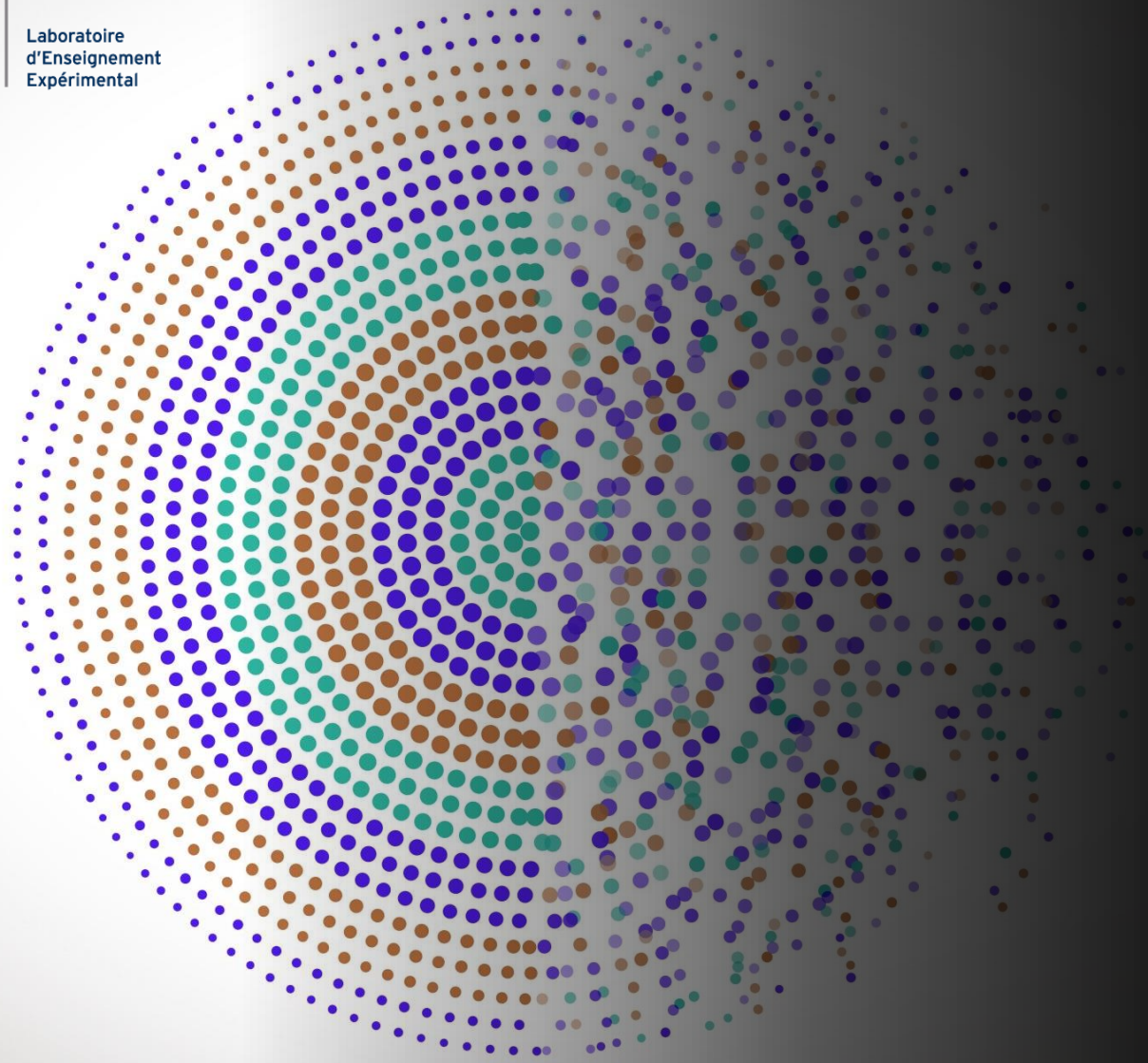
INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental



Un monde d'objets

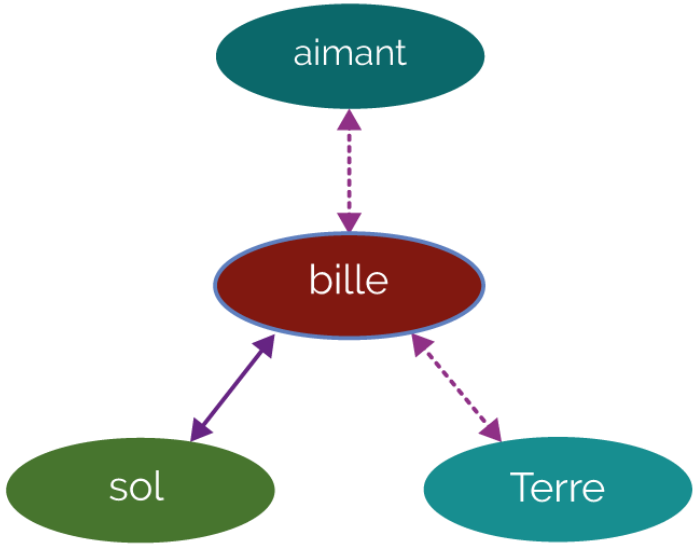
Outils Numériques / Semestre 5 / Institut d'Optique / ONIP-2

Un monde d'objets

https://masevaux.fr/objets_trouves/

- Des objets qui interagissent

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



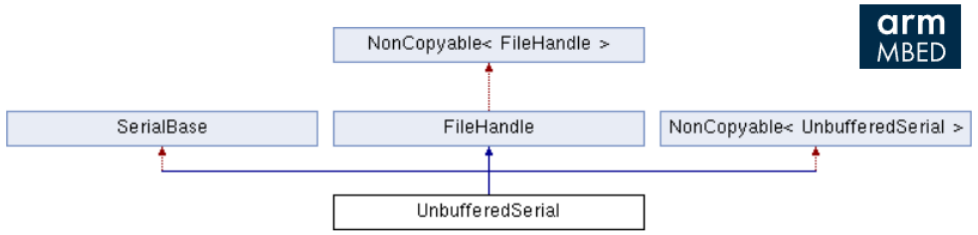
https://www.maxicours.com/se/cours/les-diagrammes-objet-interaction/

