Fast Track to Python

Code Examples

./01-BasicPythonSyntax/

./01-BasicPythonSyntax/hello.py

```
#!/usr/bin/env python
def sayhello(name):
    print 'Hello, ' + name

sayhello('Rick')
```

./01-BasicPythonSyntax/reverse_list.py

```
def reverse_list(lst):
    return lst[::-1]

print reverse_list([1,2,3])
```

./01-BasicPythonSyntax/sum_even_values.py

```
def sum_even_values(lst):
    result = 0
    lst1 = lst[::2]
    for element in lst1:
        result += element
    return result

print sum_even_values([1,2,3])
print sum_even_values([100, 200])
```

./01-BasicPythonSyntax/sum_values.py

```
def sum_values(lst):
    result = 0
    for element in lst:
        result += element
    return result

print sum_values([1,2,3])
print sum_values([100, 200])
```

./01-BasicPythonSyntax/telephone_directory.py

```
def add_number(directory, name, number):
    directory[name] = number

def remove_number(directory, name):
    del directory[name]

def lookup_number(directory, name):
```

```
    return directory[name]

directory = {}
add_number(directory, 'Rick', '404.452.5202')
print lookup_number(directory, 'Rick')
print directory
remove_number(directory, 'Rick')
print directory

def remove_number_alt(directory, name):
    try:
        del directory[name]
    except KeyError:
        pass # ignore KeyError

remove_number_alt(directory, 'Nobody')
print directory
```

./02-Builtins/

./02-Builtins/ascii2str.py

```
def ascii2str(lst):
    s = ''
    characters = map(chr, lst)
    for ch in characters:
        s += ch
    return s

print ascii2str([86, 77, 87, 97, 114, 101])

def ascii2str_alt(lst):
    characters = map(chr, lst)
    return ''.join(characters)

print ascii2str_alt([86, 77, 87, 97, 114, 101])
```

./02-Builtins/make_dict.py

```
def make_dict(keys, values):
    return dict(zip(keys, values))

print make_dict(['Rick Copeland'], ['404.452.5202'])
```

./03-FileIO/

./03-FileIO/print_file.py

```
def print_file(fp):
    for line in fp:
        print line[:-1]

fp = open('/etc/hosts')
print_file(fp)
fp.close()

def print_file_line_numbers(fp):
```

```
    for index, line in enumerate(fp):
        print index + 1, line[:-1]

fp = open('/etc/hosts')
print_file_line_numbers(fp)
fp.close()
```

./04-UsingModules/

./04-UsingModules/change_path.py

```
import sys

sys.path = []

import time
```

./04-UsingModules/convert_dt_to_ts.py

```
import time

def dt_to_ts(dt):
    return time.mktime(dt.timetuple())
```

./04-UsingModules/datetime_every_second.py

```
import datetime
import time

while True:
    print datetime.datetime.now()
    time.sleep(1)
```

./04-UsingModules/e_to_the_jpi.py

```
import math

print math.e ** (1j*math.pi)
```

./04-UsingModules/print_file.py

```
def print_file_line_numbers(fp):
    for index, line in enumerate(fp):
        print index + 1, line,

import StringIO

fp = StringIO.StringIO('''The quick
brown fox
jumped over
the lazy dog''')

print_file_line_numbers(fp)
```

./04-UsingModules/print_file_debug.py

```
def print_file_line_numbers(fp):
    import pdb; pdb.set_trace()
    for index, line in enumerate(fp):
        print index + 1, line[:-1]

import StringIO

fp = StringIO.StringIO('''The quick
brown fox
jumped over
the lazy dog''')

print_file_line_numbers(fp)
```

./04-UsingModules/print_my_argv.py

```
import sys

print sys.argv
```

./04-UsingModules/print_my_path.py

```
import sys
import os.path

print os.path.abspath(sys.argv[0])
```

./04-UsingModules/time_every_second.py

```
import time

while True:
    print time.time()
    time.sleep(1)
```

./05-Strings/

./05-Strings/get_word_count.py

```
import StringIO

text = '''
The quick brown fox jumped over the lazy dog.
The dog was very lazy and the fox was quite quick.
'''

def isalpha(ch):
    return ch.isalpha()

def get_words_count(fp):
    result = {}
    for line in fp:
        for word in line.split():
            word = ''.join(filter(isalpha, word.lower()))
            if word in result:
                result[word] += 1
```

```
            else:
                result[word] = 1
    return result

print get_words_count(StringIO.StringIO(text))
```

