PYTHON LAB BOOK

Python For Programmers

UCSC Extension Online

Lab 18 Wrap Up

Topics

- Exceptions
- Namespaces
- Nests
- Pitfalls
- Finding Modules and Help

v.4.0

lab17_1_1.py

```
#!/usr/bin/env python
"""lab17_1_1.py Unit test for clock_def.py"""
import unittest
import sys
import time
sys.path.insert(0, '..')
import lab_15_Overriding.lab15 as clock_def

class TestClock(unittest.TestCase):

    test_values = [(hours, minutes) for hours in range(-10, 25) \
                   for minutes in range(-122, 122)]

    def testInitEqual(self):
        now = time.ctime()[11:16]
        self.assertEqual(clock_def.Clock(now), clock_def.Clock())
        self.assertRaises(ValueError, clock_def.Clock, (1, 2, 3))
        for (hours, minutes) in TestClock.test_values:
            clocks = [clock_def.Clock(hours, minutes)]
            clocks += [clock_def.Clock((hours, minutes))]
            clocks += [clock_def.Clock("%d:%02d" % (hours, minutes))]
            clocks += [clock_def.Clock({'hr': hours, 'min': minutes})]
            clocks += [clock_def.Clock(hr=hours, min=minutes)]
            clock_str = str(clocks[0])
            clock_repr = repr(clocks[0])
            clock_mins = clocks[0].MinutesSince12()
            for each in clocks[1:]:
                self.assertTrue(each.min<60 and each.min>=0)
                self.assertTrue(each.hr>=1 and each.hr<13)
                self.assertEqual(str(each), clock_str)
                self.assertEqual(repr(each), clock_repr)
                self.assertEqual(clocks[0], eval("clock_def." + repr(each)))
                self.assertEqual(each.MinutesSince12(), clock_mins)
                self.assertEqual(each, clocks[0])

    def testAddSub(self):
        self.c1 = clock_def.Clock(12, 59)
        for (hours, minutes) in TestClock.test_values:
            c2 = clock_def.Clock(hours, minutes)
            c_sum = self.c1 + c2
            c_diff = self.c1 - c2
            c3 = c_sum + c_diff # should be 2 * self.c1
            c4 = clock_def.Clock(self.c1.hr * 2, self.c1.min * 2)
            self.assertEqual(c3, c4)
```

```

class ClockTestSuite(unittest.TestSuite):
    """Ignore this class for now, it is for aggregating tests into a
    test suite."""

    def __init__(self):
        unittest.TestSuite.__init__(self, map(
            TestClock, ("testInitEqual", "testAddSub")))

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

"""
$ lab17_1_1.py
..
----------------------------------------------------------------------
Ran 2 tests in 16.185s

OK
$
"""
```

lab17_1_2.py

```
#!/usr/bin/env python
"""lab17_1_2.py Test for the Money class"""
import unittest
import sys
sys.path.insert(0, '..')
import lab_16_New_Style_Classes.lab16_2 as money_def

class TestMoney(unittest.TestCase):

    def testFormat(self):
        self.assertEqual(str(money_def.Money(-123.21)), "-$123.21")
        self.assertEqual(str(money_def.Money(40.50)), "$40.50")
        self.assertEqual(str(money_def.Money(-1001.011)), "-$1,001.01")
        self.assertEqual(str(money_def.Money(123456789.999)),
                         "$123,456,790.00")
        self.assertEqual(str(money_def.Money(.10)), "$0.10")
        self.assertEqual(str(money_def.Money(.01)), "$0.01")
        self.assertEqual(str(money_def.Money(.055)), "$0.06")

    def testAdd(self):
        self.assertAlmostEqual(money_def.Money(10) + money_def.Money(20),
                               money_def.Money(30))
    def testRepr(self):
        self.assertAlmostEqual(eval(
            'money_def.' + repr(money_def.Money(44.123))),
                               money_def.Money(44.123))

    def testSub(self):
        self.assertAlmostEqual(money_def.Money('-11.111000'),
                               money_def.Money('-11.111000'))
        self.assertAlmostEqual(
            money_def.Money(44.333) - money_def.Money(55.444),
            money_def.Money(-11.111))

    def testNeg(self):
        self.assertAlmostEqual(-money_def.Money(10.00),
                               money_def.Money(-10.00))
    def testMult(self):
        self.assertAlmostEqual(2 * money_def.Money(-11.11),
                               money_def.Money(-22.22))
        self.assertAlmostEqual(money_def.Money(-22.22),
                               money_def.Money(11.11) * -2)
    def testDiv(self):
        self.assertAlmostEqual((money_def.Money(44.44))/4,
                                money_def.Money(11.11))

if __name__ == '__main__':
    unittest.main()
"""$ lab17_1_2.py
.......
----------------------------------------------------------------------
Ran 7 tests in 0.003s

OK
$ """
```

testsuite.py

```
#!/usr/bin/env python
"""Demonstration of a test suite."""
import unittest
import lab17_1_1 as clock_test_def
import lab17_1_2 as money_test_def

Clock_suite = clock_test_def.ClockTestSuite()
Money_suite = unittest.makeSuite(money_test_def.TestMoney, 'test')

all_test_suites = unittest.TestSuite((Clock_suite, Money_suite))

unittest.TextTestRunner().run(all_test_suites)

"""
$ testsuite.py
.........
----------------------------------------------------------------------
Ran 9 tests in 10.215s

OK

$
"""

