



Artículo

Introducción al módulo collections

**David Aroesti** ⌚ 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 posee 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):  
        ...  
  
    def __getitem__(self, key):  
        password, key = key.split(':')  
  
        if self._password_is_valid(password):  
            return self._get_item(key)  
  
        return None  
  
my_secret_dict = SecretDict(...)
```



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).

En conclusión, el módulo `collections` es una gran fuente de utilerías que nos permiten escribir código más “pythonico” y más eficiente.



Escribe aquí tu pregunta

+ 2



wilantury Estudiante • hace 20 días

<https://docs.python.org/3/library/collections.html?highlight=collections#collections.namedtuple>



2



Jann Haider Ramos Andrade Estudiante • hace 27 días

Counter me pareció muy interesante, funciona genial para hacer estadística pues en tres simples pasos podemos organizar la información de los datos y crear un diccionario para guardar el valor y la moda:



Python comprehensions

2



David Martínez Jiménez Estudiante • hace 2 meses

No me parece de lo más adecuado que estando viendo programación estructurada con python y se ponga de trancazo una explicación y código orientado a objetos, esto puede causar mucha confusión al momento de leerlo y más a los programadores que apenas están aprendiendo

13



Rayan Renato Figueroa Asencios • hace 2 meses

conuerdo contigo, yo recién estoy aprendiendo y la verdad no entendí nada.

2



Edwin Jesset Barrientos Gonzales Estudiante • hace 2 meses

Interesante modulo, a ponerlos en practica

1



Adrián Andrés Sosa Estudiante • hace 2 meses

24 I'll have to try it

25 1

26 Santiago Jose Ibarra Aquino Estudiante • hace 3 meses

Me parece muy bueno esta libreria ya que es bastane util para ciertos tipos de tareas

27 1

28 innacroft Estudiante • hace 3 meses

29 No me gustó, . deberían hacer un video, sería más explicativo...

30 13

31



Python comprehensions

32

2

33



Maclovinhm Estudiante • hace 3 meses

¿Esté modulo solo sirve para diccionarios?



1



Esau Rosales Estudiante • hace 5 meses

Este post me ayudo a entender el tema:

[Módulo Collections](#)



12



freddymm • hace 13 días

Muy buena la pagina!

1



thejb Estudiante • hace 7 meses

Es un poco dificil de entender el concepto, pero aca les dejo algo que me sirvio muchisimo



2



Facundo Nicolás García Martoni • hace 7 meses

¡Hola @thejb! Al parecer hubo un error y no se puede ver lo que compartiste 😞 ¿Podrías hacerlo de nuevo?

4



thejb Estudiante • hace 7 meses

Esto no queda bien claro, me perdí, y creo que es bastante avanzado, para lo que venimos aquí a aprender a programar, se me esta siendo muy difícil el curso... porque aveces el profe aplica varios temas avanzados, alguien me ayuda con este modulo?



3



Python comprehensions

3



Sergio Andres Rubiano Rodriguez · hace 5 meses

Hola, es bueno contarte que hay un curso anterior a este de python, si no lo haz tomando, te recomiendo que lo tomes

2

[Ver más respuestas](#)

Rodolfo Nicacio Ugalde Ochoa Estudiante · hace 8 meses

Me costo entender este concepto. Creo que seria más entendible si siguieran enseñando como en los videos.
Les dejo mi aporte a lo que entendi(no es todo) pero si me sirvio de mucho lo saque de otra página. Saludos.

- :link: [[collections – Container datatypes](https://docs.python.org/3/library/collections.html)]
(<https://docs.python.org/3/library/collections.html>)

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.

1. ``namedtuple()`` factory function for creating tuple subclasses with named fields
2. ``deque`` list-like container with fast appends and pops on either end
3. ``ChainMap`` dict-like class for creating a single view of multiple mappings
4. ``Counter`` dict subclass for counting hashable objects
5. ``OrderedDict`` dict subclass that remembers the order entries were added
6. ``defaultdict`` dict subclass that calls a factory function to supply missing values
7. ``UserDict`` wrapper around dictionary objects for easier dict subclassing
8. ``UserList`` wrapper around list objects for easier list subclassing
9. ``UserString`` wrapper around string objects for easier string subclassing

Counter

```
```python
from collections import Counter
```

```
list = [1,2,3,4,1,2,6,7,3,8,1]
Counter(list)
Counter({1: 3, 2: 2, 3: 2, 4: 1, 6: 1, 7: 1, 8: 1})
```



Python comprehensions

```
list = [1,2,3,4,1,2,6,7,3,8,1]
cnt = Counter(list)
print(cnt[1])
Output:
3
```

Counter tiene tres funciones adicionales.

