# Day - Five

# Data Analysis using Python, Plotting Graphs and Database interactions

- Revisit MySQL Database
- Numpy
- Pandas
- Plotting Graphs using Python (MatPlotlib)
- Logging Module
- Misc Tricks - yield, comprehension
- What's Next in your Path?

# MySQL

**Python Library/Connector Required to connect, retrieve, update data with MySQL database**

- MySQL Connector/Python - from Official MySQL Site
- https://pypi.python.org/pypi/mysqlclient (mysqlclient)
- https://github.com/farcepest/moist - Moist
- PyMySQL

# MySQL

```
mysql> create database pythoncourse;
Query OK, 1 row affected (0.00 sec)

mysql> use pythoncourse;
Database changed
mysql>
```

```
mysql> CREATE TABLE presidents (
    -> id INT(11) NOT NULL AUTO_INCREMENT,
    -> name VARCHAR(45) NOT NULL,
    -> age INT(11) NOT NULL,
    -> PRIMARY KEY (id)
    -> ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> desc presidents;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| name  | varchar(45) | NO   |     | NULL    |                |
| age   | int(11)     | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

**mysql>CREATE DATABASE YOURFIRSTNAME_LASTNAME_PYTHON;**

# MySQL

```
mysql> INSERT INTO presidents (id, name, age) VALUES (1, 'Donalt T', 74);
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO presidents (name, age) VALUES ('Barack O', 54);
Query OK, 1 row affected (0.06 sec)

mysql> select * from presidents;
+----+----------+-----+
| id | name     | age |
+----+----------+-----+
|  1 | Donalt T |  74 |
|  2 | Barack O |  54 |
+----+----------+-----+
2 rows in set (0.00 sec)
```

```
mysql> select * from presidents ORDER BY age;
+----+----------+-----+
| id | name     | age |
+----+----------+-----+
|  2 | Barack O |  54 |
|  1 | Donalt T |  74 |
+----+----------+-----+
2 rows in set (0.00 sec)
```

# Python + MySQL

```
#!/usr/bin/python3
import mysql.connector

config = {
  'user': 'root',
  'password': 'welcome123',
  'host': '127.0.0.1',
  'database': 'mysql',
  'raise_on_warnings': True,

}

# Open database connection
db = mysql.connector.connect(**config)
#db = mysql.connector.connect(user='root', password='welcome123', host='localhost', database='mysql')

# prepare a cursor object using cursor() method
cursor = db.cursor()

# execute SQL query using execute() method.
cursor.execute('SELECT VERSION()')

# Fetch a single row using fetchone() method.
data = cursor.fetchone()

print ("Database version : {} ".format( data))

# disconnect from server
db.close()
```

mysql-dictionary.py

```
$ ./mysql-dictonary.py
Database version : ('5.7.19-0ubuntu0.16.04.1',)
```

# Python + MySQL

## Let us use "ConfigParser" Module

python_mysql_dbconfig.py

```
from configparser import ConfigParser


def read_db_config(filename='mysql.ini', section='mysql'):
    """ Read database configuration file and return a dictionary object
    :param filename: name of the configuration file
    :param section: section of database configuration
    :return: a dictionary of database parameters
    """
    # create parser and read ini configuration file
    parser = ConfigParser()
    parser.read(filename)

    # get section, default to mysql
    db = {}
    if parser.has_section(section):
        items = parser.items(section)
        for item in items:
            db[item[0]] = item[1]
    else:
        raise Exception('{0} not found in the {1} file'.format(section, filename))

    return db
$
```

```
>>> from python_mysql_dbconfig import read_db_config
>>> read_db_config()
{'host': 'localhost', 'database': 'pythoncourse', 'username': 'root', 'password': 'welcome123'}
>>>
```

# Python + MySQL

```
from mysql.connector import MySQLConnection, Error
from python_mysql_dbconfig import read_db_config


def connect():
    """ Connect to MySQL database """

    db_config = read_db_config()
    #print (db_config)
    try:
        print('Connecting to MySQL database...')
        conn = MySQLConnection(**db_config)

        if conn.is_connected():
            print('connection established.')
        else:
            print('connection failed.')

    except Error as error:
        print(error)

    finally:
        conn.close()
        print('Connection closed.')


if __name__ == '__main__':
    connect()
```

```
$ ./python_mysql_connect2.py
Connecting to MySQL database...
connection established.
Connection closed.
$
```

python_mysql_connect2.py

# Python+MySQL

Query data - one row at one time. The fetchone() method returns the next row of a query result set or None in case there is no row left.

```
def query_with_fetchone():
    try:
        dbconfig = read_db_config()
        conn = MySQLConnection(**dbconfig)
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM presidents")

        row = cursor.fetchone()

        while row is not None:
            print(row)
            row = cursor.fetchone()
```

mysql_query.py

```
$ ./mysql_query.py
(1, 'Donalt T', 74)
(2, 'Barack O', 54)
```

# Python + MySQL

Query data - all row in one go. The fetchall() method is used. Use this method only when table Is small - less number of rows

```
def query_with_fetchall():
    try:
        dbconfig = read_db_config()
        conn = MySQLConnection(**dbconfig)
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM presidents")
        rows = cursor.fetchall()

        print('Total Row(s):', cursor.rowcount)
        for row in rows:
            print(row)

    except Error as e:
        print(e)

    finally:
        cursor.close()
        conn.close()
```

```
$ ./mysql_query_fetchall.py
Total Row(s): 2
(1, 'Donalt T', 74)
(2, 'Barack O', 54)
```

mysql_query_fetchall.py

# Python + MySQL

MySQL Connector/Python provides us with the fetchmany() method that returns the next number of rows (n) of the result set, which allows us to balance between time and memory space. Let's take a look at how do we use fetchmany() method.

```
def iter_row(cursor, size=10):
    while True:
        rows = cursor.fetchmany(size)
        if not rows:
            break
        for row in rows:
            yield row

def query_with_fetchmany():
    try:
        dbconfig = read_db_config()
        conn = MySQLConnection(**dbconfig)
        cursor = conn.cursor()

        cursor.execute("SELECT * FROM presidents")

        for row in iter_row(cursor, 10):
            print(row)

    except Error as e:
        print(e)
```

mysql-query_fetchmany.py

# Insert

```
def insert_presidents(name, age):
    query = "INSERT INTO presidents(name,age) " \
            "VALUES(%s,%s)"
    args = (name, age)

    try:
        db_config = read_db_config()
        conn = MySQLConnection(**db_config)

        cursor = conn.cursor()
        cursor.execute(query, args)

        if cursor.lastrowid:
            print('last insert id', cursor.lastrowid)
        else:
            print('last insert id not found')

        conn.commit()
    except Error as error:
        print(error)

    finally:
        cursor.close()
        conn.close()

def main():
   insert_presidents('G. Bush', 50)
```

mysql_insert_records.py

# insert-multiple

```
def insert_presidents(presidents):
    query = "INSERT INTO presidents(name,age) " \
            "VALUES(%s,%s)"

    try:
        db_config = read_db_config()
        conn = MySQLConnection(**db_config)

        cursor = conn.cursor()
        cursor.executemany(query, presidents)

        conn.commit()
    except Error as error:
        print(error)

    finally:
        cursor.close()
        conn.close()

def main():
   presidents = [ ('A', 10), ('B', 20), ('C', 30)]
   insert_presidents(presidents)
```

mysql_insert_multiple_records.py

# Update Record

```
def update_presidents(president_id, name):
    # read database configuration
    db_config = read_db_config()