./05-Strings/get_words.py

```
import StringIO

text = '''
The quick brown fox jumped over the lazy dog.
The dog was very lazy and the fox was quite quick.
'''

def isalpha(ch):
    return ch.isalpha()

def get_words(fp):
    result = []
    for line in fp:
        for word in line.split():
            word = ''.join(filter(isalpha, word.lower()))
            result.append(word)
    return result

print get_words(StringIO.StringIO(text))
```

./05-Strings/print_centered_words.py

```
def print_centered_words(words):
    for word in words:
        print word.title().center(80)

print_centered_words(['The', 'quick', 'brown',  'fox'])
```

./06-Regex/

./06-Regex/regex_tests.py

```
import re
re_integer = re.compile(r'(\d+)')

def find_integers(fp):
    result = []
    for line in fp:
        for match in re_integer.finditer(line):
            result.append(int(match.group(1)))
    return result

text = '''The 42nd number in a list of integers starting at
0 is actually 41. It's a surprising result, but one that
computer scientists have dealt with since the days of IBM mainframes.'''

import StringIO
print 'Integers:', find_integers(StringIO.StringIO(text))
```

```
re_capword = re.compile(r"(\W|^)([A-Z][A-Za-z']*)")
def find_capwords(fp):
    result = []
    for line in fp:
        for match in re_capword.finditer(line):
            result.append(match.group(2))
    return result

print 'Capwords:', find_capwords(StringIO.StringIO(text))

re_br = re.compile(r'<br>')
def bad_html_to_xhtml(fp):
    result = []
    for line in fp:
        new_line = re_br.sub('<br/>', line)
        result.append(new_line)
    return ''.join(result)

print 'Bad html to xhtml:', bad_html_to_xhtml(StringIO.StringIO('''
<div>
This works ok
<br>
</div>'''))
```

./07-Functions/

./07-Functions/log_function.py

```
def log(format, *args, **kwargs):
    if len(args):
        print format % args
    else:
        print format % kwargs

log('The pair is (%r,%r)', 1, 2)
log('The value of a is %(a)r', a='foo')
log('This does not have any arguments')
```

./08-AdvancedFunctions/

./08-AdvancedFunctions/myfilter.py

```
def myfilter(function, sequence):
    result = []
    for item in sequence:
        if function(item):
            result.append(item)
    return result

print myfilter(lambda x: x%2==0, range(10))
print myfilter(lambda x: x%2==1, range(10))

## Talk about "truthy" values
```

./08-AdvancedFunctions/postorder.py

```
mytree = ('root',
```

```
          ('child-L',
           ('child-LL', (), ()),
           ('child-LR', (), ())),
          ('child-R',
           ('child-RL', (), ()),
           ('child-RR', (), ())))

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

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

print postorder_tree_map(print_node, mytree)
```

./09-Logging/

./09-Logging/fileconfig.ini

```
[loggers]
keys = root, mylogger

[handlers]
keys = stream, file, http

[formatters]
keys = basic, precise

[logger_root]
level = DEBUG
handlers = stream

[logger_mylogger]
qualname = mylogger
level = INFO
handlers = stream, file, http
propagate = 0

[handler_stream]
class = StreamHandler
formatter = basic
args = (sys.stderr,)

[handler_file]
class = handlers.WatchedFileHandler
args = ('/tmp/log_file.log','w')
formatter = precise

[handler_http]
class = handlers.HTTPHandler
args = ('localhost:9022', '/log', 'GET')
formatter = precise
```

```
[formatter_basic]
format = %(message)s

[formatter_precise]
format = %(asctime)s %(levelname)-8s %(name)-15s %(message)s
datefmt = %Y-%m-%d %H:%M:%S
```

./09-Logging/log_example.py

```
import logging.config

logging.basicConfig(
    level=logging.INFO,
    format='%(pathname)s:%(lineno)s %(levelname)s %(levelname)-8s %(name)-15s %(message)s'
    )
log = logging.getLogger()
log.info('Log here')

log.info("And here")
```

./09-Logging/log_file_config.py

```
import logging.config

logging.config.fileConfig('09-Logging/fileconfig.ini')

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

root.error('Info from root')
mylogger.error('Info from mylogger')
```

