Python programming tutorial

SUITE 2026:
Summer Undergraduate Information and Technology Experience

Thursday May 7th 2026

Jean-Marie Coquillat
(Using many previous slides from
Minya Bai and Hannah Fronenberg)

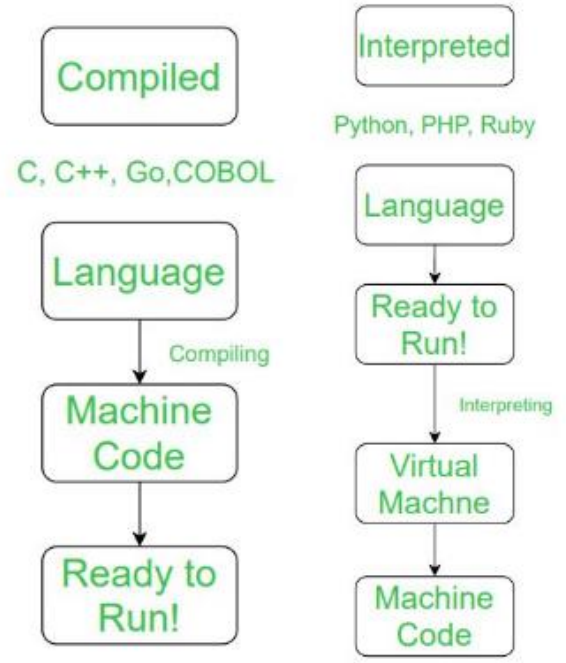
Overview

- What is Python?
- The Basics
  - Syntax and Variables
  - Lists, Dictionaries
  - Functions and Classes
- Libraries and Packages
  - Numpy
  - Matplotlib
  - Scipy
- Practice exercises
- Fight snakes (if time allows)

What is Python?

Python is a **programming language**

- Developed by Benevolent Dictator For Life Guido van Rossum in 1989 during his Christmas vacations.
- Named after Monty Python, the latest version is Python 3.14.4, update from a month ago
- It is an interpreted language meaning it is not directly compiled into machine instructions.
- It is an object-oriented language.



Credit: Hannah Fronenberg

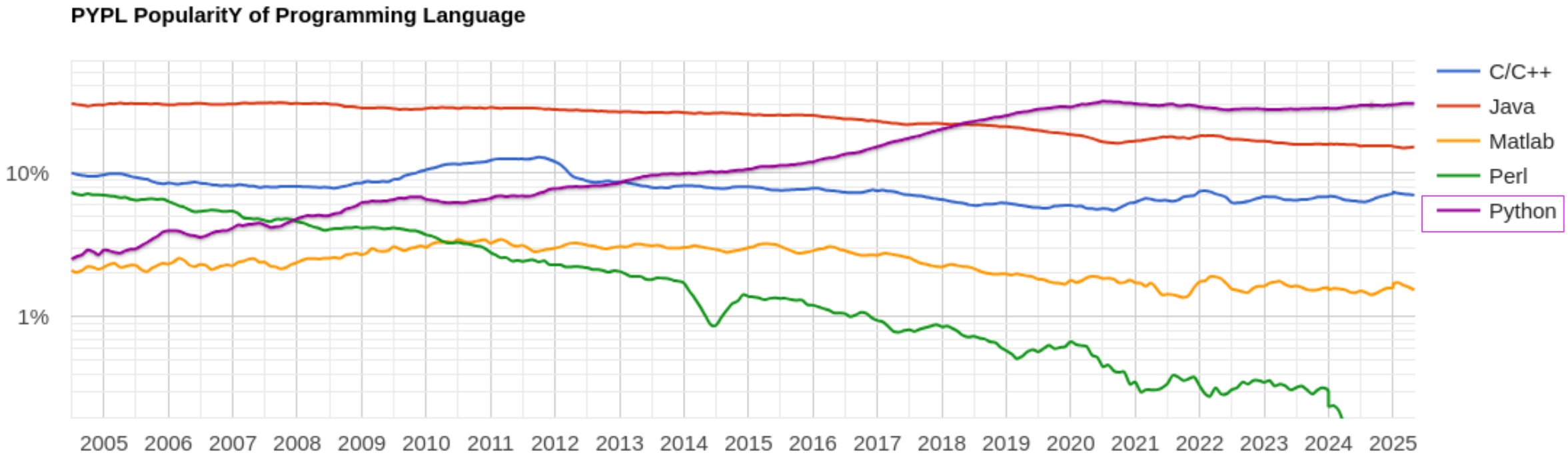
Why use Python?

Pros	Cons
- Free - Versatile and easy to learn - Fast to develop - Lots of libraries to use, several quite commonly used in physics - Easy to add non-python extensions - Open source with big community of users, great documentation - Intuitive syntax	- Not as fast as other languages - Memory can be an issue - Not easy to parallelize and do threaded computation - Not good for mobile development - Intuitive syntax

Credit: Minya Bai

Increase Usage of Python

- The PYPL PopularitY of Programming Language Index is created by analyzing how often language tutorials are searched on Google.
- Python is particularly popular in the fields of data analysis and machine learning, contributing to this rise.