Un monde d'objets informatiques

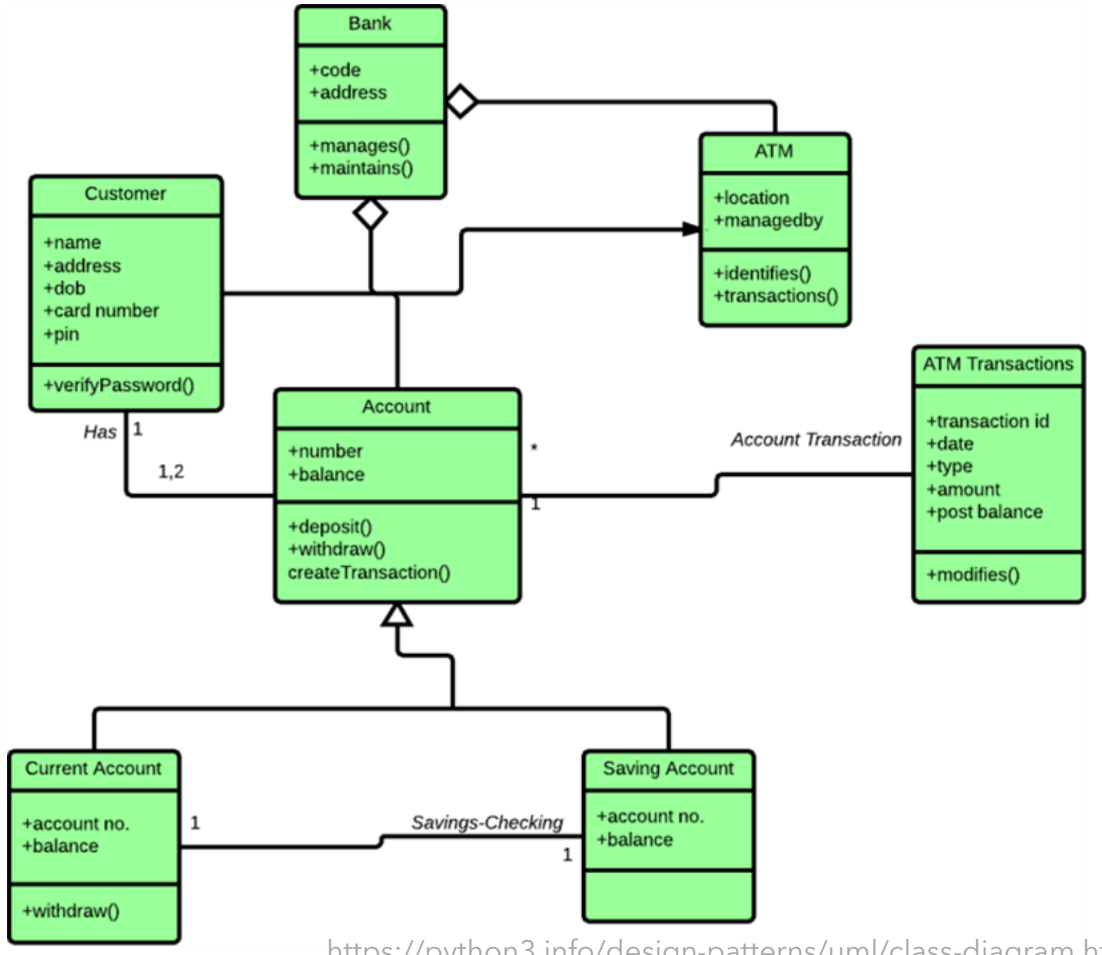
- Mise en œuvre informatique

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

UnbufferedSerial



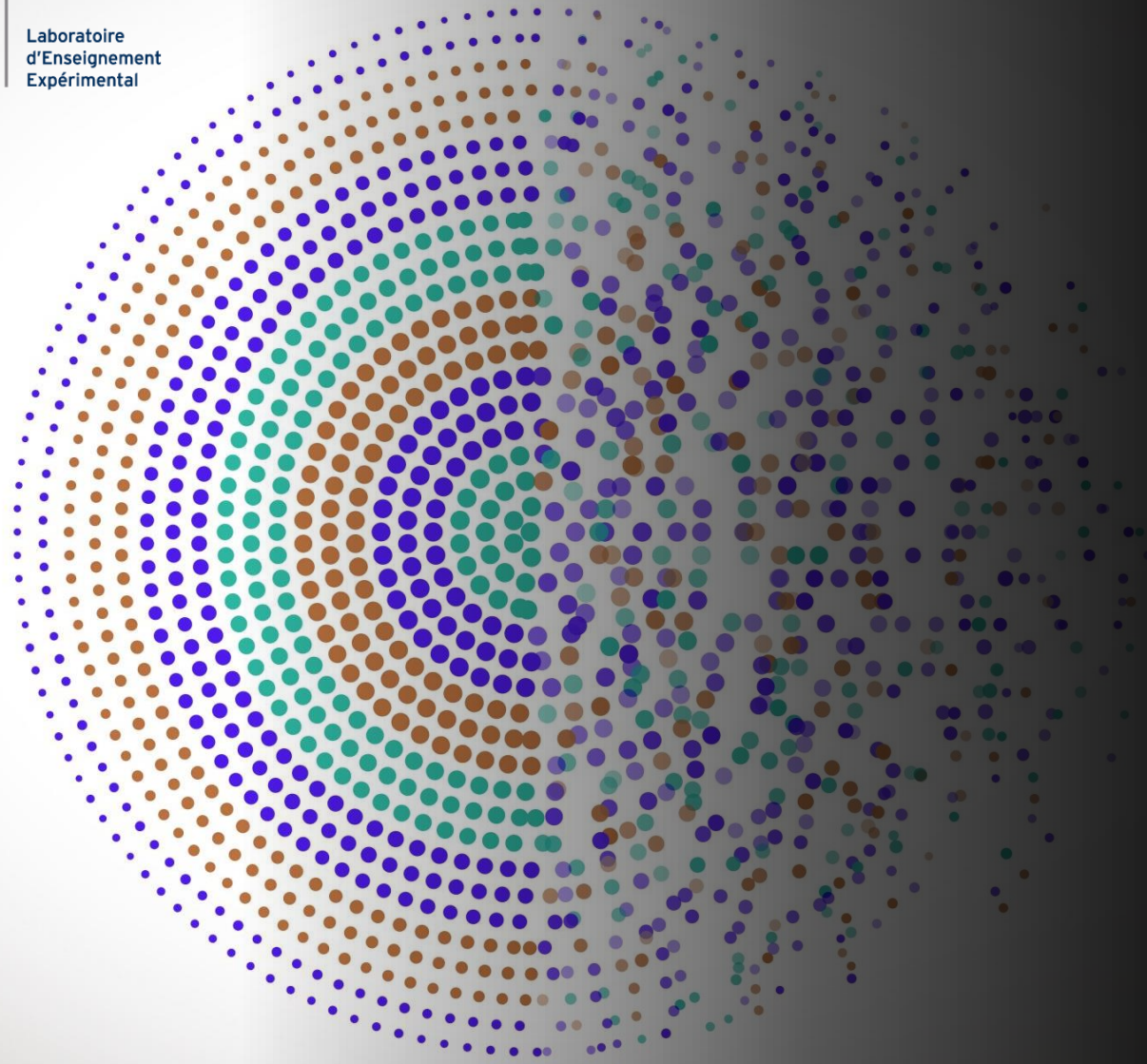
`UnbufferedSerial` *class hierarchy*



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

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

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



Et avec Python ?

Outils Numériques / Semestre 5 / Institut d'Optique / ONIP-2

C’est quoi cette syntaxe ???

```
import numpy
```

- Que représentent ces différentes syntaxes ?

```
v = numpy.array([1, 2, 3])
```

```
a = v.max()
```

```
print( v.shape )
```

C’est quoi cette syntaxe ???

```
import numpy
```

- Que représentent ces différentes syntaxes ?

```
v = numpy.array([1, 2, 3])
```

array est une fonction de la bibliothèque Numpy

```
print( type( v ) )
```

```
a = v.max()
```

```
print( v.shape )
```

C'est quoi cette syntaxe ???

```
import numpy
```

```
v = numpy.array([1, 2, 3])
```

array est une fonction de la bibliothèque Numpy

```
print( type( v ) )
```

v est un objet de type **ndarray** (dont la définition est donnée dans la bibliothèque Numpy)

numpy.ndarray

class **numpy.ndarray**(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html

