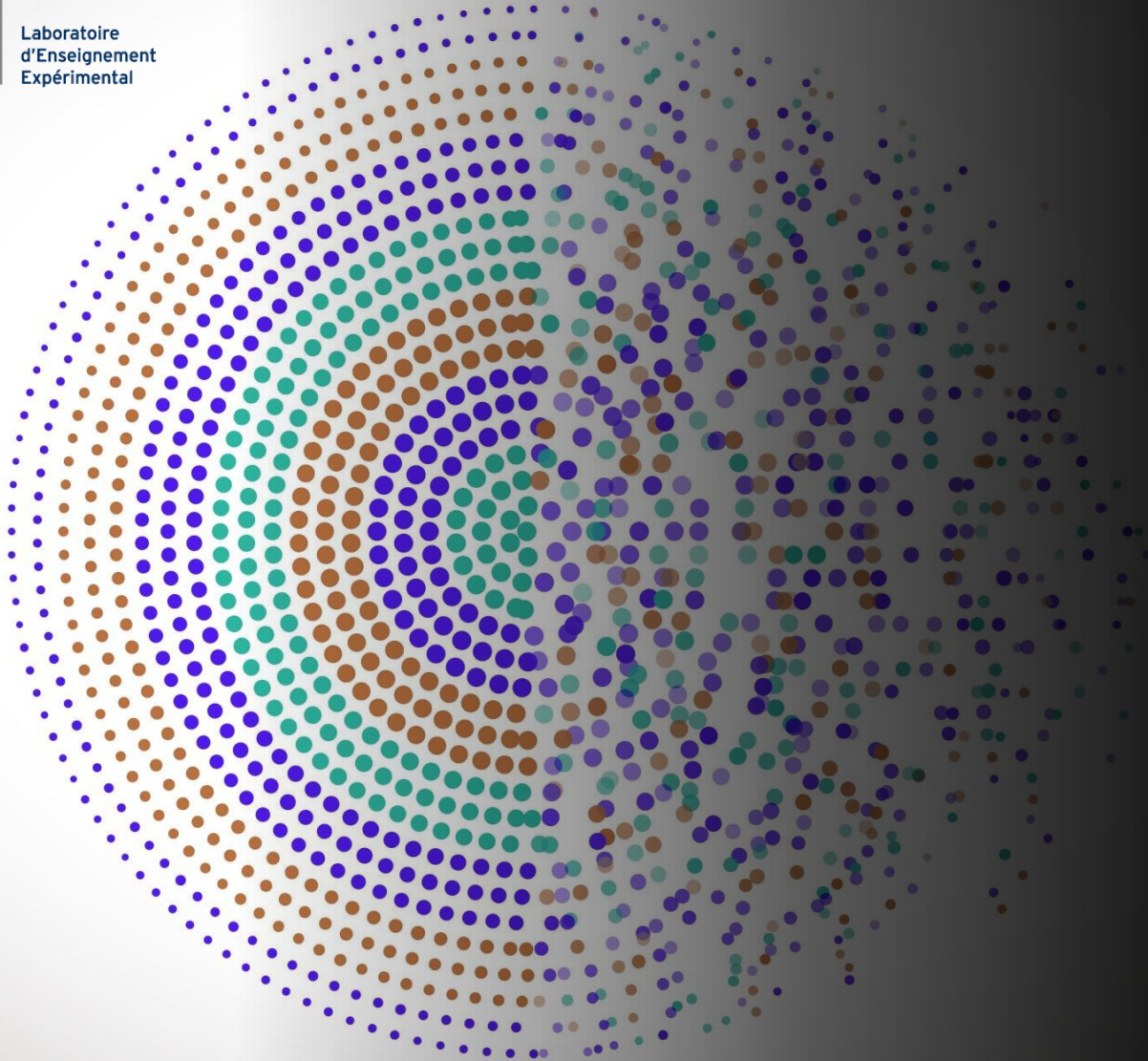
ETAT

COMPORTEMENT

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*)





