



Un monde d'objets

Outils Numériques / Semestre 5
/ Institut d'Optique / B0_3

Un monde d'objets

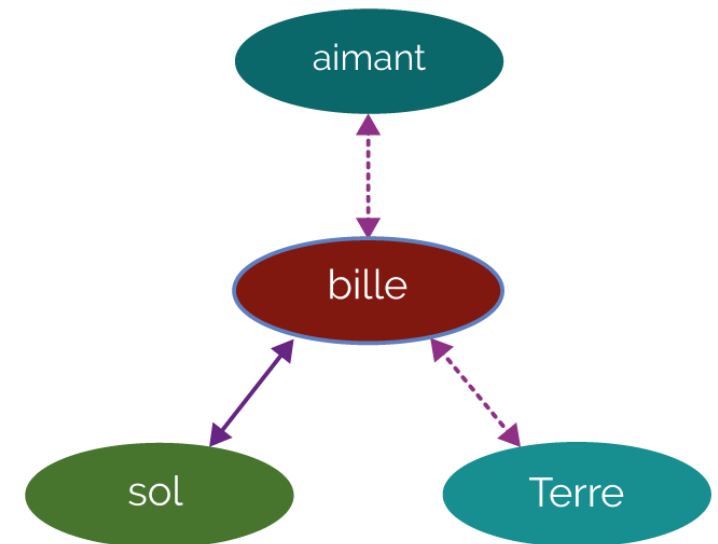


https://masevaux.fr/objets_trouves/

- Des objets qui interagissent



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



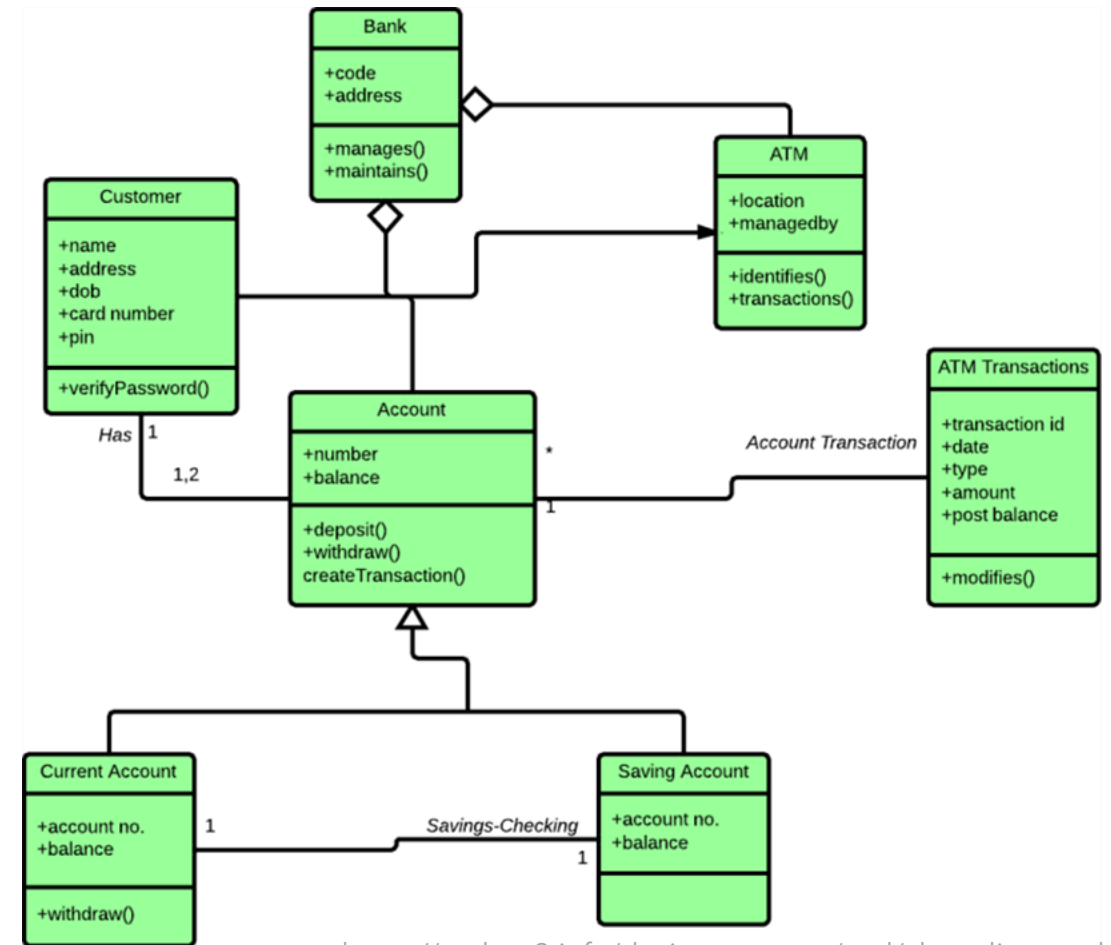
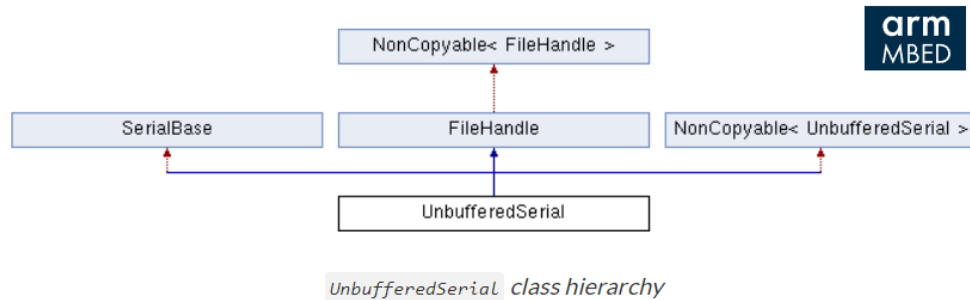
<https://www.maxicours.com/se/cours/les-diagrammes-objet-interaction/>