C’est quoi cette syntaxe ???

```
import numpy
```

- Que représentent ces différentes syntaxes ?

```
v = numpy.array([1, 2, 3])
```

array est une fonction de la bibliothèque Numpy

```
print( type( v ) )
```

v est un objet de type **ndarray** (dont la définition est donnée dans la bibliothèque Numpy)

```
a = v.max()
```

```
print( v.shape )
```

C'est quoi cette syntaxe ???

```
import numpy
```

```
v = numpy.array([1, 2, 3])
```

array est une fonction de la bibliothèque Numpy

```
print( type( v ) )
```

v est un objet de type **ndarray** (dont la définition est donnée dans la bibliothèque Numpy)

```
a = v.max()
```

max est une méthode de la classe **ndarray** qui retourne un flottant ou un entier

```
print( v.shape )
```

max est un attribut de la classe **ndarray** qui retourne un ***Tuple*** de nombres

C'est quoi cette syntaxe ???

import numpy

Numpy est module qui contient des fonctions mais aussi des **classes** avec leurs attributs et méthodes

v **= numpy.array**([1, 2, 3])

array est une fonction de la bibliothèque Numpy

print(**type**(*v*))

v est un objet de type **ndarray** (dont la définition est donnée dans la bibliothèque Numpy)

Attributes: `T` : *ndarray*

View of the transposed array.

`data` : *buffer*

Python buffer object pointing to the start of the array's data.