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
```

ETAT

```
    def move(self):
```

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

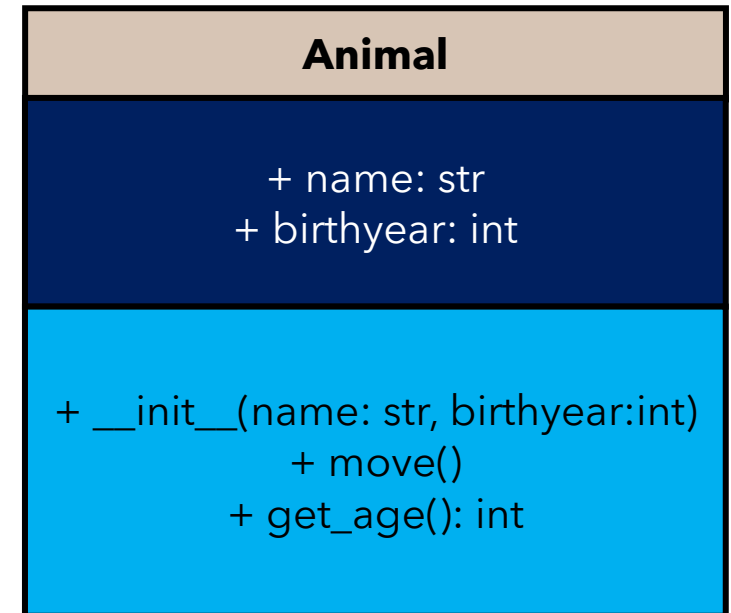
```
    def get_age(self) -> int:
```

```
        return datetime.date.today().year - self.birthyear
```

COMPOTEMENT

ETAT

COMPOTEMENT



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**

COMPOTEMENT

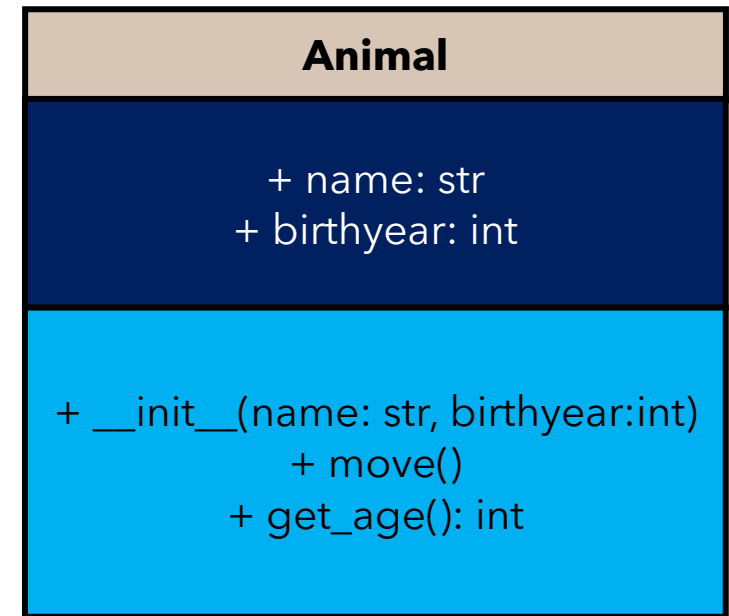
Fonctions associées à un objet (instance d'une classe), nommées **méthodes**

`__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

COMPOTEMENT



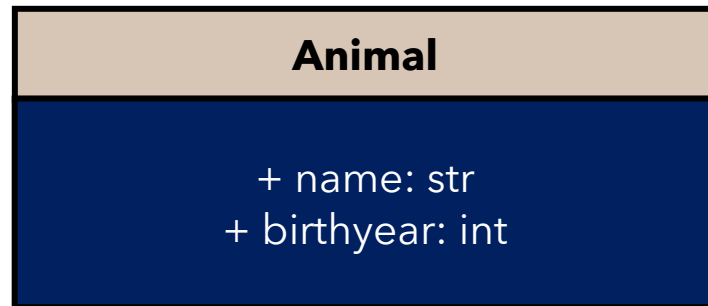
`self` est le mot clé utilisé pour **accéder aux méthodes et attributs d'instance**

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)
```

