ENPM 809T

UMCP, Mitchell

Python

- Python is an ***interpreted*** programming language
  - First released in 1991
- Source code file is read line-by-line and each statement is converted into machine language and executed before the next line is read
- Compiled languages are generally more efficient and faster than interpreted languages
  - The conversion to machine language only occurs one time
  - Then the machine language code can be executed whenever the program needs to be run
- Interpreted languages are often more **flexible** and changes can be easily made to the program at runtime

Python

- Unlike Matlab, Python is **open source**
- Python can be installed on any computer
- The source code is available for inspection
- Support and development are community driven (good…and bad)

Python

- Python has recently undergone a major redesign
- Python 3.x
  - Syntax has changed and is not compatible with 2.x
- Python 2.x
  - Most commonly used by scientific community
- Python 2.7
  - Bridge between the versions
  - Syntax from both will work

Python Basics

- Comments begin with #
  - Can be at the beginning of the line or in middle
    - # This is a comment
    - A = 45 # This sets the value of A
- Python is case sensitive
  - Apple = 'red'
  - apple = 67.1
- Names cannot start with a number
- Beware of "**Reserved Words**"
- Variables are empty (None or NULL) until assigned a value

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

https://docs.python.org/2.5/ref/keywords.html

Running a Python Program

- **python filename.py**
  - This will run the program then return to the command prompt
- **python -i filename.py**
  - Runs the program in "interactive" mode
  - When program ends, remain in the python environment and can examine the data types and values of variables

```
(pysteve) C:\Users\steve>python test.py
All packages imported properly!

(pysteve) C:\Users\steve>python -i test.py
All packages imported properly!
>>>
```

Data Types

- Python is ***dynamically typed***
- …the opposite of *statically-typed*
  - e.g. C, C++, Java
- Most variables do not need to be declared as real, integer, string, etc.
- The type of a variable is defined by the value assigned to it
- The type of a variable can change throughout the program execution

Data Types

- Boolean
  - Any non-Zero value becomes True
  - Zero or None is False
- Integer
  - Between -2147483648 and 2147483648 (32-bit)
  - Any integer larger, smaller, or ending in a capital L is a Long Integer
- Float
  - Automatically 64-bit or Double Precision
  - E or e indicates Scientific Notation

https://docs.scipy.org/doc/numpy-1.13.0/user/basics.types.html

Data Types

- To determine data type
  - type(var)
- Casting: force data type change
  - S = 16.6938
  - S = int(S)

Casting

- Initial data type is set by the value assigned
- Change to Float
  - x = float(x)
- Change to Integer
  - x = int(x) or x = long(x) for Long Integer
- Change to String
  - x = str(x)
- Change to Boolean
  - x = bool(x)

Python can be quirky!
...always consider data types

Strings

- Can be any combination of characters including non-ASCII
- Strings are surrounded by quotes: single, double or triple quotes
  - Triple quotes preserve formatting ('', or """)
- Tab (\t) and new line (\n)
- Escape character is \
  - If you actually want the characters \t to be in the string, not a tab you must "escape" it.
  - Text = "A tab is included in a string as \\t"

Strings

- x = 123.456789
- Text = "The value of x is {0:7.2f}".format(x)
  - print(Text)
  - The value of x is 123.46

Mathematical Operators

- + Addition
- - Subtraction
- * Multiplication
- / Division
- // truncating division
  - 16.3 // 5.2 => 3.0
- ** power
  - 2**3 = 8
- % modulo (returns the remainder)
  - 5 % 3 => 2

Flow Control

- User input from the command line
  - value = input("prompt string ")

Flow Control

- Conditional Statements
- The conditional ends with a colon and the code block is **indented**

```
if x:
        do this
else:
        do this
```

Flow Control

- Conditional Statements
- The conditional ends with a colon and the code block is indented

```
if x < 12:
        do this
elif x >= 12 and x < 30:
        do this
        and this
        and this
else:
        do this
```

Flow Control

- Loops
- Same format as conditionals

while *condition*:
Do this

Flow Control

- Loops
- Same format as conditionals

```
while x < 10:
        print(x)
```

Flow Control

- Loops
- Same format as conditionals

```
while x < 10:
        print(x)
        x = x + 1
```

