Python.

Объекты и
Метапрограммирование.

Пользовательские атрибуты и методы

```
>>> class C(object):
    classattr = "attr on class"
    def f(self):
        return "function f"
>>> C.__dict__
{'classattr': 'attr on class', '__module__': '__main__',
    '__doc__': None, 'f': <function f at 0x008F6B70>}


>>> c = C()
>>> print c.__dict__
{}
>>> c.classattr is C.__dict__['classattr']
True
>>> c.f is C.__dict__['f']
False
```

Атрибут __dict__

- Таблица ключ-значение
- Хранит имена пользовательских атрибутов объекта

Поиск имен

- <объект>.<атрибут>

1. Поиск значения <атрибут> в таблице <объект>.__dict__ и во встроенных атрибутах
2. <объект>.__class__.__dict__ и во встроенных <объект>.__class__
3. Поиск в объектах <объект>.__class__.__bases__

Примеры 1, 2

Дескрипторы

```
a.x = 1 <==> setattr(a, 'x', 1)
del a.x <==> delattr(a, 'x')
a.x <==> getattr(a, 'x')
```

- setattr и delattr влияют и изменяют только сам объект (точнее a.__dict__)

Интерфейс дескриптора

Методы __get()__ , __set()__ и __delete()__

```
class Desc(object):
    def __get__(self, obj, cls=None):
        pass
    def __set__(self, obj, val):
        pass
    def __delete__(self, obj):
        pass
```

Слабый дескриптор

```
class Weak(object):
    def __get__(self, obj, cls):
        return "WeakValue"
class A(object):
    a = Weak()

>>> i = A()
>>> print i.a
WeakValue

>>> i.a = "New Value"
>>> print i.a
"New Value"
```

Сильный дескриптор

```
>>> class Strong(object):
    def __get__(self, obj, cls):
        return "StrongValue"
    def __set__(self, obj, cls):
        pass

>>> class A(object):
    a = Weak()
>>> i=A()
>>> print i.a
StrongValue
>>> i.a = "NewValue"
>>> print i.a
StrongValue
```

Классы

- Классы (типы) — это объектные фабрики. Их главная задача — создавать объекты, обладающие определенным поведением.
- Классы определяют поведение объектов с помощью своих атрибутов

Инстанциирование объекта

2 этапа: сначала его создание, потом инициализация

def __new__(cls, ...) — статический метод, который создает объект класса cls.

def __init__(self, ...) — метод класса, который инициализирует созданный объект.

Пример

```
class A(object):
    pass
```

- Нет __new__ и __init__

```
>>> object.__dict__['__init__']
<slot wrapper '__init__' of 'object'
objects>
>>> object.__dict__['__new__']
<built-in method __new__ of type object
at 0x82e780>
```

```
a = object.__new__(A)
object.__init__(a)
```

Пример 3

Метаклассы

Для класса (типа), так же как и для обычного объекта, существует класс (тип), который создает классы и определяет поведение класса.

По умолчанию для всех определяемых классов метаклассом является type.

Создание объекта-типа

- XClass = XMetaClass(name, bases, attrs)

- class A: pass

```
>>> type('A', (object,), {})
<class '__main__.A'>
```

Пример

```
class B(A):
...     def foo(self):
...         42

type('B', (A,), {'foo': lambda self: 42})
```

type

- type(a) — вызов с одним аргументом, возвращает тип объекта,
- type(name, bases, attrs) — вызов с тремя аргументами — это вызов конструктора класса.