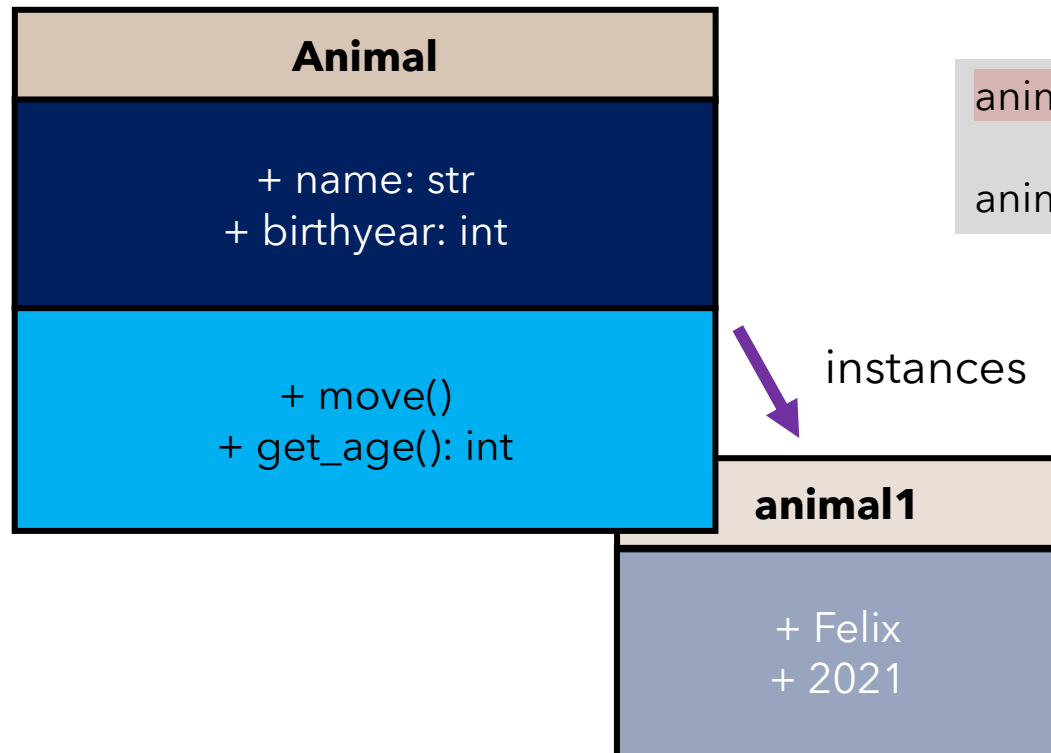
instances



Utilisation d'une classe

Accès aux attributs d'un objet

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



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

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

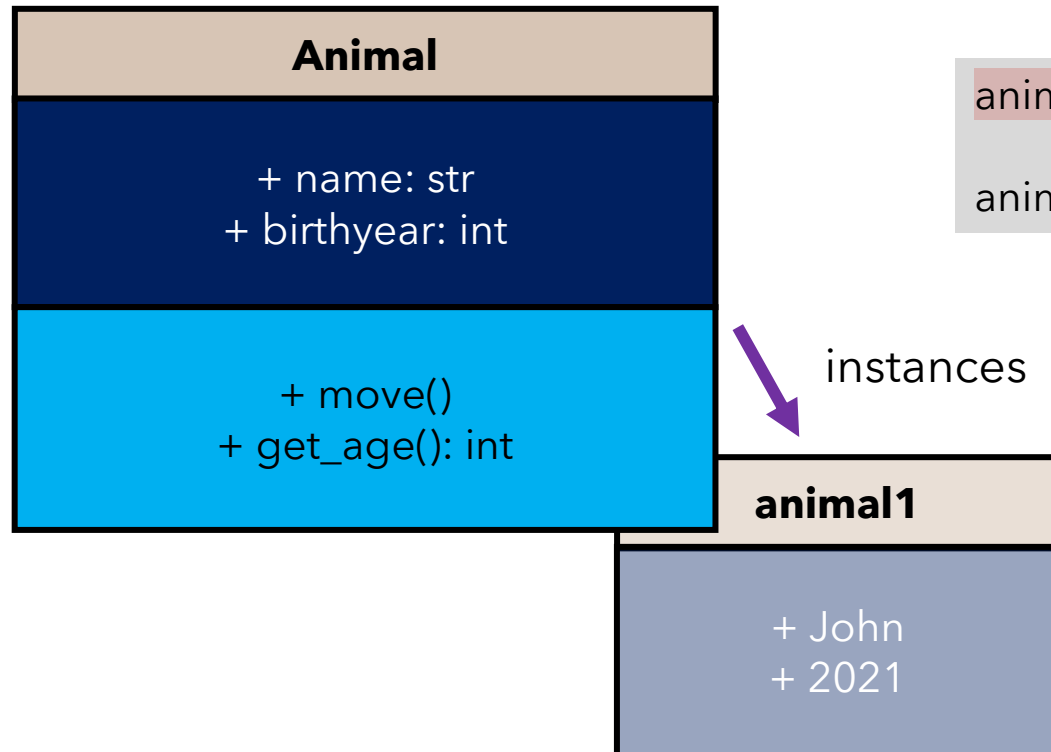
```
print("Animal 1 Name = ", animal1.name)
```

```
>>> 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
```