Flow Control

- Loops
- Same format as conditionals

```
while x < 10:
        print(x)
        x += 1
```

Flow Control

- Loops
- Same format as conditionals

```
x = 0
while x < 10:
        print(x)
        x += 1
```

Lists & Tuples

- Able to store multiple values and different data types
  - Lists
    - Values can be changed
    - list = [2.5, 8.9e5, 'rain', [5,9,34.6], None]
  - Tuples
    - Values can NOT be changed after created
    - tuple = (2.5, 8.9e5, 'rain', [5,9,34.6], None)

https://webcourses.ucf.edu/courses/1249560/pages/python-lists-vs-numpy-arrays-what-is-the-difference

Lists & Tuples

- Accessing Data
  - Index
    - 0 is the first value
    - -1 is the last value
    - Index represents the boundary not the value held

Lists & Tuples

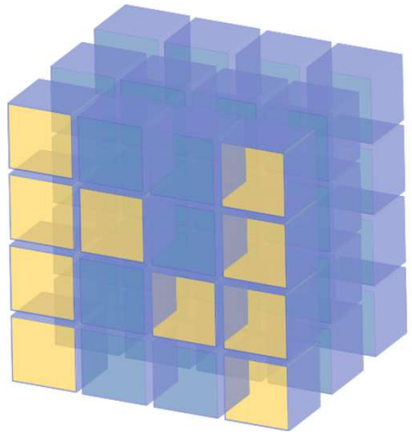
- Accessing Data
- **list = [2.5, 8.9e5, 'rain', [5,9,34.6], None]**
  - list[2]
    - 'rain'
  - list[-1]
    - None
  - list[0:2]
    - [2.5, 8.9e5]

Lists & Tuples

- Functions 'foo(list)' and methods 'list.foo()'
  - **list = [2.5, 8.9e5, 'rain', [5,9,34.6], None]**
  - len(list)
    - 5
  - list.count(None)
    - 1
  - list.index('rain')
    - 2

NumPy

- Fundamental package for **scientific computing** in Python
- Functionality for:
  - Multidimensional arrays
  - High-level math functions such as Fourier Transform
  - Pseudorandom number generators



NumPy

https://www.numpy.org/

NumPy

- Lists and Tuples can hold multiple data types
- NumPy arrays: all data stored as same type
- **import numpy** as **np**
- 1-D, 2-D, ….X-D
  - No limit to the number of dimensions
  - A = np.array([4,89,-1])
  - A = np.array([4.89,-1], dtype=*type*)
- 2-D
  - B = np.array([[23,41,3],[14,92,1]])

Accessing NumPy Array Data

- Start counting at 0
- End point non-inclusive
- A colon **:** means all
- A comma **,** separates dimensions

Accessing NumPy Array Data

- 1-D array A = np.array([100, 8, 54, 33, -500, 2])
- A #All of A
- A[3] #Value at index 3
- A[:4] #Values from beginning to 4
- A[4:] #Values from 4 to end
- A[::-1] #All values reverse order

Accessing NumPy Array Data

- 2-D array B = np.array([[32, 65, 800, 98],[-9, 43, 28, 150]])
- B #All of B
- B[1,0] #Value at row 1 column 0
- B[:,3] #Column 3 values for ALL rows
- B[1:,2] #Column 2 value for rows 1 to end

NumPy

- np.shape(A) # returns dimensions as tuple
- np.size(A) # number of elements
- len(A) # number of items of an object

NumPy Arrays

- c = np.array([[2.4, -23.73, 0],[0.76, 34, 23.4]], dtype=np.float)
  - np.shape(c)
  - (2,3)
- d = np.zeros((2,3), dtype=np.int)
- e = np.ones((4,4), dtype=np.float)
- f = np.empty((5,2), dtype=np.float)
- g = np.arange(5, 20, 0.5)
- h = np.linspace(1, 10, 20)

Combine NumPy Arrays

- np.concatenate((Arr1,Arr2), axis=0)