1. Elements
2. Most\_common([n])
3. Subtract([iterable-or-mapping])

*elements()*

```
cnt = Counter({1:3,2:4})
print(list(cnt.elements()))
Output:
[1, 1, 1, 2, 2, 2, 2]
```

*most\_common()*

```
list = [1,2,3,4,1,2,6,7,3,8,1]
cnt = Counter(list)
print(cnt.most_common())
Output:
[(1, 3), (2, 2), (3, 2), (4, 1), (6, 1), (7, 1), (8, 1)]
```

*subtract*

```
cnt = Counter({1:3,2:4})
deduct = {1:1, 2:2}
cnt.subtract(deduct)
print(cnt)
Output:
Counter({1: 2, 2: 2})
```

## defaultdict

Crear diccionarios con el constructor defaultdict()

```
from collections import defaultdict
```

```
nums = defaultdict(int)
nums['one'] = 1
```



Python comprehensions

```
-
print(nums['two'])
Output:
2

count = defaultdict(int)
names_list = "Mike John Mike Anna Mike John John Mike Mike Britney Smith Anna
Smith".split()
for names in names_list:
 count[names] +=1
print(count)
Output:
defaultdict(<class 'int'>, {'Mike': 5, 'Britney': 1, 'John': 3, 'Smith': 2, 'Anna':
2})
```

## OrderedDict

Ordenar diccionario(s)

```
from collections import OrderedDict
```

```
od = OrderedDict()
od['a'] = 1
od['b'] = 2
od['c'] = 3
print(od)
Output:
OrderedDict([('a', 1), ('b', 2), ('c', 3)])
```

```
for key, value in od.items():
 print(key, value)
Output
a 1
b 2
c 3
```

## deque

```
from collections import deque
```

```
list = ["a","b","c"]
deq = deque(list)
```



```

deq.append("d")
deq.appendleft("e")
print(deq)
Output
deque(['e', 'a', 'b', 'c', 'd'])

```

```

deq.pop()
deq.popleft()
print(deq)
Output
deque(['a', 'b', 'c'])

```

```

list = ["a","b","c"]
deq = deque(list)
print(deq)
print(deq.clear())
Output
deque(['a', 'b', 'c'])
None

```

```

list = ["a","b","c"]
deq = deque(list)
print(deq.count("a"))
Output
1

```

## <h5>ChainMap</h5>

```

from collections import ChainMap

```

```

dict1 = { 'a' : 1, 'b' : 2 }
dict2 = { 'c' : 3, 'b' : 4 }
chain_map = ChainMap(dict1, dict2)
print(chain_map.maps)
Output
[{'b': 2, 'a': 1}, {'c': 3, 'b': 4}]

```

```

dict2['c'] = 5
print(chain_map.maps)
Output
[{'b': 2, 'a': 1}, {'c': 5, 'b': 4}]

```





```
dict1 = { 'a' : 1, 'b' : 2 }
dict2 = { 'c' : 3, 'b' : 4 }
chain_map = ChainMap(dict1, dict2)
print (list(chain_map.keys()))
print (list(chain_map.values()))
Output
['b', 'a', 'c']
[2, 1, 3]

dict3 = {'e' : 5, 'f' : 6}
new_chain_map = chain_map.new_child(dict3)
print(new_chain_map)
Output
ChainMap({'f': 6, 'e': 5}, {'a': 1, 'b': 2}, {'b': 4, 'c': 3})
```

## <h5>namedtuple()</h5>

```
from collections import namedtuple

Student = namedtuple('Student', 'fname, lname, age')
s1 = Student('John', 'Clarke', '13')
print(s1)
print(s1.fname)
Output
Jhon
Student(fname='John', lname='Clarke', age='13')
```

## Creando namedtuple usando una lista

```
s2 = Student._make(['Adam', 'joe', '18'])
print(s2)
Output
Student(fname='Adam', lname='joe', age='18')
```

## Creando una nueva instancia usando una instancia existente

```
s2 = s1._asdict()
print(s2)
Output
OrderedDict([('fname', 'John'), ('lname', 'Clarke'), ('age', '13')])
```



```
s2 = s1._replace(age='14')
print(s1)
print(s2)
Output
Student(fname='John', lname='Clarke', age='13')
Student(fname='John', lname='Clarke', age='14')
```



16



u.mitchel · hace 7 meses

Wowww excelente aporte, muchas gracias, no había logrado entender, pero tu lo solucionaste todo.

