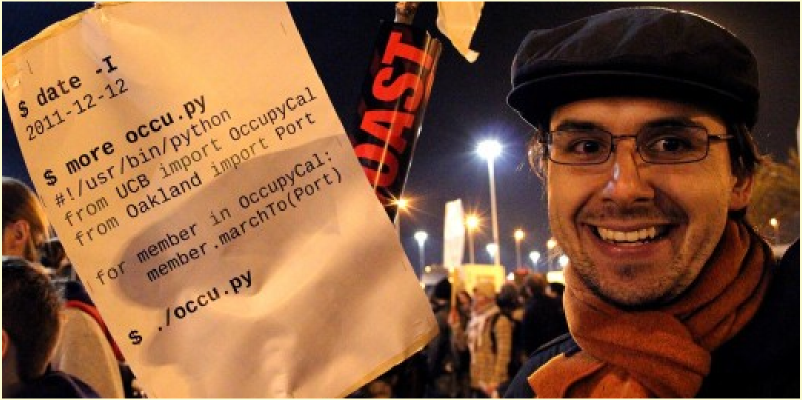
Testing, Debugging, Logging

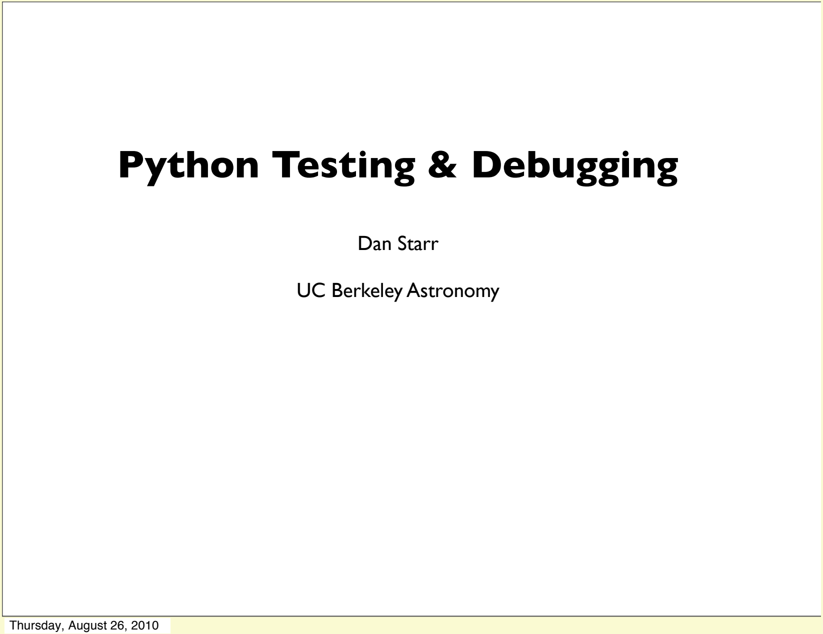
Paul Ivanov

UC Berkeley Vision Science Graduate Program
Redwood Center for Theoretical Neuroscience



```
$ date -I
2011-12-12

$ more occu.py
#!/usr/bin/python
from UCB import OccupyCal
from Oakland import Port

for member in OccupyCal:
    member.marchTo(Port)

$ ./occu.py
```

August 22, 2012



Python Testing & Debugging

Dan Starr

UC Berkeley Astronomy

Thursday, August 26, 2010

credit: Dan Starr (UC Berkeley)

Testing: nose

- Ensures functionality of components
- Test-driven development
- Easier code refactoring

test_simple.py

```
def testTrue():
    assert True == 1

def testFalse():
    assert False == 0
```

In same directory:

```
BootCamp> nosetests
..
----------------------
Ran 2 tests in 0.010s

OK
BootCamp>
```

Debugging: pdb

- Examine variables prior to Traceback errors
- Step through code near suspect code

But First:

- **Errors & Exceptions**
- **Traceback module**
- **Logging**

credit: Dan Starr (UC Berkeley)

Errors & Exceptions

Syntax Errors:

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

```
>>> while True print 'Hello world'
  File "<stdin>", line 1, in ?
    while True print 'Hello world'
                   ^
SyntaxError: invalid syntax
```

Exceptions:

- Caught during runtime

```
>>> (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

credit: Dan Starr (UC Berkeley)

Traceback module

Utilities to render Python Traceback objects

Allows a program to:

- Catch an exception within a try/except
- print the traceback,
- and continue on

file: tryexcept1.py

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

```
>>> import tryexcept1

>>> tryexcept1.example1()
Traceback (most recent call last):
  File "tryexcept1.py", line 5, in example1
    raise SyntaxError, "example"
SyntaxError: example
...still running...
```

credit: Dan Starr (UC Berkeley)

Traceback module

Utilities to render Python Traceback objects

Access to the Traceback element's (filename, line number, function name, text)

file: tryexcept1.py