- **>>>** a = np.array([[1, 2], [3, 4]])
- **>>>** b = np.array([[5, 6]])
- **>>>** np.concatenate((a, b), axis=0)
- array([[1, 2],
- [3, 4],
- [5, 6]])
- **>>>** np.concatenate((a, b.T), axis=1)
- array([[1, 2, 5],
- [3, 4, 6]])

https://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays

Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                  Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)  Create an array of ones
>>> d = np.arange(10,25,5)           Create an array of evenly
                                     spaced values (step value)
>>> np.linspace(0,2,9)               Create an array of evenly
                                     spaced values (number of samples)
>>> e = np.full((2,2),7)             Create a constant array
>>> f = np.eye(2)                    Create a 2X2 identity matrix
>>> np.random.random((2,2))          Create an array with random values
>>> np.empty((3,2))                  Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myFile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64      Signed 64-bit integer types
>>> np.float32    Standard double-precision floating point
>>> np.complex    Complex numbers represented by 128 floats
>>> np.bool       Boolean type storing TRUE and FALSE values
>>> np.object     Python object type
>>> np.string_    Fixed-length string type
>>> np.unicode_   Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape          Array dimensions
>>> len(a)           Length of array
>>> b.ndim           Number of array dimensions
>>> e.size           Number of array elements
>>> b.dtype          Data type of array elements
>>> b.dtype.name     Name of data type
>>> b.astype(int)    Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                    Subtraction
 array([[-0.5,  0. ,  0. ],
        [-3. , -3. , -3. ]])
>>> np.subtract(a,b)             Subtraction
>>> b + a                        Addition
 array([[ 2.5,  4. ,  6. ],
        [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                  Addition
>>> a / b                        Division
 array([[ 0.66666667,  1.        ,  1.        ],
        [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)               Division
>>> a * b                        Multiplication
 array([[  1.5,   4. ,   9. ],
        [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)             Multiplication
>>> np.exp(b)                    Exponentiation
>>> np.sqrt(b)                   Square root
>>> np.sin(a)                    Print sines of an array
>>> np.cos(b)                    Element-wise cosine
>>> np.log(a)                    Element-wise natural logarithm
>>> e.dot(f)                     Dot product
 array([[ 7.,  7.],
        [ 7.,  7.]])
```

Comparison

```
>>> a == b                       Element-wise comparison
 array([[False,  True,  True],
        [False, False, False]], dtype=bool)
>>> a < 2                        Element-wise comparison
 array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)         Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()              Array-wise sum
>>> a.min()              Array-wise minimum value
>>> b.max(axis=0)        Maximum value of an array row
>>> b.cumsum(axis=1)     Cumulative sum of the elements
>>> a.mean()             Mean
>>> b.median()           Median
>>> a.corrcoef()         Correlation coefficient
>>> np.std(b)            Standard deviation
```

Copying Arrays

