TRIUMF

Introduction to Scientific Computing with Python

Dr. Pietro Giampa

TRIUMF
May 2020

Discovery, accelerated

Lecture 1

Outline:

1. Why Do We Need Programming in Physics?
2. Basic Variables and Operations
3. Building Loops
4. Set Conditional Statement
5. Constructing Functions
6. Exercises

TRIUMF

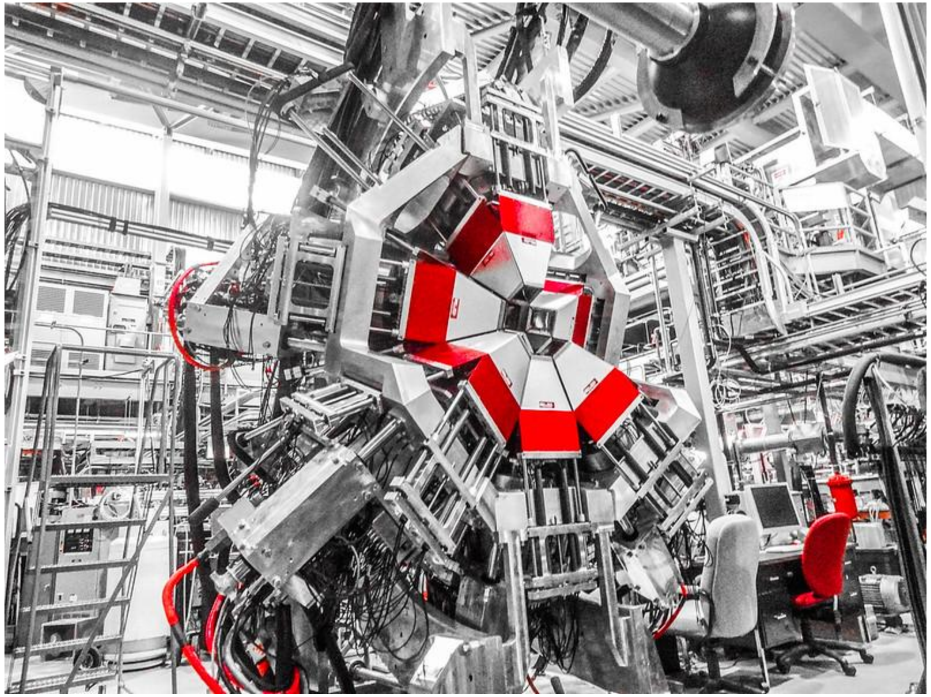
1.1 - Why Do We Need Programming in Physics?

Multiple Tasks:

1. Multi-steps calculations
2. Data-streaming
3. Data-visualization
4. Data-handling
5. Simulations

Applications at TRIUMF:

1. Beam monitoring software
2. Experimental data-analysis
3. Theoretical calculations
4. Optical simulations
5. ….. and more



Discovery, accelerated

TRIUMF

1.2 - Basic Variables and Operations

Available Variables Type:

1. NUMBERS (10, 28.89, 0x100, 3.1j)
2. STRING (‘Hello World’, ‘Type Here’, ‘I’m a string Example’)
3. BOOLEAN (true, false)

Numbers Options:

1. INTEGER - signed integer - [0, 1, 2, 3, 4 …. etc]
2. LONG - long integer, octal or hexadecimal - [51924361L, 0x19323L .... etc]
3. FLOAT - floating point real values - [0.32, 100.2, 54.67899 .... etc]
4. COMPLEX - standard complex numbers - [7.12j, 0.876j, 23j ... etc]

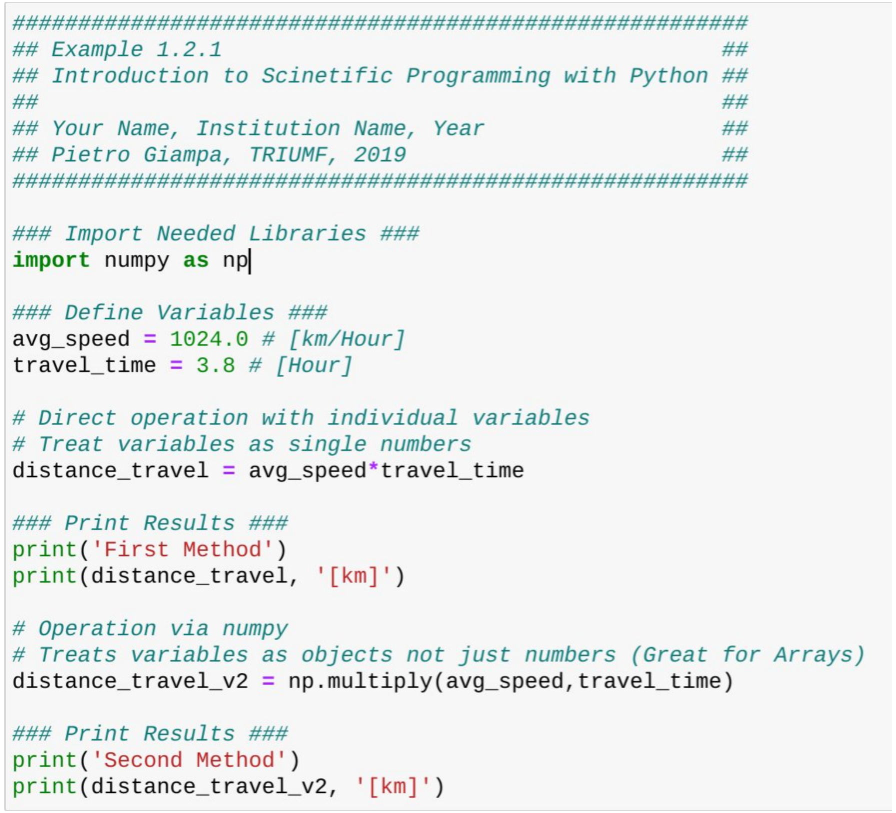
TRIUMF

1.2 - Basic Variables and Operations

Example 1.2.1

You are on a plane from Vancouver to Ottawa and you know the avg cruising speed (1024.0 [km/h]) and the length of the travel (3.8 [h]) and you want to estimate the distance you traveled.

PS: Air Canada removed distance traveled from their interactive Map (very annoying)



```
########################################################
## Example 1.2.1                                      ##
## Introduction to Scinetific Programming with Python ##
##                                                    ##
## Your Name, Institution Name, Year                  ##
## Pietro Giampa, TRIUMF, 2019                        ##
########################################################

### Import Needed Libraries ###
import numpy as np

### Define Variables ###
avg_speed = 1024.0 # [km/Hour]
travel_time = 3.8 # [Hour]

# Direct operation with individual variables
# Treat variables as single numbers
distance_travel = avg_speed*travel_time

### Print Results ###
print('First Method')
print(distance_travel, '[km]')

# Operation via numpy
# Treats variables as objects not just numbers (Great for Arrays)
distance_travel_v2 = np.multiply(avg_speed,travel_time)

### Print Results ###
print('Second Method')
print(distance_travel_v2, '[km]')
```

1.2 - Basic Variables and Operations

Example 1.2.2

You are running an experiment and you have to monitor a given observable, let's say the level of the LN2 dewar you are using for cooling your experiment. You want to write a small program that lets you input a LN2 level in % and convert that number into liters.
(100% = 255 L, assume linearity)

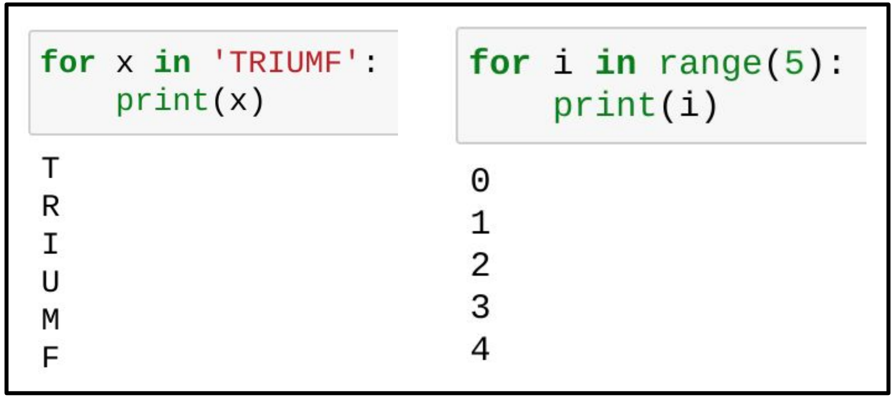
```
########################################################
## Example 1.2.2                                      ##
## Introduction to Scinetific Programming with Python ##
##                                                    ##
## Pietro Giampa, TRIUMF, 2019                        ##
########################################################

### Prompt Keyboard Input ###
Level = input('Enter Current LN2 Level [%]')

### Convert from String to Float
Level = float(Level)

### From percentage to fraction
Level = Level/100.

### Convert % to Liters ####
Liters_Converter = 255.
Level_L = Level*Liters_Converter

### Print Results ###
print('LN2 Level -',Level_L,'[Liters]')
```

1.3 - Building Loops

Loops are a fundamental block in programming. While we are gonna focus on Python, the basic concept of loops is the same for basically all programming languages (C++, Java .... etc).

