Objetos en CPython

Jesús Espino García

jesus.espino@kaleidos.net
@jespinog

PyConES 2013

Kaleidos
beautiful code

24 de Noviembre de 2013

Introducción

- Python 3.3
- Usaré ctypes para los ejemplos.
- La estructura de un objeto en cpython.
- Los objetos escritos en c de python.
- El proceso de creación de un nuevo objeto.

¿Qué es un tipo en python?

- Un tipo es una clase
- Es una estructura compuesta de datos y slots
- Los slots son punteros a funciones que definen comportamientos
- Los tipos son objetos de python
- Los tipos son de objetos de tipo tipo

¿Qué es un tipo en python?

```
>>> class Prueba:
...   pass
...
>>> type(Prueba)
<class 'type'>
>>> isinstance(Prueba, object)
True
>>> isinstance(type, object)
True
>>> type(type)
<class 'type'>
```

¿Qué es una instancia?

- Es exactamente lo mismo que un objeto.
- Es una zona reservada de la memoria con datos.
- Tiene un tipo (y solo 1) que determina qué puede hacer el objeto.
- El tipo de un objeto no cambia a lo largo de su vida (existen excepciones).

¿Qué es una instancia?

```
>>> prueba = Prueba()
>>> type(prueba)
<class '__main__.Prueba'>
>>> prueba
<__main__.Prueba object at 0x7f3555af9bd0>
>>> Prueba
<class '__main__.Prueba'>
>>> id(prueba)
139867047566288
>>> hex(id(prueba))
'0x7f3555af9bd0'
```

