

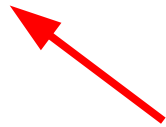
Métodos mágicos en Python 3

Jesús Cea Avión

jcea@jcea.es

@jcea

<https://www.jcea.es/>



- Métodos que permiten definir o alterar comportamientos aparentemente implícitos.
- Alteración de clases estándar.
- Implementación de protocolos del lenguaje.

```
>>> import this
```

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

[...]

Readability counts.

[...]

In the face of ambiguity, refuse the temptation to guess.

```
>>> dir(int)
```

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
 '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
 '__floor__', '__floordiv__', '__format__', '__ge__', '__getattr__',
 '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__int__',
 '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__',
 '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__',
 '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__',
 '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
 '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__',
 '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
 '__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_length',
 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real',
 'to_bytes']
```

```
>>> help(int.__lt__)
```

```
__lt__(self, value, /)
    Return self<value.
```

```
>>> class intX(int) :  
...     def __lt__(self, v) :  
...         return True  
...     def __gt__(self, v) :  
...         return True
```

```
>>> a=intX(10)
```

```
>>> a
```

```
10
```

```
>>> a<5
```

```
True
```

```
>>> a>20
```

```
True
```

```
>>> a<a
```

```
True
```

```
>>> a>a
```

```
True
```

```
>>> a<=5
```

```
False
```

```
>>> a>=20
```

```
False
```

```
>>> class extender(int) :  
...     def __mul__(self, v) :  
...         return v * int('1'*self)
```

```
>>> a=extender(4)
```

```
>>> a
```

```
4
```

```
>>> 3*a
```

```
12
```

```
>>> a*3
```

```
3333
```

```
>>> a*a
```

```
1234321
```

Si cambiamos a `self*'1'`:
RuntimeError: maximum recursion depth exceeded while calling a Python object

- Interoperatividad de tipos. **Abstract Base Classes.**
- Un “dir” muestra los métodos mágicos definidos, pero no todos los posibles:

```
>>> a = 5; a +=1; print(a)
```

```
6
```

```
>>> a.__iadd__
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
AttributeError: 'int' object has no attribute '__iadd__'
```

- El Zen de Python es una guía valiosa:
 - Belleza y elegancia.
 - Evitar sorpresas.
 - Explícito mejor que implícito.
 - Legibilidad.

```
vector = vector1 + vector2
```

```
vector = 5 * vector1
```

```
dotprod = vector1*vector2
```

```
if poly1 < poly2 :
```

```
poly = poly1 * poly2
```

```
If user1 in user2.amigos :
```

```
user2.amigos += user1
```

```
vector= vector1.add(vector2)
```

```
vector = vector1.resize(5)
```

```
escalar = vector1.dotprod(v2)
```

```
if poly1.area < poly2.area :
```

```
poly = poly1.intersect(poly2)
```

```
if user2.is_amigo(user1) :
```

```
user2.add_amigo(user1)
```

- Aritméticos: `mul`, `abs`, `add`, `neg`, `float`, `ceil`, `lshift`, ...
 - Para “`a*b`”, “`b*a`” y “`a *=b`”.
- Lógicos: `and`, `or`, `xor`, `not`, `lt`, `le`, `gt`, `ge`, `eq`, `ne`, ...
- Conversión: `float`, `format`, `repr`, `str`, `bytes`, `int`, `bool`, ...
- Clase: `class`, `doc`, `new`, `subclasscheck`, `slots`, ...
- Gestión de instancias: `init`, `del`, `isinstancecheck`, ...
- Interacción: `hash`, `getattr`, `getattribute`, `setattr`, `delattr`, `dir`, `call`, `len`, `getitem`, `setitem`, `delitem`, `iter`, `contains`, `reversed`, ...

- Context Managers: `enter`, `exit`.

with open("file", "r") as f :

- Descriptores: `set`, `get`, `delete`.

vector.x = 5 if poligono.area < 10 :

- Copiar objetos: `copy`, `deepcopy`.

copy.copy(objeto)

- Pickling: `getinitargs`, `getnewargs`, `getstate`, `setstate`, `reduce`, `reduce_ex`.

pickle.dumps(grafo)

- Varios: `sizeof`

sys.getsizeof(objeto)

Si hay tiempo:

- `__del__`: Ciclos.
- `__new__`: Singleton.
- `__slots__`: Weakrefs.
- `__copy__`, `__deepcopy__`: objetos inmutables.

- Python 3.4.2 documentation – 3. Data model – 3.3. Special method names
<https://docs.python.org/3/reference/datamodel.html#special-method-names>
- A Guide to Python's Magic Methods
<http://www.rafekettler.com/magicmethods.html>
- Dive into Python 3 – Special Method Names
<http://www.diveintopython3.net/special-method-names.html>
- No nos metemos con atributos mágicos como:
 - `__name__`
 - `__qualname__`
 - `__class__`
 - `__doc__`
 - `__dict__`
 - `__weakref__`
 - ...

Lo siguiente queda
como ejercicio para casa :)
(¡muahahahaha!)

Pickle:

- <https://docs.python.org/3/library/pickle.html>.
- Ojo, deserializar SOLO desde fuentes seguras: `find_class()`.
- Métodos mágicos: `getinitargs()`, `getnewargs()`, `getnewargs_ex()`, `getstate()`, `setstate()`, `reduce()`, `reduce_ex()`.
- Object DB: `persistent_id()`, `persistent_load()`.
- No todo es serializable, aunque “`getstate()`” ayuda. Ejemplo ROCKS.

Metaclasses: Métodos mágicos en Python 3

- `__prepare__`: Inicializa el “namespace”. Por ejemplo, diccionario ordenado, prohibir métodos duplicados, verificar APIs.

```
class meta(type):
```

```
    def __prepare__(name, bases, **kwds):
```

```
        class dictNoDups(dict):
```

```
            def __setitem__(self, k, v):
```

```
                if k in self:
```

```
                    raise RuntimeError('¡Nombre duplicado! %s' %k)
```

```
                return super().__setitem__(k, v)
```

```
        return dictNoDups()
```

```
class ejemplo(meta):
```

```
    def a(self):
```

```
        pass
```

```
    def b(self):
```

```
        pass
```

```
    a = 5
```

```
Traceback (most recent call last):
```

```
File "z.py", line 10, in <module>
```

```
class ejemplo(meta):
```

```
File "z.py", line 15, in ejemplo
```

```
a = 5
```

```
File "z.py", line 6, in __setitem__
```

```
raise RuntimeError('¡Nombre duplicado! %s' %k)
```

```
RuntimeError: ¡Nombre duplicado! a
```

Cosas que te pueden morder el culo (la vida es dura)

- Limitación: los métodos mágicos deben definirse a nivel de clase (es decir, en el tipo), no de instancia (consistencia interna del intérprete):

```
>>> class obj :  
...     def __len__(self) :  
...         return 3  
...  
>>> a = obj()  
>>> a.__len__ = lambda : 5  
>>> len(a)  
3
```


- Normalmente también ignoran `__getattr__()` (velocidad a costa de flexibilidad):

```
>>> class obj :
...     def __getattr__(*dummy) :
...         1/0
...
>>> a = obj()
>>> len(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'obj' has no len()

>>> a.abc
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in __getattr__
ZeroDivisionError: division by zero
```

Veda de preguntas abierta
:-) :-) :-)