FOR LOOPS: A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

WHILE LOOP: With the while loop we can execute a set of statements as long as a condition is true.

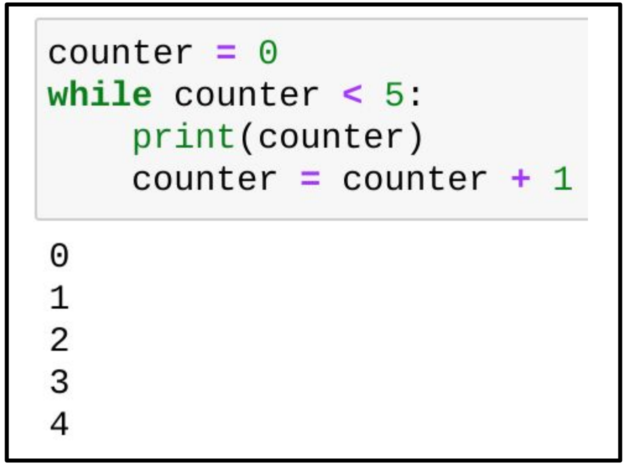


```python
for x in 'TRIUMF':
    print(x)
```

```
T
R
I
U
M
F
```

```python
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```



```python
counter = 0
while counter < 5:
    print(counter)
    counter = counter + 1
```

```
0
1
2
3
4
```

TRIUMF

1.3 - Building Loops

Example 1.3.1

Let's use Poisson statistics as an example. Say you have an experiment with an average event rate of 3.2 events every hour. You want to estimate the probability of getting either 0, 1, 2, 3, 4, and 5 events in the one hour span.

Recall the Poisson distribution:

$$P(k, \lambda) = \frac{\lambda^k \cdot e^{-\lambda}}{k!}$$

Where k is the expected number of events and λ is the average event rate of the experiment.

```
#########################################################
## Example 1.3.1                                       ##
## Introduction to Scinetific Programming with Python ##
##                                                     ##
## Pietro Giampa, TRIUMF, 2019                         ##
#########################################################

### Define Variables ###
lambda_value = 3.2 # [events/hour]

### Initiate FOR loop ###
for k in range(6):
    Probability = (np.power(lambda_value,k)*np.exp(-lambda_value))/(np.math.factorial(k))
    print(k,'-',Probability)
```

Discovery,
accelerated

1.4 - Set Conditional Statements

Conditional statements, alongside loops, are the absolute basics of programming. Similar to loops, we will focus on the Python syntax, however, the basic concepts of conditional statements holds true for most languages (C++, Jave .... etc).

Decision making is required when we want to execute a code only if a certain condition is satisfied.

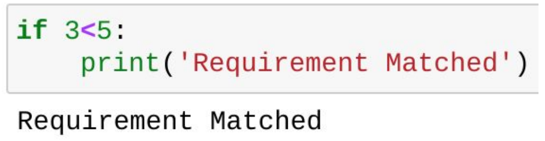
Conditionals:

- < - Less than …
- > - Bigger than ...
- <= - Less or equal to ...
- >= - Bigger or equal to ...
- == - Equal to ...
- != - Different than ...
- and - and option
- or - or option

Discovery, accelerated

1.4 - Set Conditional Statements

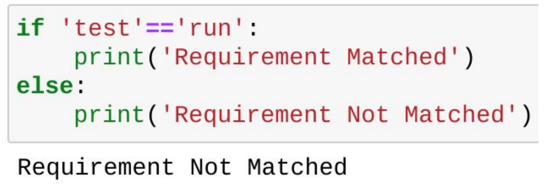
IF STATEMENT: Python makes a decision based on a test expression. if the requirement is passed the subsequent actions are taken, if not the program skips ahead.



```
if 3<5:
    print('Requirement Matched')
```

```
Requirement Matched
```

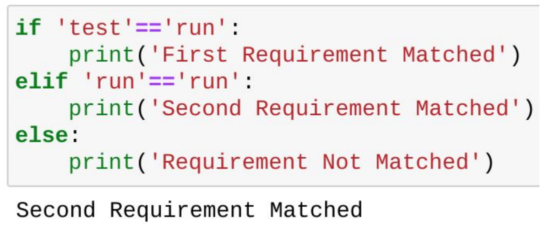
ELSE STATEMENT: The else statement must follows an if statement. The structure works as follows: if the requirement is passed the subsequent actions are taken, if not the program executes the actions from the else statement



```
if 'test'=='run':
    print('Requirement Matched')
else:
    print('Requirement Not Matched')
```

```
Requirement Not Matched
```

ELIF STATEMENT: Sometimes multiple if-statements in consecutive order are necessary (imagine a decision tree). In that case, IF is only used for the first iteration, while elif brings the second conditional.



```
if 'test'=='run':
    print('First Requirement Matched')
elif 'run'=='run':
    print('Second Requirement Matched')
else:
    print('Requirement Not Matched')
```

```
Second Requirement Matched
```

TRIUMF

1.4 - Set Conditional Statements

Example 1.4.1

Generate two random numbers between 0 and 1. Check if both numbers are less than 0.5 multiply them together, if only one of the is less than 0.5 add them together, and if they are both bigger or equal than 0.5 subtract them. Print the result on the screen (include both random numbers).

```python
########################################################
## Example 1.4.1                                      ##
## Introduction to Scinetific Programming with Python ##
##                                                    ##
## Pietro Giampa, TRIUMF, 2019                        ##
########################################################

### Define Variables ###
number1 = np.random.rand()
number2 = np.random.rand()
result = 0.0 # initialize result

### Check Conditionals ###
if number1<0.5 and number2<0.5:
    result = np.multiply(number1,number2)
elif number1<0.5 or number2<0.5:
    result = np.add(number1,number2)
else:
    result = np.subtract(number1,number2)

### Print Results ###
print(result, number1, number2)
```

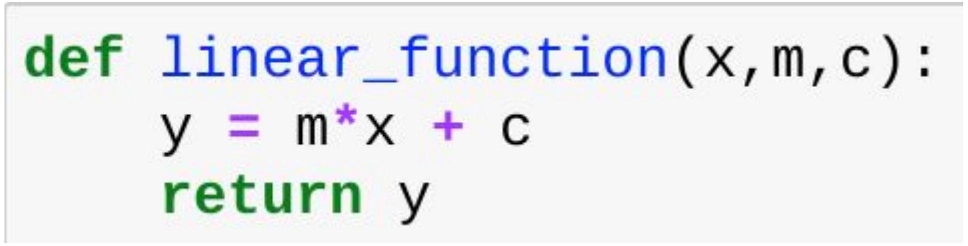
Discovery, accelerated

1.5 - Building Functions

Some actions, calculations or conditionals might be used recurrently in a given program, in those cases functions are the best way to go.

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

functions can generate actions (like print()) or return values (individual or arrays).



