

**DATA 301**  
**Introduction to Data Analytics**  
**Python Data Analytics**

Dr. Ramon Lawrence  
University of British Columbia Okanagan  
ramon.lawrence@ubc.ca

# Python File Input/Output

---

Many data processing tasks require reading and writing to files.

Open a file for reading:

```
infile = open("input.txt", "r")
```

I/O Type



Open a file for writing:

```
outfile = open("output.txt", "w")
```

Open a file for read/write:

```
myfile = open("data.txt", "r+")
```

# Reading from a Text File (as one String)

---

```
infile = open("input.txt", "r")
```

```
val = infile.read() ← Read all file as one string  
print(val)
```

```
infile.close() ← Close file
```



## Reading from a Text File (line by line)

---

```
infile = open("input.txt", "r")
for line in infile:
    print(line.strip('\n'))
infile.close()
```

```
# Alternate syntax - will auto-close file
with open("input.txt", "r") as infile:
    for line in infile:
        print(line.strip('\n'))
```

# Writing to a Text File

---

```
outfile = open("output.txt", "w")

for n in range(1,11):
    outfile.write(str(n) + "\n")

outfile.close()
```

# Other File Methods

---

```
infile = open("input.txt", "r")

# Check if a file is closed
print(infile.closed) # False

# Read all lines in the file into a list
lines = infile.readlines()
infile.close()
print(infile.closed) # True
```

# Use Split to Process a CSV File

---

```
with open("data.csv", "r") as infile:
    for line in infile:
        line = line.strip(" \n")
        fields = line.split(",")
        for i in range(0, len(fields)):
            fields[i] = fields[i].strip()
        print(fields)
```

# Using `csv` Module to Process a CSV File

---

```
import csv

with open("data.csv", "r") as infile:
    csvfile = csv.reader(infile)
    for row in csvfile:
        if int(row[0]) > 1:
            print(row)
```



# List all Files in a Directory

---

```
import os  
print(os.listdir("."))
```

# Python File I/O Question

---

**Question:** How many of the following statements are **TRUE**?

- 1) A Python file is automatically closed for you.
- 2) If you use the `with` syntax, Python will close the file for you.
- 3) To read from a file, use `w` when opening a file.
- 4) The `read()` method will read the entire file into a string.
- 5) You can use a `for` loop to iterate through all lines in a file.

**A) 0**      **B) 1**      **C) 2**      **D) 3**      **E) 4**

# Try it: Python Files

---

**Question 1:** Write a Python program that writes to the file `test.txt` the numbers from 20 to 10 in descending order.

**Question 2:** Write a Python program that reads your newly created `test.txt` file line by line and only prints out the value if it is even.

**Question 3:** Print out the contents of the CSV census file from:

[https://people.ok.ubc.ca/rlawrenc/teaching/301/notes/code/data/province\\_population.csv](https://people.ok.ubc.ca/rlawrenc/teaching/301/notes/code/data/province_population.csv)

- Try to print out only the provinces with population > 1 million people and only the 2015 data. You will need to use `float()` and remove commas in data.



# Internet Terminology Basics

---

An **Internet Protocol (IP) address** is an identifier for a computer on the Internet.

- IP version 4 (IPv4) address is 4 numbers in the range of 0 to 255. The numbers are separated by dots. Example: 142.255.0.1
- IP version 6 (IPv6) address has 16 numbers from 0 to 255 represented in hexadecimal. Example: 2002:CE57:25A2:0000:0000:0000:CE57:25A2

A **domain name** is a text name for computer(s) that are easier to remember. A **domain** is a related group of networked computers.

- Domain names are organized **hierarchically**. The most general part of the hierarchy is at the end of the name.
- Example: `people.ok.ubc.ca`
  - `ca` – Canadian domain, `ubc` – University of British Columbia, `ok` – Okanagan campus, `people` – name of computer/server on campus

# Internet Terminology Basics (2)

A **uniform resource locator (URL)** is an address of an item on the Internet. A URL has three parts:

- **Protocol:** `http://` - Hypertext Transfer Protocol
  - Tells the computer how to handle the file
- **Server computer's domain name or IP address**
- **Item's path and name:**
  - Tells the server which item (file, page, resource) is requested and where to find it.