`dtype` : *dtype object*

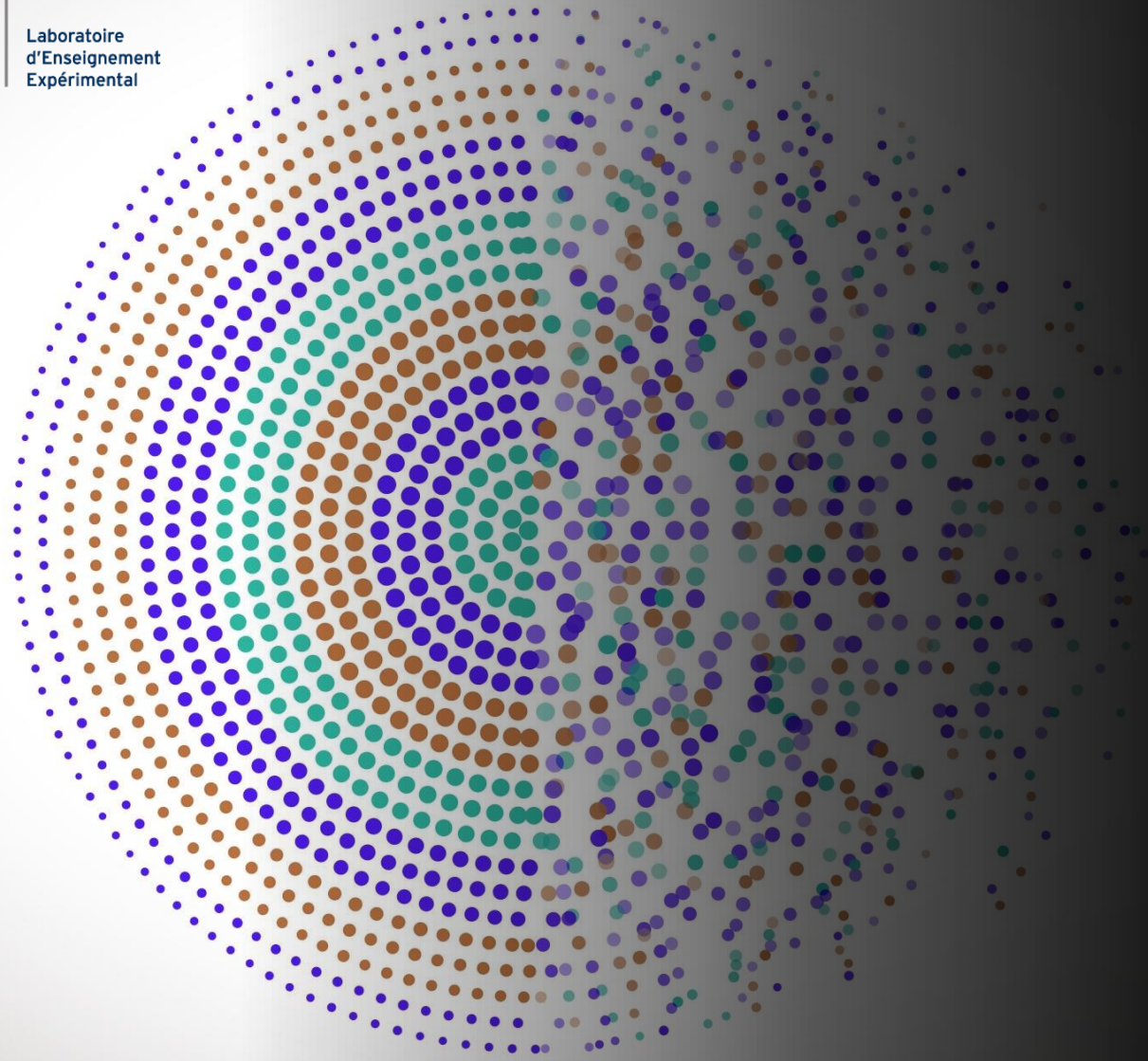
Data-type of the array's elements.