Slide stolen from 2023's EIEIOO talk by Hannah Fronenberg

Versions

"Online I see some stuff in python2 and some stuff in python3, does it matter which one I use?"

YES

These two versions are mildly incompatible:

python2	python3
print a , print "hello world"	print(a), print("hello world")
7/2 = 3 (integer division)	7/2 = 3.5 (floating point division) 7//2 = 3 (integer division)

Rule #1: We always use python3

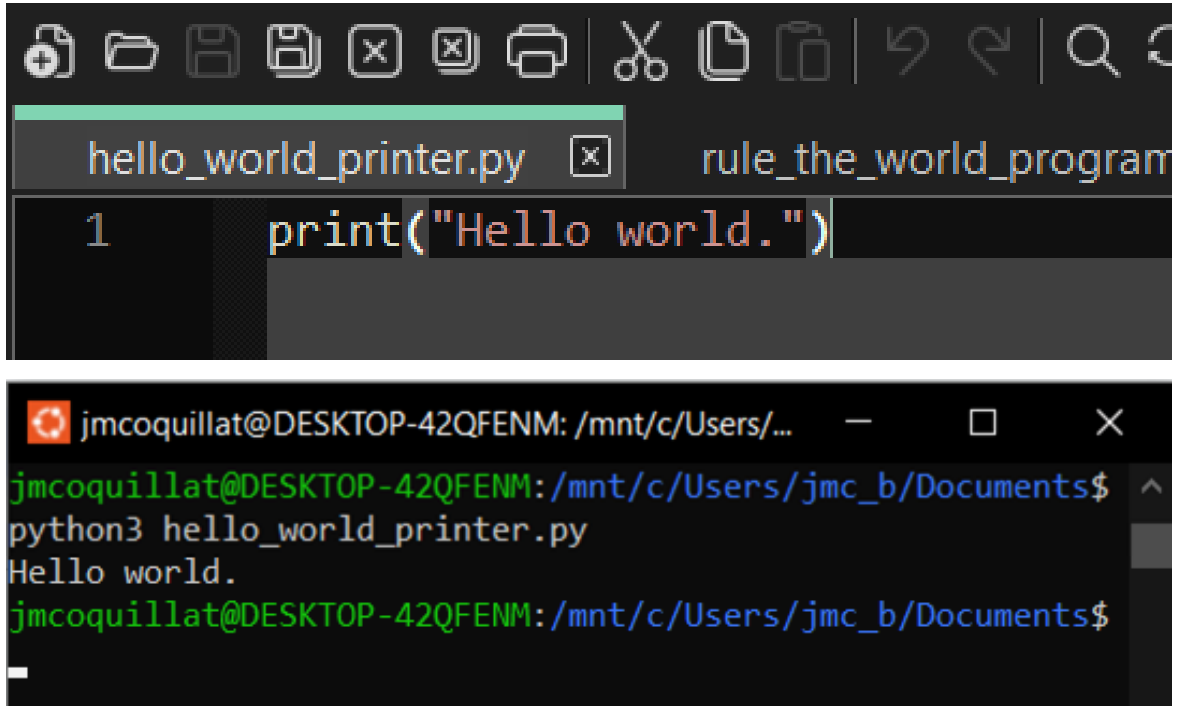
Rule#2: We never ever ever ever use python2 (unless you're using someone's old code that they rudely did not update for you, but it's okay because at least they optimized it and it runs really fast)

You can check which version you have by typing `$ python -V` into the command line

Two ways to code in python

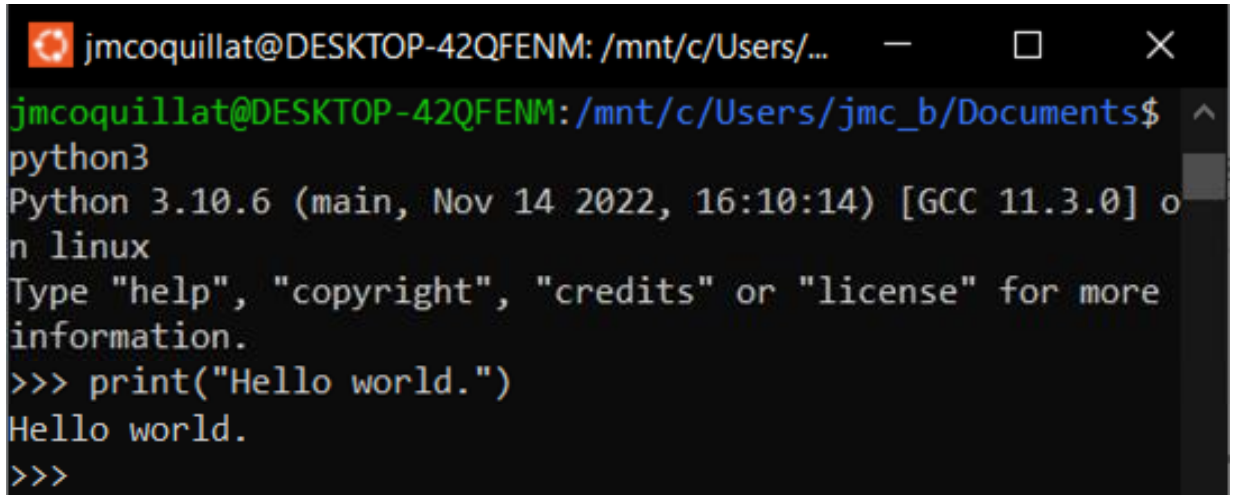
SCRIPT MODE

The code is already written, and is run at the end all at once.



