Hello, world!

In [2]:
```
print "Hello, world!"
```

```
Hello, world!
```

In [3]:
```
def sayhello():
    print "Hello, world!"
```

In [4]:
```
sayhello()
```

```
Hello, world!
```

In [5]:
```
def sayhello(name):
    print "Hello, " + name
```

In [6]:
```
sayhello('Rick')
```

```
Hello, Rick
```

Exercise: Write a “Hello, world” program that uses a function with a parameter to greet you by name

Basic types

In [7]:
```
1 # integer
```

Out[7]: 1

In [8]:
```
3.14 # floating-point
```

Out[8]: 3.14

In [9]:
```
3.14e17 # floating-point (scientific notation)
```

Out[9]: 3.14e+17

In [10]:
```
123456789123456789123456789 # long integer (unbounded)
```

Out[10]: 123456789123456789123456789L

In [11]:
```
# Type introspection
print type(4)
print type(3.14)
```

```
<type 'int'>
<type 'float'>
```

In [12]:
```
1 + 1j
```

Out[12]: (1+1j)

In [13]:
```
1j * 1j
```

Out[13]: (-1+0j)

Variables and basic arithmetic

In [14]:
```
x = 1
x
```

Out[14]: 1

In [15]:
```
x + 1
```

Out[15]: 2

In [16]:
```
y = x * 5
y
```

Out[16]: 5

In [17]:
```
y *= 3 # in-place assigment
print y
```

```
15
```

In [18]:
```
y % 2 # modulo arithmetic
```

Out[18]: 1

In [19]:
```
y / 2 # result dependent on Python version
```

Out[19]: 7

In [20]:
```
y / 2.0 # force floating point
```

Out[20]: 7.5

In [21]:
```
y // 2.0 # force integer (floor) division)
```

Out[21]: 7.0

In [22]:
```
2==2
```

Out[22]: True

In [23]:
```
2 <= 1 # comparisons are < > <= >= == !=
```

Out[23]: False

In [84]:
```
2 != 3
```

Out[84]: True

In [85]:
```
# exponentiation
2 ** 4
```

Out[85]: 16

In [25]:
```
True and False
```

Out[25]: False

In [26]:
```
True or False
```

Out[26]: True

In [27]:
```
True and not False
```

Out[27]: True

Strings

In [28]:
```
print 'This is a string with single quotes'
print "This is a string with double quotes"
```

```
This is a string with single quotes
This is a string with double quotes
```

In [29]:
```
print "We can embed 'single quotes' within double quotes without escaping"
print 'We can also embed "double quotes" within single quotes without escaping'
print "But we must escape \"double quotes\" inside double quotes and single quotes inside singl
```

```
We can embed 'single quotes' within double quotes without escaping
We can also embed "double quotes" within single quotes without escaping
But we must escape "double quotes" inside double quotes and single quotes inside single quotes
```

In [30]:
```
print "Strings can be" + " concatenated together " + "using the '+' operator"
```

```
Strings can be concatenated together using the '+' operator
```

In [31]:
```
print '''We can also begin a string with three single-quote characters
or three double-quote characters. This allows us to build multi-line
strings without using the \\n escape sequence.'''
```

```
We can also begin a string with three single-quote characters
or three double-quote characters. This allows us to build multi-line
strings without using the \n escape sequence.
```

In [32]:
```
print u'Unicode strings have a "u" prefix.'
print u'You can use encoded unicode characters in your source code: 海淀区科学院南路2号融科资讯中心C
```

```
Unicode strings have a "u" prefix.
You can use encoded unicode characters in your source code: 海淀区科学院南路2号融科资讯中心C座南楼8层
```

We *can* use escape sequences inside strings, however. Escape sequences start with a backslash character (). Some common escape sequences appear below:

- \n - newline
- \r - carriage return
- \t - tab character
- \ - literal backslash

- \x20 - hex escape (in this case, ASCII character 0x20, the space)
- \u4e0b - unicode escape (hex sequence)

You can index into a string using either a single integer or a "slice." The result is always another string (there is no 'char' type in Python).

In [33]:
```
s = 'The quick brown fox'
s[0]
```

Out[33]: 'T'

In [34]:
```
type(s[0])
```

Out[34]: str

In [35]:
```
s[1:15]
```

Out[35]: 'he quick brown'

In [36]:
```
s[1:15:2] # 'step' by 2
```

Out[36]: 'h uc rw'

In [37]:
```
s[:3] # beginning of range omitted
```

Out[37]: 'The'

In [38]:
```
s[3:] # end of range omitted
```

Out[38]: ' quick brown fox'

In [39]:
```
s[-1] # negative indexing from the end
```

Out[39]: 'x'

In [40]:
```
s[-3:] # last 3 characters
```

Out[40]: 'fox'

In [41]:
```
s[::-1] # negative step reverses the sequence
```

Out[41]: 'xof nworb kciuq ehT'

In [86]:
```
len(s) # length of string
```

Out[86]: 19

In [89]:
```
x = u'海'
print len(x) # unicode length
```

```
1
```

In [94]:
```
y = x.encode('utf-8') # encode to bytes ('str')
len(y)
```

Out[94]: 3

In [96]:
```
# decode utf-8 back to unicode
print unicode(y, 'utf-8')
```

```
海
```

Lists

Lists are mutable, dynamically typed sequences, and they are used frequently in Python.

In [42]:
```
# This is a list
print [ 1,2,3]
```

```
[1, 2, 3]
```

In [43]:
```
# We can append to lists
lst = [ 1, 2, 3 ]
lst.append(4)
print lst
```

```
[1, 2, 3, 4]
```

In [44]:
```
# We can index into lists (0-based indexing)
print lst[2]
```

```
3
```

In [45]:
```
# We can insert, remove, and update
lst.insert(1, 42)
print lst
```

```
[1, 42, 2, 3, 4]
```

In [46]:
```
lst.remove(2)
print lst
```

```
[1, 42, 3, 4]
```

In [47]:
```
del lst[2]
print lst
```

```
[1, 42, 4]
```

In [48]:
```
lst[1] = 43
print lst
```

```
[1, 43, 4]
```

In [49]:
```
# pop() removes and returns the last element
print lst.pop()
```

```
4
```

In [50]:
```
print lst
```

```
[1, 43]
```

In [51]:
```
# Lists are dynamically typed
lst = [ 'This', 'is', 'a', 'list', 'with', 7, 'elements' ]
lst
```

Out[51]:
```
['This', 'is', 'a', 'list', 'with', 7, 'elements']
```

In [52]:
```
# Lists can be sliced just like strings
lst[::-2]
```

Out[52]:
```
['elements', 'with', 'a', 'This']
```

Tuples

Tuples are *immutable* sequences.

In [53]:
```
# this is a tuple
print (1, 2, 3)
```

```
(1, 2, 3)
```

In [54]:
```
print ('Tuples', 'are', 'also', 'dynamically', 'typed', 42)
```

```
('Tuples', 'are', 'also', 'dynamically', 'typed', 42)
```

In [55]:
```
# Tuples can be indexed
t = (1,2,3)
print t[0]
```

```
1
```

In [56]:
```
# Tuples can be "unpacked"
x,y = (1,2)
print x
print y
```

```
1
2
```

Basic control structures

In [57]:
```
if True:
    print 'This is true'
else:
    print 'This is false'
```

```
This is true
```

In [58]:
```
x = 0
while x < 5:
    x = x + 1
    print x
```

```
1
2
3
4
5
```

In [59]:
```
lst = [ 1, 2, 3 ]
for x in lst:
    print x
```

```
1
2
3
```

In [60]:
```
print range(3)
for x in range(3):
    print x
```

```
[0, 1, 2]
0
1
2
```

In [61]:
```
print xrange(3)
for x in xrange(3):
    print x
```

```
xrange(3)
0
1
2
```

In [62]:
```
for x in xrange(10):
    if x % 2 == 0:
        continue
    if x > 5: break
    print x
```

```
1
3
5
```

Exercises

- Write a function that sums the values in a list using a `for` loop
- Write a function that sums the even-numbered values in a list
- Write a function that returns the reversed version of a list

Dicts

A *dict* is a hash table (also known as a "dictionary"). Dicts are pervasive in Python.

In [63]:
```
print { 'key': 'value' }
```

```
{'key': 'value'}
```

In [64]:
```
d = { 'key1': 1, 'key2': 'foo' }
print d
```

```
{'key2': 'foo', 'key1': 1}
```

In [65]:
```
d['key1']
```

Out[65]: 1

In [66]:
```
d['key3'] = 'bar'
```

In [67]:
```
print d
```

```
{'key3': 'bar', 'key2': 'foo', 'key1': 1}
```

In [68]:
```
# dicts are unordered, but we can get a list of their keys,
# values, or (key,value) pairs
print d.keys()
```

```
['key3', 'key2', 'key1']
```

In [69]:
```
print d.values()
```

```
['bar', 'foo', 1]
```

In [70]:
```
print d.items()
```

```
[('key3', 'bar'), ('key2', 'foo'), ('key1', 1)]
```

In [71]:
```
# dict keys can be any *immutable* type in Python
d = { 'foo': 1, 2: 'bar' }
print d
```

```
{2: 'bar', 'foo': 1}
```

In [72]:
```
d[(1,2)] = 'baz'
print d
```

```
{(1, 2): 'baz', 2: 'bar', 'foo': 1}
```

Items can be removed using the `del` keyword

In [73]:
```
del d[2]
print d
```

```
{(1, 2): 'baz', 'foo': 1}
```

A `dict` can be iterated through in a space-efficient using `iterkeys`, `itervalues`, and `iteritems`:

In [74]:
```
for k in d.iterkeys():
    print k
```

```
(1, 2)
foo
```

In [75]:
```
for v in d.itervalues():
    print v
```

```
baz
1
```

In [76]:
```
for k,v in d.iteritems():
    print k, v
```

```
(1, 2) baz
foo 1
```

In [77]:
```
'foo' in d # Test for key membership
```

Out[77]:
```
True
```

Exceptions

Python handles errors by throwing *exceptions*. For instance, trying to read a non-existent key in a dict:

In [78]:
```
d['does not exist']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/vagrant/<ipython-input-78-17d039c47522> in <module>()
----> 1 d['does not exist']

KeyError: 'does not exist'
```

To handle exceptions gracefully, we must enclose them in a `try:` *block*:

In [79]:
```
try:
    x = d['does not exist']
    print 'This statement never executes!'
except KeyError:
    print 'There was a key error!'
```

```
There was a key error!
```

We can also write code that will *always* run, whether an exception is raised or not:

In [80]:
```
try:
    x = d['does not exist']
except KeyError:
    print 'There was a key error!'
finally:
    print 'This always runs!'
```

```
There was a key error!
This always runs!
```

In [81]:
```
try:
    x = d['key1']
except KeyError:
    print 'There was a key error!'
finally:
    print 'This always runs!'
```

```
There was a key error!
This always runs!
```

If we want code that only runs when there is *not* an error, we can use the `else:` clause:

In [82]:
```
try:
    x = d['key1']
except KeyError:
    print 'There was a key error!'
else:
    print 'The try: block completed without error.'
finally:
    print 'This always runs!'
```

```
There was a key error!
This always runs!
```

To *cause* an exception, use the `raise` keyword:

In [83]:
```
raise KeyError('This is a key error')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/vagrant/<ipython-input-83-04dfd7050815> in <module>()
----> 1 raise KeyError('This is a key error')

KeyError: 'This is a key error'
```

Exercises

- Write a collection of functions to manage a telephone directory. The directory should be stored in a dict and passed as the first argument to each function. These functions should include `add_number(directory, name, number)`, `remove_number(directory, name)`, and `lookup_number(directory, name)`.
- Update your function so that `remove_number` does not raise an exception when you remove a non-existent entry.

Useful Builtins

zip

In [1]:
```
zip([1,2,3], ['a', 'b', 'c'])
```

Out[1]:
```
[(1, 'a'), (2, 'b'), (3, 'c')]
```

In [3]:
```
x = [[ 1, 2, 3 ],
     [ 4, 5, 6 ],
     [ 7, 8, 9 ] ]
```

In [5]:
```
zip(x[0], x[1], x[2])
```

Out[5]:
```
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

In [6]:
```
zip(*x)
```

Out[6]:
```
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

enumerate

In [7]:
```
lst = [ 'foo', 'bar', 'baz' ]
```

In [8]:
```
# Don't do this:
for x in range(len(lst)):
    print x, lst[x]
```

```
0 foo
1 bar
2 baz
```

In [11]:
```
# Do this instead:
for index, x in enumerate(lst):
    print index, x
```

```
0 foo
1 bar
2 baz
```

eval

In [18]:
```
eval('5+5')
```

Out[18]:
```
10
```

In [19]:
```
x = [1,2,3]
r_x = repr(x)
eval(r_x)
```

Out[19]: [1, 2, 3]

dir

In [23]:
```
x = 5
dir(x)
```

Out[23]:
```
['__abs__',
 '__add__',
 '__and__',
 '__class__',
 '__cmp__',
 '__coerce__',
 '__delattr__',
 '__div__',
 '__divmod__',
 '__doc__',
 '__float__',
 '__floordiv__',
 '__format__',
 '__getattribute__',
 '__getnewargs__',
 '__hash__',
 '__hex__',
 '__index__',
 '__init__',
 '__int__',
 '__invert__',
 '__long__',
 '__lshift__',
 '__mod__',
 '__mul__',
 '__neg__',
 '__new__',
 '__nonzero__',
 '__oct__',
 '__or__',
 '__pos__',
 '__pow__',
 '__radd__',
 '__rand__',
 '__rdiv__',
 '__rdivmod__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rfloordiv__',
 '__rlshift__',
 '__rmod__',
 '__rmul__',
 '__ror__',
 '__rpow__',
 '__rrshift__',
 '__rshift__',
 '__rsub__',
 '__rtruediv__',
 '__rxor__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__truediv__',
 '__trunc__',
 '__xor__',
 'bit_length',
 'conjugate',
 'denominator',
 'imag',
 'numerator',
```

In [24]:
```
x.bit_length()
```

Out[24]:
```
3
```

In [35]:
```
x = 'Foo'
dir(x)
```

Out[35]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__getslice__',
 '__gt__',
 '__hash__',
 '__init__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '_formatter_field_name_split',
 '_formatter_parser',
 'capitalize',
 'center',
 'count',
 'decode',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'index',
 'isalnum',
 'isalpha',
 'isdigit',
 'islower',
 'isspace',
 'istitle',
 'isupper',
 'join',
 'ljust',
 'lower',
 'lstrip',
 'partition',
 'replace',
 'rfind',
 'rindex',
 'rjust',
 'rpartition',
 'rsplit',
 'rstrip',
 'split',
```

In [37]:
```
help(x.partition)
```

```
Help on built-in function partition:

partition(...)
    S.partition(sep) -> (head, sep, tail)