Un monde d'objets informatiques

- Mise en œuvre informatique

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

UnbufferedSerial



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



Et avec Python ?

Outils Numériques / Semestre 5
/ Institut d'Optique / B0_3

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 méthode 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 méthode 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.)

C'est quoi cette syntaxe ???

```
import numpy
```

- Que représentent ces différentes syntaxes ?

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

array est une méthode 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 méthode 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 un **ensemble de classes** avec leurs attributs et méthodes

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

array est une méthode 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*

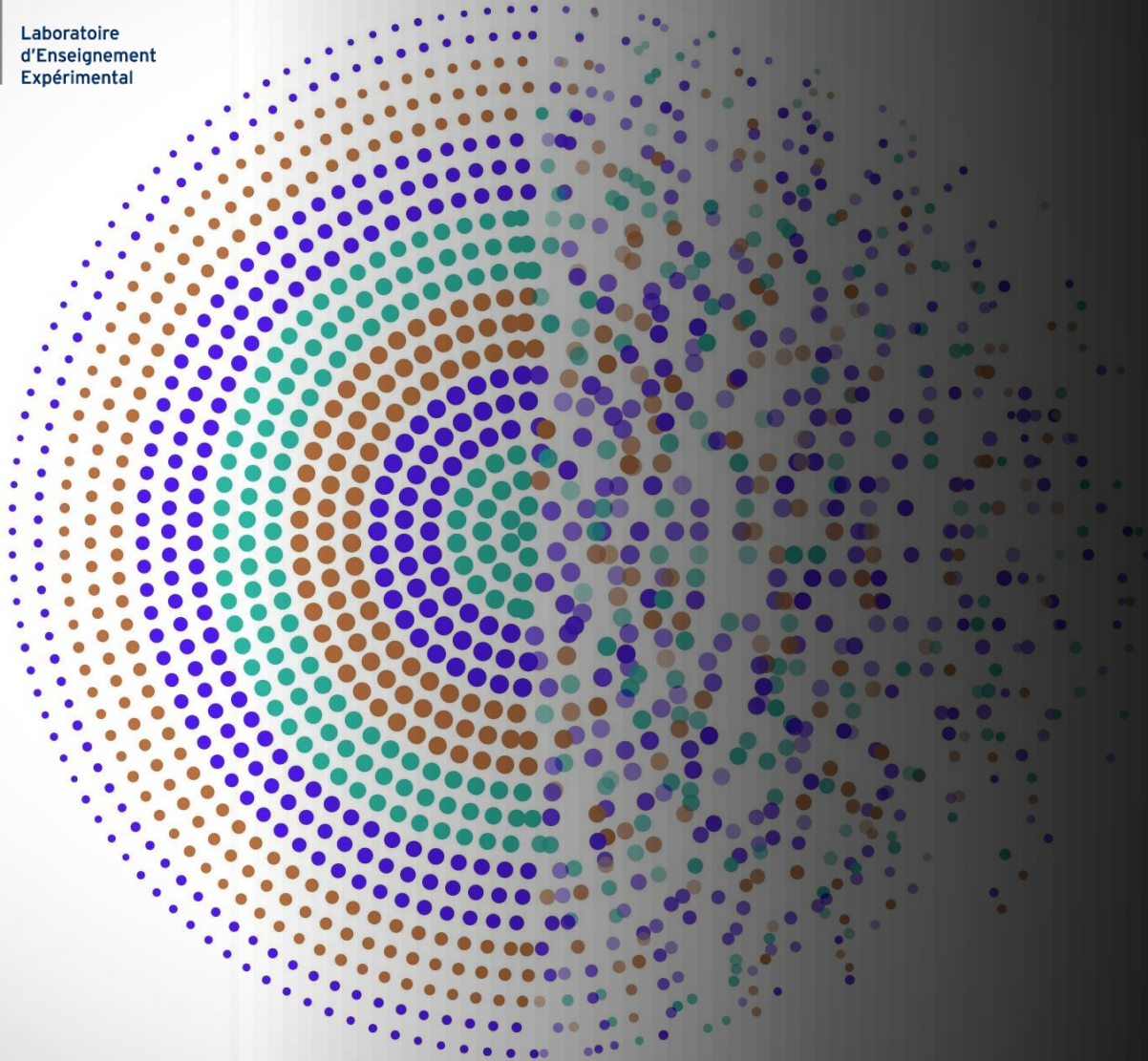
Python buffer object pointing to the start of the array's data.

