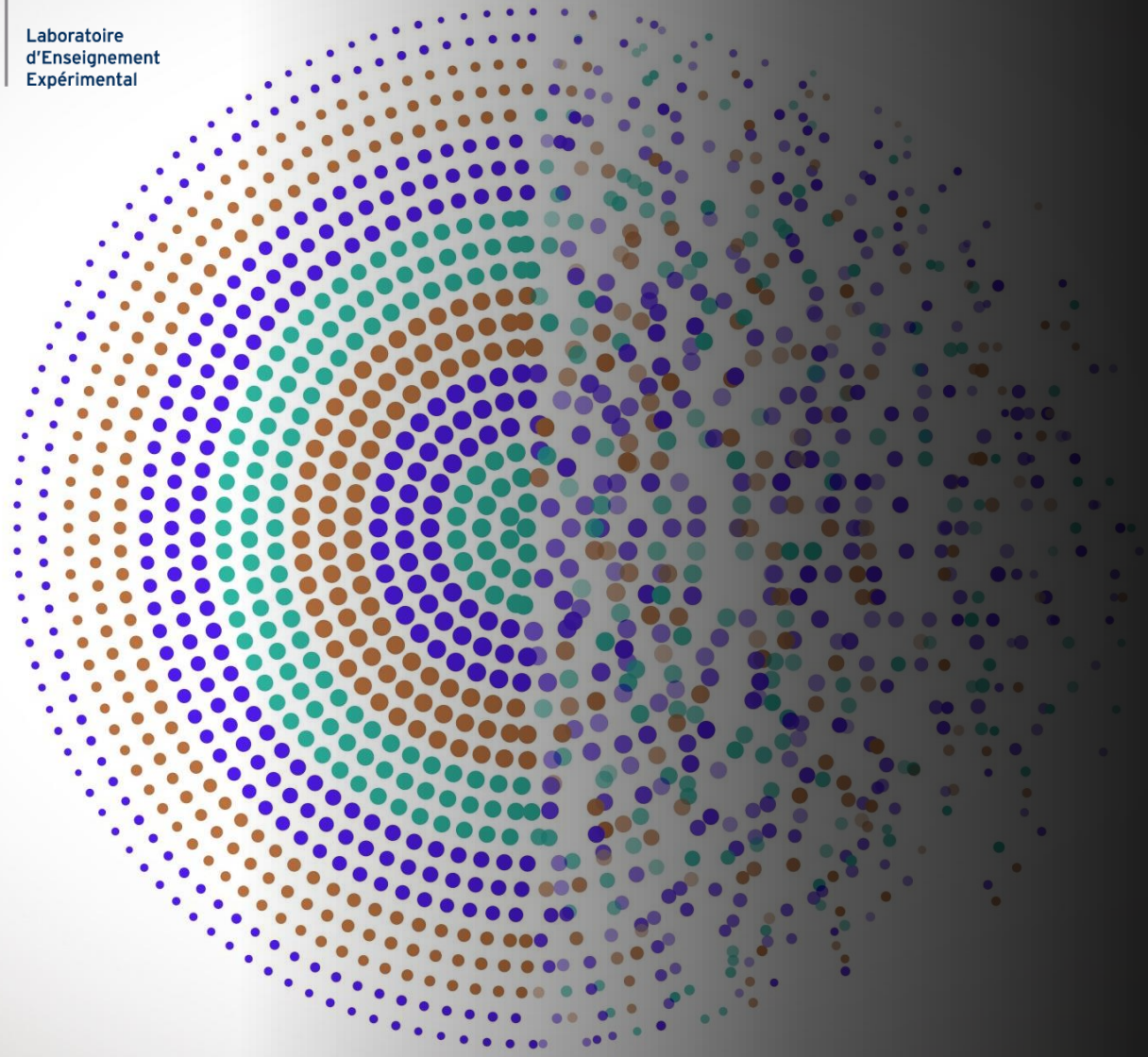
INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental



ONIP-2

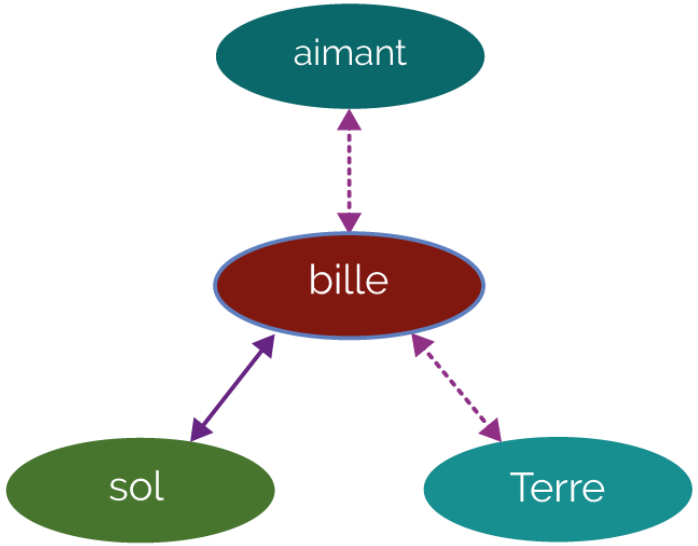
Programmation Orientée Objet

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

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

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…

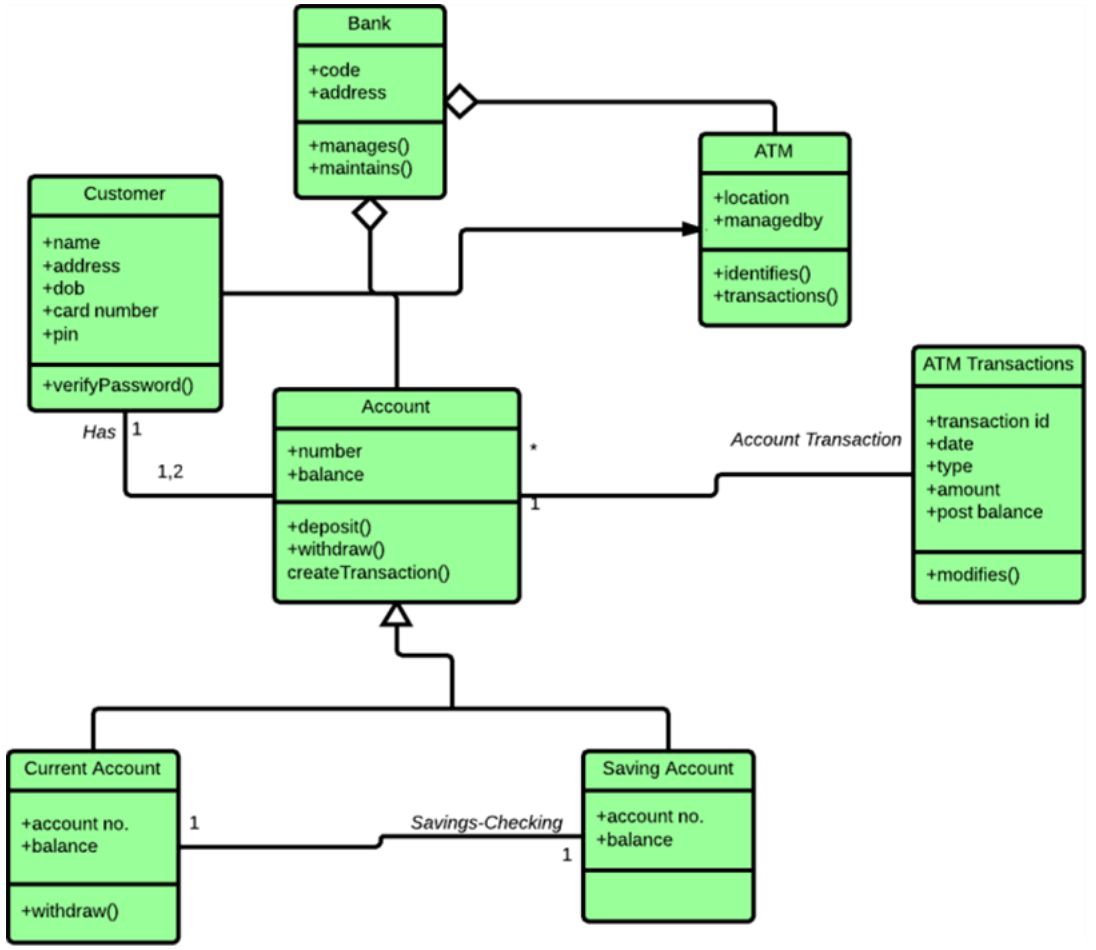
TRAIN

marque, type, vitesse max…

rouler, freiner, klaxonner…

Des objets en informatique

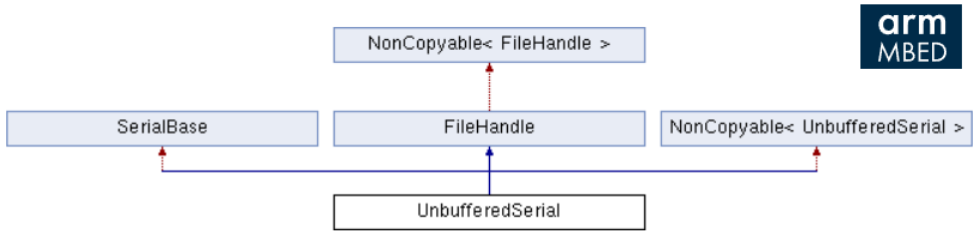
Un objet est une **instance** de **classe**, possédant son propre état et son propre comportement



