Introduction of Python's Object Model

Mike <mikeandmore@gmail.com>

Introduction to Python

- It's a dynamic duck type language
- It has a build-in OO support
- Everything is an object
- With simplicity style of programming
- Widely used

Introduction to Python

- It has both “class" definition and “function” definition support

```
class sample(object):
    def __init__(self):
        print 'construct'

    def do_something(self, x):
        print 'invoking with ', x

def func(x, y):
    if y == 0:
        return "error"
    return x + y
```

Facts of Everything

- Object!
- The “class" is derived from object or a meta-class(in python which is called a Type.TypeType)
- The "function" is mostly like a "functor" in C++

Facts of Everything

- Instance...(?)
- Defining a class or a function DOES eats your memory.
- Static class attributes (Copy when class needs to be constructed)
- Function definition is a instance of a descriptor(?)
- See lang_intro2.py as a example

Facts of Everything

- In Python, everything is a object instance, from None, to definition
- Every object is build up through a binding mechanism, bind all the 'self' argument when spawning a instance
- See facts1.py

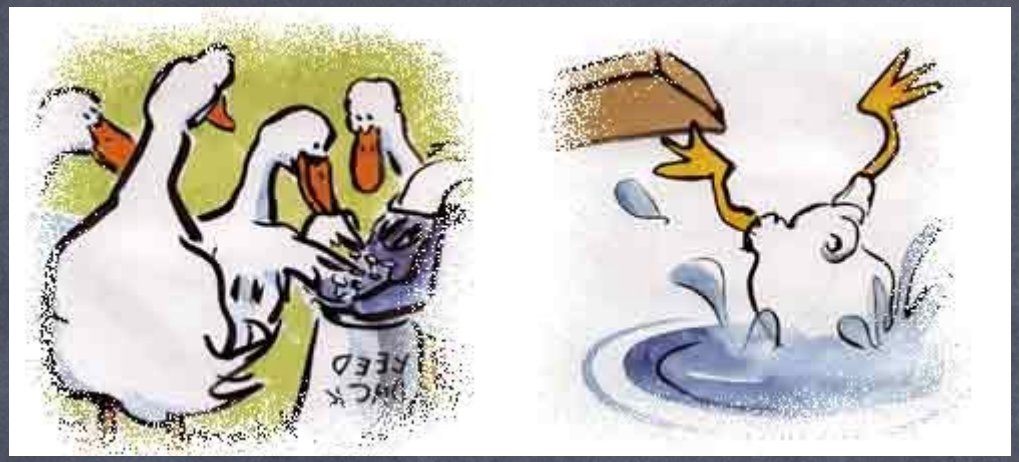
Descriptor

- Descriptor as a binding policy
- Some descriptor are build-in, like function, property, staticmethod, classmethod
- Of course you can write your own descriptor throw overloading __get__ __set__ and __delete__
- See descriptor1.py & method_desc.py & prop_desc.py

Duck Typing

- Every object(instance) is not all the same even with the same type/parent class
- Because each object can bind other values whenever they want
- But they're alike...