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 <i>a</i> evaluate to True.
argmax ([axis, out, keepdims])	Return indices of the maximum values along the given axis.



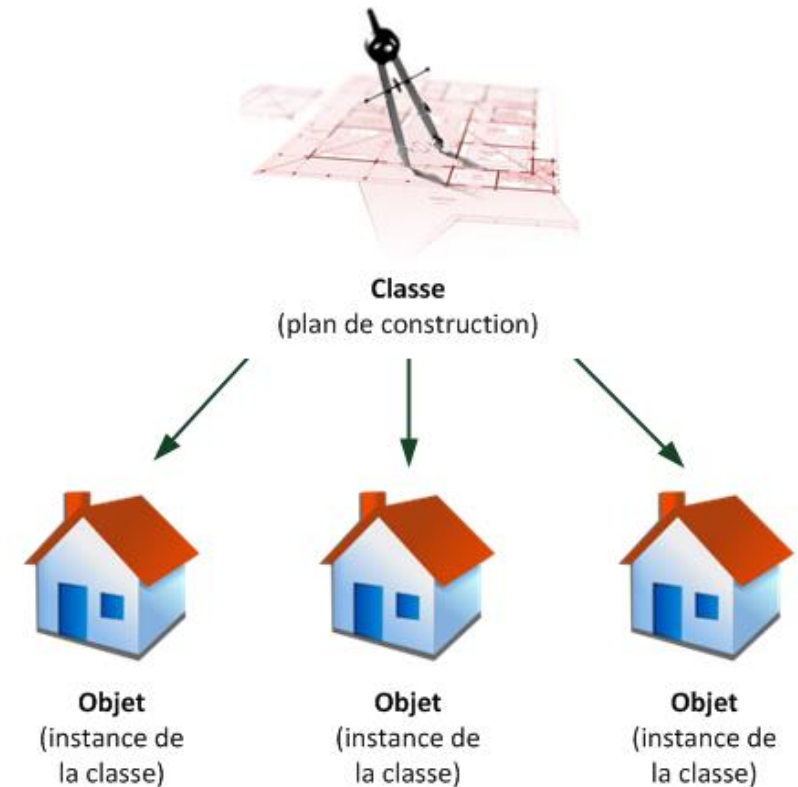
Classes et objets en Python

Outils Numériques / Semestre 5
/ Institut d'Optique / B0_3

Programmation orientée objet

Éléments de base

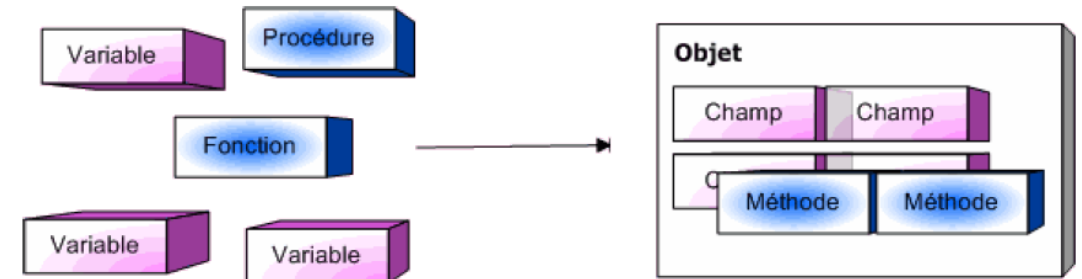
- **Classe** : rassemblement de différents **attributs** (état d'un objet) et **méthodes** (actions possibles d'un objet)
- **Objet** : instance d'une classe



Programmation orientée objet

Concepts fondamentaux

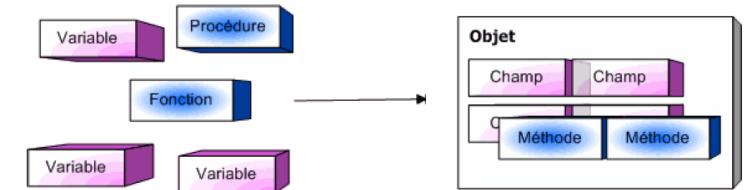
- **Encapsulation** : regroupement de différentes données et fonctions sous une même entité
- **Héritage** : arborescence de classes permettant la spécialisation



Programmation orientée objet

Concepts fondamentaux

- **Encapsulation** : regroupement de différentes données et fonctions sous une même entité
- **Héritage** : arborescence de classes permettant la spécialisation



