# Interacting with the OS & Filesystem

- Making OS calls
    - Executing shell commands
    - Environment variables
    - Working with paths
- File Operations
    - Read, write, append & modes
- Specialzed filetypes
    - JSON
    - CSV
    - XML / HTML
- Serialization with `pickle`

## Making OS Calls

Sometimes we'd like to be able to make system calls from Python. Giant list of availiable operations here (https://docs.python.org/2/library/os.html).

In [8]:

```
import os

print "Current working directory:", os.getcwd()
print "Process ID:", os.getpid()
```

```
Current working directory: /Users/will/Code/develop_intelligence/notebo
oks
Process ID: 47075
```

### Executing Shell Commands from Python

Be careful!

In [20]:
```python
import subprocess

# hello world
print subprocess.check_output(["echo", "Hello World!"])

# ls -la
output = subprocess.check_output(['ls', '-la'])
print output
```

```
Hello World!

total 15304
drwxr-xr-x  16 will  staff      544 Apr  8 13:31 .
drwxr-xr-x  20 will  staff      680 Apr  8 04:39 ..
-rw-r--r--@  1 will  staff     6148 Apr  8 13:31 .DS_Store
drwxr-xr-x  13 will  staff      442 Apr  8 04:55 .ipynb_checkpoints
-rw-r--r--   1 will  staff    26188 Apr  8 01:03 01_Syntax.ipynb
-rw-r--r--   1 will  staff    38240 Apr  8 02:03 02_Data_Structures.ipy
nb
-rw-r--r--   1 will  staff    19373 Apr  8 02:39 03_Functions_and_Scop
e.ipynb
-rw-r--r--   1 will  staff    22131 Apr  8 02:53 04_Classes and OOP.ipy
nb
-rw-r--r--   1 will  staff     7436 Apr  8 02:56 05_Exception Handling.
ipynb
-rw-r--r--   1 will  staff    26465 Apr  8 13:12 06_Strings and Regex.i
pynb
-rw-r--r--   1 will  staff    11795 Apr  8 03:19 07_Developer Tools.ipy
nb
-rw-r--r--   1 will  staff    21252 Apr  8 13:31 08_Interacting with th
e OS & Filesystem.ipynb
-rw-r--r--   1 will  staff    12446 Apr  8 11:44 09_Making HTTP Request
s.ipynb
-rw-r--r--   1 will  staff     8341 Apr  8 11:35 10_Scripting and Comma
nd Line Arguments.ipynb
-rw-r--r--@  1 will  staff  3837876 Apr  8 00:07 Python Setup and Virtu
alenvs.key
-rw-r--r--@  1 will  staff  3770731 Apr  8 02:55 Why Python.key
```

## Environment Variables

In bash on Unix:

```
$ export ENV=DEVELOPMENT
```

Changing environment variables in Python is easy.

```
In [33]: import os

         # set environment variable
         os.environ["ENV"] = "DEVELOPMENT"

         # get environment variable value
         print os.environ.get("ENV", None)

         DEVELOPMENT
```

## Working with Paths

In order to work on all platforms (Windows, OSX, Linux), you should avoid making filepaths like so:

```
In [ ]: filepath = "directory_name" + "/" + "folder" + "/" + "file.txt"
```

Instead, you can achieve cross platform compatibility by constructing all your paths with `os.path.join()`:

```
In [20]: import os

         # take variable number of positional args
         filepath = os.path.join('directory_name', 'folder', 'file.txt')
         print filepath

         # note that you need the leading slash
         usrbin = os.path.join('/', 'usr', 'bin')
         print usrbin

         directory_name/folder/file.txt
         /usr/bin
```

# File I/O

Opening a file in Python is as easy as:

```
filename = os.path.join('files', 'notes.txt')
reading_mode = 'r'
f = open(filename, reading_mode)
print f.read()  # entire contents of the file
f.close()  # need to close file handle
```

## A Safer Way

We actually should be more careful - we don't want people forgetting to close files or leaving system in a bad state. For this, Python uses context managers which take care of opening, closing, and generally safely executing operations.

