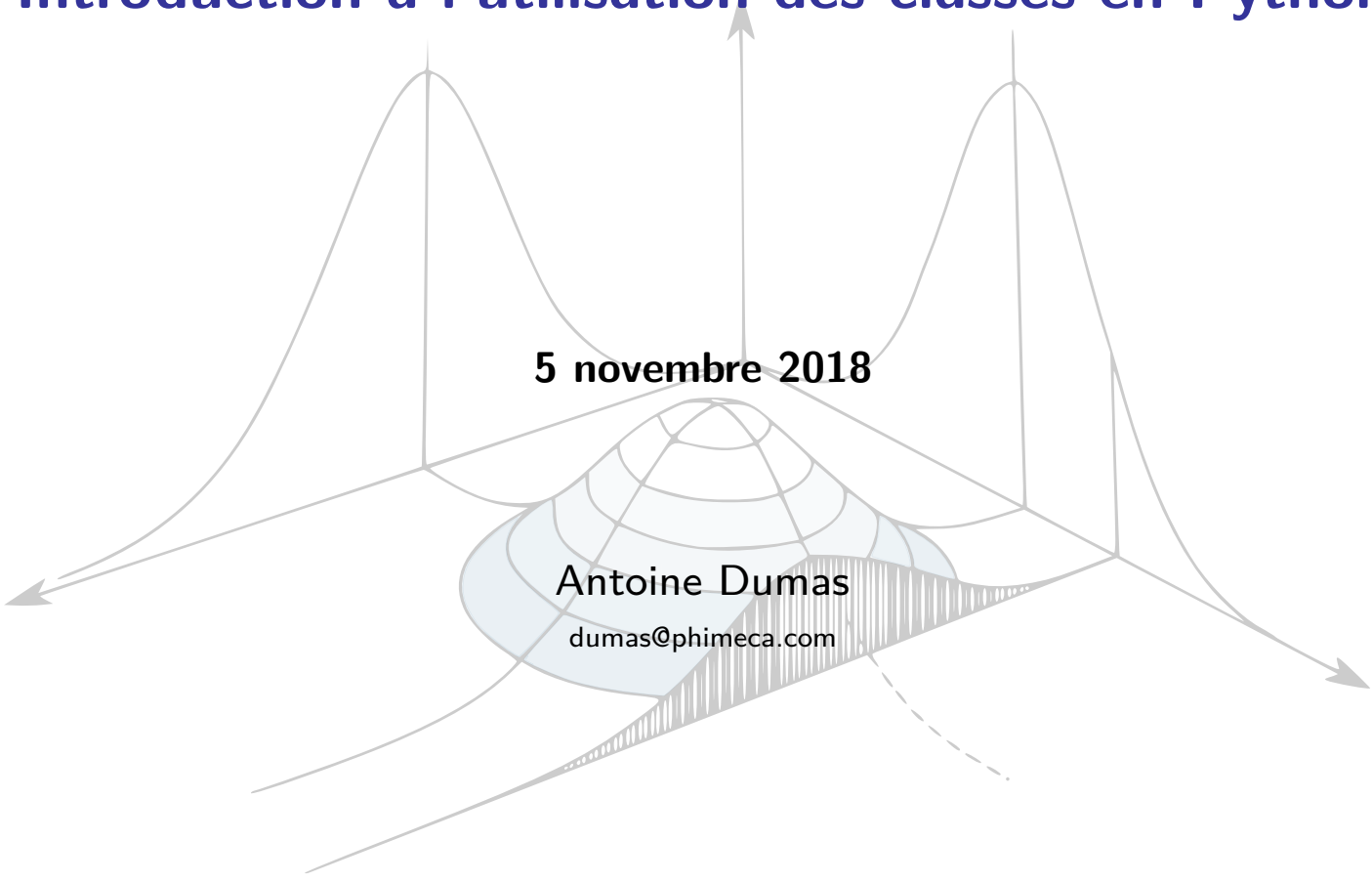
PHIMECA
Solutions for robust engineering

Introduction à l'utilisation des classes en Python



5 novembre 2018

Antoine Dumas
dumas@phimeca.com

Sommaire

Sommaire

Pourquoi les classes ?

- Le langage python est orienté objet, tout est objet. Il est donc naturel de développer des classes permettant d'utiliser cette propriété : on fait alors de la POO.
- Cela permet de structurer le code et de le rendre beaucoup plus lisible lors du développement de gros projet.
- Il est plus aisé de sauvegarder une instance d'une classe que plusieurs variables ou fonctions seules.
- L'implémentation des classes se rapproche du mécanisme en C++ : permet de définir des attributs, des méthodes, faire de l'héritage, surcharger des méthodes. Cependant en Python tout est public, et il n'y a pas besoin de déclarer le type des variables.

Sommaire

Implémenter une classe

La déclaration d'une classe a une forme similaire à la définition d'une fonction. Les mot-clés utilisés ici sont :

- `Class` : déclare une classe, par convention le nom est de type *CamelCase*.
- `self` : définit l'objet lui-même, il est obligatoire de le mettre comme premier paramètre des méthodes de la classe. Cependant, lors de l'appelle à la méthode, il n'est pas considéré comme un argument de la méthode.

```
class MyClass()
    def my_function(self):
        print('Hello world !')

>>> instance = MyClass()
>>> instance.my_function()
>>> 'Hello world !'
```

Implémenter une classe

Lors de l'instanciation (création d'un objet à partir d'une classe), il est possible de définir un état initial, à l'image d'un constructeur en C++.