```
import traceback
def example2():
    try:
        raise SyntaxError, "example"
    except:
        stack_list = traceback.extract_stack()
        for (filename, linenum, functionname, text) in stack_list:
            print "%s:%d %s()" % (filename, linenum, functionname)
    print "...still running..."
```

```
>>> tryexcept1.example2()
/usr/bin/ipython:27 <module>()
/var/lib/python-support/python2.5/IPython/Shell.py:924 mainloop()
/var/lib/python-support/python2.5/IPython/Shell.py:911 OnTimer()
/var/lib/python-support/python2.5/IPython/Shell.py:484 runcode()
/var/lib/python-support/python2.5/IPython/iplib.py:2078 runcode()
<ipython console>:1 <module>()
tryexcept1.py:16 example2()
...still running...
```

credit: Dan Starr (UC Berkeley)

Logging

Logging is useful when:

- Non-fatal errors need to be recorded
    - (e.g.: Tracebacks caught with try/except statements)
- Varying error/warning severity levels are needed
- High volumes of diagnostic output is generated
- Want to record errors separate 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')
```

```
>>> import loggin1
>>> loggin1.make_logs()

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

Log Levels

```
NOTSET = 0
DEBUG = 10
INFO = 20
WARN = 30
WARNING = 30
ERROR = 40
CRITICAL = 50
FATAL = 50
```

credit: Dan Starr (UC Berkeley)

Logging

Using time-stamps and formatting:

file: loggin2.py

```
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")
```

Log Levels

```
NOTSET = 0
DEBUG = 10
INFO = 20
WARN = 30
WARNING = 30
ERROR = 40
CRITICAL = 50
FATAL = 50
```

```
>>> import loggin2
>>> loggin2.make_logs()

BootCamp> cat loggin2.log
2010-08-23 23:01:14,397 - some_identifier - INFO - This is an info message
2010-08-23 23:01:14,398 - some_identifier - WARNING - This is a warning message
2010-08-23 23:01:14,398 - some_identifier - ERROR - This is an error message
```

credit: Dan Starr (UC Berkeley)

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)
```

```
>>> import my_assertions
>>> my_assertions.do_string_stuff('cats')
>cats< length: 4
>>> my_assertions.do_string_stuff(3.14)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "my_assertions.py", line 2, in do_string_stuff
    assert type(val) == type("")
AssertionError
```

More descriptive *assert* error:

```
def do_string_stuff_better(val):
    val_type = type(val)
    assert val_type == type(""), "Given a %s" % (str(val_type))
```

```
>>> my_assertions.do_string_stuff_better(3.14159)
...
AssertionError: Given a <type 'float'>
```

credit: Dan Starr (UC Berkeley)

Python Testing Tools and Packages

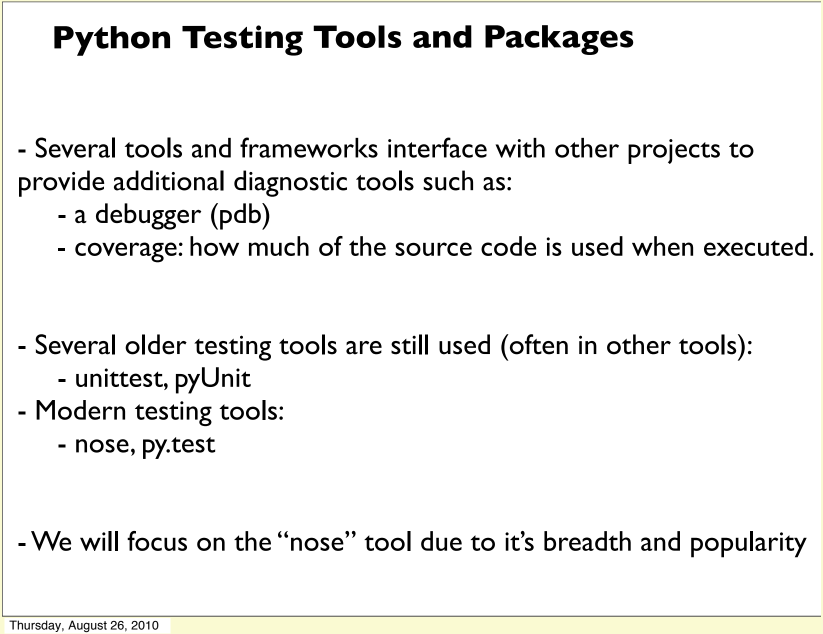
Testing Framework Components

- A test discovery 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 / regular expression.

- Unit testing software then uses these identified files and modules
  - and evaluates their testing functions and assert statements.

- Then a tool such as “nose” summarizes which tests passed or failed.

credit: Dan Starr (UC Berkeley)



Python Testing Tools and Packages

- 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.