```python
def linear_function(x,m,c):
    y = m*x + c
    return y
```

1.5 - Building Functions

Example 1.5.1

Construct a function to model radioactive decays by following the standard radioactive decay law:

$$N(t) = N_0 \cdot e^{-\lambda \cdot t}$$

Where N(t) is the number of remaining daughters at time t, N0 is the starting daughters concentration and λ is the half-life. If you inject 4.6×10^{15} Nuclei of ^{222}Rn in your detector, write a program that determines how many Rn daughters you will measure after 1 day, 4 days and 9 days.

```
########################################################
## Example 1.5.1                                      ##
## Introduction to Scinetific Programming with Python ##
##                                                    ##
## Pietro Giampa, TRIUMF, 2019                        ##
########################################################

### Set Decay Law as Function ###
def DecayLaw(N0, lmbd, t):
    Nt = N0*np.exp(-lmbd*t)
    return Nt

### Define Variables ###
time1 = 1. # [days]
time2 = 4. # [days]
time3 = 9. # [days]
lmbd = 3.8215 # [days]
N0 = 4.6E15 # [nuclei]

### Do Some Math ###
Nt1 = DecayLaw(N0,lmbd,time1)
Nt2 = DecayLaw(N0,lmbd,time2)
Nt3 = DecayLaw(N0,lmbd,time3)

### Print Results ###
print('After',time1,'days:',Nt1)
print('After',time2,'days:',Nt2)
print('After',time3,'days:',Nt3)
```

Lecture 2

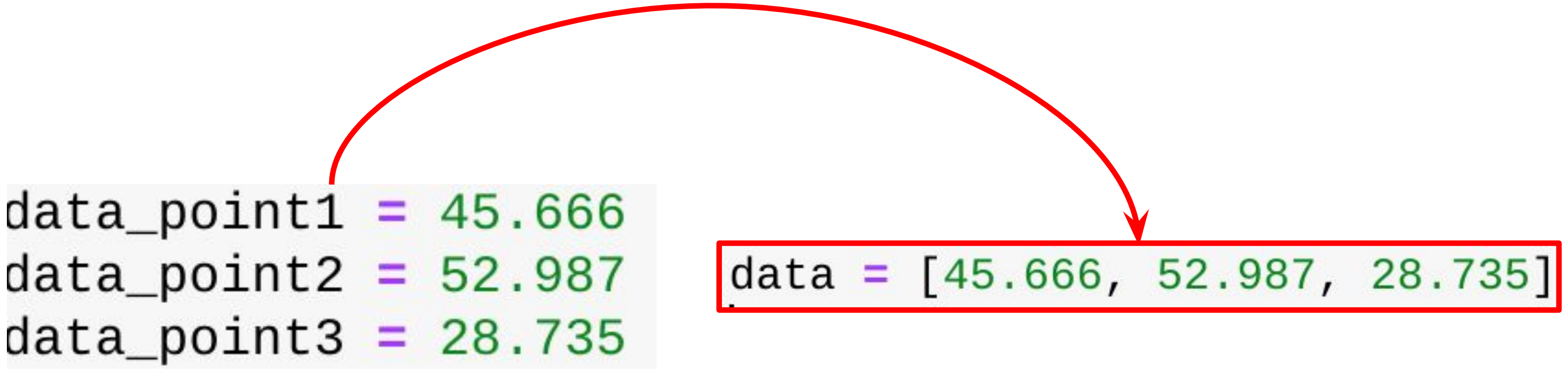
Outline:

1. Introduction To Arrays
2. Arrays Operation
3. Data-Visualization with Matplotlib
4. Exercises

TRIUMF

2.1 - Introduction To Arrays

An ARRAY is a special variable, which can hold more than one value per time.



```
data_point1 = 45.666
data_point2 = 52.987
data_point3 = 28.735
```

LAST LECTURE

```
data = [45.666, 52.987, 28.735]
```

THIS LECTURE

You can access array entries anytime in your code by simple passing the array name and the number of the entry you require:

data[0] = 45.666

Discovery, accelerated

TRIUMF

2.1 - Introduction To Arrays

```
data = [45.666, 52.987, 28.735]
```

Let’s review the ABC of Arrays with Python:

- How do I check the length of a given array? **len(array)**

```
nentries = len(data)
print('Number of Entries in the Array:',nentries)
```

```
Number of Entries in the Array: 3
```

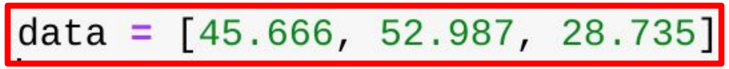
- How do you add entries to an array? **array.append(x)**
  (NB: No restrictions on the entry type)

```
data.append(56.565)
data.append('Entry')
nentries = len(data)
print('Current Data Array:',data)
print('Number of Entries in the Array:',nentries)
```

```
Current Data Array: [45.666, 52.987, 28.735, 56.565, 'Entry']
Number of Entries in the Array: 5
```

TRIUMF

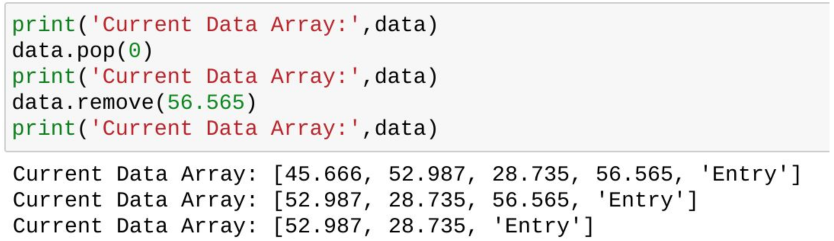
2.1 - Introduction To Arrays



```
data = [45.666, 52.987, 28.735]
```

Let’s review the ABC of Arrays with Python:

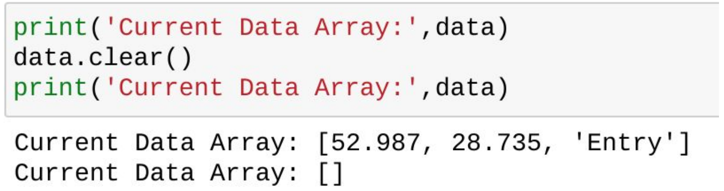
- How do I remove elements from an array? **array.pop(entry #) or array.remove(x)**



```
print('Current Data Array:',data)
data.pop(0)
print('Current Data Array:',data)
data.remove(56.565)
print('Current Data Array:',data)
```

```
Current Data Array: [45.666, 52.987, 28.735, 56.565, 'Entry']
Current Data Array: [52.987, 28.735, 56.565, 'Entry']
Current Data Array: [52.987, 28.735, 'Entry']
```

- How do you I completely remove all entries from an array? **array.clear()**



```
print('Current Data Array:',data)
data.clear()
print('Current Data Array:',data)
```

```
Current Data Array: [52.987, 28.735, 'Entry']
Current Data Array: []
```

TRIUMF

2.1 - Introduction To Arrays

```
data = [45.666, 52.987, 28.735]
```

Let’s review the ABC of Arrays with Python:

- **Arrays can be multidimensional**, no just 1-D, but 2-D, 3-D etc

```
data_3b3 = [['x1','x2','x3'],['y1','y2','y3'],['z1','z2','z3']]
print('Select Element 1x3:',data_3b3[0][2])
```

```
Select Element 1x3: x3
```

Discovery, accelerated

TRIUMF

2.1 - Introduction To Arrays

Example 2.1.1

Use the data below to setup two arrays where the first corresponds to the data timestamp and the second is the actual data. Remove all the data with timestamp prior to 13:00:00, then add a new entry with timestamp = 15:52:00 and data = 56.7. Combine the two one-dimensional arrays in to one multi-dimensional array, and print the first entry in the new array.

TIME	Data
12:40:00	45.3
13:52:00	53.7
14:32:00	100.4
14:53:00	134.8
15:32:00	76.3

```
#########################################################
## Example 2.1.1                                       ##
## Introduction to Scinetific Programming with Python ##
##                                                     ##
## Pietro Giampa, TRIUMF, 2020                         ##
#########################################################

### Define Needed Libraries ###
import numpy as np

### Define Arrays ###
timestamp = ['12:40:00','13:52:00','14:32:00','14:52:00','15:32:00']
data_points = [45.3, 53.7, 100.4, 134.8, 76.3]
entry_to_remove = []

