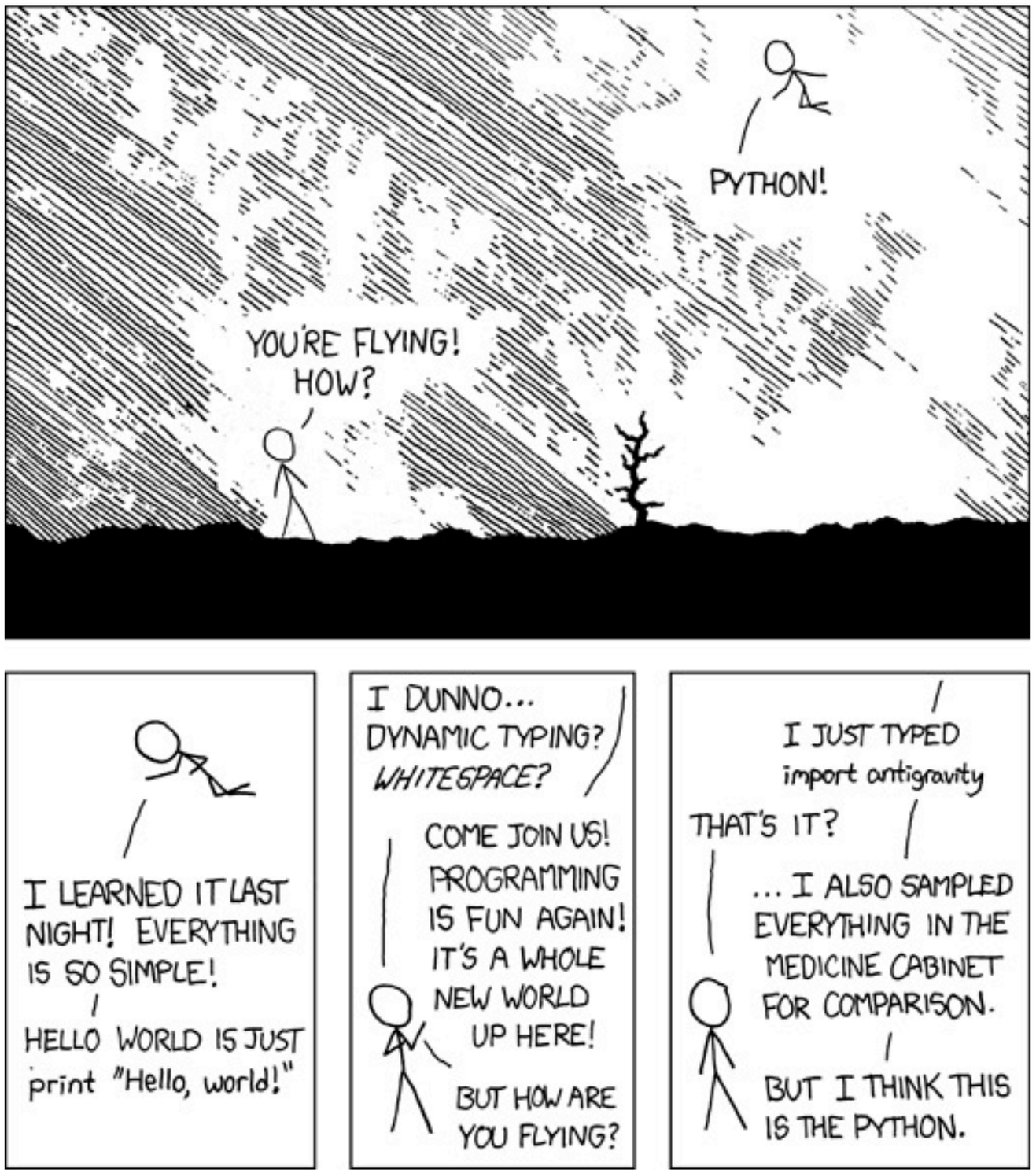
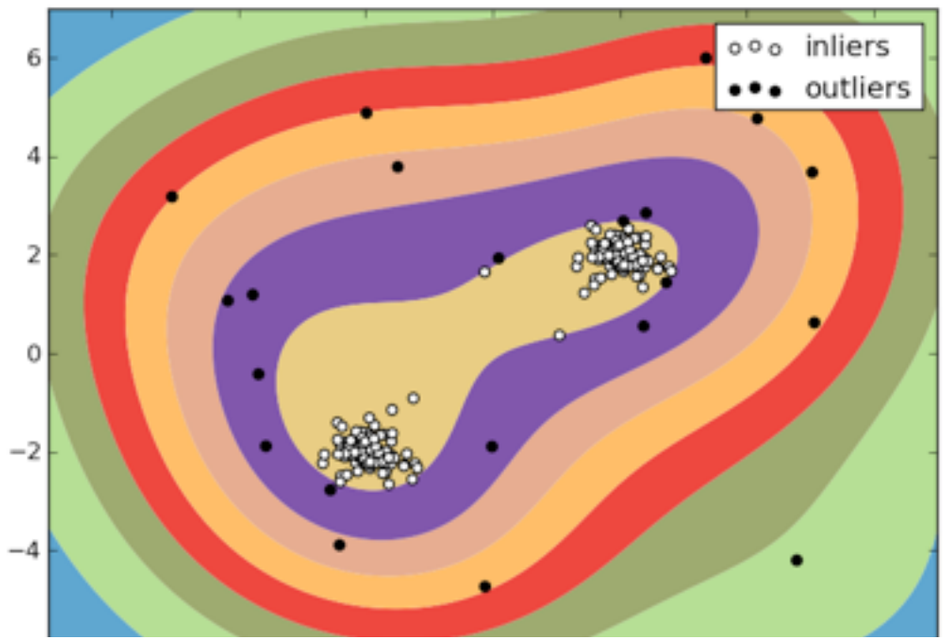