Example:

`http://people.ok.ubc.ca/rlawrenc/teaching/301/index.html`

↑  
http protocol

↑  
server domain name

↑  
location of file/resource on server

# Accessing (GET) Web Sites via URL with Python

---

```
import urllib.request
loc="http://people.ok.ubc.ca/rlawrenc/teaching/301"
site = urllib.request.urlopen(loc)
contents = site.read()
print(contents)
site.close()
```

# Google Search with Python

---

```
import urllib
url = "http://www.google.com/search?hl=en&q=data+analysis"
headers={'User-Agent':'Mozilla/5.0 (Windows NT 6.1)'}
request = urllib.request.Request(url, None, headers)
response = urllib.request.urlopen(request)
data = response.read()
data = data.decode()      # Convert from Unicode to ASCII
print(data)
request.close()
```

# Sending Data (PUT) to URL with Python

---

```
import urllib.parse
import urllib.request

url = 'http://cosc304.ok.ubc.ca/rlawrenc/tomcat/provinceState.jsp'
headers={'User-Agent':'Mozilla/5.0 (Windows NT 6.1)'}
# Build and encode data
values = {'country' : 'US'}
data = urllib.parse.urlencode(values)
data = data.encode('ascii')
req = urllib.request.Request(url, data, headers)
with urllib.request.urlopen(req) as response:
    page = response.read()
    print(page)
```



# Python Web/URL Question

---

**Question:** How many of the following statements are **TRUE**?

- 1) An IPv4 address has 4 numbers between 0 and 256 inclusive.
- 2) A domain name is hierarchical with most specific part at the end.
- 3) Typically, a URL will reference more than one resource/item.
- 4) Python uses the file module for accessing URLs.

**A) 0**      **B) 1**      **C) 2**      **D) 3**      **E) 4**

# Try it: Python URLs

---

**Question 1:** Write a Python program that connects to any web page and prints its contents.

**Question 2:** Write a Python program that connects to:

[https://people.ok.ubc.ca/rlawrenc/teaching/301/notes/code/data/province\\_population.csv](https://people.ok.ubc.ca/rlawrenc/teaching/301/notes/code/data/province_population.csv)

and outputs the CSV data.

- Modify your program to print each province and its 2015 population in descending sorted order.

# Handling Errors and Exceptions

---

An **exception** is an error situation that must be handled or the program will fail.

- **Exception handling** is how your program deals with these errors.

Examples:

- Attempting to divide by zero
- An array index that is out of bounds
- A specified file that could not be found
- A requested I/O operation that could not be completed normally
- Attempting to follow a null or invalid reference
- Attempting to execute an operation that violates some kind of security measure

# ★ The `try-except` Statement

---

The *`try-except statement`* will handle an exception that may occur in a block of statements:

Execution flow:

- The statements in the `try` block are executed.
- If no exception occurs:
  - If there is an `else` clause, it is executed.
  - Continue on with next statement after `try`.
- If an exception occurs:
  - Execute the code after the `except`.
- If the optional `finally` block is present, it is always executed regardless if there is an exception or not.
- Keyword `pass` is used if any block has no statements.