    # prepare query and data
    query = """ UPDATE presidents
                SET name = %s
                WHERE id = %s """

    data = (name, president_id)

    try:
        conn = MySQLConnection(**db_config)
        cursor = conn.cursor()
        cursor.execute(query, data)
        conn.commit()

    except Error as error:
        print(error)

    finally:
        cursor.close()
        conn.close()
if __name__ == '__main__':
    update_presidents(3, 'George Bush')
```

mysql_update_record.py

# Delete Record

```
from mysql.connector import MySQLConnection, Error
from python_mysql_dbconfig import read_db_config

def delete_president(president_id):
    db_config = read_db_config()

    query = "DELETE FROM presidents WHERE id = %s"

    try:
        # connect to the database server
        conn = MySQLConnection(**db_config)

        # execute the query
        cursor = conn.cursor()
        cursor.execute(query, (president_id,))

        # accept the change
        conn.commit()

    except Error as error:
        print(error)

    finally:
        cursor.close()
        conn.close()

if __name__ == '__main__':
    delete_president(6)
```

mysql_delete_record.py

# Numpy

Numpy - Numerical Python

- ndarray:  multi-dimensional array proving vectorized arithmetic operations
- Mathematical operations on entire array without having to use loops
- Tools to write, read data from disk, tools to work with memory-mapped files
- Linear Algebra, Random Number Generation, and Fourier Transform capabilities.
- Tools to integrate code with lower level languages like C, C++

# Data Used

| | 2012 - 100 meter Olympics Result | | | |
|---|---|---|---|---|
| | Time | Height (Inches) | Weight (Pounds) | Country |
| Usain Bolt | 9.63 | 76.77 | 209.439 | JAM |
| Yohan Blake | 9.75 | 70.86 | 167.551 | JAM |
| Justin Gatlin | 9.79 | 72.83 | 182.984 | USA |
| Tyson Gay | 9.8 | 70.07 | 165.347 | USA |
| Ryan Bailey | 9.88 | 75.98 | 216.053 | USA |
| Churandy Martina | 9.94 | 70.07 | 163.142 | NED |
| Richard Thompson | 9.98 | 70.01 | 176.37 | TTO |
| Asafa Powell | 11.99 | 70.8 | 191.802 | JAM |

# Revisit Lists

- Variables with more than one variables, right?  Collection of values
- Can Hold different data-types (strings, integers, float or objects - anything can go in single List, right)
- Append, Change, Delete, Iterate  - etc operations
- Powered with lots of tricks - comprehensions

# List Comprehension

A list comprehension provides a compact way of mapping a list into another list by applying a function to each of the elements of the list.

```
#!/usr/bin/python3

my_list = [1, 2, 3, 4, 5, 6, 7]
print (my_list)

my_squares = [x ** 2 for x in my_list]
print (my_squares)
```

```
$ ./comprehension_one.py
[1, 2, 3, 4, 5, 6, 7]
[1, 4, 9, 16, 25, 36, 49]
```

comprehension_one.py

# Dictionary Comprehension

```
#!/usr/bin/python3

a_dict = {'a': 1, 'b': 2, 'c': 3}
print (a_dict)


new_dict =  {value:key for key, value in a_dict.items()}
print (new_dict)
```

comprehension_two.py

```
$ ./comprehension_two.py
{'a': 1, 'b': 2, 'c': 3}
{1: 'a', 2: 'b', 3: 'c'}
```

# Lambda Function

The lambda function enables us to create anonymous functions.

These are functions without a name.

```
>>> f = lambda a, b, c: a * b + c
>>> f(3, 4, 5)
17
>>> y = lambda m, n : m ** n
>>> y(2, 5)
32
```

Lambda Functions don't support multi-statement functions or functions that don't return a value.

# Python Filter

The “filter” function operates on a list and returns a subset of that list after applying the filtering rule

```
>>> output_list = filter(f, input_list)
```

```
>>> my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> output_list = filter(lambda x: x > 5, my_list)
>>> type(output_list)
<class 'filter'>
>>> list(output_list)
[6, 7, 8, 9, 10]
```

# Python Map

The “map” function transforms a given list into a new list by transforming each element using a rule

```
>>> output_list = map(f, input_list)
```

```
>>> my_list = [1, 5, 7, 8, 11]
>>> output_list = map(lambda x: x * x, my_list)
>>> type(output_list)
<class 'map'>
>>> list(output_list)
[1, 25, 49, 64, 121]
```

map() can take multiple lists as input arguments.

# Quiz

**What's the output of following code?**

```
#!/usr/bin/python3

sentence = "the quick brown fox jumps over the lazy dog"
words = sentence.split()
word_lengths = [len(word) for word in words if word != "the"]
print(word_lengths)
```

# Quiz

**What's the output of following script?**

```
#!/usr/bin/python3
y = 6

z = lambda x: x * y
print (z(8))
```

# Quiz

**Whats value of variable x?**

```
>>> d = lambda x: x * 5
>>> e = lambda x: x * 2
>>> x = d(2)
>>> x = e(x)
>>> x = d(x * x)
>>>
>>>
>>>
>>> print(x)
```

# Quiz (cont ...)

```
>>> a = range(2,9)
>>> x = map(lambda x, y: x ** y, a, reversed(a))
```

```
>>> a = range(2,9)
>>> a_map = map(lambda x, y: x + y, a, reversed(a))
```

What's Happening here?

# Quiz

**What is the output of following code?**

```
#!/usr/bin/python3
def writer():
        title = 'Sir'
        name = (lambda x:title + ' ' + x)
        return name

who = writer()
who('Arthur')
```

# Task - 1

1. Develop script “filter_positive_numbers.py”
2. Sample Run looks like below:

$ filter_positive_numbers.py 10 -20 100 2 -5 -4 -23 10

The positive numbers entered are: [10, 100, 2, 10]

You are expected to use - comprehension, lambda function, map, filter here.

# Revisit Lists - Count BMI List Way

```
#!/usr/bin/python3
# height in inches, weight in pounds
height = [76.77,  70.86,  72.83,  70.07,  75.98,  70.07,  70.01, 70.8]
weight = [209.439, 167.551, 182.984, 165.347, 216.053, 163.142, 176.37, 191.802]

m_height = [x * 0.0254 for x in height]
k_weight = [x * 0.453592 for x in weight]

#print (m_height)
#print (k_weight)

# kg/m^2

msqr_height = [x * x for x in m_height]
#print (msqr_height)

bmi  = [k_weight[i]/msqr_height[i] for i in range(len(msqr_height))]
print (bmi)
```

bmi_lists.py

# Numpy - Look at bmi example

```
#!/usr/bin/python3

import numpy as np

# height in inches, weight in pounds
height = [76.77,  70.86,  72.83,  70.07,  75.98,  70.07,  70.01, 70.8]
weight = [209.439, 167.551, 182.984, 165.347, 216.053, 163.142, 176.37, 191.802]

np_height = np.array(height)
np_weight = np.array(weight)


np_height_meters  = np_height  * 0.0254
np_weight_kgs = np_weight *  0.453592

#print (m_height)
#print (k_weight)

# kg/m^2

np_height_meters_sqr  = np_height_meters ** 2
#print (msqr_height)

bmi = np_weight_kgs / np_height_meters_sqr

print (bmi)
```

bmi_lists.py

# Numpy - A few Observations

```
>>> import numpy as np
>>> my_numpy = np.array([11.00, 12, 'numpy', True, False])
>>> type(my_numpy)
<class 'numpy.ndarray'>
>>> my_numpy
array(['11.0', '12', 'numpy', 'True', 'False'],
      dtype='<U32')
>>> my_list = [11.00, 12, 'numpy', True, False]
>>> type(my_list)
<class 'list'>
>>> my_list + my_list
[11.0, 12, 'numpy', True, False, 11.0, 12, 'numpy', True, False]
>>> my_numpy + my_numpy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: ufunc 'add' did not contain a loop with signature matching types dtype('<U32') dtype('<U32') dtype('<U32')
>>> my_numpy_n = np.array([1, 2, 3])
>>> type(my_numpy_n)
<class 'numpy.ndarray'>
>>> my_numpy_n
array([1, 2, 3])
>>> my_numpy_n + my_numpy_n
array([2, 4, 6])
>>>
```

# Numpy - A few Observations

