Testing, Debugging, Logging

John ZuHone (GSFC, Code 662)

(with slides shamelessly stolen from the UC-Berkeley Boot Camp; credits to Paul Ivanov, Dan Starr, Stéfan van der Walt)

file: test_simple.py

Testing: nose

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

Debugging: pdb

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

But First:

- Errors & Exceptions
- Traceback module
- Logging

Testing: nose

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

file: test_simple.py

```
def testTrue():
    assert True == 1
def testFalse():
    assert False == 0
```

Debugging: pdb

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

But First:

- Errors & Exceptions
- Traceback module
- Logging

Testing: nose

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

Debugging: pdb

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

file: test_simple.py

```
def testTrue():
    assert True == 1
def testFalse():
    assert False == 0
```

```
BootCamp> nosetests
```

<u>But First:</u>

- Errors & Exceptions
- Traceback module
- Logging

Testing: nose

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

file: test_simple.py

```
def testTrue():
    assert True == 1
def testFalse():
    assert False == 0
```

```
BootCamp> nosetests
..
```

Debugging: pdb

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

But First:

- Errors & Exceptions
- Traceback module
- Logging

Testing: nose

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

file: test_simple.py

```
def testTrue():
    assert True == 1
def testFalse():
    assert False == 0
```

```
BootCamp> nosetests
..
----------------------
```

Debugging: pdb

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

<u>But First:</u>

- Errors & Exceptions
- Traceback module
- Logging

Testing: nose

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

file: test_simple.py

```
def testTrue():
    assert True == 1
def testFalse():
    assert False == 0
```

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

Debugging: pdb

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

But First:

- Errors & Exceptions
- Traceback module
- Logging

Testing: nose

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

file: test_simple.py

```
def testTrue():
    assert True == 1
def testFalse():
    assert False == 0
```

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

OK
```

Debugging: pdb

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

<u>But First:</u>

- Errors & Exceptions
- Traceback module
- Logging

Testing: nose

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

Debugging: pdb

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

file: test_simple.py

```
def testTrue():
    assert True == 1
def testFalse():
    assert False == 0
```

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

<u>But First:</u>

- Errors & Exceptions
- Traceback module
- Logging

Testing: nose

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

Debugging: pdb

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

file: test_simple.py

```
def testTrue():
    assert True == 1
def testFalse():
    assert False == 0
```

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

<u>But First:</u>

- Errors & Exceptions
- Traceback module
- Logging

Errors and 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'

```

- **Exceptions:**
  - Caught during runtime

```
>>> (1/0)
Traceback (most recent call last):

```

Errors and 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 ?
```

- **Exceptions:**
  - Caught during runtime

```
>>> (1/0)
Traceback (most recent call last):
```

Errors and 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'
```

- **Exceptions:**
  - Caught during runtime

```
>>> (1/0)
Traceback (most recent call last):
```

Errors and 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'
                   ^
```

- **Exceptions:**
  - Caught during runtime

```
>>> (1/0)
Traceback (most recent call last):
```

Errors and 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):
```

Errors and 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):
```

Errors and 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
```

Errors and 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
```

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: tryexcept0.py

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: tryexcept0.py

```
import traceback
def example0():
    try:
        raise SyntaxError, "example"
    except: traceback.print_exc()
        print "...still running..."
```

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: tryexcept0.py

```
import traceback
def example0():
    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...
```

Traceback Module

Utilities to render Python Traceback objects

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

file: tryexcept1.py

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 example1():
    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..."
```

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 example1():
    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..."
```

```
>>> import tryexcept1
```

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 example1():
    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..."
```

```
>>> import tryexcept1
>>> tryexcept1.example2()
```

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 example1():
    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..."
```

```
>>> import tryexcept1
>>> tryexcept1.example2()
/usr/bin/ipython:27 <module>()
```

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 example1():
    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..."
```

```
>>> import tryexcept1
>>> tryexcept1.example2()
/usr/bin/ipython:27 <module>()
/var/lib/python-support/python2.5/IPython/Shell.py:924 mainloop()
```

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 example1():
    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..."
```

```
>>> import tryexcept1
>>> 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()
```

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 example1():
    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..."
```

```
>>> import tryexcept1
>>> 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()
```

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 example1():
    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..."
```

```
>>> import tryexcept1
>>> 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...
```

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 example1():
    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..."
```

```
>>> import tryexcept1
>>> 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...
```

Logging

<u>Logging is useful when:</u>

- 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

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')
```

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')
```

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

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

Logging

Using time-stamps and formatting:

file: loggin2.py

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")
```

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

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")
```

```
>>> 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
```

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

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

file: my_assertions.py

- More descriptive *assert* error:

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

file: my_assertions.py

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

- More descriptive *assert* error:

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

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')
```

- More descriptive *assert* error:

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

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')
>>> my_assertions.do_string_stuff(3.14)
```

- More descriptive *assert* error:

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

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')
>>> my_assertions.do_string_stuff(3.14)
Traceback (most recent call last):
```

- More descriptive *assert* error:

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

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')
>>> 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
```

- More descriptive *assert* error:

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

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')
>>> 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("")
```

- More descriptive *assert* error:

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

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')
>>> 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:

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

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')
>>> 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))
```

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

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')
>>> 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)
```

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

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')
>>> 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'>
```

Assert

- Use assert for error catching statements
- assert statements can be disabled with optimize flags: or system environment variable:
  - `python -O my_script.py`
  - `export PYTHONOPTIMIZE=True`