Задание метакласса

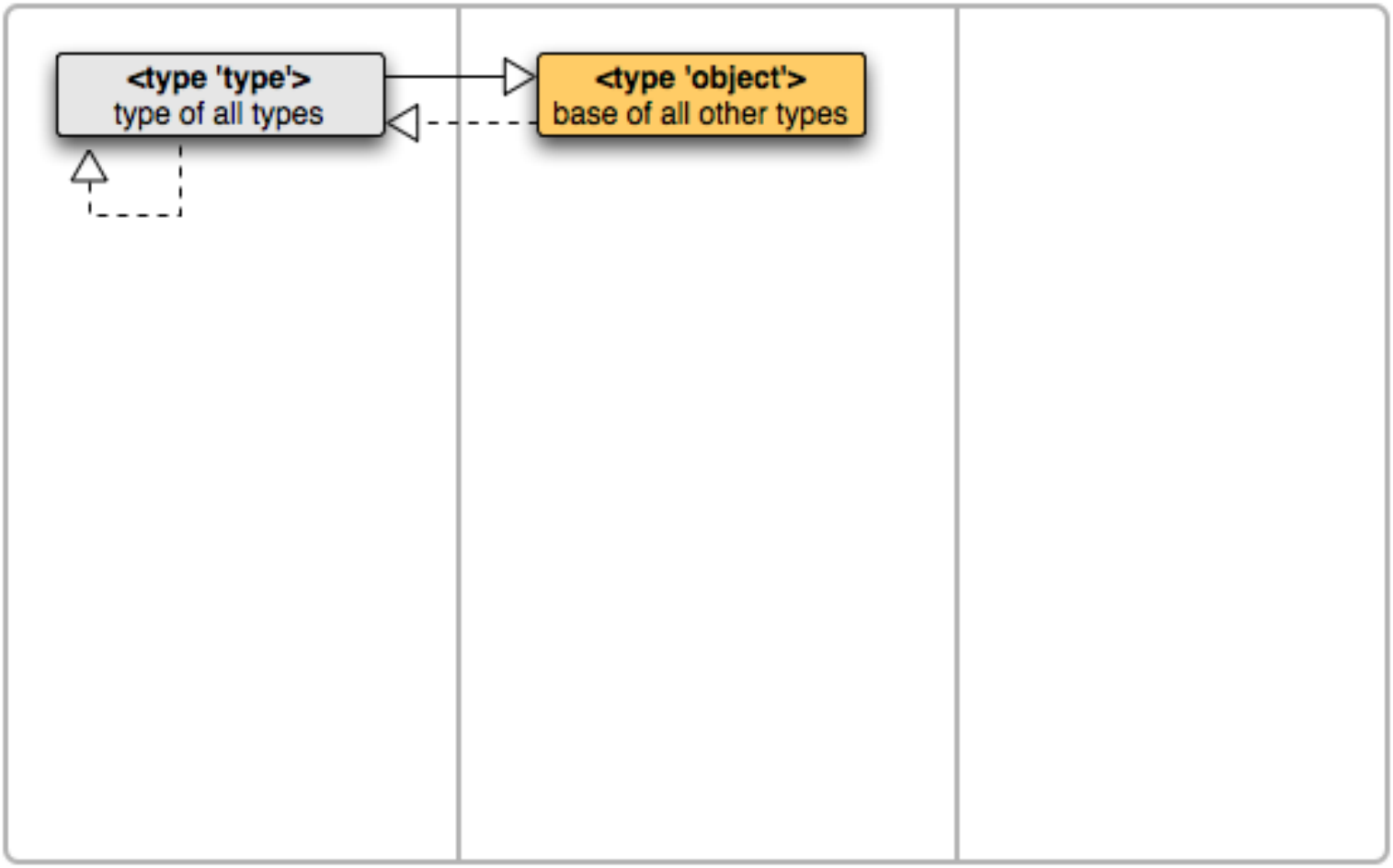
```
class A(object):
    __metaclass__ = Meta
```

Курица или яйцо

```
>>> object
<type 'object'>
>>> type
<type 'type'>
>>> type(object)
<type 'type'>
>>> type(type)
<type 'type'>
```