    Search for the separator sep in S, and return the part before it,
    the separator itself, and the part after it.  If the separator is not
    found, return S and two empty strings.
```

`ord` and `chr`

In [38]:
```
ord('a')
```

Out[38]: 97

In [39]:
```
chr(97)
```

Out[39]: 'a'

`map` and `filter`

In [41]:
```
x = map(ord, 'Foo')
x
```

Out[41]: [70, 111, 111]

In [42]:
```
map(chr, x)
```

Out[42]: ['F', 'o', 'o']

In [43]:
```
def is_even(num):
    return num % 2 == 0
filter(is_even, range(10))
```

Out[43]: [0, 2, 4, 6, 8]

`sum`, `max`, `min`, and `len`

In [44]:
```
sum(range(10))
```

Out[44]: 45

In [45]:
```
max(range(10))
```

Out[45]: 9

In [46]:
```
min(range(10))
```

Out[46]: 0

In [47]:
```
len(range(10))
```

Out[47]: 10

repr

In [48]:
```
repr(1)
```

Out[48]: '1'

In [49]:
```
repr('foo')
```

Out[49]: "'foo'"

In [50]:
```
repr([1,2,4])
```

Out[50]: '[1, 2, 4]'

Basic types

In [51]:
```
int('5')
```

Out[51]: 5

In [52]:
```
int('ff', base=16)
```

Out[52]: 255

In [53]:
```
float('5')
```

Out[53]: 5.0

In [54]:
```
float(5)
```

Out[54]: 5.0

In [57]:
```
list(), list([1,2,3])
```

Out[57]: ([], [1, 2, 3])

In [58]:
```
tuple([1,2,3])
```

Out[58]: (1, 2, 3)

In [61]:
```
keys = range(4)
values = ['a', 'b', 'c', 'd']
dct = dict(zip(keys, values))
dct
```

Out[61]:
```
{0: 'a', 1: 'b', 2: 'c', 3: 'd'}
```

In [62]:
```
dict(foo=1, bar=2, baz=3)
```

Out[62]:
```
{'bar': 2, 'baz': 3, 'foo': 1}
```

In [63]:
```
unicode('abcd')
```

Out[63]:
```
u'abcd'
```

Exercises

- Given that you have a list of keys and a list of values, how would you create a `dict` containing the key/value pairs
- Write a function that converts a list of ASCII values to a string. Test it on the string `[86, 77, 87, 97, 114, 101]`

File I/O

In [14]:
```
fp = open('/etc/hosts')
print fp.read()
fp.close()
```

```
127.0.0.1       localhost
127.0.1.1       precise64

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

In [15]:
```
fp = open('/etc/hosts')
print repr(fp.read(40))
print repr(fp.read(40))
print repr(fp.read(40))
fp.close()
```

```
'127.0.0.1\tlocalhost\n127.0.1.1\tprecise64\n'
'\n# The following lines are desirable for'
' IPv6 capable hosts\n::1     ip6-localhos'
```

In [16]:
```
fp = open('/etc/hosts')
for line in fp:
    print repr(line)
fp.close()
```

```
'127.0.0.1\tlocalhost\n'
'127.0.1.1\tprecise64\n'
'\n'
'# The following lines are desirable for IPv6 capable hosts\n'
'::1     ip6-localhost ip6-loopback\n'
'fe00::0 ip6-localnet\n'
'ff00::0 ip6-mcastprefix\n'
'ff02::1 ip6-allnodes\n'
'ff02::2 ip6-allrouters\n'
```

In [17]:
```
fp = open('/etc/hosts')
fp.readlines()
```

Out[17]:
```
['127.0.0.1\tlocalhost\n',
 '127.0.1.1\tprecise64\n',
 '\n',
 '# The following lines are desirable for IPv6 capable hosts\n',
 '::1     ip6-localhost ip6-loopback\n',
 'fe00::0 ip6-localnet\n',
 'ff00::0 ip6-mcastprefix\n',
 'ff02::1 ip6-allnodes\n',
 'ff02::2 ip6-allrouters\n']
```

In [18]:
```
print fp.read()
```

In [20]:
```
fp.seek(10)
print fp.read()
```

```
localhost
127.0.1.1       precise64

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

In [21]:
```
fp.tell()
```

Out[21]: 224

In [22]:
```
fp = open('/tmp/testfile.txt', 'w')
fp.write('abcd\n')
fp.close()
```

In [25]:
```
fp = open('/tmp/testfile.txt')
fp.read()
```

Out[25]: 'abcd\n'

In [26]:
```
fp.write('hi')
```

```
---------------------------------------------------------------------------
IOError                                   Traceback (most recent call last)
/vagrant/<ipython-input-26-40576b043654> in <module>()
----> 1 fp.write('hi')

IOError: File not open for writing
```

In [28]:
```
fp = open('/tmp/testfile.txt', 'a')
fp.write('hi\n')
fp.close()
open('/tmp/testfile.txt').read()
```

Out[28]: 'abcd\nhi\nhi\n'

In [29]:
```
fp = open('/tmp/testfile.txt', 'w')
fp.write('hi\n')
fp.close()
open('/tmp/testfile.txt').read()
```

Out[29]: 'hi\n'

In [31]:
```
fp = open('/tmp/testfile.txt', 'w')
fp.write('hi\n')
fp.seek(0)
fp.read()
```

```
---------------------------------------------------------------------------
IOError                                   Traceback (most recent call last)
/vagrant/<ipython-input-31-6bf0db77664c> in <module>()
      2 fp.write('hi\n')
      3 fp.seek(0)
----> 4 fp.read()

IOError: File not open for reading
```

In [40]:
```
fp = open('/tmp/testfile.txt', 'w+')
fp.write('hi\n')
fp.seek(0)
fp.read()
```

Out[40]: 'hi\n'

In [41]:
```
fp = open('/tmp/testfile.txt', 'r+')
fp.seek(2)
fp.write('there\n')
fp.seek(0)
fp.read()
```

Out[41]: 'hithere\n'

In [42]:
```
fp = open('/tmp/testfile.txt', 'a+')
fp.write('again\n')
fp.seek(0)
fp.read()
```

Out[42]: 'hithere\nagain\n'

Exercises

- Write a function that will print out a text file, line by line
- Enhance the function to print a line number before each line

Using Modules

Python's basic unit of reusable code is the *module*. You can access the functions and classes inside a module using the `import` statement. One of the most important modules is the `sys` module:

In [89]:
```
import sys
print sys
```

```
<module 'sys' (built-in)>
```

The `import` statement can also be used to *alias* a module:

In [90]:
```
import sys as mysys
print mysys
```

```
<module 'sys' (built-in)>
```

We can also `import` one or more names from a module:

In [91]:
```
from sys import path
print path
```

```
['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-linux2', '/usr/lib/python2.7/lib-tk', '/usr/
```

In [92]:
```
from sys import path as mypath
print mypath
```

```
['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-linux2', '/usr/lib/python2.7/lib-tk', '/usr/
```

One of Python's nicest features is its *introspection* capabilities. For instance, to get a list of the properties available on an object, we can use the builtin function `dir()`:

In [93]:
```
dir(sys)
```

Out[93]:
```
['__displayhook__',
 '__doc__',
 '__excepthook__',
 '__name__',
 '__package__',
 '__stderr__',
 '__stdin__',
 '__stdout__',
 '_clear_type_cache',
 '_current_frames',
 '_getframe',
 '_mercurial',
 'api_version',
 'argv',
 'builtin_module_names',
 'byteorder',
 'call_tracing',
 'callstats',
 'copyright',
 'displayhook',
 'dont_write_bytecode',
 'exc_clear',
 'exc_info',
 'exc_type',
 'excepthook',
 'exec_prefix',
 'executable',
 'exit',
 'exitfunc',
 'flags',
 'float_info',
 'float_repr_style',
 'getcheckinterval',
 'getdefaultencoding',
 'getdlopenflags',
 'getfilesystemencoding',
 'getprofile',
 'getrecursionlimit',
 'getrefcount',
 'getsizeof',
 'gettrace',
 'hexversion',
 'last_traceback',
 'last_type',
 'last_value',
 'long_info',
 'maxint',
 'maxsize',
 'maxunicode',
 'meta_path',
 'modules',
 'path',
 'path_hooks',
 'path_importer_cache',
 'platform',
 'prefix',
 'py3kwarning',
 'pydebug',
 'setcheckinterval',
 'setdlopenflags',
 'setprofile',
 'setrecursionlimit',
 'settrace',
 'stderr',
```

If we want more information about something, we can also use the `help()` builtin function:

```
In [94]: help(sys.setprofile)
```

```
Help on built-in function setprofile in module sys:

setprofile(...)
    setprofile(function)

    Set the profiling function.  It will be called on each function call
    and return.  See the profiler chapter in the library manual.
```

The `sys` module

`sys` contains functions and variables related to the running Python program. In particular, if you wish to use command-line arguments, these are accessed via `sys.argv`.

```
In [95]: print sys.argv
```

```
['-c', '-f', '/home/vagrant/.ipython/profile_default/security/kernel-d6b2e81a-770d-4c48-912b-a11
```

You can also see the exact executable file containing your Python interpreter:

```
In [96]: print sys.executable
```

```
/usr/bin/python
```

Access to the standard input, output, and error streams is also through the `sys` module:

```
In [97]: print sys.stdin, sys.stdout, sys.stderr
```

```
<open file '<stdin>', mode 'r' at 0x7f951289e150> <IPython.zmq.iostream.OutStream object at 0x15
```

```
In [98]: sys.stderr.write('This is written to the stderr stream\n')
```

```
This is written to the stderr stream
```

```
In [99]: sys.stderr.write('This is written to the stdout stream\n')
```

```
This is written to the stdout stream
```

The `sys.path` variable gives you access to the search path the Python interpreter uses to find modules to `import`:

```
In [100]: sys.path
```

```
Out[100]: ['',
           '/usr/lib/python2.7',
           '/usr/lib/python2.7/plat-linux2',
           '/usr/lib/python2.7/lib-tk',
           '/usr/lib/python2.7/lib-old',
           '/usr/lib/python2.7/lib-dynload',
           '/usr/local/lib/python2.7/dist-packages',
           '/usr/lib/python2.7/dist-packages',
           '/usr/lib/pymodules/python2.7',
           '/usr/lib/python2.7/dist-packages/IPython/extensions']
```

You can also access various constants describing your system such as the largest integer:

In [101]:
```
sys.maxint
```

Out[101]: 9223372036854775807

The os module

Where the `sys` module gives access to information about the current Python process, the `os` module provides several functions for accessing low-level operating system information:

In [102]:
```
import os
dir(os)
```

Out[102]:
```
['EX_CANTCREAT',
 'EX_CONFIG',
 'EX_DATAERR',
 'EX_IOERR',
 'EX_NOHOST',
 'EX_NOINPUT',
 'EX_NOPERM',
 'EX_NOUSER',
 'EX_OK',
 'EX_OSERR',
 'EX_OSFILE',
 'EX_PROTOCOL',
 'EX_SOFTWARE',
 'EX_TEMPFAIL',
 'EX_UNAVAILABLE',
 'EX_USAGE',
 'F_OK',
 'NGROUPS_MAX',
 'O_APPEND',
 'O_ASYNC',
 'O_CREAT',
 'O_DIRECT',
 'O_DIRECTORY',
 'O_DSYNC',
 'O_EXCL',
 'O_LARGEFILE',
 'O_NDELAY',
 'O_NOATIME',
 'O_NOCTTY',
 'O_NOFOLLOW',
 'O_NONBLOCK',
 'O_RDONLY',
 'O_RDWR',
 'O_RSYNC',
 'O_SYNC',
 'O_TRUNC',
 'O_WRONLY',
 'P_NOWAIT',
 'P_NOWAITO',
 'P_WAIT',
 'R_OK',
 'SEEK_CUR',
 'SEEK_END',
 'SEEK_SET',
 'ST_APPEND',
 'ST_MANDLOCK',
 'ST_NOATIME',
 'ST_NODEV',
 'ST_NODIRATIME',
 'ST_NOEXEC',
 'ST_NOSUID',
 'ST_RDONLY',
 'ST_RELATIME',
 'ST_SYNCHRONOUS',
 'ST_WRITE',
 'TMP_MAX',
 'UserDict',
 'WCONTINUED',
 'WCOREDUMP',
 'WEXITSTATUS',
 'WIFCONTINUED',
 'WIFEXITED',
 'WIFSIGNALED',
```

In [103]:
```
os.listdir('/usr')
```

Out[103]:
```
['include', 'src', 'local', 'lib', 'sbin', 'share', 'bin', 'games']
```

In [104]:
```
fd = os.popen('ls -l')
```

In [105]:
```
print fd.read()
```

```
total 112
-rw-r--r-- 1 vagrant vagrant 37639 Oct  5 00:16 Python Basic Syntax.ipynb
-rw-r--r-- 1 vagrant vagrant 17385 Oct  5 00:32 String Processing.ipynb
-rw-r--r-- 1 vagrant vagrant 45232 Oct  5 00:06 Using Modules.ipynb
-rw-r--r-- 1 vagrant vagrant  4008 Oct  3 03:20 Vagrantfile
```

Besides normal modules in Python, there are also modules containing other modules. These are called *packages*. The `os` module is such a package; inside it is the `os.path` module, used for manipulating filesystem pathnames:

In [106]:
```
os.path
```

Out[106]:
```
<module 'posixpath' from '/usr/lib/python2.7/posixpath.pyc'>
```

In [107]:
```
os.path.abspath('.')
```

Out[107]:
```
'/vagrant'
```

In [108]:
```
os.path.dirname(sys.executable)
```

Out[108]:
```
'/usr/bin'
```

In [109]:
```
os.path.basename(sys.executable)
```

Out[109]:
```
'python'
```

In [110]:
```
os.path.join('/usr/local', 'bin', 'foo')
```

Out[110]:
```
'/usr/local/bin/foo'
```

In [111]:
```
os.path.normpath('/usr/local/bin/../../bin')
```

Out[111]:
```
'/usr/bin'
```

In [112]:
```
os.path.expanduser('~')
```

Out[112]:
```
'/home/vagrant'
```

In [113]:
```
os.path.expandvars('$HOME')
```

Out[113]:
```
'/home/vagrant'
```

In the `os` module, `os.path` is always available. This is not always the case. In some cases, you must import a submodule directly using a dotted import notation. (In the case of `os.path`, this is not necessary, but it will serve for illustration:

In [114]:
```
import os.path
```

The `math` module

Although simple arithmetic operations are supported by Python's syntax, whenever you need to perform more complex math, you'll need to `import` the `math` module:

In [115]:
```
import math
help(math)
```

```
Help on built-in module math:

NAME
    math

FILE
    (built-in)

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(...)
        acosh(x)

        Return the hyperbolic arc cosine (measured in radians) of x.

    asin(...)
        asin(x)

        Return the arc sine (measured in radians) of x.