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

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

UnbufferedSerial

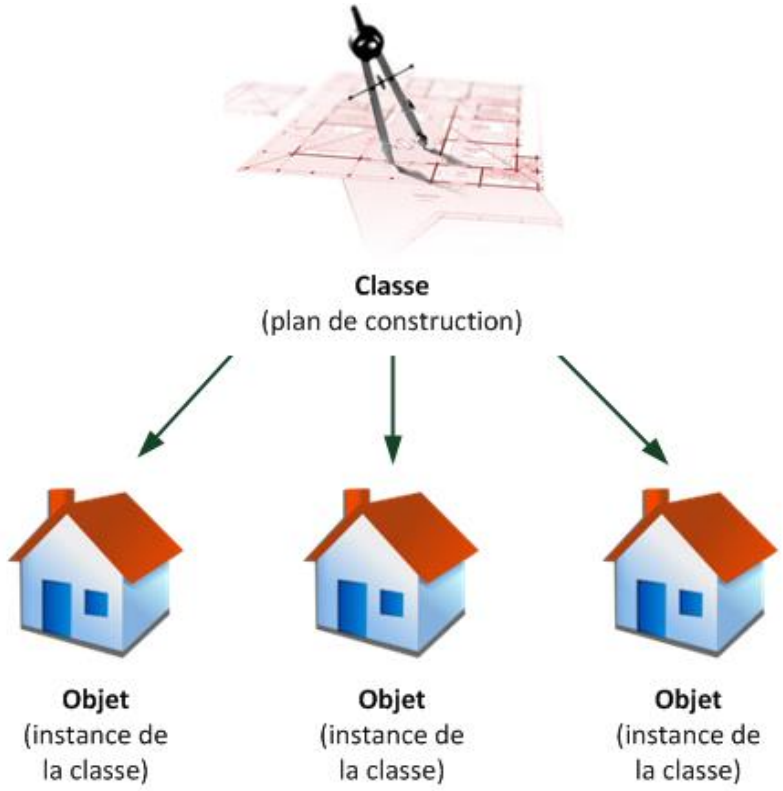


`UnbufferedSerial` *class hierarchy*

Programmation orientée objet

Eléments de base

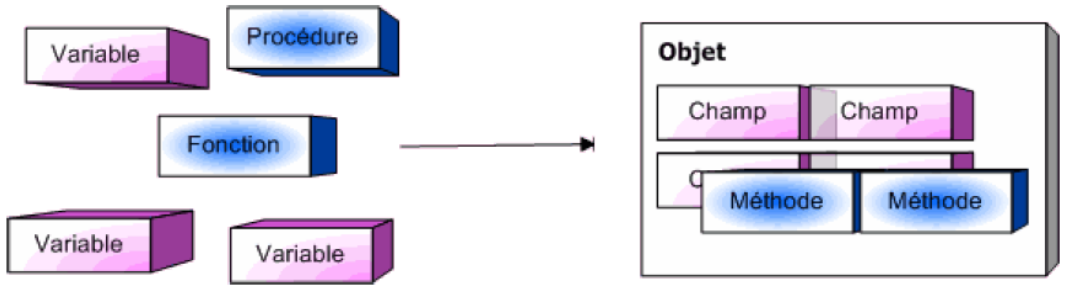
- **Classe** : rassemblement de différents **attributs** (état d'un objet) et **méthodes** (actions possibles d'un objet)
- **Objet** : instance d'une classe



