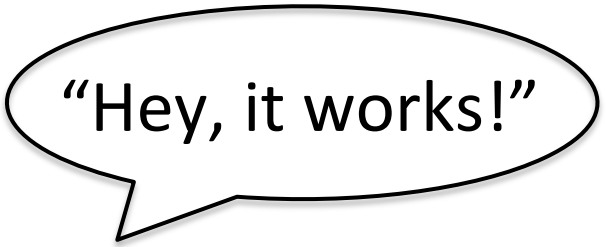
Developing via Testing & Debugging

You know, those error
messages can be pretty handy…

Terri J. Brandt
Slides adapted from Dan Starr and Paul Ivanov.

Verify that it works!



Who said that?

Can we create a system where a programmer can tell the user how to verify that the code works for them?

Errors, Exceptions, and Traceback, oh my!

Errors, Exceptions, and Traceback, oh my!

Syntax Errors:

- Caught by Python parser, prior to execution
- arrow marks the last parsed command / syntax, which gave an error

```
In [1]: while True print 'Hello world'
  File "<ipython-input-1-f4b9dbd125c8>", line 1
    while True print 'Hello world'
                   ^
SyntaxError: invalid syntax
```

Exceptions:

- Caught during runtime

```
In [2]: (1/0)
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-de3693740b15> in <module>()
----> 1 (1/0)

ZeroDivisionError: integer division or modulo by zero
```

Errors, Exceptions, and Traceback, oh my!

Traceback module provides utilities to render python Traceback objects.

Allows a program to:

- Catch an exception within a try/except statement
- Print the traceback, and
- Continue running

File: tryexcept1.py

```
import traceback
def example1():
    try:
        raise SyntaxError, "example"
    except:
        traceback.print_exc()
    print "...still running..."
```

```
In [1]: import tryexcept1
In [2]: tryexcept1.example()
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-2-e4dc9ead3b7a> in <module>()
----> 1 tryexcept1.example()

AttributeError: 'module' object has no attribute 'example'
```

Errors, Exceptions, and Traceback, oh my!

Traceback module provides utilities to render python Traceback objects.

Access Traceback's elements (filename, line number, function name, text):

File: tryexcept1.py

```
def example2():
    """ Access the (filename, etc) of each Traceback stack element.
    """
    try:
        raise SyntaxError
    except:
        stack_list = traceback.extract_stack()
        for (filename, linenum, functionname, text) in stack_list:
            print "%s:%d %s()" % (filename, linenum, functionname)
    print "...still running..."
```

Errors, Exceptions, and Traceback, oh my!

```
try:
    raise SyntaxError
except:
    stack_list = traceback.extract_stack()
    for (filename, linenum, functionname, text) in stack_list:
        print "%s:%d %s()" % (filename, linenum, functionname)
print "...still running..."
```

```
In [3]: tryexcept1.example2()
/opt/local/bin/ipython:7 <module>()
/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/IPython/frontend/terminal/ipapp.py:389 launch_new_instance()
/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/IPython/frontend/terminal/ipapp.py:363 start()
/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/IPython/frontend/terminal/interactiveshell.py:467 mainloop()
/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/IPython/frontend/terminal/interactiveshell.py:586 interact()
/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/IPython/core/interactiveshell.py:2617 run_cell()
/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/IPython/core/interactiveshell.py:2696 run_ast_nodes()
/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/IPython/core/interactiveshell.py:2746 run_code()
<ipython-input-3-5564eee6dc6a>:1 <module>()
tryexcept1.py:16 example2()
...still running...
```

Plus logging!

Fancier debug w print statements piped to log file.

Chop code up into segments with log statements…

…and then record in .log file for later comprehension.

Plus logging!

Logging is useful when:

- Recording non-fatal errors
  - e.g.: Tracebacks caught with try/except statements
- Varying error/warning severity levels are needed
- Generating high volumes of diagnostic output
- Recording errors separately from standard I/O print statements

File: loggin1.py

```
import logging
LOG_FILENAME = 'loggin1.log'
logging.basicConfig(filename=LOG_FILENAME,level=logging.WARNING)

def make_logs():
    logging.debug('This is a debug message')
    logging.warning('This is a warning message')
    logging.error('This is an error message')
```

Log Levels:
NOTSET = 0
DEBUG = 10
INFO = 20
WARN = 30
WARNING = 30
ERROR = 40
CRITICAL = 50
FATAL = 50

```
In [1]: import loggin1
In [2]: loggin1.make_logs()

$ cat loggin1.log
WARNING:root:This is a warning message
ERROR:root:This is an error message
```

Plus logging!