    asinh(...)
        asinh(x)

        Return the hyperbolic arc sine (measured in radians) of x.

    atan(...)
        atan(x)

        Return the arc tangent (measured in radians) of x.

    atan2(...)
        atan2(y, x)

        Return the arc tangent (measured in radians) of y/x.
        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(...)
        atanh(x)

        Return the hyperbolic arc tangent (measured in radians) of x.

    ceil(...)
        ceil(x)

        Return the ceiling of x as a float.
        This is the smallest integral value >= x.

    copysign(...)
        copysign(x, y)

        Return x with the sign of y.

    cos(...)
        cos(x)

        Return the cosine of x (measured in radians)
```

In [116]:
```
math.sqrt(2)
```

Out[116]:
```
1.4142135623730951
```

In [117]:
```
math.pi
```

Out[117]:
```
3.141592653589793
```

In [118]:
```
math.sin(math.pi / 4)
```

Out[118]:
```
0.7071067811865475
```

Exercises

- Create a python script that prints out its command-line arguments
- Update `sys.path` in a Python script to be the empty list. What happens when you try to `import time`?
- Create a Python script that prints out its own absolute path when run using `sys.argv` and `os.path.abspath`
- Calculate the value of e raised to the (j * pi) power

The `time` and `datetime` modules

Working with dates and times in Python is performed using these two modules. The `time` module contains lower-level C-like timestamp manipulation functions (similar to what you would find in <`time.h`>). `datetime` contains higher-level objects for dealing with datetime components:

In [119]:
```
import time
time.time()
```

Out[119]:
```
1349368421.4513
```

In [120]:
```
time.asctime()
```

Out[120]:
```
'Fri Oct  5 00:33:41 2012'
```

In [121]:
```
time.ctime()
```

Out[121]:
```
'Fri Oct  5 00:33:41 2012'
```

In [122]:
```
time.gmtime()
```

Out[122]:
```
time.struct_time(tm_year=2012, tm_mon=10, tm_mday=4, tm_hour=16, tm_min=33, tm_sec=41, tm_wday=
```

In [123]:
```
time.mktime(time.gmtime())
```

Out[123]:
```
1349339621.0
```

In [124]:
```
time.localtime()
```

Out[124]:
```
time.struct_time(tm_year=2012, tm_mon=10, tm_mday=5, tm_hour=0, tm_min=33, tm_sec=41, tm_wday=
```

In [125]:
```
time.mktime(time.localtime())
```

Out[125]:
```
1349368421.0
```

In [126]:
```
time.sleep(0.1)
```

In [127]:
```
import datetime
datetime.datetime.now()
```

Out[127]:
```
datetime.datetime(2012, 10, 5, 0, 33, 41, 665122)
```

In [128]:
```
datetime.datetime.utcnow()
```

Out[128]:
```
datetime.datetime(2012, 10, 4, 16, 33, 41, 672292)
```

In [129]:
```
now = datetime.datetime.utcnow()
print repr(now.date())
print repr(now.time())
```

```
datetime.date(2012, 10, 4)
datetime.time(16, 33, 41, 679374)
```

In [130]:
```
now.month
```

Out[130]:
```
10
```

In [131]:
```
now.ctime()
```

Out[131]:
```
'Thu Oct  4 16:33:41 2012'
```

In [132]:
```
now.strftime('%Y-%m-%d')
```

Out[132]:
```
'2012-10-04'
```

In [133]:
```
datetime.datetime.strptime('2012-10-05', '%Y-%m-%d')
```

Out[133]:
```
datetime.datetime(2012, 10, 5, 0, 0)
```

In [134]:
```
now.timetuple()
```

Out[134]:
```
time.struct_time(tm_year=2012, tm_mon=10, tm_mday=4, tm_hour=16, tm_min=33, tm_sec=41, tm_wday=
```

In [135]:
```
time.mktime(now.timetuple())
```

Out[135]:
```
1349339621.0
```

In [136]:
```
datetime.date.today()
```

Out[136]:
```
datetime.date(2012, 10, 5)
```

In [137]:
```
datetime.date.min
```

Out[137]:
```
datetime.date(1, 1, 1)
```

In [138]:
```
datetime.date.max
```

Out[138]:
```
datetime.date(9999, 12, 31)
```

In [139]:
```
datetime.time.min
```

Out[139]:
```
datetime.time(0, 0)
```

In [140]:
```
datetime.time.max
```

Out[140]:
```
datetime.time(23, 59, 59, 999999)
```

In [141]:
```
local_now = datetime.datetime.now()
utc_now = datetime.datetime.utcnow()
difference = utc_now - local_now
difference
```

Out[141]:
```
datetime.timedelta(-1, 57600, 136)
```

Files and `StringIO`

We've touched a bit on files (`sys.stdin`, etc.) but not much. Files are opened using the `open` builtin:

In [142]:
```
fp = open('Using Modules.ipynb')
```

In [143]:
```
fp.read(100)
```

Out[143]:
```
'{\n "metadata": {\n  "name": "Using Modules"\n }, \n "nbformat": 2, \n "worksheets": [\n  {\n
```

In [144]:
```
fp.seek(10)
fp.read(100)
```

Out[144]:
```
'ta": {\n  "name": "Using Modules"\n }, \n "nbformat": 2, \n "worksheets": [\n  {\n   "cells":
```

We can also treat a file as a sequence of lines:

In [145]:
```
fp.seek(0)
num_lines = 0
for line in fp:
    num_lines += 1
num_lines
```

Out[145]:
```
1856
```

Many places where we might want to use a file, it's actually more convenient to use a string. In those cases, we can create a *file-like object* using the `StringIO` module:

In [146]:
```
import StringIO
fp = StringIO.StringIO('This is a file-like object')
```

In [147]:
```
fp.read()
```

Out[147]:
```
'This is a file-like object'
```

In [148]:
```
fp.seek(4)
fp.read(10)
```

Out[148]:
```
' is a file'
```

In [149]:
```
fp.tell()
```

Out[149]:
```
14
```

We can also write to file-like objects:

In [150]:
```
fp = StringIO.StringIO()
fp.write('Hello, there')
```

In [151]:
```
fp.seek(0)
fp.read()
```

Out[151]:
```
'Hello, there'
```

We can also get the underlying buffer of the object using `getvalue()`:

In [152]:
```
fp.getvalue()
```

Out[152]:
```
'Hello, there'
```

Debugging using `pdb`

We can enter an interactive debugger from a Python file by importing the `pdb` module and setting a breakpoint:

```
import pdb
pdb.set_trace()
```

Exercises

- Write a script that prints the current value of `time.time()` every second
- Update the script to print the value of `datetime.datetime.now()`
- Update to print the value of `datetime.datetime.utcnow()`
- Write a function to convert from a datetime object (as in `datetime.datetime.utcnow()` into a "seconds since the epoch" timestamp (as in `time.time()`)
- Create a `StringIO` object that contains several lines, separated by '\n' characters. Pass that object to your function that prints a file with line numbers.
- Add the following line to your file printing function at the beginning: `import pdb; pdb.set_trace()`. Step through the execution of the function using 'n'. (You can also (c)ontinue running the program to exit the debugger.)
- `import hashlib`. Use `dir()` to determine the contents of the hashlib module. Which hashing modules are available?

String "Interpolation"

In [2]:
```
'My favorite number is %d' % 42
```

Out[2]: 'My favorite number is 42'

In [3]:
```
'My favorite number is %(num)d' % { 'num': 42 }
```

Out[3]: 'My favorite number is 42'

In [4]:
```
'The coordinates are (%d,%d)' % (5,6)
```

Out[4]: 'The coordinates are (5,6)'

In [5]:
```
'Hello, %s' % 'World'
```

Out[5]: 'Hello, World'

In [6]:
```
'Hello, %s' % 42 # calls str() on arguments
```

Out[6]: 'Hello, 42'

In [7]:
```
'Hello, %r' % 'World' # calls repr() on arguments
```

Out[7]: "Hello, 'World'"

In [8]:
```
import math
'Pi, to 3 decimals, is %.3f' % math.pi
```

Out[8]: 'Pi, to 3 decimals, is 3.142'

In [15]:
```
'Padding with spaces: "%20d"' % 42
```

Out[15]: 'Padding with spaces: "   42"'

In [14]:
```
'Padding with spaces: "%-20d"' % 42
```

Out[14]: 'Padding with spaces: "42   "'

In [13]:
```
'Padding with zeros: "%.20d"' % 42
```

Out[13]: 'Padding with zeros: "00000000000000000042"'

In [16]:
```
'Float padding: "%10.4f"' % math.pi
```

Out[16]: 'Float padding: 3.1416'

In [18]:
```
'Int padding: "%10.4d"' % 42
```

Out[18]: 'Int padding: "   0042"'

String methods

In [19]:
```
text = '''
The quick brown
 fox jumped
over the
    lazy
dog
'''
```

In [67]:
```
words = text.split()
words
```

Out[67]:
```
['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog']
```

In [68]:
```
' - '.join(words)
```

Out[68]:
```
'The - quick - brown - fox - jumped - over - the - lazy - dog'
```

In [21]:
```
text.splitlines()
```

Out[21]:
```
['', 'The quick brown', ' fox jumped', 'over the', '    lazy    ', 'dog']
```

In [22]:
```
text
```

Out[22]:
```
'\nThe quick brown\n fox jumped\nover the\n    lazy    \ndog\n'
```

In [23]:
```
text.strip()
```

Out[23]:
```
'The quick brown\n fox jumped\nover the\n    lazy    \ndog'
```

In [24]:
```
text.lstrip()
```

Out[24]:
```
'The quick brown\n fox jumped\nover the\n    lazy    \ndog\n'
```

In [25]:
```
text.rstrip()
```

Out[25]:
```
'\nThe quick brown\n fox jumped\nover the\n    lazy    \ndog'
```

In [30]:
```
text.title()
```

Out[30]:
```
'\nThe Quick Brown\n Fox Jumped\nOver The\n    Lazy    \nDog\n'
```

In [31]:
```
'55'.isdigit()
```

Out[31]:
```
True
```

In [32]:
```
'55a'.isdigit()
```

Out[32]:
```
False
```

In [35]:
```
'55a'.isalnum()
```

Out[35]: True

In [36]:
```
'33'.isalpha()
```

Out[36]: False

In [37]:
```
text.startswith('\n')
```

Out[37]: True

In [38]:
```
text.endswith('\n')
```

Out[38]: True

In [39]:
```
text.index('fox')
```

Out[39]: 18

In [40]:
```
text[18:]
```

Out[40]: 'fox jumped\nover the\n   lazy   \ndog\n'

In [42]:
```
text.find('gopher')
```

Out[42]: -1

In [43]:
```
text.index('gopher')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/vagrant/<ipython-input-43-bbb5e3c56822> in <module>()
----> 1 text.index('gopher')

ValueError: substring not found
```

In [44]:
```
'foo bar baz foo'.find('foo')
```

Out[44]: 0

In [46]:
```
'foo bar baz foo'.find('foo', 1)
```

Out[46]: 12

In [29]:
```
print 'Hello'.center(80)
```

```
                                      Hello
```

In [47]:
```
'foo bar baz foo'.rfind('foo')
```

Out[47]: 12

In [52]:
```
'foo, bar, baz, foo'.split(',')
```

Out[52]:
```
['foo', ' bar', ' baz', ' foo']
```

In [54]:
```
first, rest = 'foo, bar, baz, foo'.split(',', 1)
rest
```

Out[54]:
```
' bar, baz, foo'
```

In [55]:
```
rest, last = 'foo, bar, baz, foo'.rsplit(',', 1)
rest
```

Out[55]:
```
'foo, bar, baz'
```

In [56]:
```
text.lower()
```

Out[56]:
```
'\nthe quick brown\n fox jumped\nover the\n    lazy    \ndog\n'
```

In [57]:
```
text.upper()
```

Out[57]:
```
'\nTHE QUICK BROWN\n FOX JUMPED\nOVER THE\n    LAZY    \nDOG\n'
```

In [60]:
```
text.title().swapcase()
```

Out[60]:
```
'\ntHE qUICK bROWN\n fOX jUMPED\noVER tHE\n    lAZY    \ndOG\n'
```

String templates

In [61]:
```
import string
```

In [62]:
```
tmpl = string.Template('''
Dear $to,

I am intrigued by your ideas and wish to
subscribe to your newsletter.

Best regards,
$from
''')
```

In [64]:
```
dct = {'to': 'Rick', 'from': 'Stuart'}
print tmpl.substitute(dct)
```

```
Dear Rick,

I am intrigued by your ideas and wish to
subscribe to your newsletter.

Best regards,
Stuart
```

In [66]:
```
dct = { 'to': 'Rick' }
print tmpl.safe_substitute(dct)
```

```
Dear Rick,

I am intrigued by your ideas and wish to
subscribe to your newsletter.

