Python
Les énumérations

Nicolas Delestre

INSA
ROUEN NORMANDIE

Rappels

Définitions

- Définition générale : ≪ Un type énuméré (appelé souvent énumération ou juste enum, parfois type énumératif ou liste énumérative) est un type de données qui consiste en un ensemble de valeurs constantes ≫ (Wikipédia)

- Dans la cadre de Python : ≪ Une énumération est un ensemble de noms symboliques, appelés membres, liés à des valeurs constantes et uniques. Au sein d'une énumération, les membres peuvent être comparés entre eux et il est possible d'itérer sur l'énumération elle-même. ≫ (Documentation Python)

INSA ROUEN NORMANDIE

Création d'une énumération

- La création d'une énumération est réalisée par la création d'une classe possédant autant d'attributs de classe qu'il y a d'éléments dans l'énumération
- On crée cette classe énumération à partir de la classe `Enum` qui est proposée par le module standard `enum`
- Deux méthodes de création :
  1. par héritage de la classe `Enum`
  2. par instanciation de la classe `Enum` (qui retourne une classe)

INSA ROUEN NORMANDIE

Création par héritage 1 / 2

- Création d'une sous classe de la classe `Enum` avec déclaration d'un attribut de classe pour chaque élément de l'énumération
  - la bonne pratique veut que les identifiants des attributs soient en majuscule
  - on peut ajouter/redéfinir des méthodes
  - l'héritage d'une énumération est autorisée si la super classe (sous classe d'`Enum`) n'a défini aucun attribut de classe
- À chaque élément de l'énumération est associé une valeur, qui peut être de tout type (`int`, `str`, etc.). Cette valeur peut être attribuée automatiquement en utilisant des instances de la classe `auto` (à partir de python 3.6)

```
class Couleur(Enum):
    BLANC = 1
    NOIR = 2
    BLEU = 3
    ROUGE = 4
    VERT = 5
```

```
class Jour(Enum):
    LUNDI = auto()
    MARDI = auto()
    MERCREDI = auto()
    JEUDI = auto()
    VENDREDI = auto()
    SAMEDI = auto()
    DIMANCHE = auto()
```

Création par héritage 2 / 2

- Deux éléments (attributs de classe) peuvent avoir la même valeur (sauf si utilisation du décorateur de classe `unique`) : le deuxième est dit *alias* du premier

```
class JourEntreprise(Enum):
    OUVRE = 1
    FERIE= 2
    DIMANCHE = 2
```

```
>>> from enum import unique
>>> @unique
... class ERREUR(Enum):
...     VAL1 = 1
...     VAL2 = 2
...     ALIAS_VAL2 = 2
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
  ...
    raise ValueError('duplicate values found in %r: %s' %
ValueError: duplicate values found in <enum 'ERREUR'>: ALIAS_VAL2 -> VAL2
```

INSA ROUEN NORMANDIE

Création par instanciation de la classe Enum

- L'instanciation de la classe `Enum` retourne une classe énumération
- L'initialiseur admet comme paramètre formel (deux premiers obligatoires) :
  - `value` : le nom de la classe énumération à créer (la bonne pratique veut que ce nom soit identique à l'identifiant de la variable qui va référencer la classe)
  - `names` : une chaîne de caractères (éléments séparés par une virgule et/ou des espaces), ou une séquence listant le nom des éléments, ou un dictionnaire
  - `module` : le nom du module de la classe créée
  - `qualname` : la position de la classe dans le module
  - `type` : type des valeurs des éléments
  - `start` : valeur du premier élément (à partir de Python 3.5)
- Par défaut, les valeurs sont des `int` et la valeur du premier élément vaut 1
- Inconvénient : il est impossible d'ajouter ou de redéfinir des méthodes

```
JourDeLaSemaine = Enum("JourDeLaSemaine", "LUNDI MARDI MERCREDI JEUDI VENDREDI SAMEDI DIMANCHE")
Couleur = Enum("Couleur", {"BLANC":"A", "NOIR":"B", "BLEU":"C", "ROUGE":"D", "VERT":"F"})
```