Programmation orientée objet

Concepts fondamentaux

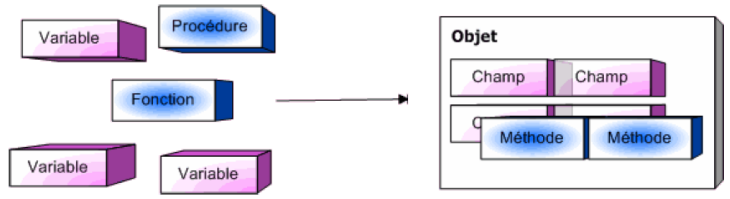
- **Encapsulation** : regroupement de différentes données et fonctions sous une même entité
- **Héritage** : arborescence de classes permettant la spécialisation
  (notion non abordée dans ce module)



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)*

ETAT

Méthodes

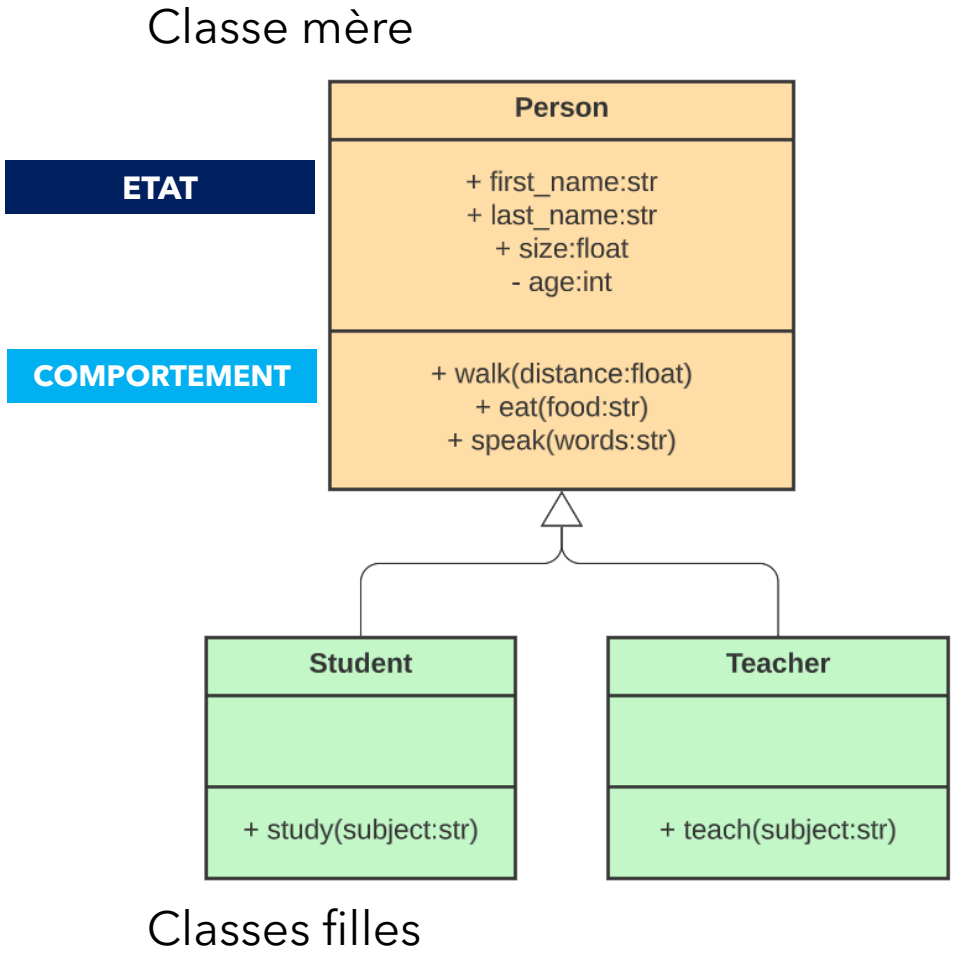
- *max ([axis…])*
- *resize (new_shape…)*

