Programming in Python

Week 6

Classes and Instances

Syllabus:
http://briandorsey.github.com/uwpython_onsite/

Brian Dorsey
brian@dorseys.org
http://briandorsey.info

Introductions, comments (...)

review assignment

trouble with seeing raw HTML after changing print_time.py to print HTML?

mime-type

Lightning Talks!
(15)

Unit A
(25)

readings this week

questions?

OO is a weak point for me

I'll be talking about how Python implements OO concepts, not how OO design.

parse, manipulate, output

parse, manipulate, output

all functions have to agree on shape of the data

end up with a bunch of functions which all take the same shape data

OO helps deal with the complexity by keeping data and behavior close

what is OO?

Do you have a sense of what it is?

I have no idea.

Concept	Count	Percentage
Inheritance	71	81%
Object	69	78%
Class	62	71%
Encapsulation	55	63%
Method	50	57%
Message Passing	49	56%
Polymorphism	47	53%
Abstraction	45	51%
Instantiation	31	35%
Attribute	29	33%
Information Hiding	28	32%
Dynamic Binding	13	15%
Relationship	12	14%
Interaction	10	12%
Class Hierarchy	9	10%
Abstract Data Type	7	8%
Object-Identity Independence	6	7%
Collaboration	5	6%
Aggregation	4	5%
Association	4	5%
Object Model	4	5%
Reuse	3	3%
Cohesion	2	2%
Coupling	2	2%
Graphical	2	2%
Persistence	2	2%
Composition	1	1%
Concurrency	1	1%
Dynamic Model	1	1%
Extensibility	1	1%
Framework	1	1%
Genericity	1	1%
Identifying Objects	1	1%
Modularization	1	1%
Naturalness	1	1%
Safe Referencing	1	1%
Typing	1	1%
Virtual Procedures	1	1%
Visibility	1	1%

The Quarks of Object-Oriented Development - Deborah J. Armstrong - http://agp.hx0.ru/oop/quarks.pdf

data abstraction
encapsulation
messaging
modularity
polymorphism
inheritance

I won't be defining them today. :)

Also, the literature gets very deep, very quickly.

Long history, term confusion.

What is "Object-Oriented Programming?"

- Bjarne Stroustrup, designer of C++

http://www2.research.att.com/~bs/whatis.pdf

- Deborah J. Armstrong

http://agp.hx0.ru/oop/quarks.pdf

for the purposes of this class...

"Objects can be thought of as wrapping their data within a set of functions designed to ensure that the data are used appropriately, and to assist in that use"

-- http://en.wikipedia.org/wiki/Object-oriented_programming

http://en.wikipedia.org/wiki/File:Close_up_of_an_Oreo_cookie.JPG

OO is the dominant model for the last 2 decades

shows up everywhere, everyone needs a basic understanding

OO == good

everything else == bad

signs that the dominance is fading

Where does Python fit?

Python is OO first,
with support for
other models

Even without classes,
Python is powerful

often you don't need
them.

Or rather, you don't need to write your own classes.

Built-in types are already objects.

OO is much more important in Java, C# -

because they are missing many Python features.

Or from the other side:

Python isn't "pure" OO.

You can write procedural scripts, lambda, etc.

In Python there are alternatives so we have to confront a question that Java (etc.) programmers never face:

should we use classes here -- or not?

Modularity:

provided by modules

put related variables and functions together, separate namespaces

Compound data:

In Python provided by tuples, lists, dictionaries, etc...

So what do Python classes add?

Classes
define new types
they
extend the language

(do you want/need this?)

type:

set of values, with operations

python provides lots of types already

numbers, lists, dictionaries...

Note:

foo.bar syntax is same for
modules,
classes,
instances

Where new data types really help:

extend the language!

like numpy seamlessly adds array and matrix ... for science!

Features that really make classes helpful:

inheritance
and
polymorphism

inheritance:

define new types
from old,
only describe the
differences

eliminates duplicated code

polymorphism:

same operations (names) do the right thing with different types

eliminates if..elif..else branching on data type

often arises when building many similar components that are not quite the same

only gain advantages if you define several/many related classes: design problem

examples:

GUIs,
other system software,
collections with for x in s,
file-like with open/read/
write/close,
simulations, ...

