NumPy Basics

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:
```
>>> import numpy as np
```

I/O

Saving & Loading On Disk
```
>>> np.save('myarray.npy', a)
>>> np.load('myarray.npy')
```

Saving & Loading Text Files
```
>>> np.savetxt('myfile.txt', a, delimiter=';')
```

Data Types
```
>>> np.int64
>>> np.float32
>>> np.complex
>>> np.bool
>>> np.object
>>> np.string
>>> np.unicode
```

Creating Arrays
```
>>> a = np.array([1,2,3])
>>> b = np.array([[1,2,3], [4,5,6]])
>>> c = np.array([[1,2,3], [4,5,6]], dtype=np.int16)
```

Initial Placeholders
```
>>> np.zeros((3,4))
>>> np.ones((2,3), dtype=np.float32)
>>> np.linspace(0,2,9)
```

Array Mathematics
```
>>> np.add(a,b)
>>> np.subtract(a,b)
>>> np.multiply(a,b)
>>> np.divide(a,b)
>>> np.exp(b)
```

Array-wise comparison
```
>>> a == b
array([[ True, False, False],
       [False, False, False]], dtype=bool)
```

Comparison
```
>>> a < b
array([[ True,  True,  True],
       [False, False, False]], dtype=bool)
```

Aggregate Functions
```
>>> a.sum()
```

Selecting Elements
```
>>> a[1:3]
```

Slicing
```
>>> a[1,2]
```

Reshaping Arrays
```
>>> b.reshape(3,2)
```

Joining Arrays
```
>>> array([0, 1, 2])
```

Fancy Indexing
```
>>> a[[0,2,1,0], [0,1,2,0]]
```

Array-wise natural logarithm
```
>>> np.log(a)
```

Adding/Removing Elements
```
>>> np.append(h,g)
```

Manipulation
```
>>> np.insert(a, 1, 5)
```

Sorting Arrays
```
>>> np.sort(a)
```

Copying Arrays
```
>>> np.copy(a)
```

Sorting Arrays
```
>>> a.sort()
```

Subsetting, Slicing, Indexing
```
>>> a[2]
>>> b[1,2]
```

Adding/Removing Elements
```
>>> np.append(b, [1,2,1])
```

Changing Array Shape
```
>>> np.ravel(b)
```

Reshaping Arrays
```
>>> b.reshape(2,3)
```

Array-wise comparison
```
>>> a == b
array([[ True, False, False],
       [False, False, False]], dtype=bool)
```

Element-wise comparison
```
>>> a < b
array([[ True,  True,  True],
       [False, False, False]], dtype=bool)
```

Array-wise sum
```
>>> a.sum()
```

Array-wise minimum value
```
>>> a.min()
```

Array-wise maximum value
```
>>> a.max()
```

Array-wise cumulative sum of the elements
```
>>> a.cumsum(axis=1)
```

Array-wise mean
```
>>> a.mean()
```

Array-wise median
```
>>> a.median()
```

Array-wise correlation coefficient
```
>>> a.corrcoef()
```

Array-wise standard deviation
```
>>> np.std(a)
```

Using Indexing
```
>>> a[0:2]
```

Selecting First Two Elements
```
>>> a[0:2]
```

Selecting Elements at Index 0 and 1
```
>>> a[[0,1]]
```

Selecting All Elements at Row 0 in Column 1
```
>>> a[:,1]
```

Selecting All Elements at Row 0
```
>>> a[0, :]
```

Selecting All Elements at Row 0 and 1
```
>>> a[[0,1]]
```

Selecting Row 0 and 1
```
>>> a[0:2]
```

Selecting All Elements at Row 0 and 1
```
>>> a[[0,1]]
```

Selecting All Elements at Row 0
```
>>> a[0, :]
```

Selecting All Elements at Row 0 and 1
```
>>> a[[0,1]]
```

Selecting Row 0 and 1
```
>>> a[0:2]
```

Selecting All Elements at Row 0 and 1
```
>>> a[[0,1]]
```

Selecting Row 0 and 1
```
>>> a[0:2]
```

Selecting All Elements at Row 0 and 1
```
>>> a[[0,1]]
```

Selecting Row 0 and 1
```
>>> a[0:2]
```

Selecting All Elements at Row 0 and 1
```
>>> a[[0,1]]
```

Selecting Row 0 and 1
```
>>> a[0:2]
```

Selecting All Elements at Row 0 and 1
```
>>> a[[0,1]]
```

Selecting Row 0 and 1
```
>>> a[0:2]
```

Selecting All Elements at Row 0 and 1
```
>>> a[[0,1]]
```

Selecting Row 0 and 1
```
>>> a[0:2]
```

Selecting All Elements at Row 0 and 1
```
>>> a[[0,1]]
```

Selecting Row 0 and 1
```
>>> a[0:2]
```

Selecting All Elements at Row 0 and 1
```
>>> a[[0,1]]
```

Selecting Row 0 and 1
```
>>> a[0:2]
```

Selecting All Elements at Row 0 and 1
```
>>> a[[0,1]]
```

Selecting Row 0 and 1
```
>>> a[0:2]
```
Syntax – Creating DataFrames

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

def = pd.DataFrame(
    {
        "a": [4, 5, 6],
        "b": [7, 8, 9],
        "c": [10, 11, 12],
    }, index = [1, 2, 3])