Best regards,
$from
```

Exercises

- Write a function that will read a file and return a list of all the words in the file.
- Enhance this function to return a `dict` containing each distinct word as a key and the number of occurrences as the value.
- Write a function that takes a list of strings as a parameter and prints them, in title case, centered for an 80-column display.

Regular Expressions

In [1]:
```
import re
text = 'The quick brown fox jumped over the lazy dog'
```

In [2]:
```
match = re.search('quick', text)
match.start()
```

Out[2]: 4

In [3]:
```
match.end()
```

Out[3]: 9

In [4]:
```
match = re.match('quick', text)
print match
```

```
None
```

In [5]:
```
match = re.match('.*quick', text)
print match
```

```
<_sre.SRE_Match object at 0x247b718>
```

In [6]:
```
match = re.search('brown ([a-z]+) jumped', text)
match
```

Out[6]: <_sre.SRE_Match at 0x2486030>

In [7]:
```
match.groups()
```

Out[7]: ('fox',)

In [8]:
```
match = re.search('brown ([a-z]+)', text)
match
```

Out[8]: <_sre.SRE_Match at 0x2486120>

In [9]:
```
match.groups()
```

Out[9]: ('fox',)

In [10]:
```
match = re.search('brown ([a-z]+?)', text)
match
```

Out[10]: <_sre.SRE_Match at 0x2486198>

In [11]:
```
match.groups()
```

Out[11]: ('f',)

In [12]:
```
s = 'foo bar baz baz'
re.search('foo (.*) baz', s).groups()
```

Out[12]:
```
('bar baz',)
```

In [13]:
```
s = 'foo bar baz baz'
re.search('foo (.*?) baz', s).groups()
```

Out[13]:
```
('bar',)
```

In [14]:
```
m = re.search('brown (?P<animal>[a-z]+) jumped', text)
m.groups()
```

Out[14]:
```
('fox',)
```

In [15]:
```
m.groupdict()
```

Out[15]:
```
{'animal': 'fox'}
```

In [16]:
```
re_time = re.compile('(\\d{2}):(\\d{2}):(\\d{2})')
```

In [17]:
```
re_time.match('01:23:45').groups()
```

Out[17]:
```
('01', '23', '45')
```

You can make regex strings more readable by using 'raw strings':

In [18]:
```
re_time = re.compile(r'(\d{2}):(\d{2}):(\d{2})')
```

In [19]:
```
re_time.match('01:23:45').groups()
```

Out[19]:
```
('01', '23', '45')
```

You can also use 'verbose mode':

In [20]:
```
re_time = re.compile(r'''
(\d{2}) # hour
:
(\d{2}) # minute
:
(\d{2}) # second
''', re.VERBOSE)
```

In [21]:
```
re_time.match('01:23:45').groups()
```

Out[21]:
```
('01', '23', '45')
```

In [22]:
```
text = '''The quick
brown fox
jumped over
the lazy dog'''
```

In [23]:
```
print re.search('quick brown', text)
```

```
None
```

In [24]:
```
print re.search('quick( |\n)brown', text)
```

```
<_sre.SRE_Match object at 0x2486468>
```

In [25]:
```
print re.search('quick.brown', text)
```

```
None
```

In [26]:
```
print re.search('quick.brown', text, re.DOTALL)
```

```
<_sre.SRE_Match object at 0x247b718>
```

In [27]:
```
for m in re.finditer('(\w+)', text):
    print m.group(1)
```

```
The
quick
brown
fox
jumped
over
the
lazy
dog
```

In [28]:
```
print re.sub('quick', 'slow', text)
```

```
The slow
brown fox
jumped over
the lazy dog
```

In [29]:
```
def sub_length(match):
    return str(len(match.group(1)))

print re.sub('(\w+)', sub_length, text)
```

```
3 5
5 3
6 4
3 4 3
```

In [30]:
```
print re.sub('(\w+)', sub_length, text, 4)
```

```
3 5
5 3
jumped over
the lazy dog
```

In [31]:
```
large_text = open('Regular Expressions.ipynb').read()
```

In [32]:
```
regular_split = large_text.split()
```

In [33]:
```
regular_split[:10]
```

Out[33]:
```
['{',
 '"metadata":',
 '{',
 '"name":',
 '"Regular',
 'Expressions"',
 '},',
 '"nbformat":',
 '2,',
 '"worksheets":']
```

In [34]:
```
better_split = re.split('\W*', large_text)
better_split[:10]
```

Out[34]:
```
['',
 'metadata',
 'name',
 'Regular',
 'Expressions',
 'nbformat',
 '2',
 'worksheets',
 'cells',
 'cell_type']
```

Exercises

- Write a function that finds all integers in a file using regular expressions
- Write a function that finds all capitalized words in a file
- Write a function that replaces all instances of `'<br>'` in a file with `'<br/>'`

Functions in Python

Functions without arguments are straighforward. You can use the `def` keyword to define a function:

In [1]:
```
def myfunction():
    print 'Called myfunction'
myfunction()
```

```
Called myfunction
```

We can also define functions with arguments:

In [2]:
```
def myfunction2(a, b):
    print 'Called myfunction2(%r,%r)' % (a,b)
myfunction2('avalue', 'bvalue')
```

```
Called myfunction2('avalue','bvalue')
```

When calling a function with arguments, you may also pass the arguments *by name* rather than *positionally.* This is often useful when there are a large number of arguments and you don't want to remember the order in which they appear.

In [3]:
```
myfunction2(b='bvalue first', a='avalue second')
```

```
Called myfunction2('avalue second','bvalue first')
```

`lambda` functions

Python provides a special form of defining functions that consist of nothing more than a single expression using the `lambda` keyword:

In [4]:
```
lambda_adder = lambda a,b: a+b
lambda_adder(1, 2)
```

Out[4]: 3

Note the fact that `lambda` returns the function object itself. This form is often used when a function needs to be passed as an argument to another function, as in a callback. Also note the fact that the `return` statement is implicit.

The equivalent function defined with `def` would be the following

In [5]:
```
def lambda_adder_equiv(a,b):
    return a+b
lambda_adder_equiv(1,2)
```

Out[5]: 3

Default arguments

You can define a function with default argument values. If a value is not passed for an argument with a default value, the default will be used instead:

In [6]:
```
def myfunction3(a, b='default value'):
    print 'Called myfunction3(%r,%r)' % (a,b)
myfunction3('avalue')
```

```
Called myfunction3('avalue','default value')
```

You can, of course, override the default:

In [7]:
```
myfunction3('avalue', 'bvalue')
```

```
Called myfunction3('avalue','bvalue')
```

Variable arguments

We can define a function that takes any number of arguments using the `*args` syntax. The value of the `args` parameter below is any *positional* arguments that remain after accounting for other arguments:

In [8]:
```
def va_adder(prompt, *args):
    print 'Called va_adder(%r, *%r)' % (prompt, args)
    return sum(args)
va_adder('ThePrompt>', 1,2,3)
```

```
Called va_adder('ThePrompt>', *(1, 2, 3))
```

Out[8]:
```
6
```

Likewise, we can call a function with a `tuple` of arguments using a similar syntax:

In [9]:
```
def normal_function(a,b):
    print 'Called normal_function(%r, %r)' % (a,b)
argument_tuple = ('avalue', 'bvalue')
normal_function(*argument_tuple)
```

```
Called normal_function('avalue', 'bvalue')
```

If you want to define a function with variable *keyword* arguments, you can do that as well with the `**kwargs` syntax. In this case, the keyword arguments are passed as a `dict`:

In [10]:
```
def kw_function(**kwargs):
    print kwargs
kw_function(a='avalue', b='bvalue')
```

```
{'a': 'avalue', 'b': 'bvalue'}
```

Of course, we can also pass a dictionary as the keyword arguments of a function:

In [11]:
```
arguments = { 'a': 'avalue', 'b': 'bvalue' }
normal_function(**arguments)
```

```
Called normal_function('avalue', 'bvalue')
```

Exercise

Write a function with the signature `def log(format, *args, **kwargs):` which prints a line, formatted according to the format string. Some sample results are below:

```
>>> log('The pair is (%r,%r)', 1, 2)
The pair is (1,2)
>>> log('The value of a is %(a)r', a='foo')
```

```
The value of a is 'foo'
```

Advanced Functions

We can write a function that calls another function, even itself. When a function calls itself, this is referred to as *recursion*:

In [13]:
```
def recursive_adder(first, *rest):
    print 'Call recursive_adder(%r, *%r)' % (first, rest)
    if rest:
        return first + recursive_adder(*rest)
    else:
        return first
recursive_adder(1,2,3)
```

```
Call recursive_adder(1, *(2, 3))
Call recursive_adder(2, *(3,))
Call recursive_adder(3, *())
```

Out[13]: 6

Functions are just regular Python objects (they are so-called *first class* functions). This means that they can be passed as arguments to other functions or assigned variable names:

In [14]:
```
def doubler(a):
    return a * 2

def my_map(mapf, sequence):
    result = []
    for item in sequence:
        result.append(mapf(item))
    return result

my_map(doubler, [1,2,3])
```

Out[14]: [2, 4, 6]

As seen above, first class functions can be used to traverse data structures. Another common data structure is a tree. We can implement tree traversal functions to visit each node:

In [28]:
```
# Store the tree as nodes of (value, left, right)

mytree = ('root',
          ('child-L', (), ()),
          ('child-R',
           ('child-RL', (), ()),
           ('child-RR', (), ())))

def preorder_tree_map(function, node, level=0):
    value, left, right = node
    result = [function(level, value)]
    if left:
        result += preorder_tree_map(function, left, level+1)
    if right:
        result += preorder_tree_map(function, right, level+1)
    return result

def print_node(level, value):
    print ('    ' * level) + repr(value)
    return value

preorder_tree_map(print_node, mytree)
```

```
'root'
    'child-L'
    'child-R'
        'child-RL'
        'child-RR'
```

Out[28]: ['root', 'child-L', 'child-R', 'child-RL', 'child-RR']

In [29]:
```
def inorder_tree_map(function, node, level=0):
    value, left, right = node
    result = []
    if left:
        result += inorder_tree_map(function, left, level+1)
    result.append(function(level, value))
    if right:
        result += inorder_tree_map(function, right, level+1)
    return result

inorder_tree_map(print_node, mytree)
```

```
    'child-L'
'root'
        'child-RL'
    'child-R'
        'child-RR'
```

Out[29]: ['child-L', 'root', 'child-RL', 'child-R', 'child-RR']

Closures and lexical scoping

In [1]:
```
def make_adder(value):
    def adder(other_value):
        return value + other_value
    return adder

add5 = make_adder(5)
print add5(10)

add2 = make_adder(2)
print add2(10)
```

```
15
12
```

Exercise

- Write a function that traverses the tree above in *post*-order (recursing to the left and right children *before* running the function on the node's value itself.
- Write a version of `filter(function, sequence)` that returns the values in a sequence for which `function(item)` evaluates to `True`

Logging

Python provides a standard and configurable logging facility. You can set up the collection of loggers & handlers separately from their actual *use* in your program.

In [1]:
```
import logging
logging.basicConfig()

log = logging.getLogger()
log
```

Out[1]:
```
<logging.RootLogger at 0x25a34d0>
```

In [2]:
```
log.log(logging.CRITICAL, 'This is a critical message')
log.log(logging.FATAL, 'This is a fatal message')
```

```
CRITICAL:root:This is a critical messageCRITICAL:root:This is a fatal message
```

In [3]:
```
logging.CRITICAL, logging.ERROR, logging.WARN, logging.INFO, logging.DEBUG
```

Out[3]:
```
(50, 40, 30, 20, 10)
```

In [4]:
```
log.critical('This is critical')
```

```
CRITICAL:root:This is critical
```

In [5]:
```
log.error('This is an error')
log.warn('This is a warning')
log.info('This is info')
log.debug('This is debug')
```

```
ERROR:root:This is an errorWARNING:root:This is a warning
```

In [6]:
```
log.setLevel(logging.DEBUG)
```

In [7]:
```
log.error('This is an error')
log.warn('This is a warning')
log.info('This is info')
log.debug('This is debug')
```

```
ERROR:root:This is an errorWARNING:root:This is a warningINFO:root:This is infoDEBUG:root:This i
```

In [8]:
```
log.info('This is a message with an argument %r', 'The argument')
```

```
INFO:root:This is a message with an argument 'The argument'
```

Sub-loggers

We can configure "child loggers" of the root logger by passing a name to the `getLogger` function:

In [9]:
```
root = logging.getLogger()
mylogger = logging.getLogger('mylogger')
mylogger.setLevel(logging.INFO)
root.debug('The root logger will print debug information')
mylogger.debug('mylogger will not')
```

```
DEBUG:root:The root logger will print debug information
```

In [10]:
```
mylogger.info('Information will propagate up to other loggers')
mylogger.propagate = 0
mylogger.info('But not if we set propagate to 0')
```

```
INFO:mylogger:Information will propagate up to other loggersNo handlers could be found for logge
```

Handlers and formatters

In [11]:
```
handler = logging.StreamHandler()
mylogger.handlers = [handler]
mylogger.info('Now this is being handled by mylogger')
```

```
Now this is being handled by mylogger
```

In [12]:
```
handler.setLevel(logging.WARN)
mylogger.info('Now this is suppressed by the handler')
```

In [13]:
```
handler.setLevel(logging.INFO)
handler.formatter = logging.Formatter('%(levelname)s:%(message)s')
mylogger.info('This is a message')
```

```
INFO:This is a message
```

If we set propagate back to 1, we'll see "doubled" messages:

In [14]:
```
mylogger.propagate = 1
mylogger.info('Hello, there')
```

```
INFO:Hello, thereINFO:mylogger:Hello, there
```

Logging Configuration

In [27]:
```
import sys
import logging.config


config = {
    'version': 1,
    'loggers': {
        'root': {
            'level': logging.ERROR,
            'handlers': ['stream' ] },
        'mylogger2': {
            'level': logging.INFO,
            'handlers': [ 'stream', 'file'] } },
    'handlers': {
        'stream': {
            'class': 'logging.StreamHandler',
            'formatter': 'basic',
            'stream': sys.stdout },
        'file': {
            'class': 'logging.FileHandler',
            'formatter': 'precise',
            'filename': '/tmp/logfile.log',
            'mode': 'w' } },
    'formatters': {
        'basic': {
            'format': '%(levelname)-8s %(message)s' },
        'precise': {
            'format': '%(asctime)s %(levelname)-8s %(name)-15s %(message)s',
            'datefmt': '%Y-%m-%d %H:%M:%S' } }
}


logging.config.dictConfig(config)

root = logging.getLogger()
mylogger2 = logging.getLogger('mylogger2')

root.error('error from root')
print
mylogger2.error('error from mylogger')
print
mylogger2.info('info from mylogger')
```

```
ERROR:root:error from root
ERROR    error from mylogger
ERROR:mylogger2:error from mylogger
INFO     info from mylogger
INFO:mylogger2:info from mylogger
```

In [28]:
```
print open('/tmp/logfile.log').read()
```

```
2012-10-07 16:47:15 ERROR    mylogger2       error from mylogger
2012-10-07 16:47:15 INFO     mylogger2       info from mylogger
```

Object-Oriented Programming

Basic class operations ("glorified struct")

In [1]:
```
class MyClass(object):
    a = 5
    b = 6
print MyClass
print MyClass.a
```

```
<class '__main__.MyClass'>
5
```

In [3]:
```
obj = MyClass()
print obj.a
print obj.b
```

```
5
6
```

In [4]:
```
obj.a = 'new value for a'
print obj.a
```

```
new value for a
```

In [5]:
```
print MyClass.a
```

```
5
```

Basic methods

In [9]:
```
class MyClass(object):
    def say(self, something):
        print 'MyClass says', something
```

In [11]:
```
myobj = MyClass()
myobj.say('Hello')
```

```
MyClass says Hello
```

In [14]:
```
# Classes can have a constructor

class MyClass(object):
    def __init__(self, a, b):
        self.a = a
        self.b = b

obj = MyClass('avalue', 'bvalue')
print obj.a
```

```
avalue
```

Method access and visibility

By convention, a single leading underscore indicates that a method or variable is "protected" and should not be modified outside the class:

In [16]:
```
class MyClass(object):
    def __init__(self, a, b):
        self._a = a
        self._b = b
    def value(self):
        return self._a, self._b

obj = MyClass('avalue', 'bvalue')
print obj.value()
print obj._a
```

```
('avalue', 'bvalue')
avalue
```

To "enforce" private methods / variables, we can use a *double* leading underscore:

In [17]:
```
class MyClass(object):
    def __init__(self, a, b):
        self.__a = a
        self.__b = b
    def value(self):
        return self.__a, self.__b

obj = MyClass('avalue', 'bvalue')
print obj.value()
```

```
('avalue', 'bvalue')
```

In [18]:
```
print obj.__a
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/vagrant/<ipython-input-18-383b7dc3f3c4> in <module>()
----> 1 print obj.__a