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')
>>> 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'>
```

Python Testing Tools and Packages

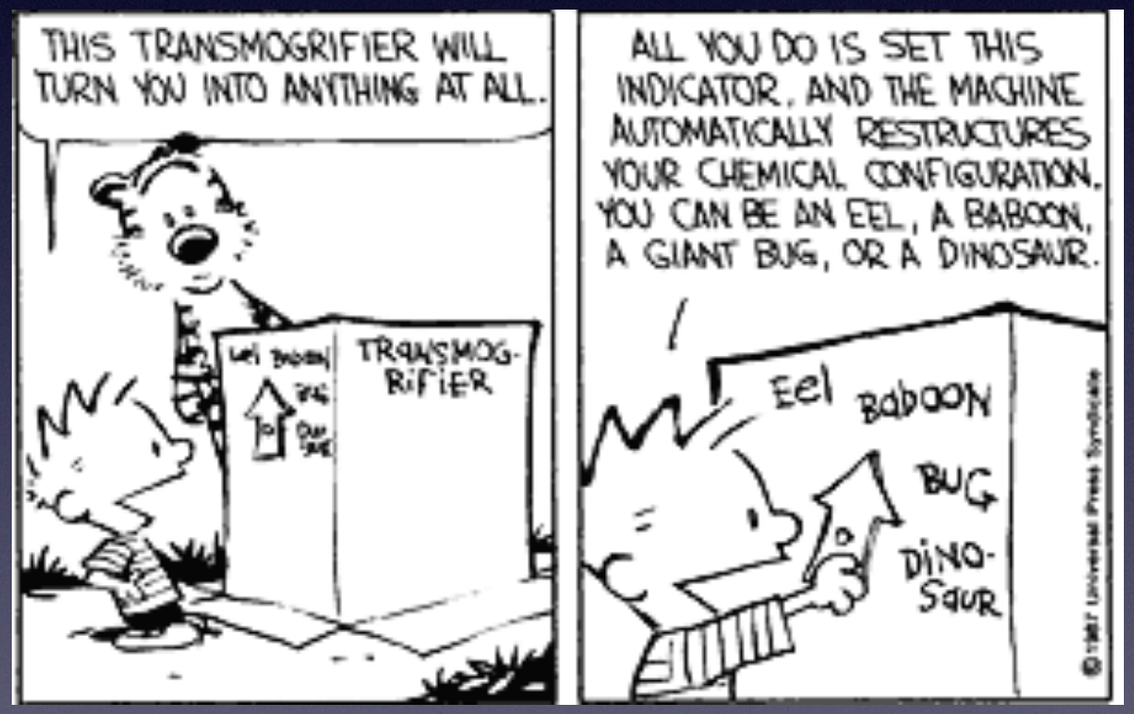
- A test discovery tool searches directories for modules and files which either:
    - have filenames which are identified for testing use
        - (generally by using a “Test” or “test” substring)
    - or files which contain classes and functions which match a substring identifier / regular expression.
- 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.

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

A simple “nose” testing example

file: nose_example1.py




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


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
```

“But I don’t have nose!”

- If using Anaconda Python:
  - `conda install nose`
- If using Enthought Canopy:
  - `enpkg nose`
  - (or use the Canopy package manager GUI)
- Otherwise, try:
  - `pip install nose`

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)`

Finds:

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

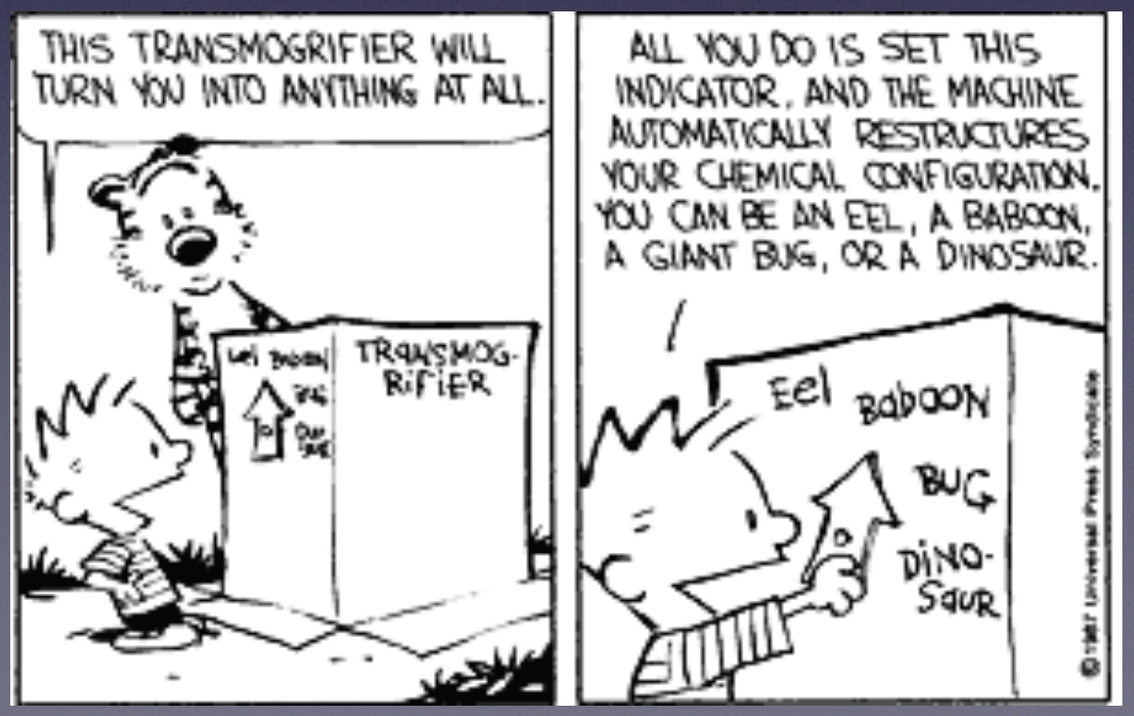
Doesn’t find:

transmogrifyTest()
sometest()