In [34]:
```python
import os

filename = os.path.join('..', 'files', 'notes.txt')
reading_mode = 'r'

print "Opening: %s" % filename
with open(filename, reading_mode) as f:
    print f.read()  # entire contents of the file

print "Is file closed?", f.closed
```

```
Opening: ../files/notes.txt
### NOTES ###
Super important notes
I hope someone reads these

Is file closed? True
```

## File `open()` modes

We need to tell Python how we've intending to use the file.

```
open(<file name>, <mode>)
```

Where mode is an `r` (read), `w` (write), or `a` (append), optionally followed by a + sign.

Here's a helpful (and exaustive) chart:

```
                  | r   r+   w   w+   a   a+
------------------|--------------------------
read              | +   +        +        +
write             |     +    +   +    +   +
create            |          +   +    +   +
truncate          |          +   +
position at start | +   +    +   +
position at end   |                   +   +
```

## Caution!

This line:

```
open("notes.txt", "w")
```

*will* erase the current (if any) contents of your file!

Be careful to use `a` for append to add to a file.

## File Operations with `shutil`

Python's `os` library is powerful, but Python also includes as a built-in a library called `shutil` which is a slightly higher-level API to do popular file operations like:

- Moving entire folders recursively
- Deleting folders recursively
- Creating compressed archives
- Moving files from place to place
- Copy/paste with file permissions

## `shutil` API examples

```
import shutil
```

- `copytree(src, dst, symlinks=False, ignore=None)` (https://docs.python.org/2/library/shutil.html#shutil.copytree) - copy `src` to `dest`, can handle simlinks and a custom function `ignore` that can filter files on a criteria
- `rmtree(path[, ignore_errors[, onerror]])` (https://docs.python.org/2/library/shutil.html#shutil.rmtree) - delete recursively a directory, can ignore errors or special action on an error
- `move(src, dst)` (https://docs.python.org/2/library/shutil.html#shutil.move) - move directory, recursive

## Filtering files with `glob`

Another simple built-in library that will make your life easier! `glob` is perfect for finding files based on a criteria or pattern:

In [19]:
```
import glob
import os

# files starting with 'example'
filepaths = [os.path.abspath(x) for x in glob.glob("../files/example*")]
print filepaths

# JSON files
print glob.glob("../files/*.json")

# for more complex patterns, use regex and os.walk
```

```
['/Users/will/Code/develop_intelligence/python/files/example.json', '/U
sers/will/Code/develop_intelligence/python/files/example.yaml']
['../files/example.json']
```

## Specialized Filetypes

We'll cover very briefly how to deal with these file formats.

- JSON
- CSV
- XML / HTML

### JSON files with `json`

Opening a JSON file can be quite easy.

Methods of `json` module:

- json.load() - loads JSON from file object
- json.loads() - loads JSON from string
- json.dump() - dumps JSON into a file
- json.dumps() - dumps JSON into a string

### Loading JSON from File with `json.load()`

In [45]:
```
import json

path = os.path.join('..', 'files', 'example.json')
with open(path, 'r') as jf:
    data = json.load(jf)

print data
```

```
{u'classname': u'Python', u'topics': [u'Syntax', u'OOP', u'Data Structu
res'], u'number_students': 25}
```

### Creating JSON String with `json.dumps()`

In [42]:
```
import json

gold_medals = {
    "USA" : 50,
    "China" : 43,
    "Russia" : 42,
}

print json.dumps(gold_medals)
```

```
{"China": 43, "Russia": 42, "USA": 50}
```

## CSVs

Here we'll briefly talk about CSVs, but there are much more powerful tools (`pandas`, `numpy`) that allow for much, much more functionality that we talk about in the higher level courses.