File: loggin2.py

Using time-stamps and formatting:

Log Levels:
NOTSET = 0
DEBUG = 10
INFO = 20
WARN = 30
WARNING = 30
ERROR = 40
CRITICAL = 50
FATAL = 50

```
import logging
logger = logging.getLogger("some_identifier")
logger.setLevel(logging.INFO)
ch = logging.StreamHandler()
ch.stream = open("loggin2.log", 'w')
formatter = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s - %(message)s")
ch.setFormatter(formatter)
logger.addHandler(ch)

def make_logs():
    logger.info("This is an info message")
    logger.debug("This is a debug message")
    logger.warning("This is a warning message")
    logger.error("This is an error message")
```

Plus logging!

```
import logging
logger = logging.getLogger("some_identifier")
logger.setLevel(logging.INFO)
ch = logging.StreamHandler()
ch.stream = open("loggin2.log", 'w')
formatter = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s - %(message)s")
ch.setFormatter(formatter)
logger.addHandler(ch)

def make_logs():
    logger.info("This is an info message")
    logger.debug("This is a debug message")
    logger.warning("This is a warning message")
    logger.error("This is an error message")
```

```
In [1]: import loggin2
In [2]: loggin2.make_logs()

$ cat loggin2.log
2013-06-14 00:28:23,193 - some_identifier - INFO - This is an info message
2013-06-14 00:28:23,193 - some_identifier - WARNING - This is a warning message
2013-06-14 00:28:23,194 - some_identifier - ERROR - This is an error message
```

Assert

Use assert for error catching statements.

Assert statements can be disabled with:

- optimize flags: python -O or
- system environment variable: PYTHONOPTIMIZE

File: my_assertions.py

```
def do_string_stuff(val):
    assert type(val) == type("")
    print ">" + val + "< length:", len(val)
```

```
In [1]: import my_assertions
In [2]: my_assertions.do_string_stuff('cats')
>cats< length: 4
In [3]: my_assertions.do_string_stuff(3.14)
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-3-a5be41a03ad1> in <module>()
----> 1 my_assertions.do_string_stuff(3.14)

12_Testing/my_assertions.pyc in do_string_stuff(val)
      1 def do_string_stuff(val):
----> 2     assert type(val) == type("")
      3     print ">" + val + "< length:", len(val)
      4
      5 def do_string_stuff_better(val):

AssertionError
```

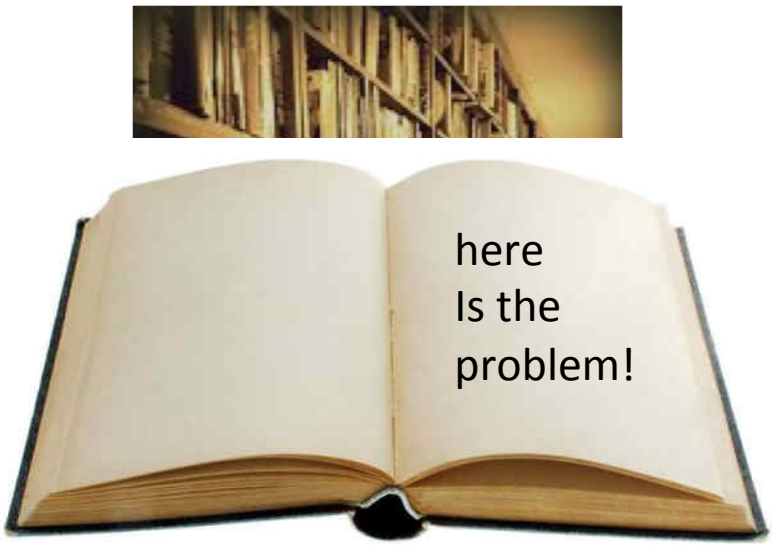
Assert

Use assert for error statements.

^more informative!

File: my_assertions.py

```
def do_string_stuff_better(val):
    val_type = type(val)
    assert val_type == type(""), "Given a %s" % (str(val_type))
    print ">" + val + "< length:", len(val)
```




```
In [4]: my_assertions.do_string_stuff_better(3.14)
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-4-9bc23bc081fe> in <module>()
----> 1 my_assertions.do_string_stuff_better(3.14)

/Users/tjbrand1/TerrisStuff/pythonLearning/bootcamp/
python-bootcamp/Lectures/12_Testing/12_Testing/
my_assertions.pyc in do_string_stuff_better(val)
      5 def do_string_stuff_better(val):
      6     val_type = type(val)
----> 7     assert val_type == type(""), "Given a %s" %
(str(val_type))
      8     print ">" + val + "< length:", len(val)
      9

AssertionError: Given a <type 'float'>
```

Python Testing Tools & Packages

A test tool searches directories for modules and files which either:

- have filenames which are identified for testing use
  - (generally by using a “Test” or “test” substring)
- or files which contain classes and functions which match a substring identifier or regular expression

Unit testing software evaluates the identified files’ and modules’ testing functions and assert statements.
A tool such as “nose” summarizes which tests passed or failed.

Several tools and frameworks interface with other projects to provide additional diagnostic tools such as:

- a debugger (pdb)
- coverage: how much of the source code is used when executed

Several older testing tools are still used (often in other tools):

- unittest, pyUnit

Modern testing tools:

- nose, py.test

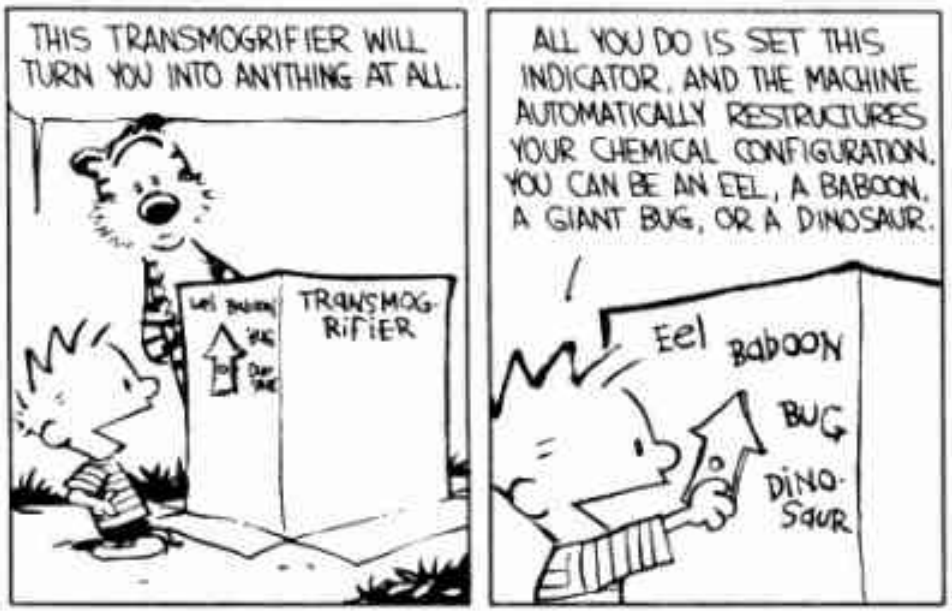
Focus on the “nose” tool due to its breadth and popularity.

A simple nose testing example:

Transmogrify! thing1 into thing2…

File: example1/nose_example1.py

```
class Transmogrifier:
    """ An important class
    """
    def transmogrify(self, person):
        """ Transmogrify someone
        """
        transmog = {'calvin':'tiger',
                    'hobbes':'chicken'}
        new_person = transmog[person]
        return new_person

def test_transmogrify():
    TM = Transmogrifier()
    for p in ['Calvin', 'Hobbes']:
        assert TM.transmogrify(p) != None
```



Nose:

Examines all files (except executables) for functions named with “test” or “Test”, matching REGEXP: ((?:^|[\b_\.-])[Tt]est).

File: example1/nose_example1.py

```
class Transmogrifier:
    """ An important class
    """
    def transmogrify(self, person):
        """ Transmogrify someone
        """
        transmog = {'calvin':'tiger',
                    'hobbes':'chicken'}
        new_person = transmog[person]
        return new_person

def test_transmogrify():
    TM = Transmogrifier()
    for p in ['Calvin', 'Hobbes']:
        assert TM.transmogrify(p) != None
```

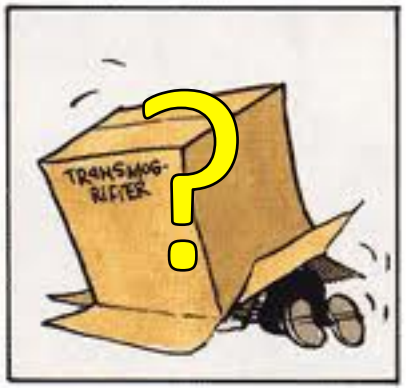
Finds:

- test_transmogrify()
- Test_transmogrify()
- Testtransmogrify()
- Transmogrify_test()

Does not find:

- transmogrifyTest()
- sometest()

Nose:

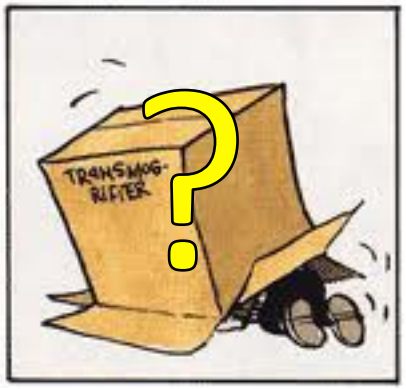


File: example1/nose_example1.py

```
class Transmogrifier:
    """ An important class
    """
    def transmogrify(self, person):
        """ Transmogrify someone
        """
        transmog = {'calvin':'tiger',
                    'hobbes':'chicken'}
        new_person = transmog[person]
        return new_person

def test_transmogrify():
    TM = Transmogrifier()
    for p in ['Calvin', 'Hobbes']:
        assert TM.transmogrify(p) != None
```

```
$ cd example1/
$ nosetests-2.7 --all-modules
E
======================================================================
ERROR: nose_example1.test_transmogrify
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/opt/…/lib/…/nose/case.py", line 197, in runTest
    self.test(*self.arg)
  File "/Users/…/nose_example1.py", line 19, in
test_transmogrify
    assert TM.transmogrify(p) != None
  File "/Users/…/nose_example1.py", line 12, in transmogrify
    new_person = transmog[person]
KeyError: 'Calvin'
----------------------------------------------------------------------
Ran 1 test in 0.012s
FAILED (errors=1)
```

Nose:



File: example1/nose_example1.py

```
class Transmogrifier:
    """ An important class
    """
    def transmogrify(self, person):
        """ Transmogrify someone
        """
        transmog = {'calvin':'tiger',
                    'hobbes':'chicken'}
         new_person = transmog[person.lower()]
        return new_person


def test_transmogrify():
    TM = Transmogrifier()
    for p in ['Calvin', 'Hobbes']:
        assert TM.transmogrify(p) != N
```

```
$ nosetests-2.7 nose_example1_fixed1.py
.
----------------------------------------------------------------
Ran 1 test in 0.001s
OK
```

Says who?
Doctests:

The doctest module:

- scans through all docstrings in a module
- executes any line starting with a '>>>'
- compares the actual output with the docstring expectation

