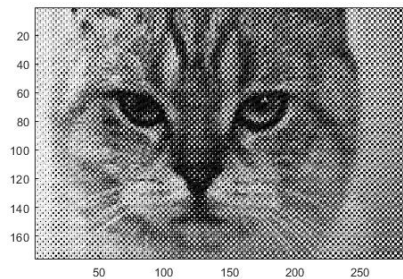
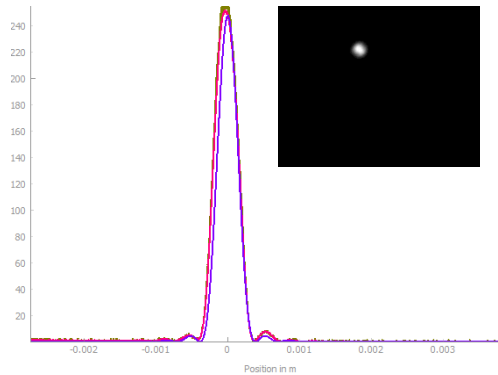
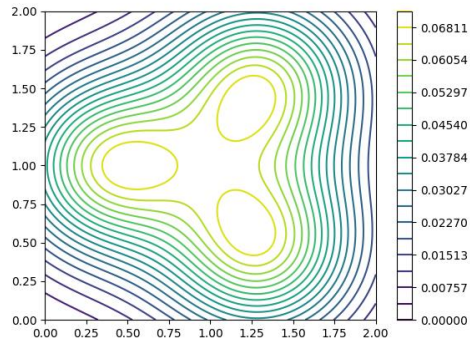


ONIP-2 / FISA

Programmation Orientée Objet

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

ONIP-2 / Déroulement



TP1 - Diffraction

TP2/3 - Filtrage Détramage

3 séquences

Programmation Objet

Filtrage

Diffraction

ENTREPRISE

TP1a

ENTREPRISE

TP1b

TP2a

TP2b

ENTREPRISE

TP3a

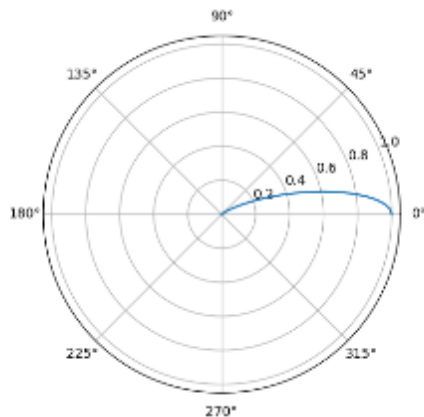
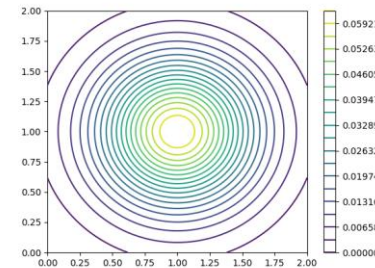
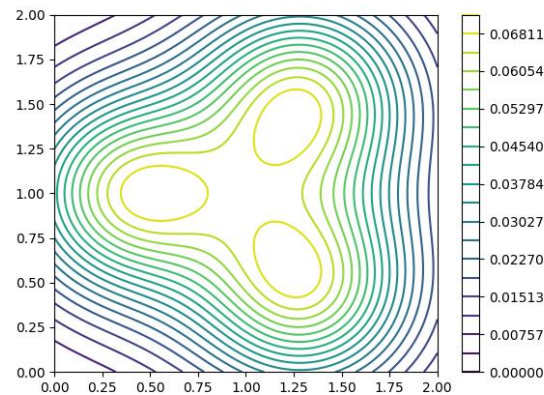
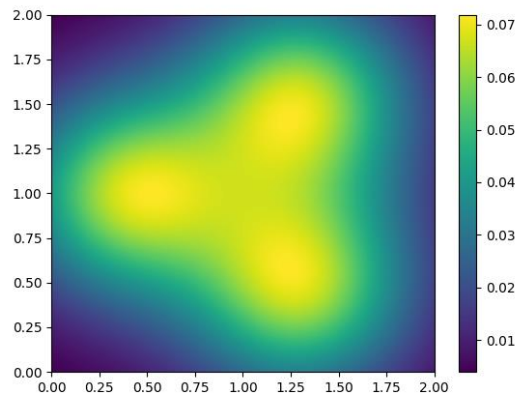
TP3b