./10-OOP1/

./10-OOP1/directory.py

```
class Directory(object):

    def __init__(self):
        self._directory = {}

    def add_number(self, name, number):
        self._directory[name] = number

    def remove_number(self, name):
        self._directory.pop(name, None)

    def lookup_number(self, name):
        return self._directory.get(name, '<<unknown>>')

    def print_directory(self):
        print 'Begin directory'
        print self._directory
        for name, number in self._directory.items():
            print '    %s: %s' % (name, number)
        print 'End directory'
```

```
d = Directory()
d.add_number('Rick', '404.452.5202')
print 'Rick has number', d.lookup_number('Rick')
d.print_directory()
print
d.remove_number('Rick')
d.print_directory()
print
print 'Rick has number', d.lookup_number('Rick')
print
d.remove_number('Rick')
```

./11-OOP2/

./11-OOP2/directory-1.py

```
class Directory(object):

    def __init__(self):
        self._directory = {}

    def add_number(self, name, number):
        self._directory[name] = number

    def remove_number(self, name):
        del self._directory[name]

    def lookup_number(self, name):
        return self._directory[name]

    def __getitem__(self, name):
        return self.lookup_number(name)

    def __setitem__(self, name, number):
        self.add_number(name, number)

    def __delitem__(self, name):
        self.remove_number(name)

    def __repr__(self):
        l = ['<Directory>']
        for name, number in self._directory.items():
            l.append('    %s: %s' % (name, number))
        l.append('</Directory>')
        return '\n'.join(l)

d = Directory()
d['Rick'] = '404.452.5202'
print "Rick's number is", d['Rick']
print d
del d['Rick']
print d
```

./11-OOP2/directory-2.py

```
class Directory(object):
```

```
    def __init__(self):
        self._directory = {}

    def add_number(self, name, number):
        self._directory[name] = number

    def remove_number(self, name):
        del self._directory[name]

    def lookup_number(self, name):
        return self._directory[name]

    def __getitem__(self, name):
        return self.lookup_number(name)

    def __setitem__(self, name, number):
        self.add_number(name, number)

    def __delitem__(self, name):
        self.remove_number(name)

    def __repr__(self):
        l = ['<Directory>']
        for name, number in self._directory.items():
            l.append('    %s: %s' % (name, number))
        l.append('</Directory>')
        return '\n'.join(l)

class DefaultDirectory(Directory):

    def __init__(self, default_number):
        self._default = default_number
        super(DefaultDirectory, self).__init__()

    def lookup_number(self, name):
        try:
            return super(DefaultDirectory, self).lookup_number(name)
        except KeyError:
            return self._default

    def remove_number(self, name):
        try:
            super(DefaultDirectory, self).remove_number(name)
        except KeyError:
            print (
                'Would have raised an exception deleting %s'
                % name)
            pass

    def __repr__(self):
        l = ['<DefaultDirectory(%r)>' % self._default]
        for name, number in self._directory.items():
            l.append('    %s: %s' % (name, number))
        l.append('</DefaultDirectory>')
        return '\n'.join(l)

d = Directory()
d['Rick'] = '404.452.5202'
print "Rick's number is", d['Rick']
print d
del d['Rick']
```

```
print d

dd = DefaultDirectory('default')
dd['Rick'] = '404.452.5202'
print 'Rick: %s' % dd['Rick']
print 'Stuart: %s' % dd['Stuart']
print dd
del dd['Stuart']
```

./12-Decorators/

./12-Decorators/log_function_calls.py

```
import logging

logging.basicConfig()

class log_call(object):

    def __init__(self, logger, level=logging.INFO):
        self._logger = logger
        self._level = level

    def __call__(self, function):
        def wrapper(*args, **kwargs):
            self._logger.log(
                self._level, 'Enter %s(*%r, **%r)', function, args, kwargs)
            result = function(*args, **kwargs)
            self._logger.log(
                self._level, 'Exit %s => %r', function, result)
            return result
        return wrapper

@log_call(logging.getLogger('mylogger'), logging.ERROR)
def will_log_to_error(a, b):
    return a + b

print will_log_to_error(1, 2)
```

./12-Decorators/read-only-property.py

```
class MyClass(object):

    def __init__(self, a):
        self._a = a

    @property
    def a(self):
        return self._a

x = MyClass('avalue')
print x.a
x.a = 'bvalue'
```

./12-Decorators/with_file.py

```
class with_file(object):
```

```
    def __init__(self, *open_args):
        self._open_args = open_args

    def __call__(self, function):
        def wrapper(*args, **kwargs):
            fp = open(*self._open_args)
            try:
                return function(fp, *args, **kwargs)
            finally:
                fp.close()
        return wrapper

@with_file('/etc/hosts')
def print_file(fp):
    for i, line in enumerate(fp):
        print '%.4d: %s' % (i+1, line.rstrip())

print_file()
```