```
>>> import numpy as np
>>> bmi = np.array([ 24.98460153,  23.46079266,  24.25439523,  23.67718338,  26.31235232, 23.36143414,  25.29895093,  26.9
005])
>>> type(bmi)
<class 'numpy.ndarray'>
>>> bmi[1]
23.460792659999999
>>> bmi[0]
24.984601529999999
>>> bmi[11]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: index 11 is out of bounds for axis 0 with size 8
>>> bmi > 24
array([ True, False,  True, False,  True, False,  True,  True], dtype=bool)
>>> bmi > 23
array([ True,  True,  True,  True,  True,  True,  True,  True], dtype=bool)
>>> bmi > 22
array([ True,  True,  True,  True,  True,  True,  True,  True], dtype=bool)
>>> bmi[bmi > 22]
array([ 24.98460153,  23.46079266,  24.25439523,  23.67718338,
        26.31235232,  23.36143414,  25.29895093,  26.90200005])
>>> bmi[bmi > 24]
array([ 24.98460153,  24.25439523,  26.31235232,  25.29895093,  26.90200005])
>>> bmi[bmi > 25]
array([ 26.31235232,  25.29895093,  26.90200005])
>>> bmi[bmi > 26]
array([ 26.31235232,  26.90200005])
>>> bmi[bmi > 26.9]
array([ 26.90200005])
>>>
```

# Quiz

1

```
>>> demo = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> demo>2
```

2

```
>>> demo = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> demo[demo >5]
```

3

```
>>> a = np.array([1, 2, 3, True])
>>> b = np.array([False, 2, 3, 4])
>>> a + b
```

4

```
>>> demo = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> demo[demo < 1]
```

A

```
array([6, 7, 8])
```

B

```
array([1, 4, 6, 5])
```

C

```
array([], dtype=int64)
```

D

```
array([False, False,  True,  True,  True,  True,  True,  True], dtype=bool)
```

# N-Dimensional Arrays Numpy

- ndarray - generic multidimensional container for homogeneous data
- shape - tuple indicating size of each dimension
- dtype - an object indicating data type of the array

```
>>> import numpy as np
>>> data = ([0.1000, 0.1200, 0.1300, 0.1400], [0.2100, 0.2200, 0.2300, 0.2400])
>>> n_data = np.array(data)
>>> n_data.shape
(2, 4)
>>> n_data.dtype
dtype('float64')
```

# Additional Functions to Create ndarrays

- zeros
- ones
- empty

```
>>> o = np.zeros(5)
>>> print(o)
[ 0.  0.  0.  0.  0.]
>>> o.dtype
dtype('float64')
>>> o.size
5
>>> one = np.ones(7)
>>> print(one)
[ 1.  1.  1.  1.  1.  1.  1.]
>>> one.dtype
dtype('float64')
>>> one.shape
(7,)
>>> one.size
7
>>>
```

```
>>> e = np.empty(9)
>>> print(e)
[  6.90268625e-310   6.90268625e-310   2.03650144e-316   6.90264011e-310
   2.37151510e-322   2.37151510e-322   2.03589354e-316   6.90268625e-310
   2.03650381e-316]
>>> e.dtype
dtype('float64')
>>>
```

# Additional Functions to Create ndarrays

```
>>> np.zeros((2, 3))
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> np.zeros((2, 3, 1))
array([[[ 0.],
        [ 0.],
        [ 0.]],

       [[ 0.],
        [ 0.],
        [ 0.]]])
>>> np.ones((2, 3, 1))
array([[[ 1.],
        [ 1.],
        [ 1.]],

       [[ 1.],
        [ 1.],
        [ 1.]]])
>>> np.ones((2, 3))
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

```
>>> np.empty((3, 2, 1))
array([[[ 0.],
        [ 0.]],

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

       [[ 0.],
        [ 0.]]])
>>> np.empty((3, 2))
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
>>> np.empty((2, 5))
array([[  6.90268625e-310,   6.90268625e-310,   2.87311426e-317,
          1.97626258e-323,   1.85541175e-316],
       [  6.90268034e-310,   6.90267919e-310,   0.00000000e+000,
          0.00000000e+000,   0.00000000e+000]])
>>> np.empty((2, 2))
array([[  6.90268625e-310,   1.88584462e-316],
       [  6.90268053e-310,   5.30563263e-317]])
>>>
```

# Task - 2

- Read File 100_meter.csv using Python
- Declare “finish_time”, “year” Lists  - empty lists
- Remove white-spaces with the help of regex
- Read Line by Line, Split Each lines at “,”
- Append first element in year list & second element in finish_time list.
- Create 2-Dimentional numpy array - called it “athelets”

# Operation between Arrays and Scalars

```
>>> my_array = np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
>>> my_array.dtype
dtype('float64')
>>> my_array * my_array
array([[  1.,   4.,   9.],
       [ 16.,  25.,  36.]])
>>> my_array - my_array
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> 1/my_array
array([[ 1.        ,  0.5       ,  0.33333333],
       [ 0.25      ,  0.2       ,  0.16666667]])
>>> my_array ** 2
array([[  1.,   4.,   9.],
       [ 16.,  25.,  36.]])
>>>
```

# Basic Indexing and Slicing

```
>>> exp_array = np.arange(10)
>>> exp_array
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> exp_array[5]
5
>>> exp_array[0]
0
>>> exp_array[2:5]
array([2, 3, 4])
>>> exp_array[2:5]=22
>>> exp_array
array([ 0,  1, 22, 22, 22,  5,  6,  7,  8,  9])
>>>
```

# Basic Indexing and Slicing (cont...)

```
>>> exp_array = np.arange(10)
>>> exp_array[5]
5
>>> exp_array[2:5]
array([2, 3, 4])
>>> exp_array[2:5] = 29
>>> exp_array
array([ 0,  1, 29, 29, 29,  5,  6,  7,  8,  9])
>>> my_slice = exp_array[2:5]
>>> my_slice
array([29, 29, 29])
>>> my_slice[1] = 44
>>> my_slice
array([29, 44, 29])
>>> exp_array
array([ 0,  1, 29, 44, 29,  5,  6,  7,  8,  9])
>>> my_slice[:] = 400
>>> exp_array
array([  0,   1, 400, 400, 400,   5,   6,   7,   8,   9])
>>>
```

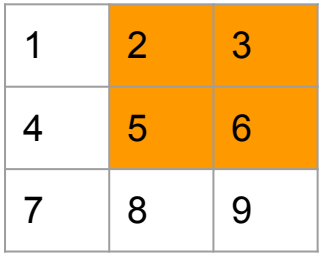
# Basic Indexing and Slicing (cont...)

```
>>> import numpy as np
>>> arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> arr2d[2]
array([7, 8, 9])
>>> arr2d[0][2]
3
>>>
```

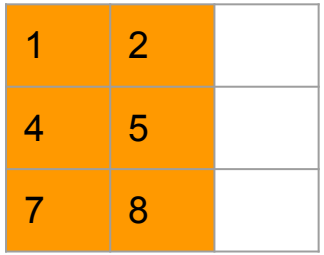
axis 1

| axis 0 | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0,0 | 0,1 | 0,2 |
| 1 | 1,0 | 1,1 | 1,2 |
| 2 | 2,0 | 2,1 | 2,2 |

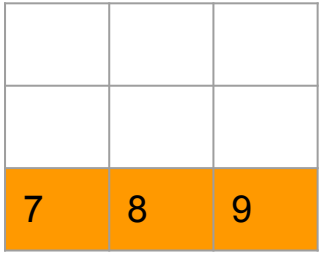
# Basic Indexing and Slicing (cont...)