### Example: Reading CSV with Fixed Columns

In [48]:
```
import csv, os

"""
File with CSV header:

first_name,last_name,zipcode
"""
path = os.path.join('..', 'files', 'names.csv')

with open(path, 'r') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        print "First: %s, last: %s, zipcode: %s" % (
            row['first_name'], row['last_name'], row['zipcode'])
```

```
First: John, last: Smith, zipcode: 89932
First: Mary, last: Jane, zipcode: 36788
First: Dale, last: Johnson, zipcode: 12999
```

### Example: Reading CSV with Variable Columns

In [51]:
```
import os, csv

path = os.path.join('..', 'files', 'time_series.csv')

with open(path, 'r') as csvf:
    reader = csv.reader(csvf, delimiter=';')
    for row_as_list in reader:
        print row_as_list
```

```
['2', '4', '6', '4', '4', '6', '3', '2', '13', '4', '5', '7']
['3', '4', '2', '1', '45', '7', '8', '6', '4']
['3', '2', '3']
['1', '1', '1', '4', '5', '6']
```

## Serialization (Pickling) with `pickle`

Why serialize Python objects?

- Saving objects for reuse
- Creating of objects is very costly in terms of time or compute
- Faster loading than JSON, CSV, etc

When NOT to serialize:

- Cross platform compatibility (it's not!)
- Serializing `lambda` functions

### `cPickle` vs. `pickle`

There are actually two different modules in Python. `cPickle` is the C-optimized version of `pickle` and can be up to 1000x faster.

For most applications, they are interchangeable, and you should use the following:

In [1]:
```
import cPickle as pickle
```

And then if you need to change later for a, custom pickler, for example, you can do so and only change the single import line.

### Creating a Serialized File

In [4]:
```
import cPickle as pickle
import os

data = {
    'text' : 'This is some text',
    'numbers' : [1, 2, 4, 7, 2, 3, 8],
    'python_only' : [True, False, None]
}

path = os.path.join('..', 'files', 'test.pkl')

with open(path, 'w') as pickle_file:
    pickle.dump(data, pickle_file)
```

### Loading from a Serialized File

In [6]:
```
import cPickle as pickle
import os

path = os.path.join('..', 'files', 'test.pkl')
with open(path, 'r') as pickle_file:
    data = pickle.load(pickle_file)

print data
print data['python_only']
```

```
{'python_only': [True, False, None], 'text': 'This is some text', 'numb
ers': [1, 2, 4, 7, 2, 3, 8]}
[True, False, None]
```

### Pickling API

- pickle.dump(obj, file_handle) - serializes obj into file handle
- pickle.dumps(obj) - serialized obj into raw string
- pickle.load(file_handle) - loads file contents into Python object
- pickle.loads(string) - loads Python object from raw serialized string

## YAML files

First, let's install a YAML parsing library:

```
$ pip install pyyaml
```

In [3]:
```
!cat ../files/example.yaml
```

```
invoice: 34843
date   : 2001-01-23
bill-to: &id001
    given  : Chris
    family : Dumars
    address:
        lines: |
            458 Walkman Dr.
            Suite #292
        city    : Royal Oak
        state   : MI
        postal  : 48046
```

```
In [12]: import yaml

         with open('../files/example.yaml', 'r') as yf:
             obj = yaml.load(yf)

         obj
```

```
Out[12]: {'bill-to': {'address': {'city': 'Royal Oak',
             'lines': '458 Walkman Dr.\nSuite #292\n',
             'postal': 48046,
             'state': 'MI'},
            'family': 'Dumars',
            'given': 'Chris'},
           'date': datetime.date(2001, 1, 23),
           'invoice': 34843}
```

## Wrap-Up

- Making OS calls
  - Executing shell commands
  - Environment variables
  - Working with paths
- File Operations
  - Read, write, append & modes
- Specialzed filetypes
  - JSON
  - CSV
  - XML / HTML
- Serialization with `pickle`