```

lab17_2.py
```
#!/usr/bin/env python
"""lab17_2.py  -- deals card hands:
lab17_2.py  -- deals 4 hands of 5 cards
lab17_2.py -p 6 -c 3  -- deals 6 hands of 3 cards
"""
import sys
import random

sys.path.insert(0, '..')
import lab_08_Comprehensions.lab08_2 as cards

class Deck:
    """An iteratiing deck of cards that destroys each card as it is
    taken with a next call - or as it is iterated with a for-loop."""

    def __init__(self):
        self._cards = cards.Cards()
        random.shuffle(self._cards)

    def __iter__(self):
        return self

    def next(self):
        try:
            return self._cards.pop()
        except IndexError:
            raise StopIteration

class GameDealer:
    def __init__(self, no_players=4, no_cards=5):
        self.no_players = int(no_players)
        self.no_cards = int(no_cards)
        self.hands = []
        self.deck = Deck()

    def DealAll(self):
        return [c for c in self.deck]

    def DealCard(self):
        try:
            return self.deck.next()
        except StopIteration:
            return "Blank"

    def DealHand(self):
        """Add a hand to the list of hands."""
        self.hands += [[self.DealCard() for i in range(self.no_cards)]]
        return self.hands[-1]

    def DealGame(self):
        """Make a list of hands, each hand is also a list."""
        return [self.DealHand() for i in range(self.no_players)]

    def __str__(self):
        """Returns a string representation of the dealt cards."""
        if not self.hands:
            self.DealGame()
        return  '\n'.join([', '.join(c for c in h) for h in self.hands])

def main():
    import optparse
    parser = optparse.OptionParser(\
        "%prog [-p number_of_players=4] [-c number_of_cards=5]")
    parser.add_option("-p", "--players", dest="no_players",
                      help="number of players", default=4)
    parser.add_option("-c", "--cards", dest="no_cards",
                      help="number of cards per hand", default=5)
    (options, args) = parser.parse_args()
    if len(args) > 0:
        parser.error("I don't recognize %s", (' '.join(args)))
    print GameDealer(options.no_players, options.no_cards)

    # To use the generator-based solution::
    # import lab_13_Function_Fancies.lab13_3 as dealer
    # dealer.PrintGame(dealer.DealGame(options.no_players, options.no_cards))

if __name__ == '__main__':
    main()
"""
$ lab17_2.py
Joker, 6 of Diamonds, Ace of Hearts, 4 of Clubs, 2 of Clubs
9 of Clubs, King of Spades, 4 of Hearts, King of Diamonds, 7 of Clubs
10 of Hearts, 5 of Diamonds, Queen of Diamonds, 2 of Hearts, 3 of Diamonds
8 of Hearts, 4 of Diamonds, 9 of Diamonds, 10 of Clubs, 3 of Hearts
$ lab17_2.py -x
Usage: lab17_2.py [-p number_of_players=4] [-c number_of_cards=5]

lab17_2.py: error: no such option: -x
$ lab17_2.py -help
```

```
Usage: lab17_2.py [-p number_of_players=4] [-c number_of_cards=5]

Options:
  -h, --help            show this help message and exit
  -p NO_PLAYERS, --players=NO_PLAYERS
                        number of players
  -c NO_CARDS, --cards=NO_CARDS
                        number of cards per hand
$ lab17_2.py -p 6 -c 3
Jack of Spades, 10 of Diamonds, Ace of Clubs
9 of Clubs, 4 of Spades, Joker
9 of Diamonds, Jack of Diamonds, 10 of Spades
9 of Hearts, 7 of Spades, 3 of Diamonds
7 of Hearts, 7 of Diamonds, King of Diamonds
Ace of Hearts, 10 of Hearts, 8 of Hearts
$ lab17_2.py -p 11
9 of Spades, King of Clubs, 5 of Spades, 6 of Hearts, Queen of Clubs
10 of Spades, 2 of Hearts, 9 of Diamonds, 3 of Clubs, Jack of Hearts
10 of Clubs, 6 of Clubs, Queen of Diamonds, 3 of Hearts, Jack of Spades
5 of Hearts, King of Spades, King of Hearts, Jack of Clubs, 10 of Hearts
8 of Hearts, Ace of Hearts, 8 of Spades, 7 of Spades, 9 of Clubs
Queen of Hearts, 5 of Diamonds, Joker, 7 of Diamonds, 8 of Diamonds
Ace of Spades, 5 of Clubs, 2 of Diamonds, 4 of Clubs, 4 of Spades
Jack of Diamonds, 2 of Clubs, 10 of Diamonds, 6 of Diamonds, 9 of Hearts
Ace of Clubs, 8 of Clubs, Joker, 7 of Clubs, 4 of Hearts
Ace of Diamonds, 3 of Diamonds, 6 of Spades, 2 of Spades, 7 of Hearts
4 of Diamonds, King of Diamonds, 3 of Spades, Queen of Spades, None
$ """
```

lab17_3.py

```
#!/usr/bin/env python

"""lab17_3.py Make a TimeOut context handler so that this code works.
"""

import signal, time

class TimeOut:

    def __init__(self, timeout):
        self.timeout = timeout

    def __enter__(self):
        def AlarmHandler(signum, frame):
            raise RuntimeError, \
                  "Timed out after %s seconds." % (self.timeout)
        self.old = signal.signal(signal.SIGALRM, AlarmHandler)
        signal.alarm(self.timeout)
        return True

    def __exit__(self, exc_type, exc_val, exc_tb):
        signal.signal(signal.SIGALRM, self.old)
        signal.alarm(0)

with TimeOut(2) as ticker:
    try:
       time.sleep(5)
    except RuntimeError, msg:
       print "Sleeping 5 timed out!", msg

with TimeOut(5) as ticker:
    try:
       time.sleep(2)
       print "Sleeping 2 didn't time out."
    except RuntimeError:
       print "Timed out!"

"""
$ lab17_3.py
Sleeping 5 timed out! Timed out after 2 seconds.
Sleeping 2 didn't time out.
$
"""
```