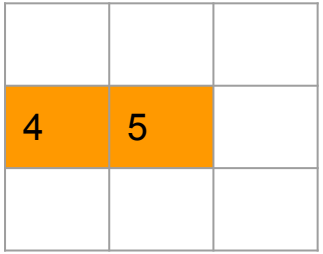

```
>>> arr2d[:2,1:]
array([[2, 3],
       [5, 6]])
```




```
>>> arr2d[:,:2]
array([[1, 2],
       [4, 5],
       [7, 8]])
```




```
>>> arr2d[2]
array([7, 8, 9])
>>> arr2d[2, :]
array([7, 8, 9])
>>> arr2d[2:, :]
array([[7, 8, 9]])
>>>
```




```
>>> arr2d[1,:2]
array([4, 5])
>>> arr2d[1:2,:2]
array([[4, 5]])
```

# Data Visualization - Plotting

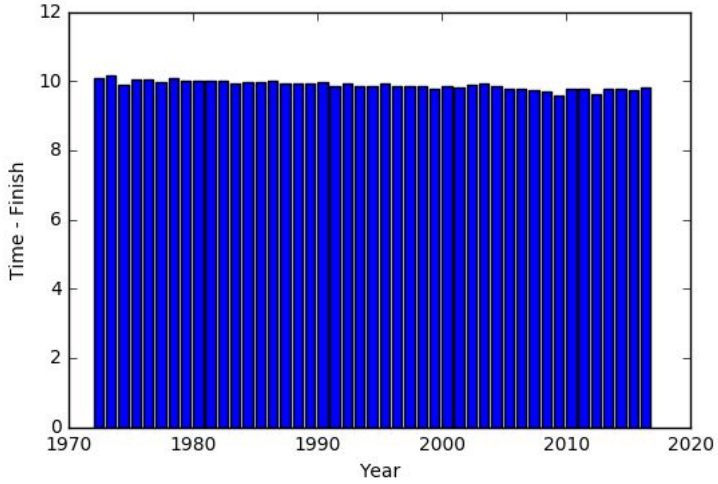
**matplotlib.pyplot** is a collection of command style functions that helps draw graphs

Types of Graph

- Bar Graphs
- Box and Whiskers (Boxplots)
- Frequency Distribution
- Histogram
- Line Graphs
- Pie Graphs
- Scatter Graphs
- Stemplots

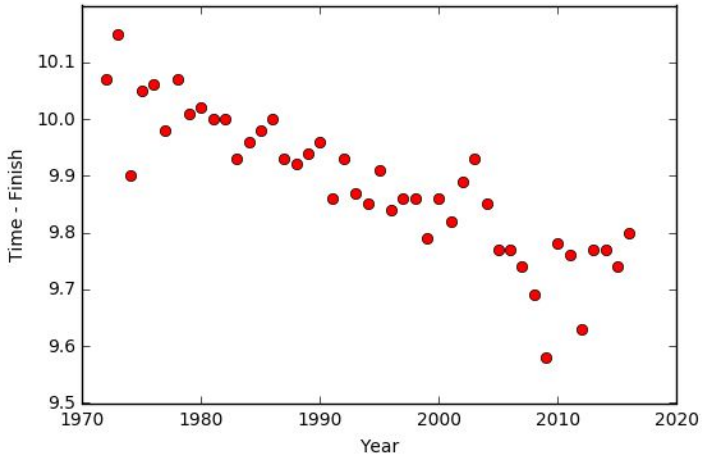
# Simple Bar Graph

```
import re
import numpy as np
year = []
finish_time = []
with open("100m_running.csv", "r") as file:
    for line in file:
        line = re.sub("\s+", "", line)
        line_elements = line.split(",")
        if line_elements[0] != '':
            year.append(int(line_elements[0]))
        if line_elements[1] != '':
            finish_time.append(float(line_elements[1]))
# Here I have two lists. year & speed_time
import matplotlib.pyplot as plt
plt.bar(year, finish_time)
plt.xlabel("Year")
plt.ylabel("Time - Finish")
plt.show()
```



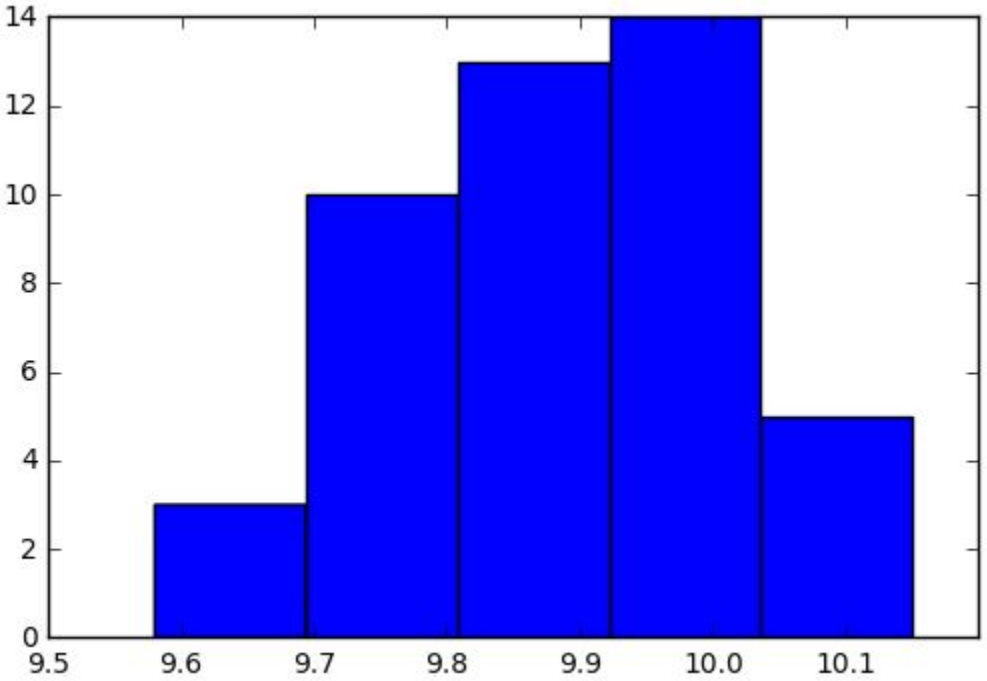
# Simple Plot

```
import re
import numpy as np
year = []
finish_time = []
with open("100m_running.csv", "r") as file:
    for line in file:
        line = re.sub("\s+", "", line)
        line_elements = line.split(",")
        if line_elements[0] != '':
            year.append(int(line_elements[0]))
        if line_elements[1] != '':
            finish_time.append(float(line_elements[1]))
# Here I have two lists. year & speed_time
import matplotlib.pyplot as plt
plt.plot(year, finish_time, 'ro')
plt.xlabel("Year")
plt.ylabel("Time - Finish")
plt.show()
```