Diagrama de herencia

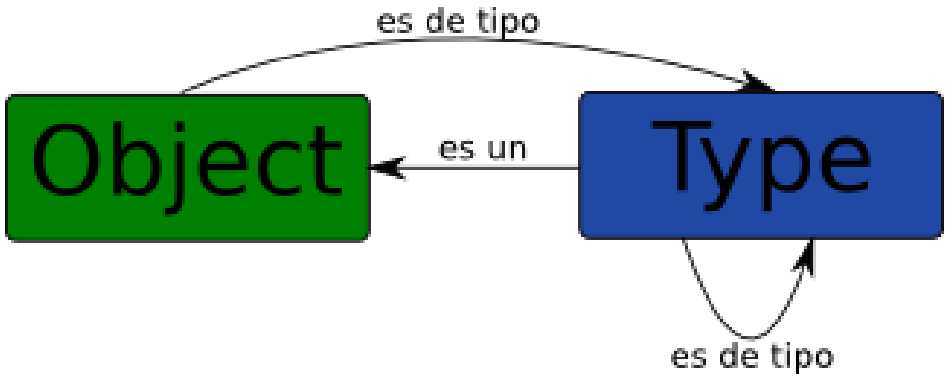


Diagrama de herencia

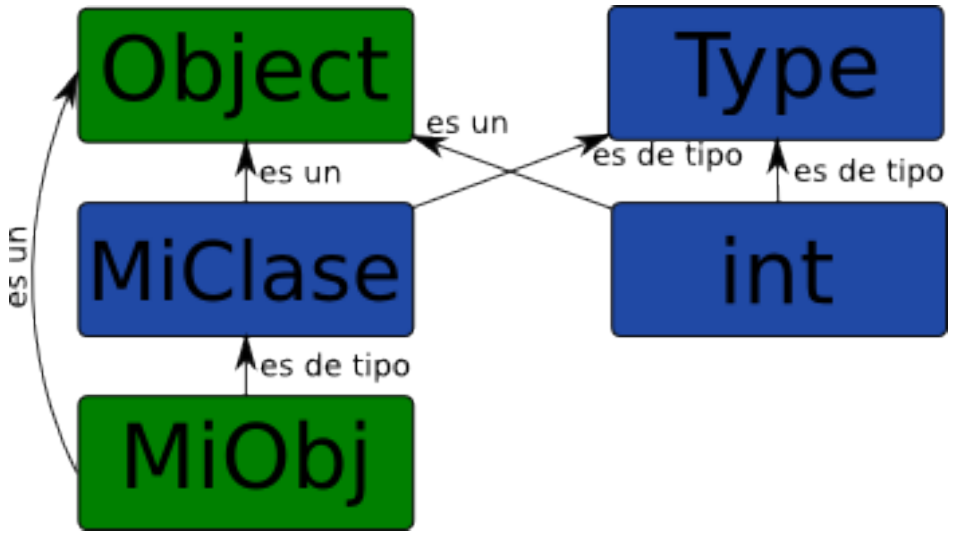
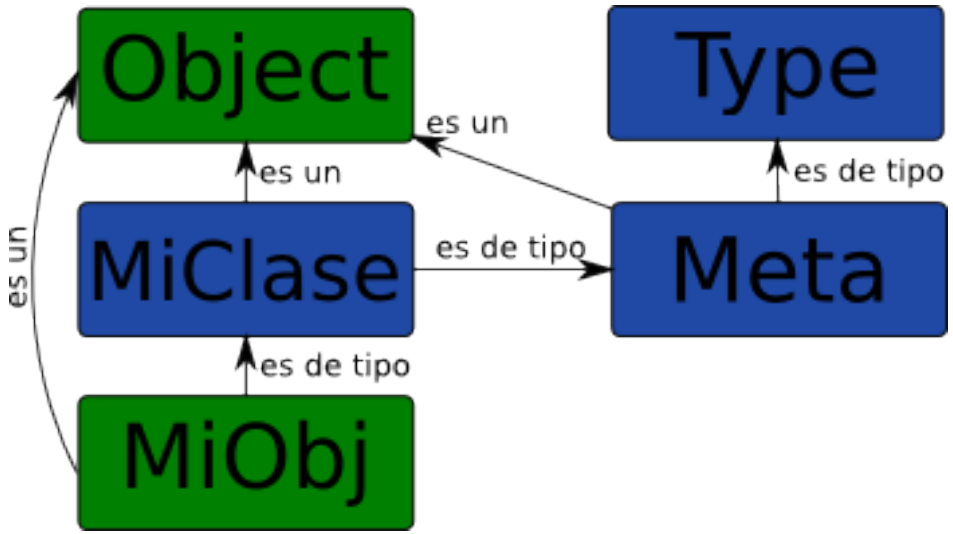
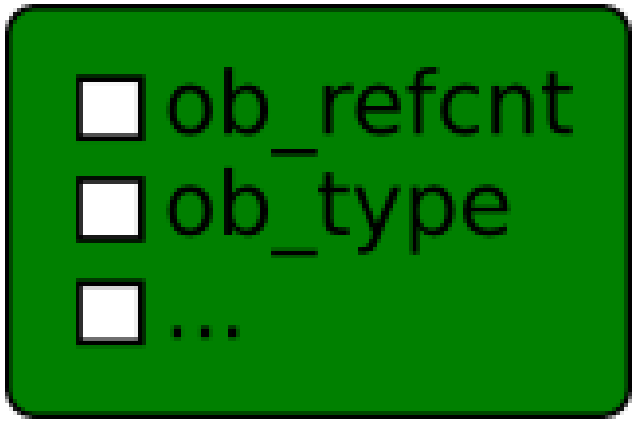


Diagrama de herencia

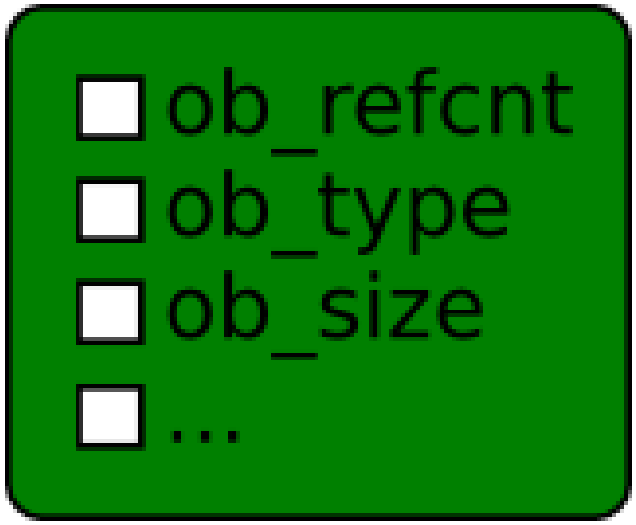


Estructura básica de un objetos



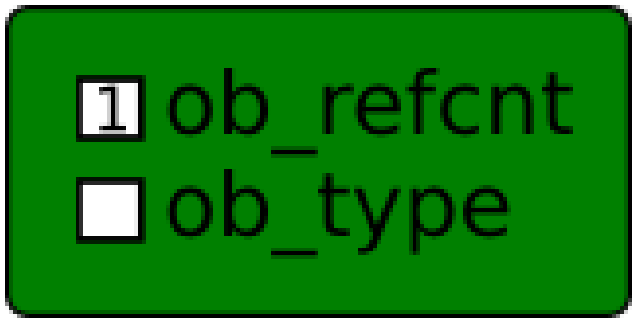
- `ob_refcnt`: contador de referencias
- `ob_type`: puntero al tipo de datos
- Otros datos específicos para este tipo

Estructura variable básica de un objetos



- `ob_refcnt`: contador de referencias
- `ob_type`: puntero al tipo de datos
- `ob_size`: tamaño del objeto
- Otros datos específicos para este tipo