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

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

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

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

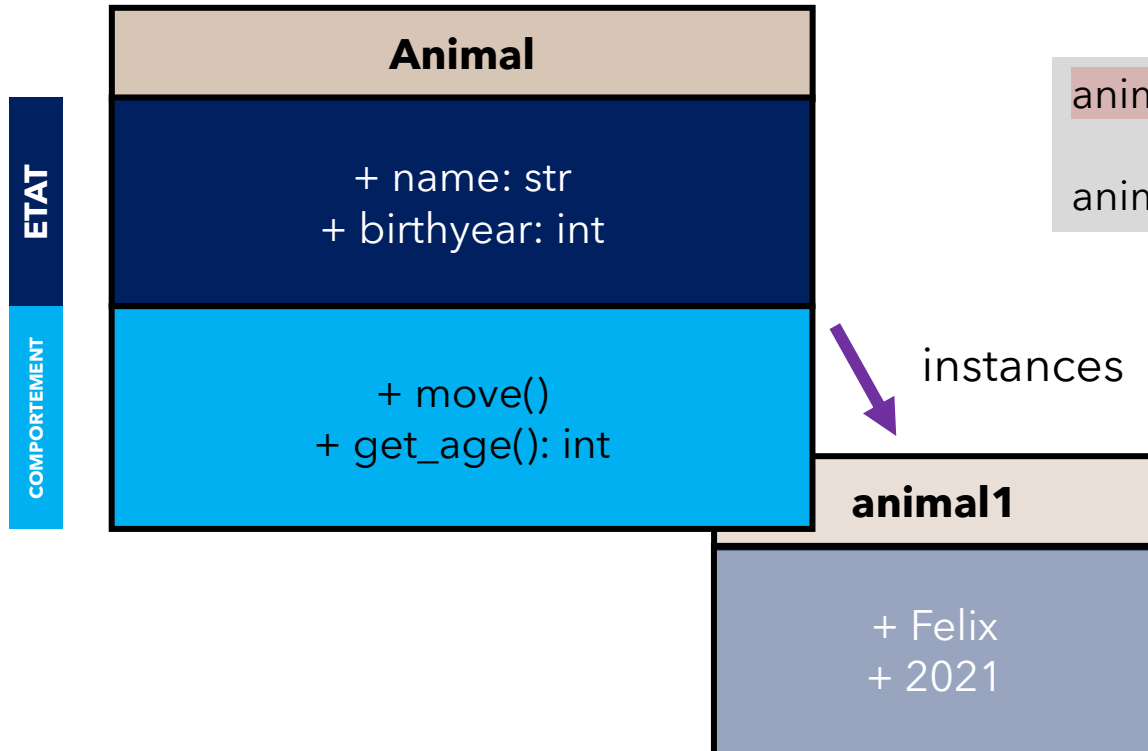
```
animal1.name = "John"
print("Animal 1 Name = ", animal1.name)
```

```
>>> 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")
```



