

Python for EECS Majors 2

Intermediate Python

Scott Wolchok
`scott@wolchok.org`

Overview

- Following along
- Exceptions
- Classes
- Web development teaser

Following Along

- Install Python 2.6 from <http://python.org/>
- ssh `username@login.engin.umich.edu` and run `python2.5`
- Don't use Python 3.x; it's not ready (poor library support)

Following Along

- Slides at <http://scott.wolchok.org/pytalks/> (both last semester and this one)
- Last talk topics were:
 - Why Python
 - Basic data types
 - Loops, if/elif/else
 - Functions
 - Examples & further reading

Exceptions

- Handling: try/except (not try/catch)
- Throwing: raise (not throw)
- Defining: inherit from Exception

Exceptions Example

```
class MyMathError(Exception): pass  
def fib(n):  
    if n < 0:  
        raise MyMathError('no negative fib')  
    elif n < 2:  
        return n  
    return n + fib(n-1)
```

Exceptions Example cont.

```
def fib_loop():  
    arg = raw_input()  
    while arg != 'quit':  
        try:  
            arg = int(arg)  
            print fib(arg)  
        except (ValueError, MyMathError) as e:  
            print e  
        arg = raw_input()
```

Classes

- Differences from C++:
 - All members **public** and **virtual**
 - Classes are objects too

```
class Spam(object):
```

```
    i = 3
```

```
    def f(self):
```

```
        return 'hi'
```

```
o = Spam()
```

```
print Spam.i, o.i, o.f() # 3, 3, 'hi'
```

Constructors

```
class MyComplex(object):  
    def __init__(self, real, imag):  
        self.real = real  
        self.imag = imag  
  
j = MyComplex(0, 1)  
j.real = 7 # Fine, attrs are public  
j.parrot = 'dead' # Fine, no declaration  
                # needed
```

Methods

```
class Foo(object):  
    def f(self, arg):  
        print arg  
x = Foo()  
x.f('hello')  
Foo.f(x, 'goodbye')
```

A Sense of Self

- self is like this in C++ and Java, but is never implicit

```
x = 7
```

```
class Printer(object):
```

```
    def __init__(self):
```

```
        self.x = 'Hi there!'
```

```
    def print_x(self):
```

```
        print x
```

```
Printer().print_x() # Prints 7, not Hi there!
```

Binding

`xf = x.f`

`xf()` # Same as `x.f()`

- When a method is looked up as an attribute of an object, it is *bound* to the object on which it is looked up
- When looked up on a class, it is *unbound* and must be passed an object
- Compare to `mem_fun` in `<functional>`
- Why? Callbacks!

Inheritance

```
class Derived(Base):
```

```
    def __init__(self):
```

```
        Base.__init__(self) # Let Base ctor run
```

```
        self.extra = 'spam'
```

```
d = Derived()
```

- Attributes are looked up first in the object, then its class, then base class, etc.
- Overriding works like in C++, but `__init__` is not an exception!

Iterators

- For loops call the built-in **iter()** function on the object being looped over
- Iterator protocol: the **next()** method returns the next element and raises an instance of **StopIteration** when done

```
class Reverse(object):  
    def __init__(self, data):  
        self.data = data  
        self.idx = len(data)  
    def __iter__(self):  
        return self # We implement next()  
    def next(self):  
        if not self.idx:  
            raise StopIteration()  
        self.idx -= 1  
        return self.data[self.idx]
```

```
class Sorted(object):  
    def __init__(self, data):  
        self.data = d = list(data)  
        d.sort()  
    def __iter__(self):  
        return iter(self.data)  
for x in Sorted(L):  
    ...
```

Generators

- Easier way to define iterators

```
def reverse(seq):
```

```
    for idx in range(len(seq) - 1, -1, -1):
```

```
        yield seq[idx]
```

- The yielded values are returned by successive calls to next()

Generator expressions

- Recall list comprehensions from last time:
`L = [x*x for x in range(10)]` # 0, 1, 4, 9, ...
- Similar syntax, but generator, not list:
`print sum(x*x for x in range(10))`
- Lots of good stuff in the **itertools** module

Duck typing

- Python doesn't have formal interfaces
- Rely on method names instead
- “If it looks like a duck and quacks like a duck, it must be a duck.”
- C++ templates work this way too!