./13-Generators/

./13-Generators/tree-print.py

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

def postorder_tree_iter(node, level=0):
    if node:
        value, left, right = node
        for item in postorder_tree_iter(left, level+1):
            yield item
        for item in postorder_tree_iter(right, level+1):
            yield item
        yield level, value

for level, node in postorder_tree_iter(mytree):
    print ' ' * level, node
```

./14-ContextManagers/

./14-ContextManagers/logger.py

```
import logging

logging.basicConfig()

class log_block(object):

    def __init__(self, logger, level=logging.INFO):
        self._logger = logger
        self._level = level
```

```
    def __enter__(self):
        self._logger.log(self._level, 'Enter')

    def __exit__(self, ex_type, ex_value, ex_tb):
        if ex_type is None:
            self._logger.log(self._level, 'Exit (no exception)')
        else:
            self._logger.log(self._level, 'Exit (with exception %s)', ex_type)
            return True

print 'This is before the with statement'

with log_block(logging.getLogger('mylogger'), logging.ERROR):
    print 'Now inside the block'
    print 'still inside block'

with log_block(logging.getLogger('mylogger'), logging.ERROR):
    print 'Now inside the 2nd block'
    print 'still inside 2nd block'
    raise ValueError

def log_decorator(logger, level=logging.INFO):
    '''Just for fun'''
    def decorator(function):
        def wrapper(*args, **kwargs):
            with log_block(logger, level):
                return function(*args, **kwargs)
        return wrapper
    return decorator
```

./14-ContextManagers/logger2.py

```
import logging
from contextlib import contextmanager

logging.basicConfig()

@contextmanager
def log_block(logger, level=logging.INFO):
    logger.log(level, 'Enter')
    try:
        yield
    except:
        logger.log(level, 'Exit (with exception)')
    else:
        logger.log(level, 'Exit (no exception)')

print 'This is before the with statement'

with log_block(logging.getLogger('mylogger'), logging.ERROR):
    print 'Now inside the block'
    print 'still inside block'

with log_block(logging.getLogger('mylogger'), logging.ERROR):
    print 'Now inside the 2nd block'
    print 'still inside 2nd block'
    raise ValueError
```

./14-ContextManagers/xml-gen.py

```
class node(object):

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

    def __enter__(self):
        print '<%s>' % self.name

    def __exit__(self, ex_type, ex_value, ex_tb):
        print '</%s>' % self.name

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

./14-ContextManagers/xml-gen2.py

```
from contextlib import contextmanager

@contextmanager
def node(name):
    print '<%s>' % name
    yield
    print '</%s>' % name

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

./15-Threading/

./15-Threading/atomic_log.py

```
import sys
import threading

log_mutex = threading.Lock()

def log(message, *args):
    with log_mutex:
        slow_log(message, *args)

def slow_log(message, *args):
    message = message % args
    for ch in message:
        sys.stdout.write(ch)
        sys.stdout.flush()
    sys.stdout.write('\n')
    sys.stdout.flush()

def target(x):
    for y in range(4):
        log('(x,y) is (%d, %d)',x,y)

threads = [ threading.Thread(target=target, args=(x,))
            for x in range(4) ]
```

```
for t in threads:
    t.start()
```

./15-Threading/condition.py

```
import time
import logging
import threading

thread_to_run = None

# Set logger to just use threadname
logging.basicConfig(
    level=logging.INFO,
    format='%(threadName)s: %(message)s')

log = logging.getLogger()

cond = threading.Condition()

def worker(y):
    global thread_to_run
    with cond:
        while thread_to_run != y:
            cond.wait()
        log.info('Running thread %d', y)
        time.sleep(0.5)
        log.info('Now done')
        thread_to_run = None
        cond.notify_all()

def coordinator(num_threads):
    global thread_to_run
    for x in range(num_threads):
        with cond:
            while thread_to_run is not None:
                cond.wait()
            thread_to_run = x
            cond.notify_all()

workers = [ threading.Thread(target=worker, args=(x,))
            for x in range(10) ]
for t in workers: t.start()

coordinator(10)
```

./15-Threading/event.py

```
import logging
import threading

# Set logger to just use threadname
logging.basicConfig(
    level=logging.INFO,
    format='%(threadName)s: %(message)s')

log = logging.getLogger()
```

```
ev = threading.Event()

def timer():
    log.info('Timer running')
    ev.set()

def target():
    log.info('Target waiting')
    ev.wait()
    log.info('Target running')
    ev.clear()