El objeto None

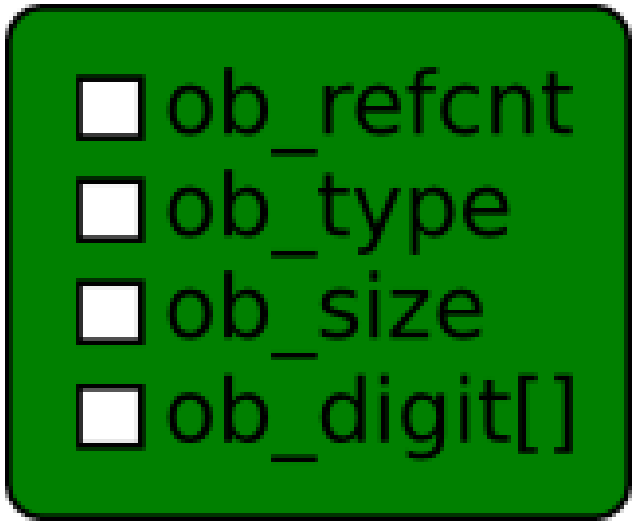


- Es el tipo más simple
- Su instancia es singleton
- No añade ningún dato extra a la estructura básica de objeto

Very bad things

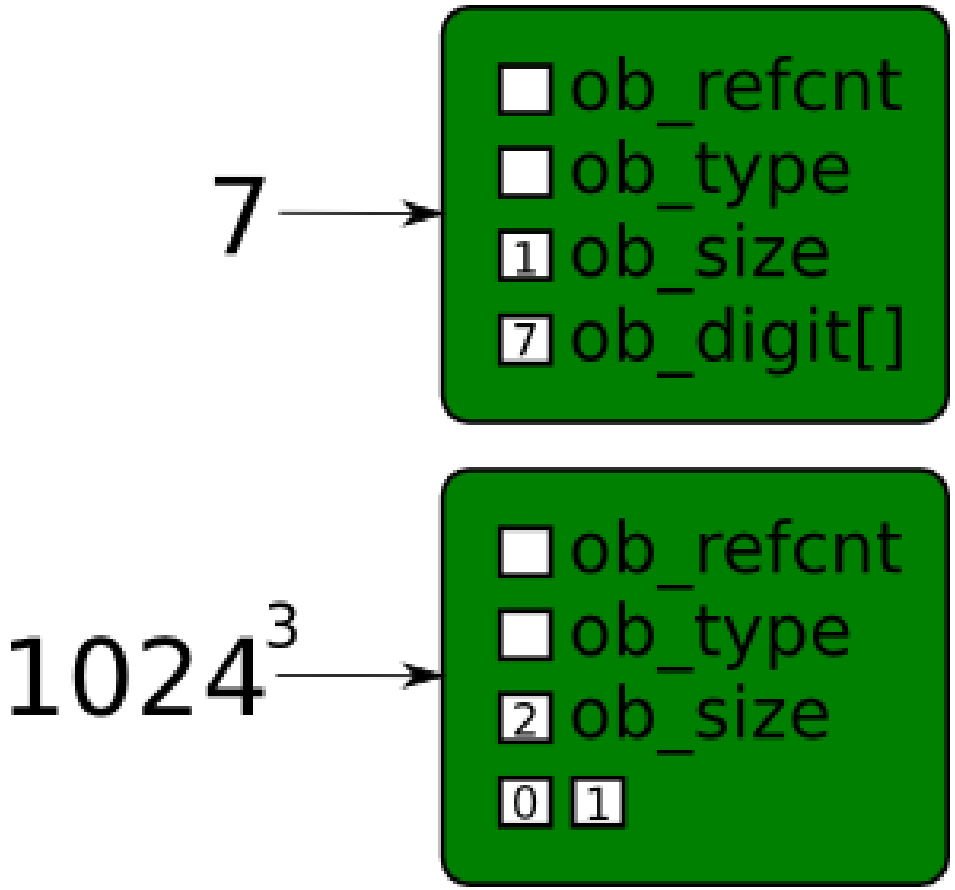
```
>>> ref_cnt = ctypes.c_long.from_address(id(None))
>>> ref_cnt.value = 0
Fatal Python error: deallocating None

Current thread 0x00007f2fb8d2a700:
  File "<stdin>", line 1 in <module>
[2]    10960 abort (core dumped)  python3
```

El objeto int



- `ob_digit`: array de enteros
- El valor del entero es `sum(map(lambda x: 1024*1024*1024, ob_size))`