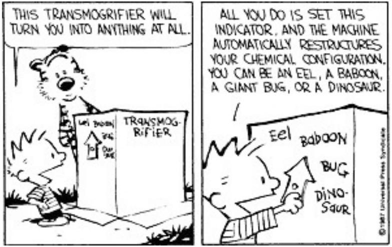
- Several older testing tools are still used (often in other tools):
    - unittest, pyUnit
- Modern testing tools:
    - nose, py.test

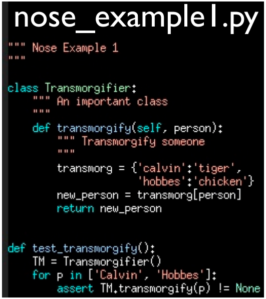
- We will focus on the “nose” tool due to it’s breadth and popularity

Thursday, August 26, 2010

credit: Dan Starr (UC Berkeley)

A simple “nose” testing example


THIS TRANSMOGRIFIER WILL TURN YOU INTO ANYTHING AT ALL.
TRANSMOG-RIFIER
ALL YOU DO IS SET THIS INDICATOR, AND THE MACHINE AUTOMATICALLY RESTRUCTURES YOUR CHEMICAL CONFIGURATION. YOU CAN BE AN EEL, A BABOON, A GIANT BUG, OR A DINOSAUR.
Eel
Baboon
BUG
DINO-SAUR



nose_example1.py

```python
""" Nose Example 1
"""

class Transmorgifier:
    """ An important class
    """
    def transmorgify(self, person):
        """ Transmorgify someone
        """
        transmorg = {'calvin':'tiger',
                     'hobbes':'chicken'}
        new_person = transmorg[person]
        return new_person

def test_transmorgify():
    TM = Transmorgifier()
    for p in ['Calvin', 'Hobbes']:
        assert TM.transmorgify(p) != None
```

credit: Dan Starr (UC Berkeley)

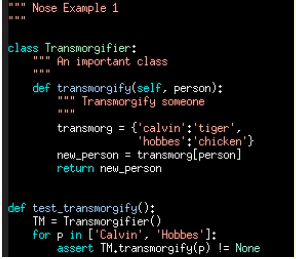
nosetests --all-modules

- Looks at all files (except executables)
- nose examines functions which are named with “test” or “Test”

- names matching REGEXP:

```
((?:^|[\b_\.-])[Tt]est)
```

nose_example1.py



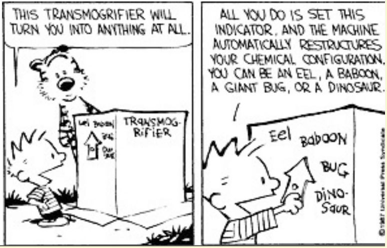
```
""" Nose Example 1
"""

class Transmorgifier:
    """ An important class
    """
    def transmorgify(self, person):
        """ Transmorgify someone
        """
        transmorg = {'calvin':'tiger',
                     'hobbes':'chicken'}
        new_person = transmorg[person]
        return new_person


def test_transmorgify():
    TM = Transmorgifier()
    for p in ['Calvin', 'Hobbes']:
        assert TM.transmorgify(p) != None
```

Finds:

test_transmogrify()
Test_transmogrify()
Testtransmogrify()
transmogrify_Test()

Doesn’t find:

transmogrifyTest()
sometest()

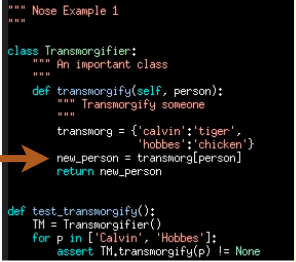

THIS TRANSMOGRIFIER WILL TURN YOU INTO ANYTHING AT ALL.
TRANSMOG-RIFIER
ALL YOU DO IS SET THIS INDICATOR, AND THE MACHINE AUTOMATICALLY RESTRUCTURES YOUR CHEMICAL CONFIGURATION. YOU CAN BE AN EEL, A BABOON, A GIANT BUG, OR A DINOSAUR.
Eel
Baboon
BUG
DINO-SAUR
©1987 Universal Press Syndicate

Thursday, August 26, 2010

credit: Dan Starr (UC Berkeley)

nosetests

nose_example1.py



```
""" Nose Example 1
"""

class Transmorgifier:
    """ An important class
    """
    def transmorgify(self, person):
        """ Transmorgify someone
        """
        transmorg = {'calvin':'tiger',
                     'hobbes':'chicken'}
        new_person = transmorg[person]
        return new_person


def test_transmorgify():
    TM = Transmorgifier()
    for p in ['Calvin', 'Hobbes']:
        assert TM.transmorgify(p) != None
```