AttributeError: 'MyClass' object has no attribute '__a'
```

In [21]:
```
print obj._MyClass__a
```

```
avalue
```

Exercise

Write a class to manage a telephone directory. The directory should be stored in a dict as an instance variable. The class should have methods `add_number(name, number)`, `remove_number(name)`, and `lookup_number(name)`.

Object Oriented Programming (part 2)

Inheritance

In [64]:
```
class Animal(object):
    def __init__(self, name):
        self._name = name
    def say(self, message):
        print '%s the animal says %s' % (self._name, message)
    def get_number_of_legs(self):
        raise NotImplementedError, 'get_number_of_legs'

class Cat(Animal):
    def __init__(self, name='Felix'):
        Animal.__init__(self, name)
    def say(self, message):
        print '%s the cat meows %s' % (self._name, message)
    def get_number_of_legs(self):
        return 4

class Dog(Animal):
    def __init__(self, name='Fido'):
        super(Dog, self).__init__(name)
    def say(self, message):
        print '%s the dog barks %s' % (self._name, message)
    def get_number_of_legs(self):
        return 4

class Monkey(Animal):
    def __init__(self, name='George'):
        Animal.__init__(self, name)
    def say(self, message):
        print '%s the monkey says %s' % (self._name, message)
    def get_number_of_legs(self):
        return 2
```

In [65]:
```
animal = Animal('Generic')
animal.say('hello')
```

```
Generic the animal says hello
```

In [66]:
```
print animal.get_number_of_legs()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
/vagrant/<ipython-input-66-3623e2c96566> in <module>()
----> 1 print animal.get_number_of_legs()

/vagrant/<ipython-input-64-798984796887> in get_number_of_legs(self)
      5         print '%s the animal says %s' % (self._name, message)
      6     def get_number_of_legs(self):
----> 7         raise NotImplementedError, 'get_number_of_legs'
      8
      9 class Cat(Animal):

NotImplementedError: get_number_of_legs
```

In [67]:
```
animal = Cat()
animal.say('hello')
```

```
Felix the cat meows hello
```

In [68]:
```
animal = Dog()
animal.say('hello')
print animal.get_number_of_legs()
```

```
Fido the dog barks hello
4
```

In [69]:
```
animal = Monkey()
animal.say('I have %s legs' % animal.get_number_of_legs())
```

```
George the monkey says I have 2 legs
```

In [70]:
```
isinstance(animal, Monkey)
```

Out[70]: True

In [71]:
```
isinstance(animal, Cat)
```

Out[71]: False

In [72]:
```
isinstance(animal, Animal)
```

Out[72]: True

In [73]:
```
issubclass(Cat, Animal)
```

Out[73]: True

In [74]:
```
class MonkeyDog(Monkey, Dog):
    pass

x = MonkeyDog('What is this thing?!')
print x.say('hello?')
```

```
What is this thing?! the monkey says hello?
None
```

In [75]:
```
print MonkeyDog.mro()
```

```
[<class '__main__.MonkeyDog'>, <class '__main__.Monkey'>, <class '__main__.Dog'>, <class '__main
```

Magic methods

In [76]:
```
print animal
```

```
<__main__.Monkey object at 0x2847590>
```

In [77]:
```
print str(animal)
```

```
<__main__.Monkey object at 0x2847590>
```

In [78]:
```
class Animal(object):
    def __init__(self, name):
        self._name = name
    def __str__(self):
        return '<Animal %s>' % self._name
```

In [79]:
```
animal = Animal('generic')
print animal
```

```
<Animal generic>
```

In [80]:
```
class Animal(object):
    def __init__(self, name):
        self._name = name
    def __str__(self):
        return '<Animal %s>' % self._name
    def __repr__(self):
        return 'Animal(%r)' % self._name
```

In [81]:
```
Animal('with repr')
```

Out[81]:
```
Animal('with repr')
```

In [82]:
```
print Animal('with repr')
```

```
<Animal with repr>
```

Override attribute access

In [83]:
```
class MyClass(object):
    def __init__(self):
        self.a = 'avalue'
    def __getattr__(self, name):
        print 'Trying to get %s' % name
        return None

x = MyClass()
print x.a
print x.unknown_attribute
```

```
avalue
Trying to get unknown_attribute
None
```

In [84]:
```
class MyClass(object):
    a = 0
    def __setattr__(self, name, value):
        print 'Set %s <= %s' % (name, value)

x = MyClass()
x.a = 'avalue'
print 'x.a is still %s' % x.a
```

```
Set a <= avalue
x.a is still 0
```

In [85]:
```
class MyClass(object):
    def __init__(self):
        self.a = 'avalue'
    def __getattribute__(self, name):
        print 'Trying to get %s' % name
        return None

x = MyClass()
print x.a
print x.unkown_attribute
```

```
Trying to get a
None
Trying to get unkown_attribute
None
```

Override Container Methods

In [86]:
```
class DefaultDict(object):
    def __init__(self, default):
        self._data = {}
        self._default = default
    def __getitem__(self, key):
        try:
            return self._data[key]
        except KeyError:
            return self._default()
    def __setitem__(self, key, value):
        self._data[key] = value
    def __delitem__(self, key):
        del self._data[key]
    def __contains__(self, key):
        return key in self._data
    def __repr__(self):
        return '<DefaultDict %r>' % self._data

mydict = DefaultDict(lambda:5)
mydict[1] = 1
mydict[2] = 2
print mydict
print 2 in mydict
```

```
<DefaultDict {1: 1, 2: 2}>
True
```

In [87]:
```
print mydict[5]
print mydict
```

```
5
<DefaultDict {1: 1, 2: 2}>
```

Other Magic Methods

- Comparison override (`__lt__`, `__gt__`, `__le__`, `__ge__`, `__eq__`, `__ne__`)
- Emulating numeric types (`__add__`, `__sub__`, etc.)
- ... more ... (full list at http://docs.python.org/reference/datamodel.html#special-method-names)

Exercises

- Update your phone directory to support looking up a number using the `[]` operator
- Create two phone directories, one which throws exceptions when looking up phone numbers, and a subclass that always returns the same number for unknown phone numbers.

Decorators

Basic decorator syntax

Python *decorators* allow us to modify function and class definitions with a special syntax.

In [3]:
```
def log_function_call(function):
    def wrapper(*args, **kwargs):
        print 'Calling %s(*%r, **%r)' % (function, args, kwargs)
        return function(*args, **kwargs)
    print 'returning wrapped %s' % function
    return wrapper

def myfunction(a, b):
    print 'myfunction(%r, %r)' % (a,b)

myfunction = log_function_call(myfunction)
```

```
returning wrapped <function myfunction at 0x27eaf50>
```

In [4]:
```
myfunction('avalue', 'bvalue')
```

```
Calling <function myfunction at 0x27eaf50>(*('avalue', 'bvalue'), **{})
myfunction('avalue', 'bvalue')
```

A nicer syntax for this uses the `@` sign:

In [6]:
```
@log_function_call
def myfunction(a,b):
    print 'myfunction(%r, %r)' % (a,b)

myfunction('avalue', 'bvalue')
```

```
returning wrapped <function myfunction at 0x27eaed8>
Calling <function myfunction at 0x27eaed8>(*('avalue', 'bvalue'), **{})
myfunction('avalue', 'bvalue')
```

We can also decorate class definitions:

In [8]:
```
def add_myproperty(cls):
    cls.myproperty = 'Magically added by decorator'
    return cls

@add_myproperty
class MyClass(object):
    def __init__(self, a, b):
        self._a = a
        self._b = b
    def __repr__(self):
        return 'MyClass(%r, %r)' % (a,b)

MyClass.myproperty
```

Out[8]:
```
'Magically added by decorator'
```

Useful decorators

In [10]:
```
class MyClass(object):
    @property
    def myproperty(self):
        print 'Calling myproperty'
        return 'myvalue'

x = MyClass()
print x.myproperty
```

```
Calling myproperty
myvalue
```

In [13]:
```
class MyClass(object):

    def __init__(self):
        self._value = None

    @property
    def myproperty(self):
        print 'Getting myproperty'
        return self._value

    @myproperty.setter
    def myproperty(self, value):
        print 'Setting myproperty'
        self._value = value


x = MyClass()
print x.myproperty
print

x.myproperty = 5
print x.myproperty
```

```
Getting myproperty
None

Setting myproperty
Getting myproperty
5
```

In [14]:
```
class MyClass(object):

    def do_something_with_instance(self):
        print 'Instance method on', self

    @classmethod
    def do_something_with_class(cls):
        print 'Class method on', cls

    @staticmethod
    def do_something_without_either():
        print 'Static method'

x = MyClass()
x.do_something_with_instance()
```

```
Instance method on <__main__.MyClass object at 0x27f3dd0>
```

In [15]:
```
x.do_something_with_class()
```

```
Class method on <class '__main__.MyClass'>
```

In [16]:
```
MyClass.do_something_with_class()
```

```
Class method on <class '__main__.MyClass'>
```

In [17]:
```
x.do_something_without_either()
```

```
Static method
```

In [18]:
```
MyClass.do_something_without_either()
```

```
Static method
```

Building your own decorators

In [25]:
```
def log_function_call(function):
    def wrapper(*args, **kwargs):
        print 'Calling %s(*%r, **%r)' % (function, args, kwargs)
        return function(*args, **kwargs)
    print 'returning wrapped %s' % function
    return wrapper

@log_function_call
def myfunction(a, b):
    print 'myfunction(%r, %r)' % (a,b)

myfunction(1,2)
```

```
returning wrapped <function myfunction at 0x27f4d70>
Calling <function myfunction at 0x27f4d70>(*(1, 2), **{})
myfunction(1, 2)
```

In [27]:
```
def log_function_call(message):
    def decorator(function):
        def wrapper(*args, **kwargs):
            print '%s: %s(*%r, **%r)' % (message, function, args, kwargs)
            return function(*args, **kwargs)
        print 'returning wrapped %s' % function
        return wrapper
    print 'returning decorator(%r)' % message
    return decorator

@log_function_call('log1')
def myfunction(a, b):
    print 'myfunction(%r, %r)' % (a,b)

myfunction(1,2)
```

```
returning decorator('log1')
returning wrapped <function myfunction at 0x27f4d70>
log1: <function myfunction at 0x27f4d70>(*(1, 2), **{})
myfunction(1, 2)
```

To simplify things a bit, we can also use the magic `__call__` method to define a decorator that takes arguments:

In [28]:
```
class log_function_call(object):
    def __init__(self, message):
        self._message = message
    def __call__(self, function):
        def wrapper(*args, **kwargs):
            print '%s: %s(*%r, **%r)' % (
                self._message, function, args, kwargs)
            return function(*args, **kwargs)
        return wrapper

@log_function_call('log1')
def myfunction(a, b):
    print 'myfunction(%r, %r)' % (a,b)

myfunction(1,2)
```

```
log1: <function myfunction at 0x27eaed8>(*(1, 2), **{})
myfunction(1, 2)
```

One useful decorator to build is one that *memoizes* function results:

In [38]:
```
def memoize(function):
    cache = {}
    def wrapper(*args, **kwargs):
        cache_key = (args, tuple(sorted(kwargs.items())))
        if cache_key in cache:
            print '-- return cached value for', cache_key
            return cache[cache_key]
        result = function(*args, **kwargs)
        cache[cache_key] = result
        return result
    return wrapper

@memoize
def my_function(a, b):
    print 'Calling my_function(%r,%r)' % (a,b)

my_function(1,2)
my_function(1,2)
my_function(1,2)
my_function(3,4)
my_function(5,6)
```

```
Calling my_function(1,2)
-- return cached value for ((1, 2), ())
-- return cached value for ((1, 2), ())
Calling my_function(3,4)
Calling my_function(5,6)
```

Exercises

- Write a class that uses `@property` to provide read-only access to an underlying "private" attribute
- Write a decorator that takes a logger and logs all entries/exits of a function
- Write a decorator that opens a file at the beginning of a function and closes it at the end, passing the opened file as the first argument of the inner function.

Generators and Iterators

Building your own generators with `yield`

In [42]:
```
def counter(start, end):
    current = start
    while current < end:
        yield current
        current += 1
```

In [43]:
```
counter(1, 10)
```

Out[43]:
```
<generator object counter at 0x1c61500>
```

In [44]:
```
x = counter(1,10)
x.next()
```

Out[44]:
```
1
```

In [45]:
```
x.next()
```

Out[45]:
```
2
```

In [46]:
```
x.next()
```

Out[46]:
```
3
```

In [47]:
```
x = counter(1,10)
list(x)
```

Out[47]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

`yield` can also be used as a function, along with the `send()` method

In [48]:
```
def accumulator(start=0):
    current = start
    while True:
        current += yield(current)
```

In [49]:
```
x = accumulator()
x.next()
```

Out[49]:
```
0
```

In [50]:
```
x.send(1)
```

Out[50]:
```
1
```

In [51]:
```
x.send(1)
```

Out[51]:
```
2
```

In [52]:
```
x.send(10)
```

Out[52]: 12

The iterator protocol

What does `for x in sequence:` *really* do?

In [53]:
```
seq = range(4)
for x in seq: print x
```

```
0
1
2
3
```

In [54]:
```
iter_seq = iter(seq)
print iter_seq
```

```
<listiterator object at 0x1c95ad0>
```

In [55]:
```
iter_seq = iter(seq)
try:
    while True:
        x = iter_seq.next()
        print x
except StopIteration:
    pass
```

```
0
1
2
3
```

Generators are their own iterators:

In [56]:
```
print counter(0, 4)
print iter(counter(0, 4))
```

```
<generator object counter at 0x1c61780>
<generator object counter at 0x1c61780>
```

In [57]:
```
for item in counter(0, 4): print item
```

