

Core OO Concepts from a Programming Perspective — Python — Part I Object Oriented Analysis and Design

Aron Trauring

T++ Technical Skills Training Program

CUNY Institute for Software Design & Development (CISDD)

New York Software Industry Association (NYSIA)

December 1st, 2004

What is Python?

- Educational Language — Guido von Rossum — Early '90s

"Python aims to encourage the creation of reusable code. Even if we all wrote perfect documentation all of the time, code can hardly be considered reusable if it's not readable."

— Guido von Rossum

- Object Oriented — Extremely Anarchistic
- “Agile Programming Language”
- Interpreted
- Byte-code compilation (similar to Java)

Python Advantage — Quicker to Develop

- 5-10 Shorter than equivalent C++
- Interpreted — eliminates compile, debug cycle
- Large savings in programmer costs

Python Advantage — Easier to Maintain

- Readability
- Large savings in programmer costs

Python Advantage — Versatile

- From scripting to mission critical
- Large savings in training and tools costs

Python Advantage — Platform Portability

- All major OSes and lots of minor ones too
- From PDAs to mainframes
- Write once deploy anywhere

Python Advantage — Language Compatibility

- Microsoft C# and Java camps IronPython (.NET CLR) and Jython
- C, C++ extensions and embedding (original CPython)
- Python as scripting wrapper

Python Advantage — ShortFoilheads Compatibility

- Internet ShortFoilheads
- CORBA, COM, SOAP and XML extensions

Python Advantage — Platform Compatibility

- Option for low-level API integration
- Java — lowest common denominator
- C# — Microsoft only

Python Advantage — FOSS

- Real settings for real applications
- Free speech, beer and markets

Python Advantage — Powerful and Easy OO Paradigm

- Dynamic typing and binding
- Everything is an object
- Object behavior can be adapted at run-time
- “Pure” OO Model
- Power of Smalltalk but much easier to learn and use

Python — Myths

- Few Python programmers — thousands and growing
- Interpreted is inefficient — C for time-sensitive
- Just a scripting language — full platform
- No support — commercial tools, training and consultancies
- No one using it — NASA, NATO, Google, IBM, Disney
- No important frameworks — Zope, Plone, Twisted

Why Python for this Course?

- This is *not* an OO programming class
- Need text language for OO modeling
- Need language to illustrate OO concepts
- Python is by far easiest to learn (which is why I know it!)
- Python is by far easiest to read (don't have to be expert to understand)
- Python purest form of OO so best for illustrating concepts
- No common language for students — Python only one compatible with all others

- Learning a bit of Python will give you additional marketable skills

Mini Exercise 1

“Print Hello World” in the language you know best

Class as Namespace Abstraction

- Classes can be viewed as a tool for defining names (i.e., attributes) that export data and logic to clients
- Conceptual — Abstract Data Type
- Protocol (Interface) — data and associated operations
- Implementation — scope

Python Class Namespace

```
class <name>(superclass,...):           # Assign to name.
    data = value                         # Shared class data
    def method(self,...):               # Methods
        self.member = value             # Per-instance data
```

Class Objects

- Provide default behavior and serve as instance factories
- Python `class` statement is an executable statement (not a declaration) run from top to bottom
- When reached and run, it generates a new class object and assigns it to the name in the class header
- Assignments inside `class` statements make class attributes
- After running a `class` statement, class attributes are accessed by name qualification: `object.name`

- Attributes of a class object record state information and behavior, to be shared by all instances created from the class
- Function def statements nested inside a class generate methods, which process instances

Instance Objects

- Instances represent concrete items in your program's domain
- Each instance object inherits class attributes and gets its own namespace
- Instance objects start out empty, but inherit attributes that live in the class object they were generated from
- Inside class method functions, the first argument (called `self` by convention) references the instance object being processed
- Assignments to attributes of `self` create or change data in the instance, not the class

Mini Exercise 2

- Basic Namespace

```
>>> class One(object): pass
>>> c1 = One()
>>> c1.name = 'Joe'
>>> c1.country = "US"
>>> c1.name
?
>>> c2 = One()
>>> c2.name = 'Moe'
>>> c2.phone = "2122121111"
>>> c2.name
?
```

```
>>> c2.country
?
>>> c1.name
?
```

- Class as Record — Defining Data

```
>>> class One(object):
    def __init__(self,name,country):
        self.name=name
        self.country=country
>>> c1 = One('Joe','USA')
>>> c2 = One('Moe','France')
>>> c2.phone = '2121111111'
>>> c1.name
?
```

```
>>> c1.phone
?
>>> c1.country
?
>>> c2.country
?
>>> c2.phone
?
```

- Class Data — Scope and Methods

```
>>> class Spam(object):
    numinstances = 0
    def __init__(self):
        Spam.numinstances += 1
    def SpamOrder(self, servings):
```

```
        while servings:
            print "Spam"
        servings -= 1
    def printNuminstances():
        print "Number of Spammers: ", Spam.numinstances
    printNuminstances = staticmethod(printNuminstances)
>>> s1 = Spam()
>>> s1.SpamOrder(5)
?
>>> s2 = Spam()
>>> s3 = Spam()
>>> Spam.printNuminstances()
?
```

- Scope Issues

```
>>> class One(object):
    OneGlob = 50
    def __init__(self,name,country):
        self.name = name
        self.country = country.
>>> s1 = One('Joe',"USA")
>>> s2 = One("Moe","France")
>>> s1.OneGlob
?
>>> s2.OneGlob
?
s2.OneGlob=45
>>> s2.OneGlob
?
>>> One.OneGlob
?
```

T++ — CISDD — NYSIA

```
>>> s1.OneGlob
```

```
?
```

Python I - OOAD

Key Ideas

- Conceptual: Classes serves as template for instance — provide default behavior and serve as instance factories
- Protocol: Methods can be for Class or Instance
- Implementation: Scope of both data and methods can be local to Class or to Instance