Python

ООП-2

Функторы

Функторы

- Функторы - это классы с определённым оператором(),
- В Python — с реализованным методом __call__.

```
class Functor:
    def __call__(self, value):
        return value*2
>>> f = Functor()
>>> f(5)
10
```

Функторы — пример №1

```
class mulN:
    def __init__(self, n):
        self.n = n
    def __call__(self, value):
        return self.n * value
>>> f = mulN(7)
>>> f(3)
21
```

Функторы — пример №2

```
class logger:
    def __init__(self, filename):
        self.file = open(filename, "a")
    def __call__(self, message):
        self.file.write(message + "\n")
    def close(self):
        self.file.close()
>>> l = logger("log.txt")
>>> l("Message")
>>> l.close()
```

Функторы — пример №3

```
def f(a, b):
    print ("first: %s, second: %s" % (a, b))
class currying:
    def __init__(self, func, param):
        self.func = func
        self.param = param
    def __call__(self, param):
        return self.func(self.param, param)
>>> c = currying(f, 5)
>>> c(7)
first: 5, second: 7
```

Python

ООП. Иерархия типов

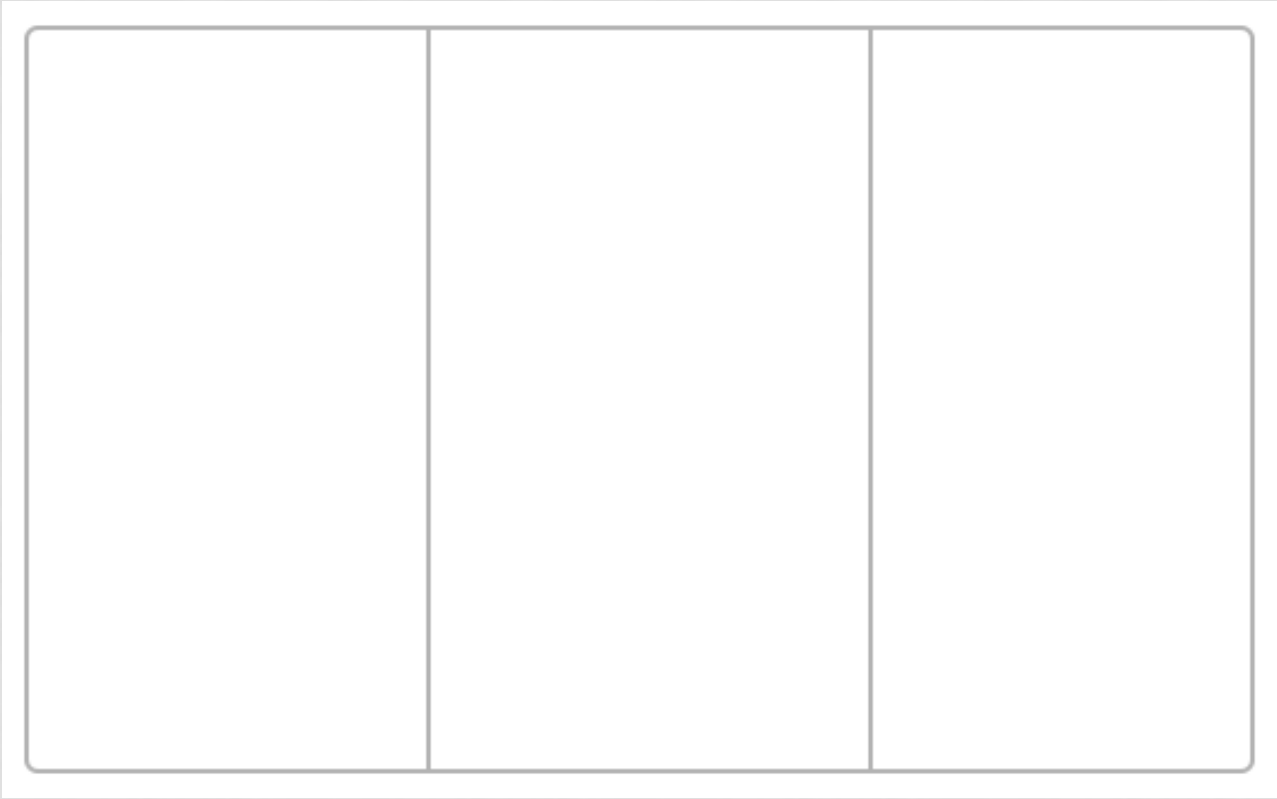
Пример про int…

```
 >>> two = 2 #1
 >>> type(two)
<type 'int'> #2
 >>> type(type(two))
<type 'type'> #3
 >>> type(two).__bases__
(<type 'object'>,) #4
 >>> dir(two) #5
['__abs__', '__add__', '__and__', '__class__', '__cmp__', '__coerce__',
 '__delattr__', '__div__', '__divmod__', '__doc__', '__float__',
 '__floordiv__', '__format__', '__getattribute__', '__getnewargs__',
 '__hash__', '__hex__', '__index__', '__init__', '__int__', '__invert__',
 '__long__', '__lshift__', '__mod__', '__mul__', '__neg__', '__new__',
 '__nonzero__', '__oct__', '__or__', '__pos__', '__pow__', '__radd__',
 '__rand__', '__rdiv__', '__rdivmod__', '__reduce__', '__reduce_ex__',
 '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__',
 '__ror__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__',
 '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__',
 '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__',
 'conjugate', 'denominator', 'imag', 'numerator', 'real']
```