File: doctests_example.py

```
def multiply(a, b):
  """
  'multiply' multiplies two numbers and
returns the result.

  >>> multiply(0.5, 1.5)
  0.75
  >>> multiply(-1, 1)
  -1
  """
  return a*b
```

```
$ nosetests-2.7 --with-doctest --doctest-tests
doctests_example.py
.
----------------------------------------------------------------
Ran 1 test in 0.006s
OK
```

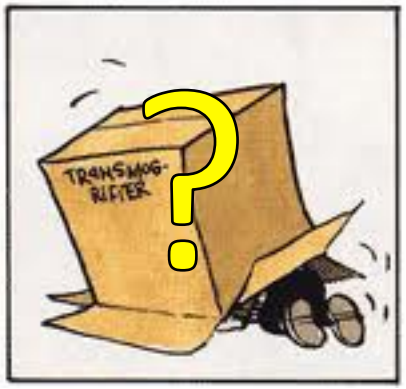
Says who? Doctests:

File: doctests_example.py

```
def multiply(a, b):
  """
  'multiply' multiplies two numbe
returns the result.

  >>> multiply(0.5, 1.5)
  0.75
  >>> multiply(-1, 1)
  -1
  """
  return a*b + 1
```

```
$ nosetests-2.7 --with-doctest --doctest-tests doctests_example.py
F
======================================================================
FAIL: Doctest: doctests_example.multiply
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/opt/…/doctest.py", line 2201, in runTest
    raise self.failureException(self.format_failure(new.getvalue()))
AssertionError: Failed doctest test for doctests_example.multiply
  File "/Users/…/doctests_example.py", line 1, in multiply
----------------------------------------------------------------------
File "/Users/…/doctests_example.py", line 5, in doctests_example.multiply
Failed example:
    multiply(0.5, 1.5)
Expected:
    0.75
Got:
    1.75
----------------------------------------------------------------------
File "/Users/…/doctests_example.py", line 7, in doctests_example.multiply
Failed example:
    multiply(-1, 1)
Expected:
    -1
Got:
    0
----------------------------------------------------------------------
Ran 1 test in 0.013s
FAILED (failures=1)
```

Nose + doctest = ?



File: nose_example1_fixed2.py

```
class Transmogrifier:
    """ An important class
    >>> 3 * 3
    9 """
    def transmogrify(self, person):
        """ Transmogrify someone
        >>> 4*4
        16    """
        transmog = {'calvin':'tiger',
                    'hobbes':'chicken'}
        new_person = transmog[person.lower()]
        return new_person

def test_transmogrify():
    TM = Transmogrifier()
    for p in ['Calvin', 'Hobbes']:
        assert TM.transmogrify(p) != None

def main():
    TM = Transmogrifier()
    for p in ['calvin', 'Hobbes']:
        print p, '->  ZAP!  ->', TM.transmogrify(p)
```

```
$ nosetests-2.7 nose_example1_fixed2.py --with-doctest --
doctest-tests -vv
nose.config: INFO: Ignoring files matching ['^\\.', '^_',
'^setup\\.py$']
nose_example1_fixed2.test_transmogrify ... ok
Doctest: nose_example1_fixed2.Transmogrifier ... ok
Doctest:
nose_example1_fixed2.Transmogrifier.transmogrify ... ok
----------------------------------------------------------------
Ran 3 test in 0.012s
OK
```

Test-Driven Development

Know when you're done!

- Write requirements
- Make tests for the requirements
- Code class/methods
- Test if code satisfies requirements
- Iterate!

Requirements:

- Animal('owl').move == 'fly'
- Animal('cat').move == 'walk'
- Animal('fish').move == 'swim'
- Animal('owl').speak == 'hoot'
- Animal('cat').speak == 'meow'
- Animal('fish').speak == ''

Test-Driven Development

Know when you're done!

- Write requirements
- Make tests for the requirements
- Code class/methods
- Test if code satisfies requirements
- Iterate!

Requirements:

- Animal('owl').move == 'fly'
- Animal('cat').move == 'walk'
- Animal('fish').move == 'swim'
- Animal('owl').speak == 'hoot'
- Animal('cat').speak == 'meow'
- Animal('fish').speak == ''

File: animals/animals_0.py

```
"""
Test Driven Development using animals and Nose testing.
"""

def test_moves():
    assert Animal('owl').move() == 'fly'
    assert Animal('cat').move() == 'walk'
    assert Animal('fish').move() == 'swim'

def test_speaks():
    assert Animal('owl').speak() == 'hoot'
    assert Animal('cat').speak() == 'meow'
    assert Animal('fish').speak() == ''
```

Test-Driven Development

Know when you're done!

- Write requirements
- Make tests for the requirements
- Code class/methods
- Test if code satisfies requirements
- Iterate!

