

Metaclasses 😱

Ayudantía 8



¡Las clases también son objetos!

*En Python y lenguajes similares como Ruby.

**No nos cansaremos de repetir que todo es objeto.

Las clases son objetos

```
class ClaseCualquiera:  
    pass
```

```
misma_clase = ClaseCualquiera
```

```
instancia = misma_clase()
```

Se asigna la misma clase a la variable `misma_clase`.

`misma_clase` NO será un objeto distinto. Es otra referencia (nombre) para exactamente la misma clase.

Recién aquí se instancia un objeto de la `ClaseCualquiera`

Clases *on-the-fly*

Las clases se pueden crear en una línea de la siguiente manera:

```
UnaClase = type("UnaClase",  
                (SuperClase,),  
                {"imprimir": lambda self, s: print(s),  
                 "nombre": "John Doe"})
```

Nombre de la clase

Tupla con las
superclases

Diccionario con los
atributos y
métodos de la
clase

Recuerda que las
funciones también
son objetos que
pueden ir en un
diccionario

Clases *on-the-fly*

Las clases se pueden crear en una línea de la siguiente manera:

```
UnaClase = type("UnaClase",  
                (SuperClase,),  
                {"imprimir": lambda self, s: print(s),  
                 "nombre": "John Doe"})
```

Si algún día necesitan crear varias clases, lo pueden automatizar usando un ciclo y la creación *on-the-fly*.

(Sobre las tuplas,)

En Python, los paréntesis sirven para delimitar cualquier cosa. Por ejemplo, un string de varias líneas:

```
texto = (“Este es un string que cumple con ”  
        “PEP8 porque las líneas del código no ”  
        “superan los 80 caracteres.”)
```

La variable `texto` **no es una tupla** que contiene un string, es solo un string.

```
tupla = (“Hola”, )  
no_tupla = (“RIP”)
```

**Las clases son
instancias de
metaclases**

*Les dijimos que eran objetos

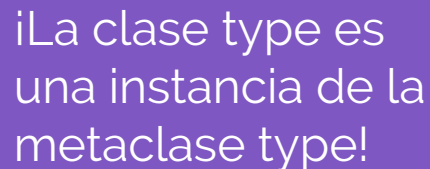
Metacalse

Clase cuyas instancias son clases.
Es la clase de las clases.

- La metacalse por default es la clase *type*.
 - Las metaclasses heredan de la clase *type*.
 - Toda clase es, al menos indirectamente, instancia de la clase *type*.
-

¡Pueden hacer la prueba!

```
print(type(int))  
>> <class 'int'>  
print(type(type(int)))  
>> <class 'type'>  
print(type(type))  
>> <class 'type'>
```



¡La clase type es
una instancia de la
metaclase type!

La metacalse por *default* es type

Es necesario especificar la metacalse:

```
class ClaseCualquiera(metaclass=SuperMetaclass):  
    pass
```

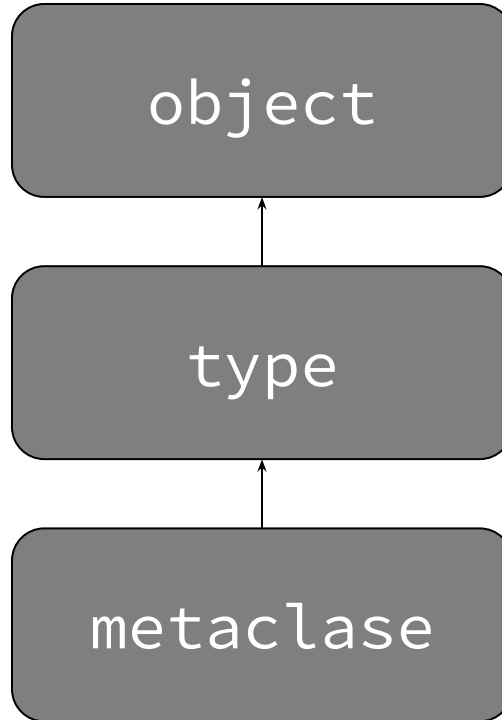
¡Después de especificar las clases desde donde hereda!

Por default la metacalse será type

```
class ClaseCualquiera(metaclass=type):  
    pass
```

Esto es innecesario

**Las metaclases heredan de `type`
y toda clase es indirectamente instancia de `type`**



La pregunta del millón...

¿Y esto cómo afecta a Boca?

OK, te creo.

¿Cómo comienzo a crear metaclasses?