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

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
```

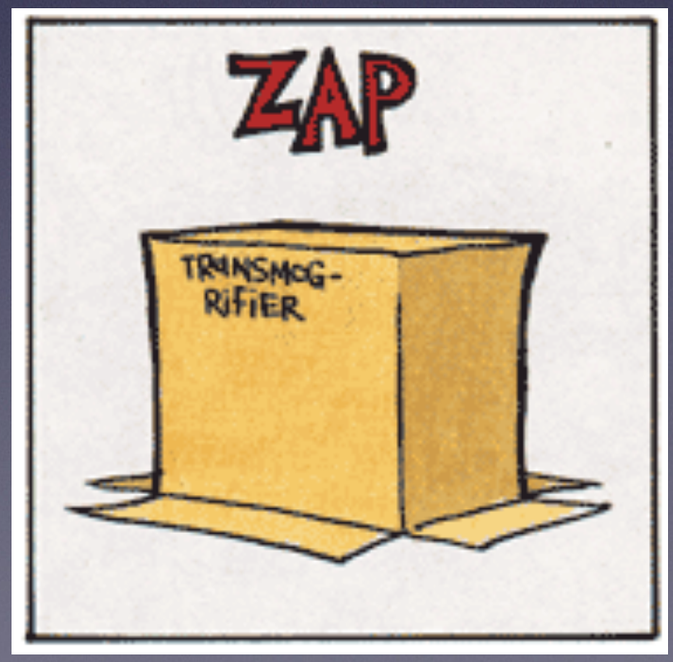


nosetests

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

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
```



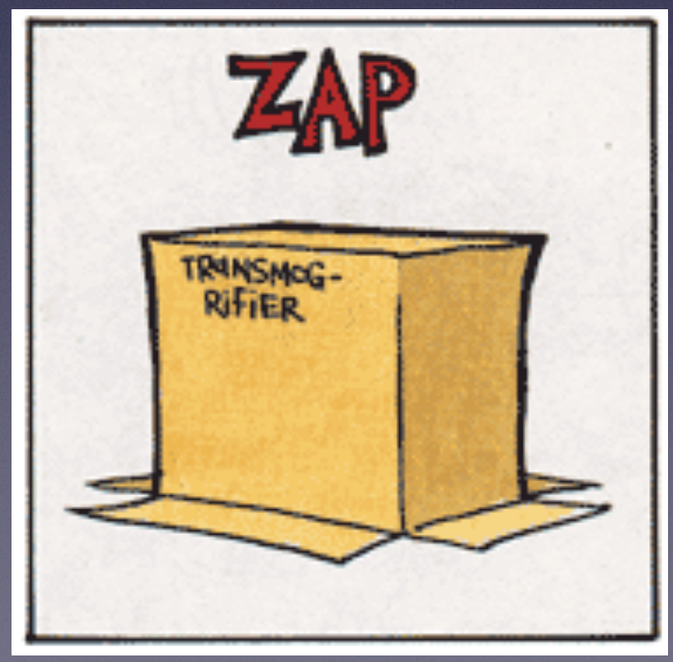
nosetests

```
[jzuhone@gs66-quark:example1] $ nosetests --all-modules
```

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

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
```



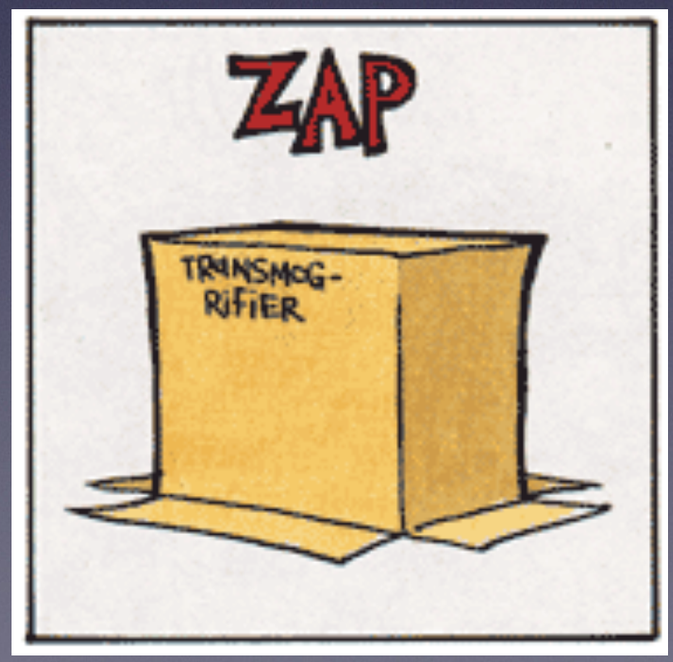
nosetests

