# CS 683 Emerging Technologies
## Fall Semester, 2005
### Doc 4 Exceptions, Modules and Unit Tests

#### Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lambda Environments</td>
<td>5</td>
</tr>
<tr>
<td>Exceptions</td>
<td>6</td>
</tr>
<tr>
<td>Catching an Exception</td>
<td>6</td>
</tr>
<tr>
<td>Defining an Exception</td>
<td>7</td>
</tr>
<tr>
<td>Modules</td>
<td>8</td>
</tr>
<tr>
<td>How Python finds Modules</td>
<td>9</td>
</tr>
<tr>
<td>Import statement</td>
<td>11</td>
</tr>
<tr>
<td>Modules and Executable code</td>
<td>12</td>
</tr>
<tr>
<td>Packages</td>
<td>16</td>
</tr>
<tr>
<td>XUnit tests</td>
<td>18</td>
</tr>
<tr>
<td>Why Unit Testing</td>
<td>25</td>
</tr>
<tr>
<td>JUnit</td>
<td>32</td>
</tr>
</tbody>
</table>

Copyright ©, All rights reserved. 2005 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.
References

Python Tutorial, Guido van Rossum,
http://www.python.org/doc/current/tut/tut.html

Python Reference Manual, Guido van Rossum,
http://docs.python.org/ref/ref.html

Python Library Reference, Guido van Rossum,
http://docs.python.org/lib/lib.html

Learning Python, Lutz & Ascher, O'Reilly, 1999

Brian Marick’s Testing Web Site:
http://www.testing.com/

Testing for Programmers, Brian Marick, Available at:
http://www.testing.com/writings.html
### Just for Fun

**Reading a Webpage**

```python
import urllib2

webPage = urllib2.urlopen('http://www.eli.sdsu.edu')

print(webPage.info())  # print http headers

for line in webPage:
    print(line),
```

### Sending Mail

```python
import smtplib

mailServer = smtplib.SMTP('cs.sdsu.edu')
sender = 'whitney@cs.sdsu.edu'
receiver = 'whitney@rohan.sdsu.edu'
message = """"To: whitney@rohan.sdsu.edu
From: whitney@cs.sdsu.edu

This is a sample email.
"""
mailServer.sendmail(sender, receiver, message)
mailServer.quit()
```
import re, urllib2

def hrefsIn(aString):
    return aString.split(' </a>')</def

def linkIn(aHref):
    pattern = re.compile('href="([^"]*)_+', re.IGNORECASE)
    result = pattern.search(aHref)
    if result:
        return result.groups()[0]
    else:
        return ""
def linksIn(aString):
    linksAndEmpty = map(linkIn, hrefsIn(aString))
    return filter(lambda x: x != '', linksAndEmpty)

def spider(aUrl):
    webPage = urllib2.urlopen(aUrl)
    links = []
    for line in webPage:
        links = links + linksIn(line)
    return links

print spider('http://www.eli.sdsu.edu/')
Lambda Environments

x = 5

test = lambda y: x + y
print test(1)  # prints 6

x = 20
print test(1)  # prints 21

def local(function):
    x = 10
    print function(1)

local(test)  # prints 21

def localEnvironment():
    z = 10
    return lambda y: y + z

z = 2
function = localEnvironment();
print function(1)  # prints 11
Exceptions
Catching an Exception

General forms

```python
try:
    <block>
except <name>:
    <except block>
except <name>, <data>:
    <except block2>
else:
    <else block>
```

```python
try:
    <block>
finally:
    <finally block>
```

```python
def myBad(list, index):
    print list[index]

try:
    myBad([0, 1], 3)
except IndexError:
    print 'BadIndex'
```

Built-in Exceptions
Defining an Exception

class SampleError(Exception):
    pass

try:
    raise SampleError
except SampleError:
    print 'I got it'
else:
    print 'No Error'
**Modules**

**Simple Example**

In a file called eli.py place:

```python
def whitney():
    print 'whitney'
```

This is now a module which can be used by python code

---

**Using the Module**

```python
import eli
eli.whitney()
```
How Python finds Modules

Python interpreter searches in order:

- Current directory
- Directories listed in environment variable PYTHONPATH
- Installation-dependent path for standard modules

So create a directory for your modules and set your PYTHONPATH to include that directory.