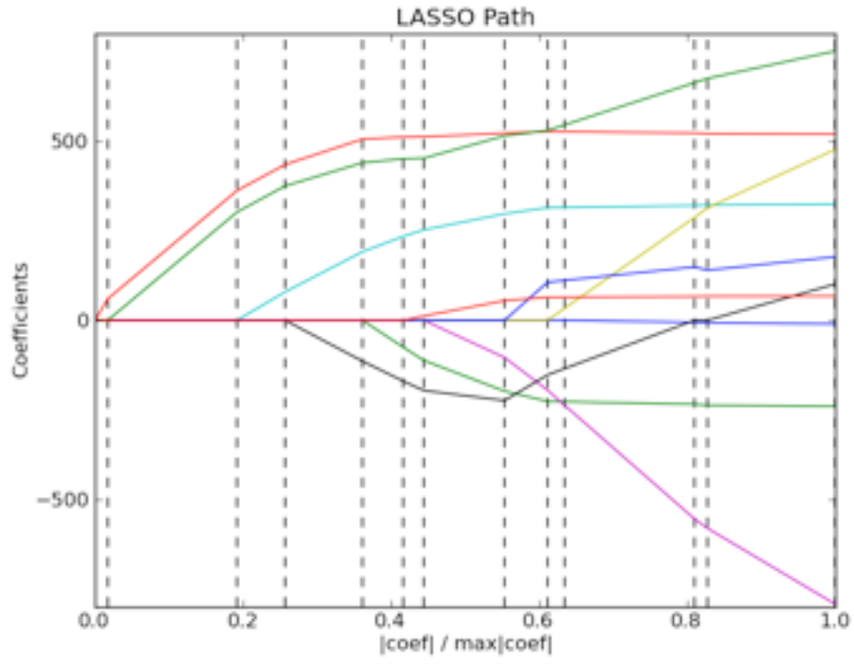
$ easy_install scikit-learn
from scikits.learn import svm