t1 = threading.Thread(target=target)
t1.start()

t2 = threading.Timer(3, timer)
t2.start()
```

./15-Threading/lock1.py

```
import logging
import threading

# Set logger to just use threadname
logging.basicConfig(
    level=logging.INFO,
    format='%(threadName)s: %(message)s')

log = logging.getLogger()

lock = threading.Lock()

def thread_target(y):
    lock.acquire()
    log.info('Enter')
    for item in range(y):
        log.info('%s', item)
    log.info('Exit')
    lock.release()

threads = [ threading.Thread(target=thread_target, args=(4,))
            for x in range(4) ]

log.info('Starting threads')
for i, t in enumerate(threads):
    log.info('Starting thread %d', i)
    t.start()
log.info('All threads started')
```

./15-Threading/lock2.py

```
import logging
import threading

# Set logger to just use threadname
logging.basicConfig(
    level=logging.INFO,
    format='%(threadName)s: %(message)s')
```

```
log = logging.getLogger()

lock = threading.Lock()

def thread_target(y):
    with lock:
        log.info('Enter')
        for item in range(y):
            log.info('%s', item)
        log.info('Exit')

threads = [ threading.Thread(target=thread_target, args=(4,))
            for x in range(4) ]

log.info('Starting threads')
for i, t in enumerate(threads):
    log.info('Starting thread %d', i)
    t.start()
log.info('All threads started')
```

./15-Threading/print_time.py

```
import time
import threading

def print_time():
    while True:
        print time.ctime()
        time.sleep(1)

t = threading.Thread(target=print_time)
t.setDaemon(True)
t.start()

time.sleep(10)
```

./15-Threading/queue.py

```
import time
import Queue
import logging
import threading

logging.basicConfig(
    level=logging.INFO,
    format='%(threadName)s: %(message)s')

log = logging.getLogger()

q = Queue.Queue()

def producer():
    for x in range(10):
        time.sleep(0.5)
        log.info('>>> %s', x)
        q.put(x)

def consumer():
    while True:
```

```
        x = q.get()
        log.info('<<< %s', x)

t_producer = threading.Thread(target=producer)
t_consumer = threading.Thread(target=consumer)
t_consumer.setDaemon(True)

t_producer.start()
time.sleep(2)
t_consumer.start()
t_producer.join()
time.sleep(0)
```

./15-Threading/sem1.py

```
import time
import logging
import threading

# Set logger to just use threadname
logging.basicConfig(
    level=logging.INFO,
    format='%(threadName)s: %(message)s')

log = logging.getLogger()

sem = threading.Semaphore(4)

def thread_target(y):
    with sem:
        log.info('Enter')
        time.sleep(1)
        log.info('Exit')

threads = [ threading.Thread(target=thread_target, args=(4,))
            for x in range(10) ]

for i, t in enumerate(threads):
    t.start()
```

./15-Threading/threading1.py

```
import logging
import threading

logging.basicConfig(
    level=logging.INFO)

log = logging.getLogger('main')

def thread_target(x, y):
    log = logging.getLogger('thread-%d' % x)
    log.info('Enter')
    for item in range(y):
        log.info('%s', item)
    log.info('Exit')

threads = [ threading.Thread(target=thread_target, args=(x, 4))
            for x in range(4) ]
```

```
log.info('Starting threads')
for i, t in enumerate(threads):
    log.info('Starting thread %d', i)
    t.start()
log.info('All threads started')
```

./15-Threading/threading2.py

```
import logging
import threading

logging.basicConfig(
    level=logging.INFO)

log = logging.getLogger('main')

def thread_target(x, y):
    log = logging.getLogger('thread-%d' % x)
    log.info('Enter')
    for item in range(y):
        log.info('%s', item)
    log.info('Exit')

threads = [ threading.Thread(target=thread_target, args=(x, 4))
            for x in range(4) ]

log.info('Starting threads')
for i, t in enumerate(threads):
    log.info('Starting thread %d', i)
    t.start()
    # Wait for thread to complete
    t.join()
    log.info('Joined thread %d', i)
log.info('All threads started')
```

./15-Threading/threading3.py

```
import logging
import threading

# Set logger to just use threadname
logging.basicConfig(
    level=logging.INFO,
    format='%(threadName)s: %(message)s')

log = logging.getLogger()

def thread_target(x, y):
    log.info('Enter')
    for item in range(y):
        log.info('%s', item)
    log.info('Exit')

threads = [ threading.Thread(target=thread_target, args=(x,4))
            for x in range(4) ]

log.info('Daemonizing threads')
for i, t in enumerate(threads):
```

```
    t.setDaemon(True)

log.info('Starting threads')
for i, t in enumerate(threads):
    log.info('Starting thread %d', i)
    t.start()
log.info('All threads started')
```

./15-Threading/threadlocal.py