# Histogram - Finish Time

```
plt.hist(finish_time, bins=5)
plt.show()
```

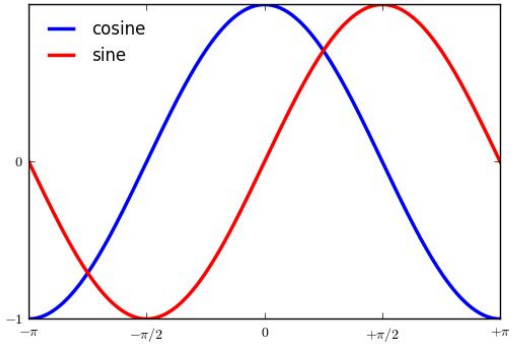


# Customizing Graphs

```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C,S = np.cos(X), np.sin(X)
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],[r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
plt.yticks([-1, 0, +1],[r'$-1$', r'$0$', r'$+1$'])
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="cosine")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-", label="sine")
plt.legend(loc='upper left', frameon=False)

plt.show()
```



# Google Stock Price

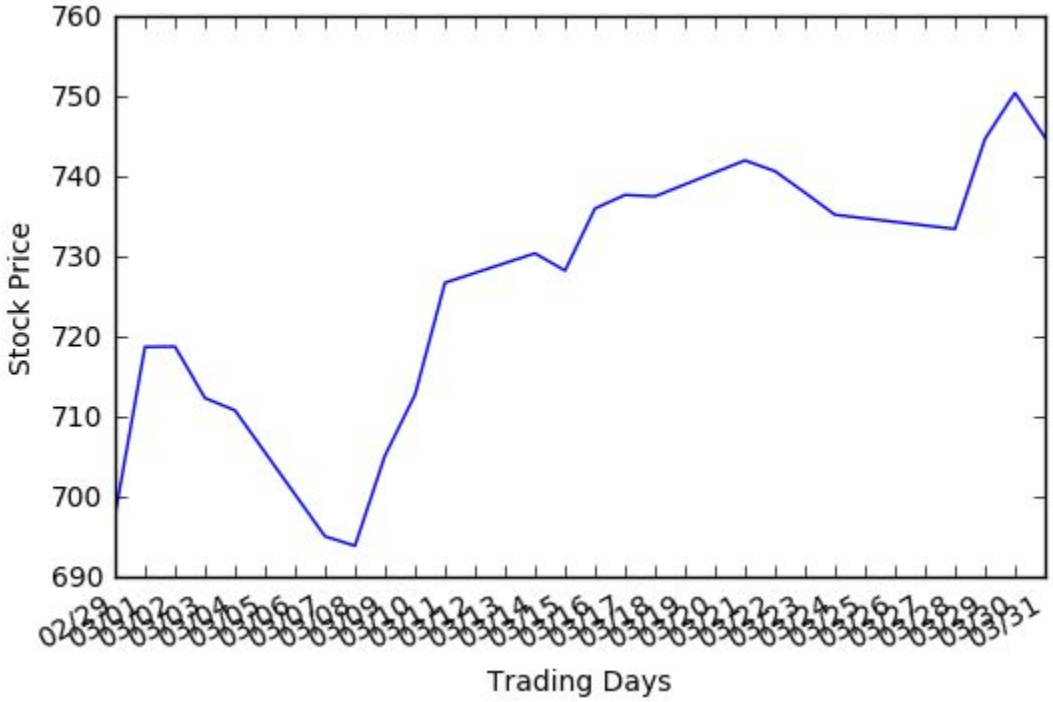
```
import datetime as dt
prices = []
dates = []
with open('GOOG.txt', "r") as f:
    for line in f:
        data = line.split(",")
        prices.append(data[1])
        day_y = data[0][0:4]
        day_m = data[0][4:6]
        day_d = data[0][6:8]
        dates.append(str(day_m) + '/' + str(day_d) + '/' + str(day_y))

days = [dt.datetime.strptime(d,'%m/%d/%Y').date() for d in dates]

import matplotlib.pyplot as plt
import matplotlib.dates as mdates

plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d'))
plt.gca().xaxis.set_major_locator(mdates.DayLocator())
plt.xlabel("Trading Days")
plt.ylabel("Stock Price")
plt.plot(days,prices)
plt.gcf().autofmt_xdate()
plt.show()
```

# Google Stock Price (cont...)



# Pandas Data Structures

- Series
- DataFrame

Both of them are built on the top of Numpy (... so are fast)

# Series

- A Series is a one-dimensional object similar to an array, list, or column in a table.
- It will assign a labeled index to each item in the Series.
- By default, each item will receive an index label from 0 to N, where N is the length of the Series minus one.

```
>>> import pandas as pd
>>> s = pd.Series(['Usain Bolt', 9.63, 'Yohan Blake', 9.75, 'Justin Gatlin', 9.79])
>>> s
0       Usain Bolt
1             9.63
2      Yohan Blake
3             9.75
4    Justin Gatlin
5             9.79
dtype: object
>>>
```

# Series (Cont...)

```
>>> s = pd.Series(['Usain Bolt', 9.63, 'Yohan Blake', 9.75, 'Justin Gatlin', 9.79], index=['A', 'B', 'C', 'D', 'E', 'F'])
>>> s
A       Usain Bolt
B             9.63
C      Yohan Blake
D             9.75
E    Justin Gatlin
F             9.79
dtype: object
>>>
```

**You can also use Dictionary to create Series**

```
>>> runners  = {'Usain Bolt': 9.63, 'Yohan Blake': 9.75, 'Justin Gatlin': 9.79}
>>> runners
{'Yohan Blake': 9.75, 'Usain Bolt': 9.63, 'Justin Gatlin': 9.79}
>>> r = pd.Series(runners)
>>> r
Justin Gatlin    9.79
Usain Bolt       9.63
Yohan Blake      9.75
dtype: float64
>>>
```

# Series (Cont...)

## Accessing Elements are Easy

```
>>> r['Usain Bolt']
9.6300000000000008
>>> r['Yohan Blake']
9.75
>>>
```

```
>>> r[['Yohan Blake', 'Yohan Blake']]
Yohan Blake    9.75
Yohan Blake    9.75
dtype: float64
>>>
>>> r[['Yohan Blake', 'Usain Bolt']]
Yohan Blake    9.75
Usain Bolt     9.63
dtype: float64
>>>
```

```
>>> runners  =  {'Usain Bolt': 9.63, 'Yohan Blake': 9.75, 'Justin Gatlin': 9.79, 'Tyson Gay': 9.8, 'Ryan Bailey': 9.88, 'Churandy Martina': 9.94, 'Richard T
ompson': 9.98, 'Asafa Powell': 11.99}
>>>
>>> r = pd.Series(runners)
>>> r
Asafa Powell        11.99
Churandy Martina     9.94
Justin Gatlin        9.79
Richard Thompson     9.98
Ryan Bailey          9.88
Tyson Gay            9.80
Usain Bolt           9.63
Yohan Blake          9.75
dtype: float64
>>> r[r>10]
Asafa Powell    11.99
dtype: float64
>>> r[r<9.8]
Justin Gatlin    9.79
Usain Bolt       9.63
Yohan Blake      9.75
dtype: float64
>>>
```

# Series (Cont...)

```
>>> r <9.8
Asafa Powell         False
Churandy Martina     False
Justin Gatlin         True
Richard Thompson     False
Ryan Bailey          False
Tyson Gay            False
Usain Bolt            True
Yohan Blake           True
dtype: bool
>>> my_requirement = r<9.8
>>> r[my_requirement]
Justin Gatlin    9.79
Usain Bolt       9.63
Yohan Blake      9.75
dtype: float64
>>>
```

Actually r<9.8 or r > 10
are series with True & False values.

Those can be passed to r - returning corresponding “True” value Series.