```
0
1
2
3
```

We can also define our own iterator classes (though generators are usually more readable):

In [58]:
```
class Counter(object):
    def __init__(self, start, end):
        self._start = start
        self._end = end
    def __iter__(self):
        return CounterIterator(self._start, self._end)

class CounterIterator(object):
    def __init__(self, start, end):
        self._cur = ctr._start
        self._end = ctr._end
    def next(self):
        result = self._cur
        self._cur += 1
        if result < self._end:
            return result
        else:
            raise StopIteration

ctr = Counter(0, 5)
print list(ctr)
```

```
[0, 1, 2, 3, 4]
```

Loop comprehensions

In [59]:
```
[ x*2 for x in range(4) ]
```

Out[59]:
```
[0, 2, 4, 6]
```

In [67]:
```
lst = [ ]
for x in range(4):
    lst.append(x*2)
lst
```

Out[67]:
```
[0, 2, 4, 6]
```

In [60]:
```
[ (x,y) for x in range(4) for y in range(4) ]
```

Out[60]:
```
[(0, 0),
 (0, 1),
 (0, 2),
 (0, 3),
 (1, 0),
 (1, 1),
 (1, 2),
 (1, 3),
 (2, 0),
 (2, 1),
 (2, 2),
 (2, 3),
 (3, 0),
 (3, 1),
 (3, 2),
 (3, 3)]
```

In [62]:
```
[ [ (r,c) for c in range(4) ]
  for r in range(4) ]
```

Out[62]:
```
[[(0, 0), (0, 1), (0, 2), (0, 3)],
 [(1, 0), (1, 1), (1, 2), (1, 3)],
 [(2, 0), (2, 1), (2, 2), (2, 3)],
 [(3, 0), (3, 1), (3, 2), (3, 3)]]
```

In [63]:
```
[ x for x in range(10) if x % 2 == 0 ]
```

Out[63]:
```
[0, 2, 4, 6, 8]
```

In [66]:
```
[ x * 4
  for x in range(10)
  if x % 2 == 0
  if x % 3 == 0 ]
```

Out[66]:
```
[0, 24]
```

Generator expressions

In [68]:
```
[ x for x in range(10) if x % 2 == 0 ]
```

Out[68]:
```
[0, 2, 4, 6, 8]
```

In [69]:
```
( x for x in range(10) if x % 2 == 0 )
```

Out[69]:
```
<generator object <genexpr> at 0x1c61730>
```

In [70]:
```
gen = ( x for x in range(10) if x % 2 == 0 )
```

In [71]:
```
gen.next()
```

Out[71]:
```
0
```

In [72]:
```
gen.next()
```

Out[72]:
```
2
```

In [79]:
```
list(gen)
```

Out[79]:
```
[6, 8]
```

Exercises

- Write a generator that will yield the nodes of a tree and their depth in post-order
- Write a loop that uses that generator to *print* the nodes of a tree in post-order

Context Managers

In [2]:
```
with open('/etc/hosts') as fp:
    print fp.read()
print fp
```

```
127.0.0.1       localhost
127.0.1.1       precise64

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

<closed file '/etc/hosts', mode 'r' at 0x1882930>
```

In [5]:
```
try:
    with open('/etc/hosts') as fp:
        raise KeyError
        print fp.read()
except KeyError:
    print 'handle keyerror'

print fp
```

```
handle keyerror
<closed file '/etc/hosts', mode 'r' at 0x1882930>
```

In [7]:
```
with open('/etc/hosts') as fp_i, open('/tmp/hosts', 'w') as fp_o:
    fp_o.write(fp_i.read())
```

In [8]:
```
with open('/tmp/hosts') as fp:
    print fp.read()
```

```
127.0.0.1       localhost
127.0.1.1       precise64

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Context manager protocol

In [16]:
```
class CM(object):
    def __enter__(self):
        print 'Entering CM'
        return self
    def __exit__(self, ex_type, ex_val, ex_tb):
        print 'Exiting CM'
        if ex_type == KeyError:
            # Re-raise same exception
            return False
        # Don't re-raise
        print 'Swallowing %s inside CM' % ex_type
        return True
```

In [20]:
```
with CM() as cm:
    print 'Inside with statement', cm
```

```
Entering CM
Inside with statement <__main__.CM object at 0x192f8d0>
Exiting CM
Swallowing None inside CM
```

In [21]:
```
try:
    with CM():
        print 'About to raise KeyError'
        raise KeyError
except KeyError:
    print 'Catching KeyError outside CM'
```

```
Entering CM
About to raise KeyError
Exiting CM
Catching KeyError outside CM
```

In [22]:
```
with CM():
    print 'About to raise ValueError'
    raise ValueError
```

```
Entering CM
About to raise ValueError
Exiting CM
Swallowing <type 'exceptions.ValueError'> inside CM
```

Exercises

- Write a context manager that logs the entry and exit of a block of code (similar to the decorator before)
- Write a context manager that prints out balanced XML nodes. Use the test code below.

Test code:

```
with node('html'):
    with node('body'):
        with node('h1'):
            print 'Page Title'
```

You should see the following result:

```
<html>
<body>
<h1>
Page Title
</h1>
</body>
</html>
```

Contextlib

In [23]:
```
import contextlib
```

In [25]:
```
@contextlib.contextmanager
def so_much_easier():
    print 'Entering block'
    try:
        yield
        print 'Exiting block cleanly'
    except:
        print 'Exiting block with exception'
```

In [26]:
```
with so_much_easier():
    print 'Inside block'
```

```
Entering block
Inside block
Exiting block cleanly
```

In [28]:
```
with so_much_easier():
    print 'Raising ValueError'
    raise ValueError
```

```
Entering block
Raising ValueError
Exiting block with exception
```

`contextlib` also provides a facility to support the `with` statement with context manager-like objects that don't actually support the protocol, but *do* have a `close()` method:

In [29]:
```
class MyClass(object):
    def __init__(self):
        print 'Perform some resource acquisition'
    def close(self):
        print 'Close the resource'
```

In [30]:
```
with contextlib.closing(MyClass()) as myobj:
    print 'myobj is', myobj
```

```
Perform some resource acquisition
myobj is <__main__.MyClass object at 0x19c4450>
Close the resource
```

In [31]:
```
try:
    with contextlib.closing(MyClass()) as myobj:
        print 'raising ValueError'
        raise ValueError
except:
    print 'handling exception'
```

```
Perform some resource acquisition
raising ValueError
Close the resource
handling exception
```

Exercises

- Update your context managers from the previous exercise to use the `@contextmanager` decorator

Subprocess

In [1]:
```
import subprocess
```

Calling subprocesses

There are several convenience methods for calling subprocesses, either using or discarding their output:

- call()
- check_call()
- check_output()

In [2]:
```
subprocess.call('ls')
```

Out[2]:
```
0
```

In [3]:
```
subprocess.check_call('ls')
```

Out[3]:
```
0
```

In [4]:
```
subprocess.check_output('ls')
```

Out[4]:
```
'01-BasicPythonSyntax\n01-BasicPythonSyntax_files\n01-BasicPythonSyntax.html\n01-BasicPythonSyntax.ipynb\n02-
Builtins\n02-Builtins_files\n02-Builtins.html\n02-Builtins.ipynb\n03-FileIO\n03-FileIO_files\n03-FileIO.html\n03-
FileIO.ipynb\n04-UsingModules\n04-UsingModules_files\n04-UsingModules.html\n04-UsingModules.ipynb\n05-Strings\n05-
Strings_files\n05-Strings.html\n05-Strings.ipynb\n06a-Packages\n06-Regex\n06-Regex_files\n06-Regex.html\n06-
Regex.ipynb\n07-Functions\n07-Functions_files\n07-Functions.html\n07-Functions.ipynb\n08-AdvancedFunctions\n08-
AdvancedFunctions_files\n08-AdvancedFunctions.html\n08-AdvancedFunctions.ipynb\n09-Logging\n09-Logging_files\n09-
Logging.html\n09-Logging.ipynb\n10-OOP1\n10-OOP1_files\n10-OOP1.html\n10-OOP1.ipynb\n11-OOP2\n11-OOP2_files\n11-
OOP2.html\n11-OOP2.ipynb\n12-Decorators\n12-Decorators_files\n12-Decorators.html\n12-Decorators.ipynb\n13-
Generators\n13-GeneratorsAndIterators_files\n13-GeneratorsAndIterators.html\n13-GeneratorsAndIterators.ipynb\n14-
ContextManagers\n14-ContextManagers_files\n14-ContextManagers.html\n14-ContextManagers.ipynb\n15-Threading\n16-
Multiprocessing\n17-Subprocess\n17-Subprocess.ipynb\n18-Virtualenv\n19-Testing\n20-
MoreModules.ipynb\nfabfile.py\nfabfile.pyc\nFastTrackToPython.pdf\nindex.md\nVagrantfile\n'
```

In [5]:
```
subprocess.check_output(['ls', '-a'])
```

Out[5]:
```
'.\n..\n01-BasicPythonSyntax\n01-BasicPythonSyntax_files\n01-BasicPythonSyntax.html\n01-
BasicPythonSyntax.ipynb\n02-Builtins\n02-Builtins_files\n02-Builtins.html\n02-Builtins.ipynb\n03-FileIO\n03-
FileIO_files\n03-FileIO.html\n03-FileIO.ipynb\n04-UsingModules\n04-UsingModules_files\n04-UsingModules.html\n04-
UsingModules.ipynb\n05-Strings\n05-Strings_files\n05-Strings.html\n05-Strings.ipynb\n06a-Packages\n06-Regex\n06-
Regex_files\n06-Regex.html\n06-Regex.ipynb\n07-Functions\n07-Functions_files\n07-Functions.html\n07-
Functions.ipynb\n08-AdvancedFunctions\n08-AdvancedFunctions_files\n08-AdvancedFunctions.html\n08-
AdvancedFunctions.ipynb\n09-Logging\n09-Logging_files\n09-Logging.html\n09-Logging.ipynb\n10-OOP1\n10-
OOP1_files\n10-OOP1.html\n10-OOP1.ipynb\n11-OOP2\n11-OOP2_files\n11-OOP2.html\n11-OOP2.ipynb\n12-Decorators\n12-
Decorators_files\n12-Decorators.html\n12-Decorators.ipynb\n13-Generators\n13-GeneratorsAndIterators_files\n13-
GeneratorsAndIterators.html\n13-GeneratorsAndIterators.ipynb\n14-ContextManagers\n14-ContextManagers_files\n14-
ContextManagers.html\n14-ContextManagers.ipynb\n15-Threading\n16-Multiprocessing\n17-Subprocess\n17-
Subprocess.ipynb\n18-Virtualenv\n19-Testing\n20-
MoreModules.ipynb\nfabfile.py\nfabfile.pyc\nFastTrackToPython.pdf\n.git\nindex.md\n.vagrant\nVagrantfile\n'
```

In [6]:
```
subprocess.call(['ls', '/directory/does/not/exist'])
```

Out[6]:
```
2
```

In [7]:
```
try:
    subprocess.check_output(['ls', '-a', '/directory/does/not/exist'])
except subprocess.CalledProcessError, error:
    print 'Exception raised: %s' % error
```

```
Exception raised: Command '['ls', '-a', '/directory/does/not/exist']' returned non-zero exit status 2
```

In [8]:
```
try:
    subprocess.check_call(['ls', '-a', '/directory/does/not/exist'])
except subprocess.CalledProcessError, error:
    print 'Exception raised: %s' % error
```

```
Exception raised: Command '['ls', '-a', '/directory/does/not/exist']' returned non-zero exit status 2
```

In [9]:
```
subprocess.check_output('ls -a', shell=True)
```

Out[9]:
```
'.\n..\n01-BasicPythonSyntax\n01-BasicPythonSyntax_files\n01-BasicPythonSyntax.html\n01-
BasicPythonSyntax.ipynb\n02-Builtins\n02-Builtins_files\n02-Builtins.html\n02-Builtins.ipynb\n03-FileIO\n03-
FileIO_files\n03-FileIO.html\n03-FileIO.ipynb\n04-UsingModules\n04-UsingModules_files\n04-UsingModules.html\n04-
UsingModules.ipynb\n05-Strings\n05-Strings_files\n05-Strings.html\n05-Strings.ipynb\n06a-Packages\n06-Regex\n06-
Regex_files\n06-Regex.html\n06-Regex.ipynb\n07-Functions\n07-Functions_files\n07-Functions.html\n07-
Functions.ipynb\n08-AdvancedFunctions\n08-AdvancedFunctions_files\n08-AdvancedFunctions.html\n08-
AdvancedFunctions.ipynb\n09-Logging\n09-Logging_files\n09-Logging.html\n09-Logging.ipynb\n10-OOP1\n10-
OOP1_files\n10-OOP1.html\n10-OOP1.ipynb\n11-OOP2\n11-OOP2_files\n11-OOP2.html\n11-OOP2.ipynb\n12-Decorators\n12-
Decorators_files\n12-Decorators.html\n12-Decorators.ipynb\n13-Generators\n13-GeneratorsAndIterators_files\n13-
GeneratorsAndIterators.html\n13-GeneratorsAndIterators.ipynb\n14-ContextManagers\n14-ContextManagers_files\n14-
ContextManagers.html\n14-ContextManagers.ipynb\n15-Threading\n16-Multiprocessing\n17-Subprocess\n17-
Subprocess.ipynb\n18-Virtualenv\n19-Testing\n20-
MoreModules.ipynb\nfabfile.py\nfabfile.pyc\nFastTrackToPython.pdf\n.git\nindex.md\n.vagrant\nVagrantfile\n'
```

In [10]:
```
import shlex
command_string = 'ls -l -a'
print shlex.split(command_string)
subprocess.check_output(shlex.split(command_string))
```

```
['ls', '-l', '-a']
```

