# Scientific Computing with Python

Quick Introduction

# Libraries and APIs

- A library is a collection of implementations of behavior (definitions)
- An Application Programming Interface (API) describes that behavior (declarations)

Declaration

```
int max(int a, int b);
```

Definition

```
int max(int a, int b) {
    if (a > b) { return a; }
    else { return b; }
}
```

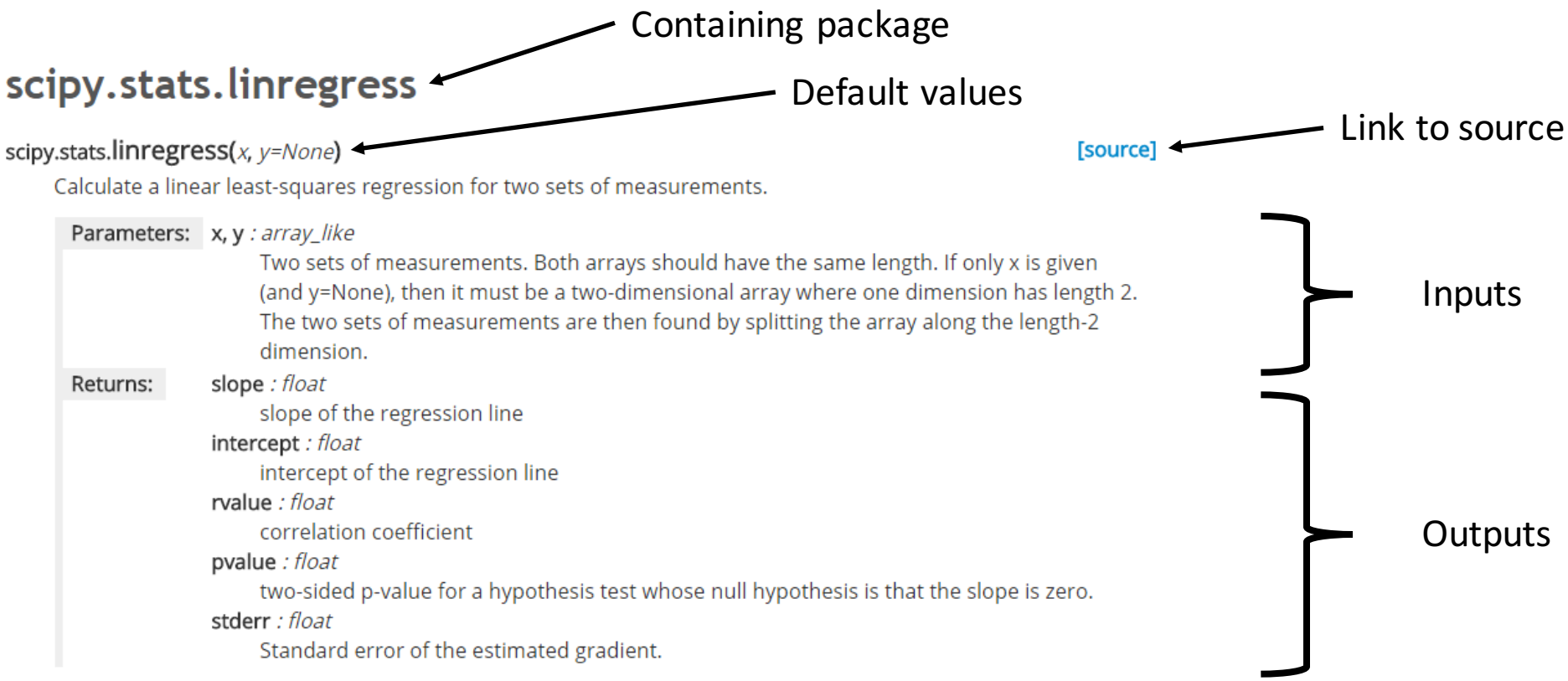
- Python only has declarations
- In Python, libraries are organized into packages, which are referenced using the `import` statement
- The same interfaces can be implemented by multiple libraries
  - Dill vs. Pickle
  - pyFFTW vs. scipy.fftpack
- Most Python libraries installed using `pip` command
- E.g. `$sudo pip install pandas`