```
[jzuhone@gs66-quark:example1] $
```

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

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
```



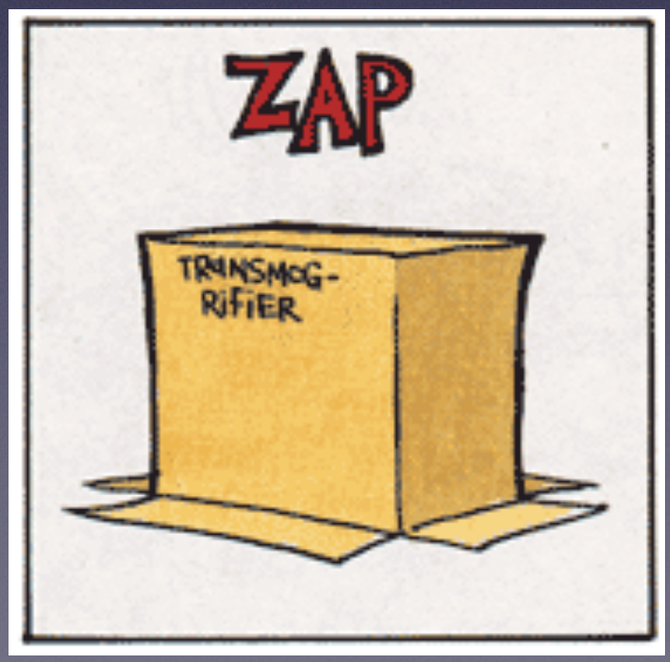
nosetests

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

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
```

```
[jzuhone@gs66-quark:example1] $
```

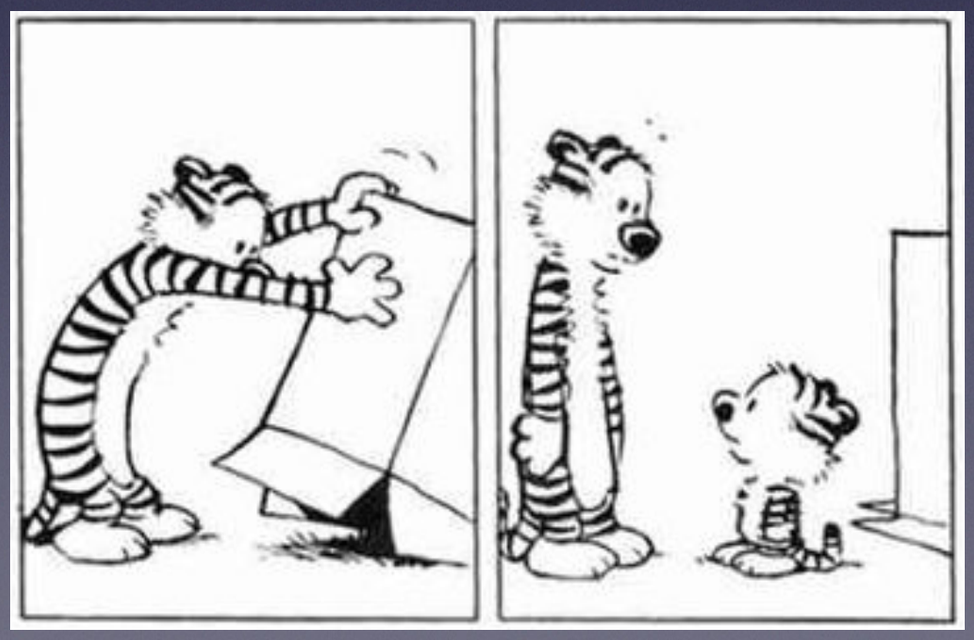


nosetests

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

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) != None
```



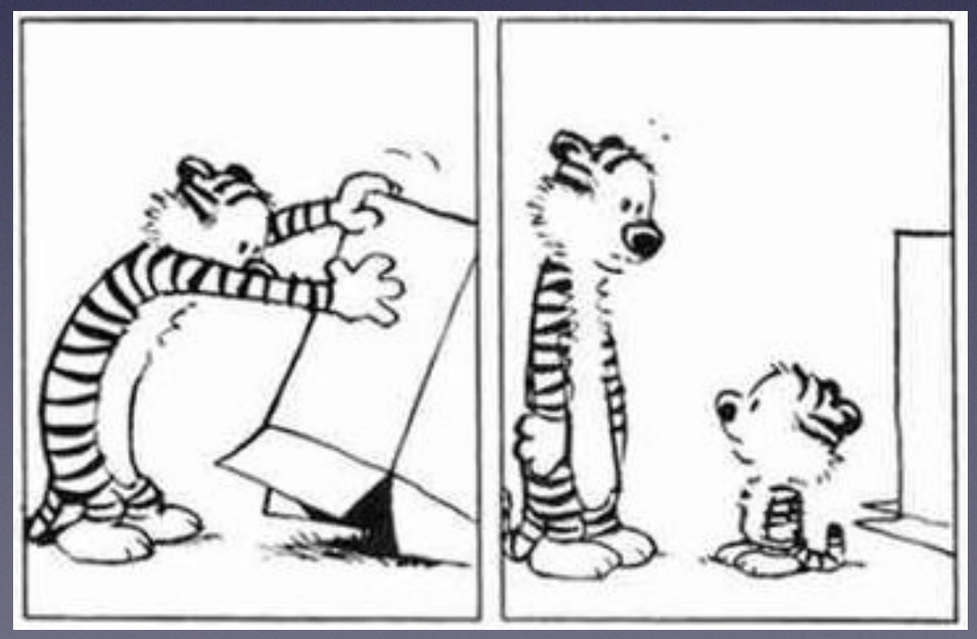
nosetests

Fixed!