Out[10]:
```
'total 2824\ndrwxr-xr-x  1 vagrant vagrant    2482 Oct  9 20:46 .\ndrwxr-xr-x 24 root    root       4096 Oct  3
03:20 ..\ndrwxr-xr-x  1 vagrant vagrant     238 Oct  9 06:52 01-BasicPythonSyntax\ndrwxr-xr-x  1 vagrant vagrant
510 Oct  8 08:14 01-BasicPythonSyntax_files\n-rw-r--r--  1 vagrant vagrant  167970 Oct  8 08:14 01-
BasicPythonSyntax.html\n-rw-r--r--  1 vagrant vagrant   37640 Oct  7 10:58 01-BasicPythonSyntax.ipynb\ndrwxr-xr-x
1 vagrant vagrant     136 Oct  8 13:17 02-Builtins\ndrwxr-xr-x  1 vagrant vagrant     510 Oct  8 08:15 02-
Builtins_files\n-rw-r--r--  1 vagrant vagrant   73444 Oct  8 08:15 02-Builtins.html\n-rw-r--r--  1 vagrant vagrant
16244 Oct  7 10:58 02-Builtins.ipynb\ndrwxr-xr-x  1 vagrant vagrant     102 Oct  8 13:41 03-FileIO\ndrwxr-xr-x  1
vagrant vagrant     510 Oct  8 08:15 03-FileIO_files\n-rw-r--r--  1 vagrant vagrant   41620 Oct  8 08:15 03-
FileIO.html\n-rw-r--r--  1 vagrant vagrant    9576 Oct  7 10:58 03-FileIO.ipynb\ndrwxr-xr-x  1 vagrant vagrant
374 Oct  9 06:52 04-UsingModules\ndrwxr-xr-x  1 vagrant vagrant     510 Oct  8 08:15 04-UsingModules_files\n-rw-r--
r--  1 vagrant vagrant  137356 Oct  8 08:15 04-UsingModules.html\n-rw-r--r--  1 vagrant vagrant   45766 Oct  7
10:59 04-UsingModules.ipynb\ndrwxr-xr-x  1 vagrant vagrant     170 Oct  8 15:59 05-Strings\ndrwxr-xr-x  1 vagrant
vagrant     510 Oct  8 08:15 05-Strings_files\n-rw-r--r--  1 vagrant vagrant   79928 Oct  8 08:15 05-
Strings.html\n-rw-r--r--  1 vagrant vagrant   17750 Oct  7 10:59 05-Strings.ipynb\ndrwxr-xr-x  1 vagrant vagrant
238 Oct  8 18:02 06a-Packages\ndrwxr-xr-x  1 vagrant vagrant     102 Oct  8 16:26 06-Regex\ndrwxr-xr-x  1 vagrant
vagrant     510 Oct  8 08:15 06-Regex_files\n-rw-r--r--  1 vagrant vagrant   64794 Oct  8 08:15 06-Regex.html\n-rw-
r--r--  1 vagrant vagrant   12896 Oct  7 10:59 06-Regex.ipynb\ndrwxr-xr-x  1 vagrant vagrant     102 Oct  9 10:12
07-Functions\ndrwxr-xr-x  1 vagrant vagrant     510 Oct  8 08:16 07-Functions_files\n-rw-r--r--  1 vagrant vagrant
45536 Oct  8 08:16 07-Functions.html\n-rw-r--r--  1 vagrant vagrant    8062 Oct  9 08:01 07-Functions.ipynb\ndrwxr-
xr-x  1 vagrant vagrant     136 Oct  9 10:59 08-AdvancedFunctions\ndrwxr-xr-x  1 vagrant vagrant     510 Oct  8
08:16 08-AdvancedFunctions_files\n-rw-r--r--  1 vagrant vagrant   34295 Oct  8 08:16 08-AdvancedFunctions.html\n-
rw-r--r--  1 vagrant vagrant    5848 Oct  7 17:05 08-AdvancedFunctions.ipynb\ndrwxr-xr-x  1 vagrant vagrant     170
Oct  7 16:25 09-Logging\ndrwxr-xr-x  1 vagrant vagrant     510 Oct  8 08:16 09-Logging_files\n-rw-r--r--  1 vagrant
vagrant   47018 Oct  8 08:16 09-Logging.html\n-rw-r--r--  1 vagrant vagrant   10741 Oct  7 16:48 09-
Logging.ipynb\ndrwxr-xr-x  1 vagrant vagrant     102 Oct  9 19:35 10-OOP1\ndrwxr-xr-x  1 vagrant vagrant     510
Oct  8 08:16 10-OOP1_files\n-rw-r--r--  1 vagrant vagrant   35789 Oct  8 08:16 10-OOP1.html\n-rw-r--r--  1 vagrant
vagrant    6375 Oct  7 17:05 10-OOP1.ipynb\ndrwxr-xr-x  1 vagrant vagrant     136 Oct  9 14:19 11-OOP2\ndrwxr-xr-x
1 vagrant vagrant     510 Oct  8 08:17 11-OOP2_files\n-rw-r--r--  1 vagrant vagrant   72396 Oct  8 08:17 11-
OOP2.html\n-rw-r--r--  1 vagrant vagrant   14740 Oct  7 17:50 11-OOP2.ipynb\ndrwxr-xr-x  1 vagrant vagrant     170
Oct  9 15:13 12-Decorators\ndrwxr-xr-x  1 vagrant vagrant     510 Oct  8 08:17 12-Decorators_files\n-rw-r--r--  1
vagrant vagrant   64594 Oct  8 08:17 12-Decorators.html\n-rw-r--r--  1 vagrant vagrant   12311 Oct  7 18:36 12-
Decorators.ipynb\ndrwxr-xr-x  1 vagrant vagrant     102 Oct  9 16:17 13-Generators\ndrwxr-xr-x  1 vagrant vagrant
510 Oct  8 08:17 13-GeneratorsAndIterators_files\n-rw-r--r--  1 vagrant vagrant   71092 Oct  8 08:17 13-
GeneratorsAndIterators.html\n-rw-r--r--  1 vagrant vagrant   12557 Oct  7 19:06 13-
GeneratorsAndIterators.ipynb\ndrwxr-xr-x  1 vagrant vagrant     204 Oct  9 17:36 14-ContextManagers\ndrwxr-xr-x  1
vagrant vagrant     510 Oct  8 08:18 14-ContextManagers_files\n-rw-r--r--  1 vagrant vagrant   47276 Oct  8 08:18
14-ContextManagers.html\n-rw-r--r--  1 vagrant vagrant    9241 Oct  7 19:42 14-ContextManagers.ipynb\ndrwxr-xr-x  1
vagrant vagrant     544 Oct  8 07:42 15-Threading\ndrwxr-xr-x  1 vagrant vagrant     272 Oct  8 07:46 16-
Multiprocessing\ndrwxr-xr-x  1 vagrant vagrant     136 Oct  9 19:30 17-Subprocess\n-rw-r--r--  1 vagrant vagrant
19419 Oct  9 20:09 17-Subprocess.ipynb\ndrwxr-xr-x  1 vagrant vagrant     204 Oct  9 18:54 18-Virtualenv\ndrwxr-xr-
x  1 vagrant vagrant     850 Oct  9 20:42 19-Testing\n-rw-r--r--  1 vagrant vagrant   25677 Oct  9 20:45 20-
MoreModules.ipynb\n-rw-r--r--  1 vagrant vagrant    1211 Oct  8 08:32 fabfile.py\n-rw-r--r--  1 vagrant vagrant
1823 Oct  8 08:33 fabfile.pyc\n-rw-r--r--  1 vagrant vagrant 1548914 Oct  8 08:20 FastTrackToPython.pdf\ndrwxr-xr-x
1 vagrant vagrant     442 Oct  9 19:44 .git\n-rw-r--r--  1 vagrant vagrant    1358 Oct  9 20:46 index.md\n-rw-r--r--
-  1 vagrant vagrant      61 Oct  3 03:19 .vagrant\n-rw-r--r--  1 vagrant vagrant    4008 Oct  3 03:20
Vagrantfile\n'
```

Popen

The above methods are just wrappers around the `Popen` constructor:

In [11]:
```
help(subprocess.Popen.__init__)
```

```
Help on method __init__ in module subprocess:

__init__(self, args, bufsize=0, executable=None, stdin=None, stdout=None, stderr=None, preexec_fn=None,
```

```
close_fds=False, shell=False, cwd=None, env=None, universal_newlines=False, startupinfo=None, creationflags=0)
unbound subprocess.Popen method
    Create new Popen instance.
```

In [12]:
```
sp = subprocess.Popen('ls', stdout=subprocess.PIPE)
print sp.stdout
```

```
<open file '<fdopen>', mode 'rb' at 0x185e930>
```

In [13]:
```
print sp.stdout.read(20)
```

```
01-BasicPythonSyntax
```

In [14]:
```
sp.wait()
```

Out[14]:
```
0
```

In [15]:
```
sp = subprocess.Popen('vi')
```

In [16]:
```
sp.poll()
```

In [17]:
```
sp.terminate()
```

In [18]:
```
sp.wait()
```

Out[18]:
```
-15
```

In [19]:
```
sp = subprocess.Popen(
    'ls',
    stdout=subprocess.PIPE)
print sp.communicate()
```

```
('01-BasicPythonSyntax\n01-BasicPythonSyntax_files\n01-BasicPythonSyntax.html\n01-BasicPythonSyntax.ipynb\n02-
Builtins\n02-Builtins_files\n02-Builtins.html\n02-Builtins.ipynb\n03-FileIO\n03-FileIO_files\n03-FileIO.html\n03-
FileIO.ipynb\n04-UsingModules\n04-UsingModules_files\n04-UsingModules.html\n04-UsingModules.ipynb\n05-Strings\n05-
Strings_files\n05-Strings.html\n05-Strings.ipynb\n06a-Packages\n06-Regex\n06-Regex_files\n06-Regex.html\n06-
Regex.ipynb\n07-Functions\n07-Functions_files\n07-Functions.html\n07-Functions.ipynb\n08-AdvancedFunctions\n08-
AdvancedFunctions_files\n08-AdvancedFunctions.html\n08-AdvancedFunctions.ipynb\n09-Logging\n09-Logging_files\n09-
Logging.html\n09-Logging.ipynb\n10-OOP1\n10-OOP1_files\n10-OOP1.html\n10-OOP1.ipynb\n11-OOP2\n11-OOP2_files\n11-
OOP2.html\n11-OOP2.ipynb\n12-Decorators\n12-Decorators_files\n12-Decorators.html\n12-Decorators.ipynb\n13-
Generators\n13-GeneratorsAndIterators_files\n13-GeneratorsAndIterators.html\n13-GeneratorsAndIterators.ipynb\n14-
ContextManagers\n14-ContextManagers_files\n14-ContextManagers.html\n14-ContextManagers.ipynb\n15-Threading\n16-
Multiprocessing\n17-Subprocess\n17-Subprocess.ipynb\n18-Virtualenv\n19-Testing\n20-
MoreModules.ipynb\nfabfile.py\nfabfile.pyc\nFastTrackToPython.pdf\nindex.md\nVagrantfile\n', None)
```

Pipelines

In [20]:
```
sp1 = subprocess.Popen(
    [ 'ls', '-l' ],
    stdin=subprocess.PIPE,
    stdout=subprocess.PIPE)
sp2 = subprocess.Popen(
    ['grep', 'FileIO'],
    stdin=sp1.stdout,
    stdout=subprocess.PIPE)
sp1.stdin.close()
print sp2.communicate('FileIO\nSomethingElse')[0]
```

```
drwxr-xr-x 1 vagrant vagrant     102 Oct  8 13:41 03-FileIO
drwxr-xr-x 1 vagrant vagrant     510 Oct  8 08:15 03-FileIO_files
-rw-r--r-- 1 vagrant vagrant   41620 Oct  8 08:15 03-FileIO.html
-rw-r--r-- 1 vagrant vagrant    9576 Oct  7 10:58 03-FileIO.ipynb
```

Exercises

- Write a script that uses `os.listdir()` and `subprocess.check_output()` to run a command on every file in a directory (for instance, the `stat` command)
- Write a script that creates a pipeline containing two or more commands

In [20]:
```
```

More stdlib Modules and Builtins

Sorting and reversing lists

In [1]:
```
lst = [ 'fast', 'track', 'to', 'python' ]
lst.sort()
lst
```

Out[1]:
```
['fast', 'python', 'to', 'track']
```

In [2]:
```
lst = [ 'fast', 'track', 'to', 'python' ]
print sorted(lst)
print lst
```

```
['fast', 'python', 'to', 'track']
['fast', 'track', 'to', 'python']
```

In [3]:
```
class User(object):
    def __init__(self, first, last):
        self.first = first
        self.last = last
    def __repr__(self):
        return '<User %s %s>' % (self.first, self.last)

users = [ User('Rick', 'Copeland'),
          User('Stuart', 'Kerr'),
          User('Tim', 'Allen'),
          User('Barack', 'Obama'),
          User('Mitt', 'Romney')
        ]

print sorted(users)
```

```
[<User Stuart Kerr>, <User Barack Obama>, <User Rick Copeland>, <User Tim Allen>, <User Mitt Romney>]
```

In [4]:
```
print sorted(users, key=lambda u:u.last)
```

```
[<User Tim Allen>, <User Rick Copeland>, <User Stuart Kerr>, <User Barack Obama>, <User Mitt Romney>]
```

In [5]:
```
print sorted(users, key=lambda u:u.first)
```

```
[<User Barack Obama>, <User Mitt Romney>, <User Rick Copeland>, <User Stuart Kerr>, <User Tim Allen>]
```

In [6]:
```
sorted_users = sorted(users, key=lambda u:(u.last, u.first))
print sorted_users
```

```
[<User Tim Allen>, <User Rick Copeland>, <User Stuart Kerr>, <User Barack Obama>, <User Mitt Romney>]
```

In [7]:
```
print reversed(sorted_users)
for user in reversed(sorted_users):
    print user
```

```
<listreverseiterator object at 0x22a5e10>
<User Mitt Romney>
<User Barack Obama>
<User Stuart Kerr>
<User Rick Copeland>
<User Tim Allen>
```

In [8]:
```
sorted(users, key=lambda u:(u.last, u.first), reverse=True)
```

Out[8]:
```
[<User Mitt Romney>,
 <User Barack Obama>,
 <User Stuart Kerr>,
 <User Rick Copeland>,
 <User Tim Allen>]
```