# Series(

One can change values on the fly

You can also check the presence of the value

```
>>> print ('Ironman' in r)
False
>>> print ('Superman' in r)
True
>>> print ('Milkha Singh' in r)
False
>>>
```

```
>>> r[['Superman']]
Superman   NaN
dtype: float64
>>> r['Superman']=5.0
>>> r['Batman'] = 6.0
>>> r
Asafa Powell         11.99
Churandy Martina      9.94
Justin Gatlin         9.79
Richard Thompson      9.98
Ryan Bailey           9.88
Tyson Gay             9.80
Usain Bolt            9.63
Yohan Blake           9.75
Superman              5.00
Batman                6.00
dtype: float64
>>> r[r<9] = 1.0
>>> r
Asafa Powell         11.99
Churandy Martina      9.94
Justin Gatlin         9.79
Richard Thompson      9.98
Ryan Bailey           9.88
Tyson Gay             9.80
Usain Bolt            9.63
Yohan Blake           9.75
Superman              1.00
Batman                1.00
dtype: float64
>>>
```

# Series (Cont...)

Mathematics

```
>>> r * r
Asafa Powell          143.7601
Churandy Martina       98.8036
Justin Gatlin          95.8441
Richard Thompson       99.6004
Ryan Bailey            97.6144
Tyson Gay              96.0400
Usain Bolt             92.7369
Yohan Blake            95.0625
Superman                1.0000
Batman                  1.0000
dtype: float64
```

```
>>> r/2
Asafa Powell          5.995
Churandy Martina      4.970
Justin Gatlin         4.895
Richard Thompson      4.990
Ryan Bailey           4.940
Tyson Gay             4.900
Usain Bolt            4.815
Yohan Blake           4.875
Superman              0.500
Batman                0.500
dtype: float64
```

```
>>> r + r
Asafa Powell          23.98
Churandy Martina      19.88
Justin Gatlin         19.58
Richard Thompson      19.96
Ryan Bailey           19.76
Tyson Gay             19.60
Usain Bolt            19.26
Yohan Blake           19.50
Superman               2.00
Batman                 2.00
dtype: float64
```

```
>>> np.square(r)
Asafa Powell          143.7601
Churandy Martina       98.8036
Justin Gatlin          95.8441
Richard Thompson       99.6004
Ryan Bailey            97.6144
Tyson Gay              96.0400
```

# Series (Cont...)

Mathematics

```
>>> new_r = pd.Series({'Lalita Baber': 15.0,  'O P Jaisha': 16.0, 'Kavita Raut': 20.0
>>> new_r
Kavita Raut     20.0
Lalita Baber    15.0
O P Jaisha      16.0
dtype: float64
```

```
>>> r + new_r
Asafa Powell       NaN
Batman             NaN
Churandy Martina   NaN
Justin Gatlin      NaN
Kavita Raut        NaN
Lalita Baber       NaN
O P Jaisha         NaN
Richard Thompson   NaN
Ryan Bailey        NaN
Superman           NaN
Tyson Gay          NaN
Usain Bolt         NaN
Yohan Blake        NaN
dtype: float64
```

# Series (Cont...)

```
>>> new_r = pd.Series({'Lalita Baber': 15.0,  'O P Jaisha': 16.0, 'Kavita Raut': 20.0, 'Usain Bolt':9.63})
>>> r + new_r
Asafa Powell          NaN
Batman                NaN
Churandy Martina      NaN
Justin Gatlin         NaN
Kavita Raut           NaN
Lalita Baber          NaN
O P Jaisha            NaN
Richard Thompson      NaN
Ryan Bailey           NaN
Superman              NaN
Tyson Gay             NaN
Usain Bolt          19.26
Yohan Blake           NaN
dtype: float64
>>>
```

```
>>> r
Asafa Powell        11.99
Churandy Martina     9.94
Justin Gatlin        9.79
Richard Thompson     9.98
Ryan Bailey          9.88
Tyson Gay            9.80
Usain Bolt           9.63
Yohan Blake          9.75
Superman             1.00
Batman               1.00
dtype: float64
```

# Series (Cont ...)

isnull & notnull instance methods

```
>>> new_r = pd.Series({'Lalita Baber': 15.0,  'O P Jaisha': 16.0, 'Kavita Raut': 20.0, 'Usain Bolt':9.63,  'Milkha Singh':None})
>>> new_r.isnull()
Kavita Raut     False
Lalita Baber    False
Milkha Singh     True
O P Jaisha      False
Usain Bolt      False
dtype: bool
>>> new_r.notnull()
Kavita Raut      True
Lalita Baber     True
Milkha Singh    False
O P Jaisha       True
Usain Bolt       True
dtype: bool
>>>
```

# Task - 3

- Use Fruits Data - create dictionary to work on Protein details.
  - Use Fruits Names as Keys and values as Protein
  - Name dictionary as - “fruits”
- Create Series from “fruits” - call it “s_fruits”
- Perform
  - >, multiplication, np.square, addition, set values on the fly.
- Add “Onion” as key, with value= None, Recreate “s_fruits” Series
  - Observe “NaN” value
- Create new dictionary - “new_fruits” with keys “Raisins”, “Onion” (use same values of Protein as above)
- Perform “s_fruits” + “new_fruits”

# DataFrame

"tabular" data: a data structure representing cases (rows), each of which consists of a number of observations or measurements (columns).

| Player Name | Time | Height (Inches) | Weight (Pounds) | Country |
|---|---|---|---|---|
| Usain Bolt | 9.63 | 76.77 | 209.439 | JAM |
| Yohan Blake | 9.75 | 70.86 | 167.551 | JAM |
| Justin Gatlin | 9.79 | 72.83 | 182.984 | USA |
| Tyson Gay | 9.8 | 70.07 | 165.347 | USA |
| Ryan Bailey | 9.88 | 75.98 | 216.053 | USA |
| Churandy Martina | 9.94 | 70.07 | 163.142 | NED |
| Richard Thompson | 9.98 | 70.01 | 176.37 | TTO |
| Asafa Powell | 11.99 | 70.8 | 191.802 | JAM |

# DataFrame

```
>>> data = {'name':['Usain Bolt','Yohan Blake','Justin Gatlin','Tyson Gay','Ryan Bailey','Churandy Martina','Richard Thompson','Asafa Powell'],
...         'speed':[9.63,9.75,9.79,9.8,9.88,9.94,9.98,11.99],
...         'height':[76.77,70.86,72.83,70.07,75.98,70.07,70.01,70.8],
...         'weight':[209.439,167.551,182.984,165.347,216.053,163.142,176.37,191.802]}
>>>
```

```
>>> data
{'speed': [9.63, 9.75, 9.79, 9.8, 9.88, 9.94, 9.98, 11.99], 'name': ['Usain Bolt', 'Yohan Blake', 'Justin Gatlin', 'Tyson Gay', 'Ryan Bailey', 'Churandy Mart
ina', 'Richard Thompson', 'Asafa Powell'], 'height': [76.77, 70.86, 72.83, 70.07, 75.98, 70.07, 70.01, 70.8], 'weight': [209.439, 167.551, 182.984, 165.347,
216.053, 163.142, 176.37, 191.802]}
>>>
```

```
>>> runners = pd.DataFrame(data, columns=["name", "speed", "height", "weight"])
>>> runners
                name  speed  height   weight
0        Usain Bolt   9.63   76.77  209.439
1       Yohan Blake   9.75   70.86  167.551
2     Justin Gatlin   9.79   72.83  182.984
3         Tyson Gay   9.80   70.07  165.347
4       Ryan Bailey   9.88   75.98  216.053
5  Churandy Martina   9.94   70.07  163.142
6  Richard Thompson   9.98   70.01  176.370
7      Asafa Powell  11.99   70.80  191.802
>>>
```