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

Object и type



Объекты-типы

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

Объекты-типы

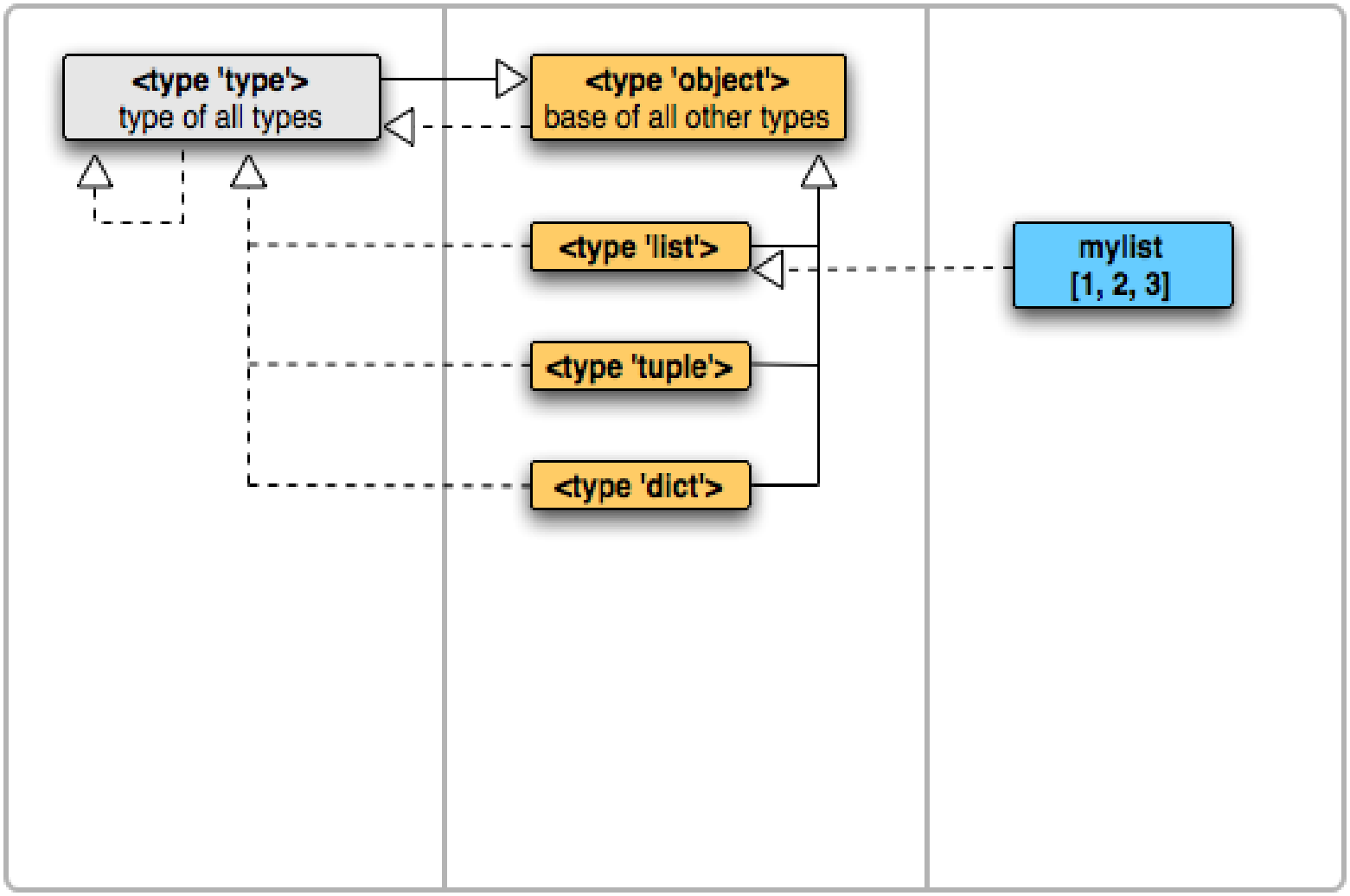
- Могут представлять абстрактные типы данных в программе
- Могут быть унаследованы другими объектами
- Можно создавать их экземпляры
- Типом любого объекта-типа является <type 'type'>
- Тип == класс

Эксперименты

```
>>> 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'>
```