# Python Exceptions Example

---

**try:**

```
num = int(input("Enter a number:"))  
print("You entered:", num)
```

} try block  
exit if error

**except ValueError:**

```
print("Error: Invalid number")
```

} only execute  
if exception

**else:**

```
print("Thank you for the number")
```

} only execute if  
no exception

**finally:**

```
print("Always do finally block")
```

} always  
execute

# Question: Exceptions

---

**Question:** What is the output of the following code if enter 10?

```
try:
    num = int(input("Enter num:"))
    print(num)
except ValueError:
    print("Invalid")
else:
    print("Thanks")
finally:
    print("Finally")
```

**A)** 10  
**B)** 10  
**C)** Invalid  
**D)** 10  
Thanks  
**E)** 10  
Thanks  
Finally

## Question: Exceptions (2)

---

**Question:** What is the output of the following code if enter hat?

```
try:
    num = int(input("Enter num:"))
    print(num)
except ValueError:
    print("Invalid")
else:
    print("Thanks")
print("Finally")
```

A) hat

B) Invalid

C) Invalid

Finally

D) hat

Thanks

Finally

E) Finally

# Try it: Python Exceptions

---

**Question:** Write a Python program that reads two numbers and converts them to integers, prints both numbers, and then divides the first number by the second number and prints the result.

- If get an exception `ValueError` when converting to an integer, print `Invalid`.
- If get a `ZeroDivisionError`, print `Cannot divide by 0!`



# Python Modules

---

A Python *module* or *library* is code written by others for a specific purpose. Whenever coding, make sure to look for modules that are already written for you to make your development faster!

Modules are imported using the import command:

```
import modulename
```

Useful modules for data analytics:

- Biopython (bioinformatics), NumPy (scientific computing/linear algebra), scikit-learn (machine learning), pandas (data structures), BeautifulSoup (HTML/Web)

# Biopython

---

Biopython (<http://biopython.org>) is a Python library for biological and bioinformatics computation.

## Features:

- parsers for bioinformatics file formats (BLAST, Clustalw, FASTA, Genbank)
- access to online services (NCBI - National Center for Biotechnology Information)
- sequence class
- clustering/classification (k Nearest Neighbors, Naïve Bayes, Support Vector Machines)
- Integration with BioSQL (sequence database schema)

# Biopython Installation

---

Install in Anaconda by:

```
conda install biopython
```

Check if successfully installed and current version by:

```
import Bio  
print(Bio.__version__)
```

# Biopython Example - Using Sequences

---

```
# Create a sequence as a string
```

```
from Bio.Seq import Seq
my_seq = Seq("AGTACACTGGT")
print(my_seq)
```

```
# Read a FASTA file and print sequence info
```

```
from Bio import SeqIO
for seq_record in SeqIO.parse("sequence.fasta", "fasta"):
    print(seq_record.id)
    print(repr(seq_record.seq))
    print(len(seq_record))
    print(seq_record.seq.complement())
```

# Biopython Transcription Example

---

```
# Transcription
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC

coding_dna = Seq("TGCATTGGGTGCTGA", IUPAC.unambiguous_dna)
template_dna = coding_dna.reverse_complement()
messenger_rna = coding_dna.transcribe()

print("Coding:      ", coding_dna)
print("Template:     ", template_dna)
print("Messenger RNA: ", messenger_rna)
print("Translation:   ", messenger_rna.translate())
```

# Biopython - Entrez Database Search

---

Entrez is a federated database enabling retrieval of data from many health sciences databases hosted by the NCBI.

```
# Retrieve data from nucleotide database as FASTA
from Bio import Entrez
from Bio import SeqIO
Entrez.email = "test@test.com"
# Providing GI for single entry lookup
handle = Entrez.efetch(db="nucleotide", rettype="fasta",
retmode="text", id="3288717")
record = SeqIO.read(handle, "fasta")
handle.close()
print(record)
```

# Biopython - BLAST

---

BLAST (Basic Local Alignment Search Tool) compares an input sequence with database and returns similar sequences.