classe ***numpy.ndarray***

Attributs

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

Méthodes

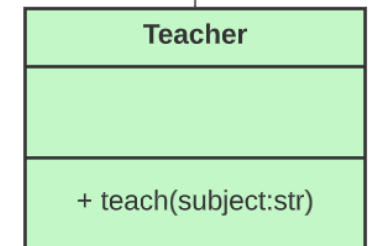
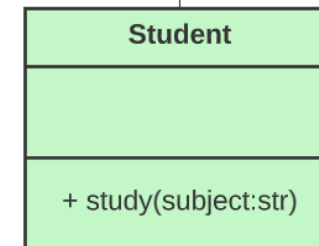
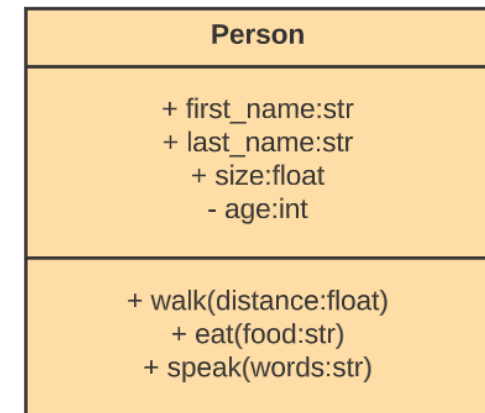
- *max* ([axis...])
- *resize* (new_shape...)

Programmation orientée objet

Concepts fondamentaux

- **Encapsulation** : regroupement de différentes données et fonctions sous une même entité
- **Héritage** : arborescence de classes permettant la spécialisation

Classe mère



Classes filles

Exemple d'une classe

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

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

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

    def speak(self):
        print(f"\t[ {self.name} ] is saying {self.sound}")
```

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

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")
    print("Animal 2 Name = ", animal2.name)

    print(animal1)
```