Unix Example

Setting PYTHONPATH

```
setenv PYTHONPATH /Users/whitney/Courses/683/Fall05/pythonCode/modules
```

Now place the file eli.py in the directory

```
/Users/whitney/Courses/683/Fall05/pythonCode/modules
```

and from any location on the machine run:

```
import eli
whitney()
```
**sys.path and PYTHONPATH**
sys.path contains the PYTHONPATH

sys.path can be accessed and changed at runtime

```
import sys
print sys.path

sys.path.append('/Users/whitney/Courses/683/Fall 05/pythonCode/modules')
print sys.path

import eli
eli.whitney()
```
Import statement

Simple import

Need to use full name of module elements

```python
import eli
eli.whitney()
```

Import with names

Can use short name of imported element

```python
from eli import whitney
whitney()
```

Import all names

* imports all names from the module except those starting with _

```python
from eli import *
whitney()
```
**Modules and Executable code**

**ely.py**

```python
x = 0
y = [1, 2]
print 'eli module run'

def whitney():
    print 'whitney'

def printValues():
    print x, y
```

When loaded for the first time code in the module is run

**Example**

```python
import eli
eli.whitney()
from eli import y
print y
```

**Output**

```
eli module run
whitney
[1, 2]
```
Reloading a module

if you need to reload a module use reload()

Example

```python
import eli
eli.y[0] = 'cat'
eli.printValues()

reload(eli)
eli.printValues()
```

Output

```
eli module run 0 ['cat', 2]
eli module run 0 [1, 2]
```
**Explain this One!**

```python
# ely.py
x = 0
y = [1, 2]
print 'eli module run'

def printValues():
    print x, y
```

**Program**

```python
from eli import x, y, printValues

printValues()
y[0] = 'cat'
x = 'dog'
printValues()
```

**Output**

```
0 [1, 2]
0 ['cat', 2]
```
dir()

List all names defined in a module

import sys, eli

print dir(eli)
pres dir(sys)

Output

['__builtins__', '__doc__', '__file__', '__name__', 'printValues', 'x', 'y']

['__displayhook__', '__doc__', '__excepthook__', '__name__', '__stderr__', '__stdin__', '__stdout__', '_getframe', 'api_version', 'argv', 'builtin_module_names', 'byteorder', 'call_tracing', 'callstats', 'copyright', 'some lines removed here', 'setdlopenflags', 'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout', 'version', 'version_info', 'warnoptions']
Packages
Hierarchal modules
foo.bar.whitney()

Example
In a directory in PYTHONPATH create a subdirectory - foo
In foo place an __init__.py file (it can be empty)
Any .py file in foo is now an accessible module
The file test.py is now foo.test

Can nest directories
Program and imported Module

__name__

- Module attribute
- Set to '__main__' if file is run as a program
- Set to module name if file is imported as a module

Example

```python
def hello():
    print 'Hello'

if __name__ == '__main__':
    hello()
```

As Program

Al 60->python nameExample.py
Hello

As Module

import nameExample
nameExample.hello()

Output

Hello
XUnit tests

Example

Queue code to test

class Queue:
    def __init__(self):
        self._elements = []

    def size(self):
        return len(self._elements)

    def enqueue(self, a):
        self._elements.append(a)

    def dequeue(self):
        first = self._elements[0]
        self._elements = self._elements[1:]
        return first
import unittest

class TestQueue(unittest.TestCase):

    def testEnqueue(self):
        queue = Queue()
        self.assertEqual(queue.size(), 0)
        queue.enqueue(1)
        self.assertEqual(queue.size(), 1)

    def testDequeue(self):
        queue = Queue()
        queue.enqueue(1)
        queue.enqueue(2)
        queue.enqueue(3)
        self.assertEqual(queue.size(), 3)
        self.assertEqual(queue.dequeue(), 1)
        self.assertEqual(queue.size(), 2)
        self.assertEqual(queue.dequeue(), 2)
        self.assertEqual(queue.size(), 1)
        self.assertEqual(queue.dequeue(), 3)
        self.assertEqual(queue.size(), 0)

if __name__ == '__main__':
    unittest.main()

main runs all methods starting with 'test'
Result of Running the tests

.. 
-------------------------------------
Ran 2 tests in 0.000s

OK
What happens if tests fail?

```python
import unittest

class TestQueue(unittest.TestCase):

    def testEnqueue(self):
        queue = Queue()
        self.assertEqual(queue.size(), 42)
        queue.enqueue(1)
        self.assertEqual(queue.size(), 1)

if __name__ == '__main__':
    unittest.main()
```

Output

```
F
================================
FAIL: testEnqueue (__main__.TestQueue)
---------------------------------------------------------
Ran 1 test in 0.004s
FAILED (failures=1)
```