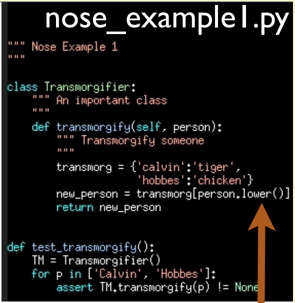
```
BootCamp> cd example1
BootCamp> ls
nose_example1.py
BootCamp> nosetests --all-modules

======================================================================
=
ERROR: nose_example1.test_transmogrify
----------------------------------------------------------------------
-
Traceback (most recent call last):
  File "/usr/lib/python2.5/site-packages/nose-0.11.4-py2.5.egg/nose/
case.py", line 186, in runTest
    self.test(*self.arg)
  File "/home/training/src/bootdemo/example1/example1/
nose_example1.py", line 19, in test_transmogrify
    assert TM.transmogrify(p) != None
  File "/home/training/src/bootdemo/example1/example1/
nose_example1.py", line 12, in transmogrify
    new_person = transmog[person]
KeyError: 'Calvin'

----------------------------------------------------------------------
-
Ran 1 test in 0.003s
FAILED (errors=1)
```

Thursday, August 26, 2010

credit: Dan Starr (UC Berkeley)

nosetests

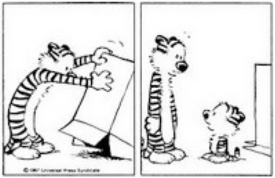
```
BootCamp> cd example1
BootCamp> ls
nose_example1.py
BootCamp> nosetests --all-modules
E
======================================================================
=
ERROR: nose_example1.test_transmogrify
----------------------------------------------------------------------
-
Traceback (most recent call last):
  File "/usr/lib/python2.5/site-packages/nose-0.11.4-py2.5.egg/nose/
case.py", line 186, in runTest
    self.test(*self.arg)
  File "/home/training/src/bootdemo/example1/example1/
nose_example1.py", line 19, in test_transmogrify
    assert TM.transmogrify(p) != None
  File "/home/training/src/bootdemo/example1/example1/
nose_example1.py", line 12, in transmogrify
    new_person = transmog[person]
KeyError: 'Calvin'

----------------------------------------------------------------------
-
Ran 1 test in 0.003s
FAILED (errors=1)
BootCamp> nosetests --all-modules
.
----------------------------------------------------------------------
-
Ran 1 test in 0.003s

OK
BootCamp>
```



nose_example1.py

```
""" Nose Example 1
"""

class Transmorgifier:
    """ An important class
    """
    def transmorgify(self, person):
        """ Transmorgify someone
        """
        transmorg = {'calvin':'tiger',
                     'hobbes':'chicken'}
        new_person = transmorg[person.lower()]
        return new_person


def test_transmorgify():
    TM = Transmorgifier()
    for p in ['Calvin', 'Hobbes']:
        assert TM.transmorgify(p) != None
```

Fixed



credit: Dan Starr (UC Berkeley)

doctests

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
```

The *doctest* module

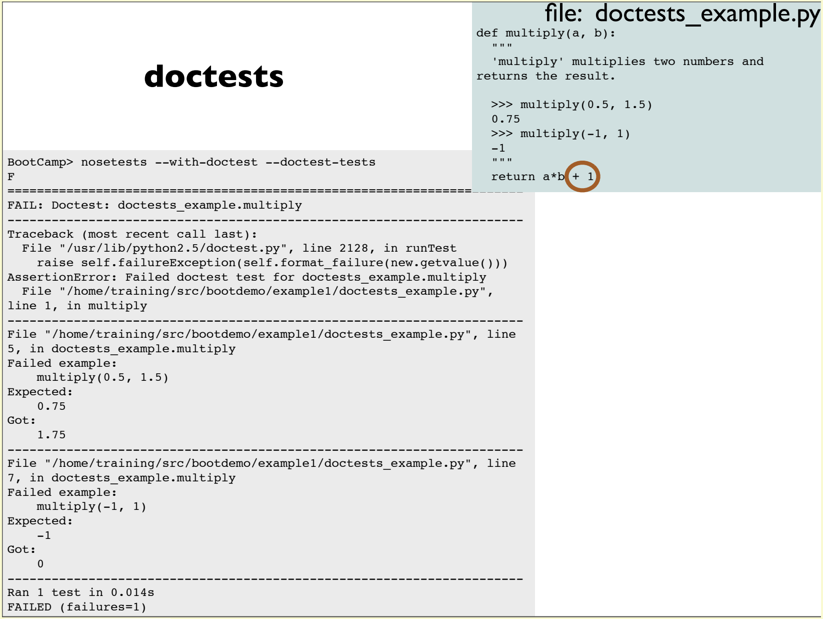
- scans through all of the docstrings in a module

- executes any line starting with a >>>

- compares the actual output with the expected output contained in the docstring.

```
BootCamp> nosetests --with-doctest --doctest-tests
.
----------------------------------------------------------------------
-
Ran 1 test in 0.012s

OK
BootCamp>
```

credit: Dan Starr (UC Berkeley)



doctests

file: doctests_example.py

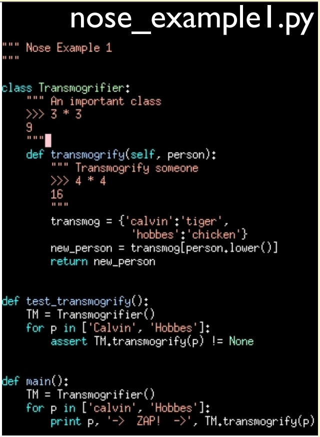
```
def multiply(a, b):
  """
  'multiply' multiplies two numbers and
returns the result.