2



Genaro Flores · hace 4 meses

Buen aporte, se agradece bastante:).

1



Edwin Hernandez · el mes pasado

Excelente aporte:

Inicialmente no me gustaban las tuplas ya que su acceso normal es por índice y creaba una clase, pero con namedtuple se me hace una pasada, muy útil cuando solo necesitas un contenedor.

```
from collections import namedtuple

student = namedtuple("Student", "name, lastname, age")
s1 = student("Edwin", "Perez", "13")
s2 = student("David", "Lopez", "15")
students = []
students.append(s1)
students.append(s2)
for student in students:
 print(f"Name: {student.name} Lastname: {student.lastname} Age: {student.age}")````
```



Python comprehensions



mauriciovicenciomolina Estudiante • hace 9 meses

No entiendo a que te refieres con "Comportamiento de un diccionario"



2



Facundo Nicolás García Martoni • hace 6 meses

¡Hola Mauricio! Con esto el profesor se refiere a extender las funcionalidades que tiene un diccionario normal a partir de un módulo de Python 😊

2



Jose Oliva Rivera Estudiante • hace 9 meses

Existe alguna clase referente a este tema, despues de este punto?



1



Facundo Nicolás García Martoni • hace 7 meses

Si quieres, puedes visitar estos enlaces sobre el módulo collections para ampliar el tema:

- [Documentación oficial](#)
- [Introducción a collections](#)

1



Luis Eduardo Sotoj Teque Estudiante • hace 10 meses

David, podrias explicar mas a detalle etae tema por favor, habemos varios que no hemos entendido el tema.

Gracias,



1



Facundo Nicolás García Martoni • hace 6 meses

¡Hola @lestjs! ¿Qué es lo que no entendiste? Te dejo este [link](#) para que puedas revisar una explicación distinta a esta sobre el módulo **collections**

1



Python comprehensions



3



Diego Alexander Forero Higuera · hace 10 meses

collections es un módulo que tiene Python y permite extender las funcionalidades de los diccionarios para crear tus propios diccionarios con funcionalidades muy específicas, otro concepto que se enseña aquí son NumedTuples que es muy similar a lo que se conoce en php como array asociativo.

2



Facundo Nicolás García Martoni · hace 6 meses

¡Hola @Beto92! ¿Qué es lo que no entendiste? Te dejo este [link](#) para que puedas revisar una explicación distinta a esta sobre el módulo **collections**

1

[Ver más respuestas](#)

Victor Inojosa Estudiante · el año pasado

Me gustó mucho el Counter ... por acá les dejo algunas de las cosas que ví en el `help(collections.Counter)`:

- Saber cuales son los N elementos que más apariciones tiene en la lista con el método `most_common()`
- Establecer un elemento en cero pero que se mantenga en la lista de elementos (asignación a cero)

```
import collections

my_string = 'abracadabra'

c = collections.Counter(my_string)

print(c.most_common(3)) #returned [('a', 5), ('r', 2), ('b', 2), ('c', 1), ('d', 1)]

print(c['b'])#returned2
c['b'] = 0#remove all 'b' but keep the element. Returned nothing
print(c)
```



2



IsmaelFajardo · hace 11 meses



Python comprehensions

1



Félix Alejandro Zelaya Orellana Estudiante · el año pasado

Genial!!



1



edoar17 Estudiante · el año pasado

Alguien que aclare por favor el primer ejemplo.

Por que a las funciones se les pasa como parametro

(self, password)?

no deberia ser:

(self, self['password'])??

porque "password" y "key" son keys del diccionario que se la pasa como parametro a la clase al crear my\_secret\_dict? Y para acceder a ellas hay que usar el ['password']?



