Python Cheatsheet

Contents

[Python Cheatsheet 1](#_Toc449501283)

[Arrays (numpy) 8](#_Toc449501284)

[Add a column to an array 8](#_Toc449501285)

[Add a dimension to an array 8](#_Toc449501286)

[Apply a function to every element of an array 8](#_Toc449501287)

[Concatenate Arrays 8](#_Toc449501288)

[Convert a list to an array 9](#_Toc449501289)

[Count the number of equal items in two arrays 9](#_Toc449501290)

[Create an array of constants 9](#_Toc449501291)

[Create an array of zeros 9](#_Toc449501292)

[2D 9](#_Toc449501293)

[Create and add rows to an array 10](#_Toc449501294)

[Dimensions of an array 10](#_Toc449501295)

[Dot Product of 2 arrays 11](#_Toc449501296)

[Element-wise Multiplication 11](#_Toc449501297)

[Load a CSV file into a numpy array 11](#_Toc449501298)

[Expand the Rank (number of dimensions) of an array 11](#_Toc449501299)

[Select a Subset of Rows 12](#_Toc449501300)

[Select Rows from an Array which Match a List of Values for some Column 12](#_Toc449501301)

[Shuffle an Array 12](#_Toc449501302)

[Slicing an Array 13](#_Toc449501303)

[Return the last 3 elements 13](#_Toc449501304)

[Standardize an Array 13](#_Toc449501305)

[Sort a numpy.ndarray 14](#_Toc449501306)

[Split an Array into 2 parts 14](#_Toc449501307)

[Transpose a numpy.ndarray 14](#_Toc449501308)

[Classes 15](#_Toc449501309)

[Class static methods 15](#_Toc449501310)

[Control Statements 16](#_Toc449501311)

[for-next loops 16](#_Toc449501312)

[for-next over multiple variables 16](#_Toc449501313)

[for-next loops using xrange to improve performance 16](#_Toc449501314)

[while loops 16](#_Toc449501315)

[CSV Files 17](#_Toc449501316)

[Read from a CSV file 17](#_Toc449501317)

[Write to a CSV file 17](#_Toc449501318)

[Database Functions 17](#_Toc449501319)

[Postgres 17](#_Toc449501320)

[Connect to a Postgres database 17](#_Toc449501321)

[Execute SQL query on Postgres 18](#_Toc449501322)

[SQLite3 18](#_Toc449501323)

[Create a SQLite3 Database 18](#_Toc449501324)

[Data Types 18](#_Toc449501325)

[Insert Values into Database 19](#_Toc449501326)

[Read from a Database Table 19](#_Toc449501327)

[Parameterized Queries 19](#_Toc449501328)

[DataFrame (pandas) 20](#_Toc449501329)

[Add a column to a DataFrame 20](#_Toc449501330)

[Add a row to a DataFrame 20](#_Toc449501331)

[Apply a function, with arguments 21](#_Toc449501332)

[Change column names 21](#_Toc449501333)

[Change values in one column based on values in a different column 22](#_Toc449501334)

[Concatenate two data frames 22](#_Toc449501335)

[Copy a column from another DataFrame 22](#_Toc449501336)

[Correlation between columns 22](#_Toc449501337)

[Count the distinct values in a DataFrame column 23](#_Toc449501338)

[Create a DataFrame from scratch 23](#_Toc449501339)

[Create a DataFrame which has only one column 23](#_Toc449501340)

[Delete Rows Having Nulls in Certain Columns 24](#_Toc449501341)

[Extract a column from a DataFrame into a Series 24](#_Toc449501342)

[Get the rows in a DataFrame having a null in some column 24](#_Toc449501343)

[Fast update of a DataFrame column 25](#_Toc449501344)

[Filter out na values from a column 25](#_Toc449501345)

[Get DataFrame column names 25](#_Toc449501346)

[Get DataFrame column values 26](#_Toc449501347)

[Get dimensions of a DataFrame 26](#_Toc449501348)

[Get row count from a DataFrame 26](#_Toc449501349)

[Get rows from a DataFrame by index 26](#_Toc449501350)

[Get Rows from a DataFrame which Match an Element of a List 27](#_Toc449501351)

[Get unique values from a DataFrame column 27](#_Toc449501352)

[Insert a column into a DataFrame 27](#_Toc449501353)

[Max value of a DataFrame column 28](#_Toc449501354)

[Randomly Split a DataFrame 28](#_Toc449501355)

[Read a CSV file into a DataFrame 29](#_Toc449501356)

[Parse formatted dates while reading a CSV 29](#_Toc449501357)

[Select a cell from a DataFrame 29](#_Toc449501358)

[Select rows from a DataFrame by value of a column 29](#_Toc449501359)

[Select rows from a DataFrame by values of multiple columns 29](#_Toc449501360)

[Select rows from a DataFrame by values of multiple columns 31](#_Toc449501361)

[Sort a DataFrame 31](#_Toc449501362)

[Substitute for na values in a column 31](#_Toc449501363)

[Summary statistics for a DataFrame 32](#_Toc449501364)

[Write a DataFrame to a csv file 32](#_Toc449501365)

[Wrapping CSV file columns in quotes 32](#_Toc449501366)

[Date Functions 33](#_Toc449501367)

[Add a time interval to a datetime 33](#_Toc449501368)

[Calculate a time interval 33](#_Toc449501369)

[Calculate a time interval in seconds, days 33](#_Toc449501370)

[Convert a datetime to Epoch Seconds 34](#_Toc449501371)

[Convert an Epoch to a time 34](#_Toc449501372)

[Convert string to date 34](#_Toc449501373)

[Microseconds 35](#_Toc449501374)

[Date Time Format Strings 36](#_Toc449501375)

[Another method: 36](#_Toc449501376)

[Create an arbitrary datetime 36](#_Toc449501377)

[datetime with time zone 37](#_Toc449501378)

[Get the current datetime 37](#_Toc449501379)

[Get year, month, day, hour, minute, second, milliseconds, weekday 37](#_Toc449501380)

[ISO Weekday 38](#_Toc449501381)

[Time Zone Names 38](#_Toc449501382)

[Dictionaries 38](#_Toc449501383)

[Convert a DataFrame to a Dictionary 38](#_Toc449501384)

[Create a dictionary 38](#_Toc449501385)

[Get a value for a key in the dict 38](#_Toc449501386)

[Get the keys from a dictionary 39](#_Toc449501387)

[Is a key in a dictionary? 39](#_Toc449501388)

[Directories 39](#_Toc449501389)

[Check if a Directory exists 39](#_Toc449501390)

[Concatenate a Directory and File Name 39](#_Toc449501391)

[Create a Directory 40](#_Toc449501392)

[Delete all the files and folders in a directory 40](#_Toc449501393)

[Delete all the files in a directory 40](#_Toc449501394)

[Get the Current Working Directory 40](#_Toc449501395)

[Read the files in a directory. 40](#_Toc449501396)

[Read the files in a directory with a specific extension 41](#_Toc449501397)

[Set the working directory 41](#_Toc449501398)

[Exception Handling 41](#_Toc449501399)

[try-except 41](#_Toc449501400)

[Print the traceback and stack trace 41](#_Toc449501401)

[Files 43](#_Toc449501402)

[Copy a file between from one directory to another 43](#_Toc449501403)

[Delete a file 43](#_Toc449501404)

[Does a file exist? 43](#_Toc449501405)

[Extract the file name from a path 43](#_Toc449501406)

[Open File dialog 43](#_Toc449501407)

[Read a text file into a string 43](#_Toc449501408)

[Read all the lines in a file into a list 44](#_Toc449501409)

[Read a text file line by line 44](#_Toc449501410)

[Read a CSV file 44](#_Toc449501411)

[Write to a Text File 44](#_Toc449501412)

[Geocoding 44](#_Toc449501413)

[Geography 45](#_Toc449501414)

[Distance between two coordinates 45](#_Toc449501415)

[Hash Functions 45](#_Toc449501416)

[Installing packages 45](#_Toc449501417)

[easy\_install 45](#_Toc449501418)

[json 46](#_Toc449501419)

[Reading a json file into a dict 46](#_Toc449501420)

[Lambdas 46](#_Toc449501421)

[Conditional Lambdas 46](#_Toc449501422)

[Libraries 46](#_Toc449501423)

[Find the Function Available in a Library 46](#_Toc449501424)

[Lists 47](#_Toc449501425)

[Concatenate 2 lists 47](#_Toc449501426)

[Copy a list 47](#_Toc449501427)

[Create a list containing a number of constants 47](#_Toc449501428)

[Count the Number of Occurences of an Item in a List 47](#_Toc449501429)

[Creating and Appending to a List 47](#_Toc449501430)

[Last items in a List 48](#_Toc449501431)

[Randomly Split a List 48](#_Toc449501432)

[Replace an item in a list 48](#_Toc449501433)

[Sort a list 48](#_Toc449501434)

[Shuffle the items in a list 49](#_Toc449501435)

[Using lambda and map on a list 49](#_Toc449501436)

[Machine Learning 49](#_Toc449501437)

[Euclidean Distance 49](#_Toc449501438)

[One-Hot Encoder 49](#_Toc449501439)

[Math Functions 49](#_Toc449501440)

[Exponentiation 49](#_Toc449501441)

[Largest float 51](#_Toc449501442)

[Median 51](#_Toc449501443)

[Modulo 51](#_Toc449501444)