# When to use a library

- Programmer time (your time) is a nonrenewable resource
- Biology is hard – computing should FREE YOUR MIND for biological problems
- Use a library if you think the task has been done before (e.g. parsing most file types)
- Use a library if time spent looking for and learning a library is LESS THAN the time to reimplement
- Use a library for tasks which are NOT part of your scientific contribution
- Reimplement only when there is a tangible benefit
- "Premature optimization is the root of all evil"
  - Your time is more valuable than compute time
  - Profile code before optimizing
  - `python -m cProfile -s cumtime myprogram.py`
  - Or `%%prun` in a jupyter notebook

# API Documentation

- Declarations + formatted comments can be automatically converted to documentation



# numpy + scipy + matplotlib = PYLAB

- Three most important scientific libraries in Python, with functionality similar to MATLAB
- Numpy is usually imported `as np` and `scipy as sp`
- Matplotlib is usually used through the pyplot interface (`matplotlib.pyplt as plt`)
- Numpy provides typed numerical arrays and array and matrix math
  - Create arrays with `np.array([])` or `np.zeros()`, `np.ones()`
  - Indexed and sliced like Python lists, element-wise arithmetic, broadcasting
  - Logical operations (`< > & | ~`) on arrays produce binary masks that function as indices

```
>>> a = np.array([1, 3, 0])
>>> b = np.array([0, 3, 2])
>>> a > b
array([ True, False, False], dtype=bool)
```

  - Convert from logical index to numerical index using `where`
- Numpy also provides many functions like `np.mean, np.std, np.max, np.argmax, np.sort, np.argsort, np.maximum,` etc.

# scipy

- Scipy contains packages with more advanced functionality built on numpy
- `sp.special`
- `sp.integrate`
- `sp.optimize`
- `sp.interpolate`
- `sp.linalg`
- `sp.spatial`
- `sp.stats`