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

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

```
animal1.move()
```

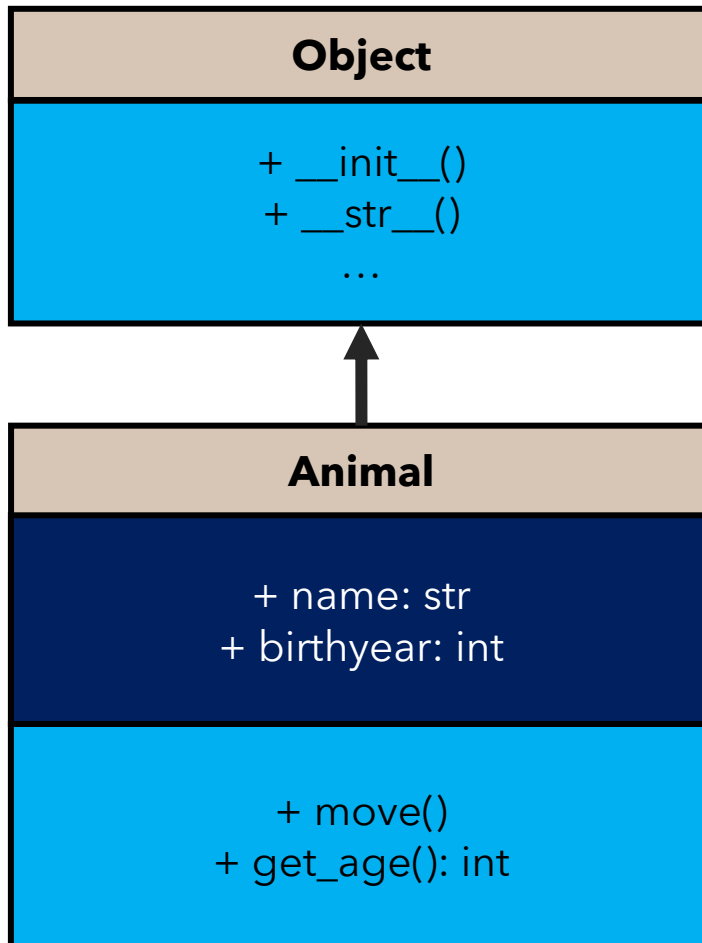
```
>>> [ Felix ] is moving
```

```
animal2.move()
```

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

Objets en Python

Gestion des objets



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

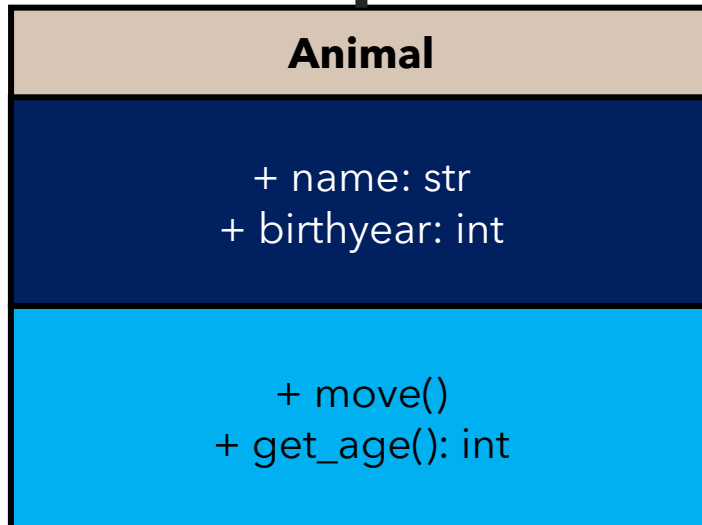
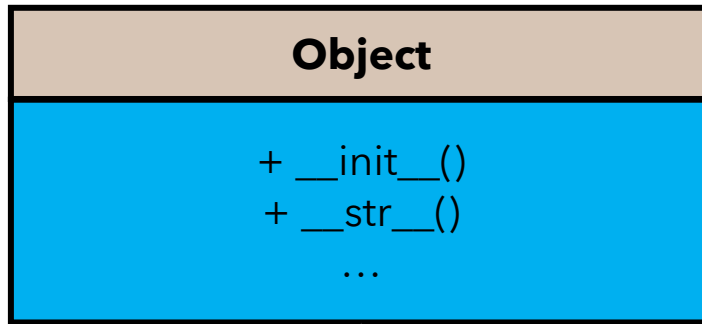
```
print(animal1)
```

```
>>> <__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
```