assert_.py

```
#!/usr/bin/env python
"""The "assert" statement is useful while debugging.  It goes away
under any optimization."""

def main():
    x = input("Give me positive x please: ")
    assert x > 0
    print "Good. %s is positive." % x

if __name__ == '__main__':
    main()

"""
$ assert_.py
Give me positive x please: 3.14
Good. 3.14 is positive.
$ assert_.py
Give me positive x please: 0
Traceback (most recent call last):
  File "./assert_.py", line 11, in <module>
    main()
  File "./assert_.py", line 7, in main
    assert x > 0
AssertionError
$
"""
```

Exceptions

```
>>> help('exceptions')
```

gives you lots of info about exceptions. For example, exceptions are classes, in a hierarchy:

```
Exception
 |
 +-- SystemExit
 +-- StopIteration
 +-- StandardError
 |    +-- KeyboardInterrupt
 |    +-- ImportError
 |    +-- EnvironmentError
 |    |    +-- IOError
 |    |    +-- OSError
 |    |         +-- WindowsError
 |    +-- EOFError
 |    +-- RuntimeError
 |    |    +-- NotImplementedError
 |    +-- NameError
 |    |    +-- UnboundLocalError
 |    +-- AttributeError
 |    +-- SyntaxError
 |    |    +-- IndentationError
 |    |         +-- TabError
 |    +-- TypeError
 |    +-- AssertionError
 |    +-- LookupError
 |    |    +-- IndexError
 |    |    +-- KeyError
 |    +-- ArithmeticError
 |    |    +-- OverflowError
 |    |    +-- ZeroDivisionError
 |    |    +-- FloatingPointError
 |    +-- ValueError
 |    |    +-- UnicodeError
 |    +-- ReferenceError
 |    +-- SystemError
 |    +-- MemoryError
 +---Warning
      +-- UserWarning
      +-- DeprecationWarning
      +-- SyntaxWarning
```

```
                +-- OverflowWarning
                +-- RuntimeWarning
```

This:

```
try:
   something
except ArithmeticError:
   pass
```

catches all 3 arithmetic errors: `OverflowError`, `ZeroDivisionError`, and `FloatingPointError`.

The `help('exceptions')` also shows this about each Exception class and subclass:

```
class ArithmeticError(StandardError)
 |  Base class for arithmetic errors.
 |
 |  Method resolution order:
 |      ArithmeticError
 |      StandardError
 |      Exception
 |
 |  Methods inherited from Exception:
 |
 |  __getitem__(...)
 |
 |  __init__(...)
 |
 |  __str__(...)
```

When you collect an exception:

```
try:
    something
except ArithmeticError, msg:
    print msg
```

The `msg` is actually an instance of the `ArithmeticError` class, and:

```
print msg
```

calls `str(msg)`, as it always does, and the `Exception` class's `__str__` gets called.

So, you cannot:

```
print "This happened: " + msg
```

but you can:

```
print "This happened: ", msg
```

or

```
print "This happened: " + str(msg)
```

The syntax for catching multiple exceptions can be:

```
try:
    something
except ExceptOne, msg:
    pass
except ExceptTwo, msg:
    pass
```

-or-

```
try:
    something
except (ExceptOne, ExceptTwo), msg:
    pass
```

You don't have to collect the msg:

```
try:
    something
except ExceptOne:
    pass
```

You can add a generic `except Exception` at the end to collect all the exceptions that you didn't specifically name, or specifically name their parents in the Exception hierarchy, but be careful to track those errors so you can fix them. Don't lose control of your code!
`else`

You can always add an `else`. The `else` clause will happen when no exceptions were raised:

```
try:
    something
except (ExceptOne, ExceptTwo), msg:
    pass
else:
    something_else
```

If you are running Python 2.5, you can add a finally, which will absolutely happen, if the 'try' suite succeeds or if it fails, or even if it contains a 'return' of 'sys.exit(n)'.

```
try:
    something
except (ExceptOne, ExceptTwo), msg:
    pass
else:
    something_else
finally:
    something_that_absolutely_will_happen
```

However, for earlier Python versions, you are allowed either 'except' or 'finally', not both with one 'try'. So you need to use this form:

```
try:
    try:
        something
    finally:
        something_that_absolutely_will_happen
except (ExceptOne, ExceptTwo), msg:
    pass
else:
    something_else
```

for the same effect.

raise1.py

```
#!/usr/bin/env python
"""You can raise an exception anytime you take a notion."""

def GetPositiveNumber(prompt):
    said = raw_input(prompt)
    number = float(said)
    if number < 0:
        raise ValueError, "Number given must be positive."
    return number

if __name__ == '__main__':
    print GetPositiveNumber("Positive number please: ")

"""
$ raise1.py
Positive number please: -2
Traceback (most recent call last):
  File "./raise.py", line 10, in ?
    GetPositiveNumber("Positive number please: ")
  File "./raise.py", line 8, in GetPositiveNumber
    raise ValueError, "Number given must be positive."
ValueError: Number given must be positive.
$ """