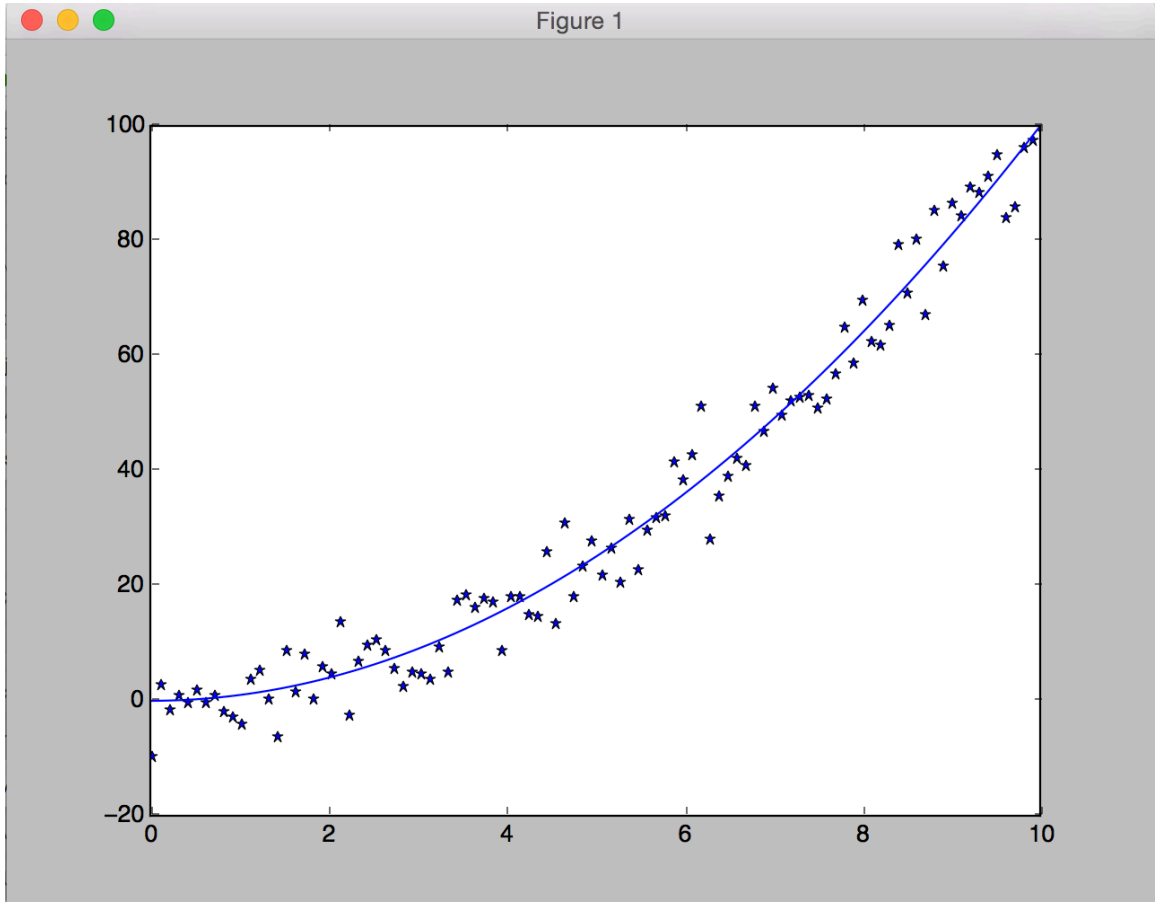
# matplotlib

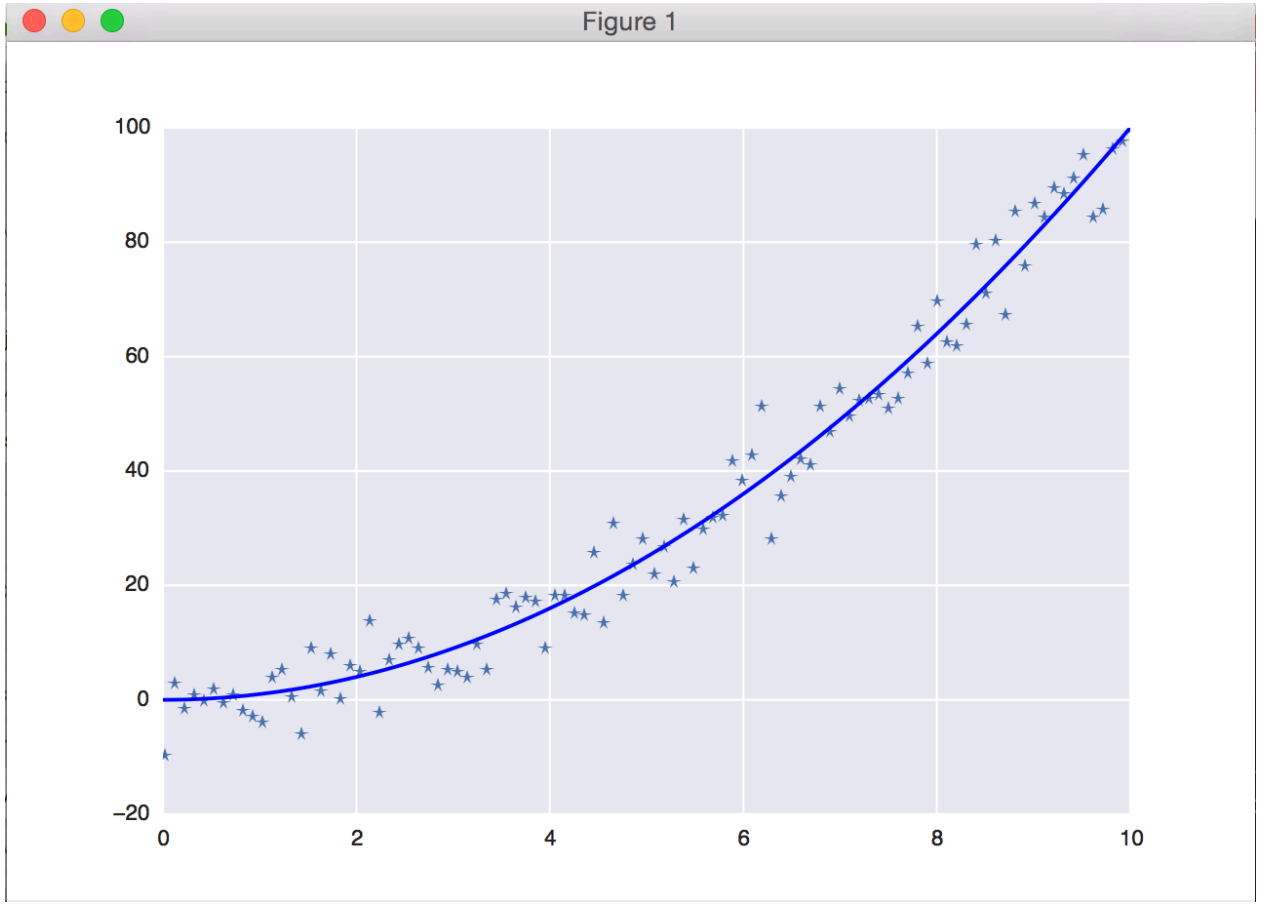
- The pyplot interface is highly similar to plotting in MATLAB
- The figure is the whole plotting window
- A figure can contain one or more axes
- The last used or selected figure is the current figure (same for axes)
- Functions like `plt.xlim, plt.xlabel, plt.set_xticklabels` modify current figure/axes
- Publication quality figures often require customization of figure size, DPI, axes labels and so forth
  - Parameters like size can be set on creation
  - Use `plt.gcf()` and `plt.gca()` to get handles to the current figure/axes