# DataFrame

```
>>> food = pd.read_csv("food.csv")
>>> food
                    Food  Index  Calories  Cholesterol  Total_Fat  Sodium  \
0          Frozen Broccoli      1      73.8          0.0        0.8    68.2
1              Carrots,Raw      2      23.7          0.0        0.1    19.2
2              Celery, Raw      3       6.4          0.0        0.1    34.8
3              Frozen Corn      4      72.2          0.0        0.6     2.5
4      Lettuce,Iceberg,Raw      5       2.6          0.0        0.0     1.8
5      Peppers, Sweet, Raw      6      20.0          0.0        0.1     1.5
6          Potatoes, Baked      7     171.5          0.0        0.2    15.2
7                     Tofu      8      88.2          0.0        5.5     8.1
8          Roasted Chicken      9     277.4        129.9       10.8   125.6
9       Spaghetti W/ Sauce     10     358.2          0.0       12.3  1237.1
10     Tomato,Red,Ripe,Raw     11      25.8          0.0        0.4    11.1
11        Apple,Raw,W/Skin     12      81.4          0.0        0.5     0.0
12                  Banana     13     104.9          0.0        0.5     1.1
13                  Grapes     14      15.1          0.0        0.1     0.5
14     Kiwifruit,Raw,Fresh     15      46.4          0.0        0.3     3.8
15                 Oranges     16      61.6          0.0        0.2     0.0
16                  Bagels     17      78.0          0.0        0.5   151.4
17            Wheat Bread     18      65.0          0.0        1.0   134.5
18            White Bread     19      65.0          0.0        1.0   132.5
19          Oatmeal Cookies     20      81.0          0.0        3.3    68.9
20               Apple Pie     21      67.2          0.0        2.1    75.4
```

# DataFrame

```
>>> food = pd.read_csv("food.csv")
>>> food
                    Food  Index  Calories  Cholesterol  Total_Fat  Sodium  \
0          Frozen Broccoli      1      73.8          0.0        0.8    68.2
1              Carrots,Raw      2      23.7          0.0        0.1    19.2
2              Celery, Raw      3       6.4          0.0        0.1    34.8
3              Frozen Corn      4      72.2          0.0        0.6     2.5
4      Lettuce,Iceberg,Raw      5       2.6          0.0        0.0     1.8
5      Peppers, Sweet, Raw      6      20.0          0.0        0.1     1.5
6          Potatoes, Baked      7     171.5          0.0        0.2    15.2
7                     Tofu      8      88.2          0.0        5.5     8.1
8          Roasted Chicken      9     277.4        129.9       10.8   125.6
9       Spaghetti W/ Sauce     10     358.2          0.0       12.3  1237.1
10     Tomato,Red,Ripe,Raw     11      25.8          0.0        0.4    11.1
11        Apple,Raw,W/Skin     12      81.4          0.0        0.5     0.0
12                  Banana     13     104.9          0.0        0.5     1.1
13                  Grapes     14      15.1          0.0        0.1     0.5
14     Kiwifruit,Raw,Fresh     15      46.4          0.0        0.3     3.8
15                 Oranges     16      61.6          0.0        0.2     0.0
16                  Bagels     17      78.0          0.0        0.5   151.4
17            Wheat Bread     18      65.0          0.0        1.0   134.5
18            White Bread     19      65.0          0.0        1.0   132.5
19          Oatmeal Cookies     20      81.0          0.0        3.3    68.9
20               Apple Pie     21      67.2          0.0        2.1    75.4
```

# DataFrame

```
>>> food.head()
                Food  Index  Calories  Cholesterol  Total_Fat  Sodium  \
0     Frozen Broccoli      1      73.8          0.0        0.8    68.2
1         Carrots,Raw      2      23.7          0.0        0.1    19.2
2         Celery, Raw      3       6.4          0.0        0.1    34.8
3         Frozen Corn      4      72.2          0.0        0.6     2.5
4  Lettuce,Iceberg,Raw      5       2.6          0.0        0.0     1.8

   Carbohydrates  Dietary_Fiber  Protein    Vit_A  Vit_C  Calcium  Iron  \
0           13.6            8.5      8.0   5867.4  160.2    159.0   2.3
1            5.6            1.6      0.6  15471.0    5.1     14.9   0.3
2            1.5            0.7      0.3     53.6    2.8     16.0   0.2
3           17.1            2.0      2.5    106.6    5.2      3.3   0.3
4            0.4            0.3      0.2     66.0    0.8      3.8   0.1

   Price/Serving ($)
0               0.16
1               0.07
2               0.04
3               0.18
4               0.02
>>> food.tail(1)
                  Food  Index  Calories  Cholesterol  Total_Fat  Sodium  \
63  Beanbacn Soup,W/Watr     64     172.0          2.5        5.9   951.3

    Carbohydrates  Dietary_Fiber  Protein  Vit_A  Vit_C  Calcium  Iron  \
63           22.8            8.6      7.9  888.0    1.5     81.0   2.0

    Price/Serving ($)
63               0.67
>>> food.tail(3)
                  Food  Index  Calories  Cholesterol  Total_Fat  Sodium  \
61  New E Clamchwd,W/Mlk     62     163.7         22.3        6.6   992.0
62  Crm Mshrm Soup,W/Mlk     63     203.4         19.8       13.6  1076.3
```

- head()
- tail()
- info()
- describe()
- dtype
- columns

# DataFrame

```
>>> food.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64 entries, 0 to 63
Data columns (total 14 columns):
Food                64 non-null object
Index               64 non-null int64
Calories            64 non-null float64
Cholesterol         64 non-null float64
Total_Fat           64 non-null float64
Sodium              64 non-null float64
Carbohydrates       64 non-null float64
Dietary_Fiber       64 non-null float64
Protein             64 non-null float64
Vit_A               64 non-null float64
Vit_C               64 non-null float64
Calcium             64 non-null float64
Iron                64 non-null float64
Price/Serving ($)   64 non-null float64
dtypes: float64(12), int64(1), object(1)
memory usage: 7.1+ KB
>>>
```

```
>>> runners.describe()
            speed     height      weight
count    8.000000   8.000000    8.000000
mean    10.095000  72.173750  184.086000
std      0.773656   2.756555   20.171263
min      9.630000  70.010000  163.142000
25%      9.780000  70.070000  167.000000
50%      9.840000  70.830000  179.677000
75%      9.950000  73.617500  196.211250
max     11.990000  76.770000  216.053000
>>>
```

```
>>> import pandas as pd
>>> raw_data = {
...             'subject': ['1', '2', '3', '4', '5'],
...             'first_name': ['Michel', 'Mark', 'Matt', 'Ryan', 'Gary'],
...             'last_name': ['Phelps', 'Spitz', 'Biondi', 'Lochte', 'Hall']
...           }
>>>
>>> raw_data
{'subject': ['1', '2', '3', '4', '5'], 'first_name': ['Michel', 'Mark', 'Matt', 'Ryan', 'Gary'], 'last_name': ['Phelps', 'Spitz', 'Biondi', 'Lochte', 'Hall'
]}
>>> df_a = pd.DataFrame(raw_data, columns = ['subject', 'first_name', 'last_name'])
>>> df_a
  subject first_name last_name
0       1     Michel    Phelps
1       2       Mark     Spitz
2       3       Matt    Biondi
3       4       Ryan    Lochte
4       5       Gary      Hall
>>>
```

```
>>> raw_data = {
...             'subject': ['4', '5', '6', '7', '8', '9'],
...             'first_name': ['Ian', 'Aaron', 'Nathan', 'Tom', 'Don', 'Johny'],
...             'last_name': ['Thorpe', 'Peirsol', 'Adrian', 'Jager', 'Schollander', 'Weis']
...            }
>>> df_b = pd.DataFrame(raw_data, columns = ['subject', 'first_name', 'last_name'])
>>> df_b
  subject first_name    last_name
0       4        Ian       Thorpe
1       5      Aaron      Peirsol
2       6     Nathan       Adrian
3       7        Tom        Jager
4       8        Don  Schollander
5       9      Johny         Weis
>>>
```

```
>>> raw_data = {
...             'subject': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
...             'test'   : [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]
...            }
>>> df_n = pd.DataFrame(raw_data, columns = ['subject','test'])
>>> df_n
  subject  test
0       1    51
1       2    15
2       3    15
3       4    61
4       5    16
5       7    14
6       8    15
7       9     1
8      10    61
9      11    16
>>>
>>>
```