Plus debugger opens on error
**setUp & tearDown**

setUp() is run before each test method

tearDown() is run after each test method

```python
import unittest

class TestQueue(unittest.TestCase):
    def setUp(self):
        self.queue = Queue()

    def testEnqueue(self):
        self.assertEqual(self.queue.size(), 0)
        self.queue.enqueue(1)
        self.assertEqual(self.queue.size(), 1)

    def testDequeue(self):
        self.queue.enqueue(1)
        self.queue.enqueue(2)
        self.assertEqual(self.queue.size(), 2)
        self.assertEqual(self.queue.dequeue(), 1)
        self.assertEqual(self.queue.size(), 1)
        self.assertEqual(self.queue.dequeue(), 2)
        self.assertEqual(self.queue.size(), 0)

if __name__ == '__main__':
    unittest.main()
```
Some Useful TestCase methods

• assert_(expression)
  • failUnless(expression)

• assertEqual(first, second[, msg])
  • failUnlessEqual(first, second[, msg])

• assertNotEqual(first, second[, msg])
  • failIfEqual(first, second[, msg])

• assertAlmostEqual(first, second[, places[, msg]])
  • failUnlessAlmostEqual(first, second[, places[, msg]])

• assertNotAlmostEqual(first, second[, places[, msg]])
  • failIfAlmostEqual(first, second[, places[, msg]])

• assertRaises(exception, callable, ...)
  • failUnlessRaises(exception, callable, ...)

See on-line docs for details
import unittest

class TestQueue(unittest.TestCase):
    def setUp(self):
        self.queue = Queue()

    def testEnqueue(self):
        self.assertEqual(self.queue.size(), 0)
        self.queue.enqueue(1)
        self.assertEqual(self.queue.size(), 1)

    def testEmptyDequeue(self):
        self.assertRaises(IndexError, self.queue.dequeue)

if __name__ == '__main__':
    unittest.main()
Why Unit Testing

If it is not tested it does not work

The more time between coding and testing

• More effort is needed to write tests
• More effort is needed to find bugs
• Fewer bugs are found
• Time is wasted working with buggy code
• Development time increases
• Quality decreases

Without unit tests

• Code integration is a nightmare
• Changing code is a nightmare
Why Automated Tests?

What is wrong with:

• Using print statements
• Writing driver program in main
• Writing small sample programs to run code
• Running program and testing it be using it
Repeatability & Scalability

Need testing methods that:

• Work with N programmers working for K months (years)
• Help when modify code 6 months after it was written
• Check impact of code changes in rest of system

Practices that work in a school project may not be usable in industry

Standard industry practices may seem overkill in a school project

Work on building good habits and skills
We have a QA Team, so why should I write tests?

How long does it take QA to test your code?

How much time does your team spend working around bugs before QA tests?

How easy is it to find & correct the errors after QA finds them?

Most programmers have an informal testing process

With a little more work you can develop a useful test suite
When to Write Unit Tests

First write the tests
Then write the code to be tested

Writing tests first saves time

- Makes you clear of the interface & functionality of the code
- Removes temptation to skip tests
What to Test

Everything that could possibly break

Test values

• Inside valid range
• Outside valid range
• On the boundary between valid/invalid

GUIs are very hard to test

• Keep GUI layer very thin
• Unit test program behind the GUI, not the GUI
Common Things Handled Incorrectly

Adapted with permission from “A Short Catalog of Test Ideas” by Brian Marick, http://www.testing.com/writings.html

Strings
Test using empty String

Collections
Test using:

- Empty Collection
- Collection with one element
- Collection with duplicate elements
- Collections with maximum possible size

Numbers
Test using:

- Zero
- The smallest number
- Just below the smallest number
- The largest number
- Just above the largest number
JUnit

Free frameworks for Unit testing

JUnit written by Kent Beck & Erich Gamma

Available at: http://www.junit.org/

Ports are available in at least

<table>
<thead>
<tr>
<th>.NET</th>
<th>Ada</th>
<th>AppleScript</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>C#</td>
<td>C++</td>
<td>Curl</td>
<td>Delphi</td>
</tr>
<tr>
<td>Eiffel</td>
<td>Eiffel</td>
<td>Flash</td>
<td>Forte 4GL</td>
</tr>
<tr>
<td>Gemstone/S</td>
<td>Haskell</td>
<td>HTML</td>
<td>Jade</td>
</tr>
<tr>
<td>LISP</td>
<td>Objective-C</td>
<td>Oracle</td>
<td>Palm</td>
</tr>
<tr>
<td>Perl</td>
<td>Php</td>
<td>PowerBuilder</td>
<td>Python</td>
</tr>
<tr>
<td>Ruby</td>
<td>Scheme</td>
<td>Smalltalk</td>
<td>Visual Basic</td>
</tr>
<tr>
<td>XML</td>
<td>XSLT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See http://www.xprogramming.com/software.htm to download ports