# W5-S5-C2-Complement-classes-new-style

December 14, 2014

## 1 Classes "new style"

### 1.1 Complément - niveau intermédiaire

#### 1.1.1 La classe `object`

Vous pouvez trouver du code dans lequel les classes héritent de la classe `object`, comme dans cet exemple tiré du module standard `zipfile` :

```
In []: from modtools import show_module
       import zipfile
       show_module(zipfile, beg='class ZipFile(', end="file:")
```

Ceci nous donne l'occasion de citer le module `zipfile`, qui permet de lire ou écrire, de manière transparente, des fichiers compressés au format `zip`.

#### 1.1.2 Les classes *new-style*

Mais le sujet de ce complément est d'expliquer pourquoi la classe `ZipFile` hérite de la classe `object`. Voici ce qu'il faut en retenir en version courte.

Aux alentours de la version 2.2 - 2.3 de python, le langage a été amélioré pour régler quelques problèmes qui existaient dans le système de types. En substance, et pour rester synthétique :

- Il n'était pas possible dans les anciennes versions de spécialiser un type prédéfini. Par exemple, si vous vous souvenez de la classe `collections.OrderedDict`, il s'agit d'une spécialisation du type *builtin* `dict`. Il n'aurait pas été possible d'implémenter cette classe avec les anciennes versions du langage.
- Dans l'ancien modèle mental, les classes et les types jouent un rôle différent. Or dans la logique d'un langage orienté objet, le type d'une instance, c'est sa classe. Nous allons y revenir avec des exemples.

Pour améliorer le langage, la notion de classe *new-style* a alors été introduite, et pour **ne pas casser la compatibilité ascendante**, il a été convenu que pour qu'une classe soit ***new-style***, il faut qu'elle **hérite** - directement ou indirectement - de la classe *builtin* `object`. Les classes qui n'héritent pas d'`object` sont appelées les *classes classiques*.

Nous reviendrons largement sur ces notions de types, de *classes classiques* et de classes *new-style* en semaine 7. Il faut retenir pour le moment qu'il y a en python 2 deux types différents de classes, les classes *new-style* et les *classes classiques* qui sont incompatibles. Cependant, pour la majorité des usages, le comportement de ces deux types de classes est le même. Les classes *new-style* sont utiles pour des usages avancés, comme l'héritage multiple sur lequel nous allons revenir dans un prochain complément, et nécessaires pour, par exemple, l'extension des types de bases.

Vous pouvez alors vous demander quand utiliser les classes *new-style* et les *classe classiques*. La réponse est simple : les classes *new-style* sont supérieures aux *classes classiques*, les classes *new-style* doivent donc être utilisées pour tous vos nouveaux programmes. D'ailleurs, en python 3, toutes les classes sont maintenant *new-style* par défaut (on n'a donc plus besoin d'hériter explicitement d'`object`) et les *classes classiques* n'existent plus. Attention cependant si vous travaillez avec du code python 2 existant qui utilise des *classes*

*classiques*, il est recommandé dans ce cas de continuer à utiliser les classes classiques. En effet, même si pour la majorité des usages, les classes classiques et *new-style* ont le même comportement, dans certains cas avancés, elles ont un comportement incompatible.

Toutes les classes de la librairie standard python sont des classes *new-style*, c'est pourquoi la classe `ZipFile` hérite de `object`.

### 1.1.3 Illustration

Une instance d'une *classe classique* a pour type le type `instance` :

```
In []: # une classe classique
       class OldStyle:
           pass
       # une instance
       old_style = OldStyle()
       # son type est juste 'instance'
       type(old_style)
```

Par contre, une instance d'une classe *new-style* a pour type la classe qu'on a utilisée pour créer l'objet :

```
In []: # une classe new-style : elle hérite de 'object'
       class NewStyle(object):
           pass
       # une instance
       new_style = NewStyle()
       # le type de l'instance est bien la classe
       type(new_style) is NewStyle
```

## 1.2 Complément - niveau avancé

### 1.2.1 Pour en savoir plus

Si ce sujet vous intéresse, vous pouvez commencer par l'article initial de Guido Van Rossum au sujet de la nouvelle implémentation.

D'autres liens pertinents sont aussi donnés ici dans la documentation python.