COMPORTEMENT

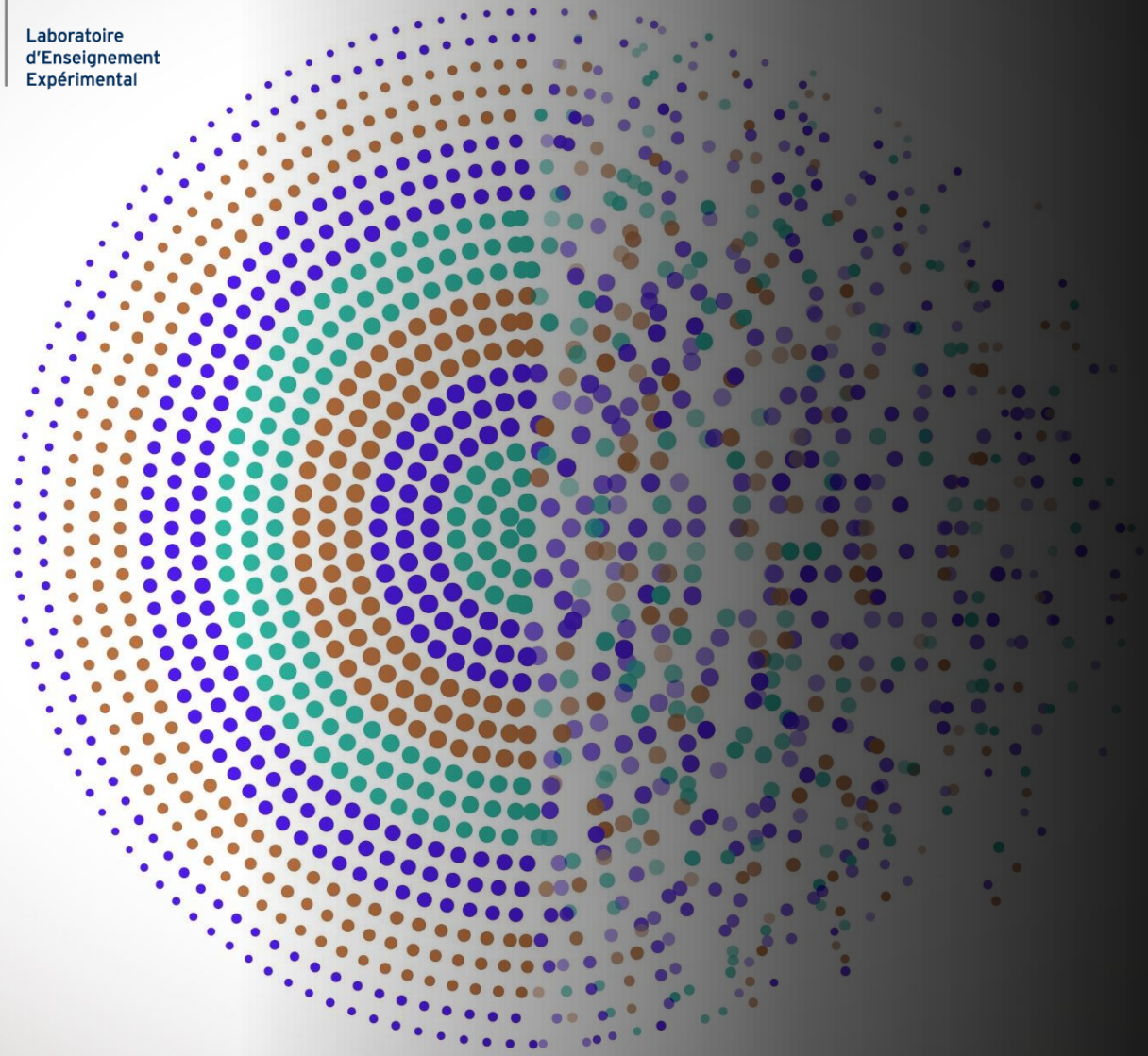
Programmation orientée objet

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)



INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental



POO en Python

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

animal_1.py

Exemple d'une classe

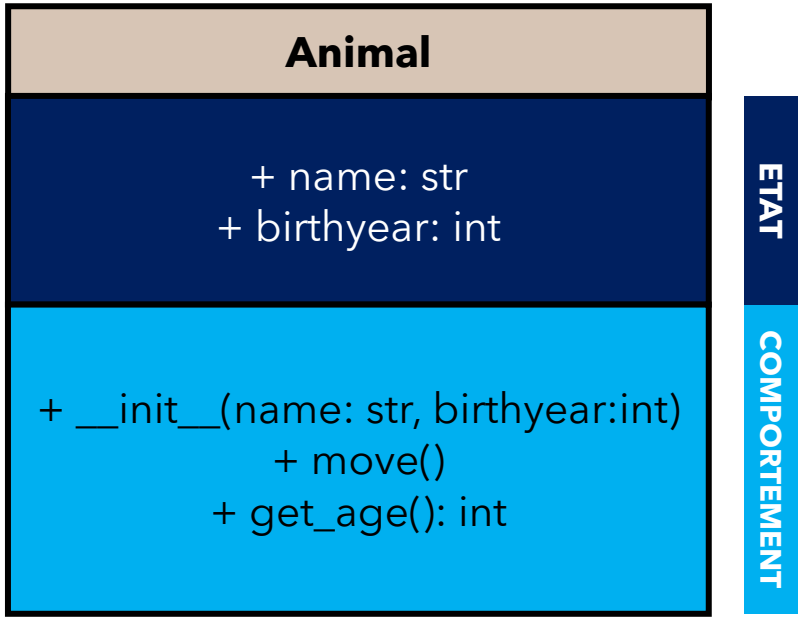
Encapsulation : regroupement de différentes données et fonctions sous une même entité

```python
import datetime

class Animal:
    """ object class Animal
    """
    def __init__(self, name:str="Hello", birthyear:int=2000):
        """ Animal class constructor
        :name: name of the animal
        :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
```



Exemple d'une classe

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

ETAT

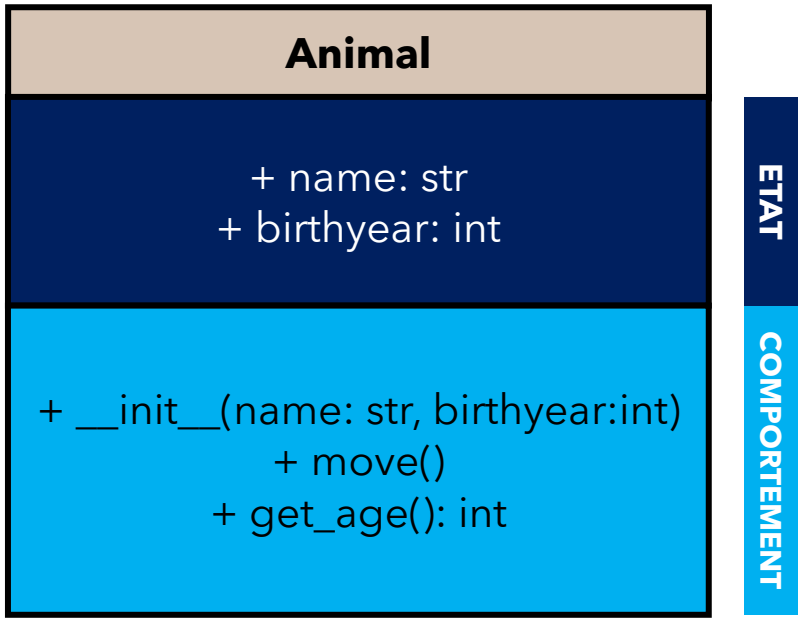
variables, *propres à un objet (instance d'une classe), nommées* ***attributs***

COMPORTEMENT

Méthodes *associées à un objet (instance d'une classe), nommées* ***attributs***

__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*



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

Utilisation d'une classe

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

```
# Test of the class Animal
if __name__ == '__main__':
    animal1 = Animal()
    print("Animal 1 Name = ", animal1.name)
    animal2 = Animal("Garfield", 2015)
    print("Animal 2 Name = ", animal2.name)

    print(animal1)

    print(f"Animal 2 is {animal2.get_age()} years old")
```

Animal
+ name: str + birthyear: int
+ __init__(name: str, birthyear:int) + move() + get_age(): int

```
Animal 1 Name =  Hello
Animal 2 Name =  Garfield
<__main__.Animal object at 0x0000020C594D2F10>
Animal 2 is 10 years old
```