```
import logging
import threading

logging.basicConfig(
    level=logging.INFO,
    format='%(threadName)s: %(message)s')

thread_local = threading.local()
thread_local.name = 'Set in main thread'

log = logging.getLogger()

def target():
    thread_local.name = 'Set in target thread'
    log.info('thread_local.name = %s', thread_local.name)

log.info('thread_local.name = %s', thread_local.name)
t = threading.Thread(target=target)
t.start()
t.join()
log.info('thread_local.name = %s', thread_local.name)
```

./15-Threading/timer.py

```
import logging
import threading

# Set logger to just use threadname
logging.basicConfig(
    level=logging.INFO,
    format='%(threadName)s: %(message)s')

log = logging.getLogger()

def hello(x):
    log.info('Hello, %s', x)

t = threading.Timer(5.0, hello, ('World',))
t.start()
log.info('Main program complete')
```

./15-Threading/timer2.py

```
import logging
import threading

# Set logger to just use threadname
logging.basicConfig(
```

```
    level=logging.INFO,
    format='%(threadName)s: %(message)s')

log = logging.getLogger()

def hello(x):
    log.info('Hello, %s', x)

t = threading.Timer(5.0, hello, ('World',))
t.start()
log.info('Main program complete')
t.cancel()
```

./16-Multiprocessing/

./16-Multiprocessing/atomic_log.py

```
import sys
import threading

log_mutex = threading.Lock()

def log(message):
    with log_mutex:
        slow_log(message)

def slow_log(message):
    for ch in message:
        sys.stdout.write(ch)
        sys.stdout.flush()
    sys.stdout.write('\n')
    sys.stdout.flush()

def target(x):
    for y in range(4):
        log('(x,y) is (%d, %d)' % (x,y))

threads = [ threading.Thread(target=target, args=(x,))
            for x in range(4) ]

for t in threads:
    t.start()
```

./16-Multiprocessing/lock1.py

```
import time
import logging
import multiprocessing

# Set logger to just use threadname
logging.basicConfig(
    level=logging.INFO,
    format='%(processName)s: %(message)s')

log = logging.getLogger()

lock = multiprocessing.Lock()
```

```
def target(y):
    with lock:
        log.info('Enter')
        for item in range(y):
            time.sleep(0.1)
            log.info('%s', item)
        log.info('Exit')

procs = [ multiprocessing.Process(target=target, args=(4,))
            for x in range(4) ]

log.info('Starting procs')
for i, t in enumerate(procs):
    log.info('Starting proc %d', i)
    t.start()
log.info('All procs started')
```

./16-Multiprocessing/print_time.py

```
import time
import multiprocessing

def print_time():
    while True:
        print time.ctime()
        time.sleep(1)

def main():
    t = multiprocessing.Process(target=print_time)
    t.start()
    time.sleep(10)
    t.terminate()

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

./16-Multiprocessing/processing1.py

```
import time
import logging
import multiprocessing

logging.basicConfig(
    level=logging.INFO,
    format='%(processName)s (%(process)s): %(message)s')

log = logging.getLogger()

def main():
    procs = [ multiprocessing.Process(target=target, args=(x, 4))
                for x in range(4) ]
    log.info('Starting procs')
    for i, p in enumerate(procs):
        log.info('Starting process %d', i)
        p.start()
    log.info('All procs started')

def target(x, y):
    log.info('Enter')
```

```
    for item in range(y):
        log.info('(%s,%s)', x, item)
        time.sleep(0.1)
    log.info('Exit')

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

./16-Multiprocessing/processing2.py

```
import time
import logging
import multiprocessing

logging.basicConfig(
    level=logging.INFO,
    format='%(processName)s (%(process)s): %(message)s')

log = logging.getLogger()

def main():
    procs = [ multiprocessing.Process(target=target, args=(x, 4))
                for x in range(4) ]
    log.info('Starting procs')
    for i, p in enumerate(procs):
        log.info('Starting process %d', i)
        p.start()
        p.join()
    log.info('All procs started')

def target(x, y):
    log.info('Enter')
    for item in range(y):
        time.sleep(0.1)
        log.info('(%s,%s)', x, item)
    log.info('Exit')