- `__init__` : lors de l'instanciation d'un objet, cette méthode est automatiquement appelée. Il est possible de définir des attributs dans cette méthode, et même appeler des méthodes existantes.
- Les attributs peuvent être appelés et modifiés dans n'importe quelle méthode de la classe ainsi qu'à partir d'une instance de la classe.

```
class MyClass()
    def __init__(self):
        self.my_attribute = 3
        self.message = 'Hello world'

    def my_function(self):
        print(self.message)
```

```
>>> instance = MyClass()
>>> print(instance.my_attribute)
>>> 3
>>> instance.my_function()
>>> 'Hello world !'
>>> instance.message = 'New message'
>>> instance.my_function()
>>> 'New message'
```

Implémenter une classe

- Lors de la définition de la classe, des arguments peuvent être donnés, avec ou sans valeur par défaut.
- Il en va de même pour les méthodes des classes.

```
class MyClass()
    def __init__(self, message):
        self.message = message

    def my_function(self, second_message):
        print(self.message)
        print(second_message)

>>> instance = MyClass('Hello World !')
>>> instance.my_function('Hello again !')
>>> 'Hello world !'
>>> 'Hello again !'
```

Documenter une classe

- Une bonne habitude est de commenter vos classes et méthodes avec des *docstring*.
- La docstring est accessible depuis l'instance avec la méthode spéciale `__doc__` ou en faisant appel à l'aide (*help*).
- Plusieurs conventions de format existent, notamment le format numpydoc. Des mots clés spécifiques sont utilisés (`Parameters`, `Return`, `Notes`, `Examples`, ...), ce qui permet de construire des documentations html automatiquement.

```
class MyClass()
    '''This is my first class.

    Parameters
    ----------
    message : str
        The message to display.
    '''
        def __init__(self, message):
            self.message = message
```

```
def my_function(self, second_message):
    '''Print 2 messages.

    Parameters
    ----------
    second_message : str
        The second message to display.
    '''
    print(self.message)
    print(second_message)
```

Les méthodes et attributs privés

- Par défaut dans Python, rien n'est privé mais par convention les attributs et méthodes privées commencent par _.
- Toutefois, ils sont accessibles depuis une instance mais sa nature privée indique qu'ils ne devraient pas être modifiés ou appelés.

```
class ComputeSquare()
    def __init__(self):
        self._private_power = 2

    def _my_private_print(self, x, result):
        print('Square of {} = {}'.format(x, result))

    def eval(self, x):
        result =  x**self._private_power
        self._my_private_print(x, result)
```

```
>>> square = ComputeSquare()
>>> square.eval(3)
>>> 'Square of 3 = 9'
>>> square._private_power = 3
>>> square.eval(3)
>>> 'Square of 3 = 27'
```

Les attributs de classes

- Les attributs peuvent être définis directement dans la classe. La différence est qu'il ne sont pas propres à l'objet créé. C'est notamment utile pour créer un compteur du nombre d'instances créées d'une même classe.
- De la même manière on peut créer des méthodes de classes (en utilisant `cls` et `classmethod`) et des méthodes statiques (avec `staticmethod`) mais leur utilisation est plus rare.

```
class MyClass()

    compteur = 0
    def __init__(self):
        MyClass.compteur += 1

>>> MyClass.compteur
>>> 0
>>> instance1 = MyClass()
>>> MyClass.compteur
>>> 1
>>> instance2 = MyClass()
>>> MyClass.compteur
>>> 2
```

L'héritage

- L'héritage permet de créer des classes mères (ou de base) et où d'autres classes vont hériter de cette classe de base. L'héritage peut être simple (un seul héritage) ou multiple.
- Il est nécessaire d'initialiser la classe mère dans la classe fille.
- Les attributs et méthodes de la classe mère sont accessibles depuis les classes enfants. Python cherche d'abord dans la classe dont l'objet est directement issu puis récursivement dans les classes dont il hérite.
- Les classes enfants peuvent surcharger les méthodes dans leur propre classe afin de les modifier.

L'héritage, exemple

```
class LivingLife()
    def __init__(self, age):
        self.age = age
    def presentation(self):
        print('I am a Living Life and I am {}.'.format(self.age))

class HumanBeing(LivingLife)
    def __init__(self, age, name):
        # initialization of parent class
        LivingLife.__init__(self, age)
        self.name = name

    # overload of presentation
    def presentation(self):
        print('I am a Human Being name {} and I am {}.'.
              format(self.name, self.age))

>>> living_body = LivingLife(24)
>>> living_body.presentation()
>>> 'I am a Living Life and I am 24.'
>>> human = HumanBeing(34, 'Paul')
>>> human.presentation()
>>> 'I am a Human Being named Paul and I am 34.'
```

Pour aller plus loin

- Grâce à l'héritage vous pouvez créer vos propres exceptions, créer vos propres algorithmes et les intégrer à des bibliothèques existantes.
- Il existent un grand nombre de méthodes spéciales (`__methodespeciale__`). Elles peuvent servir à indiquer ce que Python doit faire dans différentes circonstances: quel affichage (`__repr__`, `__str__`), que faire si je modifie ou appelle un attribut (`__setattr__` et `__getattr__`), que faire si j'additionne 2 objets (`__add__`), ...
- Les classes vous permettront de structurer le code de façon claire.
- Le développement d'une ou plusieurs classes est un premier pas vers la création de modules Python.

Sommaire

Travaux pratiques

- L'objectif est de transformer vos développements (ou la correction) de la descente de gradient en une classe que vous pourrez ensuite appeler comme un module.
- Vous pouvez par exemple paramétrer la classe pour choisir la descente classique ou régularisée.
- Il faut développer des méthodes intermédiaires pour faire les calculs répétitifs (méthodes potentiellement privées).
- Les méthodes principales permettent de réaliser le calcul à proprement parler.
- Une fois la classe créée, enregistrez là dans un fichier python et importez ce module depuis jupyter notebook pour faire le calcul.