Methods

`all`([axis, out, keepdims, where])	Returns True if all elements evaluate to True.
`any`([axis, out, keepdims, where])	Returns True if any of the elements of *a* evaluate to True.
`argmax`([axis, out, keepdims])	Return indices of the maximum values along the given axis.

https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html

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



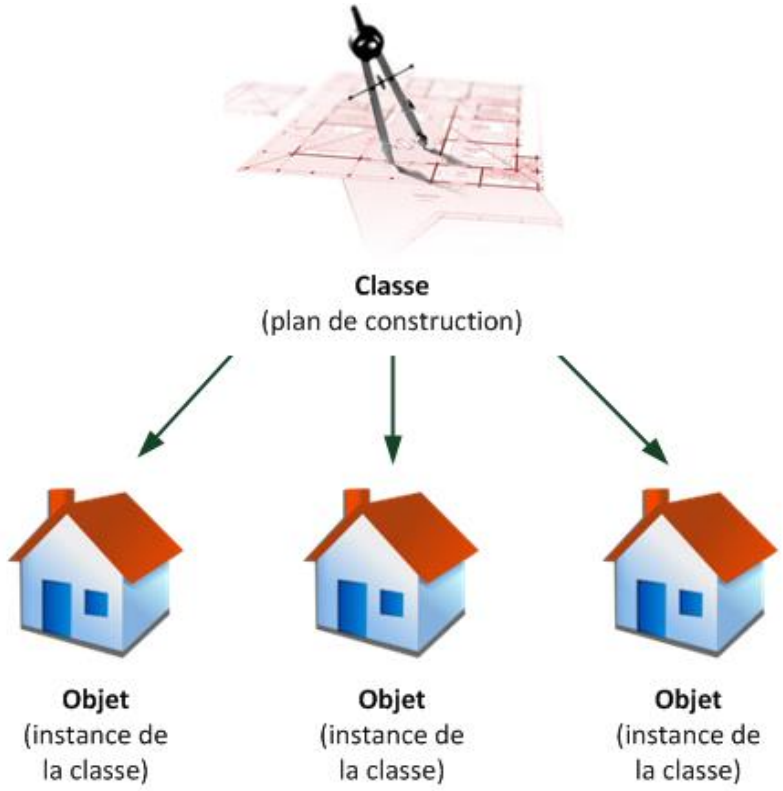
Classes et objets en Python

Outils Numériques / Semestre 5 / Institut d'Optique / ONIP-2

Programmation orientée objet

Eléments de base

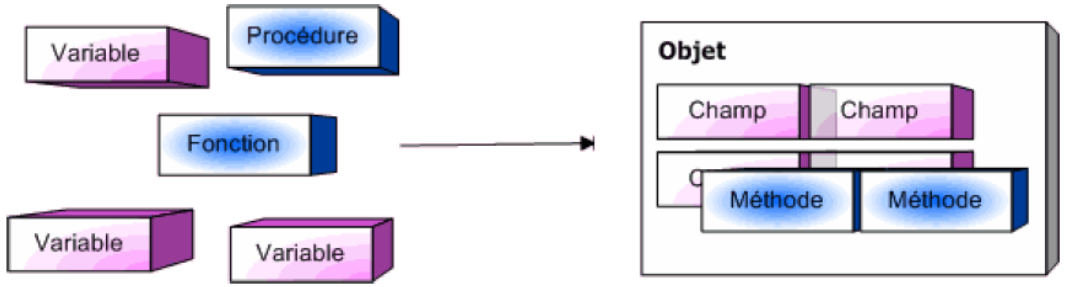
- **Classe** : rassemblement de différents **attributs** (état d'un objet) et **méthodes** (actions possibles d'un objet)
- **Objet** : instance d'une classe



Programmation orientée objet

Concepts fondamentaux

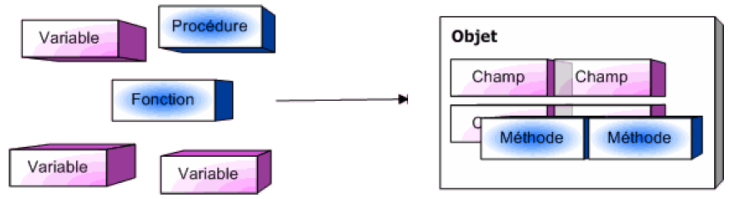
- **Encapsulation** : regroupement de différentes données et fonctions sous une même entité
- **Héritage** : arborescence de classes permettant la spécialisation
  (notion non abordée dans ce module)



