Python
Méthodes spéciales

Nicolas Delestre

INSA
ROUEN NORMANDIE

Les méthodes spéciales

- Ensemble de méthodes systèmes à définir ou redéfinir pour :
  - **définir le comportement de l'interpéteur au regard des objets (par exemple comment les représenter textuellement)**
  - **de pouvoir utiliser des opérateurs ou fonctions standards avec les objets**
  - définir le comportement des objets dans certaines situations (comme par exemple dans les dictionnaires)
- Ces méthodes ont comme identifiant `__XX__`
  - `__init__` est la méthode spéciale exécutée pour initialiser une nouvelle instance

INSA ROUEN NORMANDIE

Les méthodes `__repr__` et `__str__`

La méthode `__repr__`

- Appelée par la fonction `repr`
- C'est la méthode qui est exécutée lorsque l'interpréteur affiche l'objet
- Représentation textuelle formelle (« informatique ») d'un objet (pourrait être utilisée pour créer un nouvel objet)

Sans redéfinition de `__repr__`

```
>>> class A:
    ....pass
>>> a=A()
>>> a
<__main__.A object at 0x7f13bbd4d9d0>
```

Classe `Point2D` du module `point.py`

```
32    def __repr__(self):
33        if self.id:
34            return f"Point2D({self.x}, {self.y}, '{self.id}')"
35        return f"Point2D({self.x}, {self.y})"
```

```
>>> pt=Point2D(1,2,'A')
>>> pt
Point2D(1, 2, 'A')
```

```
>>> pt1=eval(repr(pt))
>>> pt1
Point2D(1, 2, 'A')
```

Les méthodes __repr__ et __str__

La méthode __str__

- Appelée par la fonction str
- Représentation textuelle informelle (« humaine ») d'un objet
- C'est la méthode qui est exécutée lorsqu'on affiche l'objet ou que l'on crée une chaîne formatée

Classe Point2D du module point.py

```
1    def __str__(self):
2        prefixe = self.id
3        if not prefixe:
4            prefixe = ""
5        return prefixe + f"({self.x},{self.y})"
```

```
>>> f"{pt}"
'A(1, 2)'
```

- Si __repr__ est définie mais non __str__ alors la représentation informelle utilise __repr__ (inverse non vrai)

Opérateurs (fonctions) binaires 1 / 4

Utilisation des opérateurs +, -, etc.

- Il est possible de créer des classes dont les instances pourront être utilisées comme opérandes des opérateurs « arithmétiques », par exemple des classes : Vecteur, Matrice, Fonction (mathématique), etc.

```
>>> from vecteur import Vecteur2D
>>> u = Vecteur2D(1,2,'u')
>>> v = Vecteur2D(2,3,'v')
>>> u+v
Vecteur2D(3, 5, 'u+v')
>>> u-v
Vecteur2D(-1, -1, 'u-v')
```

INSA ROUEN NORMANDIE

Opérateurs (fonctions) binaires 2 / 4

Méthodes spéciales `__add__`, `__sub__`, etc.

- Il faut pour cela définir les méthodes spéciales :
  - `__add__(self, autre)` pour l'opérateur `+`
  - `__sub__(self, autre)` pour l'opérateur `-`
  - `__mul__(self, autre)` pour l'opérateur `*`
  - `__matmul__(self, autre)` pour l'opérateur `@`
  - `__pow__(self, autre[, modulo])` pour l'opérateur `**` ou la fonction `pow`
  - `__truediv__(self, autre)` pour l'opérateur `/`
  - `__floordiv__(self, autre)` pour l'opérateur `//`
  - `__mod__(self, autre)` pour l'opérateur `%`
  - `__divmod__(self, autre)` pour la fonction `divmod`
  - `__lshift__(self, autre)` pour l'opérateur `<<`
  - `__rshift__(self, autre)` pour l'opérateur `>>`
  - `__and__(self, autre)` pour l'opérateur `and`
  - `__or__(self, autre)` pour l'opérateur `or`
  - `__xor__(self, autre)` pour l'opérateur `xor`
- Pour interpréter `x + y`, Python invoque `x.__add__(y)`