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

```
BootCamp> nosetests --with-doctest --doctest-tests
F
======================================================================
FAIL: Doctest: doctests_example.multiply
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/usr/lib/python2.5/doctest.py", line 2128, in runTest
    raise self.failureException(self.format_failure(new.getvalue()))
AssertionError: Failed doctest test for doctests_example.multiply
  File "/home/training/src/bootdemo/example1/doctests_example.py",
line 1, in multiply
----------------------------------------------------------------------
File "/home/training/src/bootdemo/example1/doctests_example.py", line
5, in doctests_example.multiply
Failed example:
    multiply(0.5, 1.5)
Expected:
    0.75
Got:
    1.75
----------------------------------------------------------------------
File "/home/training/src/bootdemo/example1/doctests_example.py", line
7, in doctests_example.multiply
Failed example:
    multiply(-1, 1)
Expected:
    -1
Got:
    0
----------------------------------------------------------------------
Ran 1 test in 0.014s
FAILED (failures=1)
```

credit: Dan Starr (UC Berkeley)

doctests

Here we combining doctests and the nosetests from the previous example



nose_example1.py

```
""" Nose Example 1
"""

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)
```

```
BootCamp> nosetests nose_example1.py --with-doctest --doctest-tests -vv
nose_example1.test_transmogrify ... ok
Doctest: nose_example1.Transmogrifier ... ok
Doctest: nose_example1.Transmogrifier.transmogrify ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.011s

OK
BootCamp>
```

credit: Dan Starr (UC Berkeley)

Test Driven Development

Using *nose* testing framework

Toy Problem: Animals

1) start with some requirements
2) make tests for these requirements
3) code the class / methods
4) test again
5) ...iterate steps 1-4...

```
BootCamp> cd animals
```

file: animals_0.py

```
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'
Animal('cat').move   == 'walk'
Animal('fish').move  == 'swim'

Animal('owl').speak  == 'hoot'
Animal('cat').speak  == 'meow'
Animal('fish').speak == ''
```

```
BootCamp> nosetests animals_0.py
EE
======================================================================
ERROR: animals_0.test_moves
...
    assert Animal('owl').move() == 'fly'
NameError: global name 'Animal' is not defined

======================================================================
ERROR: animals_0.test_speaks
...
    assert Animal('owl').speak() == 'hoot'
NameError: global name 'Animal' is not defined

----------------------------------------------------------------------
Ran 2 tests in 0.006s

FAILED (errors=2)
```

credit: Dan Starr (UC Berkeley)

Test Driven Development

- We’ve added an Animal class which meets our requirements
- Run nosetests

file: animals_1.py

```
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']
```

```
BootCamp> nosetests -vv animals_1.py
animals_1.test_moves ... ok
animals_1.test_speaks ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.003s

OK
```

credit: Dan Starr (UC Berkeley)

Test Driven Development

Additional requirements:

- Want an Animal 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: move() = “”
- Also an owl’s move()=’sleep’ during daytime

file: animals_2.py

```
from random import random
............
def test_dothings_list():
    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():
    night_hours = [0.1, 3.3, 23.9]
    noct_behaves = \
            Animal('owl').dothings(night_hours)
    for behave in noct_behaves:
        assert behave != 'sleep'
```

```
BootCamp> nosetests -vv animals_2.py
animals_2.test_moves ... ok
animals_2.test_speaks ... ok
Test that the animal does the same number of things as the number of h
our-times given. ... ERROR
animals_2.test_dothings_with_beyond_times ... ERROR
Test that an owl is awake at night. ... ERROR
======================================================================
...
AttributeError: Animal instance has no attribute 'dothings'
...
----------------------------------------------------------------------
Ran 5 tests in 0.006s
FAILED (errors=3)
```

Thursday, August 26, 2010

credit: Dan Starr (UC Berkeley)

Test Driven Development

- We’ve added functionality to the Animal class which meets our requirements
- Run nosetests

file: animals_3.py

```
.......
    def dothings(self, times):
        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
.......
```

```
BootCamp> nosetests -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 0.006s

OK
BootCamp>
```

credit: Dan Starr (UC Berkeley)

