Get/change current working directory

```
os.getcwd()
os.chdir(<directory>)
```

Separate filename and path

```
os.path.basename(<path>)
os.path.dirname(<path>)
```

Print path files under directory

```
def print_directory_contents(sPath):
    import os
    for sChild in os.listdir(sPath):
        sChildPath = os.path.join(sPath,sChild)
        if os.path.isdir(sChildPath):    # is directory
            print_directory_contents(sChildPath)
        else:
            print sChildPath    # list all files with path
            print sChild    # list all files without path

def print_dir(Path):
    import os
    for dir, sub_dir, files in os.walk(Path):
        for f in files:
            print os.path.join(dir,f)
```

Find certain file type

```
if sChild.endswith('.txt'):
```

Change directory

```
os.mkdir(<directory>)
os.rmdir(<directory>)    # remove directory
os.removedirs(<directory>)    # also remove parent directories if empty
os.remove(<file>)
os.rename(<file or directory>)
```

Open files 1

```
with open(filename,'r') as f:
    print f.read()    # or: for line in f: print line
```

Open files 2

```
try:
    f = open(filename,'r')
    print f.read()
    f.close()   # mandatory
except IOError:
    print 'Cannot open file'
```

Pandas.DataFrame

```
table = pandas.DataFrame(data = <np.ndarry | dict | DataFrame>
                         index = <array>    # np.arange(n) by default
                         columns = <array | list>    # column labels
                         dtype = <default None>
                         copy = <default False> )

table['col_name'] = […]    # add new columns
table2 = table.append(dict,ignore_index = True)    # add new rows

# can only reorder columns when creating a DataFrame.
# use table.columns = […] to rename columns in a separate line.
```

Useful functions

```
row, col = table.shape
table.count()

table.describe()
table.fillna(0.00)
table2 = table.T    # transpose
table.abs()    # abs()
```

```
table.add(<DataFrame | list | etc.>)    # element-wise add

any(table['col1'] > table['col2'])   # or all
(table['col1'] > table['col2']).any()   # or all

table.apply(func,N)    # works on a row (0) / column (1) basis of a DataFrame
table.applymap(func)    # works element-wise on a DataFrame
table['colname'].map(func)    # works element-wise on a Series.

table.corr()    # correlation matrix
table.corrwith(table2)    # pairwise correlation
table.cov()    # covariance matrix
table['colname'].var()    # variance

table2 = table.drop_duplicates(<col_name>,keep = 'last')

grouped = data.groupby(col)
grouped.sum()    # first() last() mean() count() var() corr() std() get_group()

isnull()    # element-wise check
table['col1'].iteritems()    # iterator over pairs
table.iterrows()    # iterator over rows
table.itertuples()    # similar make rows tuples

table.mean()
table.median()
table.mode()
table.max() table.min()
table.quantile(<[0,1]>)

table1.multiply(table2) table1.mul(table2)
table.prod() table.product()
table.pct_change(<lag>)
table.sort(ascending = False)
table.transpose()
```

Pandas.read_csv()

```
filepath
sep/delimiter: str, default ','
header: int or list of ints, default 'infer', header = 0: first line
index_col = True/False
squeeze: Boolean, default False. Returns series when only have one column
na_values
skip_blank_lines: default False
nrows: default None, useful for reading large files
skiprows: skip number of rows from the start
```

Indexing and Selecting

```
table.loc[<row_name>,<col_name>]    #data.loc['c','A']
table.iloc[<row_num<,<col_num>]    #data.iloc[2,0]
table[<col_name>][<row_num>]    #data['A'][2]
table.<col_name>    #data. A[2]    data.A.c
```

Sampling

```
DataFrame.sample(n=None, frac=None, replace=False, weights=None,
                 random_state=None, axis=None)
```

Fast Scalar Value Getting and Setting

```
table.at[<row_name>,<col_name>]
table.iat[<row_num<,<col_num>]
```

Boolean Indexing

```
& for and, | for or, ~ for not
table[table>0]
table[table['A']>0]
```

Indexing with isin

```
table.isin(<list of values>)
```

Where

```
# table[table<0] is equivalent to table.where(table<0)
table.where(table<0,-table)    # where(condition, otherwise)
table.where(table<0,-table,inplace = True)    # default False, create copy
```

Query (faster)

```
table.query('(A>B)&(B<C)')
table.query('A>B and B<C')
# df.query('a in b') = df[df.a.isin(df.b)]
# df.query('a not in b') = df[~df.a.isin(df.b)]
# Comparing a list to a column using ==/!= works similarly to in/not in
```

Read data and join tables

```
dfAPPL = pd.read_csv('data/APPL.csv',index_col = 'Date', na_values = ['nan'],
                           parse_dates = True, usecols = ['Date', 'Adj Close'])
start_date = '2010-01-01'
end_date = '2016-01-01'
dates = pd.date_range(start_date, end_date)
df = pd.DataFrame(index = dates)
df = df.join(dfAPPL)    # join all rows in dfAPPL that have same index in df
# df.join(dfAPPL, how = 'inner')
df.dropna()
```

numpy

```
np.empty((r,c))
np.ones((r,c))
np.zeros((r,c))
np.random.random((r,c))  np.random.rand(r,c)
np.random.normal(mu, sigma, size = (r,c))
np.random.randint(min, max, size = (r,c))
array.shape => (r,c)
array.size => r*c
array.sum(axis = 0)
array.argmax()    # returns index of maximum value
```