El objeto int



El objeto int

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> intsize = ctypes.sizeof(ctypes.c_int)
>>> x = 100
>>> ctypes.c_long.from_address(id(x) + longsize * 2)
c_long(1)
>>> ctypes.c_uint.from_address(id(x) + longsize * 3)
c_uint(100)
>>> x = 1024 * 1024 * 1024
>>> ctypes.c_long.from_address(id(x) + longsize * 2)
c_long(2)
>>> ctypes.c_uint.from_address(id(x) + longsize * 3)
c_uint(0)
>>> ctypes.c_uint.from_address(id(x) + longsize * 3 + intsize)
c_uint(1)
```

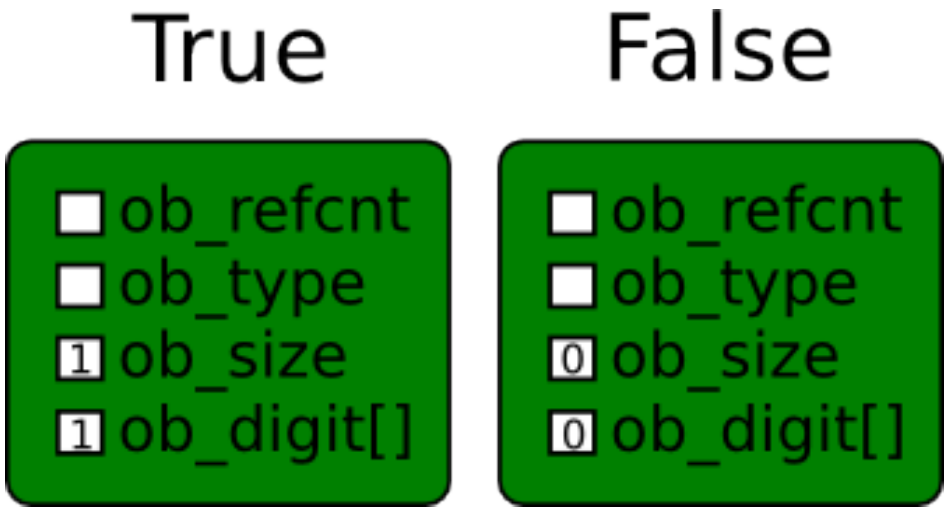
Very bad things

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> x = 1000
>>> int_value = ctypes.c_uint.from_address(id(x) + longsize * 3)
>>> int_value.value = 1001
>>> x
1001
>>> 1000
1000
```

Very bad things

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> x = 100
>>> int_value = ctypes.c_uint.from_address(id(x) + longsize * 3)
>>> int_value.value = 101
>>> x
101
>>> 100
101
>>> 100 + 2
103
```

El objeto bool



- Realmente son 2 instancias int con un tipo específico y valores 0 y 1

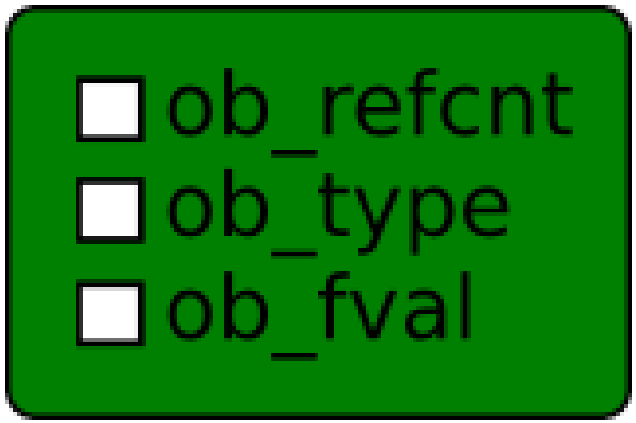
El objeto bool

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> ctypes.c_long.from_address(id(True) + longsize * 2)
c_long(1)
>>> ctypes.c_uint.from_address(id(True) + longsize * 3)
c_uint(1)
>>> ctypes.c_long.from_address(id(False) + longsize * 2)
c_long(0)
>>> ctypes.c_uint.from_address(id(False) + longsize * 3)
c_uint(0)
```

Very bad things

```
>>> val = ctypes.c_int.from_address(id(True) + longsize * 2)
>>> val.value = 0
>>> val = ctypes.c_int.from_address(id(True) + longsize * 3)
>>> val.value = 0
>>> True == False
True
```