Test Driven Development

file: animals_3.py

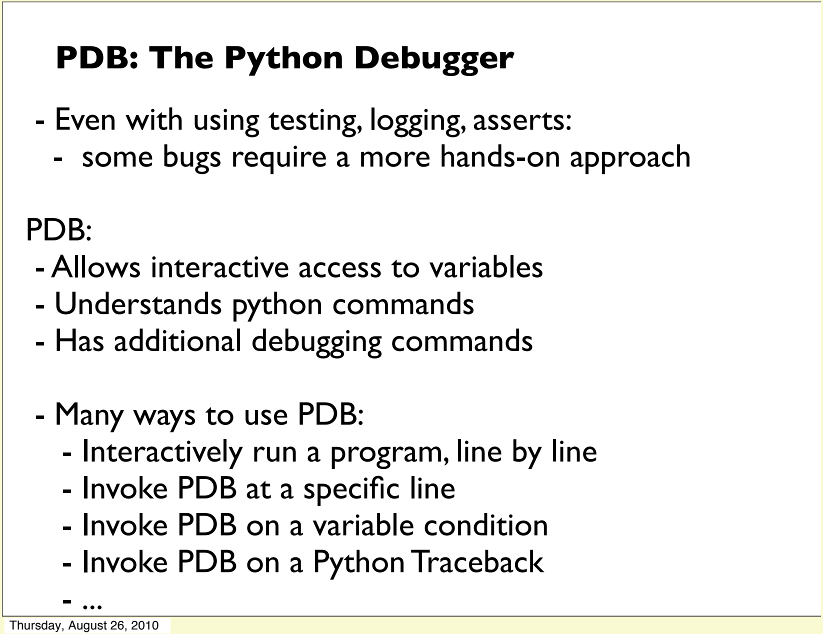
```
.......
    c = Animal('cat')
    o = Animal('owl')
    f = Animal('fish')

    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(10):
        print "time=%3.3f cat=%s owl=%s fish=%s" % ( \
                   times[i], c_do[i], o_do[i], f_do[i])
```

Running Animal.dothings()
for 10 times:

```
BootCamp> python animals_3.py
time=2.943 cat=walk owl=fly fish=swim
time=3.222 cat=walk owl=fly fish=swim
time=5.333 cat=walk owl=fly fish=swim
time=8.535 cat=walk owl=sleep fish=swim
time=8.648 cat=walk owl=sleep fish=swim
time=10.733 cat=walk owl=sleep fish=swim
time=16.024 cat=walk owl=sleep fish=swim
time=20.793 cat=walk owl=fly fish=swim
time=21.507 cat=walk owl=fly fish=swim
time=22.224 cat=walk owl=fly fish=swim
```

Pretty Plain!

credit: Dan Starr (UC Berkeley)



PDB: The 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
  - ...

credit: Dan Starr (UC Berkeley)

PDB: Passively Invoking (pdb)

a) Automatically invoking 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> ...
```

b) 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> ...
```

credit: Dan Starr (UC Berkeley)

PDB: Interactively Starting (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’
```

c) 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)
```

d) Using pdb.run():

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

e) After a Traceback Error (within a Python session)

```
>>> 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/training/src/bootdemo/example1/nose_example1.py(12)transmorgify()
-> new_person = transmorg[person]
(Pdb)
```

credit: Dan Starr (UC Berkeley)

PDB: Interactively Starting (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'
```

c) 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)
```

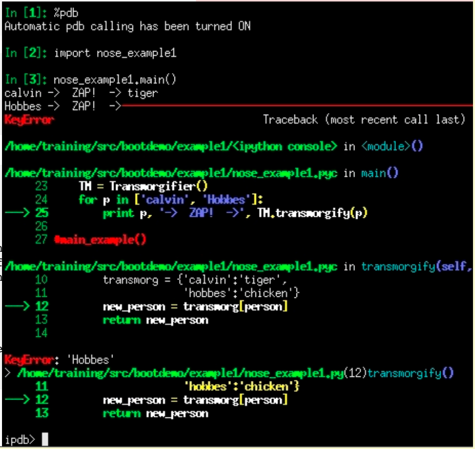
d) Using pdb.run():

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

e) After a Traceback Error (within

```
>>> 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.transmorgif
  File "nose_example1.py", line 12, in tran
    new_person = transmorg[person]
KeyError: 'Hobbes'
>>> import pdb
>>> pdb.pm()
> /home/training/src/bootdemo/example1/nose
-> new_person = transmorg[person]
(Pdb)
```

f) Within IPython...



```
In [1]: %pdb
Automatic pdb calling has been turned ON

In [2]: import nose_example1

In [3]: nose_example1.main()
calvin ->  ZAP!  -> tiger
Hobbes ->  ZAP!  ->---------------------------------------------------------------
KeyError                                  Traceback (most recent call last)

/home/training/src/bootdemo/example1/<ipython console> in <module>()

/home/training/src/bootdemo/example1/nose_example1.pyc in main()
     23     TM = Transmorgifier()
     24     for p in ['calvin', 'Hobbes']:
---> 25         print p, '->  ZAP!  ->', TM.transmorgify(p)
     26
     27 #main_example()

/home/training/src/bootdemo/example1/nose_example1.pyc in transmorgify(self,
     10         transmorg = {'calvin':'tiger',
     11                      'hobbes':'chicken'}
---> 12         new_person = transmorg[person]
     13         return new_person
     14

KeyError: 'Hobbes'
> /home/training/src/bootdemo/example1/nose_example1.py(12)transmorgify()
     11                      'hobbes':'chicken'}
---> 12         new_person = transmorg[person]
     13         return new_person

ipdb> 
```

credit: Dan Starr (UC Berkeley)