INTERACTIVE MODE

The code is run line by line, as they are written.

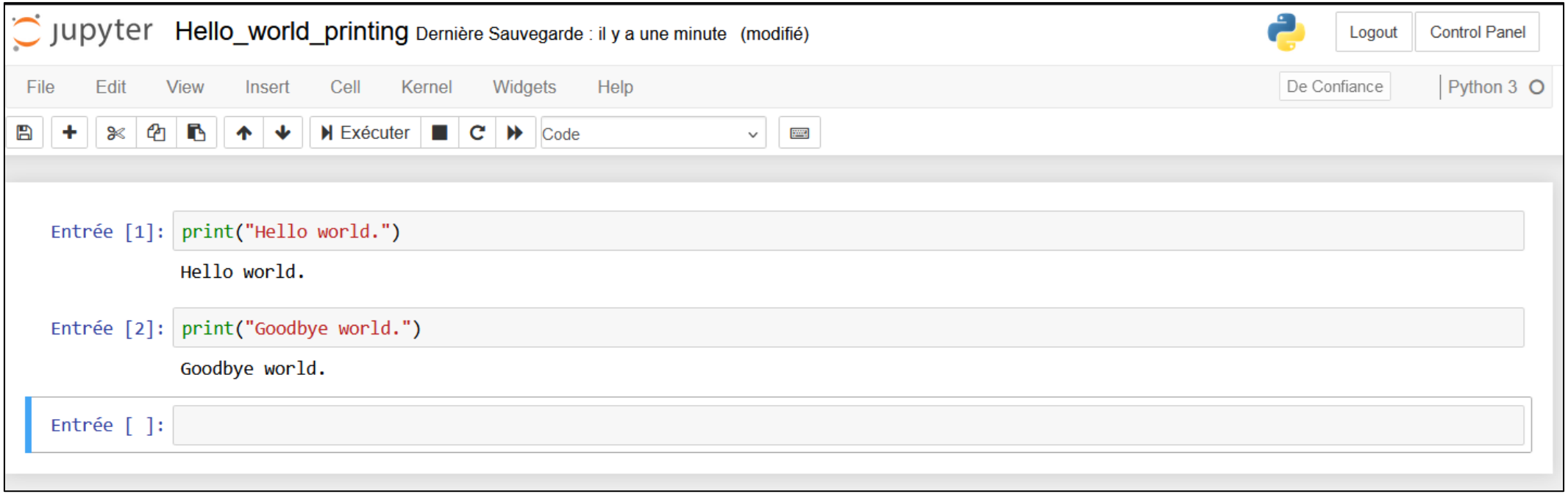


Jupyter Notebooks

- In-between of both python modes
- Code is run in blocks called “cells”
- Queen's students can use https://queensu.syzygy.ca/jupyter/

Can also run markdown to write notes!

For others who do not have access to it, https://colab.google/ can serve as an alternative.



Caution: *Cells run blocks of code independently. If you re-define a variable, you need to run the cell that defines the variables the way you need it.*

Basic mathematical operations

Operation	Symbol	Example
Assigning a variable	=	x=4
Addition	+	5+5 → 10
Subtraction	-	5-6 → -1
Multiplication	*	3*7 → 27
Division	/	7/2 → 3.5
Integer division	//	7//2 → 3
Exponentiation	**	2**3 → 8
Imaginary number	j	1j**2 → -1+0j
Modulo (remainder)	%	10%4 → 2
Powers of 10 (order of magnitude)	e	3e9 → 3×10^9

Credit: Hannah Fronenberg

Extra:

- Use parenthesis if needed
- Increment with **+=, -=, *=, /=.**

```
>>> x=1000
>>> x+=1
>>> print(x)
1001
```

x+=1
x=x+1

- Also, beware of floating-point imprecision

```
>>> 0.3-0.1-0.1-0.1
-2.7755575615628914e-17
```

Main objects in python

Object	Description	Examples
Boolean	Binary object, either True=1 or False=0.	True, False.
Integer (int)	Positive or negative integer number.	0, 1, -3, 42, -6472
Float	Positive or negative decimal number.	0.5, -12.34, 11.1111111
Complex	Number in the complex plane (use j for i)	0+1j, 12.5-34.5j, -10+1.2j
String	String of characters, inside " " or ' '	'0' , "Hello world." , ':)'
List	List of different objects, inside [ ]	[1,2,3], ["a", 2.5, [True,0.5]]
Tuple	Immutable list of same type of objects, inside ( )	(1,2,3), (True, False, False)
Numpy.array	Vectorized list from the numpy library	np.array([1,2,3])
Dictionary (dict)	Unordered list, with values associated to keys	{"a":1, "b":2, "c":3}
Function/Method	Function that can act, or output an object, often taking one or more objects as input.	print(), range(), type()

Elements of a list/array

REMEMBER:
Ordering starts at 0

You can also access a series of elements in a list

my_list[:b] ⇒ returns elements up to (but not including) index = *b*

my_list[a:] ⇒ returns all elements after (and including) index = *a*

my_list[a:b] ⇒ return elements from index *a* to *b-1* (inclusively)

E.g. *my_list = [0,1,2,3,4]*

>> my_list[:3] ⇒ [0,1,2]

>> my_list[3:] ⇒ [3,4]

>> my_list[2:4] ⇒ [2,3]

Credit: Minya Bai

Quick Note on Multi-Dimensional Arrays

- You can have a higher order arrays by having arrays of arrays.

