Les classes en python

Une *classe* permet de construire des “*objets*”

C’est une façon de lier des “données” et les “*méthodes* à leur appliquer” (fonctions)

Sans le dire, nous les utilisons depuis le début : en python “tout est objet”...

Une “réalisation” d’un objet est une *instance*

On parle de *programmation orientée objet*

Python is an object oriented programming language. Unlike procedure oriented programming, where the main emphasis is on functions, object oriented programming stresses on objects.

An object is simply a collection of data (variables) and methods (functions) that act on those data. Similarly, a class is a blueprint for that object.

Les classes en python

```python
maListe = [1,7,15]

print(type(maListe))
print(dir(maListe))

mot = "hello"

print(type(mot))
print(dir(mot))

def f(x):
    return x**2

print(type(f))
print(dir(f))
```

```
<class 'list'>
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',
'__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert',
'pop', 'remove', 'reverse', 'sort']
<class 'str'>
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',
'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',
'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
<class 'function'>
['__annotations__', '__call__', '__class__', '__closure__', '__code__', '__defaults__', '__delattr__',
'__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__get__', '__getattribute__',
'__globals__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__kwdefaults__', '__le__', '__lt__',
'__module__', '__name__', '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__']
```

Les classes en python

```python
class Mammifere:
    """un exemple minimal de classe"""

    def __init__(self):
        self.cri = "bonjour"

    def parle(self):
        print("je sais dire",self.cri)
```

Convention : débuter par une majuscule

```python
animal_1 = Mammifere()  # une instance
print(animal_1)

animal_1.parle() # une méthode appliquée à l'objet
animal_1.cri = "Haaa !"
animal_1.parle()
```

```
<__main__.Mammifere object at 0x7f57e2dc4cd0>
je sais dire bonjour
je sais dire Haaa !
```

A la création d'une instance __init__

```
class Mammifere:

    def __init__(self,parole,ans):
        self.cri = parole
        self.age = ans

    def parle(self):
        print("je sais dire",self.cri)


animal_1 = Mammifere("hello",15)  # une instance
animal_1.parle() # une méthode appliquée à l'objet
print(animal_1.age)
```

```
je sais dire hello
15
```

Plusieurs instances d’un objet

```python
class Mammifere:

    def __init__(self,parole,ans):
        self.cri = parole
        self.age = ans

    def parle(self):
        print("je sais dire",self.cri)

animal_1 = Mammifere("hello",15)  # une instance
animal_2 = Mammifere("argh", 83)  # une autre instance
animal_3 = Mammifere("areuh", 2)  # une troisième instance

animal_2.parle() # une méthode appliquée à l'objet
```

```
In [19]: runcell(0, '/home/mrigo/untitled0.py')
je sais dire argh
```

Définir/accéder aux attributs d’un objet

```
    def __init__(self,parole,ans):
        self.cri = parole
        self.age = ans
        self.aptitudes = []
```

```
    def nouvelle_aptitude(self,x):
        self.aptitudes.append(x)

    def afficher_aptitudes(self):
        for c in self.aptitudes:
            print("je sais",c)

animal_1 = Mammifere("hello",15)  # une instance
animal_1.nouvelle_aptitude("manger")
animal_1.nouvelle_aptitude("dormir")
print(animal_1.aptitudes)
animal_1.afficher_aptitudes()
```

```
['manger', 'dormir']
je sais manger
je sais dormir
```

Définir/accéder aux attributs d’un objet

```
    def supprimer_aptitude(self):
        if len(self.aptitudes)==0:
            print("rien à supprimer")
        else:
            print(self.aptitudes.pop(),"a été supprimé")

animal_1 = Mammifere("hello",15)
animal_1.nouvelle_aptitude("manger")
animal_1.nouvelle_aptitude("dormir")
animal_1.afficher_aptitudes()
animal_1.supprimer_aptitude()
animal_1.supprimer_aptitude()
animal_1.supprimer_aptitude()
animal_1.afficher_aptitudes()
```

```
je sais manger
je sais dormir
dormir a été supprimé
manger a été supprimé
rien à supprimer
```

__str__

```python
class Mammifere:

    def __init__(self,parole,ans):
        self.cri = parole
        self.age = ans

    def __str__(self):
        return "Je suis un Mammifere de {} ans".format(self.age)

    def parle(self):
        print("je sais dire",self.cri)

animal_1 = Mammifere("hello",15)
print(animal_1)
```

```
In [44]: runcell(0, '/home/mrigo/untitled1.py')
<__main__.Mammifere object at 0x7f57e32380d0>

In [45]: runcell(0, '/home/mrigo/untitled1.py')
Je suis un Mammifere de 15 ans
```

Composer les types : listes, dicos, ...

```
liste =  [Mammifere("Hello", 17) , Mammifere("Ouuu", 1)]
print(liste)
```

```
In [8]: runcell(0, '/home/mrigo/exemple_classes.py')
[<__main__.Mammifere object at 0x7f87c08aa370>, <__main__.Mammifere object at 0x7f87c0865040>]
```

Python ne fait pas appel à __str__ mais à la “représentation” repr...

```
def __repr__(self):
    return("objet Mammifere : {} ans - {}".format(self.age,self.cri))
```

```
In [9]: runcell(0, '/home/mrigo/exemple_classes.py')
[objet Mammifere : 17 ans - Hello, objet Mammifere : 1 ans - Ouuu]
```

“surcharge d’opérateurs”

```
    def __add__(self,other):
        return self.age + other.age

animal_1 = Mammifere("hello",15)
animal_2 = Mammifere("argh", 83)
print(animal_1+animal_2)
```