```
>>> h = a.view()         Create a view of the array with the same data
>>> np.copy(a)           Create a copy of the array
>>> h = a.copy()         Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()             Sort an array
>>> c.sort(axis=0)       Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]                 Select the element at the 2nd index
 3
>>> b[1,2]               Select the element at row 0 column 2
 6.0                     (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]               Select items at index 0 and 1
 array([1, 2])
>>> b[0:2,1]             Select items at rows 0 and 1 in column 1
 array([ 2.,  5.])
>>> b[:1]                Select all items at row 0
 array([[1.5, 2., 3.]])  (equivalent to b[0:1, :])
>>> c[1,...]             Same as [1,:,:]
 array([[[ 3.,  2.,  1.],
         [ 4.,  5.,  6.]]])
>>> a[ : :-1]            Reversed array a
 array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]               Select elements from a less than 2
 array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]   Select elements (1,0),(0,1),(1,2) and (0,0)
 array([ 4. ,  2. ,  6. ,  1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]   Select a subset of the matrix's rows
 array([[ 4. ,  5. ,  6. ,  4. ],   and columns
        [ 1.5,  2. ,  3. ,  1.5],
        [ 4. ,  5. ,  6. ,  4. ],
        [ 1.5,  2. ,  3. ,  1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)      Permute array dimensions
>>> i.T                      Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                Flatten the array
>>> g.reshape(3,-2)          Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))          Return a new array with shape (2,6)
>>> np.append(h,g)           Append items to an array
>>> np.insert(a, 1, 5)       Insert items in an array
>>> np.delete(a,[1])         Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)   Concatenate arrays
 array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))               Stack arrays vertically (row-wise)
 array([[ 1. ,  2. ,  3. ],
        [ 1.5,  2. ,  3. ],
        [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                     Stack arrays vertically (row-wise)
>>> np.hstack((e,f))               Stack arrays horizontally (column-wise)
 array([[ 7.,  7.,  1.,  0.],
        [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))         Create stacked column-wise arrays
 array([[ 1, 10],
        [ 2, 15],
        [ 3, 20]])
>>> np.c_[a,d]                     Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)           Split the array horizontally at the 3rd
 [array([1]),array([2]),array([3])]   index
>>> np.vsplit(c,2)           Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3. ,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]]])]
```

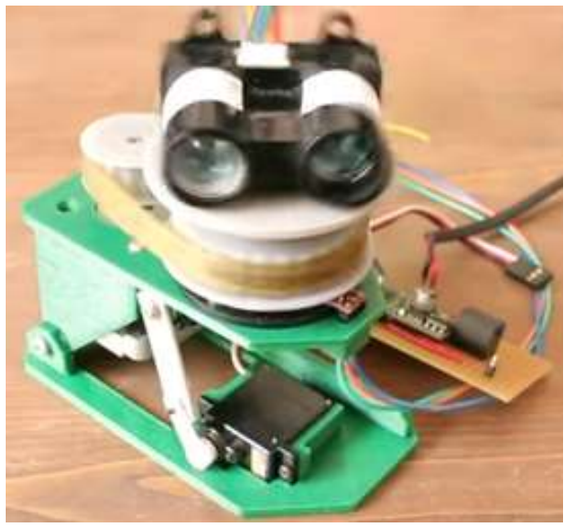
DataCamp
Learn Python for Data Science Interactively

In-Class Exercise

- Consider here a small COTS lidar sensor used for 3D mapping and navigation onboard a mobile robot
- As part of a simulation of the lidar sensor's ranging performance, complete the following:

1. Generate a NumPy array containing 10 numbers sampled from a random normal distribution with mean 30 cm and standard deviation 2.5 cm
2. Use Python and NumPy to calculate the mean and standard deviation of the array
3. Repeat the process for arrays containing 100, 1000, and 10000 random numbers

- Thinking ahead: what does this tell us about using sensors for navigation?



In-Class Exercise

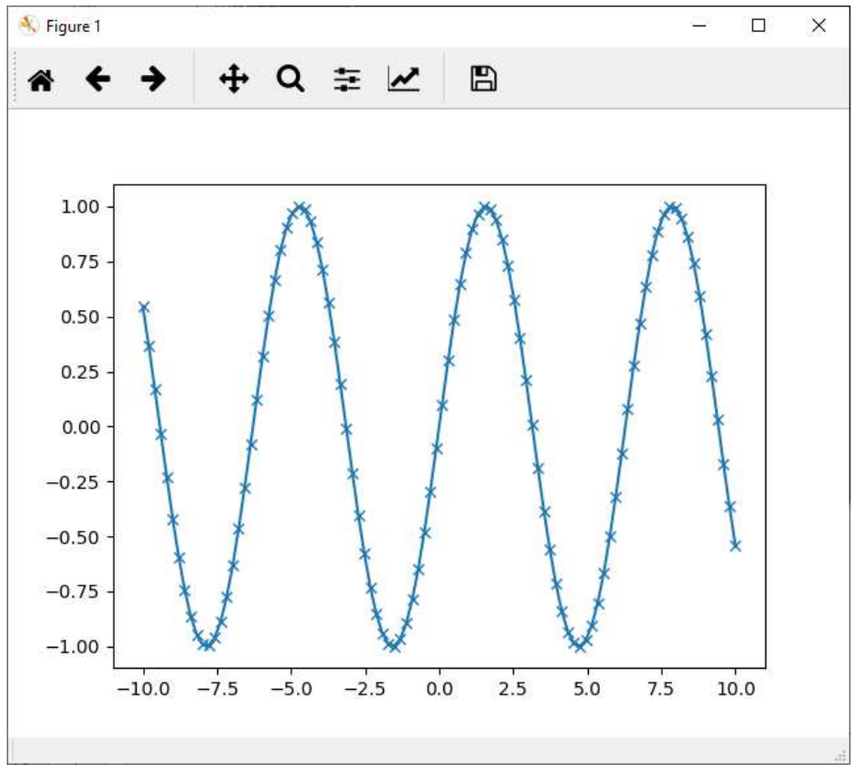
Matplotlib

- Primary **scientific plotting** library in Python
- Functions for making publication-quality visualizations
  - Line charts
  - Histograms
  - Scatter plots

https://matplotlib.org/

Matplotlib: 1-D Plotting

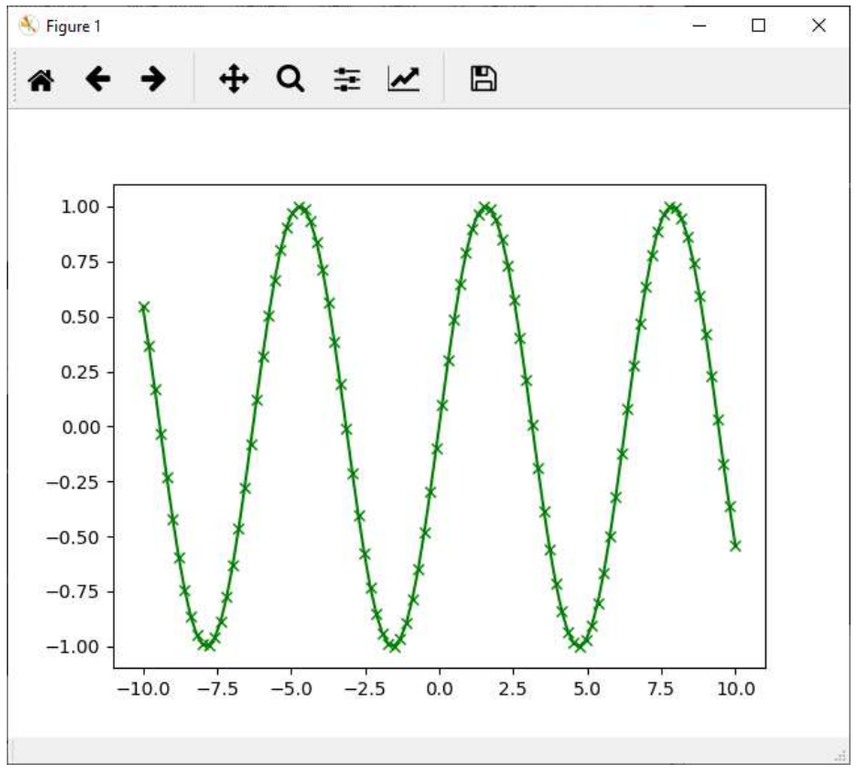
```
Anaconda Prompt - python
(base) C:\Users\steve>activate pysteve