### Find Early Datapoint ###
for k in range(len(timestamp)):
    if timestamp[k]<'13:00:00':
        entry_to_remove.append(k)


## Remove Early Data-point
for i in range(len(entry_to_remove)):
    timestamp.pop(entry_to_remove[i])
    data_points.pop(entry_to_remove[i])

### Add new Data-point ###
timestamp.append('15:52:00')
data_points.append(56.7)

### Combine Arrays into nD Array ###
data = [timestamp,data_points]
print('New n-dim Array:',data)
```

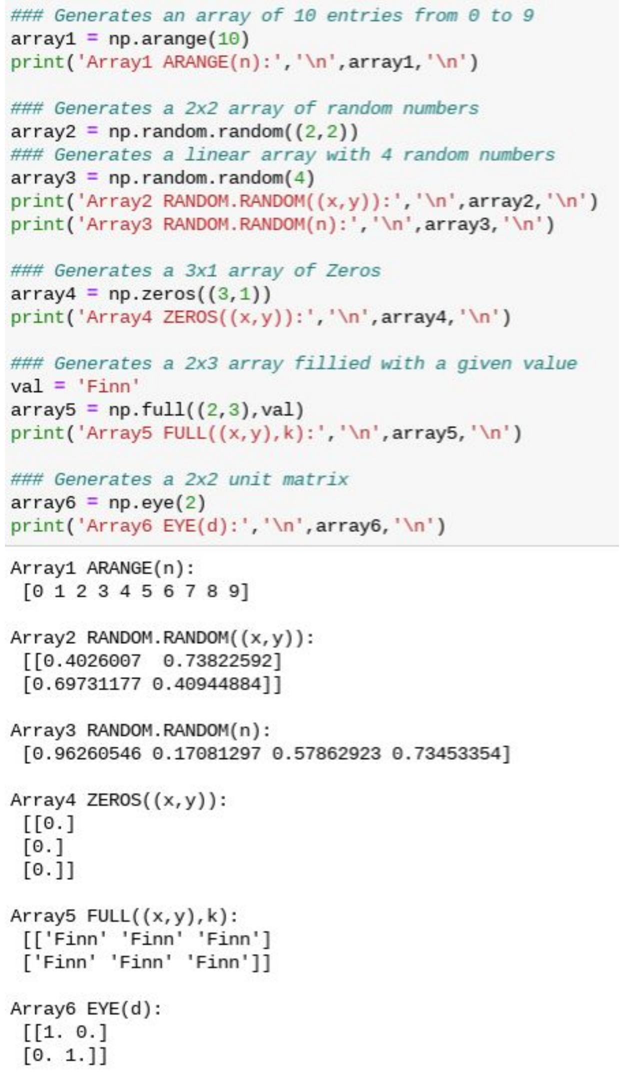
Discovery, accelerated

TRIUMF

2.2 - Arrays Operations

This is really the main reason for pushing the **numpy** library in the last lecture.

- **ARANGE(n)** = creates an array integers (or floats) of n entries, starting from 0 and increasing to n-1.
- **RANDOM.RANDOM((x,y) or n)** = creates an array of size X*Y (or a 1D array of size n) filled with random numbers generated from 0 to 1.
- **ZEROS((x,y) or n)** = Similar to the random option, generates a one or multi-dimensions array filled with zeros.
- **FULL((x,y),k)** = Generates an array of size X*Y filled with values equal to the input k.
- **EYE(d)** = Creates a unit matrix of dimension d.



```
### Generates an array of 10 entries from 0 to 9
array1 = np.arange(10)
print('Array1 ARANGE(n):','\n',array1,'\n')

### Generates a 2x2 array of random numbers
array2 = np.random.random((2,2))
### Generates a linear array with 4 random numbers
array3 = np.random.random(4)
print('Array2 RANDOM.RANDOM((x,y)):','\n',array2,'\n')
print('Array3 RANDOM.RANDOM(n):','\n',array3,'\n')

### Generates a 3x1 array of Zeros
array4 = np.zeros((3,1))
print('Array4 ZEROS((x,y)):','\n',array4,'\n')

### Generates a 2x3 array fillied with a given value
val = 'Finn'
array5 = np.full((2,3),val)
print('Array5 FULL((x,y),k):','\n',array5,'\n')

### Generates a 2x2 unit matrix
array6 = np.eye(2)
print('Array6 EYE(d):','\n',array6,'\n')
```

```
Array1 ARANGE(n):
 [0 1 2 3 4 5 6 7 8 9]

Array2 RANDOM.RANDOM((x,y)):
 [[0.4026007  0.73822592]
 [0.69731177 0.40944884]]

Array3 RANDOM.RANDOM(n):
 [0.96260546 0.17081297 0.57862923 0.73453354]

Array4 ZEROS((x,y)):
 [[0.]
 [0.]
 [0.]]

Array5 FULL((x,y),k):
 [['Finn' 'Finn' 'Finn']
 ['Finn' 'Finn' 'Finn']]

Array6 EYE(d):
 [[1. 0.]
 [0. 1.]]
```

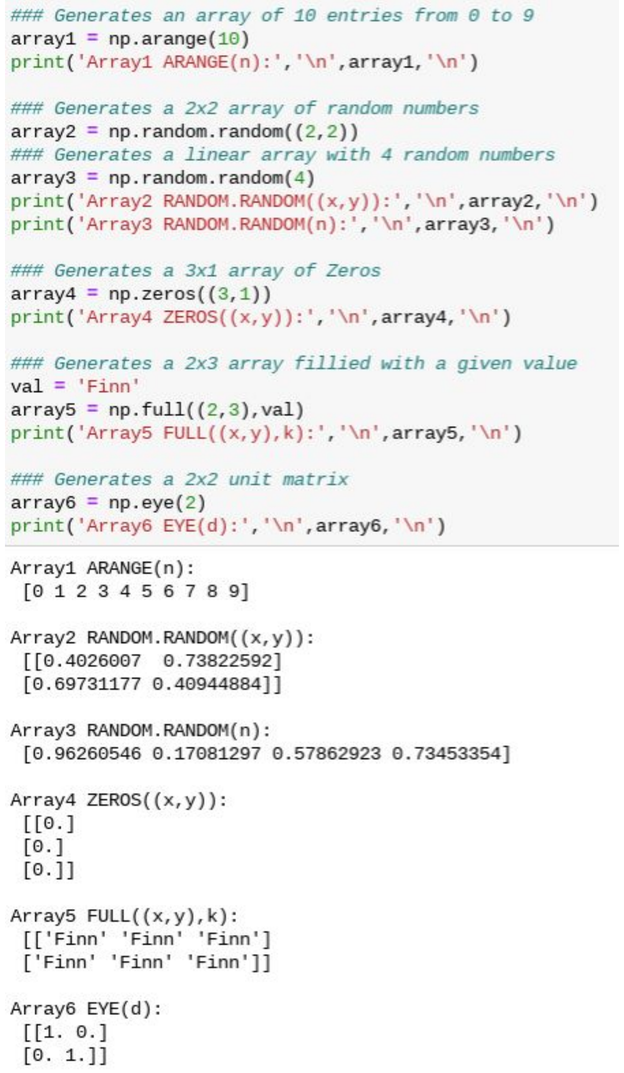
Discovery,
accelerated

TRIUMF

2.2 - Arrays Operations

Standard matrix algebra functions are also available via **numpy**.

- **numpy.dot(x,y):** Dot product between arrays (matrices) X and Y.
- **numpy.cross(x,y):** Cross product between (matrices) arrays X and Y.
- **x.T:** Transposition of array (matrix) X.
- **numpy.linalg.det(x):** Determinant of array (matrix) X.
- **numpy.linalg.norm(x):** Magnitude of array (matrix) X.
- **numpy.var(x):** Variance of array (matrix).