File: animals/animals_0.py

```
"""
Test Driven Development using animals and Nose testing
"""

def test_moves():
    assert Animal('owl').move() == 'fly'
    assert Animal('cat').move() == 'walk'
    assert Animal('fish').move() == 'swim'

def test_speaks():
    assert Animal('owl').speak() == 'hoot'
    assert Animal('cat').speak() == 'meow'
    assert Animal('fish').speak() == ''
```

Requirements:

- Animal('owl').move == 'fly'

```
$ nosetests-2.7 animals_0.py
EE
======================================================================
ERROR: animals_0.test_moves
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/opt/…/nose/case.py", line 197, in runTest
    self.test(*self.arg)
  File "/Users/…/animals_0.py", line 6, in test_moves
    assert Animal('owl').move() == 'fly'
NameError: global name 'Animal' is not defined
======================================================================
ERROR: animals_0.test_speaks
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/opt/…/nose/case.py", line 197, in runTest
    self.test(*self.arg)
  File "/Users/…/animals_0.py", line 11, in test_speaks
    assert Animal('owl').speak() == 'hoot'
NameError: global name 'Animal' is not defined
----------------------------------------------------------------------
Ran 2 tests in 0.007s
FAILED (errors=2)
```

Test-Driven Development

File: animals/animals_1.py

```
"""
Test Driven Development using animals and Nose testing.
"""
class Animal:
    """ This is an animal.
    """
    animal_defs = {'owl':{'move':'fly',
                          'speak':'hoot'},
                   'cat':{'move':'walk',
                          'speak':'meow'},
                   'fish':{'move':'swim',
                          'speak':''}}
    def __init__(self, name):
        self.name = name

    def move(self):
        return self.animal_defs[self.name]['move']

    def speak(self):
        return self.animal_defs[self.name]['speak']
…
```

```
$ nosetests-2.7 animals_1.py -vv
animals_1.test_moves ... ok
animals_1.test_speaks ... ok
----------------------------------------------------
Ran 2 tests in 0.001s
OK
```

Test-Driven Development

File: animals/animals_2.py

```
…
def test_dothings_list():
    """ Test that the animal does the same number of
things as the number of hour-times given.
    """
    times = []
    for i in xrange(5):
        times.append(random() * 24.)
    for a in ['owl', 'cat', 'fish']:
        assert len(Animal(a).dothings(times)) ==\
                                    len(times)

def test_dothings_with_beyond_times():
    for a in ['owl', 'cat', 'fish']:
        assert Animal(a).dothings([-1]) == ['']
        assert Animal(a).dothings([25]) == ['']

def test_nocturnal_sleep():
    """ Test that an owl is awake at night.
    """
    night_hours = [0.1, 3.3, 23.9]
    noct_behaves = Animal('owl').dothings(night_hours)
    for behave in noct_behaves:
        assert behave != 'sleep'
```

Additional requirements:

- Include method which takes list of times (hours = 0 & 24) and returns list of what the animal is (randomly) doing.
- Beyond hours 0 to 24: move() = “”
- An owl’s move()=‘sleep’ during daytime

File: animals/anim

```
...
def test_dothings_list
    """ Test that the an
things as the number
    """
    times = []
    for i in xrange(5):
        times.append(ra
    for a in ['owl', 'cat',
        assert len(Anima



def test_dothings_wi
    for a in ['owl', 'cat',
        assert Animal(a).
        assert Animal(a).

def test_nocturnal_sl
    """ Test that an ow
    """
    night_hours = [0.1,
    noct_behaves = An
    for behave in noct_
        assert behave !=
```

```
$ nosetests-2.7 animals_2.py -vv
animals_2.test_moves ... ok
animals_2.test_speaks ... ok
Test that the animal does the same number of things as the number of hour-times given. ... ERROR
animals_2.test_dothings_with_beyond_times ... ERROR
Test that an owl is awake at night. ... ERROR
======================================================================
ERROR: Test that the animal does the same number of things as the number of hour-times given.
----------------------------------------------------------------------
Traceback (most recent call last):
... line 42, in test_dothings_list
    assert len(Animal(a).dothings(times)) ==\
AttributeError: Animal instance has no attribute 'dothings'
======================================================================
ERROR: animals_2.test_dothings_with_beyond_times
----------------------------------------------------------------------
Traceback (most recent call last):
... line 47, in test_dothings_with_beyond_times
    assert Animal(a).dothings([-1]) == ['']
AttributeError: Animal instance has no attribute 'dothings'
======================================================================
ERROR: Test that an owl is awake at night.
----------------------------------------------------------------------
Traceback (most recent call last):
... line 54, in test_nocturnal_sleep
    noct_behaves = Animal('owl').dothings(night_hours)
AttributeError: Animal instance has no attribute 'dothings'
----------------------------------------------------------------------
Ran 5 tests in 0.003s
FAILED (errors=3)
```

Test-Driven Development

File: animals/animals_2.py

```
…
 def dothings(self, times):
       """ A method which takes a list
           of times (hours between 0 and 24) and
           returns a list of what the animal is
           (randomly) doing.
        - Beyond hours 0 to 24: the animal does: ""
       """
       out_behaves = []
       for t in times:
          if (t < 0) or (t > 24):
             out_behaves.append('')
          elif ((self.name == 'owl') and
             (t > 6.0) and (t < 20.00)):
             out_behaves.append('sleep')
          else:
             out_behaves.append( \
              self.animal_defs[self.name]['move'])
       return out_behaves
```

Additional requirements:

- Include method which takes list of times (hours = 0 & 24) and returns list of what the animal is (randomly) doing.
- Beyond hours 0 to 24: move() = “”
- An owl’s move()=‘sleep’ during daytime