  E.g. `x = [[1,2,3],`
  `[3,4,5],`
  `[5,6,7]]`

- Index higher order arrays ⇒ *x[i][j] = x[i, j]*
  - **i**-th array in the large array
  - **j**-th value in the i-th array

E.g. *x[1,0] = 3, x[2,1] = 6*

i\j	0	1	2
0	1	2	3
1	3	4	5
2	5	6	7

Credit: Minya Bai

Dictionaries

Dictionaries are a data structure that features a "*key-value*" pair.

- A defined *key* points to a list or tuple *value*.
- So in order to access the *value* for a given *key*, you need to feed it the *key* as an argument.

Example:

```
minyas_pet = {'name':'Cumin', 'animal':'cat', 'age':2}
```

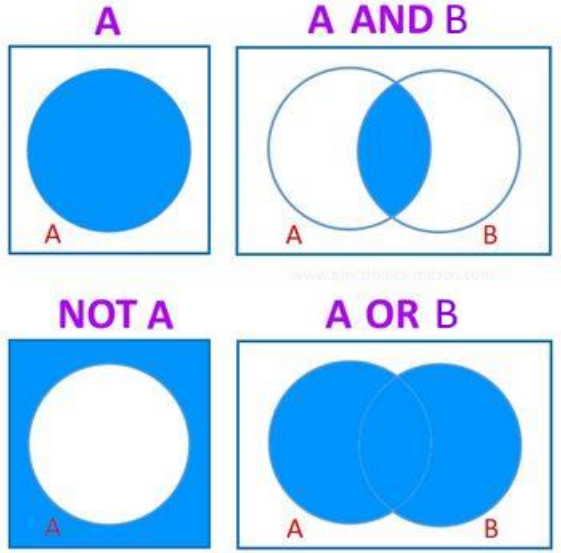
- `minyas_pet['name']` ⇒ returns `'Cumin'`
- Can similarly add to dictionary: `minyas_pet['colour']='gray'`

Credit: Minya Bai

Logic and conditions

```
if a>b:
    print("a is greater than b")
else if a<b:
    print("b is greater than a")
else:
    print("a is equal to b")
```

- Use logic statements to check if an (in)equality is **True** or **False**.
- Use logic or bitwise operators to combine logic statements.
- Use an **if** block to only execute indented code if it follows specific logic conditions.



Condition	Symbol	Example
Equal to	==	2+2 == 4 → True
Different from	!=	2*3 != 6 → False
Greater/lesser than	> / <	-3 > 5 → False
Greater/lesser or equal to	>= / <=	1/4 <= 0.25 → True

Logic operator	Bitwise operator
and	&
or	|
not	~

Loops

- Use a **`while`** loop to repeat executing code as long as a condition is met.
- Use a **`for`** loop to iterate over all elements of a list and repeat execution code for each iteration.
- Just like for **`if`** blocks, use indentation (4 spaces or 1 tab) after a **`:`** to select which part is inside the loop.
- You can nest a loop inside another loop.
- The **`break`** command will immediately end a loop.
- The **`continue`** command will stop the current iteration and skip to the next one.

```
for i in range(0,4,1):
    print(i)
```

```
n=0
while n<4:
    print(n)
    n+=1
```

Functions

```
def quad(a,b,c,both=True):
    pos=(-b+(b**2-4*a*c))/(2*a)
    neg=(-b-(b**2-4*a*c))/(2*a)
    if both: return [pos,neg]
    else: return pos
```

- Functions can be called to execute a piece of code, with optional inputs and output.
- Define a function with the **`def`** operator and indented instructions. You can give mandatory inputs and optional inputs that have a default value.
- Import function from libraries with the **`import`** operator.

Useful function	Description	Example
print(object)	Prints the value of the input.	print("1+2=",1+2) → 1+2= 3
type(object)	Returns type of the input.	type([23,4.5]) → <class 'list'>
int(numeral)	Returns the input converted to the integer type.	int(3.9) → 4
abs(number)	Returns the absolute value of the input.	abs(-10) → 10
range(min=0, max, step=1)	Returns a "list" starting at min, ending before max, with jumps of size step.	range(3) → [0,1,2] range(4,8,2) → [4,6]
len(object)	Returns the length of the input.	len("alphabet") → 8
LIST.append(element)	Appends the input at the end of the list it is applied to.	x=[1,2,3] x.append(7) print(x) → [1,2,3,7]

Errors and Exceptions

What if we don't want an error to end our analysis?
We can catch errors and deal with them accordingly without stopping the run-time of our program