[pi 51](#_Toc449501445)

[Random Numbers 51](#_Toc449501446)

[Random float 51](#_Toc449501447)

[Rounding 51](#_Toc449501448)

[General rounding 51](#_Toc449501449)

[Round to half-even 51](#_Toc449501450)

[Round to {x.0, x.5} intervals 53](#_Toc449501451)

[Square Root 53](#_Toc449501452)

[Test for nan 53](#_Toc449501453)

[Matrices 54](#_Toc449501454)

[Number of rows in a matrix 54](#_Toc449501455)

[Read a Matrix from a file 54](#_Toc449501456)

[Read the contents of a matrix column into an array 54](#_Toc449501457)

[Scale matrix columns 54](#_Toc449501458)

[Methods 55](#_Toc449501459)

[Method Header Template 55](#_Toc449501460)

[numpy 55](#_Toc449501461)

[Covariance 55](#_Toc449501462)

[r-squared 55](#_Toc449501463)

[Variance 56](#_Toc449501464)

[Object Serialization 56](#_Toc449501465)

[Create an object from a stored serialization 56](#_Toc449501466)

[Serialize and Store an Object 56](#_Toc449501467)

[pandasql 56](#_Toc449501468)

[Installing pandasql 56](#_Toc449501469)

[Querying using pandasql 57](#_Toc449501470)

[Plotting 57](#_Toc449501471)

[Histograms 57](#_Toc449501472)

[Scatter plot 58](#_Toc449501473)

[Program Execution 59](#_Toc449501474)

[Stopping program execution 59](#_Toc449501475)

[Regular expressions 59](#_Toc449501476)

[Remove punctuation 59](#_Toc449501477)

[Random Numbers 60](#_Toc449501478)

[Random number in a range 60](#_Toc449501479)

[Create a list containing some random numbers 60](#_Toc449501480)

[REST Services 60](#_Toc449501481)

[Consume a REST service 60](#_Toc449501482)

[scikit-learn 60](#_Toc449501483)

[Linear regression 60](#_Toc449501484)

[Series (pandas) 61](#_Toc449501485)

[Convert a Series to a DataFrame 61](#_Toc449501486)

[Create a Series of random numbers 61](#_Toc449501487)

[Get the value of a Series element 61](#_Toc449501488)

[SFrame 61](#_Toc449501489)

[Copy an Sframe 61](#_Toc449501490)

[First n rows of an Sframe 61](#_Toc449501491)

[One-Hot Encoding of an Sframe 61](#_Toc449501492)

[Remove a Column from an Sframe 62](#_Toc449501493)

[Select Rows from an Sframe 62](#_Toc449501494)

[Statistics 62](#_Toc449501495)

[Applying lowess smoothing 62](#_Toc449501496)

[Precision, recall, F1, support 62](#_Toc449501497)

[Strings 63](#_Toc449501498)

[Concatenate strings 63](#_Toc449501499)

[Convert a character to its ASCII integer 63](#_Toc449501500)

[Convert to float 63](#_Toc449501501)

[Convert to lower case 63](#_Toc449501502)

[Find a sub-string 63](#_Toc449501503)

[Formatted strings 64](#_Toc449501504)

[Remove Punctuation 64](#_Toc449501505)

[Replace a substring 65](#_Toc449501506)

[String Literals 65](#_Toc449501507)

[Sub-strings 65](#_Toc449501508)

[Tokenize a string 65](#_Toc449501509)

[Trim leading and trailing characters 65](#_Toc449501510)

[Trim white space 65](#_Toc449501511)

[Timers 66](#_Toc449501512)

[Sleep 66](#_Toc449501513)

[Timing Code Execution 66](#_Toc449501514)

[Tuples 66](#_Toc449501515)

[Cartesion product of two tuples 66](#_Toc449501516)

[Product of the elements in a tuple 66](#_Toc449501517)

[User Input 66](#_Toc449501518)

[Get user input from the keyboard 66](#_Toc449501519)

# Arrays (numpy)

## Add a column to an array

>>> N = 3

>>> A = np.eye(N)

>>> A

array([[ 1., 0., 0.],

[ 0., 1., 0.],

[ 0., 0., 1.]])

>>> np.c\_[A, np.zeros(N)]

array([[ 1., 0., 0., 0.],

[ 0., 1., 0., 0.],

[ 0., 0., 1., 0.]])

>>>

## Add a dimension to an array

>>> import numpy as np

>>> x = np.array([1,2])

>>> x

array([1, 2])

>>> x.shape

(2L,)

>>> y = np.expand\_dims(x, axis=0)

>>> y

array([[1, 2]])

>>> y.shape

(1L, 2L)

## Apply a function to every element of an array

get\_pred = np.vectorize(lambda x: 1 if x > 0 else -1)

y\_hat = get\_pred(scores) # scores is type numpy.ndarray

print y\_hat

output:

[ 1 -1 1 ..., -1 1 -1]

## Concatenate Arrays

>>> a = np.array([[1, 2], [3, 4]])

>>> b = np.array([[5, 6]])

>>> np.concatenate((a, b), axis=0)

array([[1, 2],

[3, 4],

[5, 6]])

>>> np.concatenate((a, b.T), axis=1)

array([[1, 2, 5],

[3, 4, 6]])

## Convert a list to an array

>>> blist = [2,4,6,8]

>>> b\_array = np.array(blist)

>>> b\_array

array([2, 4, 6, 8])

## Count the number of equal items in two arrays

import numpy as np

a = np.array([1, 2, 3, 4])

b = np.array([1, 2, 4, 3])

np.sum(a == b) # (a == b) returns [True, True, False, False], but

# np.sum() interprets the Trues as 1

output:

2

(a == b).sum()

output:

2

## Create an array of constants

import numpy as np

a = np.empty(10)

a.fill(55)

## Create an array of zeros

>>> import numpy as np

>>> np.zeros(10)

array([ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

### 2D

>>> test = np.zeros((10,10))

>>> test

array([[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])

## Create and add rows to an array

import numpy as np

arr = np.empty((0,3), float) # arr is type numpy.ndarray

arr = np.append(arr, np.array([[1,2,3]]), axis=0)

arr = np.append(arr, np.array([[4,5,6]]), axis=0)

print arr

>>>

[[ 1. 2. 3.]

[ 4. 5. 6.]]

>>>

## Dimensions of an array

>>> X\_train.shape

(80L, 5L)

## Dot Product of 2 arrays

u = np.arange(0, 5, .5)

v = np.arange(5, 10, .5)

u: [ 0. 0.5 1. 1.5 2. 2.5 3. 3.5 4. 4.5]

v: [ 5. 5.5 6. 6.5 7. 7.5 8. 8.5 9. 9.5]

dotProduct = np.dot(u,v)

dotProduct

183.75

You can also do this for a matrix and array.

Note that numpy will automatically transpose the second argument, if needed, to make the two arguments consistent.

## Element-wise Multiplication

u = np.arange(0, 5, .5)

v = np.arange(5, 10, .5)

elementWise = u \* v

u: [ 0. 0.5 1. 1.5 2. 2.5 3. 3.5 4. 4.5]

v: [ 5. 5.5 6. 6.5 7. 7.5 8. 8.5 9. 9.5]

elementWise

[ 0. 2.75 6. 9.75 14. 18.75 24. 29.75 36. 42.75]

## Load a CSV file into a numpy array

from numpy import genfromtext

filepath = "eta\_corr\_owner\_veh\_km-rem\_mins-rem\_pace\_lat\_lon\_hr\_day\_mo\_no-dupe.csv"

my\_data = genfromtxt(filepath, delimiter=',', skip\_header=1) # assumes 1 header row

## Expand the Rank (number of dimensions) of an array

>>> import numpy as np

>>> x = np.array([1,2])

>>> x

array([1, 2])

>>> x.shape

(2L,)

>>> y = np.expand\_dims(x, axis=0)

>>> y

array([[1, 2]])

>>> y.shape

(1L, 2L)

>>> len(x.shape)

1

>>> len(y.shape)

2

## Select a Subset of Rows

import numpy as np

data = np.array([

[100002, 2006, 1.1, 0.01, 6352],

[100002, 2006, 1.2, 0.84, 304518],

[100002, 2006, 2, 1.52, 148219],

[100002, 2007, 1.1, 0.01, 6292],

[10002, 2006, 1.1, 0.01, 5968],

[10002, 2006, 1.2, 0.25, 104318],

[10002, 2007, 1.1, 0.01, 6800],

[10002, 2007, 4, 2.03, 25446],

[10002, 2008, 1.1, 0.01, 6408] ])

subset1 = data[data[:,0] == 100002]

subset1:

array([[ 1.00002e+05, 2.006e+03, 1.10e+00, 1.00e-02, 6.352e+03],

[ 1.00002e+05, 2.006e+03, 1.20e+00, 8.40e-01, 3.04518e+05],

[ 1.00002e+05, 2.006e+03, 2.00e+00, 1.52e+00, 1.48219e+05],

[ 1.00002e+05, 2.007e+03, 1.10e+00, 1.00e-02, 6.292e+03]])

## Select Rows from an Array which Match a List of Values for some Column

import numpy as np

training\_shipments, test\_shipments = train\_test\_split(unique\_shipment\_id\_array,

test\_size=0.3,

random\_state=0)

print all\_data.shape

training\_data = all\_data[np.logical\_or.reduce([all\_data[:,9] == x

for x in training\_shipments])]

print my\_data.shape

(63L, 18L)