PDB: Basic Commands

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

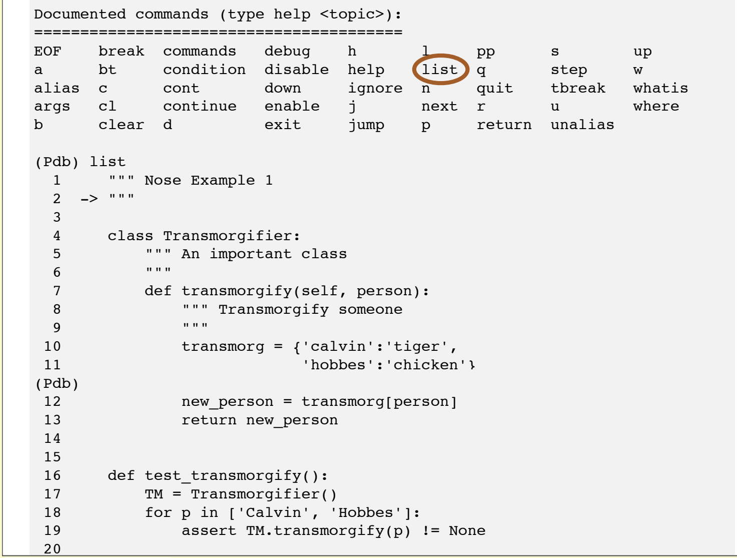
Where is my pdb.py?

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

credit: Dan Starr (UC Berkeley)

file: nose_example1.py

PDB: Basic Commands

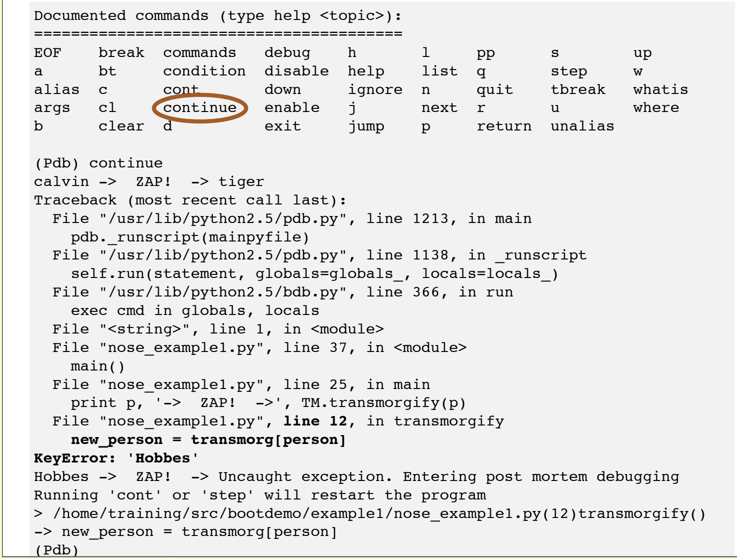


```
Documented commands (type help <topic>):
========================================
EOF    break  commands   debug    h       l     pp      s        up
a      bt     condition  disable  help    list  q       step     w
alias  c      cont       down     ignore  n     quit    tbreak   whatis
args   cl     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
```

credit: Dan Starr (UC Berkeley)

file: nose_example1.py

PDB: Basic Commands

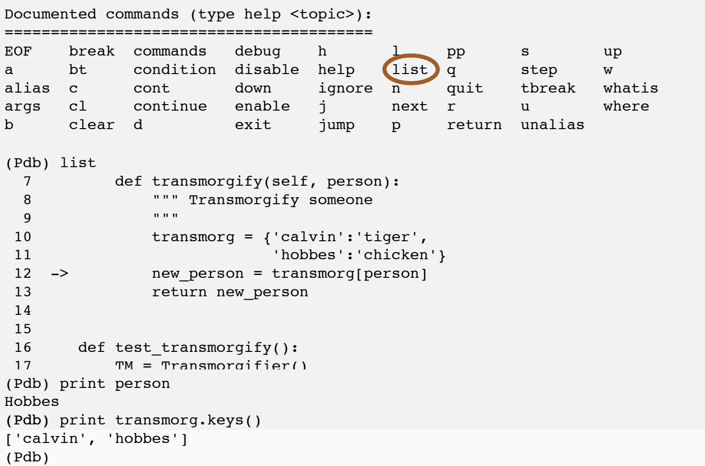


```
Documented commands (type help <topic>):
========================================
EOF    break  commands   debug    h       l     pp      s        up
a      bt     condition  disable  help    list  q       step     w
alias  c      cont       down     ignore  n     quit    tbreak   whatis
args   cl     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)
```

credit: Dan Starr (UC Berkeley)