```
### Generates an array of 10 entries from 0 to 9
array1 = np.arange(10)
print('Array1 ARANGE(n):','\n',array1,'\n')

### Generates a 2x2 array of random numbers
array2 = np.random.random((2,2))
### Generates a linear array with 4 random numbers
array3 = np.random.random(4)
print('Array2 RANDOM.RANDOM((x,y)):','\n',array2,'\n')
print('Array3 RANDOM.RANDOM(n):','\n',array3,'\n')

### Generates a 3x1 array of Zeros
array4 = np.zeros((3,1))
print('Array4 ZEROS((x,y)):','\n',array4,'\n')

### Generates a 2x3 array fillied with a given value
val = 'Finn'
array5 = np.full((2,3),val)
print('Array5 FULL((x,y),k):','\n',array5,'\n')

### Generates a 2x2 unit matrix
array6 = np.eye(2)
print('Array6 EYE(d):','\n',array6,'\n')
```

```
Array1 ARANGE(n):
 [0 1 2 3 4 5 6 7 8 9]

Array2 RANDOM.RANDOM((x,y)):
 [[0.4026007  0.73822592]
 [0.69731177 0.40944884]]

Array3 RANDOM.RANDOM(n):
 [0.96260546 0.17081297 0.57862923 0.73453354]

Array4 ZEROS((x,y)):
 [[0.]
 [0.]
 [0.]]

Array5 FULL((x,y),k):
 [['Finn' 'Finn' 'Finn']
 ['Finn' 'Finn' 'Finn']]

Array6 EYE(d):
 [[1. 0.]
 [0. 1.]]
```

Discovery,
accelerated

TRIUMF

2.2 - Arrays Operations

Example 2.2.1

If you have a particle of mass m=0.02 [kg], with a given position vector $r=(2\mathbf{i}-\mathbf{j}-3\mathbf{k})$ [m] and a velocity matrix of $v=(3\mathbf{i}+5\mathbf{j}-4\mathbf{k})$ [m]. Estimate the particle's angular momentum about the origin and calculate the magnitude.

Recall that the angular momentum $\mathbf{L}$ of a particle of mass $\mathbf{m}$ is given by:

$$\mathbf{L} = \mathbf{m}\mathbf{r}\times\mathbf{v}$$

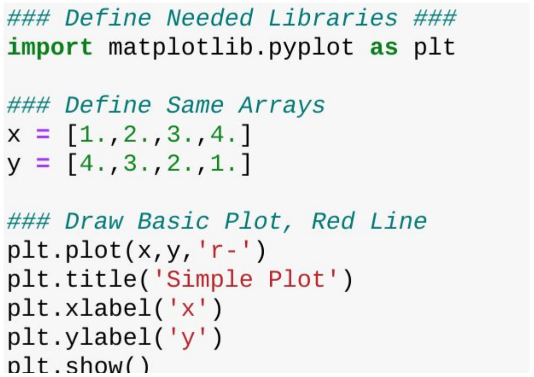
```
##########################################################
## Example 2.2.1                                        ##
## Introduction to Scinetific Programming with Python ##
##                                                      ##
## Pietro Giampa, TRIUMF, 2020                          ##
##########################################################

### Define Needed Libraries ###
import numpy as np

### Define Variables
m = 0.02 # [kg] partical mass

### Define Data Arrays
r_array = [2., -1., -3.] # [m] position array
v_array = [3., 5., -4.] # [m/s] velocity array

### Create the multi-D Angular Momentum Array
L_array = np.cross(r_array,v_array)
L_array = np.multiply(L_array,m)
L_det = np.linalg.norm(L_array)

### Print the Results
print('Angular Momentum L:',L_det,'[kg*m2/s]')
```

TRIUMF

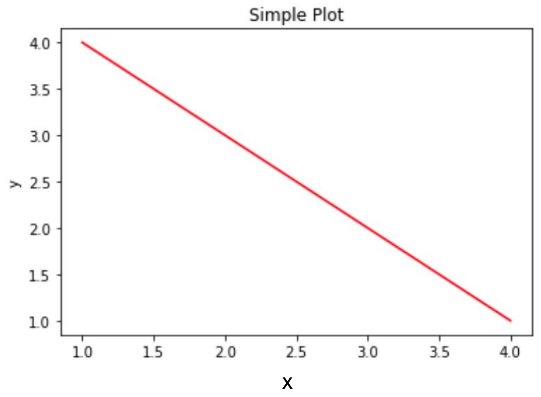
2.3 - Data-Visualization with Matplotlib

Learning how to properly visualize your data and results is a crucial component of data-analysis. If a plot is well constructed, whoever see it can extrapolate all the conclusions he/she/they need.

Matplotlib is the best available option for data-visualization in Python (not even close). The latest documentation versions can be found at: https://matplotlib.org/3.1.1/gallery/index.html (lots and lots of tutorials and examples).



```
### Define Needed Libraries ###
import matplotlib.pyplot as plt

### Define Same Arrays
x = [1.,2.,3.,4.]
y = [4.,3.,2.,1.]

### Draw Basic Plot, Red Line
plt.plot(x,y,'r-')
plt.title('Simple Plot')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

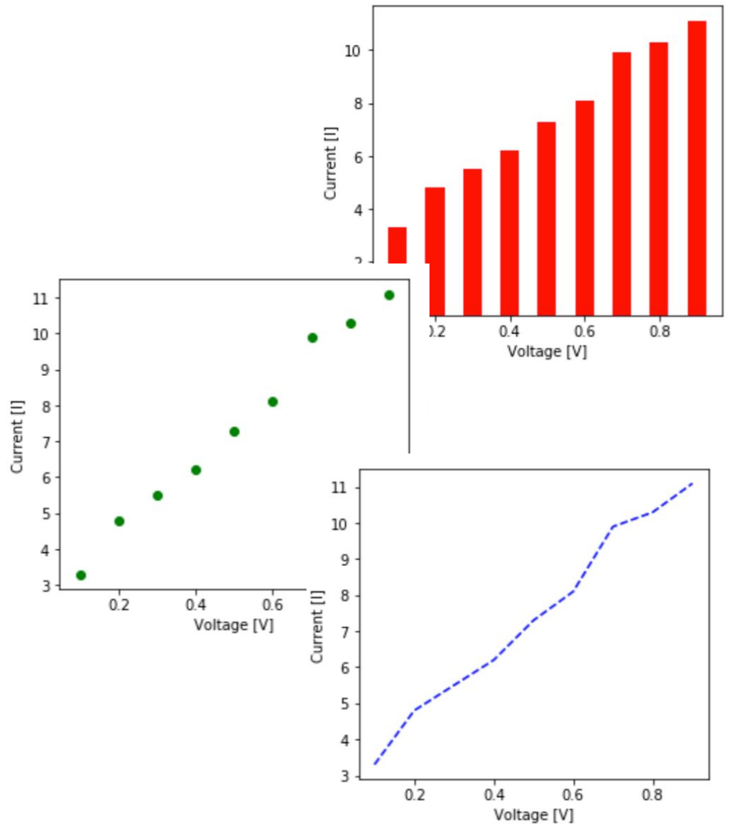


TRIUMF

2.3 - Data-Visualization with Matplotlib

Matplotlib is a huge library with many many options, here are just the basics:

- **plt.plot(x,y) -** Creates a standard 1D histogram, given arrays x and y.
- **plt.hist(x, nbins) -** Creates an histogram based on array x, given number of bins equal to nbins.
- **plt.bar(x,y,w,align='center') -** Creates a Bar-chart given x and y, with bars of width w. Bars center will be place at value x.
- **plt.scatter(x,y) -** Creates a scatter plot with Markers, given arrays x and y.
- **plt.errorbar(x,y,xerr,yerr) -** Creates a standard plot where each point is assigned an error given by the arrays xerr and yerr.
- **plt.hist2d(x,y,nbins) -** Creates a 2D histogram based on arrays x and y, and given a bin number of nbins.