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

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) != None
```



nosetests

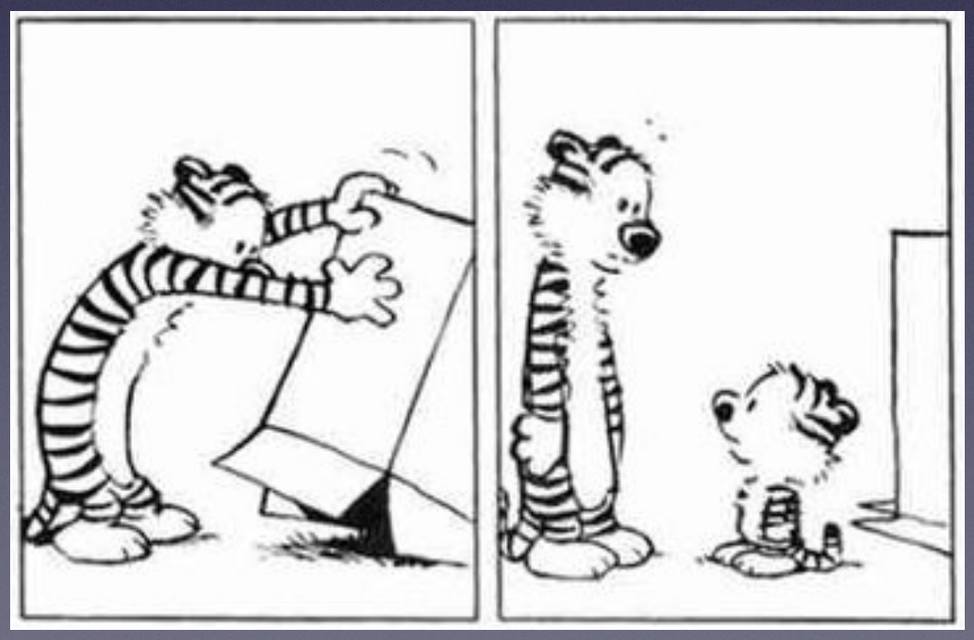
Fixed!

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

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) != None
```

```
[jzuhone@gs66-quark:example1] $ nosetests —all-modules
```



nosetests

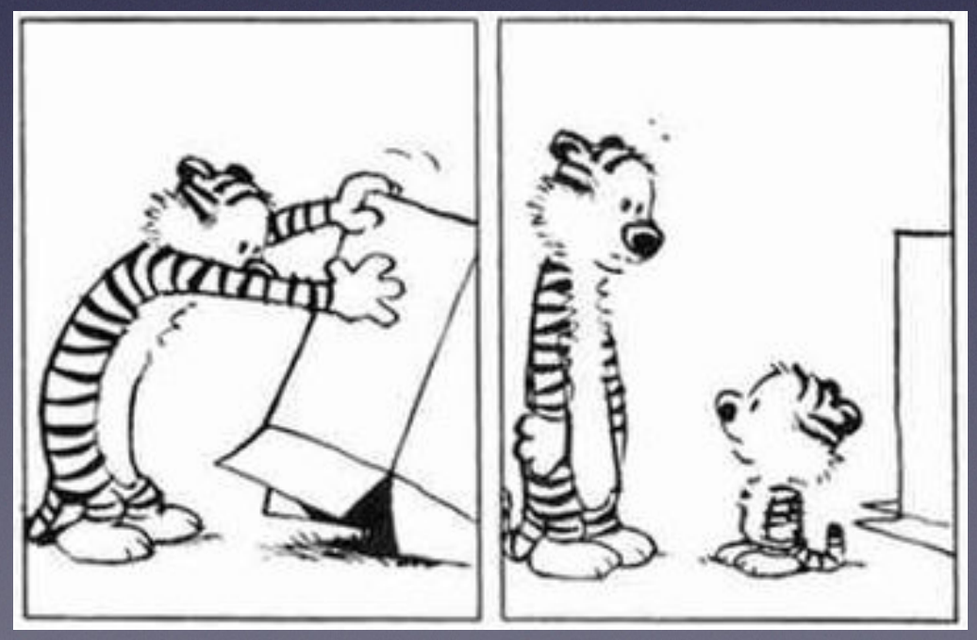
Fixed!

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

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) != None
```

```
[jzuhone@gs66-quark:example1] $
```



doctests

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.

file: doctests_example.py

doctests

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.

```
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
```

file: doctests_example.py

doctests

```
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.

file: doctests_example.py

```
BootCamp> nosetests --with-doctest —doctest-tests
```

doctests

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.

```
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
```

file: doctests_example.py

```
BootCamp> nosetests --with-doctest —doctest-tests

.
----------------------------------------------------------------
```

doctests

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.

```
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
```

file: doctests_example.py

```
BootCamp> nosetests --with-doctest —doctest-tests

.
----------------------------------------------------------------
```

doctests

```
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.

file: doctests_example.py

```
BootCamp> nosetests --with-doctest —doctest-tests

.
----------------------------------------------------------------

Ran 1 test in 0.012s
```

doctests

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.