ONIP-2 / Mini-projet – Programmation Objet

Programmation Objet

4 séances

Carte d'éclairement de sources incohérentes



Source caractérisée par leur indicatrice de rayonnement

$$I(\alpha) = I_0 \cdot \exp(-(4 \cdot \ln(2)) \cdot (\alpha/\Delta)^2)$$

Eclairement d'une source ponctuelle donnée par la formule de Bouguer

$$E = \frac{I \cdot \cos(\psi)}{d^2}$$

ENTREPRISE

TP1a

ENTREPRISE

TP1b

TP2a

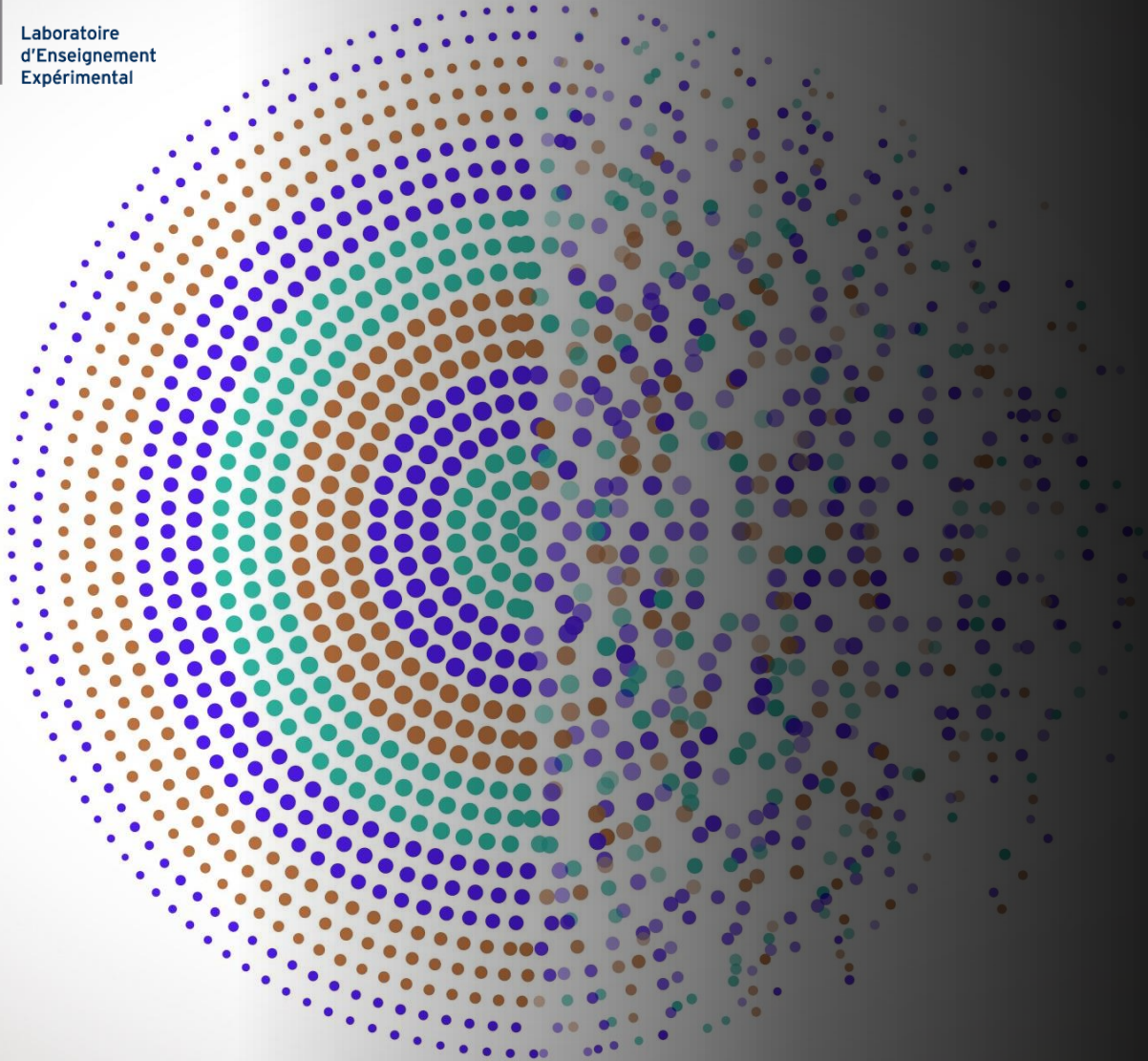
TP2b

ENTREPRISE

TP3a

TP3b

Code commenté
Validation des simulations
Figures pertinentes



Un monde d'objets

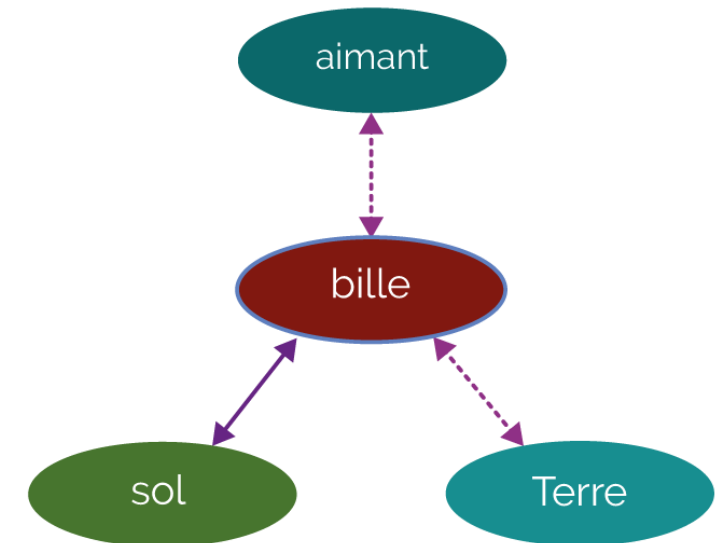
Outils Numériques / Semestre 6
/ Institut d'Optique / ONIP-2

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...

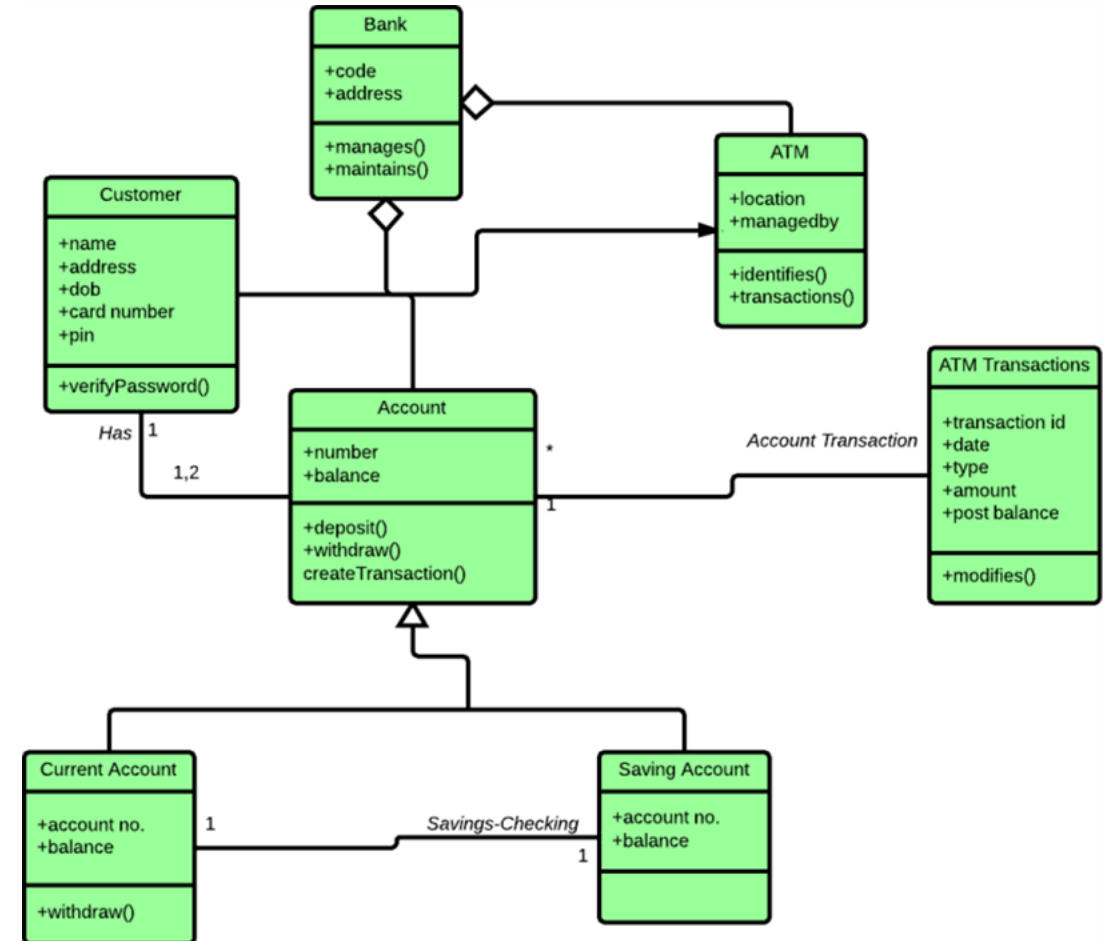
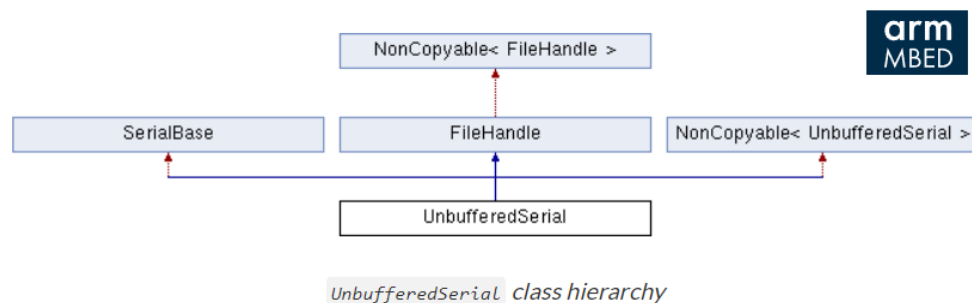
rouler, freiner, klaxonner...