El objeto float

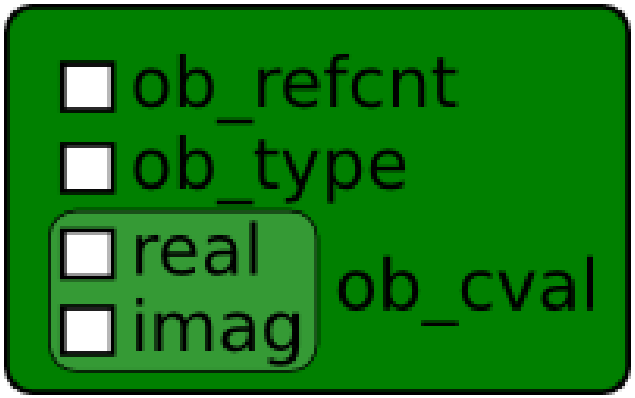


- `ob_fval`: es un double

El objeto float

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> x = 1.5
>>> ctypes.c_double.from_address(id(x) + longsize * 2)
c_double(1.5)
```

El objeto complex

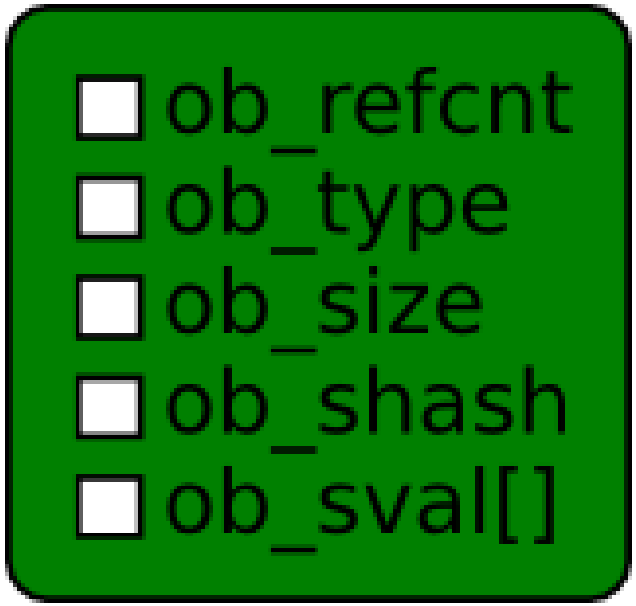


- `cval`: dos valores double `real` e `imag`

El objeto complex

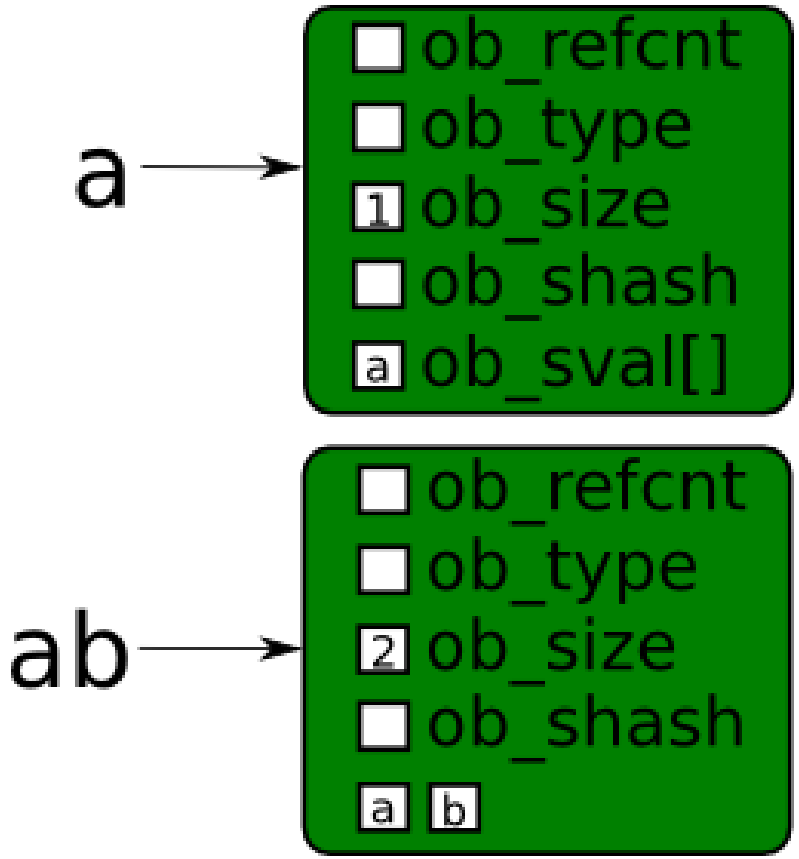
```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> doublesize = ctypes.sizeof(ctypes.c_double)
>>> x = 1 + 3j
>>> ctypes.c_double.from_address(id(x) + longsize * 2)
c_double(1.0)
>>> ctypes.c_double.from_address(id(x) + longsize * 2 + doublesize)
c_double(3.0)
```

El objeto bytes



- `ob_shash`: hash de la cadena o -1
- `ob_sval`: cadena terminada en 0 (tipo C)

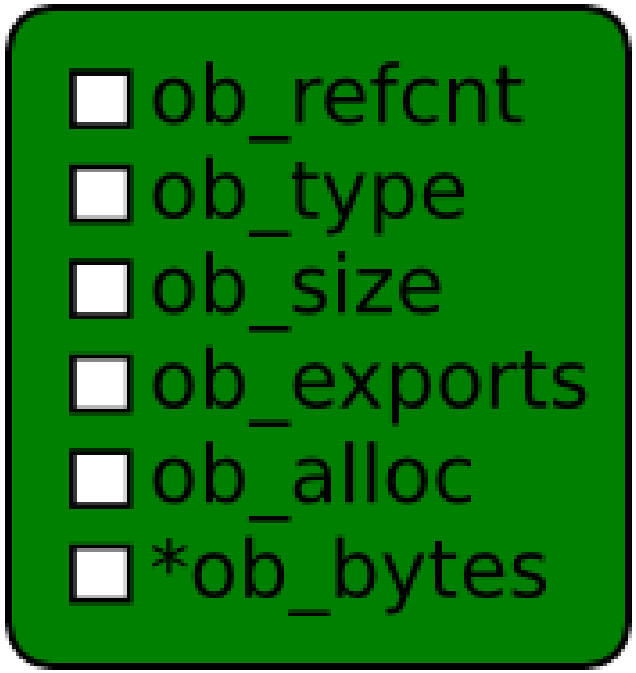
El objeto bytes



El objeto bytes

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> charsize = ctypes.sizeof(ctypes.c_char)
>>> x = b"yep"
>>> ctypes.c_long.from_address(id(x) + longsize * 2)
c_long(3)
>>> hash(x)
95469626770683243 3
>>> ctypes.c_long.from_address(id(x) + longsize * 3)
c_long(95469626770683243 3)
>>> ctypes.c_char.from_address(id(x) + longsize * 4)
c_char(b'y')
>>> ctypes.c_char.from_address(id(x) + longsize * 4 + charsize)
c_char(b'e')
>>> ctypes.c_char.from_address(id(x) + longsize * 4 + charsize * 2)
c_char(b'p')
>>> ctypes.c_char.from_address(id(x) + longsize * 4 + charsize * 3)
c_char(b'\x00')
```