- *TypeError*: Raised when function operates on an incorrect type.
- *ValueError*: Raised when function is given correct type but improper value.
- *IndexError*: Raised when index of sequence is out of range.
- *SyntaxError*: Raised by parser when there is an issue with the syntax.
- *ZeroDivisionError*: Raised when you try to divide by 0.
- And many more...

```
def division(x,y):
    try:
        print("It worked! The answer is:", x/y)
    except ZeroDivisionError:
        print("You cannot divide by zero!")


division(10,5)
division(10,0)
division(10,"pizza")
```

```
It worked! The answer is: 2.0
You cannot divide by zero!
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[19], line 12
     10 division(10,a)
     11 division(10,b)
---> 12 division(10,c)

Cell In[19], line 6, in division(x, y)
      4 def division(x,y):
      5     try:
----> 6         print("It worked! The answer is:", x/y)
      7     except ZeroDivisionError:
      8         print("You cannot divide by zero!")

TypeError: unsupported operand type(s) for /: 'int' and 'str'
```

Credit: Minya Bai

The numpy library

```
import numpy as np
```

Creating a numpy array

Function	Description	Example
np.array(list,dtype=float)	Returns a vectorized numpy array from a list. Can be multidimensional like a matrix.	x = np.array([[1,2,3],[4,5,6]]) x[0,1] → 2.0 x[:,-1] → np.array([3,6])
np.arange(min=0,max,step=1)	Same as range, but outputs an array and can contain floats instead of only integers.	np.arange(0.25,1.75,0.5) → np.array([0.25,0.75,1.25])
np.linspace(min,max,N)	Same as np.arange, but you input the number of equally spaced elements instead of the step size .	np.linspace(0.25,1.75,3) → np.array([0.25,1.0,1.75])
np.zeros(N) / np.ones(N)	Returns an array with only {input} zeros / ones	np.zeros(5) → np.array([0,0,0,0,0])
np.loadtxt(txtFile)	Imports data from a .txt file and returns it as a numpy array	f = np.loadtxt("datafile.txt")

Modifying a numpy array

Function	Description	Example
np.sort(arr) / np.argsort(arr)	Returns a sorted version of the input / the sorted argument of the input	np.sort([6,2,0]) → np.array([0,2,6]) np.argsort([6,2,0]) → np.array([2,1,0])
np.transpose(arr)	Returns a transposed version of the input	np.transpose([[1,2,3],[4,5,6]]) → np.array([[1,4],[2,5],[3,6]])

The numpy library

```
import numpy as np
```

Getting information from a numpy array

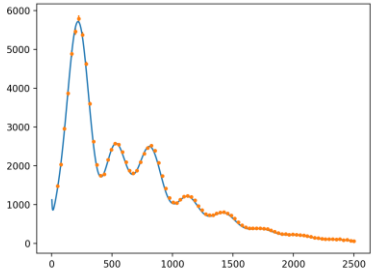
Function	Description	Example
np.min(arr) / np.max(arr)	Returns the minimum / maximum of the input	np.min([-2,41,10]) → -2.0
np.mean(arr) / np.median(arr)	Returns the mean / median of the input	np.mean([0,5,4]) → 3.0
np.sum(arr) / np.std(arr)	Returns the sum / standard deviation of the input	np.sum([-5,1,-3]) → -7.0
np.shape(arr)	Returns the shape of the input	np.shape([[1,2],[3,4],[5,6]]) → (3,2)
np.sum(arr) / np.std(arr)	Returns the sum / standard deviation of the input	np.sum([-5,1,-3]) → -7.0