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](#)

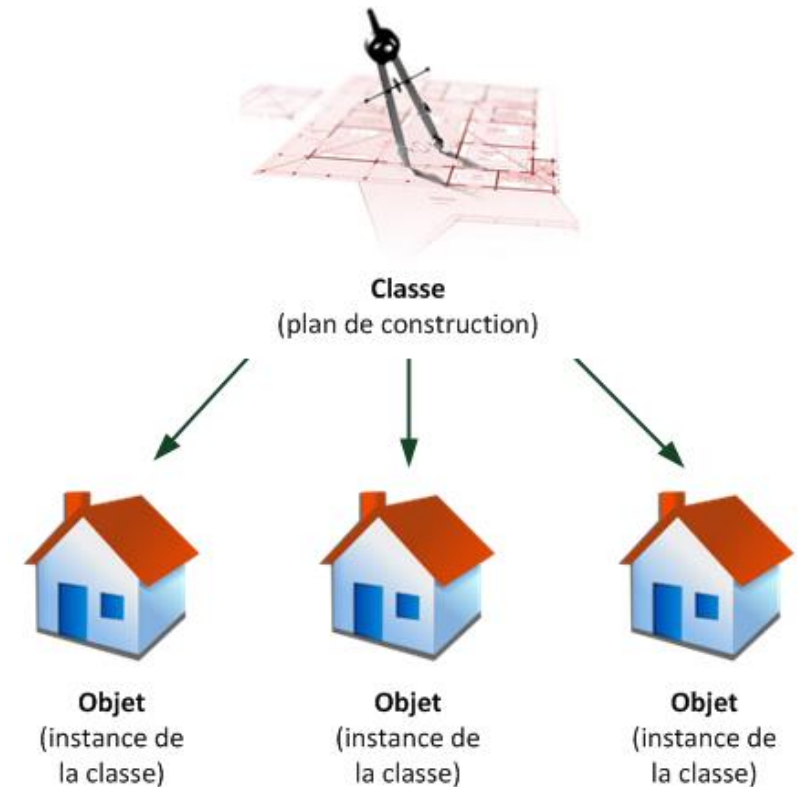
UnbufferedSerial



Programmation orientée objet

Éléments de base

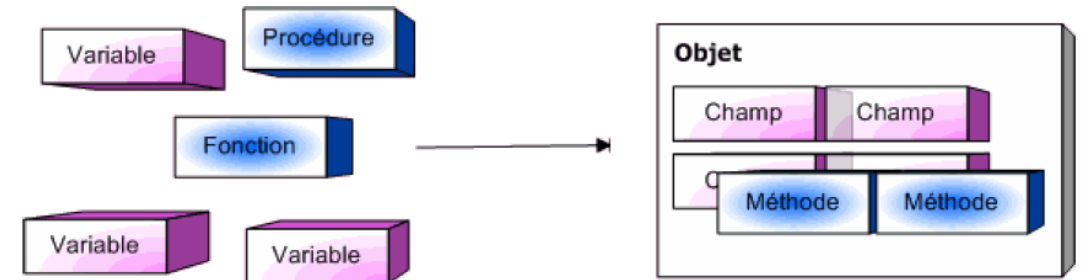
- **Classe** : rassemblement de différents **attributs** (état d'un objet) et **méthodes** (actions possibles d'un objet)
- **Objet** : instance d'une classe