(53L, 18L)

## Shuffle an Array

idx = np.arange(X.shape[0])

np.random.seed(13)

np.random.shuffle(idx)

X = X[idx]

y = y[idx]

## Slicing an Array

### Return the last 3 elements

features = np.array([1, 2, 3, 4])

print 'features:\n{0}'.format(features)

# The last three elements of features

lastThree = features[-3:]

print '\nlastThree:\n{0}'.format(lastThree)

features:

[1 2 3 4]

lastThree:

[2 3 4]

## Standardize an Array

mean = X.mean(axis=0)

std = X.std(axis=0)

X = (X - mean) / std

## Sort a numpy.ndarray

>>> unique\_dates

array(['2014-10-21', '2014-10-11', '2014-10-14', '2014-10-15',

'2014-10-16', '2014-10-17', '2014-10-18', '2014-10-20',

'2014-10-06', '2014-10-07', '2014-10-09', '2014-10-10',

'2014-10-08', '2014-09-26', '2014-10-13', '2014-10-12',

'2014-10-05', '2014-10-19', '2014-09-30', '2014-10-03',

'2014-10-01', '2014-10-02'], dtype=object)

>>> import numpy as np

>>> np.sort(unique\_dates)

array(['2014-09-26', '2014-09-30', '2014-10-01', '2014-10-02',

'2014-10-03', '2014-10-05', '2014-10-06', '2014-10-07',

'2014-10-08', '2014-10-09', '2014-10-10', '2014-10-11',

'2014-10-12', '2014-10-13', '2014-10-14', '2014-10-15',

'2014-10-16', '2014-10-17', '2014-10-18', '2014-10-19',

'2014-10-20', '2014-10-21'], dtype=object)

## Split an Array into 2 parts

import numpy as np

x = np.arange(9)

train = x[:4]

print train

val = x[4:]

print val

>>>

[0 1 2 3 4 5 6 7 8]

>>> ================================ RESTART ================================

[0 1 2 3]

[4 5 6 7 8]

## Transpose a numpy.ndarray

>>> a = np.array([[1, 2], [3, 4]])

>>> a

array([[1, 2],

[3, 4]])

>>> a.transpose()

array([[1, 3],

[2, 4]])

# Classes

class Dog:

def \_\_init\_\_(self, name):

self.name = name

self.tricks = [] # creates a new empty list for each dog

def add\_trick(self, trick):

self.tricks.append(trick)

>>> d = Dog('Fido')

>>> e = Dog('Buddy')

>>> d.add\_trick('roll over')

>>> e.add\_trick('play dead')

>>> d.tricks

['roll over']

>>> e.tricks

['play dead']

## Class static methods

""" CongregExtractor: Contains the CongregExtractor class """

class CongregExtractor:

""" CongregExtractor class """

def \_\_init\_\_(self):

self.congreg = []

@staticmethod

def stops\_to\_congreg(stop\_list):

congreg\_list = ["list"]

return(congreg\_list)

Invoke like:

from CongregExtractor import CongregExtractor as ce

my\_congregations = ce.stops\_to\_congreg(my\_items\_list)

# Control Statements

## for-next loops

>>> for x in range(0,3):

print(x)

0

1

2

## for-next over multiple variables

import itertools

for i,j in itertools.product(range(3), range(2)):

print i,j

>>>

0 0

0 1

1 0

1 1

2 0

2 1

## for-next loops using xrange to improve performance

Improves performance since xrange creates an iterator instead of a list

for i in xrange(10,7,-1):

print i

>>>

10

9

8

>>>

## while loops

count = 0

while (count < 9):

print 'The count is:', count

count = count + 1

print "Good bye!"

# CSV Files

## Read from a CSV file

import csv

with open(‘test\_tag\_speed\_km\_mins\_pace\_hour\_day\_6mos\_end\_2012\_05\_22.csv', 'rb') as csvfile:

myreader = csv.reader(csvfile, delimiter=',')

for row in myreader:

print(row) # each row is a list containing the fields

>>> df = pd.read\_csv('outlier\_data/journey\_owner\_duration\_end\_date\_with\_history.csv')

>>> df.shape

(14410, 6)

>>> type(df)

<class 'pandas.core.frame.DataFrame'>

## Write to a CSV file

import csv

with open('eggs.csv', 'wb') as csvfile:

spamwriter = csv.writer(csvfile, delimiter=' ',

quotechar='|', quoting=csv.QUOTE\_MINIMAL)

spamwriter.writerow(['Spam'] \* 5 + ['Baked Beans'])

spamwriter.writerow(['Spam', 'Lovely Spam', 'Wonderful Spam'])

# Database Functions

## 

## Postgres

import psycopg2

import psycopg2.extensions

### Connect to a Postgres database

def get\_db\_conn():

conn = None

try:

conn = psycopg2.connect("dbname='dbanalytics' " +

"user='analytics' " +

"host='analytics.ca9tgiacgkgn.us-west-2.rds.amazonaws.com' " +

"password='<password-here>'")

print "Connected to dbanalytics database"

except:

print "Unable to connect to database"

return(conn)

### Execute SQL query on Postgres

conn = get\_db\_conn()

cur = conn.cursor()

with open('UC1\_query.sql', 'r') as uc1\_query\_file:

uc1\_query = uc1\_query\_file.read()

cur.execute(uc1\_query)

test\_output = cur.fetchall()

df = pd.DataFrame(test\_output)

df.columns = ['sap\_shipment\_id', 'status\_code', 'status\_reason',

'status\_dt\_utc', 'status\_lat', 'status\_lng',

'sap\_shipping\_point', 'orig\_lat', 'orig\_lon',

'dest\_lat', 'dest\_lon', 'driver\_eta\_utc',

'lat\_lon\_corr\_names', 'sap\_ship\_so',

'planned\_shipment\_end\_date\_utc\_c',

'actual\_shipment\_end\_date\_utc',

'sap\_message\_date\_utc\_c']

## SQLite3

### Create a SQLite3 Database

import sqlite3

conn = sqlite3.connect("lat\_lon\_db")

cursor = conn.cursor()

cursor.execute("create table address(name text PRIMARY\_KEY, latitude real, longitude real)")

conn.commit()

conn.close()

## Data Types

Each value stored in an SQLite database (or manipulated by the database engine) has one of the following storage classes:

•NULL. The value is a NULL value.

•INTEGER. The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.

•REAL. The value is a floating point value, stored as an 8-byte IEEE floating point number.

•TEXT. The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).

•BLOB. The value is a blob of data, stored exactly as it was input.

### Insert Values into Database

import sqlite3

conn = sqlite3.connect("lat\_lon\_db")

conn.text\_factory = lambda x: unicode(x, 'utf-8', 'ignore')

cursor = conn.cursor()

cursor.execute("insert into address (name, latitude, longitude) values (?,?,?)",

("4704 Nelson Brogdon Blvd Ne Sugar Hill GA 30518", 34.1025827, -84.02279469999999))

conn.commit()

conn.close()

Note: failure to use the above text\_factory setting has resulted in errors like: “ProgrammingError: You must not use 8-bit bytestrings unless you use a text\_factory that can interpret 8-bit bytestrings (like text\_factory = str). It is highly recommended that you instead just switch your application to Unicode strings.”

### Read from a Database Table

import sqlite3

conn = sqlite3.connect("lat\_lon\_db")

cursor = conn.cursor()

result = cursor.execute("select \* from address")

print(result.fetchall())

### Parameterized Queries

import sqlite3

conn = sqlite3.connect("lat\_lon\_db")

cursor = conn.cursor()

result = cursor.execute("select \* from address where street\_address = ?", ['1439 Buckeye Court Auburn CA 95603'])

x = result.fetchall()

conn.close()

# DataFrame (pandas)

## Add a column to a DataFrame

from pandas import DataFrame

>>> loc\_data\_frame.shape

(86, 11)

>>> loc\_data\_frame['predict'] = np.zeros(86)

>>> loc\_data\_frame.shape

(86, 12)

-or-

>>> from pandas import DataFrame

>>> test=DataFrame()

>>> my\_list = [1,2,3]

>>> test['col\_name'] = my\_list

>>> test

col\_name

0 1

1 2

2 3

-or-

from pandas import DataFrame

import numpy as np

test=DataFrame()

my\_list = [1,2,3]

test['number'] = my\_list

test['squared'] = test['number'] \*\*2

test['is\_even'] = np.where(test.squared % 2 == 0, True, False)

test['is\_odd'] = test.apply(lambda row: (row.squared % 2 == 1), axis=1)

print test

>>>

number squared is\_even is\_odd

0 1 1 False True

1 2 4 True False

2 3 9 False True

## Add a row to a DataFrame

>>> mydf = DataFrame(columns=("A", "B"))

>>> mydf

Empty DataFrame

Columns: [A, B]

Index: []

>>> mydf.loc[len(mydf)] = [18,19]

>>> mydf

A B

0 18 19

df637 = DataFrame(columns=('status\_dt\_utc',))

df637.loc[0] = [datetime(2016, 4, 27, 12, 0, 0)]

df637.loc[1,['status\_dt\_utc']] = datetime(2016, 4, 27, 12, 1, 0)

print df637

output:

status\_dt\_utc

0 2016-04-27 12:00:00

1 2016-04-27 12:01:00

## Apply a function, with arguments

df958 = DataFrame(columns=("SOME\_LAT", "SOME\_LON"))

df958.loc[0] = [38.896, -121.077] # Auburn, CA, in LA time zone

df958['TZ'] = df958.apply(get\_tz, args=('SOME\_LAT', 'SOME\_LON'), axis=1)

assert df958.TZ[0] == 'America/Los\_Angeles'

## Change column names

df = pd.DataFrame(test\_output)

df.columns = ['sap\_shipment\_id', 'status\_code']

## Change values in one column based on values in a different column

from pandas import DataFrame, read\_csv, Series

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1,2,3]

df.loc[1] = [1,2,4]

df.loc[2] = [1,7,8]

print df

df.loc[df['B'] == 2, 'C'] = 13

print df

>>>

2015-08-31 03:22:56,512 tzwhere.py <module> 42 INFO Application started..