Test-Driven Development

File: animals/animals_3.py

```
…
 def dothings(self, times):
       """ A method which takes a list
           of times (hours between 0 and 24) and
           returns a list of what the animal is
           (randomly) doing.
        - Beyond hours 0 to 24: the animal does: ""
       """
       out_behaves = []
       for t in times:
          if (t < 0) or (t > 24):
             out_behaves.append('')
          elif ((self.name == 'owl') and
             (t > 6.0) and (t < 20.00)):
             out_behaves.append('sleep')
          else:
             out_behaves.append( \
              self.animal_defs[self.name]['move'])
       return out_behaves
```

Additional requirements:

- Include method which takes list of times (hours = 0 & 24) and returns list of what the animal is (randomly) doing.
- Beyond hours 0 to 24: move() = “”
- An owl’s move()=‘sleep’ during daytime

```
$ nosetests-2.7 -vv animals_3.py
animals_3.test_moves ... ok
animals_3.test_speaks ... ok
Test that the animal does the same number of
things as the number of hour-times given. ... ok
animals_3.test_dothings_with_beyond_times ... ok
Test that an owl is awake at night. ... ok
----------------------------------------------------------------------
Ran 5 tests in 0.002s
OK
```

Test-Driven Development

File: animals/animals_3.py

```
…
    c = Animal('cat')
    o = Animal('owl')
    f = Animal('fish')

    times = []
    for i in xrange(10):
        times.append(random() * 24.)
    times.sort()

    c_do = c.dothings(times)
    o_do = o.dothings(times)
    f_do = f.dothings(times)

    for i in xrange(len(times)):
        print "time=%3.3f cat=%s owl=%s fish=%s" % ( \
                times[i], c_do[i], o_do[i], f_do[i])
```

Run Animal.dothings() 10 times!

```
$ python animals_3.py
time=1.225 cat=walk owl=fly fish=swim
time=1.560 cat=walk owl=fly fish=swim
time=5.154 cat=walk owl=fly fish=swim
time=9.270 cat=walk owl=sleep fish=swim
time=10.647 cat=walk owl=sleep fish=swim
time=10.908 cat=walk owl=sleep fish=swim
time=17.164 cat=walk owl=sleep fish=swim
time=17.728 cat=walk owl=sleep fish=swim
time=19.281 cat=walk owl=sleep fish=swim
time=23.760 cat=walk owl=fly fish=swim
```

PDB: Python Debugger

Even with using testing, logging, asserts, some bugs require a more hands-on approach.

PDB:

- Allows interactive access to variables
- Understands python commands
- Has additional debugging commands

Many ways to use PDB:

- Interactively run a program, line by line
- Invoke PDB at a specific line
- Invoke PDB on a variable condition
- Invoke PDB on a Python Traceback
- …

PDB: Invoke Passively

Automatically invoke pdb after a Traceback error in an executed program:

```
... <your module code> ...
def invoke_pdb(type, value, tb):
import traceback, pdb
traceback.print_exception(type, value, tb)
print
pdb.pm()
... <your module code> ...
if __name__ == '__main__':
sys.excepthook = invoke_pdb
... <the rest of your module code> ...
```

Automatically invoking pdb at a certain line in an executed program:

- gives access to variables prior to a Traceback
- allows stepping through subsequent code

```
... <your module code> ...
import pdb;
pdb.set_trace()
... <your module code> ...
```

PDB: Invoke Interactively

Executing pdb.py from shell:

```
BootCamp> python /usr/lib/python2.5/pdb.py nose_example1.py
> /home/training/src/bootdemo/example1/nose_example1.py(2)<module>()
-> """
(Pdb)
```

Using pdb.run():

```
>>> import nose_example1
>>> import pdb
>>> pdb.run('nose_example1.main()')
> <string>(1)<module>()
(Pdb)
```

Where is my pdb.py?

```
>>> import pdb
>>> help(pdb)
...
FILE
/usr/lib/python2.5/pdb.py
...
>>> print pdb.__file__
'/usr/lib/python2.5/pdb.py'
```