Programmation orientée objet

Concepts fondamentaux

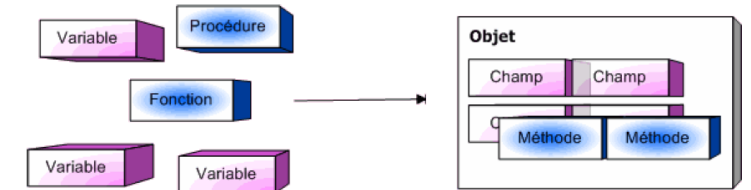
- **Encapsulation** : regroupement de différentes données et fonctions sous une même entité
- **Héritage** : arborescence de classes permettant la spécialisation
(notion non abordée dans ce module)



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

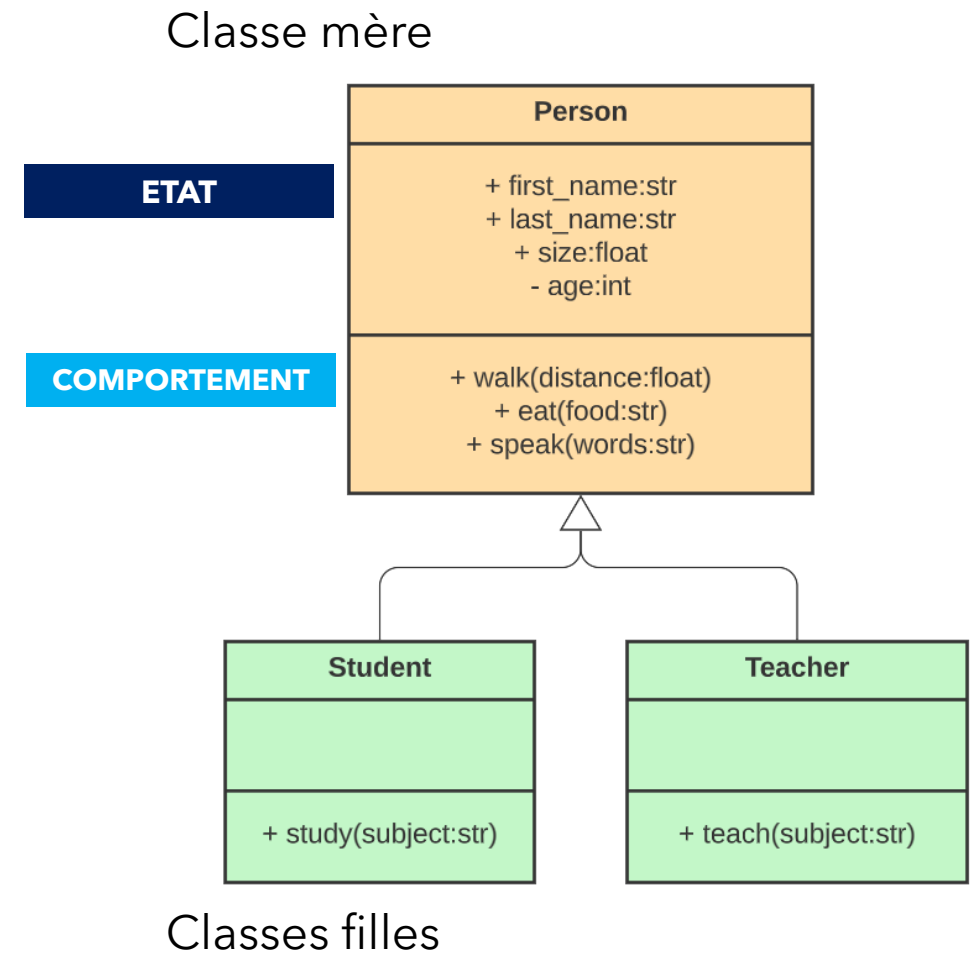
- *max* ([*axis...*])
- *resize* (*new_shape...*)