<http://blast.ncbi.nlm.nih.gov/>

```
# Retrieve data from nucleotide database as FASTA
from Bio.Blast import NCBIWWW
from Bio.Blast import NCBIXML

sequence = "ACTATTCCAAACAGCTCATAACCAGAAA"
handle = NCBIWWW.qblast("blastn", "nt", sequence)

result = handle.read()
print(result)      # Output is in XML format
```



# Biopython BLAST - Parsing Results

---

```
from Bio.Blast import NCBIWWW
from Bio.Blast import NCBIXML
sequence = "ACTATTCCAAACAGCTCATAACCAGAAA"
handle = NCBIWWW.qblast("blastn", "nt", sequence)
records = NCBIXML.parse(handle)
record = next(records)
for alignment in record.alignments:
    for hsp in alignment.hsps:
        print('\nsequence:', alignment.title)
        print('length:', alignment.length)
        print('e value:', hsp.expect)
        print(hsp.query[0:75] + '...')
        print(hsp.match[0:75] + '...')
        print(hsp.sbjct[0:75] + '...')
```



# Try it: Biopython

---

**Question:** Write a program that has a DNA sequence that you create, performs a BLAST, and then outputs the top 3 hits.

# Charts

---

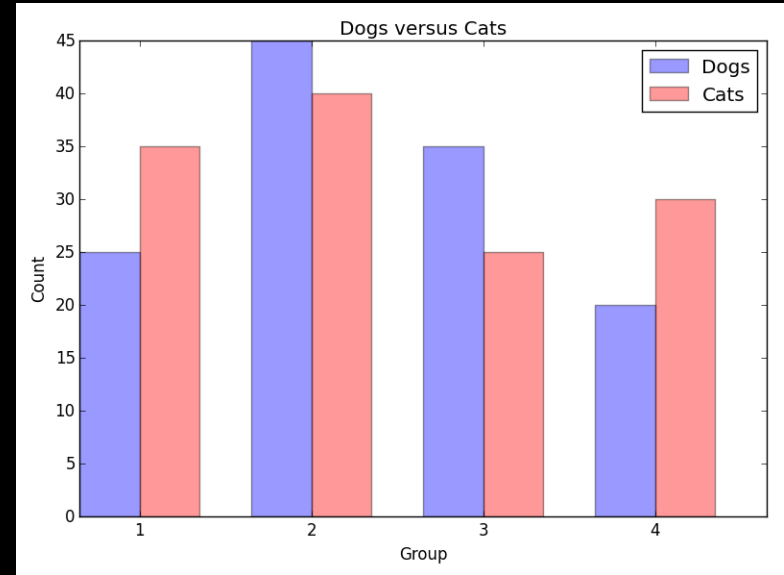
There are numerous graphing and chart libraries for Python:

- **matplotlib** (<http://matplotlib.org/>) - foundational 2D plotting library
- **ggplot** (<http://ggplot.yhathq.com/>) - based on R's ggplot2
- **pygal** - dynamic chart library
- **Bokeh** (<http://bokeh.pydata.org/>) - goal is to produce charts similar to D3.js for browsers
- **Seaborn** (<http://stanford.edu/~mwaskom/software/seaborn/>) - based on matplotlib and designed for statistical graphics

# matplotlib - Bar Chart Example

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

data1 = [25, 45, 35, 20]
data2 = [35, 40, 25, 30]
index = np.arange(len(data1))
bar_width = 0.35
opacity = 0.4
error_config = {'ecolor': '0.3'}
rects1 = plt.bar(index, data1, bar_width, alpha=opacity,
                  color='b', yerr=None, error_kw=error_config,
                  label='Dogs')
```



## matplotlib - Bar Chart Example (2)

---

```
rects2 = plt.bar(index + bar_width, data2, bar_width,
                 alpha=opacity, color='r', yerr=None,
                 error_kw=error_config, label='Cats')