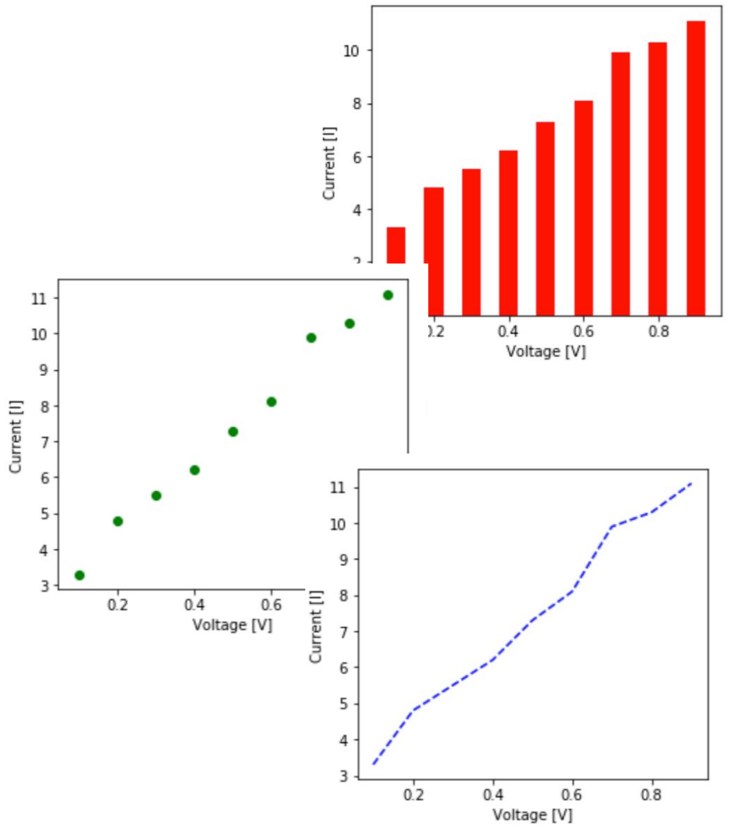


Discovery, accelerated

2.3 - Data-Visualization with Matplotlib

Useful Tips

- Always include a Title
- Always label your axis, including units where suitable
- Data should occupy a minimum of 2/3 of your canvas
- Add texts or highlights only if it enhance the results
- Less is more



TRIUMF

2.3 - Data-Visualization with Matplotlib

Example 2.3.1

You are in the lab and you are measuring the current of a circuit, as a function of the applied voltage. Your results are given below:

Voltage = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9] Current = [3.3, 4.8, 5.5, 6.2, 7.3, 8.1, 9.9, 10.3, 11.1]

Plot the results above as a standard plot, a bar chart, and a scatter plot. Include all three of them on the same Canvas.

```
########################################################
## Example 2.3.1                                      ##
## Introduction to Scinetific Programming with Python ##
##                                                    ##
## Pietro Giampa, TRIUMF, 2020                        ##
########################################################

### Define Needed Libraries ###
import numpy as np
import matplotlib.pyplot as plt

### Define X array
Voltage = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9] # [Volts]
Current = [3.3, 4.8, 5.5, 6.2, 7.3, 8.1, 9.9, 10.3, 11.1] # [mAmp]

### Split the Canvas in 3
fig, axs = plt.subplots(1, 3, figsize=(15,4)) #Note the size can be adjusted
fig.suptitle('Voltage vs Current, Example 2.3.1') #Set Title

### Plot Bar Chart, V vs I, width=0.05 with Center aligment, Red Color
axs[0].bar(Voltage,Current,width=0.05,align='center',facecolor='red')
axs[0].set_xlabel('Voltage [V]')
axs[0].set_ylabel('Current [I]')

### Scatter plot, V vs I, Markers set to green cricles
axs[1].scatter(Voltage,Current,c='g',marker='o')
axs[1].set_xlabel('Voltage [V]')
axs[1].set_ylabel('Current [I]')

### Basic Plot, V vs I, Blue dotted line
axs[2].plot(Voltage,Current,'b--')
axs[2].set_xlabel('Voltage [V]')
axs[2].set_ylabel('Current [I]')

### Plot the results
plt.show()
```

Discovery, accelerated

Lecture 3

Outline:

1. Handling Large Datasets With Pandas
2. Data Fitting With SciPy
3. Interpolation With SciPy
4. Exercises

TRIUMF

3.1 - Handling Large Datasets With Pandas

Pandas is an incredible library capable of doing almost everything needed for large dataset analysis. For documentation or instructions on how to install the library go to their website: https://pandas.pydata.org/ In general, there are two types of Pandas objects:

- **Series:** A one-dimensional data structure that can store values, and for every value it holds a unique index, too.
- **Data Frames:** A two (or more) dimensional data structure. Effectively a table with rows and columns. The columns have names and the rows have indexes.

as always, you need to remember to add the library into your code, like this:

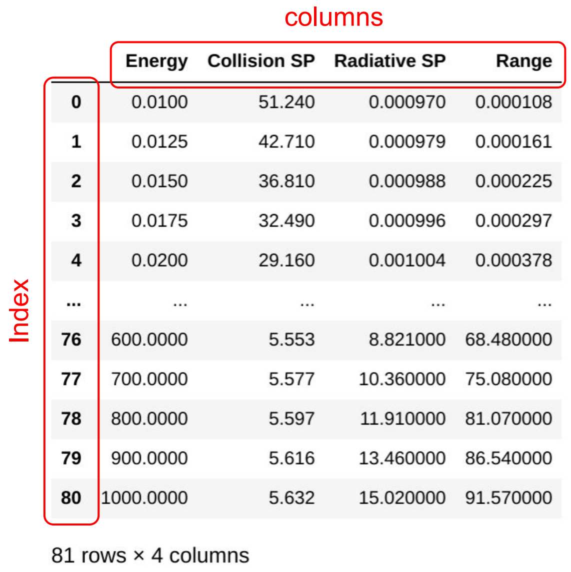
```
import pandas as pd
```

Discovery, accelerated

TRIUMF

3.1 - Handling Large Datasets With Pandas

Pandas can take almost any data files or data list as input and convert it into a DataFrame that is easy to use (whether you are working with text files, CSV files, SQL files, and others). To import data you only need a simple line of code using a function called read_csv().


columns
Index

	Energy	Collision SP	Radiative SP	Range
0	0.0100	51.240	0.000970	0.000108
1	0.0125	42.710	0.000979	0.000161
2	0.0150	36.810	0.000988	0.000225
3	0.0175	32.490	0.000996	0.000297
4	0.0200	29.160	0.001004	0.000378
...	...	...	...	...
76	600.0000	5.553	8.821000	68.480000
77	700.0000	5.577	10.360000	75.080000
78	800.0000	5.597	11.910000	81.070000
79	900.0000	5.616	13.460000	86.540000
80	1000.0000	5.632	15.020000	91.570000

81 rows × 4 columns

Discovery, accelerated

3.1 - Handling Large Datasets With Pandas

Load data from file:

```
file = 'Lecture3_DataFile.txt' ## Path + Name of the Data File
names = ['Energy','Collision SP','Radiative SP','Range'] ## Array containing the columns names
DataFrame = pd.read_csv(file, header=None, sep='\t', names=names) ## Import file to DataFrame
```

print(DataFrame)

```
       Energy  Collision SP  Radiative SP      Range
0      0.0100        51.240      0.000970   0.000108
1      0.0125        42.710      0.000979   0.000161
2      0.0150        36.810      0.000988   0.000225
3      0.0175        32.490      0.000996   0.000297
4      0.0200        29.160      0.001004   0.000378
..        ...           ...           ...        ...
76   600.0000         5.553      8.821000  68.480000
77   700.0000         5.577     10.360000  75.080000
78   800.0000         5.597     11.910000  81.070000
79   900.0000         5.616     13.460000  86.540000
80  1000.0000         5.632     15.020000  91.570000

[81 rows x 4 columns]
```

DataFrame

	Energy	Collision SP	Radiative SP	Range
0	0.0100	51.240	0.000970	0.000108
1	0.0125	42.710	0.000979	0.000161
2	0.0150	36.810	0.000988	0.000225
3	0.0175	32.490	0.000996	0.000297
4	0.0200	29.160	0.001004	0.000378
...	...	...	...	...
76	600.0000	5.553	8.821000	68.480000
77	700.0000	5.577	10.360000	75.080000
78	800.0000	5.597	11.910000	81.070000
79	900.0000	5.616	13.460000	86.540000
80	1000.0000	5.632	15.020000	91.570000