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

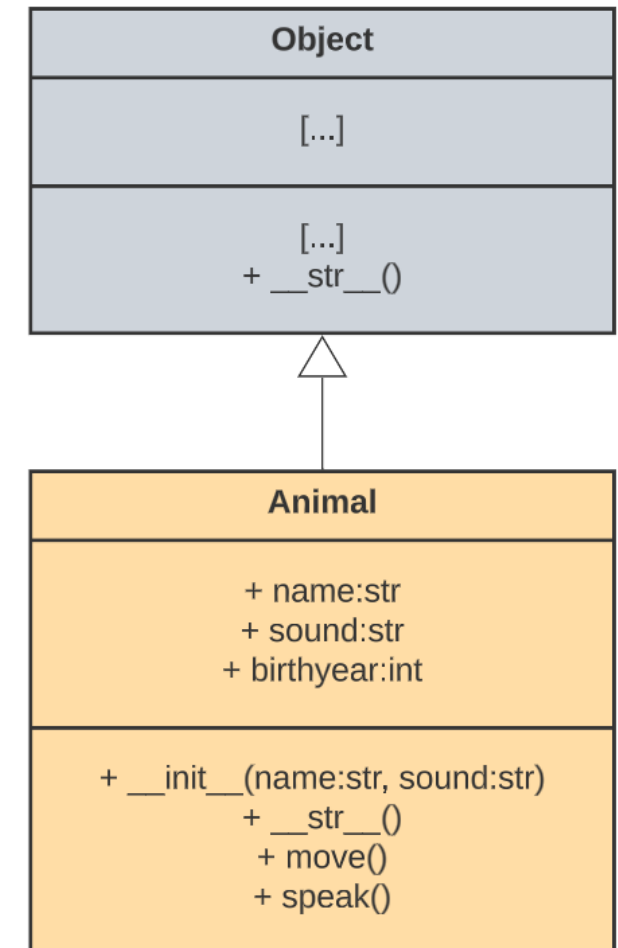
```
Animal 1 Name = Hello
Animal 2 Name = Garfield
<__main__.Animal object at 0x0000020C594D2F10>
```

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}"
    [...]
```



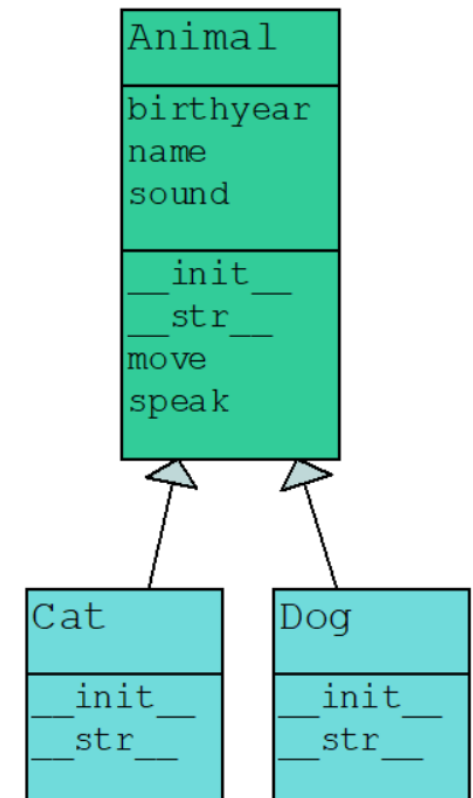
Animal 1 Name = Hello
 Animal 2 Name = Garfield
 Animal [Hello] born in 2000

Exemple de classes héritées

Héritage : arborescence de classes permettant la spécialisation

```
class Cat(Animal):
    """ Object class Cat, inherit from Animal
    """
    def __init__(self, name="Hello", sound="Miaouh"):
        """ Cat class constructor
        :name: name of the animal
        """
        super().__init__(name, sound)

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



Exemple de classes héritées

Héritage : arborescence de classes permettant la spécialisation

```
dog1 = Dog("Ralph")  
dog1.birthyear = 2012
```

```
Animal 1 Name = Hello  
Animal 2 Name = Garfield  
Animal [ Garfield ] born in 2000  
    [ Garfield ] is moving  
    [ Garfield ] is saying ...  
Animal/CAT [ Tigrou ] born in 2000  
    [ Tigrou ] is moving  
    [ Tigrou ] is saying Miaouh  
Animal/DOG [ Ralph ] born in 2012  
    [ Ralph ] is moving  
    [ Ralph ] is saying Wouaf
```