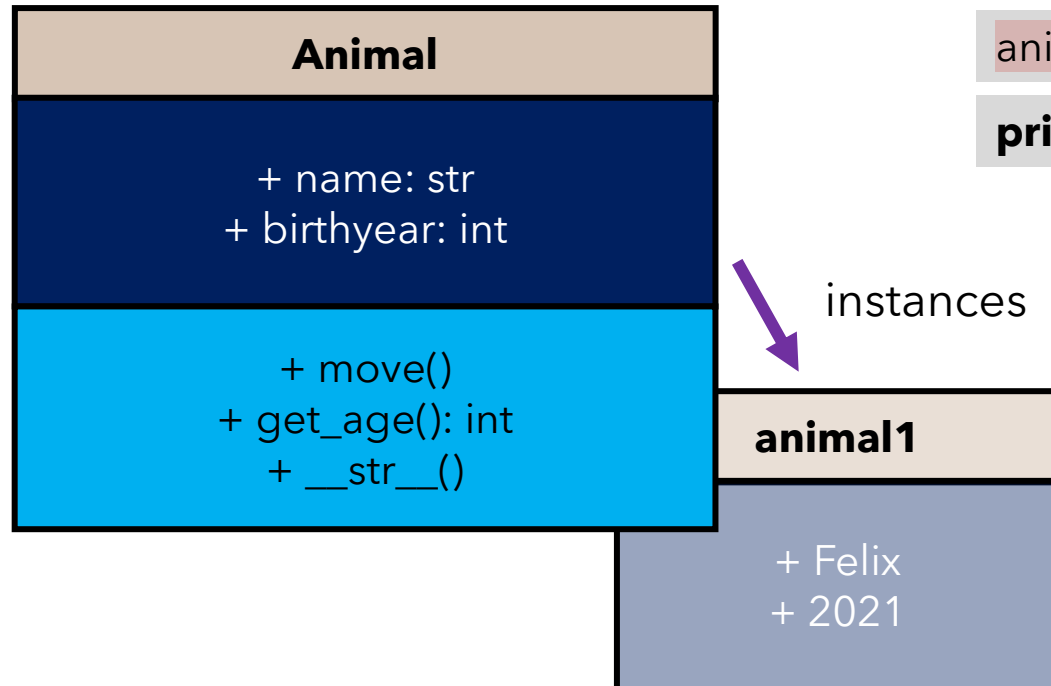

Utilisation d'une classe

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
```

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

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