El objeto bytearray

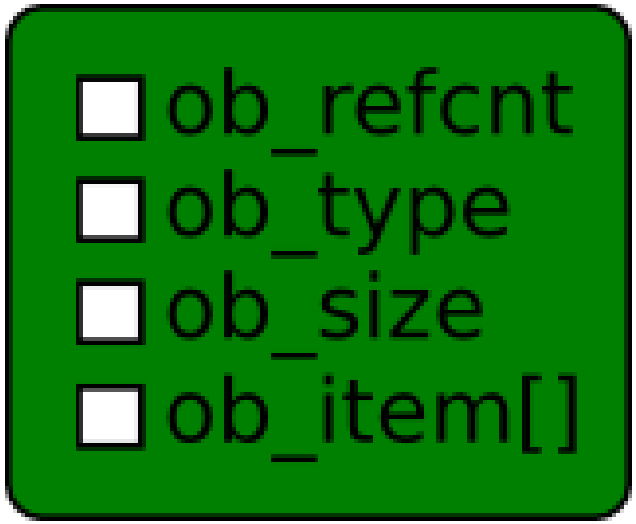


- `ob_exports`: memoryviews apuntando a este objeto
- `ob_alloc`: contabiliza el número de bytes almacenados
- `ob_bytes`: puntero a la posición de los bytes almacenados

El objeto bytearray

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> charsize = ctypes.sizeof(ctypes.c_char)
>>> x = bytearray(b"yep")
>>> ctypes.c_long.from_address(id(x) + longsize * 2)
c_long(3)
>>> ctypes.c_long.from_address(id(x) + longsize * 3)
c_long(0)
>>> ctypes.c_long.from_address(id(x) + longsize * 4)
c_char(4)
>>> addr = ctypes.c_void_p.from_address(id(x) + longsize * 5).value
>>> ctypes.c_char.from_address(addr)
c_char(b'y')
>>> ctypes.c_char.from_address(addr + charsize)
c_char(b'e')
```

El objeto tuple



- `ob_item`: array de punteros a PyObject

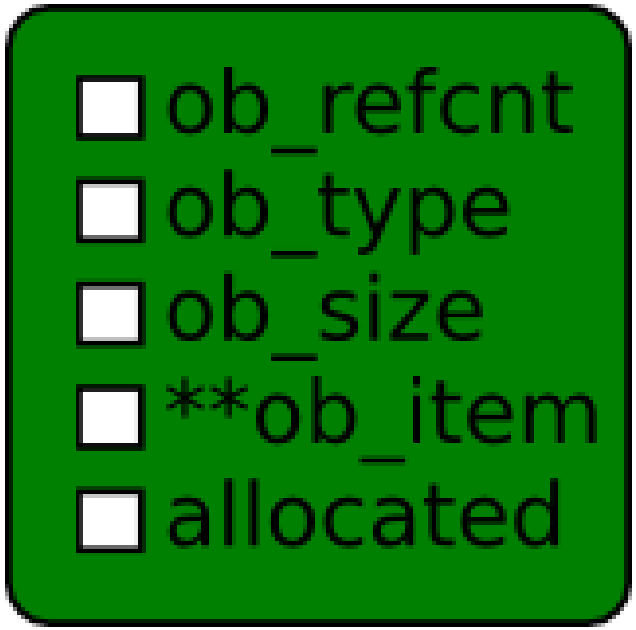
El objeto tuple

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> x = (True, False)
>>> ctypes.c_long.from_address(id(x) + longsize * 2)
c_long(2)
>>> ctypes.c_void_p.from_address(id(x) + longsize * 3)
c_void_p(140048684311616)
>>> ctypes.c_void_p.from_address(id(x) + longsize * 4)
c_void_p(140048684311648)
>>> id(True)
140048684311616
>>> id(False)
140048684311648
```

