Welcome to Pandas

pandas is a Python package (library) providing fast, flexible, and expressive data structures, designed to make working with “labeled” data easy and intuitive.
It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python.

Library Highlights

- A fast and efficient **DataFrame** object for data manipulation with integrated indexing;
- Tools for **reading and writing data** between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
- Intelligent **data alignment** and integrated handling of **missing data**: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form;
- Flexible **reshaping** of data sets;
- Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** of large data sets;
- High performance **merging and joining** of data sets;
- **Hierarchical axis indexing** provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure;
- **Time series**-functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;
- Highly **optimized for performance**, with critical code paths written in Cython or C.
- Python with *pandas* is in use in a wide variety of **academic and commercial** domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

Python vs Spreadsheet

Python is an object oriented language

Value

```
1 X = 1
```

List

```
1 X = [1,2,3,4,5]
```

Dataframe

```
1 X = pd.DataFrame({'a':[1,1,1],'b':[2,2,2]})
```

Dictionary

```
1 X = {'a':[1,2,3],'b':[1,2,3,4,5],'c':[1,2]}
```

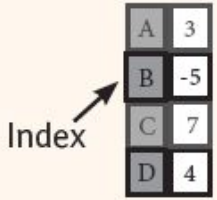
Spreadsheet style

Value	1					
List	1	2	3	4	5	6
dataframe	1	2	3	4	5	6
	1	2	3	4	5	6
	1	2	3	4	5	6
dictionary	1	2	3			
	1	2	3	4	5	
	1	2				

Pandas Data Structures

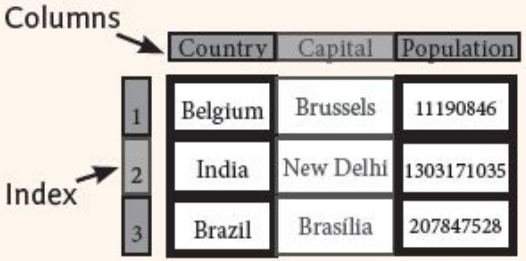
Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> pd.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 -5
```
Get one element

```
>>> df[1:]
   Country    Capital  Population
1   India  New Delhi  1303171035
2  Brazil   Brasília   207847528
```
Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc([0],[0])
 'Belgium'
>>> df.iat([0],[0])
 'Belgium'
```
Select single value by row & column

By Label

```
>>> df.loc([0], ['Country'])
 'Belgium'
>>> df.at([0], ['Country'])
 'Belgium'
```
Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
Country       Brazil
Capital     Brasília
Population  207847528
```
Select single row of subset of rows

```
>>> df.ix[:,'Capital']
0     Brussels
1    New Delhi
2     Brasília
```
Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
 'New Delhi'
```
Select rows and columns