Первое правило Python ООП

- ~~Никому не говорить про Python ООП~~
- Все объект!
- «Объекты» хранят ссылку на свой класс __class__

Начало

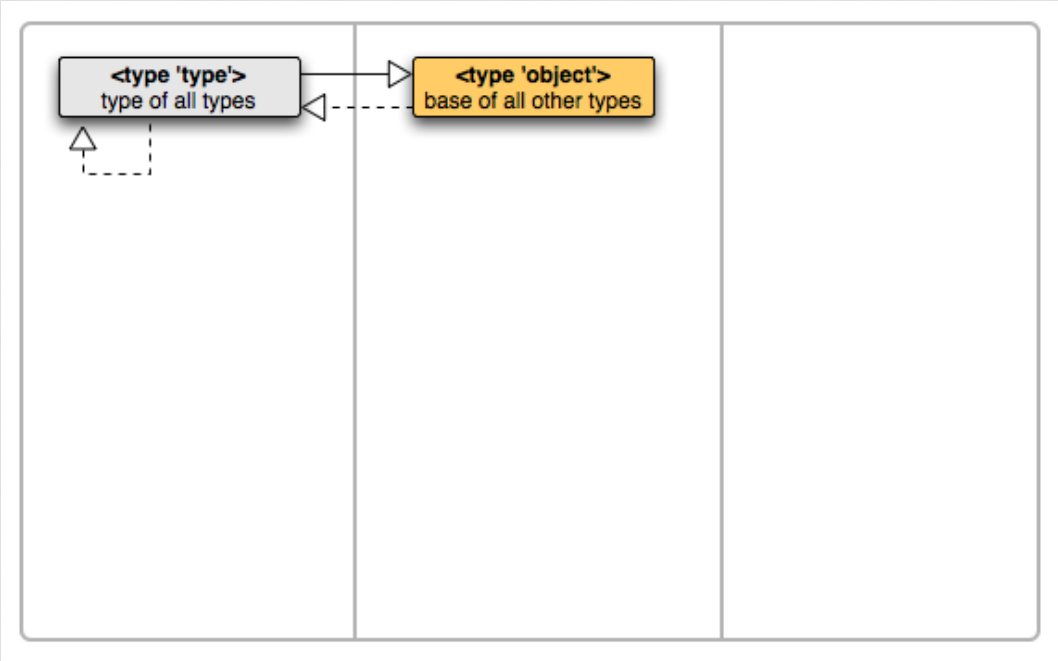


Курицы и яйца…

```
 >>> object
<type 'object'>
 >>> type
<type 'type'>
 >>> type(object)
<type 'type'>
 >>> object.__class__
<type 'type'>
 >>> object.__bases__
()
 >>> type.__class__
<type 'type'>
 >>> type.__bases__
(<type 'object'>,)
```

Object & type

Объекты object и type являются базовыми в языке Python. Исходя из предыдущего эксперимента, можно построить диаграмму отношений между ними.