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

./16-Multiprocessing/queue.py

```
import time
import logging
from multiprocessing import Queue, Process

logging.basicConfig(
    level=logging.INFO,
    format='%(processName)s: %(message)s')

log = logging.getLogger()

q = Queue()

def producer():
    for x in range(10):
        time.sleep(0.5)
        log.info('>>> %s', x)
        q.put(x)
```

```
def consumer():
    while True:
        x = q.get()
        log.info('<<< %s', x)

p_producer = Process(target=producer)
p_consumer = Process(target=consumer)

p_producer.start()
time.sleep(2)
p_consumer.start()
p_producer.join()
time.sleep(0)
p_consumer.terminate()
```

./16-Multiprocessing/shared_memory.py

```
import math
import logging
from multiprocessing import Process, Value, Array

logging.basicConfig(level=logging.INFO)

log = logging.getLogger()

def main():
    num = Value('d', 0.0)
    arr = Array('i', range(10))

    log.info('Before process, num.value = %s', num.value)
    log.info('Before process, arr = %s', list(arr))

    p = Process(target=target, args=(num, arr))
    p.start()
    p.join()

    log.info('After process, num.value = %s', num.value)
    log.info('After process, arr = %s', list(arr))

def target(num, arr):
    log.info('Running target function')
    num.value = math.pi
    for i, aval in enumerate(arr):
        arr[i] = -aval

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

./17-Subprocess/

./17-Subprocess/run_command.py

```
import os
import subprocess

for filename in os.listdir('.'):
    print subprocess.check_output(['stat', filename])
```

./17-Subprocess/run_pipeline.py

```
from subprocess import Popen, PIPE

sp1 = Popen(['ls', '-laR'], stdin=PIPE, stdout=PIPE)
sp2 = Popen(['wc', '-l'], stdin=sp1.stdout, stdout=PIPE)
sp1.stdin.close()
stdout, stderr = sp2.communicate()
print '%s lines' % stdout.strip()
```

./18-Virtualenv/

./18-Virtualenv/MyDistutilsProject/MANIFEST

```
# file GENERATED by distutils, do NOT edit
setup.py
test-script
mydistutilsproject/__init__.py
```

./18-Virtualenv/MyDistutilsProject/mydistutilsproject/init.py

```
#
def foo():
    print 'bar'
```

./18-Virtualenv/MyDistutilsProject/setup.py

```
from distutils.core import setup

version = '0.0'

setup(name='MyDistutilsProject',
      version=version,
      description="",
      long_description="""\
""",
      classifiers=[], # Get strings from http://pypi.python.org/pypi?%3Aaction=list_classifiers
      keywords='',
      author='',
      author_email='',
      url='',
      license='',
      packages=['mydistutilsproject'],
      include_package_data=True,
      zip_safe=False,
      scripts=['test-script'],
      install_requires=[
          # -*- Extra requirements: -*-
      ],
      )
```

./18-Virtualenv/MyDistutilsProject/test-script

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

```

```
3 print 'This is a test script'
```

./18-Virtualenv/MySetuptoolsProject/mysetuptoolsproject/init.py

```
#
```

./18-Virtualenv/MySetuptoolsProject/setup.cfg

```
[egg_info]
tag_build = dev
tag_svn_revision = true
```

./18-Virtualenv/MySetuptoolsProject/setup.py

```
from setuptools import setup, find_packages
import sys, os

version = '0.0'

setup(name='MySetuptoolsProject',
      version=version,
      description="",
      long_description="""\
""",
      classifiers=[], # Get strings from http://pypi.python.org/pypi?%3Aaction=list_classifiers
      keywords='',
      author='',
      author_email='',
      url='',
      license='',
      packages=find_packages(exclude=['ez_setup', 'examples', 'tests']),
      include_package_data=True,
      zip_safe=False,
      install_requires=[
          # -*- Extra requirements: -*-
      ],
      entry_points="""
      # -*- Entry points: -*-
      """,
      )
```

./19-Testing/

./19-Testing/directory.py

```
class Directory(object):

    def __init__(self):
        self._directory = {}

    def add_number(self, name, number):
        self._directory[name] = number

    def remove_number(self, name):
        del self._directory[name]
```

```
    def lookup_number(self, name):
        return self._directory[name]

    def __repr__(self):
        l = ['<Directory>']
        for name, number in self._directory.items():
            l.append('    %s: %s' % (name, number))
        l.append('</Directory>')
        return '\n'.join(l)
```

./19-Testing/simple_math.py

```
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    return a / b
```