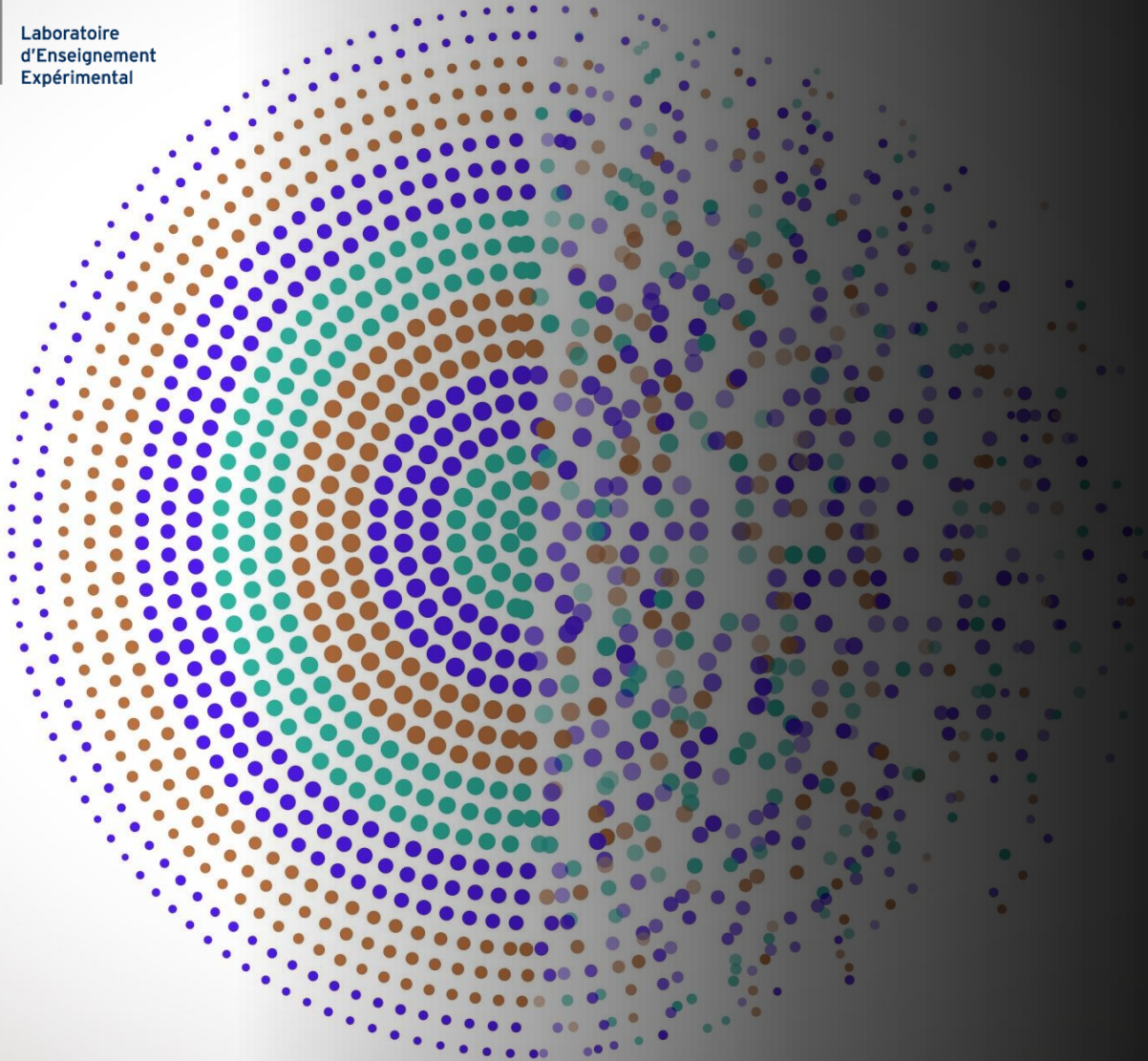
ETAT

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

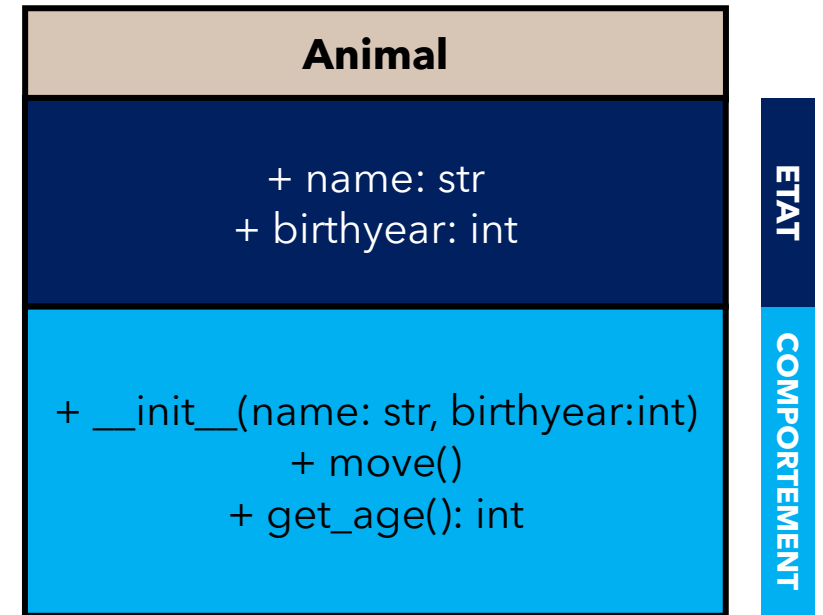
Encapsulation : regroupement de différentes données et fonctions sous une même entité

```
import datetime

class Animal:
    """ object class Animal
    """