Объекты-типы

Продолжим эксперименты:

```
>>> isinstance(object, object)
True
>>> isinstance(type, object)
True
```

Объекты object и type - это объекты-типы в языке Python.

Это означает, что

- они могут представлять абстрактные типы данных в программе;
- они могут быть унаследованы другими объектами;
- можно создавать их экземпляры;
- типом любого объекта-типа является <type 'type'>;
- одни их называют типами, другие - классами.

Второе правило

- ~~Никому не говорить про Python ООП~~
- **Class is Type is Class**

- **Type Or Non-type Test Rule**

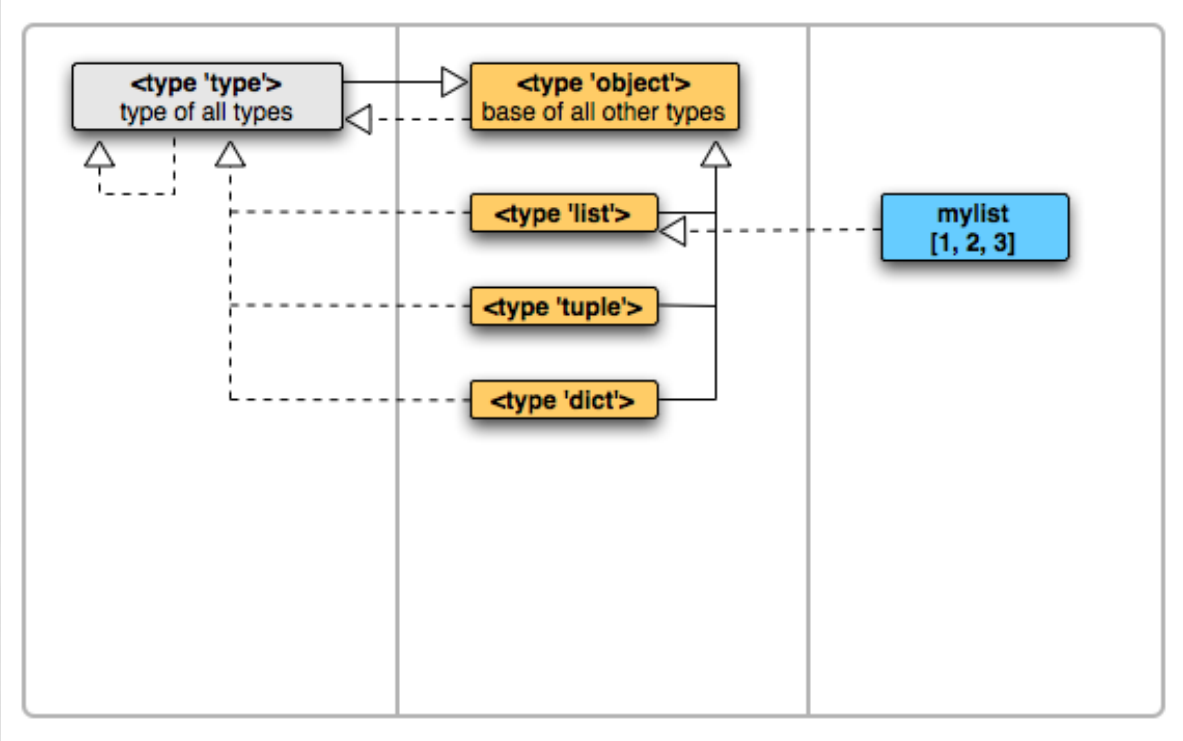
If an object is an instance of <type 'type'>, then it is a type. Otherwise, it is not a type.