A B C

0 1 2 3

1 1 2 4

2 1 7 8

A B C

0 1 2 13

1 1 2 13

2 1 7 8

## Concatenate two data frames

from pandas import concat

>>> df1

A B C

0 1 2 3

1 3 2 1

>>> df2

D E F

0 5 6 7

1 7 6 5

>>> concat([df1, df2], axis=1)

A B C D E F

0 1 2 3 5 6 7

1 3 2 1 7 6 5

## Copy a column from another DataFrame

# note input\_df will be a DataFrame

input\_df = read\_csv(input\_fname, header=0, quotechar='"',

quoting=csv.QUOTE\_ALL )

corr\_names = DataFrame(input\_df, columns=['CORRIDOR'])

## Correlation between columns

import pandas as pd

import matplotlib.pylab as plt

df = pd.read\_csv('winequality-red.csv', sep=';')

print(df.corr())

## Count the distinct values in a DataFrame column

from pandas import DataFrame, Series

import pandas as pd; import numpy as np

frame = DataFrame(records)

frame['tz'].value\_counts()

America/New\_York 1251

521

America/Chicago 400

America/Los\_Angeles 382

America/Denver 191

Europe/London 74

Asia/Tokyo 37

Pacific/Honolulu 36

Europe/Madrid 35

America/Sao\_Paulo 33

Europe/Berlin 28

Europe/Rome 27

America/Rainy\_River 25

Europe/Amsterdam 22

America/Indianapolis 20

...

Europe/Ljubljana 1

Asia/Riyadh 1

## Create a DataFrame from scratch

from pandas import DataFrame

df = DataFrame(columns=('A', 'B', 'C'))

df

Empty DataFrame

Columns: [A, B, C]

Index: []

>>> df.loc[0] = [1, 2, 3]

>>> df

A B C

0 1 2 3

## Create a DataFrame which has only one column

>>> df = DataFrame(columns=('A',))

>>> df

Empty DataFrame

Columns: [A]

Index: []

## Delete Rows Having Nulls in Certain Columns

from pandas import DataFrame

import pandas as pd

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1, 2, None]

df.loc[1] = [2, 4, 6]

print df, "\n"

missing = df.dropna(subset = ['C'])

print missing

A B C

0 1 2 None

1 2 4 6

A B C

1 2 4 6

## Extract a column from a DataFrame into a Series

my\_series = frame.al # al is one of the column names

my\_series

0 en-US,en;q=0.8

1 NaN

2 en-US

3 pt-br

4 en-US,en;q=0.8

5 en-US,en;q=0.8

6 pl-PL,pl;q=0.8,en-US;q=0.6,en;q=0.4

7 bg,en-us;q=0.7,en;q=0.3

## Get the rows in a DataFrame having a null in some column

mins\_rem\_missing\_df = input\_df[pd.isnull(input\_df['MINS\_REMAINING'])]

yields

status\_dt\_utc MINS\_REMAINING KM\_REMAINING HOUR\_OF\_DAY\_UTC \

489 2015-07-08 12:13:51 NaN 326.425442 12

490 2015-07-08 12:28:22 NaN 324.009080 12

491 2015-07-08 12:30:18 NaN 323.348925 12

492 2015-07-08 12:45:13 NaN 323.348202 12

## Fast update of a DataFrame column

from pandas import DataFrame, read\_csv

from cStringIO import StringIO

def append\_something(input):

return(input + "!")

df = read\_csv("test\_df\_data.txt", header=0)

print df

column\_series = df['header3']

modified\_df = column\_series.apply(append\_something)

df['header3'] = modified\_df

print df

>>>

header1 header2 header3

0 a b c

1 d e f

header1 header2 header3

0 a b c!

1 d e f!

>>>

## Filter out na values from a column

frame.al.dropna() # al is a column name in frame

## Get DataFrame column names

>>> from pandas import DataFrame

>>> journey\_data.columns.values

array(['OWNER', 'VEHICLE\_NUMBER', 'CORRIDOR\_NAME', 'JOURNEY\_NUMBER',

'JOURNEY\_DURATION\_HRS', 'ARRIVAL\_TIME'], dtype=object)

## Get DataFrame column values

from pandas import DataFrame, read\_csv, Series

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1,2,3]

df.loc[1] = [1,2,4]

df.loc[2] = [1,7,8]

print df.B.values

print type(df.B)

[ 2. 2. 7.]

<class 'pandas.core.series.Series'>

>>>

## Get dimensions of a DataFrame

>>> type(journey\_data)

<class 'pandas.core.frame.DataFrame'>

>>> journey\_data.shape

(14410, 6)

## Get row count from a DataFrame

>>> len(journey\_data)

14410

## Get rows from a DataFrame by index

>>> import pandas as pd

>>> df[2:4]

OWNER VEHICLE\_NUMBER CORRIDOR\_NAME \

2 Ragos KBH255J BUSIA-KAMPALA

3 COUNTRYMOTORS KBW990K BUSIA-KAMPALA

JOURNEY\_NUMBER JOURNEY\_DURATION\_HRS \

2 SGS-160868-KSM-02 5

3 SGS-KRA-2014NKUC11588-KRA-2014NKUC11588-01 14

ARRIVAL\_TIME

2 11-OCT-13 12.57.19.000000000 PM

3 26-MAR-14 03.08.02.000000000 AM

>>> type(df[2:4])

<class 'pandas.core.frame.DataFrame'>

## Get Rows from a DataFrame which Match an Element of a List

In [5]: df = DataFrame({'A' : [5,6,3,4], 'B' : [1,2,3, 5]})

In [6]: df

Out[6]:

A B

0 5 1

1 6 2

2 3 3

3 4 5

In [7]: df[df['A'].isin([3, 6])]

Out[7]:

A B

1 6 2

2 3 3

## Get unique values from a DataFrame column

from pandas import DataFrame, read\_csv, Series

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1,2,3]

df.loc[1] = [1,2,4]

df.loc[2] = [1,7,8]

unique\_val\_list = list(set(df.B))

print unique\_val\_list

>>> corr\_names = DataFrame(journey\_data, columns=['CORRIDOR\_NAME'])

>>> corr\_names

CORRIDOR\_NAME

0 ACCRA SGS OFFICE-ADENTA-MR TANKIA

1 ACCRA SGS OFFICE-PIG FARM JUNCTION TOTAL-ACCRA

2 BUSIA-KAMPALA

…

[14410 rows x 1 columns]

>>> from pandas import Series

>>> Series(corr\_names.values.ravel()).unique()

array(['ACCRA SGS OFFICE-ADENTA-MR TANKIA',

'ACCRA SGS OFFICE-PIG FARM JUNCTION TOTAL-ACCRA', 'BUSIA-KAMPALA',

'BUSIA-MOMBASA', 'CHASE TEMA-SAKAMAN TOTAL-ACCRA', …

## Insert a column into a DataFrame

owner\_codes = get\_owner\_codes(veh\_subset\_df)

output\_df.insert(1, 'OWNER\_CODE', owner\_codes, allow\_duplicates=True)

## Max value of a DataFrame column

from pandas import DataFrame, read\_csv, Series

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1,2,3]

df.loc[1] = [1,2,4]

df.loc[2] = [1,7,8]

print df

print max(df['B'])

A B C

0 1 2 3

1 1 2 4

2 1 7 8

7.0

## Randomly Split a DataFrame

import numpy as np

import pandas as pd

df = pd.DataFrame(np.random.randn(10, 2))

msk = np.random.rand(len(df)) < 0.8

train = df[msk]

test = df[~msk]

print df

print msk

print type(train)

print train

0 1

0 0.171749 0.556087

1 0.744720 0.923194

2 0.209449 -0.539187

3 0.231416 1.674134

4 -1.426739 -0.492131

5 -0.919245 -0.203287

6 -0.656182 -1.516310

7 1.053216 -0.445366

8 1.641810 0.066485

9 0.449852 -2.599164

[ True False True True True False True True True False]

<class 'pandas.core.frame.DataFrame'>

0 1

0 0.171749 0.556087

2 0.209449 -0.539187

3 0.231416 1.674134

4 -1.426739 -0.492131

6 -0.656182 -1.516310

7 1.053216 -0.445366

8 1.641810 0.066485

## Read a CSV file into a DataFrame

<http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html>

from pandas import read\_csv

df = read\_csv('eta\_exp\_2/eta\_corr\_owner\_veh\_kph\_km-rem\_mins-rem\_pace\_lat\_lon\_hr\_day\_min.csv', header=0)

// note header=0 means the column names are read from the first row of the input file

