Curso de Python

Artículo

Introducción al módulo collections

David Aroesti 23 PlatziRank 17 de Octubre de 2018

El módulo collections nos brinda un conjunto de objetos primitivos que nos permiten extender el comportamiento de las built-in collections que poseé Python y nos otorga estructuras de datos adicionales. Por ejemplo, si queremos extender el comportamiento de un diccionario, podemos extender la clase UserDict; para el caso de una lista, extendemos UserList; y para el caso de strings, utilizamos UserString.

Por ejemplo, si queremos tener el comportamiento de un diccionario podemos escribir el siguiente código:

```
class SecretDict(collections.UserDict):

    def _password_is_valid(self, password):
        ...

    def _get_item(self, key):
        ...
```

```
            if self._password_is_valid(password):
                return self._get_item(key)

            return None

my_secret_dict = SecretDict(...)
my_secret_dict['some_password:some_key'] # si el password es válido, regresa el valor
```

Otra estructura de datos que vale la pena analizar, es namedtuple. Hasta ahora, has utilizado tuples que permiten acceder a sus valores a través de índices. Sin embargo, en ocasiones es importante poder nombrar elementos (en vez de utilizar posiciones) para acceder a valores y no queremos crear una clase ya que únicamente necesitamos un contenedor de valores y no comportamiento.

```
Coffee = collections.NamedTuple('Coffee', ('size', 'bean', 'price'))
def get_coffee(coffee_type):
     If coffee_type == 'houseblend':
         return Coffee('large', 'premium', 10)
```

El módulo collections también nos ofrece otros primitivos que tienen la labor de facilitarnos la creación y manipulación de colecciones en Python. Por ejemplo, Counter nos permite contar de manera eficiente ocurrencias en cualquier iterable; OrderedDict nos permite crear diccionarios que poseen un orden explícito; deque nos permite crear filas (para pilas podemos utilizar la lista).




Python comprehensions