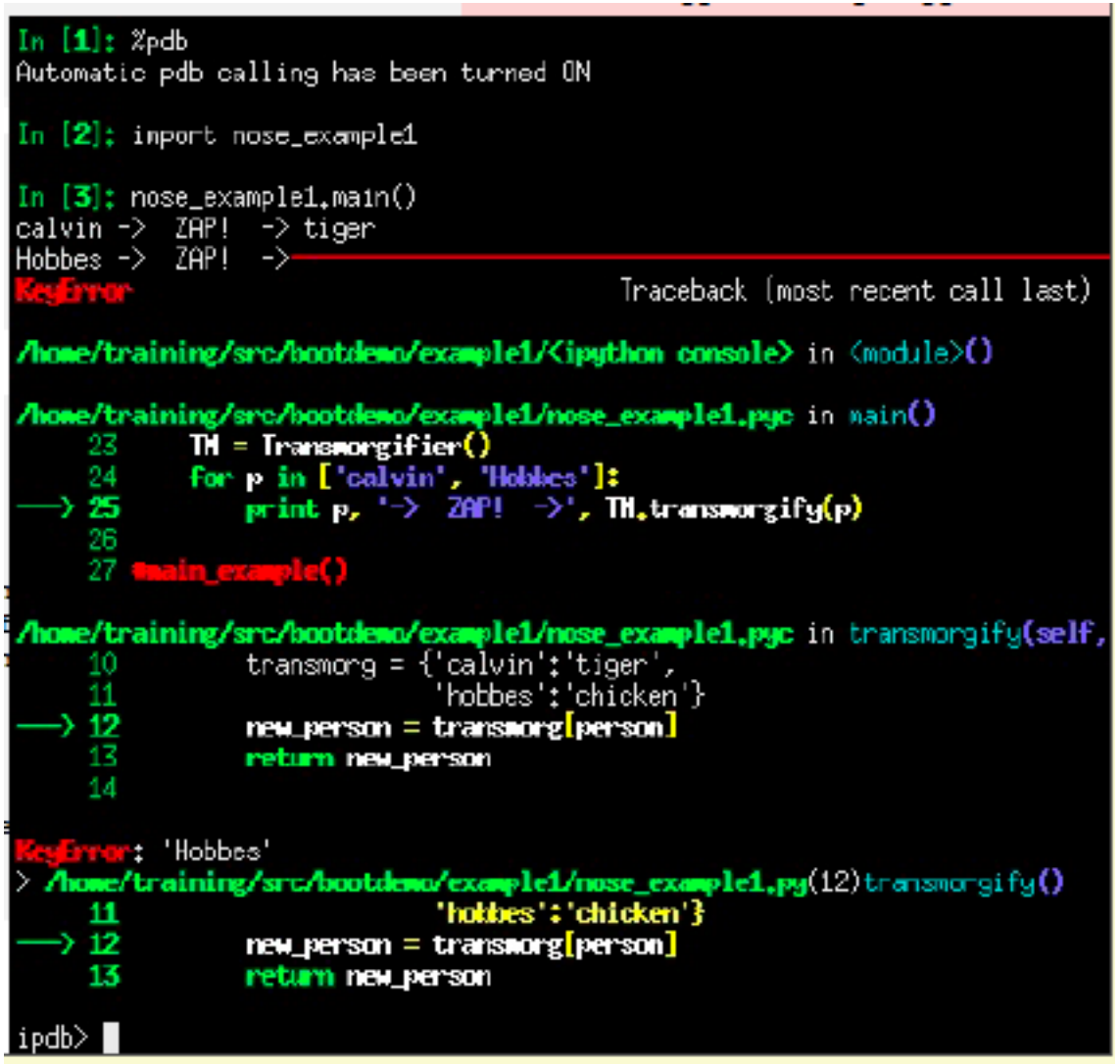
Using pdb.run():

```
>>> nose_example1.main()
calvin -> ZAP! -> tiger
Hobbes -> ZAP! ->
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "nose_example1.py", line 25, in main
    print p, '-> ZAP! ->', TM.transmorgify(p)
  File "nose_example1.py", line 12, in transmorgify
    new_person = transmorg[person]
KeyError: 'Hobbes'
```

```
>>> import pdb
>>> pdb.pm()
> /home/…/bootdemo/example1/
nose_example1.py(12)transmorgify()
-> new_person = transmorg[person]
(Pdb)
```

PDB: Invoke Interactively

within iPython:



PDB: Basic Commands

```
Documented commands (type help <topic>):
========================================
EOF     break   commands   debug    h       l       pp       s         up
a       bt      condition  disable  help    list    q        step      w
alias   c       cont       down     ignore  n       quit     tbreak    whatis
args    cl      continue   enable   j       next    r        u         where
b       clear   d          exit     jump    p       return   unalias

(Pdb) list
  1     """ Nose Example 1
  2  -> """
  3
  4     class Transmorgifier:
  5         """ An important class
  6         """
  7         def transmorgify(self, person):
  8             """ Transmorgify someone
  9             """
 10             transmorg = {'calvin':'tiger',
 11                          'hobbes':'chicken'}
(Pdb)
 12             new_person = transmorg[person]
 13             return new_person
 14
 15
 16     def test_transmorgify():
 17         TM = Transmorgifier()
 18         for p in ['Calvin', 'Hobbes']:
 19             assert TM.transmorgify(p) != None
 20
```

PDB: Basic Commands

```
Documented commands (type help <topic>):
========================================
EOF     break   commands   debug    h       l     pp       s         up
a       bt      condition  disable  help    list  q        step      w
alias   c       cont       down     ignore  n     quit     tbreak    whatis
args    cl      continue   enable   j       next  r        u         where
b       clear   d          exit     jump    p     return   unalias

(Pdb) continue
calvin ->  ZAP!  -> tiger
Traceback (most recent call last):
  File "/usr/lib/python2.5/pdb.py", line 1213, in main
    pdb._runscript(mainpyfile)
  File "/usr/lib/python2.5/pdb.py", line 1138, in _runscript
    self.run(statement, globals=globals_, locals=locals_)
  File "/usr/lib/python2.5/bdb.py", line 366, in run
    exec cmd in globals, locals
  File "<string>", line 1, in <module>
  File "nose_example1.py", line 37, in <module>
    main()
  File "nose_example1.py", line 25, in main
    print p, '->  ZAP!  ->', TM.transmorgify(p)
  File "nose_example1.py", line 12, in transmorgify
    new_person = transmorg[person]
KeyError: 'Hobbes'
Hobbes ->  ZAP!  -> Uncaught exception. Entering post mortem debugging
Running 'cont' or 'step' will restart the program
> /home/training/src/bootdemo/example1/nose_example1.py(12)transmorgify()
-> new_person = transmorg[person]
(Pdb)
```

PDB: (Pdb) help

```
Documented commands (type help <topic>):
========================================
EOF    break  commands   debug    h       l     pp      s         up
a      bt     condition  disable  help    list  q       step      w
alias  c      cont       down     ignore  n     quit    tbreak    whatis
args   cl     continue   enable   j       next  r       u         where
b      clear  d          exit     jump    p     return  unalias

Miscellaneous help topics:
==========================
exec  pdb
```

Nose + doctest + pdb = ?