(pysteve) C:\Users\steve>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
:: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>>
>>> x = np.linspace(-10, 10, 100)
>>> y = np.sin(x)
>>>
>>> plt.plot(x, y, marker = "x")
[<matplotlib.lines.Line2D object at 0x00000181145CE358>]
>>> plt.show()
```



https://matplotlib.org/2.1.1/api/_as_gen/matplotlib.pyplot.plot.html

Matplotlib: 1-D Plotting

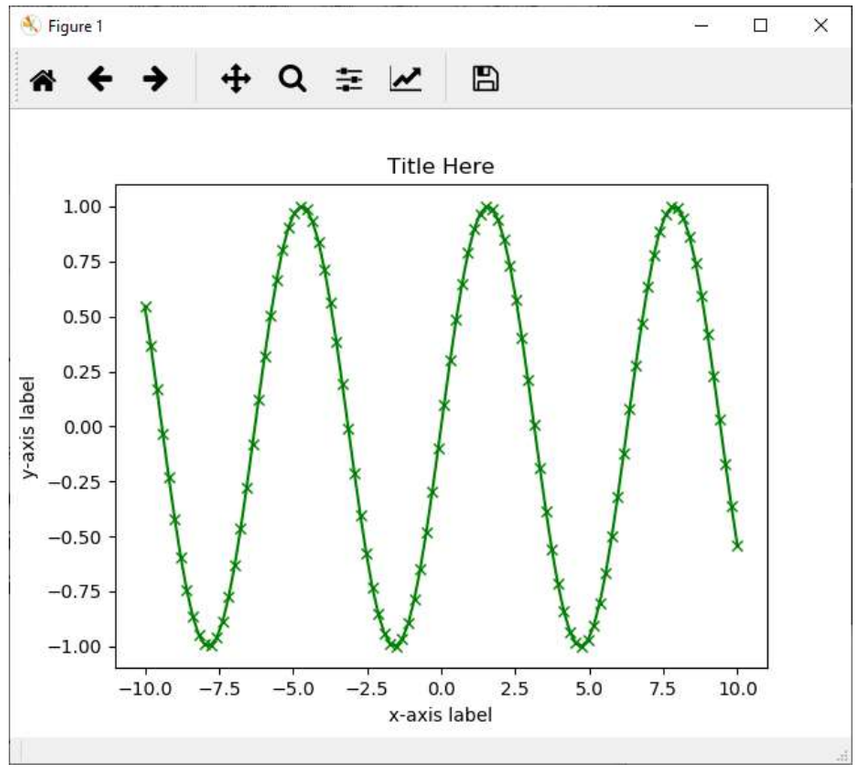
```
x = np.linspace(-10, 10, 100)
y = np.sin(x)