Utilisation d'une classe

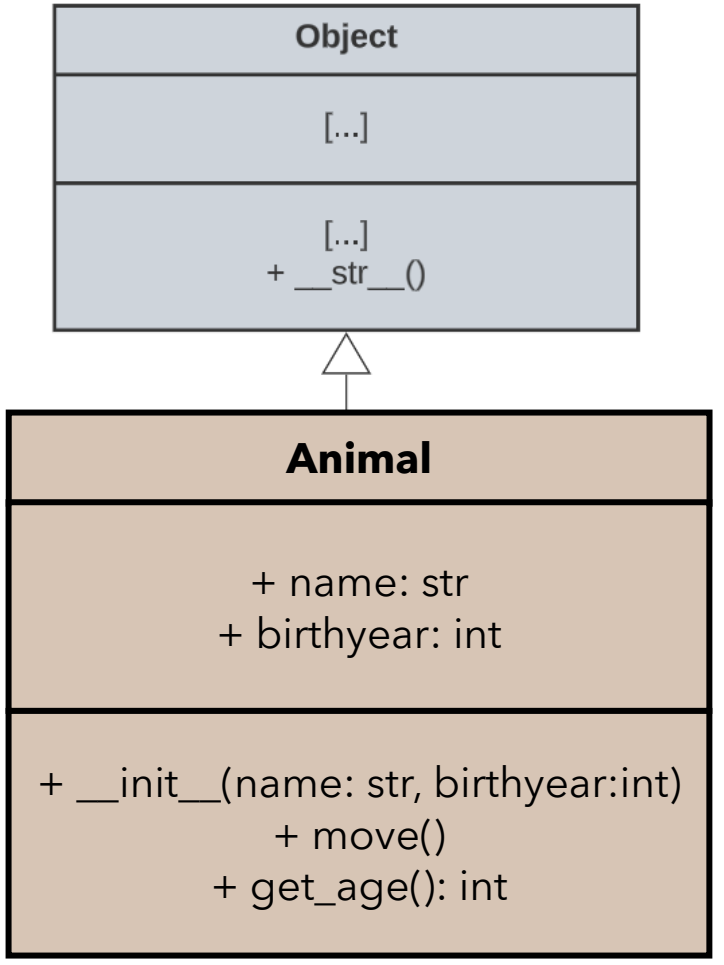
Redéfinition : définir une méthode déjà existante dans une classe mère pour spécialiser cette nouvelle classe

```
class Animal:
    """ object class Animal
    """
    [...]

    def __str__(self):
        """ Animal class display
        """
        return f"Animal [ {self.name} ] born in {self.birthyear}"

    [...]
```

```
Animal 1 Name =  Hello
Animal 2 Name =  Garfield
Animal [ Hello ] born in 2000
Animal 2 is 10 years old
```



Programmation orientée objet

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental

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:

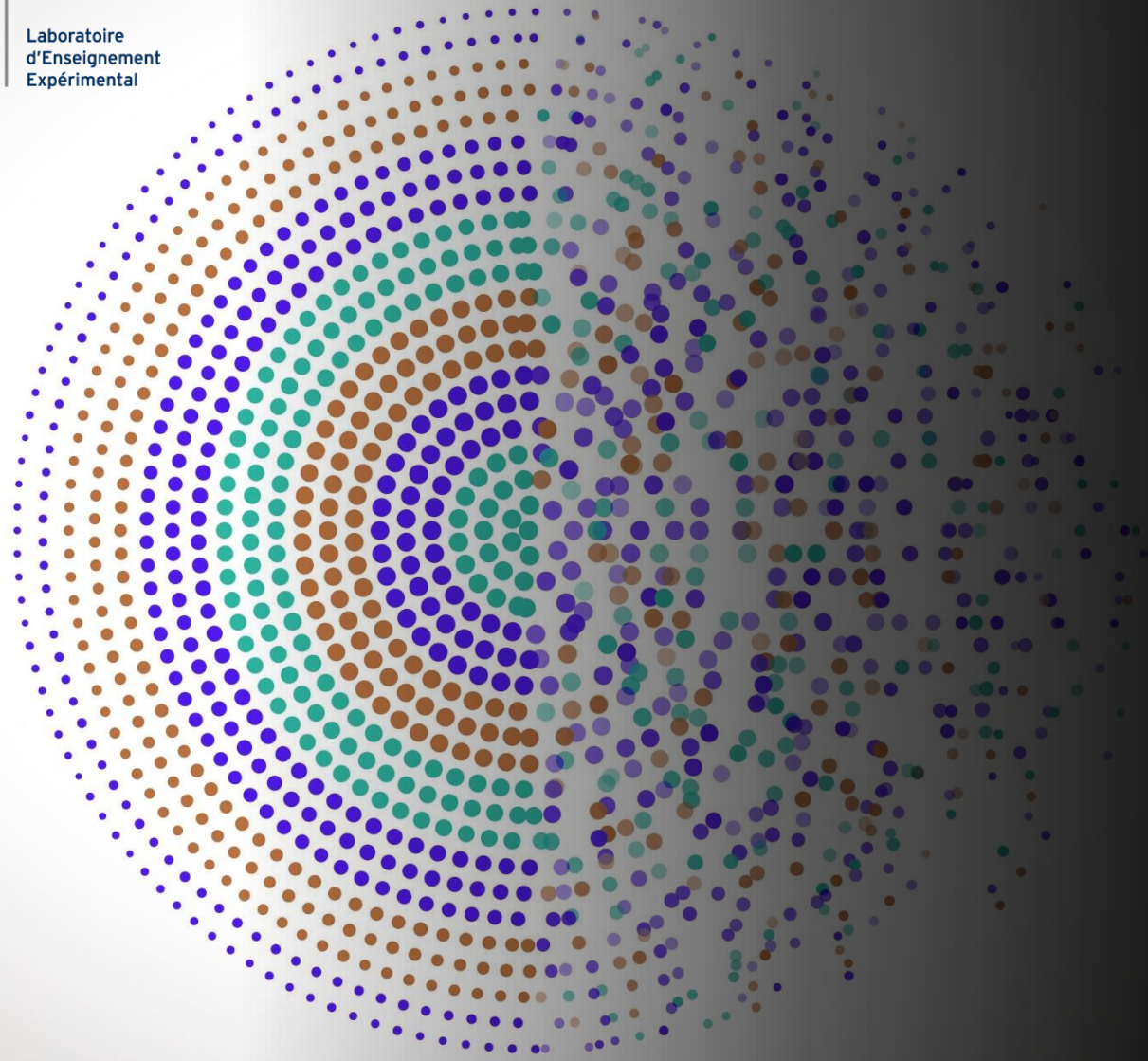
ClassName

method_name
function_name

GLOBAL_CONSTANT_NAME
global_var_name
instance_var_name

function_parameter_name
local_var_name

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental



POO
S'entrainer

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

S’entrainer à la POO

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental

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'entrainer à la POO

	ETAT	COMPORTEMENT
Point	**x** : float, **y** : float, **name**: str	**__init__**(x, y) , **__str__**() **move(**x, y**)**, **distance(Point p)**: float

Point.py

S'entrainer à la POO

	ETAT	COMPORTEMENT
Point	**x** : float, **y** : float, **name**: str	**__init__**(x, y) , **__str__**() **move(**x, y**)**, **distance(Point p)**: float

```python
class Point:
    """Class to represent a point with its coordinates in a two-dimensional
    space

    :param name: Name of the point. Default ''
    :type name: str
    :param x: X coordinate of the Point. Default 0
    :type x: float number
    :param y: Y coordinate of the Point. Default 0
    :type y: float number
    """

    def __init__(self, x_init: float=0, y_init: float=0, name_init: str='') ->
    None:
        """Constructor method
        """
        self.name: str = name_init
        self.x: float = x_init
        self.y: float = y_init
```

S'entrainer à la POO

	ETAT	COMPORTEMENT
Point	**x** : float, **y** : float, **name**: str	**__init__**(x, y) , **__str__**() **move(**x, y**)**, **distance(Point p)**: float

```
def __str__(self) -> str:
    """Display with print method
    """
    return f'Point {self.name} - X={self.x} / Y={self.y}'
```

```
def move(self, x_new, y_new):
    """Changes the coordonate of the point

    :param x_new: new X coordinate of the Point
    :type x_new: float number
    :param y_new: new Y coordinate of the Point
    :type y_new: float number
    """
    self.x = x_new
    self.y = y_new
```

```
def distance(self, point: 'Point' = None) -> None:
    """Return the distance between this point and the point given as
    parameter.

    :param point: Point to calculate the distance.
    :type point: Point
    :return: Return the distance between this point and the point giver
    parameter.
    :rtype: float

    """
    if point is None:
        return np.sqrt((self.x**2) + (self.y**2))
    else:
        return np.sqrt(((self.x-point.x)**2) + ((self.y-point.y)**2))
```

S'entrainer à la POO

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

S'entrainer à la POO

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental

	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