### Parse formatted dates while reading a CSV

DATE\_FORMAT\_STRING\_MB = r'%d-%b-%y %I.%M.%S.%f000 %p'

test\_output\_df = read\_csv('mb11\_jrny\_chckpnt\_data.csv',

header=0,

parse\_dates=['FEED\_DT',

'LEG\_PLANNED\_ARR\_DT',

'LEG\_ACTUAL\_ARR\_DT'],

date\_parser = lambda x: pd.to\_datetime(x,

format=DATE\_FORMAT\_STRING\_MB,

coerce=True)

## Select a cell from a DataFrame

>>> journey\_data[1:2]

OWNER VEHICLE\_NUMBER CORRIDOR\_NAME \

1 Total Ghana ITDEMO ACCRA SGS OFFICE-PIG FARM JUNCTION TOTAL-ACCRA

JOURNEY\_NUMBER JOURNEY\_DURATION\_HRS \

1 SGS-ITDEMO-20140722-01 2

ARRIVAL\_TIME

1 22-JUL-14 03.47.05.000000000 PM

>>> journey\_data.iloc[1][0]

'Total Ghana'

>>> journey\_data.iloc[1]['CORRIDOR\_NAME']

'ACCRA SGS OFFICE-PIG FARM JUNCTION TOTAL-ACCRA'

## Select rows from a DataFrame by value of a column

>>> journey\_data.shape

(14410, 6)

>>> subset = journey\_data[journey\_data['CORRIDOR\_NAME'] == 'MOMBASA-NAIROBI']

>>> subset.shape

(4481, 6)

## Select rows from a DataFrame by values of multiple columns

from pandas import DataFrame

df = DataFrame(columns=('A', 'B', 'C'))

df.loc[0] = [1,2,3]

df.loc[1] = [1,2,4]

df.loc[2] = [1,7,8]

print df

A B C

0 1 2 3

1 1 2 4

2 1 7 8

df2 = df[(df['A'] == 1) & (df['B'] == 2)]

print df2

A B C

0 1 2 3

1 1 2 4

## Select rows from a DataFrame by values of multiple columns

import pandas as pd

from random import randint

df = pd.DataFrame({'A': [x for x in xrange(10)],

'B': [x \* 10 for x in xrange(10)],

'C': [x \* 100 for x in xrange(10)]})

print df

subset\_df = df.loc[(df["B"] == 40) & (df["C"] == 400)]

print subset\_df

>>>

A B C

0 0 0 0

1 1 10 100

2 2 20 200

3 3 30 300

4 4 40 400

5 5 50 500

...

9 9 90 900

A B C

4 4 40 400

## Sort a DataFrame

>>> mydf

A B

0 18 19

1 3 4

>>> sorted = mydf.sort(["A", "B"])

>>> sorted

A B

1 3 4

0 18 19

## Substitute for na values in a column

from pandas import DataFrame, Series

import pandas as pd; import numpy as np

frame = DataFrame(records)

frame['tz'].value\_counts()

clean\_tz = frame['tz'].fillna('Missing')

clean\_tz

8

9

10 America/Los\_Angeles

11 America/New\_York

12 America/New\_York

13 Missing

## Summary statistics for a DataFrame

import pandas as pd

df = pd.read\_csv('winequality-red.csv', sep=';', header=0)

df.describe()



## Write a DataFrame to a csv file

veh\_data\_subset.to\_csv(out\_file\_name, index=False) # index=False suppresses row\_id

### Wrapping CSV file columns in quotes

import csv

output\_df.to\_csv('lat\_lon\_epoch\_weather4.csv', index=False, **quoting=csv.QUOTE\_ALL** )

# Date Functions

## Add a time interval to a datetime

Definition: relativedelta.relativedelta(self, dt1=None, dt2=None,

years=0, months=0, days=0, leapdays=0, weeks=0, hours=0, minutes=0,

seconds=0, microseconds=0, year=None, month=None, day=None,

weekday=None, yearday=None, nlyearday=None, hour=None, minute=None,

second=None, microsecond=None)

>>> from dateutil.relativedelta import relativedelta

>>> import datetime

>>> today = datetime.datetime.today()

>>> today

datetime.datetime(2014, 11, 20)

>>> one\_day\_relative = relativedelta(days=1)

>>> today + one\_day\_relative

datetime.date(2014, 11, 21)

# Add minutes

>>> from dateutil.relativedelta import relativedelta

>>> jetzt = datetime.now()

>>> jetzt

datetime.datetime(2014, 12, 14, 6, 24, 12, 475000)

>>> twenty\_mins\_relative = relativedelta(minutes=20)

>>> twenty\_mins\_relative

relativedelta(minutes=+20)

>>> jetzt + twenty\_mins\_relative

datetime.datetime(2014, 12, 14, 6, 44, 12, 475000)

## Calculate a time interval

>>> from datetime import date, timedelta

>>> date.today()

datetime.date(2014, 8, 25)

>>> thirty\_days\_ago = date.today() - timedelta(days=30)

>>> thirty\_days\_ago

datetime.date(2014, 7, 26)

## Calculate a time interval in seconds, days

>>> t1 = datetime.now()

>>> t1

datetime.datetime(2014, 10, 22, 14, 44, 43, 868000)

>>> t2 = datetime.now()

>>> t2 - t1

datetime.timedelta(0, 20, 740000)

>>> from datetime import datetime

>>> datetime.now()

datetime.datetime(2014, 10, 22, 14, 48, 20, 878000)

t1 = datetime(2015, 9, 9, 0, 0, 0, 0)

t2 = datetime(2015, 9, 9, 0, 2, 0, 0)

assert minutes\_between(t1, t2) == 2

>>> (t2-t1).total\_seconds()

20.74

>>> type(t2-t1)

<type 'datetime.timedelta'>

>>> from datetime import timedelta

>>> test = datetime.now()

>>> test2 = datetime.now()

>>> test

datetime.datetime(2014, 11, 20, 6, 0, 42, 884000)

>>> test2

datetime.datetime(2014, 11, 20, 6, 0, 57, 274000)

>>> d = test2 - test

>>> d

datetime.timedelta(0, 14, 390000)

>>> d.seconds

14

>>> d.days

0

## Convert a datetime to Epoch Seconds

int((datetime.datetime(2012,04,01,0,0) - datetime.datetime(1970,1,1)).total\_seconds())

1333238400

## Convert an Epoch to a time

>>> import time

>>> time.gmtime(1423371600)

time.struct\_time(tm\_year=2015, tm\_mon=2, tm\_mday=8, tm\_hour=5, tm\_min=0, tm\_sec=0, tm\_wday=6, tm\_yday=39, tm\_isdst=0)

## Convert string to date

import datetime

>>> datetime.datetime.strptime("02/05/2014", "%m/%d/%Y").date()

datetime.date(2014, 2, 5)

>>> datetime.datetime.strptime("2/5/2014", "%m/%d/%Y").date()

datetime.date(2014, 2, 5)

>>> datetime.strptime('26-MAR-14 03.08.02.000000000 AM', '%d-%b-%y %I.%M.%S.000000000 %p')

datetime.datetime(2014, 3, 26, 3, 8, 2)

from datetime import datetime

status\_dt = datetime.strptime(journey\_subset\_df.iloc[i]['FEED\_DT'],

'%d-%b-%y %I.%M.%S.000000000 %p')

### Microseconds

import datetime

DATE\_FORMAT\_STRING\_MB = r'%d-%b-%y %I.%M.%S.%f000 %p'

print datetime.strptime("19-AUG-15 09.52.14.167842000 AM", DATE\_FORMAT\_STRING\_MB)

2015-08-19 09:52:14.167842

## Date Time Format Strings

| **Directive** | **Meaning** | **Notes** |
| --- | --- | --- |
| %a | Locale’s abbreviated weekday name. |  |
| %A | Locale’s full weekday name. |  |
| %b | Locale’s abbreviated month name. |  |
| %B | Locale’s full month name. |  |
| %c | Locale’s appropriate date and time representation. |  |
| %d | Day of the month as a decimal number [01,31]. |  |
| %H | Hour (24-hour clock) as a decimal number [00,23]. |  |
| %I | Hour (12-hour clock) as a decimal number [01,12]. |  |
| %j | Day of the year as a decimal number [001,366]. |  |
| %m | Month as a decimal number [01,12]. |  |
| %M | Minute as a decimal number [00,59]. |  |
| %p | Locale’s equivalent of either AM or PM. | (1) |
| %S | Second as a decimal number [00,61]. | (2) |
| %U | Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0. | (3) |
| %w | Weekday as a decimal number [0(Sunday),6]. |  |
| %W | Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0. | (3) |
| %x | Locale’s appropriate date representation. |  |
| %X | Locale’s appropriate time representation. |  |
| %y | Year without century as a decimal number [00,99]. |  |
| %Y | Year with century as a decimal number. |  |
| %Z | Time zone name (no characters if no time zone exists). |  |
| %% | A literal '%' character. |  |

Notes:

1. When used with the [**strptime()**](http://docs.python.org/2/library/time.html#time.strptime) function, the %p directive only affects the output hour field if the %I directive is used to parse the hour.
2. The range really is 0 to 61; this accounts for leap seconds and the (very rare) double leap seconds.
3. When used with the [**strptime()**](http://docs.python.org/2/library/time.html#time.strptime) function, %U and %W are only used in calculations when the day of the week and the year are specified.
4. %f indicates microseconds.