List, tuple, dict



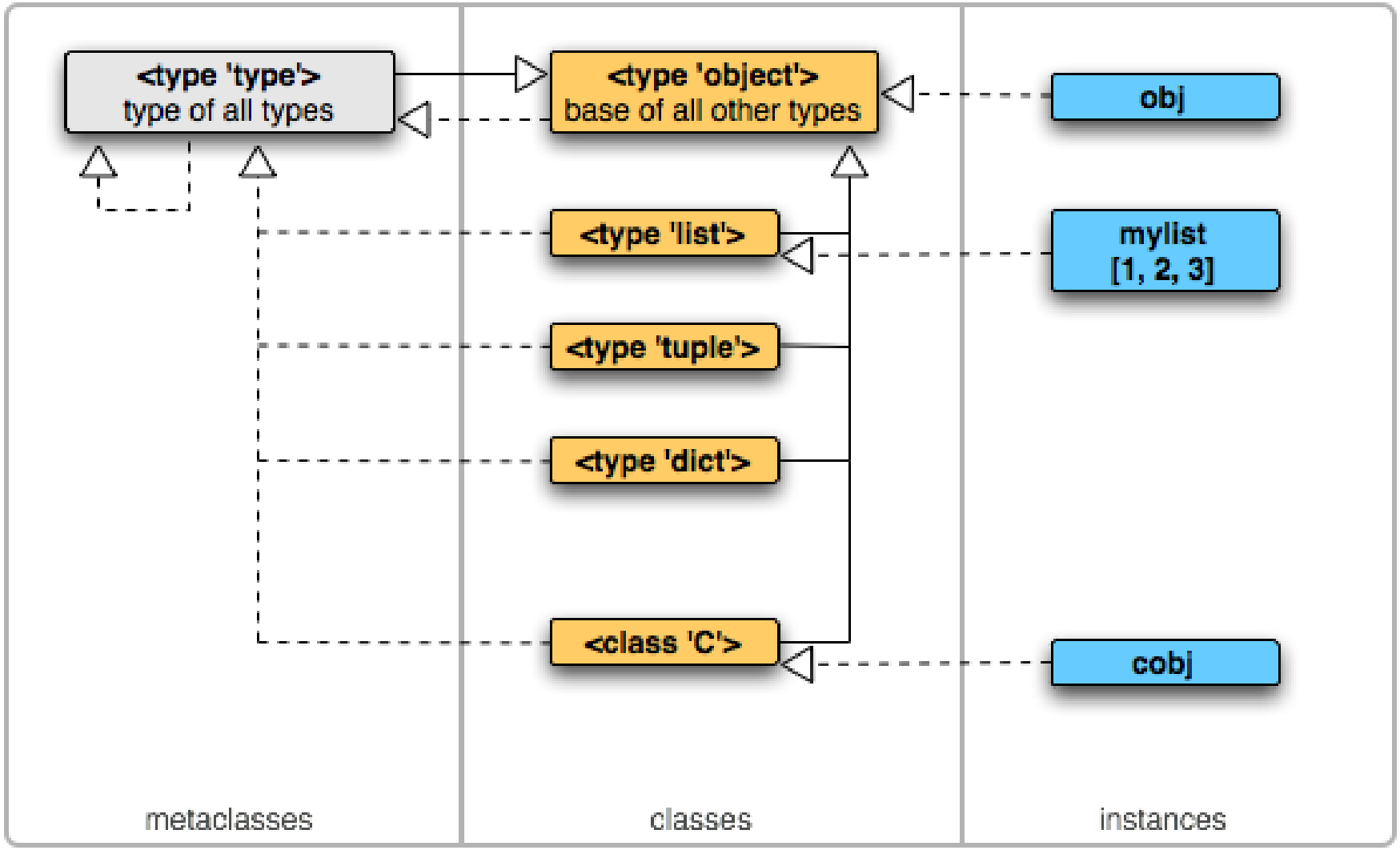
Пользовательские типы

```
class Old:
    pass
>>> old = Old()
>>> type(old)
<type 'instance'>
>>> type(Old)
<type 'classobj'>
>>> issubclass(Old, object)
False
```

Пользовательские типы

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

Отношение объектов



Пример 4

Применение?

```
class Person(models.Model):
  name = models.CharField(max_length=30)
  age = models.IntegerField()

guy = Person(name="Bob", age=35)
print guy.age
```

model.Models определяет __metaclass__