file: nose_example1.py

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
  7         def transmorgify(self, person):
  8             """ Transmorgify someone
  9             """
 10             transmorg = {'calvin':'tiger',
 11                          'hobbes':'chicken'}
 12  ->         new_person = transmorg[person]
 13             return new_person
 14
 15
 16     def test_transmorgify():
 17         TM = Transmorgifier()
(Pdb) print person
Hobbes
(Pdb) print transmorg.keys()
['calvin', 'hobbes']
(Pdb)
```

credit: Dan Starr (UC Berkeley)

PDB: Basic Commands

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

Where is my pdb.py?

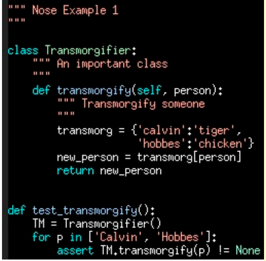
```
>>> import pdb
>>> help(pdb)
    ...
FILE
    /usr/lib/python2.5/pdb.py
    ...
>>> print pdb.__file__
'/usr/lib/python2.5/pdb.py'
```

credit: Dan Starr (UC Berkeley)

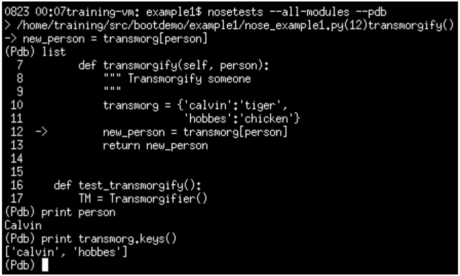
nosetests --all-modules --pdb

- allows pdb to be used to look at variables, via nose failure of a test

nose_example1.py



```
""" Nose Example 1
"""

class Transmorgifier:
    """ An important class
    """
    def transmorgify(self, person):
        """ Transmorgify someone
        """
        transmorg = {'calvin':'tiger',
                     'hobbes':'chicken'}
        new_person = transmorg[person]
        return new_person


def test_transmorgify():
    TM = Transmorgifier()
    for p in ['Calvin', 'Hobbes']:
        assert TM.transmorgify(p) != None
```



```
0823 00:07training-vm: example1$ nosetests --all-modules --pdb
> /home/training/src/bootdemo/example1/nose_example1.py(12)transmorgify()
-> new_person = transmorg[person]
(Pdb) list
  7             def transmorgify(self, person):
  8                 """ Transmorgify someone
  9                 """
 10                 transmorg = {'calvin':'tiger',
 11                              'hobbes':'chicken'}
 12  ->             new_person = transmorg[person]
 13                 return new_person
 14
 15
 16         def test_transmorgify():
 17             TM = Transmorgifier()
(Pdb) print person
Calvin
(Pdb) print transmorg.keys()
['calvin', 'hobbes']
(Pdb) █
```

Thursday, August 26, 2010

credit: Dan Starr (UC Berkeley)

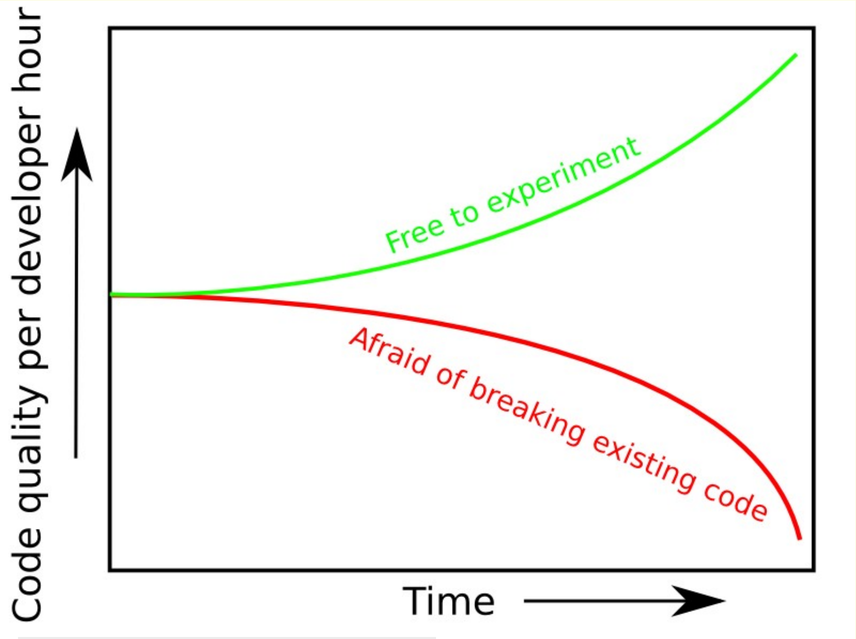
step 0 - version control

Everything.
Aggressively !
(this presentation)

Testing...

Доверяй,
но
проверяй

Trust but verify



credit: Stéfan van der Walt (UC Berkeley)

With gained confidence, leverage abstraction

When you have an itch, Scratch it !

Happy Hacking ! (Breakout)

- add a new animal by first writing the tests
- run tests for numpy, scipy, ipython
- if you find any errors, report them to the projects