Treating URLs as files

In [9]:
```
import urllib2
import contextlib
with contextlib.closing(urllib2.urlopen('http://www.baidu.com/')) as fp:
    print fp.read()
```

```
<!doctype html><html><head><meta http-equiv="Content-Type" content="text/html;charset=gb2312"><title>°Ù¶ÈÒ»ÏÂ£¬Ä㾪ÍÖªµÀ      </ti
<style>html{overflow-y:auto}body{font:12px arial;text-align:center;background:#fff}body,p,form,ul,li{margin:0;padding:0;list-
style:none}body,form,#fm{position:relative}td{text-align:left}img{border:0}a{color:#00c}a:active{color:#f60}#u{color:#999;paddin
```

```
0;text-align:right}#u a{margin:0 5px}#u .reg{margin:0}#m{width:720px;margin:0 auto;}#nv a,#nv b,.btn,#lk{font-size:14px}#fm{padd
left:110px;text-align:left}input{border:0;padding:0}#nv{height:19px;font-size:16px;margin:0 0 4px;text-align:left;text-
indent:137px;}.s_ipt_wr{width:418px;height:30px;display:inline-block;margin-right:5px;background:url(http://s1.bdstatic.com/r/ww
1.0.0.png) no-repeat -304px 0;border:1px solid #b6b6b6;border-color:#9a9a9a #cdcdcd #cdcdcd #9a9a9a;vertical-
align:top}.s_ipt{width:405px;height:22px;font:16px/22px arial;margin:5px 0 0 7px;background:#fff;outline:none;-webkit-
appearance:none}.s_btn{width:95px;height:32px;padding-top:2px\9;font-size:14px;background:#ddd url(http://s1.bdstatic.com/r/www/
1.0.0.png);cursor:pointer}.s_btn_h{background-position:-100px 0}.s_btn_wr{width:97px;height:34px;display:inline-
block;background:url(http://s1.bdstatic.com/r/www/img/i-1.0.0.png) no-repeat -202px 0;*position:relative;z-index:0;vertical-alig
img{vertical-align:top;margin-bottom:3px}#lk{margin:33px 0}#lk span{font:14px "ËÎÌå"}#lm{height:60px}#lh{margin:16px 0 5px;word-
spacing:3px}.tools{position:absolute;top:-4px;*top:10px;right:7px;}#mHolder{width:62px;position:relative;z-
index:296;display:none}#mCon{height:18px;line-height:18px;position:absolute;cursor:pointer;padding:0 18px 0
0;background:url(http://s1.bdstatic.com/r/www/img/bg-1.0.0.gif) no-repeat right -134px;background-position:right -136px\9}#mCon
span{color:#00c;cursor:default;display:block}#mCon .hw{text-decoration:underline;cursor:pointer}#mMenu
a{width:100%;height:100%;display:block;line-height:22px;text-indent:6px;text-decoration:none;filter:none\9}#mMenu,#user ul{box-s
2px #ccc;-moz-box-shadow:1px 1px 2px #ccc;-webkit-box-shadow:1px 1px 2px #ccc;filter: progid:DXImageTransform.Microsoft.Shadow(S
Direction=135, Color="#cccccc")\9;}#mMenu{width:56px;border:1px solid #9b9b9b;list-
style:none;position:absolute;right:27px;top:28px;display:none;background:#fff}#mMenu a:hover{background:#ebebeb}#mMenu
.ln{height:1px;background:#ebebeb;overflow:hidden;font-size:1px;line-height:1px;margin-top:-1px}#cp,#cp
a{color:#77c}#seth{display:none;behavior:url(#default#homepage)}#setf{display:none;}#sekj{margin-left:14px;}</style>
<script type="text/javascript">function h(obj){obj.style.behavior='url(#default#homepage)';var a = obj.setHomePage('http://www.b
</script></head>

<body>

<div id="ie6tipcon"></div>


<div id="u"><a href="http://www.baidu.com/gaoji/preferences.html" name="tj_setting">ËÑË÷Éè¶¨</a>|<a href="https://passport.baidu
login&tpl=mn&u=http%3A%2F%2Fwww.baidu.com%2F" name="tj_login" id="lb" onclick="return false;">µÇÂ¼</a><a href="https://passport.
reg&regType=1&tpl=mn&u=http%3A%2F%2Fwww.baidu.com%2F" target="_blank" name="tj_reg" class="reg">×¢²á</a></div>
<div id="m"><p id="lg"><img src="http://www.baidu.com/img/baidu_sylogo1.gif" width="270" height="129" ></p>
<p id="nv"><a href="http://news.baidu.com">ÐÂ ÎÅ</a>¡¡<b>Íø Ò³</b>¡¡<a href="http://tieba.baidu.com">Ìû °É</a>¡¡<a
href="http://zhidao.baidu.com">Öª µÀ</a>¡¡<a href="http://mp3.baidu.com">MP3</a>¡¡<a href="http://image.baidu.com">Í¼ 
href="http://video.baidu.com">ÊÓ Æµ</a>¡¡<a href="http://map.baidu.com">µØ Í¼</a></p><div id="fm"><form name="f" actio
class="s_ipt_wr"><input type="text" name="wd" id="kw" maxlength="100" class="s_ipt"></span><input type="hidden" name="rsv_bp" va
type="hidden" name="rsv_spt" value="3"><span class="s_btn_wr"><input type="submit" value="°Ù¶È¾Ö»ÏÂ" id="su" class="s_btn"
onmousedown="this.className='s_btn s_btn_h'" onmouseout="this.className='s_btn'"></span></form><span class="tools"><span id="mHo
id="mCon"><span>ÊäÈë·¨</span></div></span></span><ul id="mMenu"><li><a href="#" name="ime_hw">ÊÖÐ´</a></li><li><a href="#" name=
´Ò¾</a></li><li class="ln"></li><li><a href="#" name="ime_cl">¹Ø±Õ</a></li></ul></div>
<p id="lk"><a href="http://baike.baidu.com">°Ù¿Æ</a>¡¡<a href="http://wenku.baidu.com">ÎÄ¿â</a>¡¡<a href="http://www.hao123.com">
<span> | <a href="http://www.baidu.com/more/">¸ü¶à&gt;&gt;</a></span></p><p id="lm"></p><p ><a id="seth" onClick="h(this)" href=
onmousedown="return ns_c({'fm':'behs','tab':'homepage','pos':0})">°Ñ°Ù¶È¼ÒÉèÎªÖ÷Ò³</a><a id="setf"
href="http://www.baidu.com/cache/sethelp/index.html" onmousedown="return ns_c({'fm':'behs','tab':'favorites','pos':0})" target="
°Ù¶È¼ÒÉèÎªÖ÷Ò³</a><span id="sekj"><a href="http://www.baidu.com/search/baidukuijie_mp.html" target="_blank" onmousedown="return
ns_c({'fm':'behs','tab':'kuaijie','pos':1})">°Ñ°Ù¶È¼¯³É¿ìËÙËÑË÷¹¤¾ß</a></span></p>
<p id="lh"><a href="http://e.baidu.com/?refer=888">¼ÓÈë°Ù¶È¹ã¸æ</a> | <a href="http://top.baidu.com">ÐÑÉ·çÔÆ°ñ</a> | <a
href="http://home.baidu.com">¹ØÓÚ°Ù¶È</a> | <a href="http://ir.baidu.com">About Baidu</a></p><p id="cp">&copy;2012 Baidu <a href=
°Ù¶È×ö±Ø¶Á</a> <a href="http://www.miibeian.gov.cn" target="_blank">¾©ICP¤030173ºÅ</a> <img src="http://www.baidu.com/cache/glob
</p>
</div></body>

<script>var bds={se:{},comm : {ishome : 1,sid : "",user : "",username : "",sugHost : "http://suggestion.baidu.com/su",loginActio
</script><script type="text/javascript" src="http://s1.bdstatic.com/r/www/cache/global/js/home-1.6.js"></script><script>var bdUs
w=window,d=document,n=navigator,k=d.f.wd,a=d.getElementById("nv").getElementsByTagName("a"),isIE=n.userAgent.indexOf("MSIE")!=-1
(function(){if(/q=([^&]+)/.test(location.search)){k.value=decodeURIComponent(RegExp["\x241"])}})();if(n.cookieEnabled&&!/sug?=0/
{bds.se.sug();};function addEV(o, e, f){if(w.attachEvent){o.attachEvent("on" + e, f);}else if(w.addEventListener){ o.addEventLis
false);}}function G(id){return d.getElementById(id);}function ns_c(q){var p = encodeURIComponent(window.document.location.href),
'', mu='', img = window["BD_PS_C" + (new Date()).getTime()] = new Image();for (v in q) {sV = q[v];sQ += v + "=" + sV + "&";} mu=
;img.src = "http://nsclick.baidu.com/v.gif?pid=201&pj=www&rsv_sid=&" + sQ + "path="+p+"&t="+new Date().getTime();return true;}if
[12]/.test(d.cookie)){document.write('<script src=http://s1.bdstatic.com/r/www/cache/ime/js/openime-1.0.0.js><\/script>');}(func
G("u").getElementsByTagName("a"), nv = G("nv").getElementsByTagName("a"), lk = G("lk").getElementsByTagName("a"), un = "";var tj
["news","tieba","zhidao","mp3","img","video","map"];var tj_lk = ["baike","wenku","hao123","more"];un = bds.comm.user == "" ? ""
bds.comm.user;function _addTJ(obj){addEV(obj, "mousedown", function(e){var e = e || window.event;var target = e.target ||
e.srcElement;ns_c({'fm':'behs','tab':target.name||'tj_user','un':encodeURIComponent(un)});});}for(var i = 0; i < u.length; i++)
{_addTJ(u[i]);}for(var i = 0; i < nv.length; i++){nv[i].name = 'tj_' + tj_nv[i];}for(var i = 0; i < lk.length; i++){lk[i].name =
tj_lk[i];}})();(function() {var links = {'tj_news':   ['word', 'http://news.baidu.com/ns?tn=news&cl=2&rn=20&ct=1&ie=utf-8'],'tj_
'http://tieba.baidu.com/f?ie=utf-8'],'tj_zhidao': ['word', 'http://zhidao.baidu.com/search?pn=0&rn=10&lm=0'],'tj_mp3':    ['word
'http://mp3.baidu.com/m?f=ms&tn=baidump3&ct=134217728&rn=&lm=-1&pn=30&ie=utf-8'],'tj_img':    ['word', 'http://image.baidu.com/i
ct=201326592&cl=2&nc=1&lm=-1&st=-1&tn=baiduimage&istype=2&fm=&pv=&z=0&ie=utf-8'],'tj_video':  ['word', 'http://video.baidu.com/v
ct=301989888&s=25&ie=utf-8'],'tj_map':    ['wd', 'http://map.baidu.com/?newmap=1&ie=utf-8&s=s'],'tj_baike':  ['word',
'http://baike.baidu.com/search/word?pic=1&sug=1&enc=utf8'],'tj_wenku':  ['word', 'http://wenku.baidu.com/search?ie=utf-8']};var
[G('nv'), G('lk')],kw = G('kw');for (var i = 0, l = domArr.length; i < l;  i++) {domArr[i].onmousedown = function(e) {e = e || w
target = e.target || e.srcElement,name = target.getAttribute('name'),items = links[name],reg = new RegExp('^\\s+|\\s+\x24'),key
kw.value.replace(reg, '');if (items) {if (key.length > 0) {var wd = items[0], url = items[1],url = url + ( name === 'tj_map' ?
encodeURIComponent('&' + wd + '=' + key)  : ( ( url.indexOf('?') > 0 ? '&' : '?' ) + wd + '=' + encodeURIComponent(key)  ) );tar
else {target.href = target.href.match(new RegExp('^http://.+\\.baidu\\.com'))[0];}}name && ns_c({'fm': 'behs','tab': name,'query
encodeURIComponent(key),'un': encodeURIComponent(bds.comm.user || '')    });};})();addEV(w,"load",function(){k.focus()});w.onun
{};</script><script type="text/javascript" src="http://s1.bdstatic.com/r/www/cache/global/js/tangram-1.3.4c1.0.js"></script><scr
type="text/javascript" src="http://s1.bdstatic.com/r/www/cache/user/js/u-1.3.4.js"></script><!--[if IE 6]><![endif]-->



</html>



<!--c35a3a95824fd856-->
```

Serialization and deserialization

Python native "pickle"

In [10]:
```
import pickle
d = {'foo': [ 1,2,3], 'bar': (1+2j), 'baz': (1,2) }
d_pickle = pickle.dumps(d)
d_pickle
```

Out[10]:
```
"
(dp0\nS'baz'\np1\n(I1\nI2\ntp2\nsS'foo'\np3\n(lp4\nI1\naI2\naI3\nasS'bar'\np5\nc__builtin__\ncomplex\np6\n(F1.0\nF2.0\ntp7\nRp8\
```

In [11]:
```
d_unpickle = pickle.loads(d_pickle)
d_unpickle
```

Out[11]:
```
{'bar': (1+2j), 'baz': (1, 2), 'foo': [1, 2, 3]}
```

JSON

In [12]:
```
import json
d = {'foo': [ 1,2,3], 'bar': 'This is bar', 'baz': {1:2} }
d_json = json.dumps(d)
d_json
```

Out[12]:
```
'{"baz": {"1": 2}, "foo": [1, 2, 3], "bar": "This is bar"}'
```

In [13]:
```
json.loads(d_json)
```

Out[13]:
```
{u'bar': u'This is bar', u'baz': {u'1': 2}, u'foo': [1, 2, 3]}
```

Comma-separated value (CSV)

In [14]:
```
from StringIO import StringIO
import csv
```

In [15]:
```
fp = StringIO()
writer = csv.writer(fp)
writer.writerow([1,2,3,4,5,6])
writer.writerow(['this', 'is', 'comma', 'separated value,'])
writer.writerow(['and', 'this', 'can', 'be', 'read', 'by', 'excel'])
```

In [16]:
```
print fp.getvalue()
```

```
1,2,3,4,5,6
this,is,comma,"separated value,"
and,this,can,be,read,by,excel
```

In [17]:
```
fp.seek(0)
reader = csv.reader(fp)
for row in reader:
    print row
```

```
['1', '2', '3', '4', '5', '6']
['this', 'is', 'comma', 'separated value,']
['and', 'this', 'can', 'be', 'read', 'by', 'excel']
```

Filename matching

In [18]:
```
import glob
glob.glob('*.ipynb')
```

Out[18]:
```
['01-BasicPythonSyntax.ipynb',
 '02-Builtins.ipynb',
 '03-FileIO.ipynb',
 '04-UsingModules.ipynb',
 '05-Strings.ipynb',
 '06-Regex.ipynb',
 '07-Functions.ipynb',
 '08-AdvancedFunctions.ipynb',
 '09-Logging.ipynb',
 '10-OOP1.ipynb',
 '11-OOP2.ipynb',
```

```
 '12-Decorators.ipynb',
 '13-GeneratorsAndIterators.ipynb',
 '14-ContextManagers.ipynb',
 '17-Subprocess.ipynb',
 '20-MoreModules.ipynb']
```

Print "pretty" output

In [19]:
```
import pprint
```

In [20]:
```
x = [{'baz': {1: (2,x)}, 'foo': [1, 2, 3], 'bar': 'This is bar'}
     for x in range(10) ]
print x
```

```
[{'bar': 'This is bar', 'foo': [1, 2, 3], 'baz': {1: (2, 0)}}, {'bar': 'This is bar', 'foo': [1, 2, 3], 'baz': {1: (2, 1)}},
{'bar': 'This is bar', 'foo': [1, 2, 3], 'baz': {1: (2, 2)}}, {'bar': 'This is bar', 'foo': [1, 2, 3], 'baz': {1: (2, 3)}},
{'bar': 'This is bar', 'foo': [1, 2, 3], 'baz': {1: (2, 4)}}, {'bar': 'This is bar', 'foo': [1, 2, 3], 'baz': {1: (2, 5)}},
{'bar': 'This is bar', 'foo': [1, 2, 3], 'baz': {1: (2, 6)}}, {'bar': 'This is bar', 'foo': [1, 2, 3], 'baz': {1: (2, 7)}},
{'bar': 'This is bar', 'foo': [1, 2, 3], 'baz': {1: (2, 8)}}, {'bar': 'This is bar', 'foo': [1, 2, 3], 'baz': {1: (2, 9)}}]
```

In [21]:
```
pprint.pprint(x)
```

```
[{'bar': 'This is bar', 'baz': {1: (2, 0)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 1)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 2)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 3)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 4)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 5)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 6)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 7)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 8)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 9)}, 'foo': [1, 2, 3]}]
```

In [22]:
```
print 'x is \n%s' % pprint.pformat(x)
```

```
x is
[{'bar': 'This is bar', 'baz': {1: (2, 0)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 1)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 2)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 3)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 4)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 5)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 6)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 7)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 8)}, 'foo': [1, 2, 3]},
 {'bar': 'This is bar', 'baz': {1: (2, 9)}, 'foo': [1, 2, 3]}]
```

random numbers

In [23]:
```
import random
```

In [24]:
```
random.random()
```

Out[24]:
```
0.6208437699012457
```

In [25]:
```
random.randint(1, 100)
```

Out[25]:
```
10
```

In [26]:
```
x = range(10)
random.shuffle(x)
x
```

Out[26]:
```
[5, 6, 9, 1, 0, 3, 7, 8, 4, 2]
```