```
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

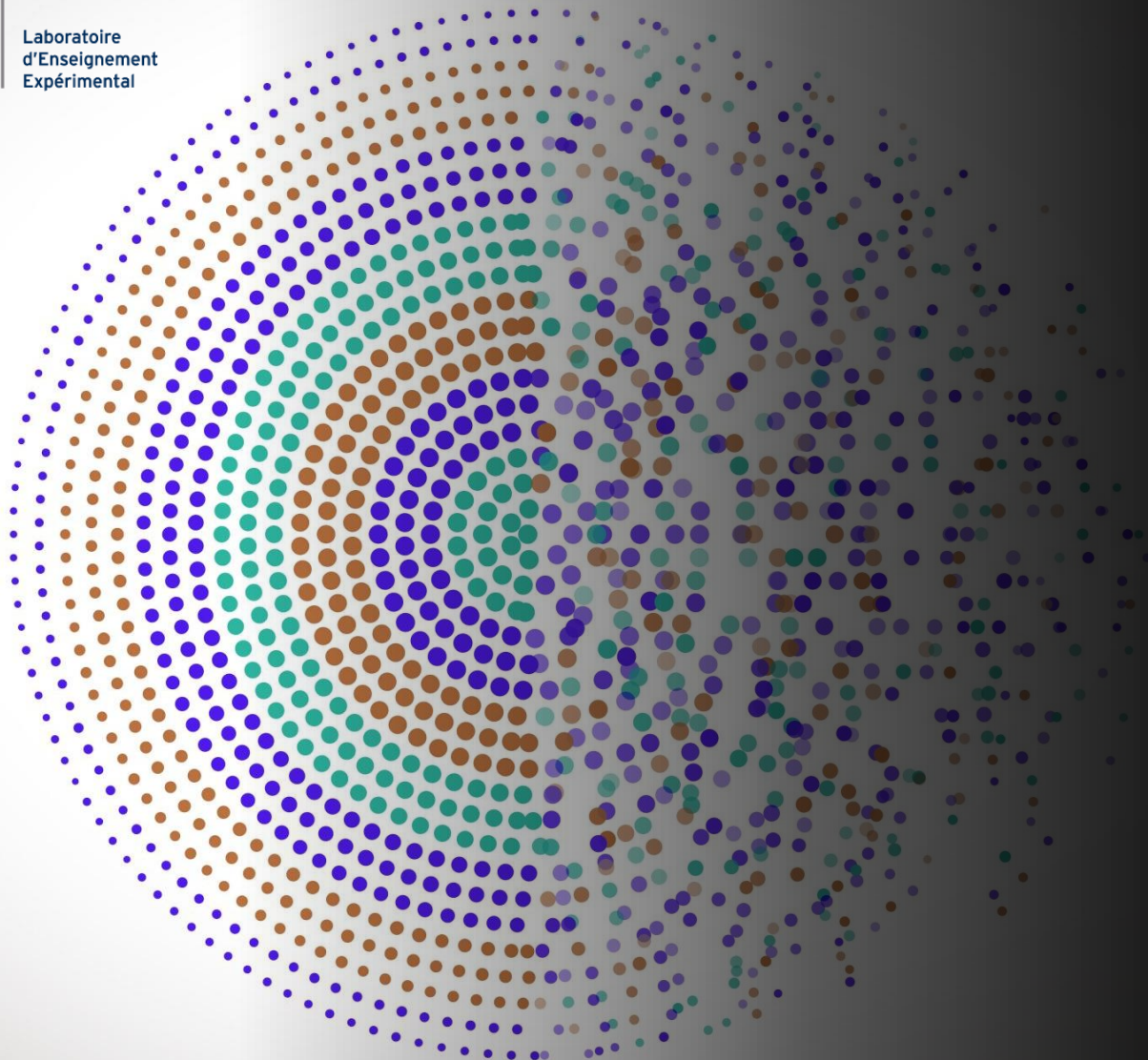
GLOBAL_CONSTANT_NAME

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

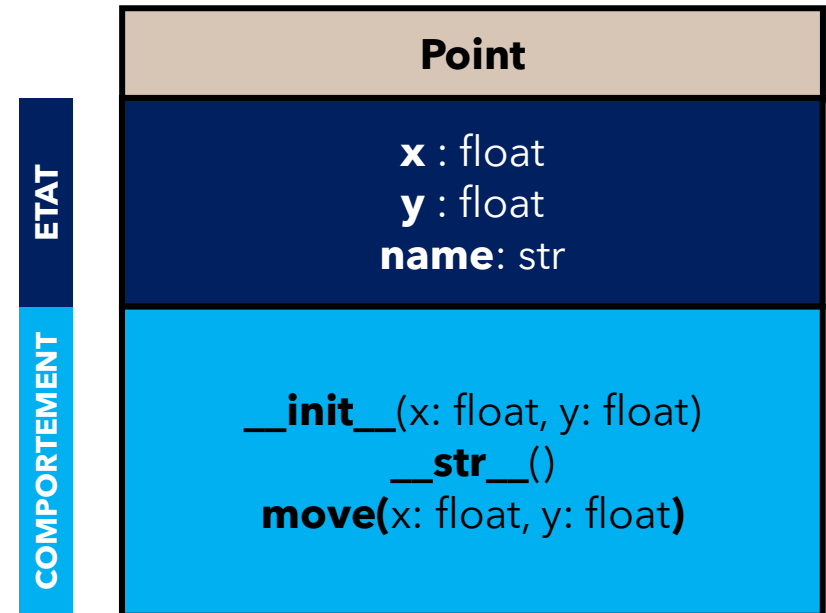
Rectangle

Cercle

Définir les classes

	ETAT	COMPORTEMENT
Point	??	??
Rectangle	??	??
Cercle	??	??

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
```

ETAT

COMPOTEMENT

Point

x : float
y : float
name: str

__init__(x: float, y: float)
__str__()
move(x: float, y: float)

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')
```

ETAT

COMPOTEMENT

Point

x : float
y : float
name: str

__init__(x: float, y: float)
__str__()
move(x: float, y: float)

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')
```

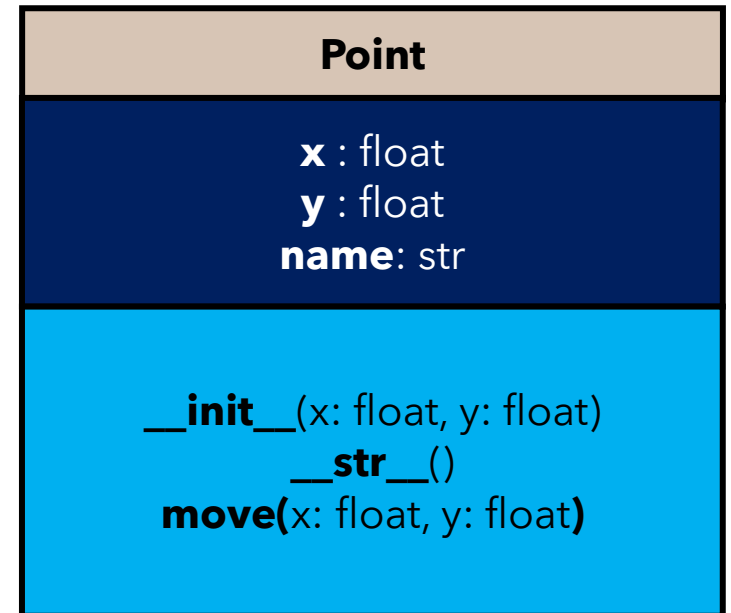
```
def __str__(self):
    str = f'p_{self.name} ( {self.x}, {self.y} ) '
    return str
```