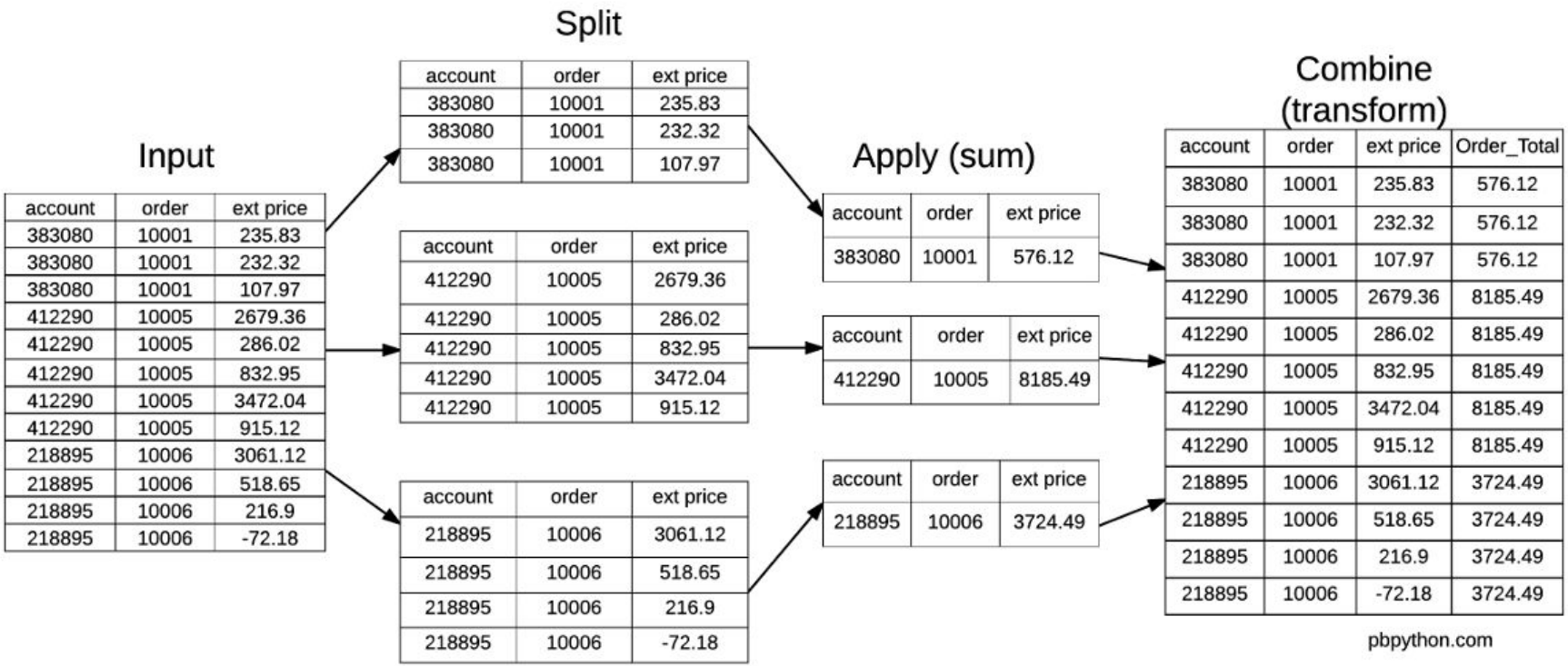
Boolean Indexing

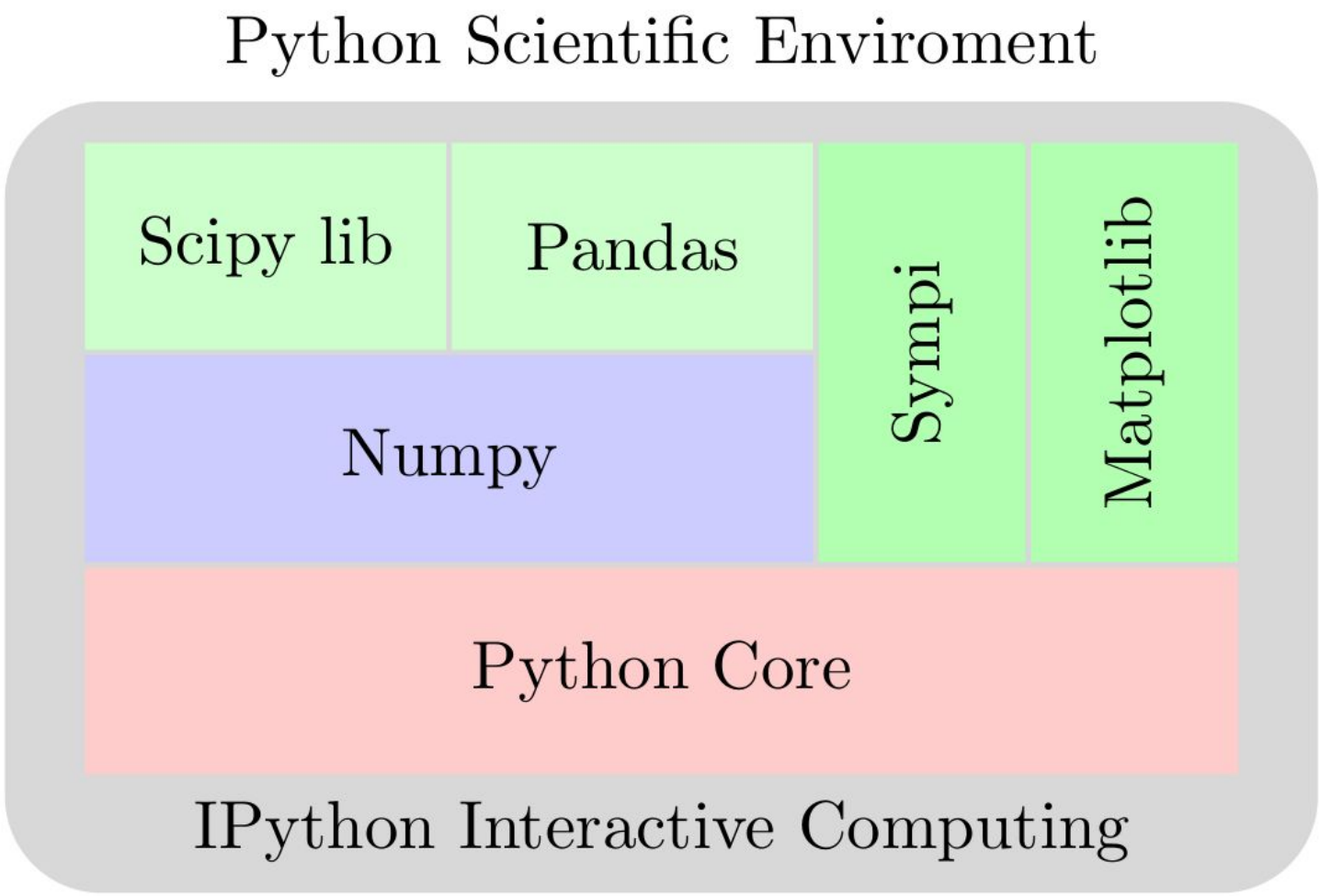
```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
```
Series `s` where value is not >1
`s` where value is <-1 or >2

```
>>> df[df['Population']>1200000000]
```
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```
Set index `a` of Series `s` to 6


Input
account order ext price
383080 10001 235.83
383080 10001 232.32
383080 10001 107.97
412290 10005 2679.36
412290 10005 286.02
412290 10005 832.95
412290 10005 3472.04
412290 10005 915.12
218895 10006 3061.12
218895 10006 518.65
218895 10006 216.9
218895 10006 -72.18
Split
account order ext price
383080 10001 235.83
383080 10001 232.32
383080 10001 107.97
account order ext price
412290 10005 2679.36
412290 10005 286.02
412290 10005 832.95
412290 10005 3472.04
412290 10005 915.12
account order ext price
218895 10006 3061.12
218895 10006 518.65
218895 10006 216.9
218895 10006 -72.18
Apply (sum)
account order ext price
383080 10001 576.12
account order ext price
412290 10005 8185.49
account order ext price
218895 10006 3724.49
Combine (transform)
account order ext price Order_Total
383080 10001 235.83 576.12
383080 10001 232.32 576.12
383080 10001 107.97 576.12
412290 10005 2679.36 8185.49
412290 10005 286.02 8185.49
412290 10005 832.95 8185.49
412290 10005 3472.04 8185.49
412290 10005 915.12 8185.49
218895 10006 3061.12 3724.49
218895 10006 518.65 3724.49
218895 10006 216.9 3724.49
218895 10006 -72.18 3724.49
pbpython.com


Python Scientific Enviroment
Scipy lib
Pandas
Sympi
Matplotlib
Numpy
Python Core
IPython Interactive Computing