Specify values for each column.

def = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])

Specify values for each row.

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

def = (pd.melt(df)
    .rename(columns={
        'variable': 'var',
        'value': 'val'})
    .query('val > 200'))

Data Wrangling
with pandas
Cheat Sheet
http://pandas.pydata.org

Syntax – Creating DataFrames

In a tidy data set:

Each variable is saved in its own column

Each observation is saved in its own row

Tidy Data – A foundation for wrangling in pandas

Tidy data complements pandas’s vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

def = (pd.melt(df)
    .rename(columns={
        'variable': 'var',
        'value': 'val'})
    .query('val > 200'))

Reshaping Data – Change the layout of a data set

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg', ascending=False)
Order rows by values of a column (high to low).

df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length', 'Height'])
Drop columns from DataFrame

Subset Observations (Rows)

df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.nsmallest(n, 'value')
Select and order bottom n entries.

df.nlargest(n, 'value')
Select and order top n entries.

Subset Variables (Columns)

df[['width', 'length', 'species']]
Select multiple columns with specific names.

df['width'] or df.width
Select single column with specific name.

df.filter(regex='regex')
Select columns whose name matches regular expression regex.

Logic in Python (and pandas)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equals</td>
</tr>
<tr>
<td>==</td>
<td>Equals</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equals</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical and</td>
</tr>
<tr>
<td></td>
<td>Logical or</td>
</tr>
<tr>
<td>~</td>
<td>Logical not</td>
</tr>
</tbody>
</table>

regex (Regular Expressions) Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\.</code></td>
<td>Matches strings containing a period <code>.</code></td>
</tr>
<tr>
<td><code>Length$</code></td>
<td>Matches strings ending with word 'Length'</td>
</tr>
<tr>
<td><code>Sepal</code></td>
<td>Matches strings beginning with the word 'Sepal'</td>
</tr>
<tr>
<td><code>^[1-5]$</code></td>
<td>Matches strings beginning with <code>x</code> and ending with 1,2,3,4,5</td>
</tr>
<tr>
<td><code>^\!*\$Species\*$</code></td>
<td>Matches strings except the string 'Species'</td>
</tr>
</tbody>
</table>

def.loc[:, 'x2': 'x4']
Select all columns between x2 and x4 (inclusive).

def.iloc[:, [1, 2, 5]]
Select columns in positions 1, 2, and 5 (first column is 0).

def.loc[df['a'] > 10, ['a', 'c', 'e']]
Select rows meeting logical condition, and only the specific columns.
pandas provides a large set of summary functions that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the pandas groups. When applied to a DataFrame, the result is returned as a pandas Series for each group. Examples:

- `sum()` Sum values of each object.
- `count()` Count non-NA/null values of each object.
- `median()` Median value of each object.
- `quantile([0.25, 0.75])` Quantiles of each object.
- `apply(function)` Apply function to each object.

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:
- `size()` Size of each group.
- `agg(function)` Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

- `shift(1)` Copy with values shifted by 1.
- `cumsum()` Cumulative sum.
- `cummax()` Cumulative max.
- `cummin()` Cumulative min.
- `cumprod()` Cumulative product.

pandas provides a large set of vector functions that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

- `max(axis=1)` Element-wise max.
- `min(axis=1)` Element-wise min.
- `abs()` Absolute value.
- `clip(lower=-10, upper=10)` Trim values at input thresholds.

Additional vector functions:
- `vector_function` Vector function.

### Windows

- `df.expanding()` Return an Expanding object allowing summary functions to be applied cumulatively.
- `df.rolling(n)` Return a Rolling object allowing summary functions to be applied to windows of length n.