File: nose_example1_fixed2.py

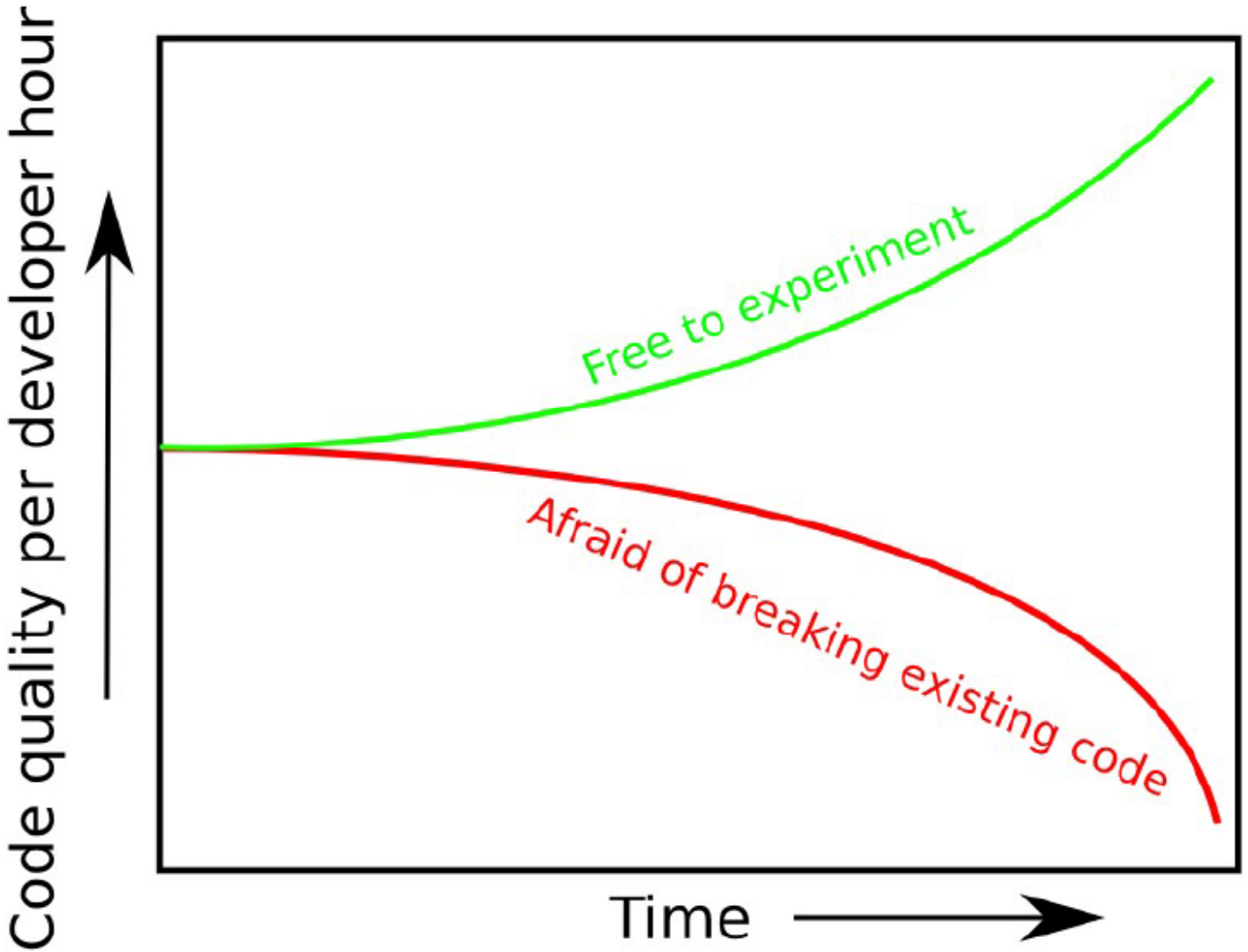
```
class Transmogrifier:
    """ An important class
    >>> 3 * 3
    9 """
    def transmogrify(self, person):
        """ Transmogrify someone
        >>> 4*4
        16    """
        transmog = {'calvin':'tiger',
                    'hobbes':'chicken'}
        new_person = transmog[person.lower()]
        return new_person

def test_transmogrify():
    TM = Transmogrifier()
    for p in ['Calvin', 'Hobbes']:
        assert TM.transmogrify(p) != None

def main():
    TM = Transmogrifier()
    for p in ['calvin', 'Hobbes']:
        print p, '->  ZAP!  ->', TM.transmogrify(p)
```

Allow pdb to be used to look at variables via nose failure of a test.

```
$ nosetests-2.7 --all-modules --pdb
> /Users/…nose_example1.py(12)transmogrify()
-> new_person = transmog[person]
(Pdb) print person
Calvin
```

Use version control…

…so this isn't you!



Breakout

Develop a new animal!
Give the animals more functionality!

Run tests for numpy, scipy, and python
(and report to the code developers!)
http://www.youtube.com/watch?v=kZKex0Y-EXY&feature=youtu.be