Very Bad Things

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> x = (1, 2, 3)
>>> tuple_size = ctypes.c_long.from_address(id(x) + longsize * 2)
>>> tuple_size.value = 2
>>> x
(1, 2)
```

El objeto lista



- `ob_item`: puntero a punteros de PyObject
- `allocated`: memoria reservada actualmente para la lista

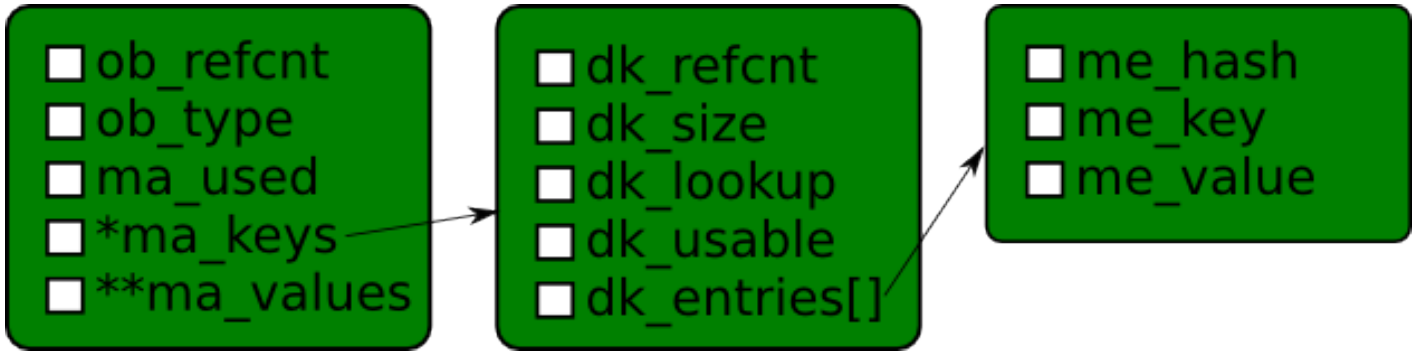
El objeto lista

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> x = [1,2,3]
>>> ctypes.c_long.from_address(id(x) + longsize * 2)
c_long(3)
>>> ctypes.c_void_p.from_address(id(x) + longsize * 3)
c_void_p(36205328)
>>> ctypes.c_void_p.from_address(36205328)
c_void_p(140048684735040)
>>> id(1)
140048684735040
>>> ctypes.c_void_p.from_address(36205328 + longsize)
c_void_p(140048684735072)
>>> id(2)
140048684735072
```

Very Bad Things

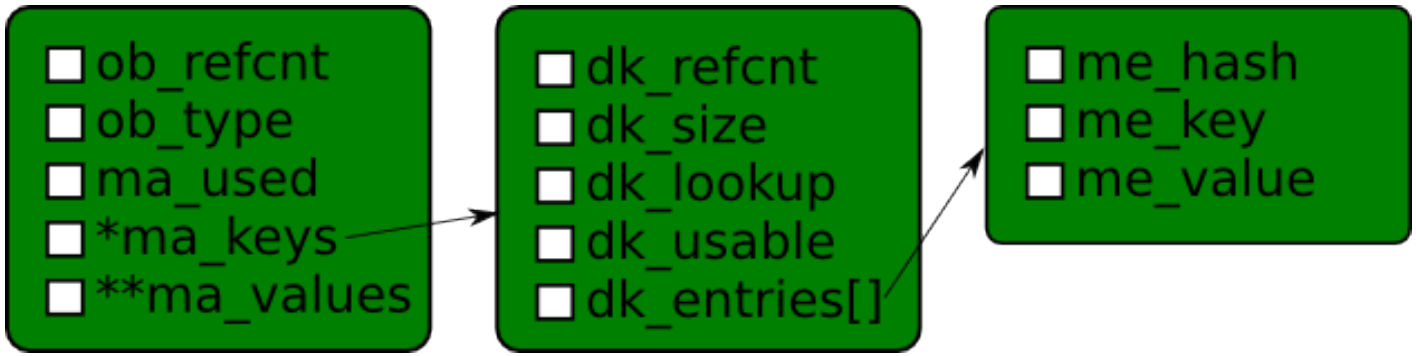
```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> x = [1,2,3,4,5,6,7,8,9,10]
>>> y = [10,9,8,7]
>>> datos_y = ctypes.c_long.from_address(id(y) + longsize * 3)
>>> datos_x = ctypes.c_long.from_address(id(x) + longsize * 3)
>>> datos_y.value = datos_x.value
>>> y
[1, 2, 3, 4]
>>> x[0] = 7
>>> y
[7, 2, 3, 4]
```

