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