Duck Typing

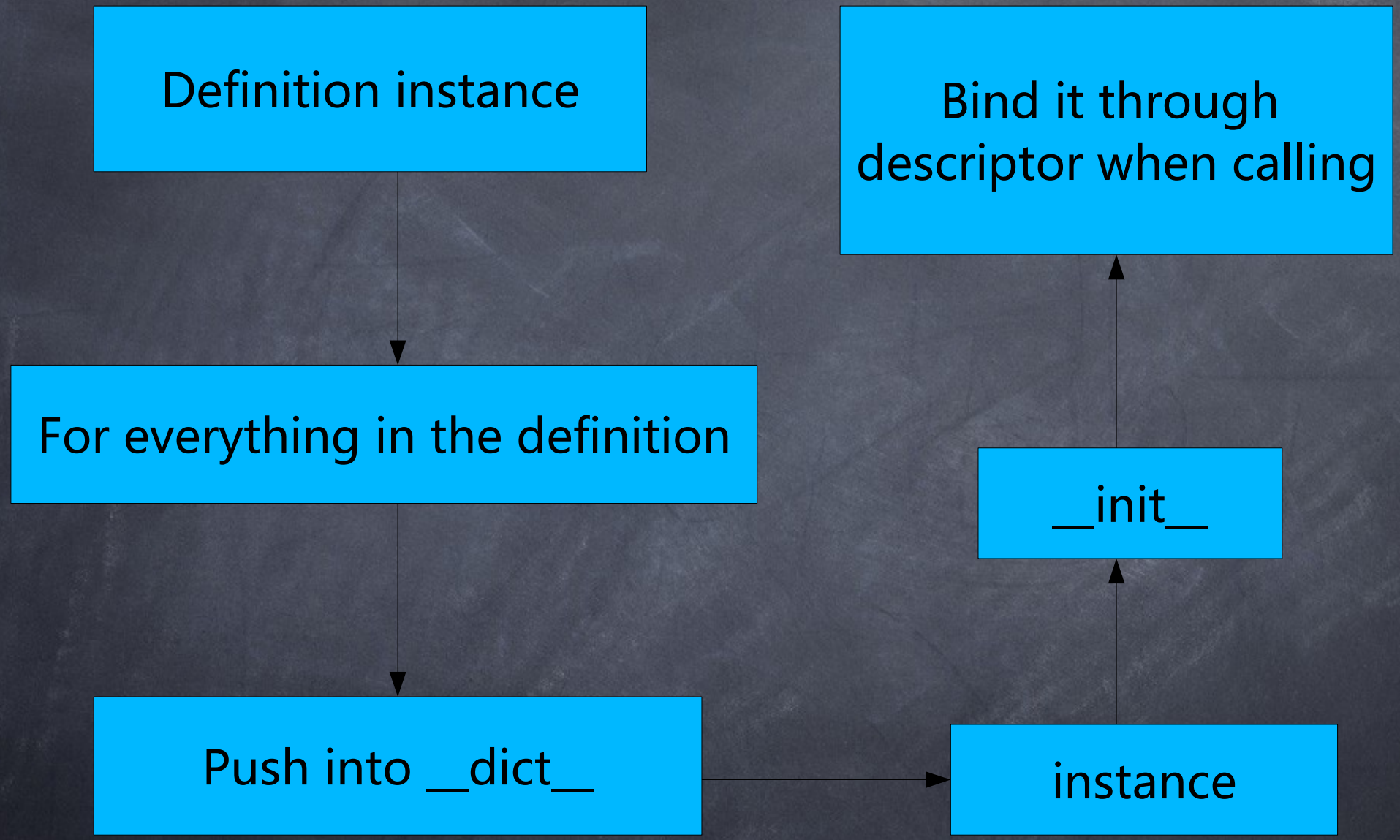


If it walks like a duck and quacks like a duck, I would call it a duck.

See duck1.py

Where did my stuff goes??

- __dict__ !!
- No matter a static member attribute, or a init attributes you bind it to self on __init__, it's all in __dict__!!
- Static attributes goes to type(x).__dict__
- The stuff you bind later is in x.__dict__
- See dict1.py


Definition instance
For everything in the definition
Push into __dict__
instance
__init__
Bind it through descriptor when calling

Multiple Inheritance

- Mixin style of multiple inheritance

Chocolate or peanuts?
See mixin.py

Mixin

- Mix the parent class in to one child class in order to change the child class's behavior
- Can be viewed as an “implemented interface"
- Python's inheritance is through Mixin DP
- Python has support for runtime Mixin (through super and __base__)

MRO

- Method Resolution Order
- New in python.... > python 2.2???
- The “object" class, object has a access to MRO through __mro__
- Multiple inheritance will push the parent class into __mro__ by order(described here http://www.python.org/download/releases/2.3/mro/)
- See super.py

Attribute Look up

- How did python vm find the attribute when x.y?
- First check out if x is a descriptor, if it is, do the stuff in the descriptor
- Or else try x.__dict__['y']
- Or else try type(x).__dict__['y']
- Or else try for parent in type(x).__mro__ ...
- See lookup.py

Super

- Python's new version has a keyword super
- Use MRO to initialize or call the parent method
- super(AAA, self) means the object in MRO next to AAA
- See super.py

End

- References:
  - www.aleax.it/Python/nylug05_om.pdf
  - http://blog.csdn.net/Jofee/archive/2006/03/30/
  - http://www.python.org/download/releases/2.3/

Thanks for your attention