```
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
```

file: doctests_example.py

```
BootCamp> nosetests --with-doctest —doctest-tests

.
----------------------------------------------------------------

Ran 1 test in 0.012s
OK
```

doctests

```
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.

file: doctests_example.py

```
BootCamp> nosetests --with-doctest —doctest-tests

.
----------------------------------------------------------------

Ran 1 test in 0.012s
OK
BootCamp>
```

doctests

file: doctests_example.py

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
```

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
```

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
```

```
[jzuhone@gs66-quark:12_Testing] $ nosetests --with-doctest --doctest-tests
```

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
```

```
F..
```

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
```

```
======================================================================
```

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
```

```
FAIL: Doctest: doctests_example.multiply
```

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
```

```
----------------------------------------------------------------------
```

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
```

```
Traceback (most recent call last):
```

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
```

```
File "/Users/jzuhone/anaconda/lib/python2.7/doctest.py", line 2226, in runTest
```

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
```

```
raise self.failureException(self.format_failure(new.getvalue()))
```

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
```

```
AssertionError: Failed doctest test for doctests_example.multiply
```

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
```

```
  File "/Users/jzuhone/Source/python-bootcamp/DataFiles_and_Notebooks/12_Testing/
doctests_example.py", line 1, in multiply
```

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
```

```
  File "/Users/jzuhone/Source/python-bootcamp/DataFiles_and_Notebooks/12_Testing/
doctests_example.py", line 1, in multiply
```

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
```

```
----------------------------------------------------------------------
```

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
```

```
File "/Users/jzuhone/Source/python-bootcamp/DataFiles_and_Notebooks/12_Testing/
doctests_example.py", line 7, in doctests_example.multiply
```

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
```

```
Failed example:
```

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
```

```
multiply(-1, 1)
```

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
```

```
Expected:
```

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
```

```
-1
```

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
```

```
Got:
```

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
```

```
0
```

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
```

```
0
```

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
```

```
0
```

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
```

```
----------------------------------------------------------------------
```

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
```

```
Ran 3 tests in 0.009s
```

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
```

```
Ran 3 tests in 0.009s
```

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
```

```
FAILED (failures=1)
```

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
```

```
[jzuhone@gs66-quark:12_Testing] $
```

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

```
BootCamp> nosetests nose_example1.py --with-doctest --doctest-tests -vv
```

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

```
BootCamp> nosetests nose_example1.py --with-doctest --doctest-tests -vv
```

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

```
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
```

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

```
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
```

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

```
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
```

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

```
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
```

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

```
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
```

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

```
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>
```

doctests

Here we combine doctests and the nosetests from the previous example

```
""" 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)
```

file: nose_example1.py

```
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>
```

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...

file: animals_0.py

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

using *nose* testing framework

```
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() == ''
```

file: animals_0.py

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...

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

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...

Requirements:

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

```
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() == ''
```

file: animals_0.py

```
BootCamp> nosetests animals_0.py
```

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...

Requirements:

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

```
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() == ''
```

file: animals_0.py

```
BootCamp> nosetests animals_0.py

EE ======================================================================
```

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...

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_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() == ''
```

```
BootCamp> nosetests animals_0.py

EE ======================================================================
ERROR: animals_0.test_moves
...
```

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...

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_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() == ''
```

```
BootCamp> nosetests animals_0.py

EE ======================================================================
ERROR: animals_0.test_moves
...
    assert Animal('owl').move() == 'fly'
```

Test-Driven Development

using *nose* testing framework

```
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() == ''
```

file: animals_0.py

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...

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
```

Test-Driven Development

using *nose* testing framework

```
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() == ''
```

file: animals_0.py

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...

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
```

Test-Driven Development

using *nose* testing framework

```
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() == ''
```

file: animals_0.py

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...

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

======================================================
```

Test-Driven Development

using *nose* testing framework

```
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() == ''
```

file: animals_0.py

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...

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
...
```

Test-Driven Development

using *nose* testing framework

```
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() == ''
```

file: animals_0.py

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...

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'
```

Test-Driven Development

using *nose* testing framework

```
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() == ''
```

file: animals_0.py

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...

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
```

Test-Driven Development

using *nose* testing framework

```
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() == ''
```

file: animals_0.py

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...

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
```

Test-Driven Development

using *nose* testing framework

```
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() == ''
```

file: animals_0.py

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...

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

------------------------------------------------------
```

Test-Driven Development

using *nose* testing framework

```
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() == ''
```

file: animals_0.py

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...

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
```

Test-Driven Development

using *nose* testing framework

```
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() == ''
```

file: animals_0.py

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...

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
```

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...

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_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() == ''
```

```
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)
```

Test-Driven Development

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

file: animals_1.py

Test-Driven Development

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

```
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']
```

file: animals_1.py

Test-Driven Development

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

```
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']
```

file: animals_1.py

```
BootCamp> nosetests -vv animals_1.py
```

Test-Driven Development

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

```
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']
```

file: animals_1.py

```
BootCamp> nosetests -vv animals_1.py
```

Test-Driven Development

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

```
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']
```

file: animals_1.py

```
BootCamp> nosetests -vv animals_1.py

animals_1.test_moves ... ok
```

Test-Driven Development

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

```
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']
```

file: animals_1.py

```
BootCamp> nosetests -vv animals_1.py

animals_1.test_moves ... ok
animals_1.test_speaks ... ok
```

Test-Driven Development

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

```
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']
```

file: animals_1.py