### Another method:

>>> from dateutil.parser import \*

>>> from dateutil.tz import \*

>>> parse('2014-09-11 13:11:07+00:00')

datetime.datetime(2014, 9, 11, 13, 11, 7, tzinfo=tzutc())

## Create an arbitrary datetime

datetime.datetime(2014, 10, 22, 14, 48, 20, 878000) # yr, mo, day, hr, min, sec, us

### datetime with time zone

import pytz

dt\_utc = datetime(2015, 10, 21, 4, 40, 0, 0, pytz.UTC)

print dt\_utc

2015-10-21 04:40:00+00:00

## Get the current datetime

>>> from datetime import datetime

>>> datetime.now()

## Get year, month, day, hour, minute, second, milliseconds, weekday

>>> from datetime import datetime

>>> t2

datetime.datetime(2014, 10, 22, 14, 45, 4, 608000)

>>> t2.year

2014

>>> t2.month

10

>>> t2.day

22

>>> t2.hour

14

>>> t2.minute

45

>>> t2.second

4

>>> t2.microsecond

608000

>>> today = datetime.now()

>>> today

datetime.datetime(2014, 10, 23, 9, 30, 37, 743000)

>>> today.weekday()

3

[Note: In python, Monday is 0, Sunday is 6]

## ISO Weekday

ISO weekday is Monday = 1, Sunday = 7

>>> from datetime import datetime

>>> jetzt = datetime.now()

>>> jetzt

datetime.datetime(2015, 3, 2, 9, 33, 14, 907000)

>>> jetzt.isoweekday()

1

## Time Zone Names

<https://en.wikipedia.org/wiki/List_of_tz_database_time_zones>

# Dictionaries

Used as has tables or associative arrays

## Convert a DataFrame to a Dictionary

>>> test = DataFrame(columns=('A', 'B'))

>>> test.loc[0] = ['Turlock', 'CA']

>>> test.loc[1] = ['Lexington', 'KY']

>>> test

A B

0 Turlock CA

1 Lexington KY

>>> test\_dict = test.set\_index('A').to\_dict()

>>> test\_dict

{'B': {'Turlock': 'CA', 'Lexington': 'KY'}}

>>> test\_dict['B']['Turlock']

'CA'

## Create a dictionary

>>> test = {}

>>> test[(1,2)] = 1

>>> (1,2) in test

True

## Get a value for a key in the dict

dict = {'Name': 'Zara', 'Age': 27}

print "Value : %s" % dict.get('Age')

print "Value : %s" % dict.get('Sex', "Never")

Value : 27

Value : Never

## Get the keys from a dictionary

>>> my\_dict = {}

>>> my\_dict['A'] = 1

>>> my\_dict

{'A': 1}

>>> my\_dict.keys() # note: returns a list

['A']

## Is a key in a dictionary?

>>> test = {}

>>> test["something"] = 1

>>> test["something"]

1

>>> "somethingelse" in test

False

>>> "something" in test

True

>>> test["somethingelse"]

Traceback (most recent call last):

File "<pyshell#5>", line 1, in <module>

test["somethingelse"]

KeyError: 'somethingelse'

# Directories

## Check if a Directory exists

import os

print(os.path.isdir("/home/el"))

## Concatenate a Directory and File Name

base\_dir = r'c:\bla\bing'

filename = r'data.txt'

os.path.join(base\_dir, filename)

'c:\\bla\\bing\\data.txt'

## Create a Directory

if not os.path.exists('kept\_models\_by\_corr'):

os.makedirs('kept\_models\_by\_corr')

## Delete all the files and folders in a directory

import os

import shutil

def clear\_dir(folder):

""" Delete all the file sin the specified path name.

Path name will be like '/path/to/folder'

"""

for root, dirs, files in os.walk(folder):

for f in files:

os.unlink(os.path.join(root, f))

for d in dirs:

shutil.rmtree(os.path.join(root, d))

## Delete all the files in a directory

import os

def clear\_dir(folder):

""" Delete all the file sin the specified path name.

Path name will be like '/path/to/folder'

"""

for the\_file in os.listdir(folder):

file\_path = os.path.join(folder, the\_file)

try:

if os.path.isfile(file\_path):

os.unlink(file\_path)

except Exception, e:

print e

## Get the Current Working Directory

>>> import os

>>> os.getcwd()

'C:\\Python27'

## Read the files in a directory.

path=r"C:\Users\bbeauchamp\Documents\Data Analytics\Customers\_and\_Projects\SGS\eta\_raw\parser"

>>> import os

>>> files = os.listdir(path)

File names are returned as elements in a list. Note that this will also read in subdirectories.

## Read the files in a directory with a specific extension

>>> import glob

>>> my\_list = glob.glob(r"C:\Users\bbeauchamp\Documents\Data Analytics\Customers\_and\_Projects\SGS\eta\_raw\parser\\*.arff") # note returns a list

## Set the working directory

>>> import os

>>> os.chdir('c:/dev/python')

>>> os.getcwd()

'c:\\dev\\python'

# Exception Handling

## try-except

**import** **sys**

**try**:

f = open('myfile.txt')

s = f.readline()

i = int(s.strip())

**except** IOError **as** e:

**print** "I/O error({0}): {1}".format(e.errno, e.strerror)

**except** ValueError:

**print** "Could not convert data to an integer."

**except**:

**print** "Unexpected error:", sys.exc\_info()[0]

**raise**

## Print the traceback and stack trace

try:

# Create the scaler

scaler = preprocessing.StandardScaler().fit(X\_for\_scaler)

# Store the scaler serialization for use during prediction

ser\_string = pickle.dumps(scaler, pickle.HIGHEST\_PROTOCOL)

text\_file = open(scaler\_output\_dir + corr\_name + ".txt", "wb")

text\_file.write(ser\_string)

text\_file.close()

except:

print "Error - Unable to generate scaler:", sys.exc\_info()[0]

print "filepath=%s" % filepath

print "X\_for\_scaler:"

for j in xrange(0, len(X\_for\_scaler)):

print "row[%d] = %s" % (j, str(X\_for\_scaler[j]))

print "\*\*\* print\_exc:"

traceback.print\_exc()

raise

# Files

## Copy a file between from one directory to another

from shutil import copyfile

copyfile(filename, new\_file\_name)

## Delete a file

os.remove("corr\_validation.csv")

## Does a file exist?

import os.path

os.path.isfile(fname)

## Extract the file name from a path

>>> pathname = "C:\Users\bbeauchamp\Documents\Data Analytics\Customers\_and\_Projects\SGS\eta\_raw\parser\F000008\_km\_min.arff"

>>> from os.path import basename

>>> print basename(pathname)

F000008\_km\_min.arff

or

import ntpath

ntpath.basename("training\_data\_by\_corr\@GENERIC\_31.37\_-92.41\_39.90\_-84.26\_train.csv")

'@GENERIC\_31.37\_-92.41\_39.90\_-84.26\_train.csv'

## Open File dialog

import Tkinter, tkFileDialog

root = Tkinter.Tk()

root.withdraw()

file\_path = tkFileDialog.askopenfilename()

## Read a text file into a string

with open('model\_train.sql', 'r') as model\_train\_sql\_file:

model\_train\_sql = model\_train\_sql\_file.read()

## Read all the lines in a file into a list

>>> text\_file = open("eta\_corr\_owner\_veh\_km-rem\_mins-rem\_pace\_lat\_lon\_hr\_day\_mo\_no-dupe.csv", "r")

>>> lines = text\_file.readlines()

>>> print len(lines)

614161

>>> print lines[1]

BUSIA-MOMBASA,PURA LOGISTICS,SLMKBN434B,37,397.0563678,1402,7.189250437,-1.49094,37.0571,14,6,9

## Read a text file line by line

filename = 'calamp\_msg\_out\_3.xml'

with open(filename) as f:

for line in f:

print(line)

## Read a CSV file

import csv

with open('C:\\Users\\bbeauchamp\\Documents\\Data Analytics\\Customers\_and\_Projects\\SGS\\' +

'GSM\_fail\_2\_or\_more\_devices\_with\_conn\_pct.csv', 'rb') as csvfile:

myReader = csv.reader(csvfile, delimiter = ',')

print( 'test')

for myRow in myReader:

print', '.join(myRow)

## 

## Write to a Text File

points\_file = open('C:\\Users\\bbeauchamp\\Documents\\Data Analytics\\Customers\_and\_Projects\\SGS\\test\_output', 'w')

points\_file.write('This is a test\n')

points\_file.close()

# Geocoding

import urllib

import json

def get\_lat\_lon(address):

print("starting")

params = { }

# params[ 'key' ] = "AIzaSyAfHtyiQmO7OpAp8WiM8RzGcBlYQqCo67w" # the actual key, of course, is not provided here

params[ 'sensor' ] = "false"

params[ 'address' ] = address

params = urllib.urlencode( params )

print "http://maps.googleapis.com/maps/api/geocode/json?%s" % params

f = urllib.urlopen( "http://maps.googleapis.com/maps/api/geocode/json?%s" % params )

reply = f.read()

decodeddata = json.loads(reply)

latitude = (decodeddata['results'][0]['geometry']['location']['lat'])

longitude = (decodeddata['results'][0]['geometry']['location']['lng'])

return([latitude, longitude])

print(get\_lat\_lon("1439 Buckeye Court Auburn CA"))

# Geography

## Distance between two coordinates

import geopy

from geopy.geocoders import Nominatim

from geopy.distance import vincenty

point\_a = (42.52574, -71.42404)

point\_b = (42.526, -71.42644)

print vincenty(point\_a, point\_b).km

# Hash Functions

>>> import hashlib

>>> print hashlib.sha1("This is a test").hexdigest()

a54d88e06612d820bc3be72877c74f257b561b19 # this is a string object

# Installing packages

## easy\_install

c:\Python27\Scripts>easy\_install googlemaps

Searching for googlemaps

Reading https://pypi.python.org/simple/googlemaps/

Reading http://sourceforge.net/projects/py-googlemaps/

…

zip\_safe flag not set; analyzing archive contents...