- Si le type de l'argument (l'opérande droite de l'opérateur) n'est pas compatible avec l'objet, la valeur retournée doit être `NotImplemented`

Opérateurs (fonctions) binaires 3 / 4

Classe Vecteur2D du module vecteur.py

```
5  class Vecteur2D:
6      def __init__(self, x: float, y:float, identifiant: str):
7          self._x = x
8          self._y = y
9          self._id = identifiant
10
11     @property
12     def x(self) -> float:
13         return self._x
14
15     @property
16     def y(self) -> float:
17         return self._y
18
19     @property
20     def id(self) -> str:
21         return self._id
```

```
38     def __add__(self, vecteur_2d: Vecteur2D) -> Vecteur2D:
39         if not isinstance(vecteur_2d, Vecteur2D):
40             return NotImplemented
41         return Vecteur2D(self.x + vecteur_2d.x,
42                          self.y + vecteur_2d.y,
43                          self.id + "+" + vecteur_2d.id)
44
45     def __sub__(self, vecteur_2d: Vecteur2D) -> Vecteur2D:
46         if not isinstance(vecteur_2d, Vecteur2D):
47             return NotImplemented
48         return Vecteur2D(self.x - vecteur_2d.x,
49                          self.y - vecteur_2d.y,
50                          self.id + "-" + vecteur_2d.id)
51
52     def __mul__(self, vecteur_2d: Vecteur2D) -> float:
53         if not isinstance(vecteur_2d, Vecteur2D):
54             return NotImplemented
55         return self.x*vecteur_2d.x + self.y*vecteur_2d.y
```

INSA ROUEN NORMANDIE

Opérateurs (fonctions) binaires 4 / 4

Méthodes spéciales réflexives : `__radd__`, `__rsub__`, etc.

- Si `x + y` est utilisée et que la classe de `x` ne possède pas la méthode `__add__` ou qu'elle retourne `NotImplemented`, Python essaye d'interpréter `y.__radd__(x)`
- Si la classe de `y` ne définit la méthode `__radd__(self, other)` ou qu'elle retourne `NotImplemented` alors une exception `TypeError` est levée

Méthodes spéciales pour les opérations arithmétiques augmentées : `__iadd__`, `__isub__`, etc.

- Utilisées pour les opérateurs arithmétiques associés à l'affectation : l'objet doit donc être mutable
- Si `x += y` (équivalent à `x = x + y`) est utilisée alors Python interprète `x.__iadd__(y)`

INSA ROUEN NORMANDIE

Opérateurs (fonctions) unaires

- Pour utiliser les opérateurs unaires +, -, etc. il faut définir les méthodes spéciales :
  - `__pos__(self)` pour l'opérateur +
  - `__neg__(self)` pour l'opérateur -
  - `__abs__(self)` pour la fonction abs
  - `__invert__(self)` pour l'opérateur ~

Classe `Point2D` du module `point.py`

```
43    def __abs__(self):
44        return Point2D(abs(self.x), abs(self.y))
```

```
>>> abs(Point2D(-3,2))
Point2D(3,2)
```

Fonctions de transtypage numériques

int, float, etc.

- L'utilisation d'une fonction de transtypage numérique appelle une méthode spéciale
  - `__int__(self)` pour la fonction `int`
  - `__round__(self)` pour la fonction `round`
  - `__float__(self)` pour la fonction `float`
  - `__complex__(self)` pour la fonction `complex`

Classe Point2D du module point.py

```
46    def __complex__(self):
47        return complex(self.x, self.y)
```

```
>>> complex(Point2D(-3,2))
(-3+2j)
```

Conclusion

Nous avons vu dans ce cours

- Le rôle des méthodes spéciales
- Deux représentations des objets sous forme de chaine de caractères
- Comment permettre l'utilisation des opérateurs et fonctions usuels

Rappel : principe général de python

Le concepteur/développeur d'une API écrit plus de code pour simplifier la vie de l'utilisateur

INSA ROUEN NORMANDIE