./19-Testing/test1.py

```
import unittest

class MyTest(unittest.TestCase):

    def test_pass(self):
        pass

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

./19-Testing/test2.py

```
import unittest

class MyTest(unittest.TestCase):

    def test_fail(self):
        assert False

    def test_fail_message(self):
        assert False, 'This is an assertion message'

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

./19-Testing/test3.py

```
import unittest
import simple_math

class MyTest(unittest.TestCase):
```

```
    def test_one_and_one(self):
        self.assertEqual(simple_math.add(1, 1), 2)

    def test_one_and_one_fail(self):
        self.assertEqual(simple_math.add(1, 1), 4)

    def test_one_and_one_fail_assert(self):
        assert simple_math.add(1,1) == 4

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

./19-Testing/test4.py

```
import unittest
import simple_math

class MyTest(unittest.TestCase):

    def test_one_and_one(self):
        self.assertRaises(ZeroDivisionError, simple_math.divide, 1, 0)

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

./19-Testing/test5.py

```
import unittest

class MyTest(unittest.TestCase):

    def test_pass(self):
        pass

    def test_fail(self):
        assert False

    def test_also_fail(self):
        raise AssertionError

    def test_error(self):
        raise ValueError

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

./19-Testing/test6.py

```
import unittest
from simple_math import add, subtract, multiply, divide

class MyTest(unittest.TestCase):

    def setUp(self):
        self.x = 1
        self.y = 1
```

```
    def tearDown(self):
        pass

    def test_add(self):
        self.assertEqual(add(self.x, self.y), 2)

    def test_subtract(self):
        self.assertEqual(subtract(self.x, self.y), 0)

    def test_multiply(self):
        self.assertEqual(multiply(self.x, self.y), 1)

    def test_divide(self):
        self.assertEqual(divide(self.x, self.y), 1)

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

./19-Testing/test7.py

```
import unittest

class MyTest(unittest.TestCase):

    def test_docstring(self):
        "This is a test docstring. It should say what's being tested."
        pass

    def test_no_docstring(self):
        pass

    def test_docstring_fail(self):
        "This is a test docstring. It should say what's being tested."
        assert False

    def test_no_docstring_fail(self):
        assert False

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

./19-Testing/test8.py

```
import unittest
import mock

def echo_data(socket):
    data = socket.recv()
    socket.send(data)

class MyTest(unittest.TestCase):

    def test_send_recv(self):
        socket = mock.Mock()
        socket.recv.return_value = 'Some data'
        echo_data(socket)
        socket.send.assert_called_with('Some data')

if __name__ == '__main__':
```

```
    unittest.main()
```

./19-Testing/test9.py

```
import doctest

def average(values):
    """Computes the arithmetic mean of a list of numbers.

    >>> print average([20, 30, 70])
    40.0
    """
    return sum(values, 0.0) / len(values)

if __name__ == '__main__':
    doctest.testmod()   # automatically validate the embedded tests
```

./19-Testing/test_directory.py

```
import unittest
import directory

class TestEmptyDirectory(unittest.TestCase):

    def setUp(self):
        self.d = directory.Directory()

    def test_add_number(self):
        self.d.add_number('name', '111.111.1111')
        self.assertEqual(
            self.d.lookup_number('name'),
            '111.111.1111')

    def test_lookup_unknown_number(self):
        self.assertRaises(KeyError, self.d.lookup_number, 'name')

    def test_remove_unknown_number(self):
        self.assertRaises(KeyError, self.d.remove_number, 'name')

    def test_repr_has_two_lines(self):
        d_repr = repr(self.d)
        self.assertEqual(len(d_repr.splitlines()), 2)

class TestNonemptyDirectory(unittest.TestCase):

    def setUp(self):
        self.d = directory.Directory()
        self.d.add_number('name', '111.111.1111')

    def test_lookup_number(self):
        self.assertEqual(
            self.d.lookup_number('name'),
            '111.111.1111')

    def test_remove_number(self):
        self.d.remove_number('name')
        self.assertRaises(KeyError, self.d.lookup_number, 'name')

    def test_repr_has_three_lines(self):
```

```
        d_repr = repr(self.d)
        self.assertEqual(len(d_repr.splitlines()), 3)

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

./19-Testing/testa.py

```
import doctest

def average(values):
    """Computes the arithmetic mean of a list of numbers.

    >>> print average([20, 30, 70])
    40.0
    """
    return sum(values, 0.0) / len(values)

if __name__ == '__main__':
    doctest.testmod()   # automatically validate the embedded tests
```