Duck typing example

```
class Duck(object):  
    def quack(self):  
        print 'Quack!'  
class Goose(object):  
    def quack(self):  
        print 'Honk!'  
ducks = [Duck(), Duck(), Goose()]  
for d in ducks: d.quack()
```

Destructors?

- They don't exist in Python!
- Not obsessed with memory management
- If you need to clean up, **try/except/finally**
 - Also **with**

And now for something completely different...
Web Programming!

Tracking CAEN Linux Users

- Want to track targets on CAEN machines
- Web site should show recent sightings

Setting Up

- `scott.wolchok.org/projects/hostinfo`
- `scott.wolchok.org/projects/netuse_client.pl`
- `scott.wolchok.org/projects/findincaen.py`
- Change `/path/to/netuse` to the path to the directory where you saved everything
- `apt-get install python-cherrypy3 python-paramiko`
 - www.cherrypy.org
 - www.lag.net/paramiko/

Brief CherryPy Overview

- Classes correspond to directories
- Methods return web pages
- Arguments are query parameters

```
import cherrypy
class Root(object):
    def index(self):
        return 'Hello!'
    index.exposed = True # Functions are
                        # objects
if __name__ == '__main__':
    cherrypy.quickstart(Root())
```

findincaen Background

- One function: `find_in_caen(targets)`
 - Returns a list of (targets, machine) pairs
 - Takes a little while to run
- ```
find_in_caen(['swolchok'])
[(['swolchok'], 'login.engin.umich.edu')]
```

# Python Threads

- In the **threading** module
- Mostly what you're used to from 482
- Threads: sequences of instructions executing at the same time
- Share variables, open files, etc.
- Lets us do multiple things at once

```
class WebFindInCaen(object):
def __init__(self, targets):
 self.targets = targets
 self.found = [] # From find_in_caen()
 self.last_scan = 'none yet'
 self.rescan_cond = threading.Condition()
 self.scan_thread = threading.Thread(
 target=self._scan_caen)
 self.scan_thread.daemon = True
 self.scan_thread.start()
```

```
def _scan_caen(self):
 self.rescan_cond.acquire()
while True:
 self.found = find_in_caen(self.targets)
 self.last_scan = datetime.datetime.now()
 time.sleep(300)
```

```
def index(self):
 cherrypy.response.headers['Content-Type'] =
 'text/plain'
 tgts = self.targets
 page = ['Last scan: %s\ ' % self.last_scan,
 'Targets %s found at:' % ', '.join(tgts)]
 f = self.found
 for ts, host in f:
 page.append('%s\t%s' % ', '.join(ts), host)
 return '\n'.join(page)
index.exposed = True
```

# Adding Targets

```
def addtarget(self, target):
 cherrypy.response.headers['Content-Type'] =
 'text/plain'
 self.targets.append(target)
return 'Added target %s' % target
```

# Forcing a rescan

```
def rescan(self):
 cherrypy.response.headers['Content-Type'] =
 'text/plain'
 if self.rescan_cond.acquire(blocking=False):
 self.rescan_cond.notify()
 self.rescan_cond.release()
 return 'Started another scan'
 else:
 return 'Already scanning'
```

# Forcing a rescan (cont.)

```
def _scan_caen(self):
 self.rescan_cond.acquire()
while True:
 self.found = find_in_caen(self.targets)
 self.last_scan = datetime.datetime.now()
 self.rescan_cond.wait(300)
 # time.sleep(300)
```

**Questions?**

# Acknowledgement

- Talk heavily derived from the Python tutorial at <http://docs.python.org/tutorial/>, which is Copyright © 2001-2010 Python Software Foundation; All Rights Reserved (see <http://docs.python.org/license.html>)