Встроенные типы

```
>>> list
<type 'list'>
>>> list.__class__
<type 'type'>
>>> list.__bases__
(<type 'object'>,)
>>> tuple.__class__, tuple.__bases__
(<type 'type'>, (<type 'object'>,))
>>> dict.__class__, dict.__bases__
(<type 'type'>, (<type 'object'>,))

>>> mylist = [1,2,3]
>>> mylist.__class__
<type 'list'>
```

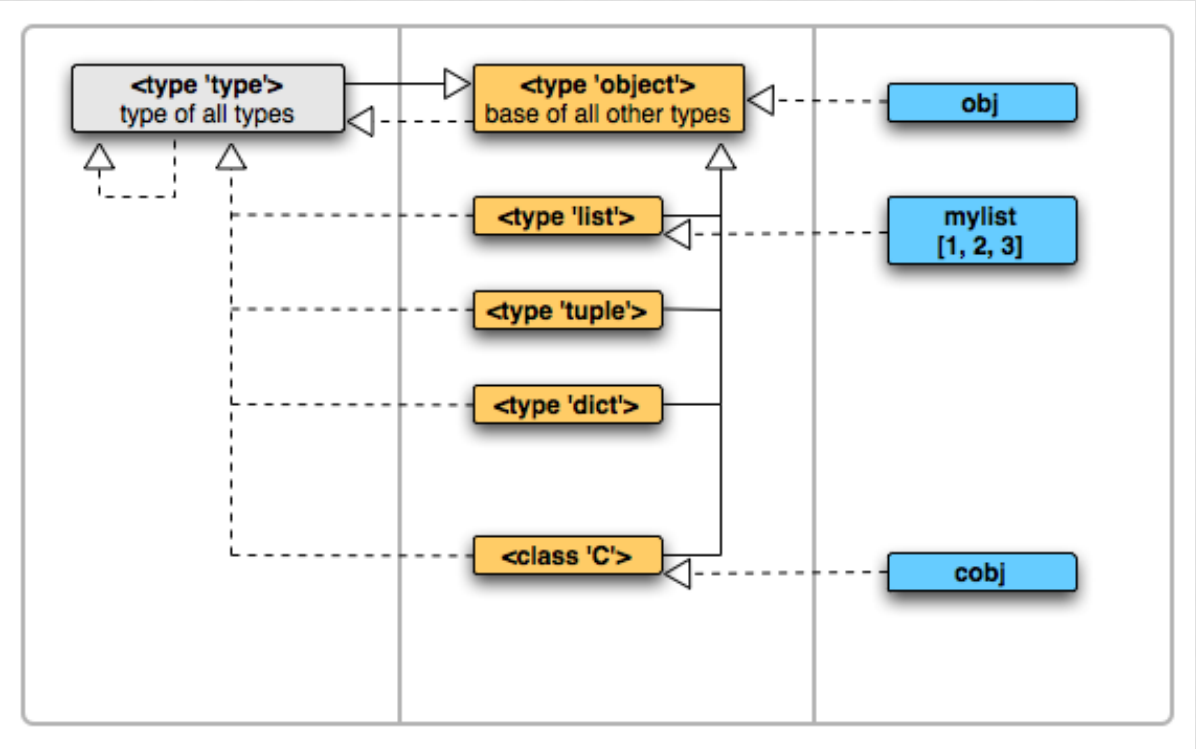
Диаграмма типов



Пользовательские типы

```
class New:
     pass
 new = New()
>>> type(new)
<class '__main__.New'>
>>> type(New)
<type 'type'>
```

Диаграмма типов



Инстанцирование

- Как Python на самом деле создает объекты?

При вызове a = A()…

- вызывается метод __call__ объекта-класса A, который
- вызывает __new__ и __init__ класса-предка A

- Но откуда он это знает???

- Потому что __call__ берется из type(A)

- А можно чтобы он брался из другого места?

Да. Метаклассы

Метаклассы. Нано-введение

Хотим:

```
class Man(object):
        pass

>>> me = Man(height = 180, weight = 80)
Traceback (most recent call last):
File "<stdin>", line 20, in <module>
     TypeError: object.__new__() takes no parameters
```

Метаклассы. Нано-введение

```
class AttributeInitType(type):
      def __call__(self, *args, **kwargs):
              obj = type.__call__(self, *args)
                 # добавим ему переданные в вызове аргументы в
                 # качестве атрибутов.
              for name in kwargs:
                    setattr(obj, name, kwargs[name])
                  # вернем готовый объект
                return obj

class Man:
      __metaclass__ = AttributeInitType

>>>me = Man(height = 180, weigth = 80)
>>>print(me.height)
180
```