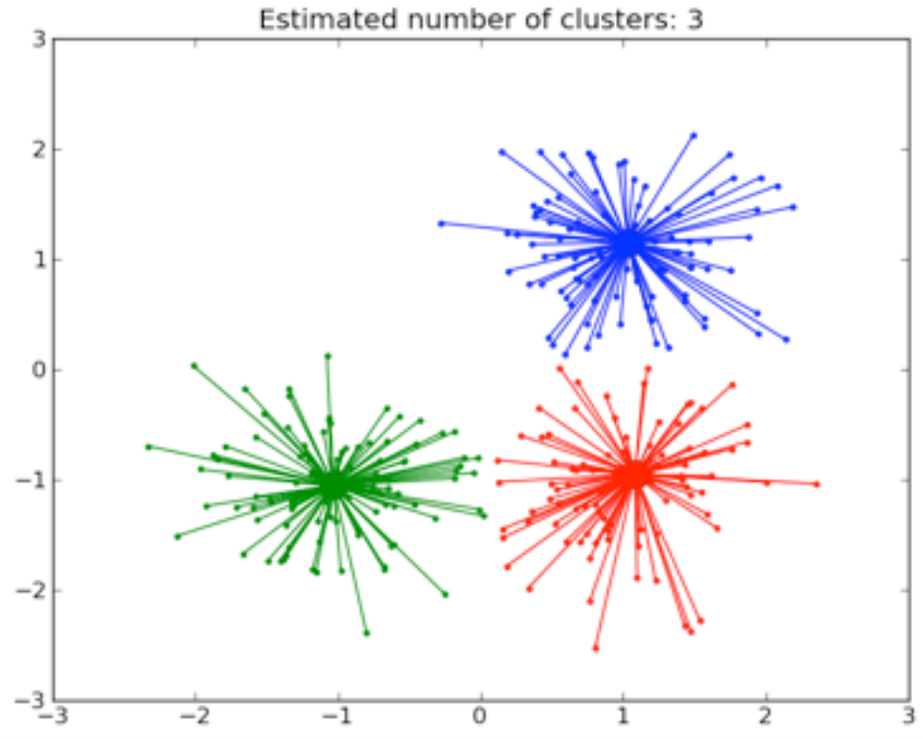
Shouyuan Chen


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

scikits.learn







Advantages

- Many useful model
- Unified API for various ML algorithms
- Very clean source code

Features

- Supervised learning
  - SVM, LASSO, GP
- Unsupervised learning
  - GMM, HMM, Clustering, MF
- Utilities
  - cross-validation, grid-search

SVM

```
from scikits.learn import svm
X = [[0, 0], [1, 1]]
Y = [0, 1]
clf = svm.SVC() # rbf kernel
clf.fit(X, Y) # train the model
print clf.predict([[.2,.2]])
  # output: 0
```

SVM

```
from scikits.learn import svm
X = [[0, 0], [1, 1]]
Y = [0, 1]
clf = svm.SVC() # rbf kernel
clf.fit(X, Y) # train the model
print clf.predict([[.2,.2]])
  # output: 0
```

Multiclass SVM

```
X = [[0], [1], [2], [3]]
Y = [0, 1, 2, 3]
clf = svm.SVC()
clf.fit(X, Y)
```

SVM

```
from scikits.learn import svm
X = [[0, 0], [1, 1]]
Y = [0, 1]
clf = svm.SVC() # rbf kernel
clf.fit(X, Y) # train the model
print clf.predict([[.2,.2]])
  # output: 0
```

Multiclass SVM

```
X = [[0], [1], [2], [3]]
Y = [0, 1, 2, 3]
clf = svm.SVC()
clf.fit(X, Y)
```

Linear SVM

```
lin_clf = svm.LinearSVC()
lin_clf.fit(X, Y)
```

SVM

libsvm

```
from scikits.learn import svm
X = [[0, 0], [1, 1]]
Y = [0, 1]
clf = svm.SVC() # rbf kernel
clf.fit(X, Y) # train the model
print clf.predict([[.2,.2]])
  # output: 0
```

Multiclass SVM

```
X = [[0], [1], [2], [3]]
Y = [0, 1, 2, 3]
clf = svm.SVC()
clf.fit(X, Y)
```

Linear SVM

```
lin_clf = svm.LinearSVC()
lin_clf.fit(X, Y)
```

SVM

```
from scikits.learn import svm
X = [[0, 0], [1, 1]]
Y = [0, 1]
clf = svm.SVC() # rbf kernel
clf.fit(X, Y) # train the model
print clf.predict([[.2,.2]])
  # output: 0
```

libsvm

Multiclass SVM

```
X = [[0], [1], [2], [3]]
Y = [0, 1, 2, 3]
clf = svm.SVC()
clf.fit(X, Y)
```