    def __init__(self, name:str="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

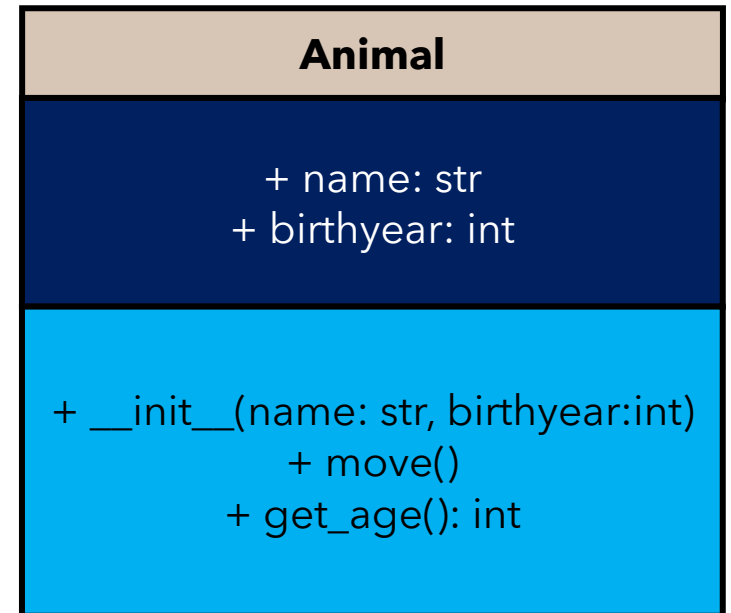
variables, propres à un objet (instance d'une classe), nommées **attributs**

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



ETAT

COMPORTEMENT

`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
```