```
BootCamp> nosetests -vv animals_1.py

animals_1.test_moves ... ok
animals_1.test_speaks ... ok
----------------------------------------------------------------
```

Test-Driven Development

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

```
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']
```

file: animals_1.py

```
BootCamp> nosetests -vv animals_1.py

animals_1.test_moves ... ok
animals_1.test_speaks ... ok
----------------------------------------------------------------
```

Test-Driven Development

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

```
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']
```

file: animals_1.py

```
BootCamp> nosetests -vv animals_1.py

animals_1.test_moves ... ok
animals_1.test_speaks ... ok
----------------------------------------------------------------------

Ran 2 tests in 0.003s
```

Test-Driven Development

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

```
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']
```

file: animals_1.py

```
BootCamp> nosetests -vv animals_1.py

animals_1.test_moves ... ok
animals_1.test_speaks ... ok
----------------------------------------------------------------------

Ran 2 tests in 0.003s
```

Test-Driven Development

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

```
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']
```

file: animals_1.py

```
BootCamp> nosetests -vv animals_1.py

animals_1.test_moves ... ok
animals_1.test_speaks ... ok
----------------------------------------------------------------------

Ran 2 tests in 0.003s

OK
```

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

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

```
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'
```

file: animals_2.py

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

```
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
```

file: animals_2.py

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

```
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
hour-times given. ... ERROR
```

file: animals_2.py

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

```
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
hour-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)
```

file: animals_2.py

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

```
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
hour-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)
```

file: animals_2.py

Test-Driven Development

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

file: animals_3.py

Test-Driven Development

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

```
.......
    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
.......
```

file: animals_3.py

Test-Driven Development

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

```
.......
    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
```

file: animals_3.py

Test-Driven Development

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

```
.......
    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
```

file: animals_3.py

Test-Driven Development

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

```
.......
    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
```

file: animals_3.py

Test-Driven Development

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

```
.......
    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
```

file: animals_3.py

Test-Driven Development

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

```
.......
    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
```

file: animals_3.py

Test-Driven Development

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

```
.......
    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
```

file: animals_3.py

Test-Driven Development

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

```
.......
    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
```

file: animals_3.py

Test-Driven Development

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

```
.......
    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>
```

file: animals_3.py

Test-Driven Development

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

```
.......
    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>
```

file: animals_3.py

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
  - ...

PDB: Basic Commands

Where is my pdb.py?

```
>>> import pdb
>>> help(pdb)
    ...
FILE
    /usr/lib/python2.7/pdb.py
    ...
>>> print pdb.__file__
'/usr/lib/python2.7/pdb.py'
```

PDB: Basic Commands

Where is my pdb.py?

```
>>> import pdb
>>> help(pdb)
    ...
FILE
    /usr/lib/python2.7/pdb.py
    ...
>>> print pdb.__file__
'/usr/lib/python2.7/pdb.py'
```

```
BootCamp> python /usr/lib/python2.5/pdb.py nose_example1.py
```

PDB: Basic Commands

Where is my pdb.py?

```
>>> import pdb
>>> help(pdb)
    ...
FILE
    /usr/lib/python2.7/pdb.py
    ...
>>> print pdb.__file__
'/usr/lib/python2.7/pdb.py'
```

```
BootCamp> python /usr/lib/python2.5/pdb.py nose_example1.py



> /home/training/src/bootdemo/example1/nose_example1.py(2)<module>()
-> """
(Pdb) help
```

PDB: Basic Commands

Where is my pdb.py?

```
>>> import pdb
>>> help(pdb)
    ...
FILE
    /usr/lib/python2.7/pdb.py
    ...
>>> print pdb.__file__
'/usr/lib/python2.7/pdb.py'
```

```
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
```

PDB: Basic Commands

Where is my pdb.py?

```
>>> import pdb
>>> help(pdb)
    ...
FILE
    /usr/lib/python2.7/pdb.py
    ...
>>> print pdb.__file__
'/usr/lib/python2.7/pdb.py'
```

```
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
```

PDB: Basic Commands

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: 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 Transmogrifier:
  5         """ An important class
  6         """
  7         def transmogrify(self, person):
  8             """ Transmogrify someone
  9             """
  10            transmog = {'calvin':'tiger',
  11                        'hobbes':'chicken'}
(Pdb)
```

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 Transmogrifier:
  5         """ An important class
  6         """
  7         def transmogrify(self, person):
  8             """ Transmogrify someone
  9             """
  10            transmog = {'calvin':'tiger',
  11                        'hobbes':'chicken'}
(Pdb)
```

PDB: Basic Commands

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: 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
```

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


calvin ->  ZAP!  -> tiger
```

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



Traceback (most recent call last):
```

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
```

```
File "/Users/jzuhone/anaconda/lib/python2.7/pdb.py", line 1314, in main
```

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._runscript(mainpyfile)
```

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
```

```
File "/Users/jzuhone/anaconda/lib/python2.7/pdb.py", line 1233, in _runscript
```

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
```

```
self.run(statement)
```

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
```

```
File "/Users/jzuhone/anaconda/lib/python2.7/bdb.py", line 400, in run
```

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
```

```
exec cmd in globals, locals
```

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
```

```
File "<string>", line 1, in <module>
```

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
```

```
File "nose_example1.py", line 2, in <module>
```

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: 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
```

```
File "nose_example1.py", line 25, in main
```

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
```

```
print p, '->  ZAP!  ->', TM.transmogrify(p)
```