Utilisation d'une énumération

- On peut utiliser une classe énumération :
  - en utilisant l'attribut de classe
  - en « instanciant » la classe
  - en utilisant la classe comme un dictionnaire (les clés sont les identifiants des attributs de classe en chaîne de caractères)
  - en utilisant la classe comme une séquence (indexée par le nom des valeurs)
- Les instances ont deux attributs : `value` et `name`

```
>>> from couleur import Couleur
>>> Couleur.BLANC
<Couleur.BLANC: 1>
>>> Couleur(1)
<Couleur.BLANC: 1>
>>> Couleur['BLANC']
<Couleur.BLANC: 1>
```

```
>>> c1=Couleur(1)
>>> c2=Couleur(1)
>>> c1 is c2
True
```

```
>>> for c in Couleur:
...     print(c)
...
Couleur.BLANC
Couleur.NOIR
Couleur.BLEU
Couleur.ROUGE
Couleur.VERT
```

```
>>> c=Couleur(1)
>>> c.value
1
>>> c.name
'BLANC'
```

Autres classes proposées par le module enum 1 / 3

IntEnum

≪ Classe de base pour créer une énumération de constantes qui sont également des sous-classes de int ≫ (documentation Python)

Extrait de https://stackoverflow.com/questions/52929954/difference-between-enum-and-intenum-in-python

```
>>> class Shape(IntEnum):
...     CIRCLE = 1
...     SQUARE = 2
...
>>> class Color(Enum):
...     RED = 1
...     GREEN = 2
...
```

```
>>> Shape.CIRCLE == Color.RED
False
>>> Shape.CIRCLE == 1
True
>>> Shape.CIRCLE+Shape.SQUARE
3
>>> Color.RED+Color.GREEN
--------------------------------------------------------------------------
TypeError
...
TypeError: unsupported operand type(s) for +: 'Color' and 'Color'
```

Flag

≪ Classe de base pour créer une énumération de constantes pouvant être combinées avec des opérateurs de comparaison bit-à-bit, sans perdre leur qualité de Flag ≫ (à partir de Python 3.6, documentation Python)

Extrait de http://zetcode.com/python/enum/

```
>>> class Perm(Flag):
...     EXECUTE = auto()
...     WRITE = auto()
...     READ = auto()
...
```

```
>>> list(Perm)
[<Perm.EXECUTE: 1>, <Perm.WRITE: 2>, <Perm.READ: 4>]
>>> Perm.EXECUTE | Perm.READ
<Perm.READ|EXECUTE: 5>
>>> Perm.EXECUTE & Perm.READ
<Perm.0: 0>
```

ROUEN NORMANDIE

Autres classes proposées par le module enum 3 / 3

IntFlag

≪ Classe de base pour créer une énumération de constantes pouvant être combinées avec des opérateurs de comparaison bit-à-bit, sans perdre leur qualité de IntFlag. Les membres de IntFlag sont aussi des sous-classes de int. ≫ (à partir de Python 3.6, documentation Python)

Extrait de http://zetcode.com/python/enum/

```
>>> class Perm(IntFlag):
...     EXECUTE = auto()
...     WRITE = auto()
...     READ = auto()
...
```

```
>>> list(Perm)
[<Perm.EXECUTE: 1>, <Perm.WRITE: 2>, <Perm.READ: 4>]
>>> Perm.EXECUTE | Perm.READ
<Perm.READ|EXECUTE: 5>
>>> Perm.EXECUTE + Perm.READ
5
```

ROUEN NORMANDIE

Conclusion

Nous avons vu :

- des rappels sur les énumérations
- qu'est ce qu'une énumération en Python
- les deux méthodes pour créer une énumération : par héritage ou par instanciation de la classe `Enum`
- comment utiliser une énumération
- trois autres types d'énumération à étendre proposés par `enum` : `IntEnum`, `Flag` et `FlagEnum`

INSA ROUEN NORMANDIE