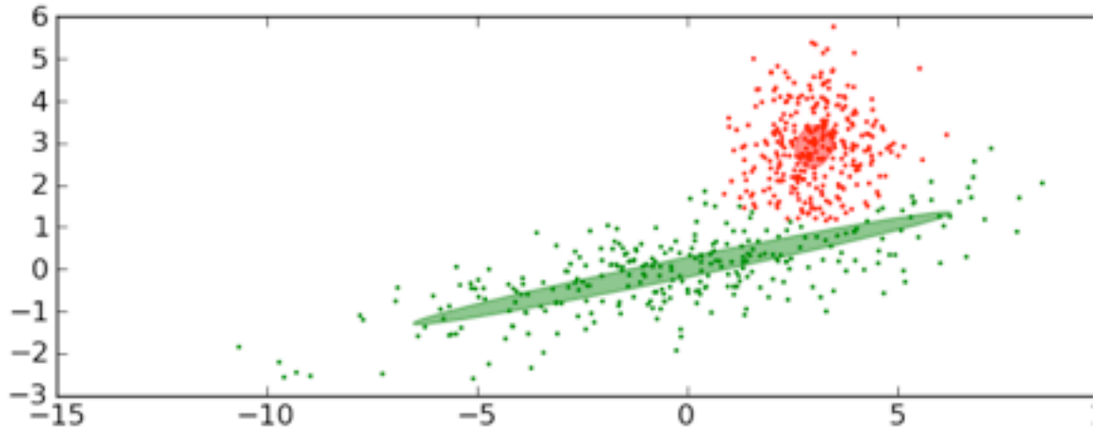
Linear SVM

liblinear

```
lin_clf = svm.LinearSVC()
lin_clf.fit(X, Y)
```

Gaussian Mixture Model

```
import numpy as np
from scikits.learn import mixture
n, m = 300, 2

# generate random sample, two components
np.random.seed(0)
C = np.array([[0., -0.7], [3.5, .7]])
X = np.r_[np.dot(np.random.randn(n, 2), C),
          np.random.randn(n, 2) + np.array([3, 3])]

clf = mixture.GMM(n_states=2, cvtype='full') # spherical/full/etc
clf.fit(X)

Y_ = clf.predict(X) # clustering results


import pylab as pl
import matplotlib as mpl

for i, (mean, covar) in enumerate(zip(clf.means, clf.covars)):
    # plot stuff..
```



Cross Validation

```
import numpy as np
from scikits.learn.cross_val import LeaveOneOut, LeavePOut, KFold
X = np.array([[0., 0.], [1., 1.], [-1., -1.], [2., 2.]])
Y = np.array([0, 1, 0, 1])


# Leave One Out
loop = LeaveOneOut(len(Y))

# Leave P Out
loop = LeavePOut(len(Y), 2)

# K-Fold
loop = KFold(len(Y), 2)


# do experiments
for train, test in loop:
  X_train, Y_train = X[train], Y[train]
  X_test, Y_test = X[test], Y[test]
```

Unified API

for all ML algorithms, [un|semi]supervised

Estimator

the basic object, implements:

```
estimator = obj.fit(data)
estimator.refit(new_data) # online
```

Predictor

For supervised learning, or some unsupervised problems, implements:

```
prediction = obj.predict(data)
```

Transformer

For filtering or modifying the data, in a supervised or unsupervised way, implements:

```
new_data = obj.transform(data)
```

Model

A model that can give a goodness of fit or a likelihood of unseen data, implements (higher is better):

```
score = obj.score(data)
```

Useful Links

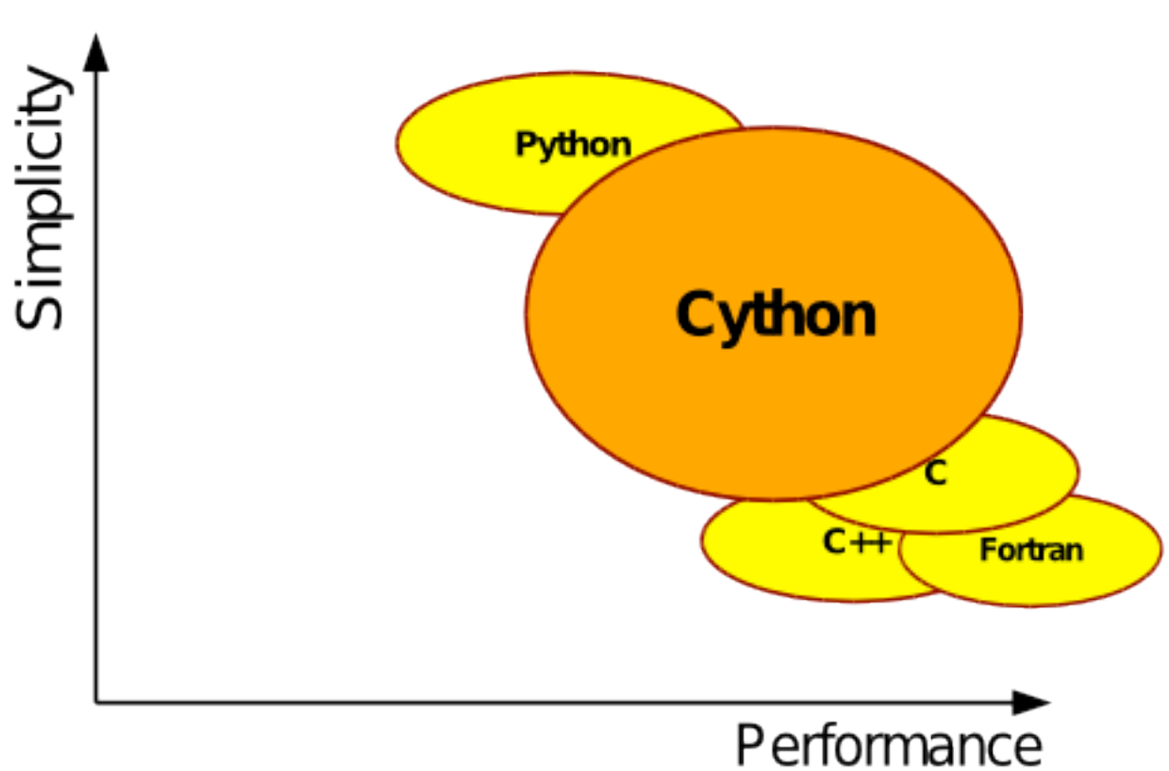
- Homepage
  - http://scikit-learn.sourceforge.net/
- Source code
  - https://github.com/scikit-learn/scikit-learn
- Statistical Machine Learning for Text Classification with scikit-learn and NLTK [PyCon2011]
  - http://www.slideshare.net/ogrisel/statistical-machine-learning-for-text-classification-with-scikitlearn-and-nltk