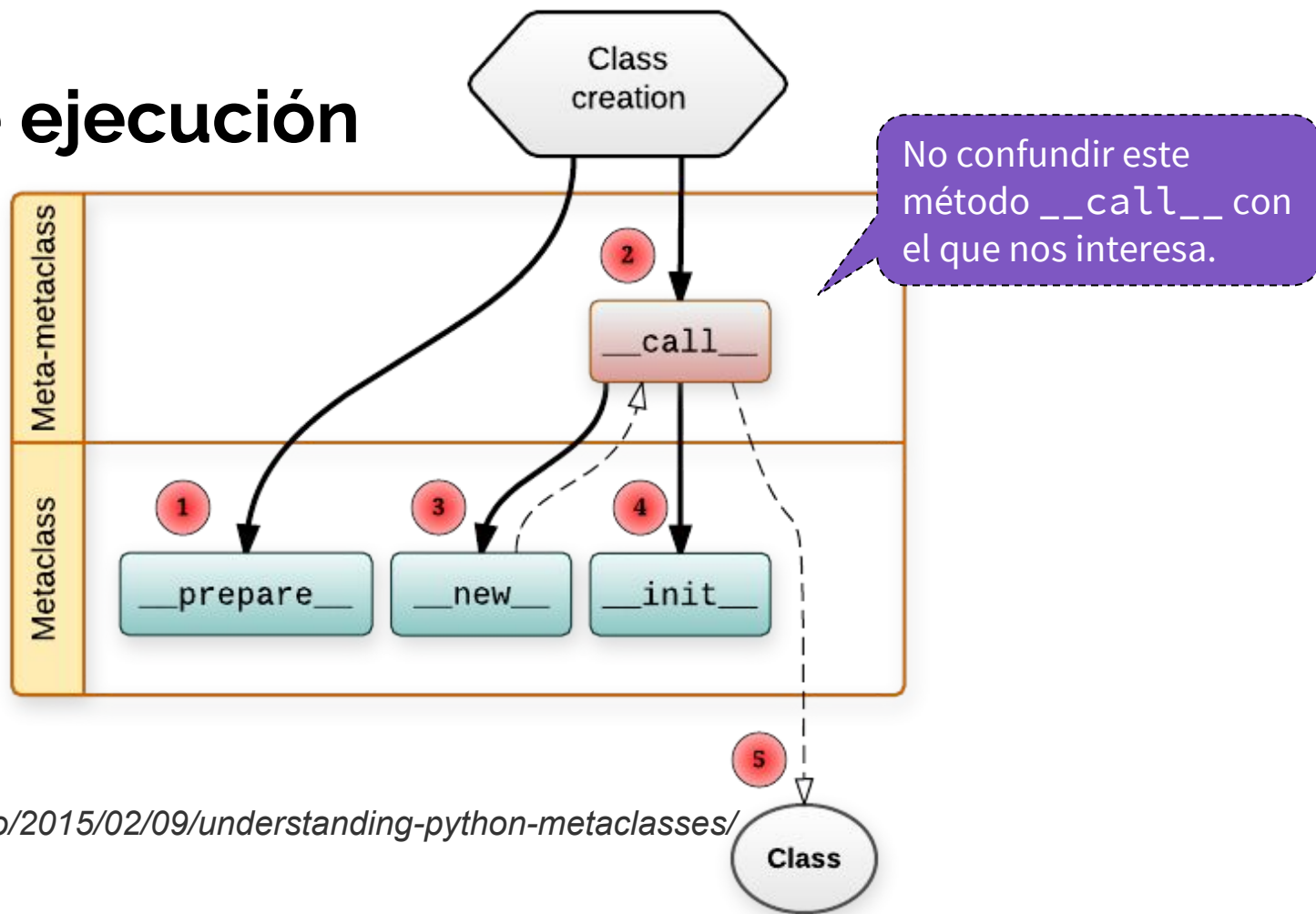
~~__prepare__()~~
__new__()
__init__()
__call__()

¡No se verá
evaluado!

Sintaxis básica de una metaclasa

```
class Metaclass(type):  
    def __new__(mcs, name, bases, attrs):  
        return super().__new__(mcs, name, bases, attrs)  
  
    def __init__(cls, name, bases, attrs):  
        super().__init__(name, bases, attrs)  
  
    def __call__(cls, *args, **kwargs):  
        return super().__call__(*args, **kwargs)
```

Orden de ejecución



Fuente:

<https://blog.ionelmc.ro/2015/02/09/understanding-python-metaclasses/>

`__new__(mcs, name, bases, attrs)`

mcs: La metaclasses. Puede entenderse como el “self” que estás acostumbrado a usar.

name: El nombre de la clase a ser creada.

bases: Una tupla de todas las superclases de la clase a ser creada.

attrs: El diccionario de atributos de la clase a ser creada. Es importante entender que con atributos nos referimos también a los métodos de la clase. ¡Hagan la prueba!

```
class A:
    atributo_de_A = 1
    def __init__(self):
        self.atributo_de_instancia_de_A = 2
```

```
a = A()
print(A.__dict__)
print(a.__dict__)
```

Atributos de la
clase A

Atributos de la
instancia a

¡Son distintos!

__init__(cls, name, bases, attrs)

cls: La clase a ser creada.

name: El nombre de la clase a ser creada.

bases: Una tupla de todas las superclases de la clase a ser creada.

attrs: El diccionario de atributos de la clase a ser creada.

¿Diferencia entre `__new__` y `__init__`?

La principal diferencia es que `__init__` recibe el argumento `cls` que hace referencia a la clase que se está creando, es decir, **la clase en este punto ya existe** y ya no podemos realizar cambios muy profundos en ella. Entonces, la parte más interesante (y enredada) se encuentra en `__new__`, donde podemos cambiar el nombre e incluso las clases de las cuales hereda la clase que estamos creando

**¡Aplicaciones interesantes se pueden encontrar
en el material de Metaclases!**

`__call__(cls, *args, **kwargs)`

cls: La clase.

args*, kwargs:** Son los argumentos que recibe el método `__init__` de la clase.

En este punto la clase ya existe en su totalidad. Este método es llamado cuando realizamos una instancia de la clase, es decir, `a = A()`. Puede que estén familiarizados con él como un método que se agrega a las clases para que sus instancias sean “ejecutables”, análogamente, `__call__` es un método que se define en las metaclasses para que sus instancias (las clases) sean “ejecutables”.

**Este tema es complicado y debe ser
estudiado con sumo cuidado.
¡Denle vueltas!**

*Encontrarán ejercicios en el material

**Las issues del Syllabus también permiten preguntas sobre la materia.