plt.xlabel('Group')
plt.ylabel('Count')
plt.title('Dogs versus Cats')
plt.xticks(index + bar_width, ('1', '2', '3', '4'))
plt.legend()
plt.tight_layout()
plt.show()
```

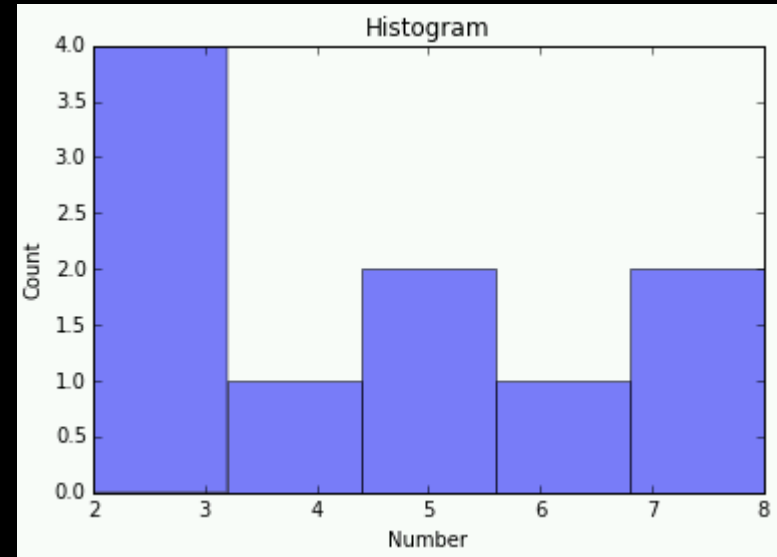
# matplotlib - Histogram Example

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

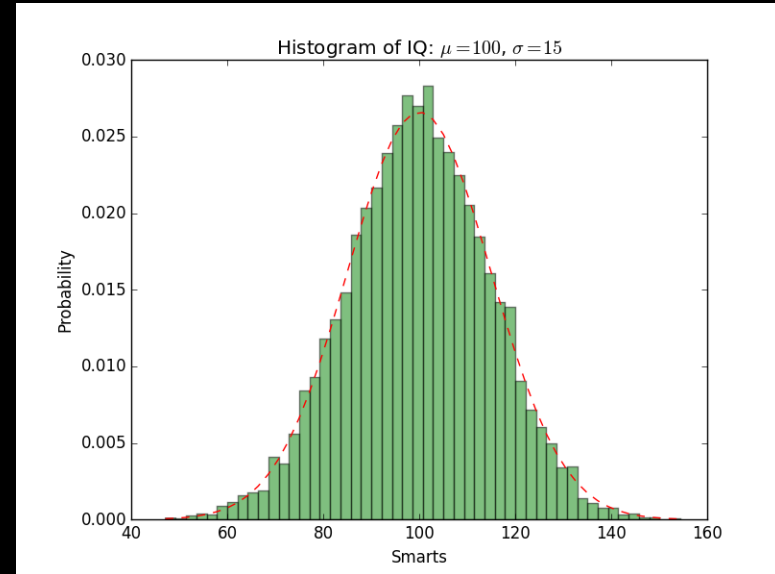
num_bins = 5
x = [5, 3, 8, 5, 2, 7, 2, 4, 6, 2]
n, bins, patches = plt.hist(x, num_bins,
                             normed=False, facecolor='blue',
                             alpha=0.5)

plt.xlabel('Number')
plt.ylabel('Count')
plt.title('Histogram')
plt.show()
```