PerformancePy

- Pure Python is slow ...
  - CPython is 1000 times slower that C in some benchmark
    - On average CPython is 50 times slower than C
- Optimizations
  - Psyco, PyPy, **Cython**, ...

What is Cython

- *"Cython is the missing link between the **simplicity** of Python and the **speed** of C"*



What is Cython

Cython is

- an Open source project
- almost a Python compiler
- an extended Python language for
  - ★ writing fast native Python extension
  - ★ interfacing Python with C libraries

How to use Cython

- you write Python code
  - Cython translates it into C code
  - your C compiler builds a shared library for CPython
  - you import your module into CPython
  - **then 20~80% gain!**
- Cython has support for
  - distutils
    - *optionally* compile Python code from setup.py!
  - embedding the CPython runtime in an executable

much higher gain is possible.

Adding type declarations

- Cython supports **optional** type declarations that
  - can be employed exactly where performance matters
  - let Cython generate **plain C** instead of C-API calls
    - can make code **100 - 1000x** faster than CPython
      - expect several 100 times in calculation loops
- With all type information
  - Cython = C with Python syntax

A Simple Example

```
# integrate_py.py

from math import sin


def f(x):
    return sin(x**2)

def integrate_f(a, b, N):
    dx = (b-a)/N
    s = 0
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

A Simple Example

```python
# integrate_py.py

from math import sin


def f(x):
    return sin(x**2)

def integrate_f(a, b, N):
    dx = (b-a)/N
    s = 0
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

```python
# integrate_cy.pyx

cdef extern from "math.h":
    double sin(double x)

cdef double f(double x):
    return sin(x**2)

cpdef double integrate_f(double a,
                         double b, int N):
    cdef double dx, s
    cdef int i

    dx = (b-a)/N
    s = 0
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

My Workflow

- prototype using pure Python
  - (optional) unit testing
  - (optional) profiling
- Translate time consuming part into Cython
  - adding type information
- Completely rewrite key data structure
  - in Cython or C

Scripting

Regex

Formatted Files

Crawler

Network Services

Databases

.....

Formatted Files

XML/HTML	lxml, pyquery
YAML	PyYAML
Matlab .m	scipy.io
CSV, JSON	Standard Library

lxml

- DOM parser
- SAX parser
- XPath
- Pretty serialization

```
from lxml import etree

# parse (dom)
root = etree.fromstring(
  """
  <root>
    <a> <b/> </a>
    <b> abc </b>
  </root>
  """)

# dom
print [child.tag for child.root]  #=> ['a', 'b']

# xpath
result = root.xpath('//b')
print result[0].tag # => 'b'
print result[1].text # = > 'abc'
```

YAML/JSON

- **much better** interchange/serialization format than XML
  - faster, smaller
  - human readable/editable
  - adopted by open-source community

```xml
<People>
  <Person>
    <name>Some Name</name>
    <age>15</age>
    <sex>Male</male>
  </Person>
  <Person>
    <name>Other Name</name>
    <age>13</age>
    <sex>Female</male>
  </Person>
</People>
```

XML

```yaml
- name: Some Name
  age: 15
  sex: Male
- name: Other Name
  age: 13
  sex: Female
```

YAML

```json
[{"name": "Some Name",
  "age": 15,
  "sex":"Male"},
 {"name": "Other Name",
  "age": 13,
  "sex": "Female"} ]
```

JSON

- Python has many incrediblely useful tools
  - databases ... web servers ... crawlers ...
- Find them at pypi.python.org and github