```

raise2.py

```
#!/usr/bin/env python
"""And, you can re-raise an exception."""

import raise1

try:
    number = raise1.GetPositiveNumber("Positive number please: ")
except ValueError, msg:
    print "That was wrong!"
    raise    # Raises last exception again

"""
$ raise2.py
Positive number please: -2
That was wrong!
Traceback (most recent call last):
  File "./raise2.py", line 7, in <module>
    number = raise1.GetPositiveNumber("Positive number please: ")
  File "/home/marilyn/python/mm/labs/lab_19_Exceptions/raise1.py", \
       line 8, in get_positive_number
    raise ValueError, "Number given must be positive."
ValueError: Number given must be positive.
$ """


```

raise3.py

```
#!/usr/bin/env python
"""The argument you give to your raise can be anything, a string is
most common, but a tuple is possible."""

import except1

def GetPositiveNumber(prompt):
    said = raw_input(prompt)
    number = float(said)
    if number < 0:
        raise ValueError, ("Number given must be positive.", number)
    return number
try:
    number = GetPositiveNumber("Positive number please: ")
except ValueError, msg:
    print "msg[0] =", msg[0]
    print "msg[1] =", msg[1]

"""
$ raise3.py
Positive number please: -1
msg[0] = Number given must be positive.
msg[1] = -1.0
$ """


```

myexcept1.py

```
#!/usr/bin/env python
"""You can invent your own exception, having it inherit from some
class in the Exception hierarchy."""

class BadNegativeNumber(ArithmeticError):
    pass

def GetPositiveInt(prompt):
    given = int(raw_input(prompt))
    if given < 0:
        raise BadNegativeNumber, ("Non-positive number given.", given)

def main():
    try:
        GetPositiveInt("Number please: ")
    except BadNegativeNumber, msg:
        print msg

if __name__ == '__main__':
    main()
"""
$ myexcept1.py
Number please: -1
('Non-positive number given.', -1)
$
"""
```

myexcept2.py
```
#!/usr/bin/env python
"""You can override and extend the functionality."""

class BadNegativeNumber(ArithmeticError):
    times = 0

    def __init__(self, *args):
        """We call to the base class initializer and add some functionality."""
        ArithmeticError.__init__(self, *args)
        BadNegativeNumber.times += 1

    def __str__(self):
        return "You messed %d %s! %s" % (BadNegativeNumber.times,
            "time" if BadNegativeNumber.times==1 else "times",
            self.args)

def GetPositiveInt(prompt):
    given = int(raw_input(prompt))
    if given < 0:
        raise BadNegativeNumber, ("Non-positive number given.", given)

def main():
    while True:
        try:
            GetPositiveInt("Number please: ")
        except BadNegativeNumber, msg:
            print msg
            continue
        except Exception, msg:
            print msg
            break

if __name__ == '__main__':
    main()
"""$ myexcept2.py
Number please: -2
You messed 1 time! ('Non-positive number given.', -2)
Number please: -3
You messed 2 times! ('Non-positive number given.', -3)
Number please: -1
You messed 3 times! ('Non-positive number given.', -1)
Number please:
invalid literal for int() with base 10: ''
$ """
```

except3.py
```
#!/usr/bin/env python
""" Optional:

Ways to get more info about your caught exception from sys."""

import sys        # gives info -- you can use the traceback
                  # module but it uses sys

def Fun(msg):
    raise ArithmeticError, msg

if __name__ == '__main__':
    try:
        Fun('catch me once')
    except:
        print 'Caught: sys.exc_type =', sys.exc_type
        print 'sys.exc_value =', sys.exc_value
        print 'sys.exc_traceback =', sys.exc_traceback
        print 'sys.exc_info() =', sys.exc_info()
    try:
        Fun(('catch me twice', [1, 2, 3]))
    except Exception, obj:
        print 'obj.args = ', obj.args
        print 'obj = ', obj
"""
$ except3.py
Caught: sys.exc_type = <type 'exceptions.ArithmeticError'>
sys.exc_value = catch me once
sys.exc_traceback = <traceback object at 0xb7f0b5a4>
sys.exc_info() = (<type 'exceptions.ArithmeticError'>,
                  ArithmeticError('catch me once', ),
                 <traceback object at 0xb7f0b5a4>)
obj.args =  ('catch me twice', [1, 2, 3])
obj =  ('catch me twice', [1, 2, 3])
$
"""
```

manynames.py

```
#!/usr/bin/env python
"""manynames.py -- from "Learning Python" by Mark Lutz
and David Ascher, published by O'Reilly.  Demonstrates
name-spaces associated with classes, functions, and
methods."""

x = 11                # Module (global) name/attribute

class C:
    x = 22            # Class attribute
    def M(self):
        x = 33        # Local identifier in method
        self.x = 44   # instance attribute

def F():
    x = 55        # Local identifier in function

def G():
    print x       # Access module x (11)

if __name__ == '__main__':
    obj = C()
    obj.M()
    print obj.x   # 44: instance
    print C.x     # 22: class (a.k.a. obj.x if no x in instance)
    print x       # 11: module (a.k.a. manynames.x outside file)
    G()   # 11: sees the global x
    try:
        print C.M.x  # fails: only visible in method
    except AttributeError, msg:
        print "C.M.x failed:", msg
    try:
        print F.x    # fails: only visible in function
    except AttributeError, msg:
        print "F.x failed:", msg
"""
$ manynames.py
44
22
11
11
C.M.x failed: 'function' object has no attribute 'x'
F.x failed: 'function' object has no attribute 'x'
$ """
```

Python Namespaces

Adapted from many snippets in "Learning Python" by Lutz & Ascher:

Names	Assignment	Reference
Unqualified	x = value makes or changes a local x unless declared global	x as in print x Look in local __dict__, then enclosing namespace's, globals and then built-ins
Qualified o can be a package, module, class, or instantiation of a class ("object" in the oop sense)	o.x = value makes or changes a an x in the object.	o.x Looks for x in o. If o is a class or an instantiation of a class, it looks in the class and in super-classes.