- `df.plot.hist()` Histogram for each column
- `df.plot.scatter(x='x', y='y')` Scatter chart using pairs of points

### Handling Missing Data

- `df.dropna()` Drop rows with any column having NA/null data.
- `df.fillna(value)` Replace all NA/null data with value.
- `df.isna()` Check for NA/null values.
- `df.notna()` Check for non-NA/null values.
- `df.fillna(0)` Fill NA/null with 0.
- `df.fillna(value)` Fill NA/null with value.
- `df.dropna()` Drop rows with any column having NA/null data.
- `df.fillna(value)` Replace all NA/null data with value.
- `df.isna()` Check for NA/null values.
- `df.notna()` Check for non-NA/null values.
- `df.fillna(0)` Fill NA/null with 0.
- `df.fillna(value)` Fill NA/null with value.

### Group Data

- `df.groupby(by="col")` Return a GroupBy object, grouped by values in column named "col".
- `df.groupby(level="ind")` Return a GroupBy object, grouped by values in index level named "ind".

### Make New Columns

- `df.assign(Name=lambda df: df.Length*df.Height)` Compute and append one or more new columns.
- `df['Volume'] = df.Length*df.Height*df.Depth` Add single column.
- `pd.qcut(df.col, n, labels=False)` Bin column into n buckets.
- `pd.rolling(df.col, n).mean()` Return a Rolling object allowing summary functions to be applied cumulatively.
- `pd.expanding(df.col).mean()` Return an Expanding object allowing summary functions to be applied cumulatively.

- `df['Volume'] = df.Length*df.Height*df.Depth` Add single column.
- `pd.qcut(df.col, n, labels=False)` Bin column into n buckets.
- `pd.rolling(df.col, n).mean()` Return a Rolling object allowing summary functions to be applied cumulatively.
- `pd.expanding(df.col).mean()` Return an Expanding object allowing summary functions to be applied cumulatively.

### Combine Data Sets

- `pd.merge(df1, df2, how='inner', on='x')` Join data. Retain only rows in both sets.
- `pd.merge(df1, df2, how='outer', on='x')` Join data. Retain all values, all rows.
- `pd.merge(df1, df2, how='left', on='x')` Join matching rows from df2 to df1.
- `pd.merge(df1, df2, how='right', on='x')` Join matching rows from df1 to df2.
- `pd.merge(df1, df2, how='outer', on='x')` Join data. Retain all values, all rows.

### Plotting

- `df.plot.hist()` Histogram for each column
- `df.plot.scatter(x='x', y='y')` Scatter chart using pairs of points
- `df.plot.scatter(x='x', y='y')` Scatter chart using pairs of points

- `df.plot.hist()` Histogram for each column
- `df.plot.scatter(x='x', y='y')` Scatter chart using pairs of points
- `df.plot.scatter(x='x', y='y')` Scatter chart using pairs of points

### Cheatsheet

- `df['w'].value_counts()` Count number of rows with each unique value of variable
- `len(df)` # of rows in DataFrame.
- `df['w'].unique()` # of distinct values in a column.
- `df.describe()` Basic descriptive statistics for each column (or GroupBy).
- `pd.qcut(df.col, n, labels=False)` Bin column into n buckets.
- `pd.rolling(df.col, n).mean()` Return a Rolling object allowing summary functions to be applied cumulatively.
- `pd.expanding(df.col).mean()` Return an Expanding object allowing summary functions to be applied cumulatively.

### Summarize Data

- `df['w'].value_counts()` Count number of rows with each unique value of variable
- `len(df)` # of rows in DataFrame.
- `df['w'].unique()` # of distinct values in a column.
- `df.describe()` Basic descriptive statistics for each column (or GroupBy).
Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

**1D Data**

```python
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

**2D Data or Images**

```python
>>> data = 2 * np.random.random((10, 10))
>>> z = np.sin(x)
>>> x = np.linspace(0, 10, 100)
>>> import numpy as np
```

**Create Plot**

```python
Create Plot
>>> fig = plt.figure()
>>> ax = fig.add_subplot()
```

**Axes**

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```python
>>> ax = fig.add_subplot(211) # row-col-num
>>> ax2 = fig.add_subplot(212) # row-col-num
```

**1D Data**

```python
>>> lines = ax.plot(x,y)
>>> ax.scatter(x,y)
```

**Vector Fields**

```python
>>> lines[0].set_xdata(x)
>>> lines[0].set_ydata(y)
```

**Data Distributions**

```python
>>> ax.hist(y)
>>> ax.boxplot(y)
>>> ax.violinplot(y)
```

**Save Plot**

```python
>>> plt.savefig('foo.png')
```

**Close & Clear**

```python
>>> plt.close()
>>> plt.clf()
```

**Plot Anatomy & Workflow**

The basic steps to creating plots with matplotlib are:

1. Prepare the Data
2. Create the Plot
3. Plotting Routines
4. Customize Plot
5. Save Plot
Scikit-Learn
Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

### A Basic Example
```python
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> from sklearn.preprocessing import StandardScaler}
```