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)*

Méthodes

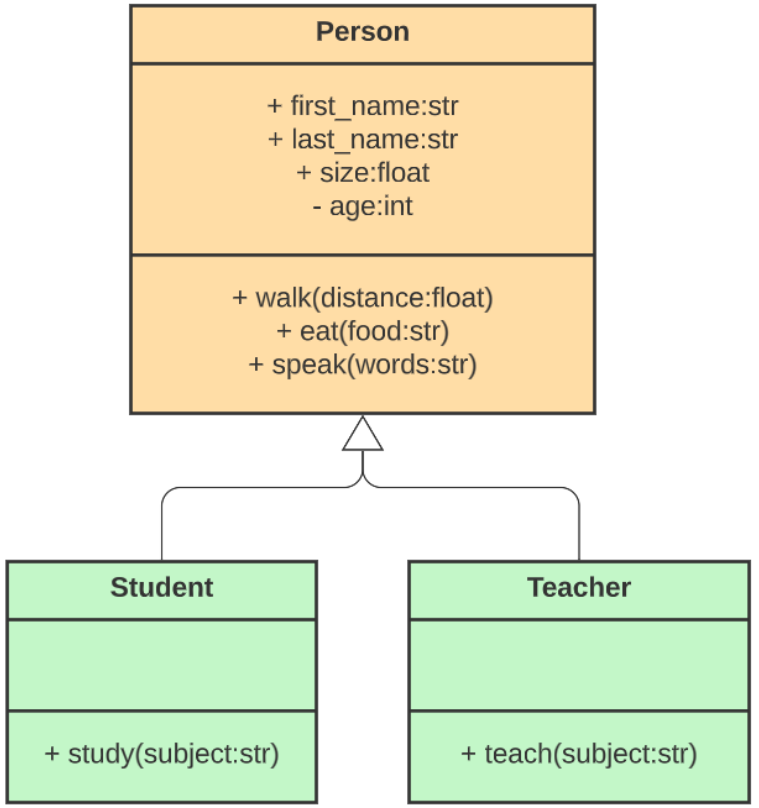
- *max ([axis…])*
- *resize (new_shape…)*

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 mère



Classes filles

Exemple d'une classe

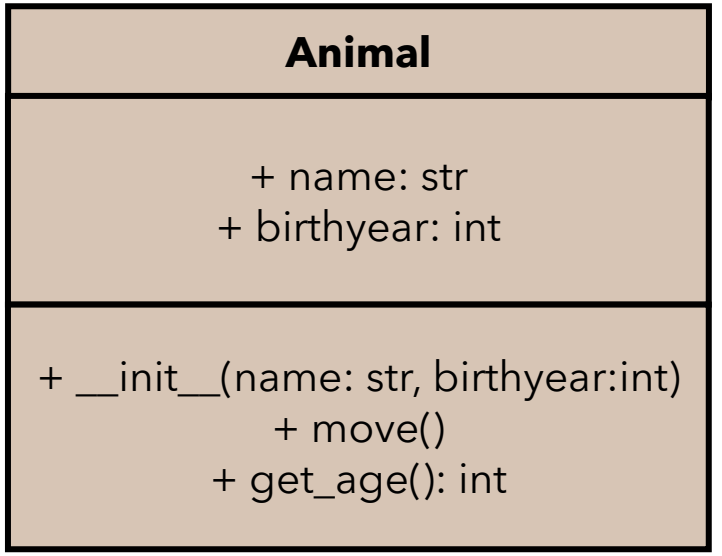
Encapsulation : regroupement de différentes données et fonctions sous une même entité

```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
```



__init__(self,...) est le constructeur : méthode appelée à l'instanciation d'un objet
self est le mot clé utilisé pour accéder aux méthodes et attributs d'instance