Adding googlemaps 2.0 to easy-install.pth file

Installed c:\python27\lib\site-packages\googlemaps-2.0-py2.7.egg

Processing dependencies for googlemaps

Finished processing dependencies for googlemaps

**Note: You must then close IDLE and reopen it.**

# json

## Reading a json file into a dict

import json

fname = r"c:/temp/test6.json"

my\_json\_file = open(fname)

my\_data = json.load(my\_json\_file)

print(my\_data["algorithmData"]["conveyance"]["id"])

# 

# Lambdas

## Conditional Lambdas

products['contains\_perfect'] = products['perfect'].apply(lambda x: 1 if x >0 else 0)

# Libraries

## Find the Function Available in a Library

import math

dir(math)

# Lists

## Concatenate 2 lists

>>> a = [1,2]

>>> b = [3,4]

>>> a+b

[1, 2, 3, 4]

## Copy a list

import copy

new\_list = copy.deepcopy(old\_list)

## Create a list containing a number of constants

>>> [1] \* 10

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

>>>

## Count the Number of Occurences of an Item in a List

test = ['A', 'B', 'C', 'A']

print test.count('A')

>>>

2

## 

## Creating and Appending to a List

>>> adds = []

>>> adds

[]

>>> adds.append("a")

>>> adds

['a']

>>> adds.append("b")

>>> adds

['a', 'b']

>>> 'a' in adds

True

>>> 'c' in adds

False

## Last items in a List

a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

a

output:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

a[-9:]

output:

[4, 5, 6, 7, 8, 9, 10, 11, 12]

## Randomly Split a List

import numpy as np

from sklearn.cross\_validation import train\_test\_split

x = range(11)

print x, type(x)

x\_train, x\_test = train\_test\_split(x, test\_size=0.3, random\_state=0)

print x\_train

print x\_test

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] <type 'list'>

[6, 1, 7, 8, 3, 0, 5]

[4, 9, 2, 10]

## Replace an item in a list

import numpy as np

list1 = [1,2,3,None,5]

print list1

for i,val in enumerate(list1):

if val is None:

list1[i] = 17

print list1

>>>

[1, 2, 3, None, 5]

[1, 2, 3, 17, 5]

>>>

## Sort a list

>>> b = [1,4,2,7,3,8]

>>> b

[1, 4, 2, 7, 3, 8]

>>> b.sort()

>>> b

[1, 2, 3, 4, 7, 8]

>>> sorted([5, 2, 3, 1, 4])

[1, 2, 3, 4, 5]

## Shuffle the items in a list

>>> test = [1,2,3,4,5]

>>> from random import shuffle

>>> shuffle(test)

>>> test

[2, 3, 1, 4, 5]

## Using lambda and map on a list

squares = map(lambda x: x\*\*2, range(10))

print squares

output:

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81

# Machine Learning

## Euclidean Distance

from sklearn.metrics.pairwise import euclidean\_distances

counts = [

[0, 1, 1, 0, 0, 1, 0, 1],

[0, 1, 1, 1, 1, 0, 0, 0],

[1, 0, 0, 0, 0, 0, 1, 0]

]

print 'Distance between 1st and 2nd documents:', euclidean\_distances(counts[0], counts[1])

print 'Distance between 1st and 3rd documents:', euclidean\_distances(counts[0], counts[2])

print 'Distance between 2nd and 3rd documents:', euclidean\_distances(counts[1], counts[2])

Distance between 1st and 2nd documents: [[ 2.]]

Distance between 1st and 3rd documents: [[ 2.44948974]]

Distance between 2nd and 3rd documents: [[ 2.44948974]]

## One-Hot Encoder

from sklearn.feature\_extraction import DictVectorizer

onehot\_encoder = DictVectorizer()

instances = [

{'city': 'New York'},

{'city': 'San Francisco'},

{'city': 'Chapel Hill'}

]

print onehot\_encoder.fit\_transform(instances).toarray()

# Math Functions

## Exponentiation

>>> 2\*\*3

8

## Largest float

>>> sys.float\_info.max

1.7976931348623157e+308

## Median

>>> import numpy as np

>>> a = np.array([1,2,3,4])

>>> np.median(a)

2.5

## Modulo

>>> 17 % 3

2

## pi

math.pi

## Random Numbers

### Random float

>>> import random

>>> random.seed(0)

>>> random.uniform(0, 0.2)

0.10956309498898086

## Rounding

### General rounding

>>> round(110.574388557174, 3)

110.574

### Round to half-even

import decimal

>>> decimal.Decimal('2.675').quantize(decimal.Decimal('.01'), rounding=decimal.ROUND\_HALF\_EVEN)

Decimal('2.68')

>>> decimal.Decimal('2.665').quantize(decimal.Decimal('.01'), rounding=decimal.ROUND\_HALF\_EVEN)

Decimal('2.66')

Note: **To round to a float equivalent of an integer**, use ‘0’ for the decimal.Decimal() argument.

def round\_half\_even(floating\_point\_str, how\_many\_dec\_points):

""" Rounds the specified floating point value (encoded as a string)

to the specified number of decimal points, using

ROUND\_HALF\_EVEN rounding method.

2.675 rounds to 2.68

2.665 rounds to 2.66

Requires import decimal

"""

rounding\_arg = decimal.Decimal(str(10 \*\* (-1 \* how\_many\_dec\_points)))

rounded = (decimal.Decimal(floating\_point\_str)

.quantize(rounding\_arg,

rounding=decimal.ROUND\_HALF\_EVEN))

return(rounded)

def self\_test():

""" Tests the code in this python program

"""

# test round\_half\_even()

assert round\_half\_even('2.675', 2) == Decimal('2.68')

assert round\_half\_even('2.665', 2) == Decimal('2.66')

print "self test complete"

### Round to {x.0, x.5} intervals

def round\_to\_half(float\_arg):

""" Round a floating point number to the nearest 1/2,

e.g round\_to\_half(1.25) = 1.0 but

round\_to\_half(1.255) = 1.5

"""

twice = float\_arg \* 2.0

rounded = decimal.Decimal(twice).quantize(decimal.Decimal('0'), rounding=decimal.ROUND\_HALF\_EVEN)

return(float(rounded) / 2.0)

## Square Root

import math

math.sqrt(25)

## Test for nan

from math import isnan

nan\_float = float(‘nan’)

>>> math.isnan(nan\_float)

True

# Matrices

## Number of rows in a matrix

row\_count = X\_train.shape[0]

## Read a Matrix from a file

>>> import numpy as np

>>> my\_data = np.genfromtxt(filepath, delimiter=',', skip\_header=8)

(my\_data will be a 2d numpy array)

## Read the contents of a matrix column into an array

>>> mydata

array([[ 1. , 19.91142191, 16. , 74. ],

[ 2. , 17.99404762, 15. , 48. ],

[ 3. , 18.94845361, 16. , 89. ],

[ 4. , 29.55978261, 21.5 , 120. ],

[ 5. , 25.80927835, 18. , 129. ],

[ 6. , 20.21631206, 16. , 16. ],

[ 7. , 18.47900763, 15. , 15. ],

[ 8. , 18.82753165, 15. , 30. ],

[ 9. , 16.14227642, 15. , 21. ],

[ 10. , 18.10933941, 15. , 37. ],

[ 11. , 18.24694377, 14. , 151. ],

[ 12. , 17.70260223, 12. , 79. ]])

>>> rainfall\_mm = mydata[:,3]

>>> rainfall\_mm

array([ 74., 48., 89., 120., 129., 16., 15., 30., 21.,

37., 151., 79.])

## Scale matrix columns

from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X)

X\_scaled = scaler.transform(X)

# Methods

## Method Header Template

def brokenTen(value):

"""Incorrect implementation of the ten function.

Note:

The `if` statement checks an undefined variable `val` instead of `value`.

Args:

value (int): A number.

Returns:

bool: Whether `value` is less than ten.

Raises:

NameError: The function references `val`, which is not available in the local or global

namespace, so a `NameError` is raised.