* A "namespace" is created by creating any of these o objects. Each namespace has a __dict__ with its local names. Namespaces are also created by functions and methods. Howver, functions and methods can't be the o in o.x because their namespaces only exist during the function, or method, call.

Nesting namespaces. Python supports namespace nesting.

Functions and classes (and methods, which are functions nested into classes) can nest nest nest within each other.

And anywhere you can make a function or a class, you can also nest in any identifier.

function_nest.py

```
#!/usr/bin/env python
"""function_nest.py Adapted from 'Learning Python'
by Mark Lutz & Davis Ascher"""

x = 11
def F1():
    x = 99              # <- Visible in the nest but not outside
    def F2():
        def F3():
            print x     # <- Can see outside namespaces
            y = 4       # <- Not visible from outside.
        F3()
    F2()

if __name__ == '__main__':
    F1()
"""!
$ python -i function_nest.py
99
>>> F2()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'F2' is not defined
>>> F1.x
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AttributeError: 'function' object has no attribute 'x'
>>>
"""
```

class_nest.py

```
#!/usr/bin/env python
"""class_nest.py class nesting scopes in the other direction."""
w = 10
class C1:
    x = 99
    class C2:
        y = 100
        class C3:
            z = 101      # <-- Visible from outside
            print w
            print z
            # print C1.x   <-- Can't see outer classes
            # print y          or any outer identifiers

if __name__ == '__main__':
    print 'About to initialize:'
    c1 = C1()
"""
$ python -i class_nest.py
10     <------ This output happened when the class was read into
101    <------ the compiler, not when an instance was instantiated.
About to initialize:
>>> dir(c1)
['C2', '__doc__', '__module__', 'x']
>>> c1.C2.C3.z
101
>>> c1.C2.C3.z = [1,2,3]
>>> L = c1.C2.C3.z
>>> L[1] = 'Boo'
>>> c1.C2.C3.z
[1, 'Boo', 3]
>>>
"""
```

Finding Modules

- To find all the modules in your installed version of Python:

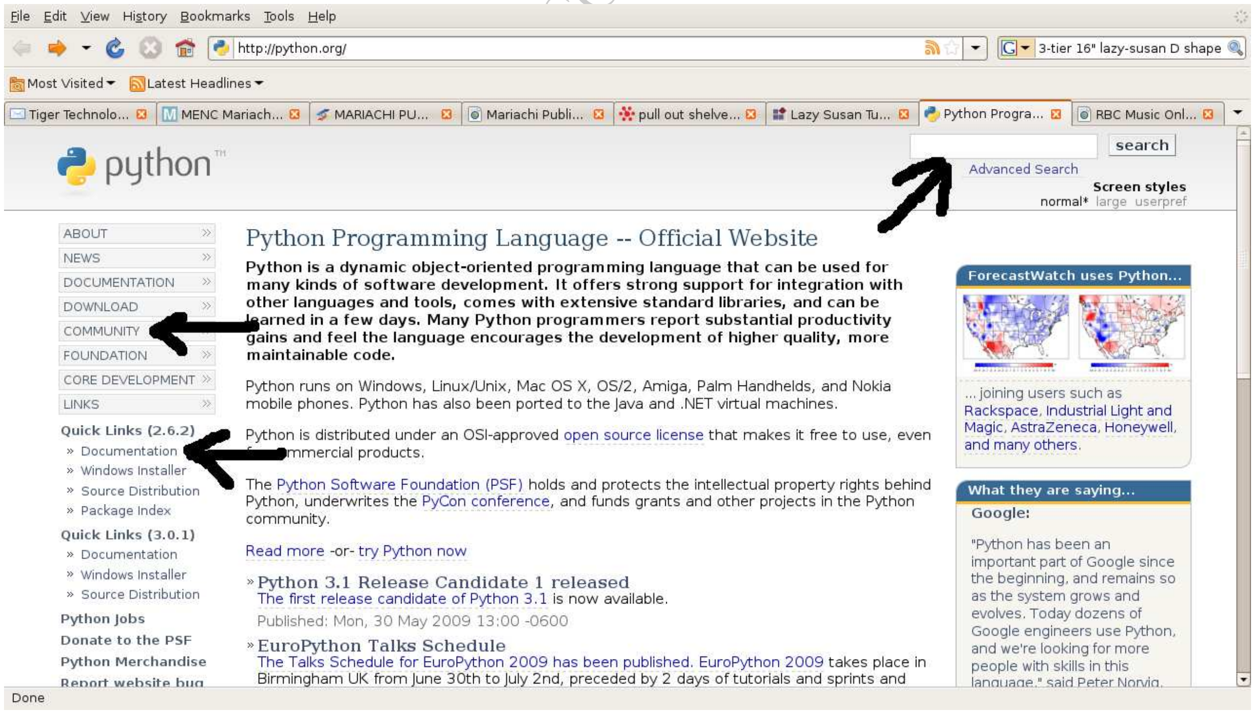
```
>>> help('modules')

Please wait a moment while I gather a list of all available modules...

BaseHTTPServer      atexit              imp                 shelve
Bastion             audiodev            imputil             shlex
[etc.]
```

- To find all the modules that exist at this moment:

1. Go to `http://python.org`:



Python Programming Language -- Official Website

Python is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Many Python programmers report substantial productivity gains and feel the language encourages the development of higher quality, more maintainable code.

Python runs on Windows, Linux/Unix, Mac OS X, OS/2, Amiga, Palm Handhelds, and Nokia mobile phones. Python has also been ported to the Java and .NET virtual machines.

