Python
Méthodes spéciales (suite)

Nicolas Delestre

INSA
ROUEN NORMANDIE

Identité et égalité

Rappels

- Les variables Python référencent des objets : on obtient la référence d'un objet grâce à la fonction standard `id`
- Deux variables peuvent référencer le même objet (utilisation de l'affectation) : on compare les références des objets grâce à l'opérateur `is`
- Deux objets peuvent être égaux : les attributs les caractérisant sont égaux
- L'opérateur == permet de tester l'égalité. Par défaut pour Python, cette égalité repose sur l'identité
- Pour les types de base, ce comportement a été redéfini

Illustration

```
>>> class A:
        pass
>>> a1=A()
>>> a2=A()
>>> a3=a1
```

```
>>> a1 is a2
False
>>> a1 is a3
True
>>> a1==a2
False
>>> a1==a3
True
```

```
>>> x=1.0
>>> y=1.0
>>> x is y
False
>>> x==y
True
```

Attention aux int

```
>>> a=256
>>> b=256
>>> a is b
True
>>> a=257
>>> b=257
>>> a is b
False
```

Les méthodes spéciales __eq__ et __ne__ 1 / 2

La méthode __eq__

- C'est la méthode qui est exécutée lorsque l'opérateur == est utilisé
  Ainsi l'interprétation `a == b` invoque `a.__eq__(b)`
- Code de `__eq__(self, autre)` (bonne pratique) :
  - Tester si le paramètre est un instance de la classe de `self` :
    `isinstance(autre, LaClasseDeSelf)` ou `isinstance(autre, self.__class__)`
  - Tester les égalités des attributs qui définissent l'égalité
  - Renvoyer `NotImplemented` si l'opérateur d'égalité n'a pas à être utilisé

La méthode __ne__

- C'est la méthode qui est exécutée lorsque l'opérateur `!=` est utilisé
- Par défaut `__ne__` renvoie la négation de l'appel à `__eq__` (sauf si le résultat est `NotImplemented`)

Les méthodes spéciales __eq__ et __ne__ 2 / 2

Point2D

```
def __eq__(self, autre):
    if not isinstance(autre, self.__class__):
        return False
    else:
        return self.x == autre.x and self.y == autre.y
```

Point3D

```
def __eq__(self, autre):
    if not isinstance(autre, self.__class__):
        return False
    else:
        return super().__eq__(autre) and self.z == autre.z
```

ROUEN NORMANDIE

Les méthodes spéciales de comparaison d'ordre

`__ge__`, `__gt__`, ...

- Les méthodes spéciales sont :
  - `__ge__` pour l'opérateur $\geq$
  - `__gt__` pour l'opérateur $>$
  - `__le__` pour l'opérateur $\leq$
  - `__lt__` pour l'opérateur $<$
- Le fait de développer `__lt__` (inversement `__gt__`) n'oblige pas à développer `__gt__` (inversement `__lt__`)

Attention

- Il n'y a pas de lien entre `__le__` ou `__ge__` et `__lt__`, `__gt__` et `__eq__` : par exemple le fait de coder uniquement `__lt__` et `__eq__` ne permet d'utiliser l'opérateur `<=`
- Il n'y a pas de lien entre `__le__` et `__ge__`
- Le décorateur `total_ordering` du module standard `funtools` permet de lever ces limites

La méthode spéciale `__hash__` 1 / 2

Rappels

- L'empreinte (*hash*) d'un objet est un nombre (inférieur à une certaine borne) qui le caractérise :
  - Deux objets égaux doivent avoir la même empreinte
  - Deux objets différents devraient avoir deux empreintes différentes (mais il peut y avoir des collisions)

La méthode `__hash__`

- En Python on obtient l'empreinte d'un objet en appelant la fonction standard `hash` qui appelle la méthode spéciale `__hash__`
- Si la méthode `__hash__` n'est pas définie, la fonction `hash` lève une exception `TypeError`
- Si la méthode `__hash__` est définie, on dit que les objets de cette classe sont *hashable*
- Pour qu'un objet puisse être utilisé comme clé d'un `dict`, ou ajouté à un set ou à un `frozenset`, il faut qu'il soit *hashable*

La méthode spéciale `__hash__` 2 / 2

Bonnes pratiques

- Dans le cas ou l'on veut que les objets soient *hashables* :
  - on définit `__hash__` en combinant les empreintes des attributs immuables de l'objet (par exemple en calculant le *hash* des additions des *hash* des attributs immuables)
  - la méthode `__eq__` doit être redéfinie : `x == y` doit impliquer `hash(x) == hash(y)`

INSA ROUEN NORMANDIE

Objet appelable

- Un appelable (*callable*) est un objet qui peut etre utilisé comme une fonction
- Tout objet dont la classe possède la méthode `__call__(self[,arg, ...])` est un appelable

Point2D

```
def __call__(self, x, y):
    self.x = x
    self.y = y
```

```
>>> from point import Point2D
>>> pt=Point2D(1,2)
>>> pt(3,4)
>>> pt.x
3
```

Conclusion

- Rappels : variables, références aux objets, identité, égalité, empreinte
- Les méthodes spéciales pour les comparaisons
- Les méthodes spéciales qui permettent aux instances d'une classe d'être :
  - *hashables*
  - appelables

INSA ROUEN NORMANDIE