```
print(pointA)
```

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

ETAT

COMPOTEMENT



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
```

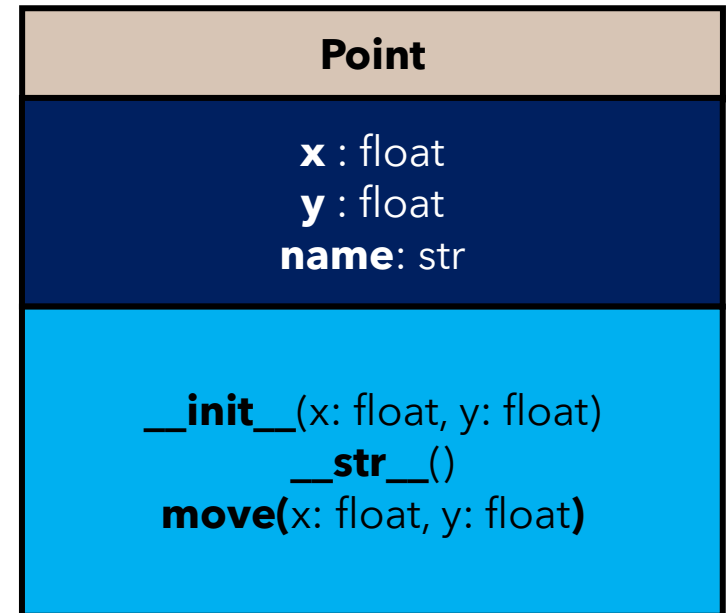
```
pointA = Point(-0.5, 5.5, 'A')
```

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

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

ETAT

COMPOTEMENT



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, ??):  
    ?
```

ETAT

COMPOTEMENT

Point

x : float
y : float
name: str

__init__(x: float, y: float)
__str__()
move(x: float, y: float)
distance(?): float

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')
```

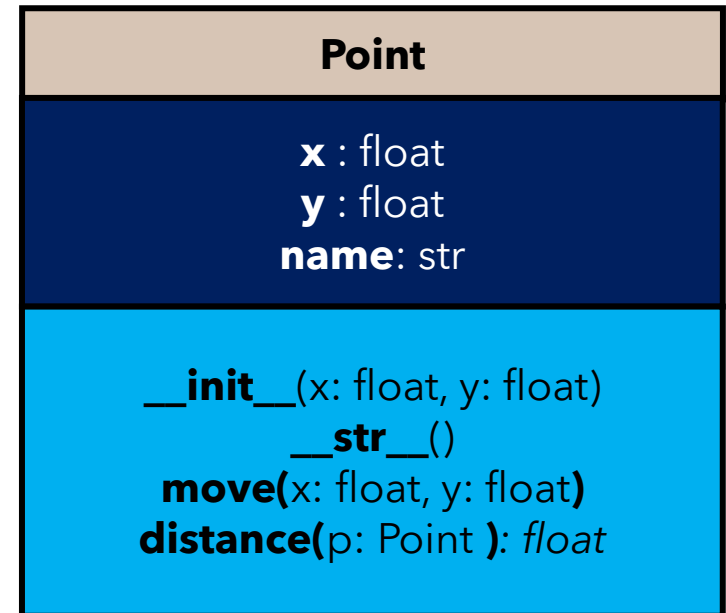
```
def distance(self, p: Point):  
    dx = self.x - p.x  
    dy = self.y - p.y  
    return np.sqrt( dx**2 + dy**2 )
```

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

```
>>> 5.0
```

ETAT

COMPOTEMENT



S'entraîner à la POO

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

S'entraîner à la POO

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