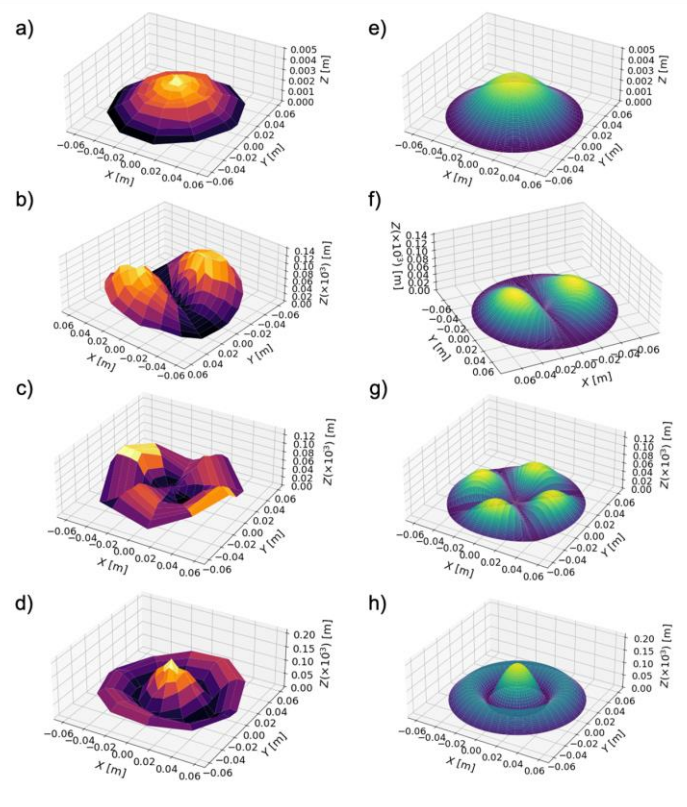
Math functions

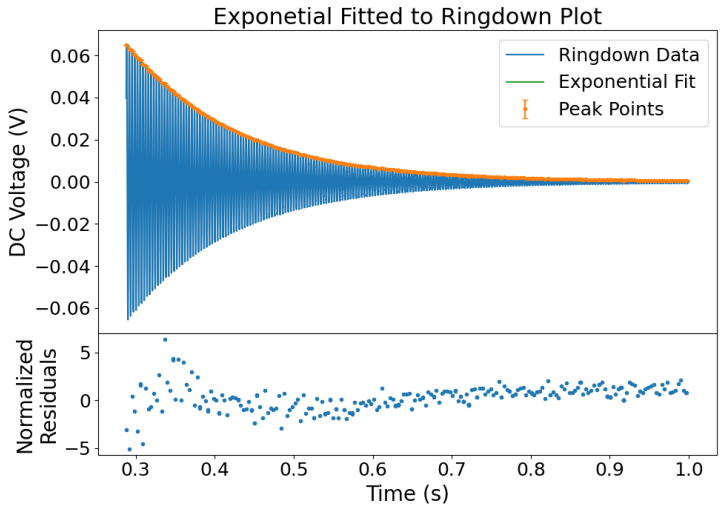
Function	Description	Example
np.sqrt(x) / np.exp(x) / np.pi	Returns $\sqrt{x}$ / e^x / π	np.exp(np.sqrt(np.pi)) → 5.885277...

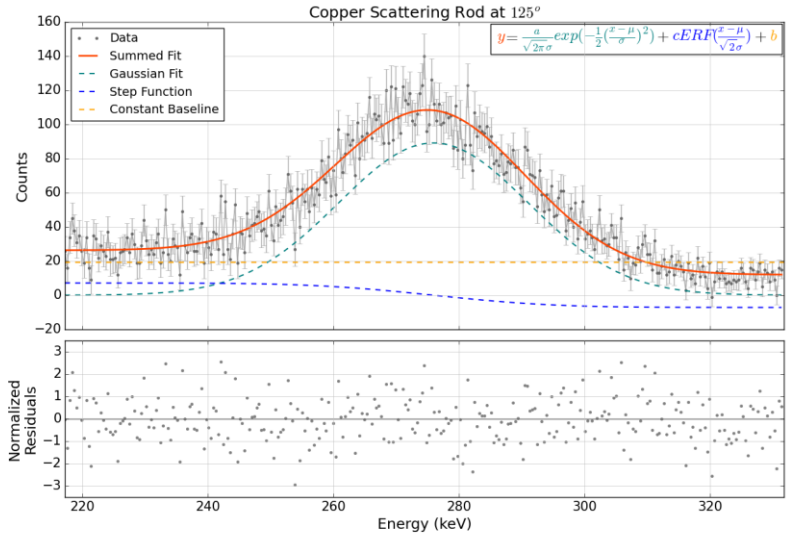
Matplotlib

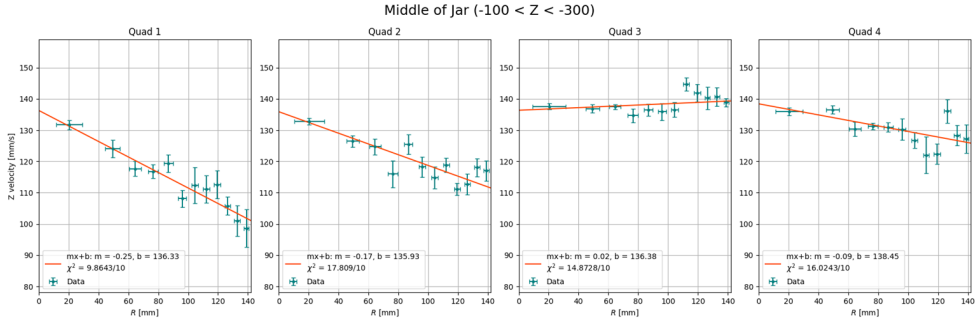
- Library used for data visualization in python
- More specifically, we like to use to pyplot package