# Join Two DataFrames - along rows

```
>>> df_new = pd.concat([df_a, df_b])
>>> df_new
  subject first_name    last_name
0       1     Michel       Phelps
1       2       Mark        Spitz
2       3       Matt       Biondi
3       4       Ryan       Lochte
4       5       Gary         Hall
0       4        Ian       Thorpe
1       5      Aaron      Peirsol
2       6     Nathan       Adrian
3       7        Tom        Jager
4       8        Don  Schollander
5       9      Johny         Weis
>>>
```

# Join Two DataFrames - along rows

```
>>> df_new = pd.concat([df_a, df_b])
>>> df_new
  subject first_name    last_name
0       1     Michel       Phelps
1       2       Mark        Spitz
2       3       Matt       Biondi
3       4       Ryan       Lochte
4       5       Gary         Hall
0       4        Ian       Thorpe
1       5      Aaron      Peirsol
2       6     Nathan       Adrian
3       7        Tom        Jager
4       8        Don  Schollander
5       9      Johny         Weis
>>>
```

# Join Two Dataframes - along axis

```
>>>
>>> pd.concat([df_a, df_b], axis=1)
   subject first_name last_name subject first_name    last_name
0        1     Michel    Phelps       4        Ian       Thorpe
1        2       Mark     Spitz       5      Aaron      Peirsol
2        3       Matt    Biondi       6     Nathan       Adrian
3        4       Ryan    Lochte       7        Tom        Jager
4        5       Gary      Hall       8        Don  Schollander
5      NaN        NaN       NaN       9      Johny        Weis
>>>
```

# Merge Two DataFrames - along one column

```
>>> pd.merge(df_new, df_n, on='subject')
   subject first_name    last_name  test
0        1     Michel       Phelps    51
1        2       Mark        Spitz    15
2        3       Matt       Biondi    15
3        4       Ryan       Lochte    61
4        4        Ian       Thorpe    61
5        5       Gary         Hall    16
6        5      Aaron      Peirsol    16
7        7        Tom        Jager    14
8        8        Don  Schollander    15
9        9      Johny         Weis     1
>>>
```

# Merge dataframes with both the left and right dataframes using the one key

```
>>> pd.merge(df_new, df_n, left_on='subject', right_on='subject')
   subject first_name    last_name  test
0        1     Michel       Phelps    51
1        2       Mark        Spitz    15
2        3       Matt       Biondi    15
3        4       Ryan       Lochte    61
4        4        Ian       Thorpe    61
5        5       Gary         Hall    16
6        5      Aaron      Peirsol    16
7        7        Tom        Jager    14
8        8        Don  Schollander    15
9        9      Johny         Weis     1
>>>
```

# Merge with outer join

```
>>> pd.merge(df_a, df_b, on='subject', how='outer')
   subject first_name_x last_name_x first_name_y  last_name_y
0        1       Michel      Phelps          NaN          NaN
1        2         Mark       Spitz          NaN          NaN
2        3         Matt      Biondi          NaN          NaN
3        4         Ryan      Lochte          Ian       Thorpe
4        5         Gary        Hall       Aaron      Peirsol
5        6          NaN         NaN      Nathan       Adrian
6        7          NaN         NaN         Tom        Jager
7        8          NaN         NaN         Don  Schollander
8        9          NaN         NaN       Johny         Weis
>>>
```

# Merge with Inner Join

```
>>> pd.merge(df_a, df_b, on='subject', how='inner')
  subject first_name_x last_name_x first_name_y last_name_y
0       4         Ryan      Lochte          Ian      Thorpe
1       5         Gary        Hall        Aaron     Peirsol
>>>
```

# Left and Right Join

```
>>> pd.merge(df_a, df_b, on='subject', how='right')
   subject first_name_x last_name_x first_name_y  last_name_y
0        4         Ryan      Lochte          Ian       Thorpe
1        5         Gary        Hall       Aaron      Peirsol
2        6          NaN         NaN      Nathan       Adrian
3        7          NaN         NaN         Tom        Jager
4        8          NaN         NaN         Don  Schollander
5        9          NaN         NaN       Johny         Weis
>>> pd.merge(df_a, df_b, on='subject', how='left')
   subject first_name_x last_name_x first_name_y last_name_y
0        1       Michel      Phelps          NaN         NaN
1        2         Mark       Spitz          NaN         NaN
2        3         Matt      Biondi          NaN         NaN
3        4         Ryan      Lochte          Ian      Thorpe
4        5         Gary        Hall       Aaron     Peirsol
>>>
>>>
```

# Add Suffixes

```
>>> pd.merge(df_a, df_b, on='subject', how='left', suffixes=('_left', '_right'))
  subject first_name_left last_name_left first_name_right last_name_right
0       1          Michel         Phelps              NaN             NaN
1       2            Mark          Spitz              NaN             NaN
2       3            Matt         Biondi              NaN             NaN
3       4            Ryan         Lochte              Ian          Thorpe
4       5            Gary           Hall           Aaron         Peirsol
>>>
```

# Yield - Generator

```
$ more demo_yield.py
def createGenerator():
    mylist = range(3)
    for i in mylist:
        yield i*i

mygenerator = createGenerator()
print(mygenerator)
print(type(mygenerator))


#for i in mygenerator:
#    print (i)

print (next(mygenerator))
print (next(mygenerator))
print (next(mygenerator))
```

demo_yeild.py

```
$ python3 demo_yield.py
<generator object createGenerator at 0x7f605b3760f8>
<class 'generator'>
0
1
4
$
```

# A few Libraries/Modules - logging

What's Logging? How does it help in case Application Crash? Servers Crash?

There are different importance levels - [1] debug, [2] info, [3] warning, [4] error and [5] critical

```
#!/usr/bin/python3
import logging

LOG_FILENAME = 'logging_example.out'
logging.basicConfig(
    filename=LOG_FILENAME,
    level=logging.DEBUG,
)

logging.debug('This message should go to the log file')
```

# A few Libraries/Modules - logging (Cont...)

```
try:
    open('/path/to/does/not/exist', 'rb')
except (SystemExit, KeyboardInterrupt):
    raise
except Exception, e:
    logger.error('Failed to open file', exc_info=True)



ERROR:__main__:Failed to open file
Traceback (most recent call last):
  File "example.py", line 6, in <module>
    open('/path/to/does/not/exist', 'rb')
IOError: [Errno 2] No such file or directory: '/path/to/does/not/exist'
```

# LAB Assignment

Study - https://dev.mysql.com/doc/employee/en/

Analyse Data - using Python, Pandas, Numpy!

Open-ended Problem - submit your Analysis Report and programs/scripts created to support your analysis.

# Thank you!