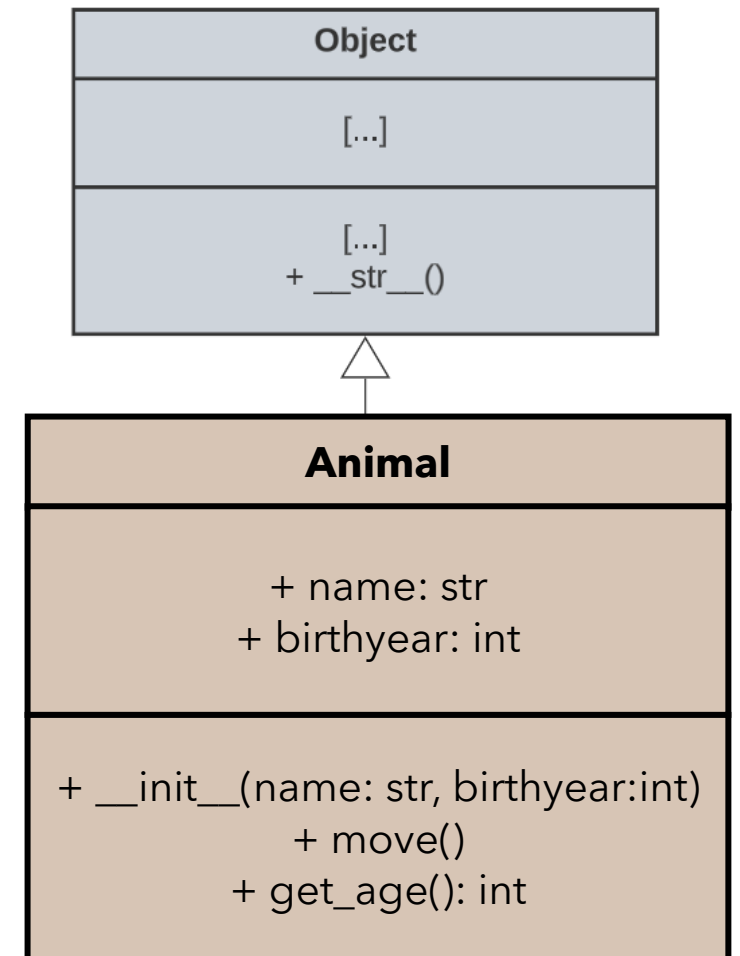
Utilisation d'une classe

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

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

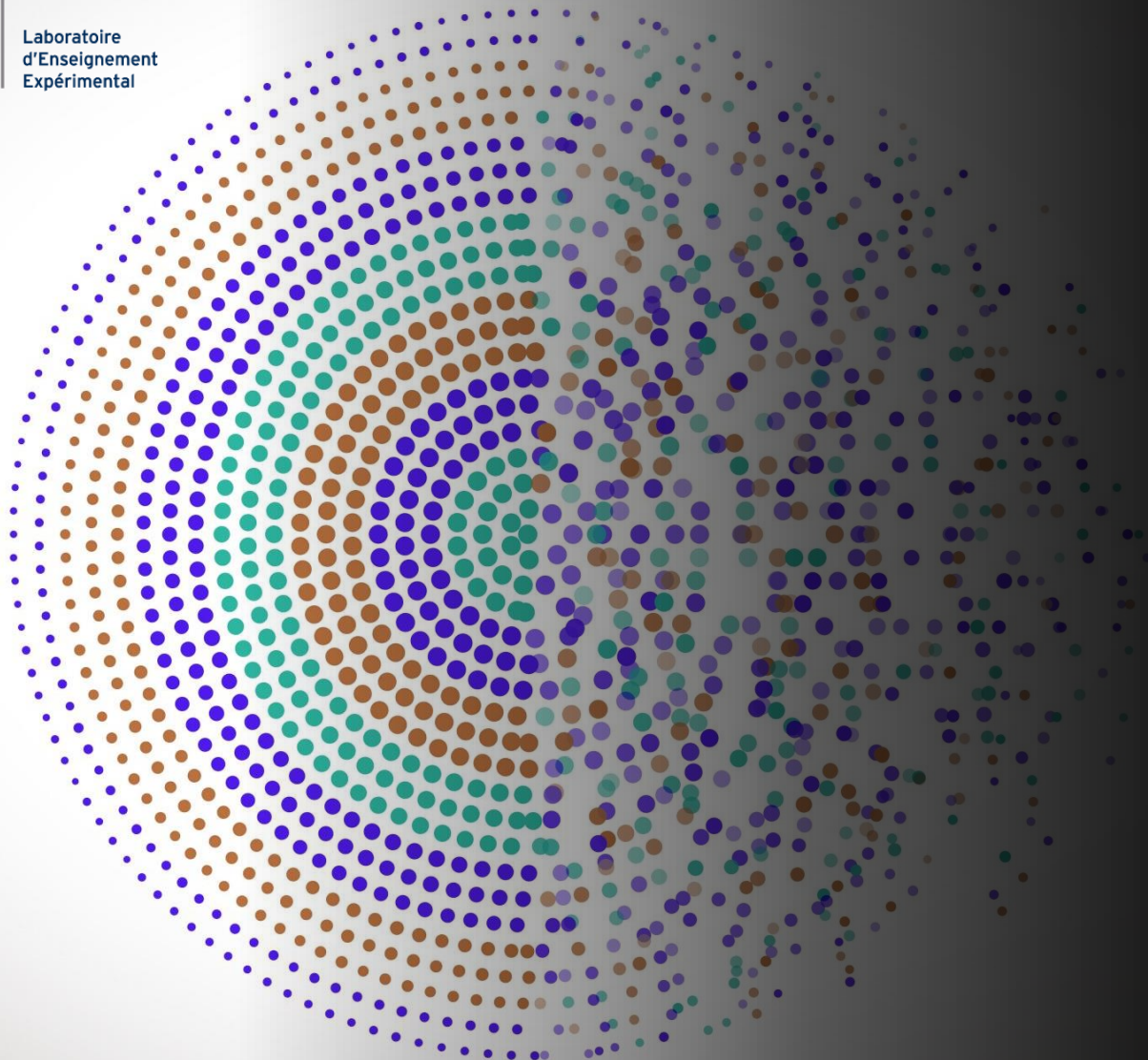
GLOBAL_CONSTANT_NAME

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

Point

ETAT

x : float, **y** : float, **name**: str

COMPORTEMENT

`__init__(x, y)` , **`__str__()`**
`move(x, y)`, **`distance(Point p)`**: float

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

```
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'entraîner à la POO

Point

ETAT

x : float, **y** : float, **name**: str

COMPORTEMENT

`__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 given
    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'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