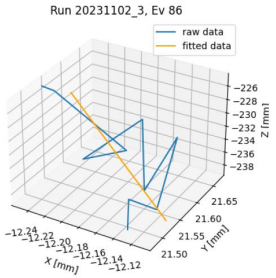


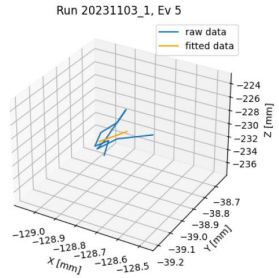


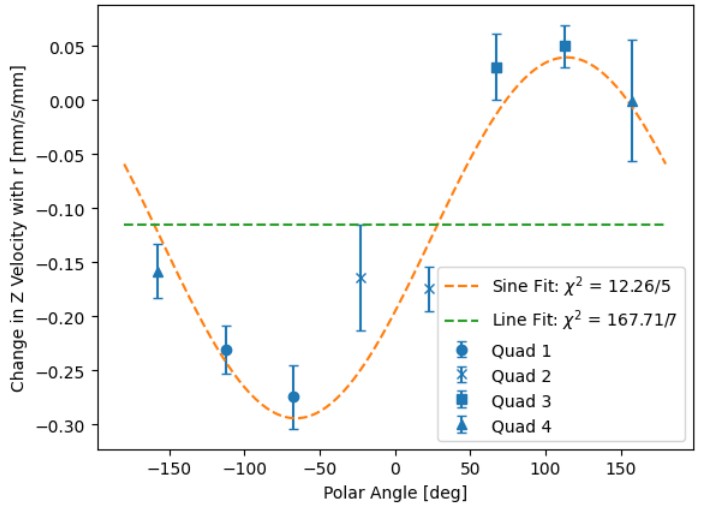












The matplotlib library

import matplotlib.pyplot as plt

Creating a plot

Function	Description	Example
plt.plot(x,y)	Plots x vs y	plt.plot(x, y, 'k.', label='data') plt.plot(x, x**2, 'r-', label='fit')
plt.errorbar(x,y,dy,dx)	Plots x vs y with errorbars	
plt.hist(data,nBins,[min,max])	Plots a 1D histogram of the data	h0 = plt.hist(x,50,[0,100],alpha=0.5) h1 = plt.hist(x,50,[0,100],density=1)
plt.hist2d(xdata,ydata,[nBinsX, nBinsY],[[xmin,xmax],[ymin,ymax]])	Plots a 2D histogram of the data	plt.hist2d(x,y,[20,50],[0,1],[0,100])

Modifying a plot

Function	Description	Example
plt.title(string)	Adds a title to the plot	plt.title("My beautiful graph")
plt.xlabel(string) / plt.ylabel(string)	Adds a label to the x / y axis	plt.xlabel("Time (s)")
plt.xscale('log') / plt.yscale('log')	Makes the x / y axis logarithmic	
plt.legend()	Adds a legend with the labelled plots	
plt.axis([xmin,xmax,ymin,ymax])	Selects the limits of each axis	plt.axis([-1,1,0,100])
plt.show()	Prints all the plots and clears them	

Get acquainted: https://matplotlib.org/stable/users/explain/quick_start.html

Matplotlib

For plotting, the typical structure of a figure is as follows:

Import library

```
import matplotlib.pyplot as plt
```

Create new figure

```
plt.figure()
```

Plot (x1,y1) and (x2,y2) data sets

```
plt.plot(x1, y1, ':', color= 'red', label= "data_set_1")
plt.plot(x2, y2, ':', color= 'blue', label= "data_set_2")
```

Label the x and y axes

```
plt.xlabel("x_axis_label")
plt.ylabel("y_axis_label")
```

These are stylistic parameters to change appearance of data

Limit the x and y axes

```
plt.axis([x0,x1,y0,y1])
```

Add a legend to your plot

```
plt.legend(loc="lower left")
```

Show the figure

```
plt.show()
```

Credit: Minya Bai

Documentation: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html

Matplotlib - Subplots

You can also plot more than 1 plot on a single figure!

Import library

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(nrows=1, ncols=2)
```

Plot (x1,y1) and (x2,y2) datasets on different subplots

```
ax[0].plot(x1, y1, '.', color= 'red', label= "data_set_1")
ax[1].plot(x2, y2, '.', color= 'blue', label= "data_set_2")
```

These are stylistic parameters to change appearance of data

Label the x and y axes for ax[0] figure

```
ax[0].set_xlabel("x_axis_label")
ax[0].set_ylabel("y_axis_label")
```

Add a legend to subplot ax[1]