Exemple d'une classe

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

ETAT

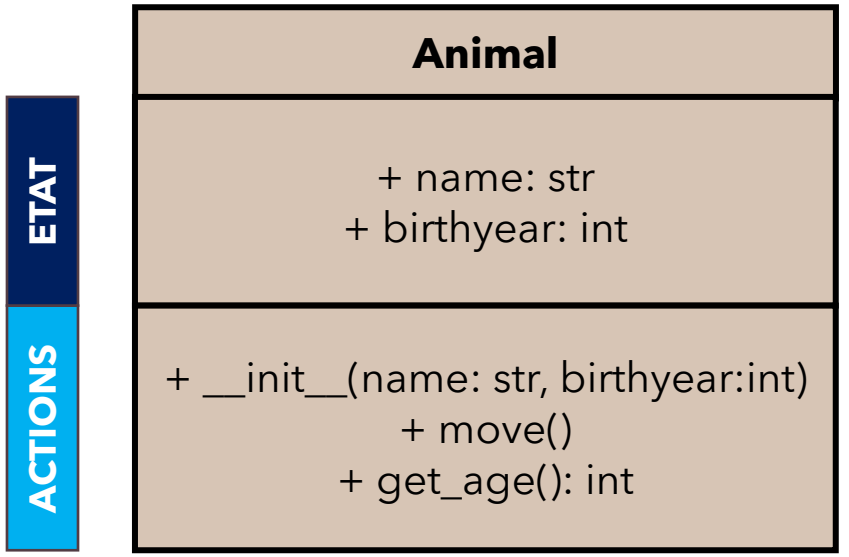
variables, *propres à un objet (instance d'une classe), nommées* ***attributs***

ACTIONS

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***

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
```

ACTIONS

ETAT

__init__

constructeur

- *méthode* ***obligatoire***
- *nécessairement nommée* ***__init__***

autres

comportements spécifiques

- *pas de limite dans le nombre de méthodes*
- *contiennent nécessairement self comme premier argument*

Exemple 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
```

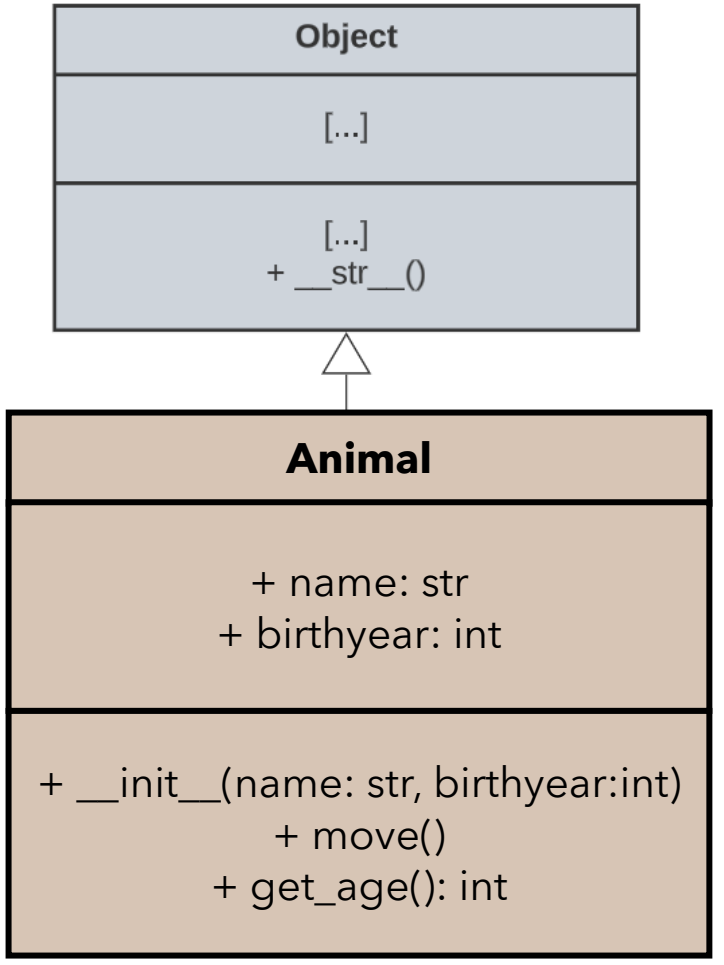
Exemple d'une classe

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
```

Exemple d'une classe

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