Another possible advantage:

style / psychology
active agents,
not passive data

Also:

helps to understand Python builtin types, operators, etc.

(more on this next week, all those __*__ methods)

Classes and objects have lots of complications

I'll discuss only typical, recommended practices and style.

Objects combine data and behavior.

If you only have one, a class probably isn't needed.

Unit B
(25)

mechanism/syntax

type vs. object

Class is executed like def.

class vs. object (type)

class syntax creates new types

```
>>> type(list)
>>> <type 'type'>
>>> class C(object):
...     pass
>>> c = C()
>>> type(C)
>>> <type 'type'>
```

class vs. instance

dot operator does
attribute lookup

can be spaces in between

attributes can be added to any instance or class

that makes them a kind of a container for variable names

a namespace

objects are mutable

add and remove attributes

classes are dynamic containers

not limited to their initial definition

hasattr()

getattr()

setattr()

dict

hasattr() & getattr() lookup on instance will fallback to class

set always changes the instance

classes are

data and behavior

when you have a few functions which all take the same parameter

it may be time to think about making them into methods on a class

in Python, that common parameter is explicit

"self" by convention

methods are also just attributes

same as how def works in a module

it creates a name pointing to a function value in the current namespace

demo:

function to method conversion

for builtin data types,
equality checking
looks inside

instances are
different by default

can override

copy() does a shallow copy of objects

same as lists

remember deepcopy?

__init__

warning ch 15 is not typical Python style

ch 17 is typical style

unusual to create empty instances, then assign attributes from outside

adding new attributes to instances is legal

it will cause you trouble eventually

strongly recommend to __init__ all attributes to a default value.

same thing:

set a variable to a default value before conditionally setting it

TypeError, count of arguments / confusing

__str__, __unicode__, __add__, __radd__

next week

objects are mutable

if you pass them
to a function,
it can change them

nested objects 15.3

```
box = Rectangle()
box.width = 100.0
box.height = 200.0
box.corner = Point()
box.corner.x = 0.0
box.corner.y = 0.0
```

sharing and copy, 15.6

```
>>> box2 = copy.copy(box)
>>> box2 is box
False
>>> box2.corner is box.corner
True
```

some thoughts

type based dispatch: isinstance()

"Five Minute Multi-methods in Python" by GvR

http://www.artima.com/weblogs/viewpost.jsp?thread=101605

usually use 'duck typing' instead

inheritance

shared implementation

"is a" relationship

composition

class contains other classes

instance contains other instances

"has a" relationship

prefer composition over inheritance

do they 'really' share all methods?

LSP

http://stackoverflow.com/questions/49002/prefer-composition-over-inheritance/53354#53354

"Here's a program design suggestion: whenever you override a method, the interface of the new method should be the same as the old. It should take the same parameters, return the same type, and obey the same preconditions and postconditions. If you obey this rule, you will find that any function designed to work with an instance of a superclass, like a Deck, will also work with instances of subclasses like a Hand or PokerHand. If you violate this rule, your code will collapse like (sorry) a house of cards."

ThinkPython 18.10

Lab B
(20)

```
git pull
svn update
```

Lab B

- play with classes

  (see quiz06.py for a starting place, if you like)
- python -i

Unit C
(25)

simulation case study

Lab C
(20)

```
git pull
svn update
```

Lab C

- Experiment with the simulation code.
  Ideas:
  Add prints.
  Break code.
  Add collision detection.
  Add a new class which works in simulation.

wrapup

Assignment

- For a portion of the system you're responsible for testing:
  Make a few classes to represent that portion. (No more than three or four classes)
- Imagine that these classes can automatically control the system, what methods would you find most useful?
- Write a few of them with a mock implementation. (accept parameters, update state, print out current state, etc)
- Write a bit of code using these objects to set up an (imaginary) test environment.