Python is distributed under an OSI-approved open source license that makes it free to use, even for commercial products.

The Python Software Foundation (PSF) holds and protects the intellectual property rights behind Python, underwrites the PyCon conference, and funds grants and other projects in the Python community.

Read more -or- try Python now

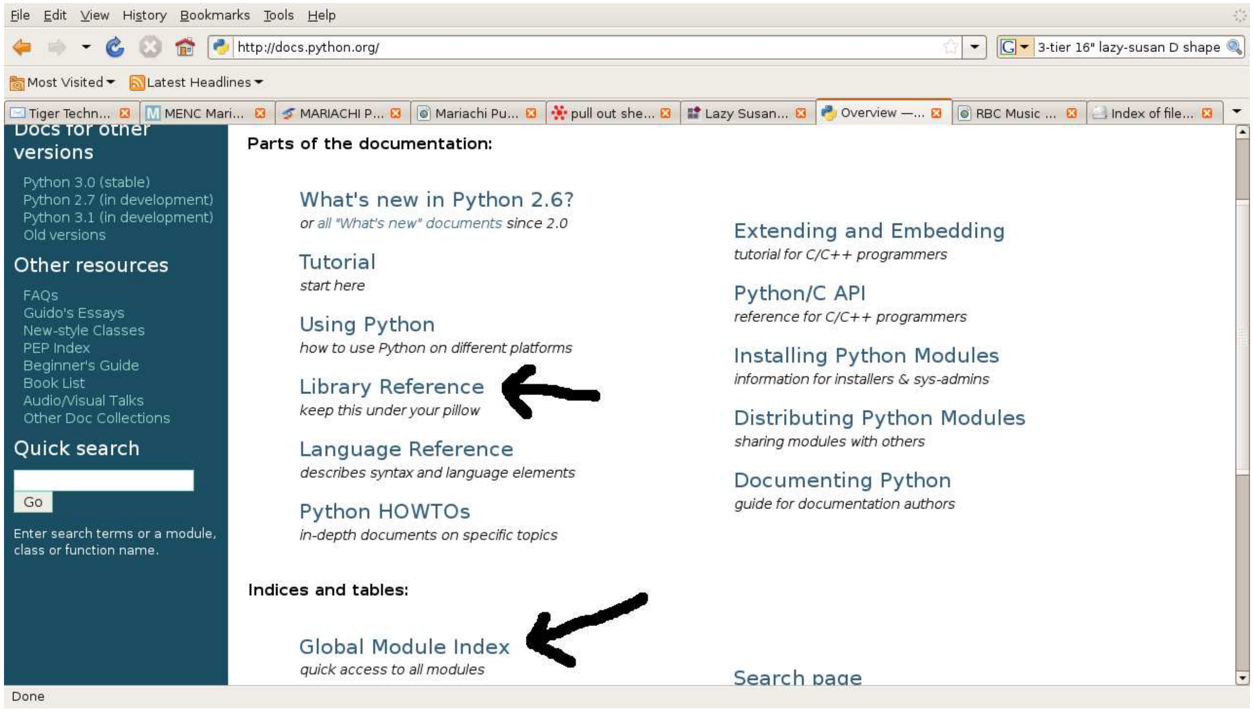
» Python 3.1 Release Candidate 1 released
The first release candidate of Python 3.1 is now available.
Published: Mon, 30 May 2009 13:00 -0600

» EuroPython Talks Schedule
The Talks Schedule for EuroPython 2009 has been published. EuroPython 2009 takes place in Birmingham UK from June 30th to July 2nd, preceded by 2 days of tutorials and sprints and

ForecastWatch uses Python...
... joining users such as Rackspace, Industrial Light and Magic, AstraZeneca, Honeywell, and many others.

What they are saying...
Google:
"Python has been an important part of Google since the beginning, and remains so as the system grows and evolves. Today dozens of Google engineers use Python, and we're looking for more people with skills in this language," said Peter Norvig,

2. Click on **Documentation** on the left, below the version that interests you.


File Edit View History Bookmarks Tools Help

http://docs.python.org/

Docs for other versions

Python 3.0 (stable)
Python 2.7 (in development)
Python 3.1 (in development)
Old versions

Other resources

FAQs
Guido's Essays
New-style Classes
PEP Index
Beginner's Guide
Book List
Audio/Visual Talks
Other Doc Collections

Quick search

Go

Enter search terms or a module, class or function name.

Parts of the documentation:

What's new in Python 2.6?
or all "What's new" documents since 2.0

Tutorial
start here

Using Python
how to use Python on different platforms

Library Reference
keep this under your pillow

Language Reference
describes syntax and language elements

Python HOWTOs
in-depth documents on specific topics

Extending and Embedding
tutorial for C/C++ programmers

Python/C API
reference for C/C++ programmers

Installing Python Modules
information for installers & sys-admins

Distributing Python Modules
sharing modules with others

Documenting Python
guide for documentation authors

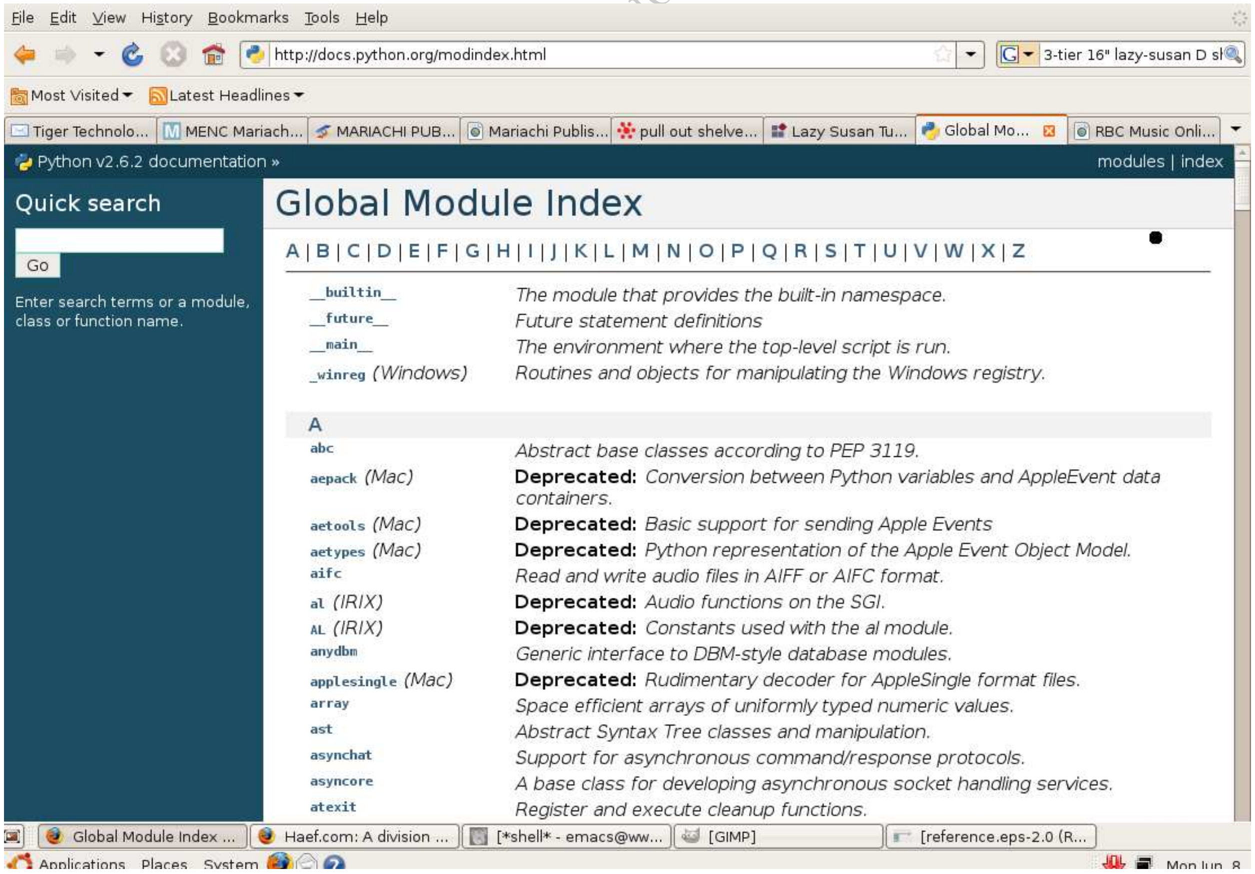
Indices and tables:

Global Module Index
quick access to all modules

Search page

Done

3. Click on **Global Module Index**:


File Edit View History Bookmarks Tools Help

http://docs.python.org/modindex.html

Python v2.6.2 documentation » modules | index

Quick search

Go

Enter search terms or a module, class or function name.

