

# Python For Data Science Cheat Sheet

## Pandas Basics

Learn Python for Data Science Interactively at [www.DataCamp.com](http://www.DataCamp.com)



### Pandas

The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.



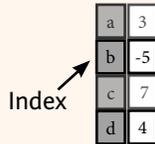
Use the following import convention:

```
>>> import pandas as pd
```

### 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

Columns

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528

Index

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'])
```

### Asking For Help

```
>>> help(pd.Series.loc)
```

### 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'
Select single value by row & column

>>> df.iat[0,0]
'Belgium'
```

#### By Label

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

>>> df.at[0, ['Country']]
'Belgium'
```

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

### Dropping

```
>>> s.drop(['a', 'c'])
Drop values from rows (axis=0)
>>> df.drop('Country', axis=1)
Drop values from columns (axis=1)
```

### Sort & Rank

```
>>> df.sort_index()
Sort by labels along an axis
>>> df.sort_values(by='Country')
Sort by the values along an axis
>>> df.rank()
Assign ranks to entries
```

### Retrieving Series/DataFrame Information

#### Basic Information

```
>>> df.shape
(rows, columns)
>>> df.index
Describe index
>>> df.columns
Describe DataFrame columns
>>> df.info()
Info on DataFrame
>>> df.count()
Number of non-NA values
```

#### Summary

```
>>> df.sum()
Sum of values
>>> df.cumsum()
Cumulative sum of values
>>> df.min()/df.max()
Minimum/maximum values
>>> df.idxmin()/df.idxmax()
Minimum/Maximum index value
>>> df.describe()
Summary statistics
>>> df.mean()
Mean of values
>>> df.median()
Median of values
```

### Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
Apply function
>>> df.applymap(f)
Apply function element-wise
```

### Data Alignment

#### Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a      10.0
b      NaN
c       5.0
d       7.0
```

#### Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
a      10.0
b     -5.0
c       5.0
d       7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

### I/O

#### Read and Write to CSV

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