“surcharge d’opérateurs”

isinstance(instance,classe) → True / False

```
    def __add__(self,other):
        if isinstance(other,Mammifere):
            return self.age + other.age
        else:
            print("pas le bon type !")
            return 0

animal_1 = Mammifere("hello",15)
animal_2 = Mammifere("argh", 83)
print(animal_1 + 2)
```

+	__add__(self, other)
-	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
//	__floordiv__(self, other)
%	__mod__(self, other)
**	__pow__(self, other)
>>	__rshift__(self, other)
<<	__lshift__(self, other)
&	__and__(self, other)
|	__or__(self, other)
^	__xor__(self, other)

Pour les comparaisons :

__lt__(s,o) <
__gt__(s,o) >
__le__(s,o) <=
__ge__(s,o) >=
__eq__(s,o) ==
__ne__(s,o) !=

```
    def __lt__(self,other):
        if self.age<other.age:
            return True
        else:
            return False

animal_1 = Mammifere("hello",15)
animal_2 = Mammifere("argh", 83)
print(animal_1<animal_2)
```

Héritage : sous-classe

Tous les chiens sont des mammifères :
→ ils héritent donc des attributs et méthodes associées à tout mammifère

Tous les mammifères ne sont pas des chiens :
→ les chiens peuvent avoir des attributs et méthodes spécifiques

Chien est une *classe dérivée* de la classe Mammifere (*classe de base*)

```
class Chien(Mammifere):
    pass

animal_1 = Mammifere("hello",15)
animal_2 = Chien("argh", 6)
```

Héritage : méthodes spécifiques

```python
class Chien(Mammifere):

    def grogner(self):
        print("Grrrr Grrr Grrr")

animal_1 = Mammifere("hello", 15)
animal_2 = Chien("Wouf",2)
animal_2.parle()
animal_2.grogner()
print(issubclass(Chien, Mammifere))
animal_1.grogner()
```

```
je sais dire Wouf
Grrrr Grrr Grrr
True
Traceback (most recent call last):

  File "/home/mrigo/exemple_classes.py", line 51, in <module>
    animal_1.grogner()

AttributeError: 'Mammifere' object has no attribute 'grogner'
```

Héritage : surcharge de __init__

```python
class Chien(Mammifere):

    def __init__(self,age):
        Mammifere.__init__(self,"Wouf",age)

    def grogner(self):
        print("Grrrr Grrr Grrr")

animal_2 = Chien(2)
animal_2.parle()
```

```
je sais dire Wouf
```

On peut bien évidemment surcharger d’autres méthodes

Héritage : surcharge de __init__

```
class Chien(Mammifere):

    def __init__(self,age):
        Mammifere.__init__(self,"Wouf",age)

    def grogner(self):
        print("Grrrr Grrr Grrr")

animal_2 = Chien(2)
print(isinstance(animal_2,Chien))
print(isinstance(animal_2,Mammifere))
```

```
True
True
```

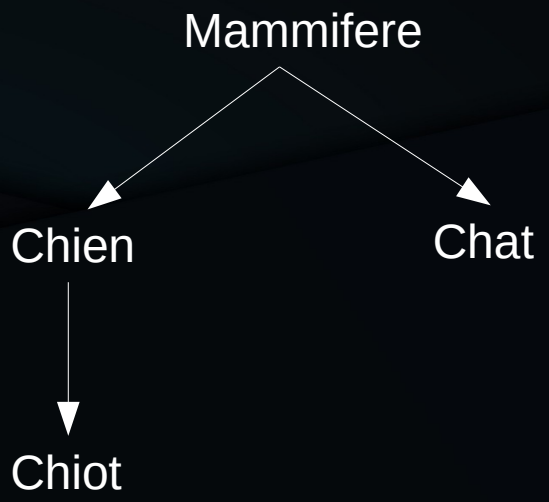
Lister les attributs

```
animal_2=Chien(3)
print(dir(animal_2))
```

```
['__add__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__',
'afficher_aptitudes', 'age', 'aptitudes', 'cri', 'grogner', 'nouvelle_aptitude',
'parle', 'supprimer_aptitude']
```

Multiplier les “niveaux”

```python
class Chien(Mammifere):

    def __init__(self,age):
        Mammifere.__init__(self,"Wouf",age)

    def grogner(self):
        print("Grrrr Grrr Grrr")

class Chat(Mammifere):
    pass

class Chiot(Chien):
    pass

animal_1=Mammifere("Hello", 17)
animal_2=Chat("Miaou",1)
animal_3=Chien(3)
animal_4=Chiot(0.5)
```



Une variable associée à la classe

```python
class Mammifere:

    compteur = 0

    def __init__(self,parole,ans):
        self.cri = parole
        self.age = ans
        self.aptitudes = []
        Mammifere.compteur += 1
```

```python
print("M", Mammifere.compteur)   # une variable de classe
animal_1=Mammifere("Hello", 17)
animal_2=Mammifere("Hello", 17)
print("1", animal_1.compteur)
print("2", animal_2.compteur)
print("M", Mammifere.compteur)

animal_1.compteur=6       # une variable de l'inst
print("1", animal_1.compteur)
print("2", animal_2.compteur)
print("M", Mammifere.compteur)

animal_3=Mammifere("Hello", 17)
print("1", animal_1.compteur)
print("M", Mammifere.compteur)
```

```
M 0
1 2
2 2
M 2
1 6
2 2
M 2
1 6
M 3
```

Convention : on utilise souvent __compteur