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
```

```
File "nose_example1.py", line 12, in transmogrify
```

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
```

```
new_person = transmog[person]
```

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
```

```
KeyError: 'Hobbes'
```

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
```

```
Hobbes ->  ZAP!  -> Uncaught exception. Entering post mortem debugging
```

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



Running 'cont' or 'step' will restart the program
```

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



> /Users/jzuhone/Source/python-bootcamp/DataFiles_and_Notebooks/12_Testing/example1/
nose_example1.py(12)transmogrify()
```

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
```

```
-> new_person = transmog[person]
(Pdb)
```

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
```

```
-> new_person = transmog[person]
(Pdb)
```

PDB: Basic Commands

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: 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
```

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
```

```
7       def transmogrify(self, person):
```

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
```

```
8           """ Transmogrify someone
```

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
```

```
9           """
```

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


10          transmog = {'calvin':'tiger',
```

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
```

```
11                      'hobbes':'chicken'}
```

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
```

```
12  ->          new_person = transmog[person]
```

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
```

```
13          return new_person
```

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: 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: 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
```

```
16  def test_transmogrify():
```

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
```

```
17        TM = Transmogrifier()
```

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) print person
```

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
```

Hobbes

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) print transmog.keys()
```

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
```

```
['calvin', 'hobbes']
```

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)
```

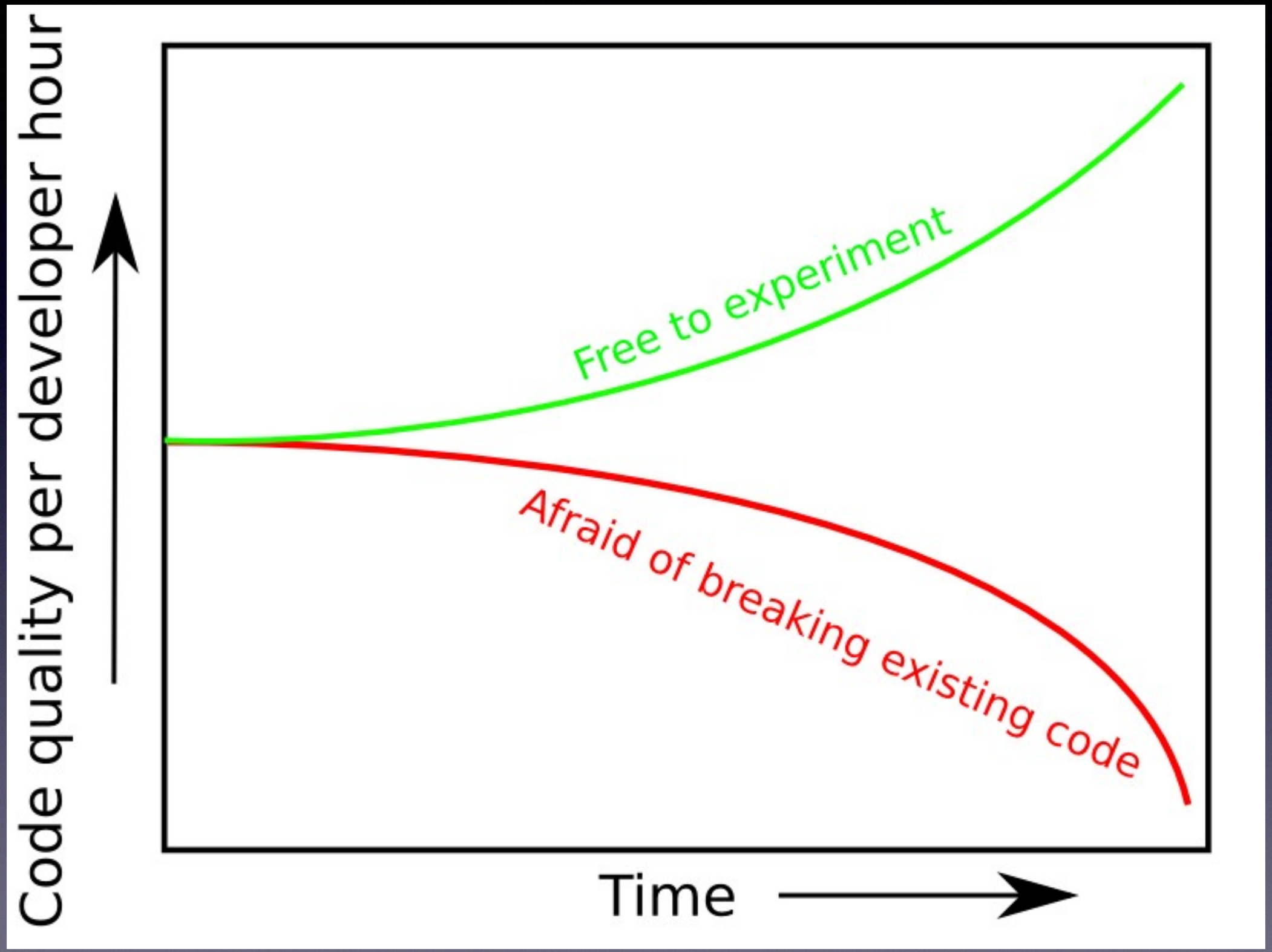
PDB: Basic Commands

```
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)
```

nosetests --all-modules --pdb

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


Code quality per developer hour
Free to experiment
Afraid of breaking existing code
Time