```
ax[1].legend()
```

Show the figure

```
plt.show()
```


Credit: Minya Bai

Matplotlib - Types of Plots

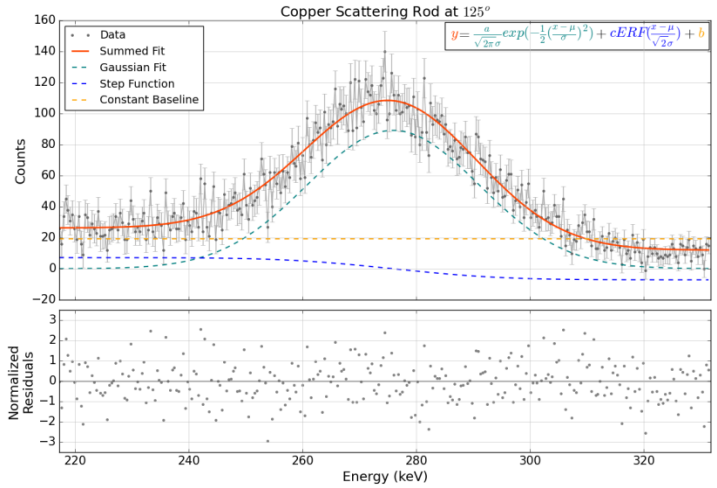
There is a plethora of graphs that you can make using this library. The common ones include:

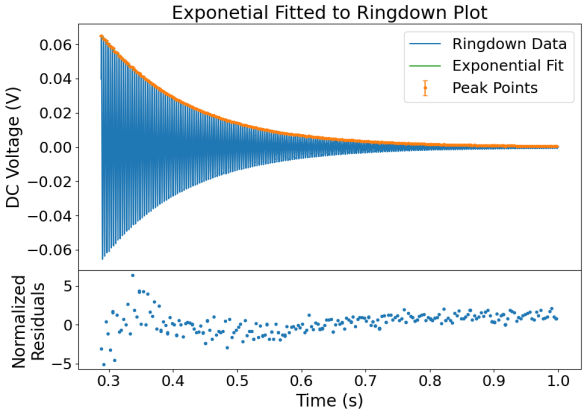
- Scatter plots
- Histograms
- 2D Histograms
- 3D Figures
- Polar Figures
- Contour Plots
- Quiver plots, etc.

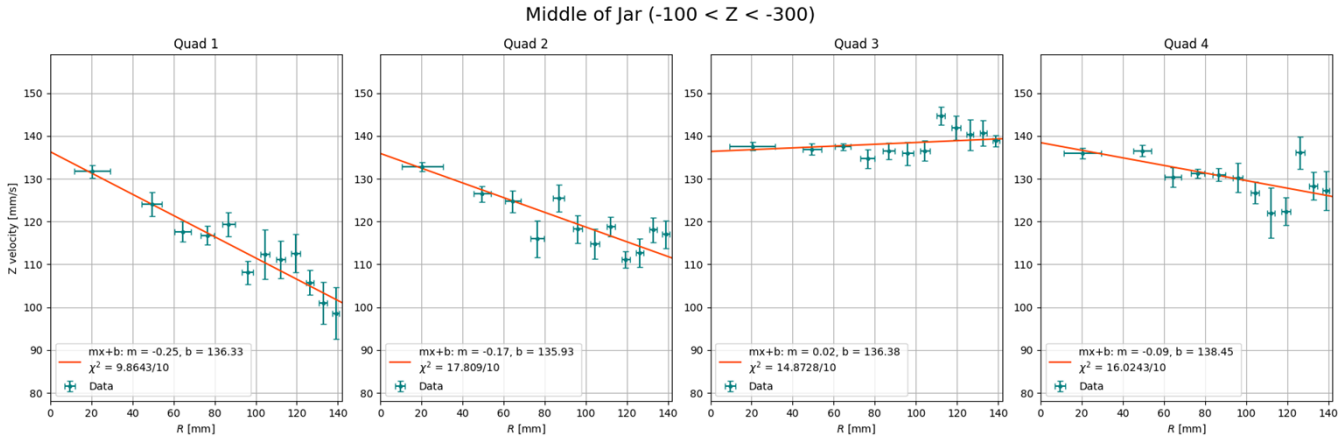
Explore the different types of plots here: https://matplotlib.org/stable/plot_types/index.html

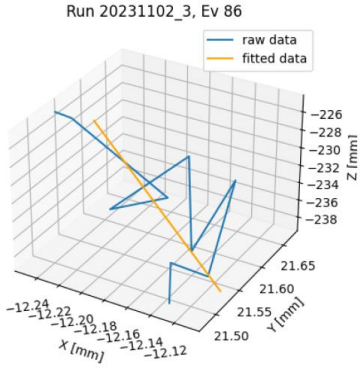
Scipy

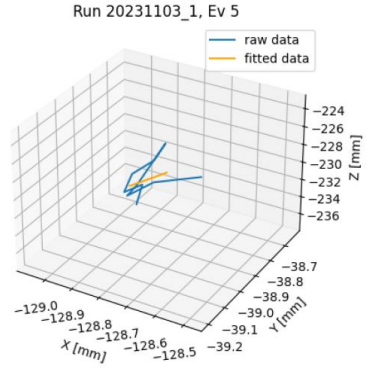
- Library that consists of many numerical algorithms that is useful for physics
- Purposes for algorithms:
  - **Optimization** ⇒ curvefit!
  - Statistics
  - Integration
  - Differential equations
  - Generation of random data sets
  - Etc.











scipy.optimize.curve_fit

- This is the scipy algorithm that is commonly used for frequentist data fitting.

Import algorithm

```
From scipy.optimize import curve_fit

def func(x, a, b):
    return a*x + b
```

Define the function you want to fit your data to

This is the data you want to fit

```
x = [data]
y = [data]
```

Fit function name

```
params, cov = curve_fit(func, x, y, p0=[a0, b0])
```

Curve_fit returns 2 outputs:
(1) the best fit parameters
(2) the covariance matrix

data

Initial parameters***

***** Look at your data and make a good guess for initial parameters**

Credit: Minya Bai

Other useful libraries

- **from scipy import stats**
  Contains multiple probability functions, from which you can get random variables or the probability density.
- **import iminuit**
  For Bayesian data fitting
- **import pandas as pd**
  Useful data analysis library
- **import time**
  Useful for timing code
- **from tqdm import tqdm**
  Tracks loop progress and estimated completion time
- Look online for the documentation on the functions you want!

Other topics for Computational Physics

- Solve systems of equations
- Interpolation and extrapolation of data
- Numeric Integration
- Random number statistics
- Max/Min of Functions
- Computationally solve ODEs/PDEs
- Solving Eigensystems
- Fourier Transforms
- Statistics
- Boundary Problems
- etc.

This was just a taster!

Credit: Minya Bai

Now, work and show what you learned!