81 rows × 4 columns

TRIUMF

3.1 - Handling Large Datasets With Pandas

```
DataFrame.head() ## Print first 5 elements
```

	Energy	Collision SP	Radiative SP	Range
0	0.0100	51.24	0.000970	0.000108
1	0.0125	42.71	0.000979	0.000161
2	0.0150	36.81	0.000988	0.000225
3	0.0175	32.49	0.000996	0.000297
4	0.0200	29.16	0.001004	0.000378

```
DataFrame.tail() ## Print the last 5 elements
```

	Energy	Collision SP	Radiative SP	Range
76	600.0	5.553	8.821	68.48
77	700.0	5.577	10.360	75.08
78	800.0	5.597	11.910	81.07
79	900.0	5.616	13.460	86.54
80	1000.0	5.632	15.020	91.57

```
DataFrame.sample(5) ## Print 5 lements, randomly selected
```

	Energy	Collision SP	Radiative SP	Range
36	2.00	3.823	0.01162	0.474400
15	0.09	9.367	0.00119	0.005544
69	250.00	5.417	3.49600	37.850000
58	50.00	5.090	0.59590	10.150000
63	90.00	5.238	1.15300	16.770000

Access any DataFrame point using column name and index

```
print('Entry 74 in Energy:',DataFrame['Energy'][73]) ## Print the 74th element in the DataFrame
```

```
Entry 74 in Energy: 450.0
```

Get Data Frame Subset

```
NewDataFrame = DataFrame[['Range','Energy']] ## New df based on a subset of DataFrame
NewDataFrame.tail() ## Print last 5 elements
```

	Range	Energy
76	68.48	600.0
77	75.08	700.0
78	81.07	800.0
79	86.54	900.0
80	91.57	1000.0

TRIUMF

3.1 - Handling Large Datasets With Pandas

Create Data Frames from Subset Selected with Conditionals:

```
LowEnergyDF = DataFrame[DataFrame.Energy < 5.0] ## Vreate New DataFrame filled with entries of Energy < 5.0 MeV
LowEnergyDF.tail() ## Print the last 5 elements
```

	Energy	Collision SP	Radiative SP	Range
37	2.5	3.873	0.01534	0.6039
38	3.0	3.924	0.01931	0.7316
39	3.5	3.973	0.02348	0.8576
40	4.0	4.020	0.02782	0.9819
41	4.5	4.063	0.03230	1.1050

Discovery,
accelerated

3.1 - Handling Large Datasets With Pandas

Create New Column From Basic Operation Between Other Columns

```
DataFrame['RangeTimesColl'] = DataFrame['Range']*DataFrame['Collision SP'] ## Create new column
DataFrame.head() ## Print the top of the Data Frame
```

	Energy	Collision SP	Radiative SP	Range	RangeTimesColl
0	0.0100	51.24	0.000970	0.000108	0.005513
1	0.0125	42.71	0.000979	0.000161	0.006889
2	0.0150	36.81	0.000988	0.000225	0.008264
3	0.0175	32.49	0.000996	0.000297	0.009650
4	0.0200	29.16	0.001004	0.000378	0.011031

TRIUMF

3.1 - Handling Large Datasets With Pandas

Pandas have a series of very simple one-line-commands that will let do basic operation on an individual or multiple columns at the same time. Here the command list:

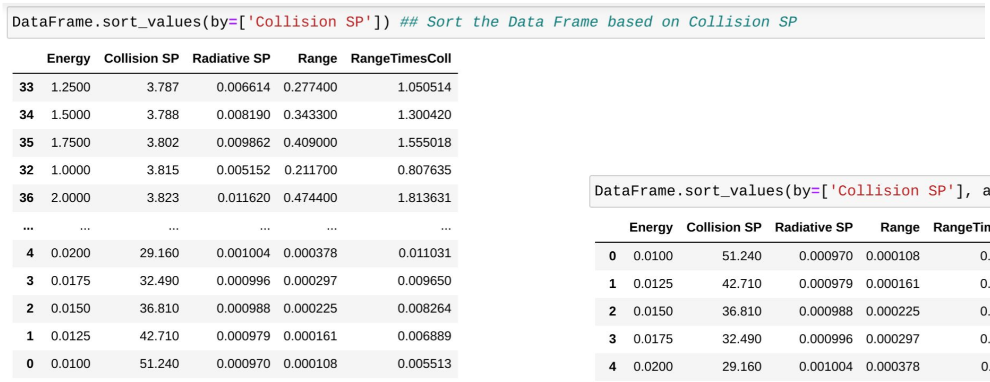
- **count()** - Returns the number of rows in each columns
- **sum()** - Returns the sum of all entries in a given column(s). (NB: Thinks get funky if you use this for non-numbers).
- **min()** - Returns the smallest value in the selected column(s).
- **max()** - Returns the maximum value in a given column(s).
- **mean()** - Returns the mean of the selected column(s).

Discovery, accelerated

TRIUMF

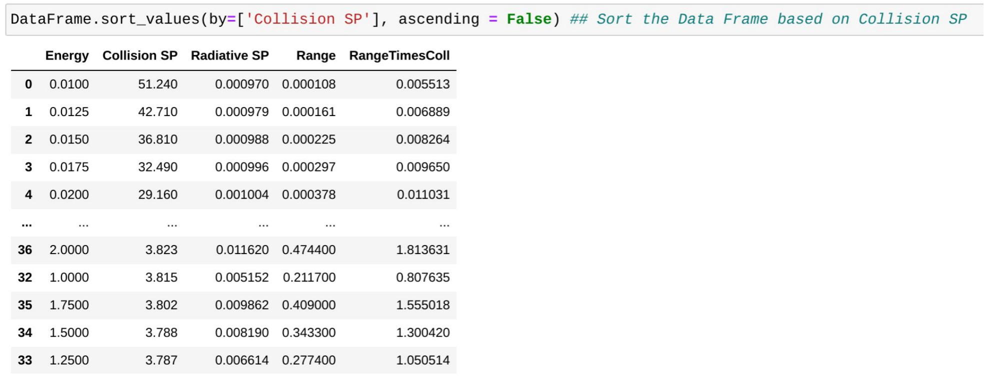
3.1 - Handling Large Datasets With Pandas

Data Frames Can Always Be Sorted By Any Column



```
DataFrame.sort_values(by=['Collision SP']) ## Sort the Data Frame based on Collision SP
```

	Energy	Collision SP	Radiative SP	Range	RangeTimesColl
33	1.2500	3.787	0.006614	0.277400	1.050514
34	1.5000	3.788	0.008190	0.343300	1.300420
35	1.7500	3.802	0.009862	0.409000	1.555018
32	1.0000	3.815	0.005152	0.211700	0.807635
36	2.0000	3.823	0.011620	0.474400	1.813631
...	...	...	...	...	...
4	0.0200	29.160	0.001004	0.000378	0.011031
3	0.0175	32.490	0.000996	0.000297	0.009650
2	0.0150	36.810	0.000988	0.000225	0.008264
1	0.0125	42.710	0.000979	0.000161	0.006889
0	0.0100	51.240	0.000970	0.000108	0.005513



```
DataFrame.sort_values(by=['Collision SP'], ascending = False) ## Sort the Data Frame based on Collision SP
```

	Energy	Collision SP	Radiative SP	Range	RangeTimesColl
0	0.0100	51.240	0.000970	0.000108	0.005513
1	0.0125	42.710	0.000979	0.000161	0.006889
2	0.0150	36.810	0.000988	0.000225	0.008264
3	0.0175	32.490	0.000996	0.000297	0.009650
4	0.0200	29.160	0.001004	0.000378	0.011031
...	...	...	...	...	...
36	2.0000	3.823	0.011620	0.474400	1.813631
32	1.0000	3.815	0.005152	0.211700	0.807635
35	1.7500	3.802	0.009862	0.409000	1.555018
34	1.5000	3.788	0.008190	0.343300	1.300420
33	1.2500	3.787	0.006614	0.277400	1.050514

```
DataFrame.sort_values(by=['Collision SP']).reset_index() ## Reset Index Based on Last Ordering
```