# seaborn

- Usually imported `as sns`
- Contains superior default fonts and styles for matplotlib
  - Activate with sns.set()
- Provides many short and sweet functions like `sns.regplot, sns.kdeplot, sns.heatmap`
- Good for producing example plots quickly
- Customization will usually require the same low-level matplotlib options
- Data analysis will usually require doing e.g. linear regression yourself in scipy (because seaborn doesn’t return regression results)

# seaborn: before and after



# pandas

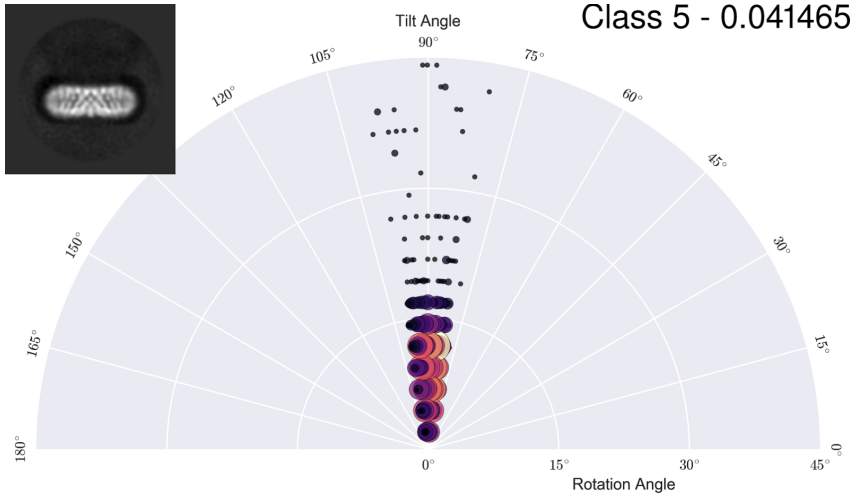
- Framework for “relational” or labelled data, import `as pd`
- Based on `pd.DataFrame` created from list of dict, dict of dict, dict of list, etc.
- Convenient parsing – `pd.read_csv(), pd.read_table()`
- Index using arbitrary labels (access a record based on the value of a field)
- Fast merging, groupby, aggregation
- Default plot labels in seaborn plots and `DataFrame.plot`, etc.
- Advanced multivariate visualization (Andrews plots and so forth)
- Real world example: `star.py and angdist.py`

# scikit & others

- Scipy toolkits (scikits) offer more specialized features from outside the main project
- `scikit-learn, scikit-image`
- Biopython has lots of useful functions for biology (`Bio.PDB, Bio.Seq,` etc.)
- Many others like `rdkit`, `opencv`, etc.

# Resources

- Documentation - http://www.scipy.org/docs.html
- Introduction to Numpy and Scipy (UCSB) - http://www.engr.ucsb.edu/~shell/che210d/numpy.pdf
- Pandas cheat sheet (U of Idaho) - http://www.webpages.uidaho.edu/~stevel/504/Pandas%20DataFrame%20Notes.pdf
- Really fancy plots – `matplotlib` & `mpl_toolkits`




```
from matplotlib.projections.polar import PolarTransform
from matplotlib.transforms import Affine2D
from mpl_toolkits.axisartist import angle_helper
from mpl_toolkits.axisartist import floating_axes
```