El objeto dict



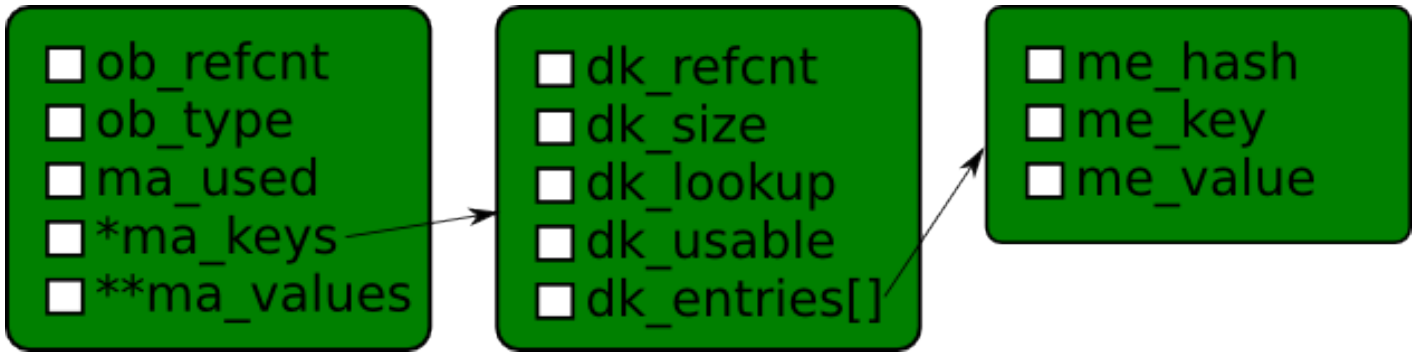
- `ma_used`: número de items.
- `ma_keys`: puntero a la estructura que almacena el diccionario.
- `ma_values`: puntero a punteros de PyObject (para slited tables y NULL para combined tables).

PyDictKeysObject



- `dk_refcnt`: contador de referencias
- `dk_size`: Tamaño total de la tabla hash para guardar entradas
- `dk_lookup`: Slot para función de búsqueda
- `dk_usable`: La fracción usable del diccionario antes de un redimensionado
- `dk_entries[n]`: Las entradas en la tabla hash

PyDictKeyEntry



- `me_hash`: Hash de la key
- `me_key`: Puntero al objeto key
- `me_value`: Puntero al objeto valor (Solo para combined tables)

El objeto dict

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> d = {1: 3, 7: 5}
>>> keys = ctypes.c_void_p.from_address(id(d) + longsize * 3).value
>>> keyentry1 = keys + longsize * 4 + longsize * hash(1) * 3
>>> keyentry7 = keys + longsize * 4 + longsize * hash(7) * 3
>>> key1 = ctypes.c_long.from_address(keyentry1 + longsize).value
>>> val1 = ctypes.c_long.from_address(keyentry1 + longsize * 2).valu
>>> key7 = ctypes.c_long.from_address(keyentry7 + longsize).value
>>> val7 = ctypes.c_long.from_address(keyentry7 + longsize * 2).valu
>>> ctypes.c_uint.from_address(key1 + longsize * 3)
c_long(1)
>>> ctypes.c_uint.from_address(val1 + longsize * 3)
c_long(3)
>>> ctypes.c_uint.from_address(key7 + longsize * 3)
c_long(7)
>>> ctypes.c_uint.from_address(val7 + longsize * 3)
c_long(5)
```

El objeto type (no completo)

- `tp_name`: Nombre de la clase
- `tp_doc`: El docstring del tipo
- `tp_dict`: El diccionario de attributos del tipo
- `tp_dictoffset`: El offset al diccionario de atributos de los objetos
- `tp_as_number`: Puntero a estructura de slots
- `tp_as_sequence`: Puntero a estructura de slots
- `tp_as_mappings`: Puntero a estructura de slots

El objeto type

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> ctypes.c_char_p.from_address(id(int) + longsize * 3)
'int'
>>> ctypes.c_char_p.from_address(id(type) + longsize * 3)
'type'
>>> class Prueba:
...     pass
...
>>> ctypes.c_char_p.from_address(id(Prueba) + longsize * 3)
'Prueba'
```

Very Bad Things

```
>>> longsize = ctypes.sizeof(ctypes.c_long)
>>> class LifeInt(int):
...   def __repr__(self):
...     return "42"
...
>>> for x in range(-5, 258):
...   type_addr = ctypes.c_long.from_address(id(x) + sizeof(c_long))
...   type_addr.value = id(LifeInt)
>>> 5 + 5
42
```

Operaciones sobre objetos

- Se obtiene el tipo del objeto
- Se ejecuta el slot apropiado pasándole el puntero al objeto
- Ejemplo:

```
x = 5
x + 2
```

```
x.ob_type->tp_as_number->tp_add(x, 2)
```

¿Cómo se almacenan los attributos?

- Los de clase en `tp_dict`
- Los de los objetos en `id(obj) + tp_dictoffset`
- No todos los tipos permiten atributos de objeto

¿Cómo se crea un objeto?

- Se llama al `tp_call` del tipo
- Este llama al `tp_new` del tipo que le devuelve un objeto inicializado en memoria
- Llama al `tp_init` del type del objeto creado
- Ejemplo:

```
class Prueba:
    pass
p = Prueba()


Prueba.ob_type->tp_call(Prueba, [], {})
    p = Prueba.tp_new(Prueba, [], {})
    p.ob_type->tp_init(p, [], {})
```

Referencias

- Código de python: Include and Objects
- Documentación de ctypes: http://docs.python.org/3/library/ctypes.html
- Documentación de la API C de Python: http://docs.python.org/3/c-api/index.html
- PEP 412 – Key-Sharing Dictionary
- Website de Eli Bendersky: http://eli.thegreenplace.net/
- Yaniv Aknin Tech Blog: http://tech.blog.aknin.name/

Dudas

. . .