Discovery,
accelerated

TRIUMF

3.1 - Handling Large Datasets With Pandas

Example 3.1.1

You are trying to measure the envelope of a certain beam at TRIUMF, and you just completed a measurement using two different detectors (DET1 and DET2). The data from your measurements are saved in Example311.txt (tab edited). The file contains 4 columns: DetectorType, X, Y, Xerr, Yerr (all data in units of [mm]). Load the data into a Pandas data frame. Then do the following:

- Print the first 5 elements, the last 5 elements and 5 random elements on the screen.
- Create a new Data Frame selecting only entries for DET1.
- Find the min, max, and mean in X and Y.
- Add the errors together, and create a new column with the result.
- Sort based on the column generated in the last bullet point, and change the index to match the new ordering.

```python
########################################################
## Example 3.1.1                                      ##
## Introduction to Scinetific Programming with Python ##
##                                                    ##
## Pietro Giampa, TRIUMF, 2020                        ##
########################################################

### Define Needed Libraries ###
import pandas as pd

### Import Data from File ###
Filename = 'Example311.txt'
names = ['DetectorType','X','Y','Xerr','Yerr']
DF = pd.read_csv(Filename,header=None,sep='\t',names=names) ## Import Data

### Print the first 5 elements, the last 5 elements and 5 random elements ###
DF.head() ## First 5 elements
DF.tail() ## Last 5 elements
DF.sample(5) ## 5 random elements

### Create a new Data Frame selecting only entries for DET1 ###
DF_DET1 = DF[DF.DetectorType == 'DET1']
DF_DET1 = DF_DET1.reset_index() ## Reset the index due to slicing

### Find the min, max, and mean in X and Y ###
X_min = DF_DET1['X'].min()
X_max = DF_DET1['X'].max()
X_mean = DF_DET1['X'].mean()
Y_min = DF_DET1['Y'].min()
Y_max = DF_DET1['Y'].max()
Y_mean = DF_DET1['Y'].mean()
print('X min:',X_min,'X max:',X_max,'X mean:',X_mean)
print('Y min:',Y_min,'Y max:',Y_max,'Y mean:',Y_mean)

### Add the errors in quadrature, and create a new column with the resul ###
DF_DET1['Error Tot'] = DF_DET1['Xerr']+DF_DET1['Yerr']

### Sort based on the column generated in the last bullet point, and change the index to match
DF_DET1.sort_values(by=['Error Tot']).reset_index().head()
```

TRIUMF

3.2 - Data Fitting With SciPy

Data fitting is one of the most useful toolsets for data analysis, it's the ultimate test between data and prediction. In Python, there are different options for data fitting. However, in this course, we will only review a method from the SciPy library called curve_fit. This is the most versatile fitting tool and adequate for all scientific levels.

Cureve_fit requires a fitting function, a dataset input and a series of optional parameters. Here the most common and useful input parameters:

- ydata - can be used to weight the fed dataset.
- p0 - Prior for floating parameters, this must be input as an array.
- bounds - Set limits on parameters floats.
- method - Allows you to change the minimization method.

Moreover, the cureve_fit function returns two arrays:

- popt - Optimal values for the parameters so that the sum of the squared residuals off(xdata, *popt) - ydata is minimized
- pcov - The estimated covariance of popt. The diagonals provide the variance of the parameter estimate. To compute one standard deviation errors on the parameters use perr = np.sqrt(np.diag(pcov)). How the sigma parameter affects the estimated covariance depends on absolute_sigma argument, as described above.

```
from scipy.optimize import curve_fit
```

3.2 - Data Fitting With SciPy

Example 3.2.1

Create a Gaussian function that receives as input parameters, A, μ, σ. Your code should then generate 100 random numbers distributed following a Gaussian distribution with A=100.0, μ=55.0, σ=32.5. Finally, fit the generated data with a gaussian fit. Print the extracted parameters on the screen next to the input parameters. Lastly. plot the data and the fit on a Canvas (include all labels).

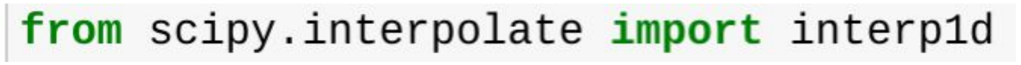
```python
########################################################
## Example 3.2.1                                      ##
## Introduction to Scinetific Programming with Python ##
##                                                    ##
## Pietro Giampa, TRIUMF, 2020                        ##
########################################################

### Define Needed Libraries ###
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

### Define Gaussian Function ###
def GaussFunc(x, A, mu, sig):
    Gauss = A * np.exp(-(x-mu)**2/sig)
    return Gauss

### Define Variabes ###
A = 100.0
mu = 55.0
sig = 32.5

### Generate Random Gaussian Numbers ###
xdata = np.arange(100)
ydata = GaussFunc(np.arange(100),A,mu,sig)

### Fit Gaussian Function ###
popt, pcov = curve_fit(GaussFunc, xdata, ydata, p0=[99.,50.,12.])

### Print Results on Screen ###
print('A Prior:',A,' | A Posterior:', popt[0])
print('mu Prior:',mu,' | mu Posterior:', popt[1])
print('sig Prior:',sig,' | sig Posterior:', popt[2])

### Plot Result Fitting ###
plt.plot(xdata, ydata,'o',label='Data')
plt.plot(xdata, GaussFunc(xdata, *popt),'r-',label='Fit')
plt.title('Exercise 3.2.1')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

3.3 - Interpolator With SciPy

The interpolator is an essential data analysis tool. Interpolation is a convenient method to create a function based on fixed data points, which can be evaluated anywhere within the domain defined by the given data using linear interpolation. An instance of this class is created by passing the 1-D vectors comprising the data.

SciPy has a great option for interpolator, more precisely the interp1d function. This enables the user to perform interpolation on any combination of two arrays (or data). Note that SciPy also offers multi-dimensional interpolators, but those are beyond the scope of this course.

interp1d requires two input arrays (x and y data), with an option for some extra parameters. For example, you can use the parameter 'kind' to set the interpolator in liner, cubic or nearest mode.



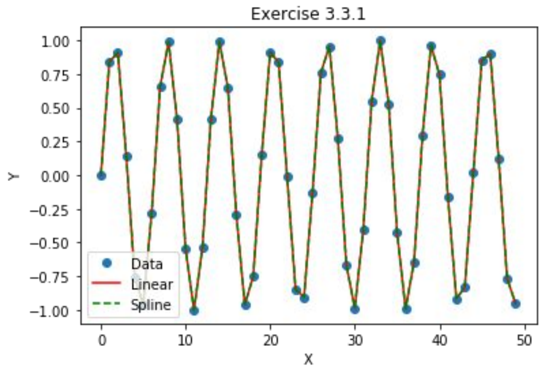
```
from scipy.interpolate import interp1d
```

TRIUMF

3.3 - Interpolator With SciPy

Example 3.3.1

Generate 50 random data points, distributed following a sin function. Interpolate the generated data using both the liner and spline method and plot the results to show any differences (include all labels).



```
###########################################################
## Example 3.3.1                                         ##
## Introduction to Scinetific Programming with Python    ##
##                                                       ##
## Pietro Giampa, TRIUMF, 2020                           ##
###########################################################

### Define Needed Libraries ###
import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

### Define Gaussian Function ###
def SinFunc(x):
    return np.sin(x)

### Generate data using SinFunc ###
xdata = np.arange(50)
ydata = SinFunc(np.arange(50))

### Interpolate the Data ###
Data_Itrp_lin = interp1d(xdata, ydata)
Data_Itrp_spl = interp1d(xdata, ydata, kind='cubic')

### Plot Results ###
plt.plot(xdata,ydata,'o',label='Data')
plt.plot(xdata,Data_Itrp_lin(xdata),'r-',label='Linear')
plt.plot(xdata,Data_Itrp_spl(xdata),'g--',label='Spline')
plt.title('Exercise 3.3.1')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

Discovery, accelerated