# matplotlib - Histogram Example #2

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
mu = 100
sigma = 15
x = mu+sigma*np.random.randn(10000)
num_bins = 50
n, bins, patches = plt.hist(x, num_bins,
                             normed=1, facecolor='green',
                             alpha=0.5)
y = mlab.normpdf(bins, mu, sigma)
plt.plot(bins, y, 'r--')
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title(r'Histogram of IQ:  $\mu=100$ ,  $\sigma=15$ ')
plt.subplots_adjust(left=0.15)
plt.show()
```



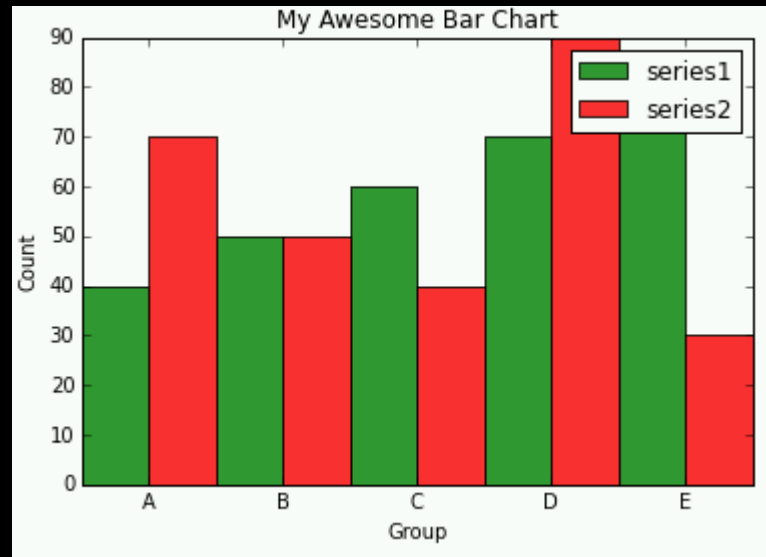
Note: Set `normed=0` to show counts rather than probabilities.

# Try it: Charts

**Question:** Write a program to create a bar chart for this data:

- series1 = [40, 50, 60, 70, 80]
- series2 = [70, 50, 40, 90, 30]

Output:



# SciPy

---

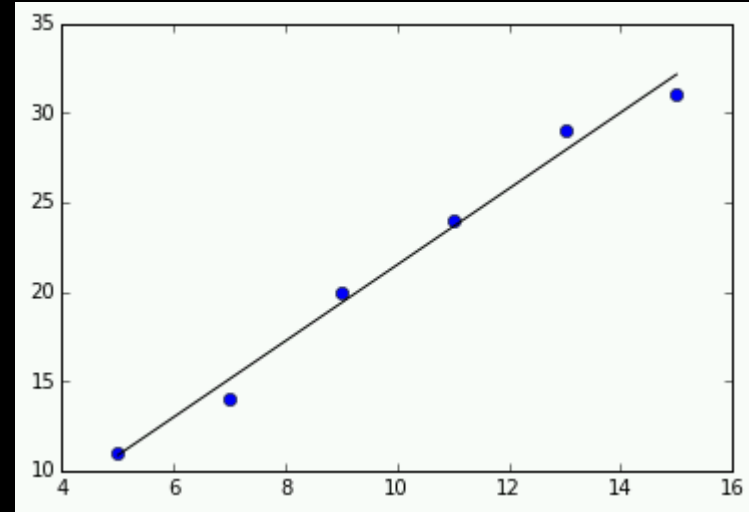
**SciPy** is group of Python libraries for scientific computing:

- NumPy (<http://www.numpy.org/>) - N-dimensional arrays, integrating C/C++ and Fortran code, linear algebra, Fourier transform, and random numbers
- SciPy (<http://www.scipy.org/>) - numerical integration and optimization
- matplotlib (<http://matplotlib.org/>) - 2D plotting library
- IPython (<http://ipython.org/>) - interactive console (Jupyter)
- SymPy (<http://www.sympy.org/>) - symbolic mathematics (equations, calculus, statistics, combinatorics, cryptography)
- pandas (<http://pandas.pydata.org/>) - data structures, reading/writing data, data merging/joining/slicing/grouping, time series



# SciPy Linear Regression Example