"""

# numpy

## Covariance

xbar = (6 + 8 + 10 + 14 + 18) / 5

ybar = (7 + 9 + 13 + 17.5 + 18) / 5

cov = ((6 - xbar) \* (7 - ybar) + (8 - xbar) \* (9 - ybar) + (10 - xbar) \* (13 - ybar) +

(14 - xbar) \* (17.5 - ybar) + (18 - xbar) \* (18 - ybar)) / 4

print cov

import numpy as np

print np.cov([6, 8, 10, 14, 18], [7, 9, 13, 17.5, 18])

print np.cov([6, 8, 10, 14, 18], [7, 9, 13, 17.5, 18])[0][1]

22.65

[[ 23.2 22.65]

[ 22.65 24.3 ]]

22.65

## r-squared

from sklearn.linear\_model import LinearRegression

X = [[6], [8], [10], [14], [18]]

y = [[7], [9], [13], [17.5], [18]]

X\_test = [[8], [9], [11], [16], [12]]

y\_test = [[11], [8.5], [15], [18], [11]]

model = LinearRegression()

model.fit(X, y)

print 'R-squared: %.4f' % model.score(X\_test, y\_test)

R-squared: 0.6620

## Variance

import numpy as np

X = [6,8,10,14,18]

print np.var(X, ddof=1)

23.2

# Object Serialization

## Create an object from a stored serialization

# Load the dictionary back from the pickle file.

import pickle

favorite\_color = pickle.load( open( "latlong.p", "rb" ) )

print(favorite\_color["330 Lee Industrial Blvd Austell, Ga 30168"])

print(favorite\_color["330 LEE INDUSTRIAL BLVD AUSTELL, GA 30168"])

## Serialize and Store an Object

tzw\_str = pickle.dumps(tzw, pickle.HIGHEST\_PROTOCOL)

text\_file = open("tzwhere\_tz\_converter.txt", "wb")

text\_file.write(tzw\_str)

text\_file.close()

# pandasql

Uses SQLite3 in the background, so syntax permitted by SQLite3 should work

## Installing pandasql

c:\>pip install -U pandasql

Collecting pandasql

Downloading pandasql-0.6.3.tar.gz

Installing collected packages: pandasql

Running setup.py install for pandasql

Successfully installed pandasql-0.6.3

## Querying using pandasql

import pandas as pd

from pandasql import sqldf

from pandasql import load\_births # returns a test dataset

births\_df = load\_births()

q = """

SELECT \*

FROM births\_df b

where b.date > '1976-01-01'

and b.date < '1977-01-01'

"""

births\_1976\_df = sqldf(q, globals())

print births\_1976\_df

>>>

date births

0 1976-01-01 00:00:00 259173

1 1976-01-01 00:00:00 257455

2 1976-02-01 00:00:00 238153

3 1976-02-01 00:00:00 236551

…

21 1976-11-01 00:00:00 258011

22 1976-12-01 00:00:00 265787

23 1976-12-01 00:00:00 265886

>>>

# Plotting

## Histograms

savi\_error = get\_error\_list(df, 'PRED\_MINS\_REM', 'MINS\_REMAINING')

plan\_error = get\_error\_list(df, 'PLAN\_ETA\_MINS\_REM', 'MINS\_REMAINING')

my\_bins = range(-3000, 3000, 100)

plt.hist(savi\_error, histtype='step', color='b', label='Savi',

normed=False, bins=my\_bins)

plt.hist(plan\_error, histtype='step', color='r', label='Plan',

normed=False, bins=my\_bins)

plt.title("ETA Error: Savi ETA vs Planned Arrival Time")

plt.xlabel("Error, minutes")

plt.ylabel("Frequency")

plt.legend()

plt.show()



## Scatter plot

import pandas as pd

import matplotlib.pylab as plt

df = pd.read\_csv('winequality-red.csv', sep=';')

plt.scatter(df['alcohol'], df['quality'])

plt.xlabel('Alcohol')

plt.ylabel('Quality')

plt.title('Alcohol vs Quality')

plt.show()



# Program Execution

## Stopping program execution

from sys import exit

exit('exiting....')

# Regular expressions

## Remove punctuation

import re

mystring = "test !123"

out = re.sub('[^A-Za-z0-9 ]+', '', mystring)

# Random Numbers

## Random number in a range

from random import randint

print randrange(0,10) # rand int in the range 0,10 inclusive

## Create a list containing some random numbers

from random import randint

random\_indices = []

for x in range (0, 1000):

random\_indices.append(randint(0, 999))

# REST Services

## Consume a REST service

import json

import urllib2

json.load(urllib2.urlopen("url"))

# scikit-learn

## Linear regression

from sklearn.linear\_model import LinearRegression

# Training data

X = [[6], [8], [10], [14], [18]]

y = [[7], [9], [13], [17.5], [18]]

# Create and fit the model

model = LinearRegression()

model.fit(X, y)

print 'A 12" pizza should cost: $%.2f' % model.predict([12])[0]

A 12" pizza should cost: $13.68

# Series (pandas)

## Convert a Series to a DataFrame

Series.to\_frame(name=None)

## Create a Series of random numbers

s = Series(np.random.randn(5))

## Get the value of a Series element

>>> this\_journey\_number[0:0]

Series([], name: JOURNEY\_NUMBER, dtype: object)

>>> this\_journey\_number.values

array(['SGS-KRA-2014NKUC11588-KRA-2014NKUC11588-01'], dtype=object)

>>> this\_journey\_number.values[0]

'SGS-KRA-2014NKUC11588-KRA-2014NKUC11588-01'

# SFrame

## Copy an Sframe

copy\_sframe = some\_frame.copy()

## First n rows of an Sframe

products.head(10)['name']

## One-Hot Encoding of an Sframe

loans\_data = risky\_loans.append(safe\_loans)

for feature in features:

loans\_data\_one\_hot\_encoded = loans\_data[feature].apply(lambda x: {x: 1})

loans\_data\_unpacked = loans\_data\_one\_hot\_encoded.unpack(column\_name\_prefix=feature)

# Change None's to 0's

for column in loans\_data\_unpacked.column\_names():

loans\_data\_unpacked[column] = loans\_data\_unpacked[column].fillna(0)

loans\_data.remove\_column(feature)

loans\_data.add\_columns(loans\_data\_unpacked)

## Remove a Column from an Sframe

Sframe.remove\_column(column\_name)

## Select Rows from an Sframe

example\_labels = sframe.SArray([-1, -1, 1, 1, 1])

subset = example\_labels[example\_labels[:] == -1]

print len(subset)

Response: 2

# Statistics

## Applying lowess smoothing

import numpy as np

import pylab as plt

import statsmodels.api as sm

x = np.linspace(0,2\*np.pi,100)

y = np.sin(x) + np.random.random(100) \* 0.2

lowess = sm.nonparametric.lowess(y, x, frac=0.1)

print type(lowess)

print lowess

plt.plot(x, y, '+')

plt.plot(lowess[:, 0], lowess[:, 1])

plt.show()



## Precision, recall, F1, support

import numpy as np

from sklearn.metrics import precision\_recall\_fscore\_support

#y\_true = np.array([0, 1, 0, 0, 1, 0])

#y\_pred = np.array([0, 1, 1, 0, 0, 1])

y\_true = np.array(['early', 'late', 'early', 'early', 'late', 'early','late'])

y\_pred = np.array(['early', 'late', 'late', 'early', 'early', 'late','late'])

precision, recall, f1, x = precision\_recall\_fscore\_support(y\_true, y\_pred, pos\_label='late', average='micro')

print precision, recall, f1

# Strings

## Concatenate strings

>>> 'is' + 'test'

'istest'

## Convert a character to its ASCII integer

>>> ord('A')

65

## Convert to float

>>> test = "123.45"

>>> type(test)

<type 'str'>

>>> test\_float = float(test)

>>> test\_float

123.45

>>> type(test\_float)

<type 'float'>

## Convert to lower case

>>> "Test".lower()

'test'

## Find a sub-string

>>> test = "something"

>>> test.find('me')

2

## Formatted strings

>>> test = "something %s" % "great"

>>> test

'something great'

## Remove Punctuation

>>> test = "some sunava #%^@$#"

>>> import string

>>> test.translate(None, string.punctuation)

'some sunava '

## Replace a substring

>>> test

'out with the old'

>>> test.replace('old', 'new')

'out with the new'

>>>

## String Literals

>>> x = r"\n"

>>> x

'\\n'

>>>

## Sub-strings

>>> x = "Hello World!"

>>> x[2:]

'llo World!'

>>> x[:2]

'He'

>>> x[:-2]

'Hello Worl'

>>> x[-2:]

'd!'

>>> x[2:-2]

'llo Worl'

## Tokenize a string

>>> test = "the world is at my fingertips"

>>> test.split(' ')

['the', 'world', 'is', 'at', 'my', 'fingertips']

## Trim leading and trailing characters

>>> '"Strip the "leading" and "trailing" double quotes"'.strip('"')

'Strip the "leading" and "trailing" double quotes'

>>> test = "A()"

>>> test

'A()'

>>> test.strip("()")

'A'

## Trim white space

>>> test = " something "

>>> len(test)

11

>>> len(test.strip())

9

# Timers

## Sleep

import time

>>> print time.ctime(); time.sleep(10); print time.ctime()

Fri Sep 26 09:09:30 2014

Fri Sep 26 09:09:40 2014

## Timing Code Execution

>>> import time

>>> start\_time = time.time()

>>> print (time.time() - start\_time)

18.8489999771

# Tuples

## Cartesion product of two tuples

import itertools

x = [1, 3, 5]

p = list( itertools.product(x, x))

print str(p)

[(1, 1), (1, 3), (3, 1), (3, 3)]

## Product of the elements in a tuple

from operator import mul

a = (1,2,3,4)

print(reduce(mul, a))

>>>

24

# User Input

## Get user input from the keyboard

>>> test\_var = raw\_input("Enter some data")

Enter some datacomething

>>> test\_var

'comething'

>>>