2



Raul Flores · el año pasado

```
class SecretDict(collections.UserDict):

 def _password_is_valid(self, password):
 ...

 def _get_item(self, key):
 ...

 def __getitem__(self, key):
 password, key = key.split(':')

 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
```

Fijate en la funcion `getitem()`, ahí está separando la key del diccionario en dos partes por un lado la key en si y por el otro el password  
Al hacer esto



Python comprehensions

ejecuta `getitem()` donde separa el valor introducido en el diccionario en dos valores

```
esta linea separa la llave introducida anteriormente
'some_password:some_key'
password, key = key.split(':')

password = 'some _password'
key = 'some_key'

y luego la verifica en'''
if self._password_is_valid(password):
 return self._get_item(key)
```

---

1



Saúl Báez Terrez Estudiante · el año pasado

Esto hubiera estado mejor en un vídeo, y no solo un texto, la metodología que usan esta dejando mucho que desear. Meten un texto a modo de introducción y vienen clases algo que no han abordado anteriormente Mal!



13



Adolam · el año pasado

Toda la razón.

---

2



Angel de Jesus Quintero Pereira · hace 11 meses

Estoy de acuerdo contigo.

También cabe recalcar que: Él coloca un ejemplo en Orientación a objetos y el ni si quiera se ha tomado la molestia de explicar la **sintaxis** orientación a objetos en python. **(Ojo no hablo de explicar la filosofía o paradigmas de la orientación a objetos, hablo de explicar la sintaxis).**

Se que muchos dirán que eso lo puede investigar, por internet. Y eso es verdad, pero creo que si el va a citar un ejemplo en orientación a objetos es preciso que el curso lo de entonces la orientación objetos en python

Creo que el curso tiene que ordenarse mejor:

A) Un poco de contexto de python

B) Orientado a funciones

1. Sintaxis y semantica (lo necesario, no tiene que ser todo)

2. Estructuras y tipos de datos. (clases prácticas con cada estructura y tipos de datos)

C) Orientación a objetos

1. sintaxis y semantica (lo necesario, no tiene que ser todo)



Python comprehensions

### 3. Mención de algunas librerías, (esto es para saber, cuales vamos a usar) D) Proyecto.

3



AxIGz • hace 10 meses

Estoy de acuerdo, es como si el curso estuviera organizado mal.

2



Facundo Nicolás García Martoni • hace 6 meses

¡Hola! Muchas gracias por sus comentarios, sepan que nos sirven mucho para mejorar Platzi cada día. Esta situación va a mejorar en una nueva actualización del curso de Python 😊

1

[Ver más respuestas](#)

Iraida Mercedes Barreto Díaz Estudiante • el año pasado

No entiendo algo, y es ese parámetro self. 😞 ¿Alguien me puede explicar por favor?



5



jadry92 • el año pasado

La palabra reservada *self* se utiliza para la creación de los métodos y variables dentro de una clase y el uso de ellos dentro la misma clase. En otros lenguajes como Java se usa la palabra *this*.

Ejemplo:

Una clase para manejar números complejos:

```
>>> classComplex:
... def __init__(self, realpart, imagpart):
... self.r = realpart
... self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

[Fuente](#)

9



Python comprehensions

2

[Ver más respuestas](#)

Roberto Sobarzo Estudiante · el año pasado

Buenísimo. Sólo un poco complejo al comienzo. Espero ir aclarando mis dudas en el camino.



2



Facundo Nicolás García Martoni · hace 6 meses

¡Hola Roberto! ¿Qué es lo que no entendiste? Te dejo este [link](#) para que puedas revisar una explicación distinta a esta sobre el módulo **collections**

1



Nestor Cepeda Platzi Team · el año pasado

Para abstraer un poquito:

```
>>> from collections import namedtuple

>>> Point = namedtuple('Point', ['x', 'y']) # Define namedtuple
>>> p = Point(10, y=20) # Creating an object
>>> p
Point(x=10, y=20)
>>> p.x + p.y
30
>>> p[0] + p[1] # Accessing the values in normal way
30
>>> x, y = p # Unpacking the tuple
>>> x
10
>>> y
20
```



35



joseangelatm19 · el año pasado

Crack!



Python comprehensions



Si vienen de ruby es muy parecido a lo que puedes hacer con Struct

---

2

[Ver más respuestas](#)



Rhonal Velasco Estudiante • hace 2 años

Muy interesante. Cada vez me gusta más Python

---



7



Python comprehensions