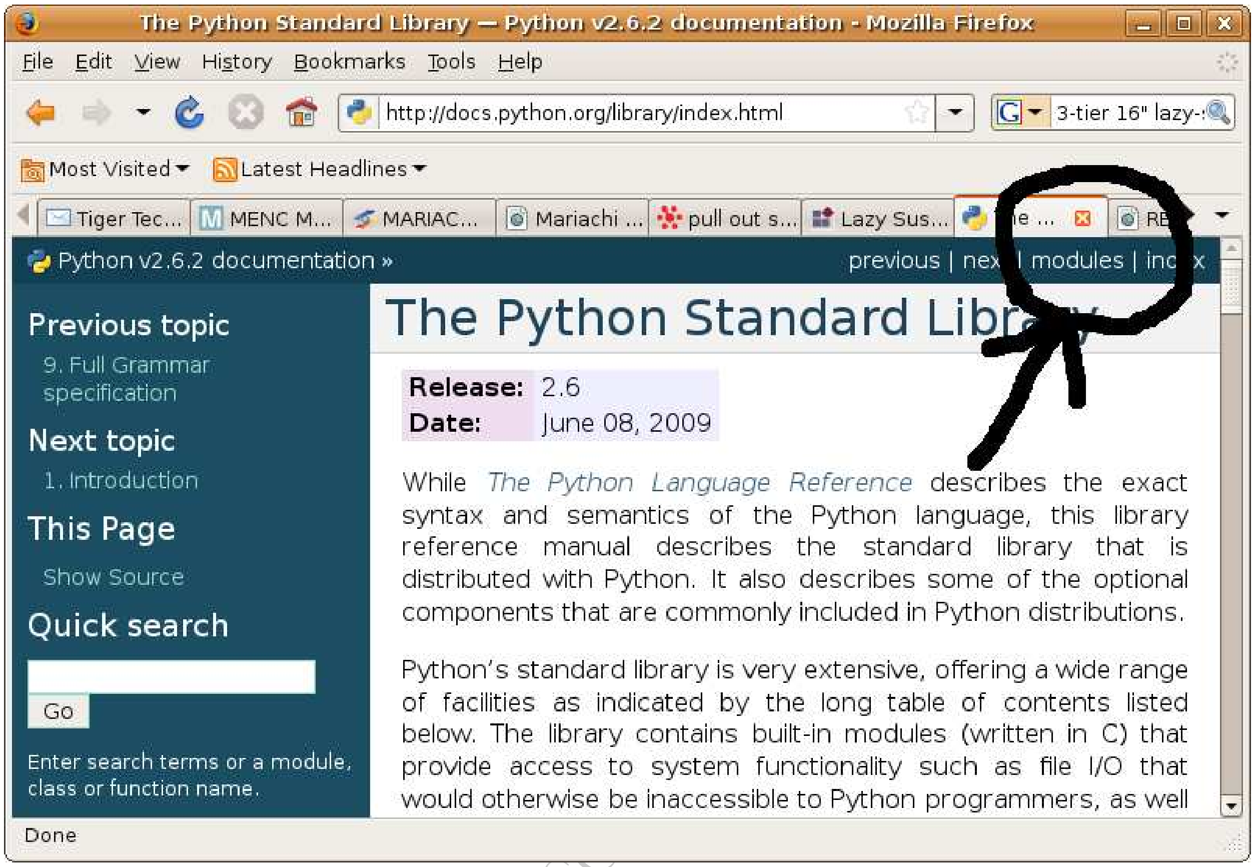
Global Module Index

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Z

`__builtin__`	*The module that provides the built-in namespace.*
`__future__`	*Future statement definitions*
`__main__`	*The environment where the top-level script is run.*
`_winreg` (Windows)	*Routines and objects for manipulating the Windows registry.*
A	
`abc`	*Abstract base classes according to PEP 3119.*
`aepack` (Mac)	**Deprecated:** *Conversion between Python variables and AppleEvent data containers.*
`aetools` (Mac)	**Deprecated:** *Basic support for sending Apple Events*
`aetypes` (Mac)	**Deprecated:** *Python representation of the Apple Event Object Model.*
`aifc`	*Read and write audio files in AIFF or AIFC format.*
`al` (IRIX)	**Deprecated:** *Audio functions on the SGI.*
`AL` (IRIX)	**Deprecated:** *Constants used with the al module.*
`anydbm`	*Generic interface to DBM-style database modules.*
`applesingle` (Mac)	**Deprecated:** *Rudimentary decoder for AppleSingle format files.*
`array`	*Space efficient arrays of uniformly typed numeric values.*
`ast`	*Abstract Syntax Tree classes and manipulation.*
`asynchat`	*Support for asynchronous command/response protocols.*
`asyncore`	*A base class for developing asynchronous socket handling services.*
`atexit`	*Register and execute cleanup functions.*

– or –

1. From that main **Documents** page, click on the **Library Reference**:
2. Then click on the `modules` link near the upper right corner.


The Python Standard Library — Python v2.6.2 documentation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://docs.python.org/library/index.html
Most Visited
Latest Headlines
Python v2.6.2 documentation »
previous | next | modules | index
Previous topic
9. Full Grammar specification
Next topic
1. Introduction
This Page
Show Source
Quick search
Go
Enter search terms or a module, class or function name.
Done

The Python Standard Library

Release: 2.6
Date: June 08, 2009

While *The Python Language Reference* describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well

Finding Help

- Use Python's builtin `dir` and `help` facility:
- Use the search box on the python.org home page.
- Click on **Community** from the python.org home page.
- Click on **Mailing Lists, Newsgroups, and Web Forums**. There are many helpful communities there. As a new Python person you might find the **tutor** email list welcoming and helpful.

Lab 18

Optional Reading

Read Guido's tutorial about exceptions. To find it, go to `python.org` and search for "tutorial". From there use `Ctrl + f` to find "Exceptions".

Exercises

1. Make an `UpIt(str)` function that returns the input string, but with all caps. Your UpIt function will be different from `str.upper()` in that, if any of the characters in the input string are already uppercase, it raises an exception. Invent your own exception and put it in a reasonable spot in the exception hierarchy.

2. Write a function, `Get_XY(prompt)`, that prompts the user and returns a tuple of two floats.

   If the user answers some form of `quit` or just hits the enter key, or `Ctrl-D` or `Ctrl-C Ctrl-C`, the function returns `None`.

   All errors are explained and the user is asked to *Please try again.*

   Use your function to collect two sides of a right triangle and give the hypotenuse. Reminder: $x^2 + y^2 = hypotenuse^2$. You'll need to `import math`.

3. Experience these brain-teasers in the lab so you're ready for work:

   (a) First think about this and make a prediction:

   ```
   >>> a_list = ['loop']
   >>> a_list += a_list
   >>> a_list

   --------------------------------------
   >>> a_list.append(a_list)
   >>> a_list

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

   Now, try it.

   Python uses **...** to indicate this infinite pattern.

   (b) Again, first think about it and then give it a try:

   ```
   >>> bottles = 100
   >>> def HowMany():
   ...     print bottles
   ```

```
...
>>> HowMany()

--------------------------------------

>>> def HowMany():
...     bottles -= 1
...     print bottles
...
>>> HowMany()
```

How can you fix this so that `HowMany`'s `bottles` is really the global bottles?

(c) Try this:

```
>>> def Snake(rattle=[]):
...     rattle += ["hiss"]
...     print rattle
...
>>> Snake([100])
--------------------------------------

>>> Snake()
--------------------------------------

>>> Snake()
--------------------------------------

>>> Snake()
--------------------------------------
```

(d) Study this one, predict the output, and then give it a try:

```
class X:
    pass
class Y:
    pass
X.a = 1
X.b = 2
X.c = 3
Y.a = X.a + X.b + X.c

for X.i in range(Y.a):
    print X.i

print dir(X)
```