```
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt
x = np.array([5, 7, 9, 11, 13, 15])
y = np.array([11, 14, 20, 24, 29, 31])
slope, intercept, r_value, p_value,
    slope_std_error = stats.linregress(x, y)
predict_y = intercept + slope * x
print("Predicted y-values:", predict_y)
pred_error = y - predict_y
print("Prediction error:", pred_error)
degr_freedom = len(x) - 2
residual_std_error = np.sqrt(np.sum(pred_error**2) / degr_freedom)
print("Residual error:", residual_std_error)
plt.plot(x, y, 'o')
plt.plot(x, predict_y, 'k-')
plt.show()
```

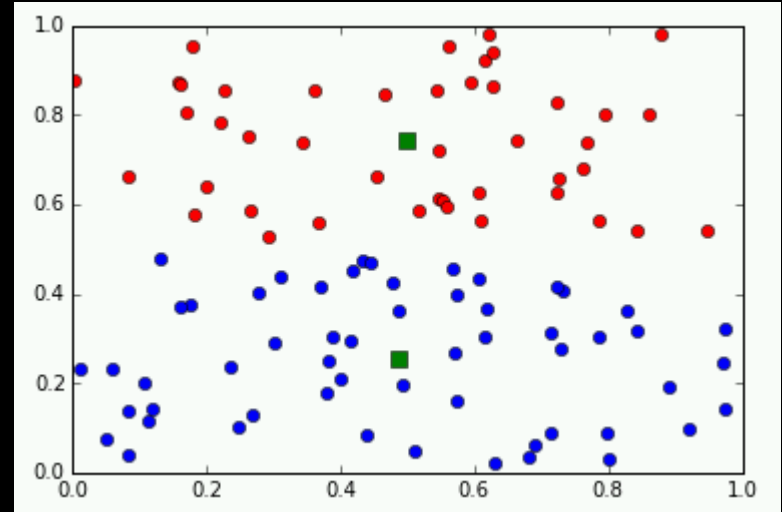


# SciPy k-Means Clustering Example

```
%matplotlib inline
import matplotlib.pyplot as plt
from scipy.cluster.vq import kmeans,vq
import random as rnd

# data generation
data = []
for i in range(0,100):
    data.append([rnd.random(), rnd.random()])

# Perform k-means clustering
numclusters = 2
centroids, _ = kmeans(data,numclusters) # Calculates centroids
idx, _ = vq(data,centroids)           # Puts each point in a cluster
```



# SciPy k-Means Clustering Example (2)

---

```
# Move data into individual lists based on clustering
clusters = []
for i in range(0, numclusters):
    clusters.append([], [])

for i in range(0, len(idx)):
    clusterIdx = idx[i]
    clusters[clusterIdx][0].append(data[i][0])
    clusters[clusterIdx][1].append(data[i][1])

# Plot data points and cluster centroids
plt.plot(clusters[0][0], clusters[0][1], 'ob',
         clusters[1][0], clusters[1][1], 'or')
plt.plot(centroids[:,0], centroids[:,1], 'sg', markersize=8)
plt.show()
```

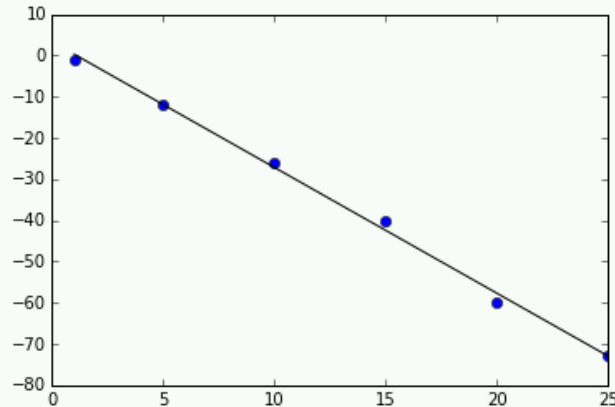
# Try it: SciPy

**Question:** Write a program that uses SciPy to perform a linear regression on this data set:

- $x = [1, 5, 10, 15, 20, 25]$
- $y = [-1, -12, -26, -40, -60, -73]$

Output:

```
Formula: -3.05 * x + 3.3 = y  
Predicted y-values: [ 0.25 -11.95 -27.2 -42.45 -57.7 -72.95]  
Prediction error: [-1.25 -0.05 1.2 2.45 -2.3 -0.05]  
Residual error: 1.89076704012
```



# scikit-learn Library

---

scikit-learn (<http://scikit-learn.org/>) is a machine learning library for Python.

Features: classification, regression, clustering, dimensionality reduction

# BeautifulSoup Library

---

BeautifulSoup (<http://www.crummy.com/software/BeautifulSoup/>) is a library to make it easy to search, navigate, and extract data from HTML and XML documents.

# Databases

---

Python can connect to databases to retrieve data. MySQL example:

```
import mysql.connector
try:
    cnx = mysql.connector.connect(user='rlawrenc', password='test',
                                  host='cosc304.ok.ubc.ca', database='WorksOn')
    cursor = cnx.cursor()
    query = ("SELECT eno, ename, salary FROM Emp WHERE title > %s "
             +"and salary < %s")
    cursor.execute(query, ('EE', 50000))
    for (eno, ename, salary) in cursor:
        print(eno, ename, salary)
    cursor.close()
except mysql.connector.Error as err:
    print(err)
finally:
    cnx.close()
```

# Try it: Databases

---

**Question:** Write a program that queries the WorksOn database and returns the employees grouped by title where the employee name is after 'J'. The output should display their title and the average salary for that title. Connection info:

- ```
cnx = mysql.connector.connect(user='rlawrenc',  
password='test', host='cosc304.ok.ubc.ca',  
database='WorksOn')
```

Output:

```
EE 30000.000000  
ME 40000.000000  
PR 20000.000000  
SA 50000.000000
```



# Map-Reduce

---

*Map-Reduce* is a technique for processing large data sets in a functional manner.

- The technique was invented by Google and is implemented in a variety of systems including Python, NoSQL databases, and a Big Data system called Hadoop.
- In Hadoop, map takes as input key-value pairs and outputs key-value pairs. The shuffle step will move pairs to particular machines based on keys. The reduce step takes a list of key-value pairs (with same key) and reduces to one value.
- It is possible to code map/reduce functions in Python for use in Hadoop cluster.

Simpler version of Map-Reduce in Python without a cluster:

- Map function - takes as input a list and a function then applies function to each element of the list to produce a new list as output
- Filter function - only keeps list elements where filter function is `True`
- Reduce function - takes result of map/filter and produces single value from list

# Python Map-Reduce Example

---

```
import functools      # For Reduce
```

```
data = [1, 2, 3, 4, 5, 6]
```

```
# Map function
```

```
def triple(x):  
    return x*3
```

```
# Filter function
```

```
def myfilter(x):  
    if x % 2 == 0:  
        return True  
    return False
```

```
# Reduce function
```

```
def sum(x, y):  
    return x+y
```

# Python Map-Reduce Example (2)

---

```
result = list(map(triple, data))
print("Result after map:",result)

result = list(filter(myfilter, result))
print("Result after filter:",result)

result = functools.reduce(sum, result)
print("Result after reduce:",result)
```

# Try it: Map-Reduce

---

**Question:** Write a map-reduce program that during the map step will subtract 2 from each element. The reduce step should return the product of all the elements in the list.

# Conclusion

---

**Python** has many libraries to help with data analysis tasks:

- reading and write to files
- `csv` module for processing CSV files
- Biopython for bioinformatics
- numerous chart libraries including `matplotlib` and `ggplot`
- SciPy - collection of libraries for scientific computing
- libraries for web access and parsing (BeautifulSoup)
- database access libraries and connectors

The ***try-except statement*** is used to handle exceptions so that the program may continue when an error condition occurs.

# Objectives

---

- Open, read, write, and close text files
- Process CSV files including using the `csv` module
- Define: IPv4/IPv6 address, domain, domain name, URL
- Read URLs using `urllib.request`.
- Define: exception, exception handling
- Use `try-except` statement to handle exceptions and understand how each of `try`, `except`, `else`, `finally` blocks are used
- Import Python modules
- Use Biopython module to retrieve NCBI data and perform BLAST
- Build charts using `matplotlib`
- Perform linear regression and k-means clustering using SciPy
- Connect to and query the MySQL database using Python
- Write simple Map-Reduce programs