plt.plot(x, y, marker="x", c="g")
plt.show()
```



https://matplotlib.org/2.1.1/api/_as_gen/matplotlib.pyplot.plot.html

Matplotlib: 1-D Plotting

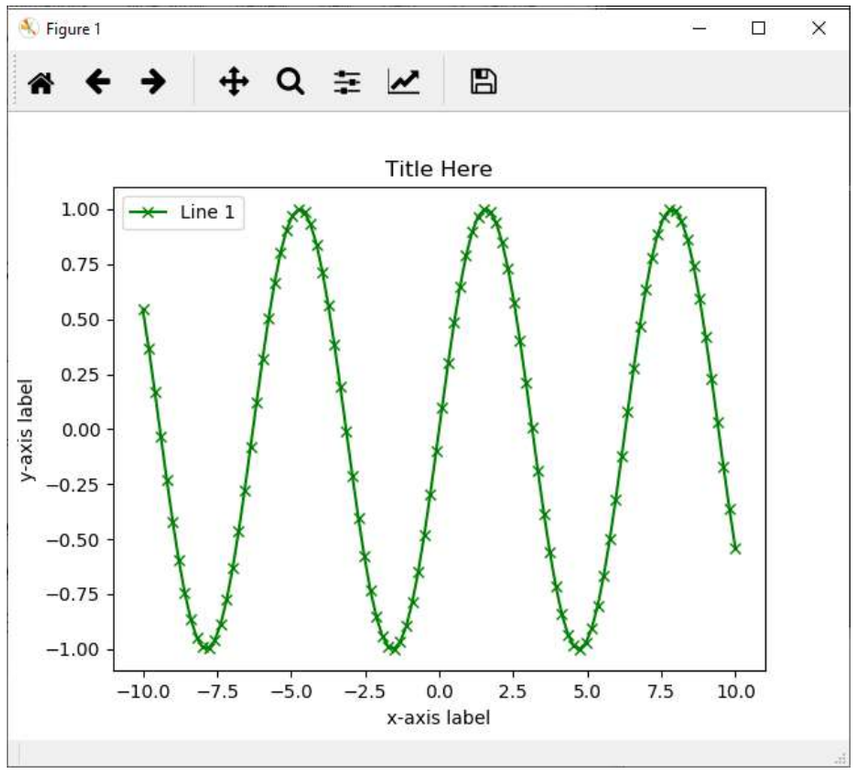
```
x = np.linspace(-10, 10, 100)
y = np.sin(x)

plt.plot(x, y, marker="x", c="g")
plt.xlabel('x-axis label')
plt.ylabel('y-axis label')
plt.title('Title Here')
plt.show()
```



https://matplotlib.org/2.1.1/api/_as_gen/matplotlib.pyplot.plot.html

Matplotlib: 1-D Plotting

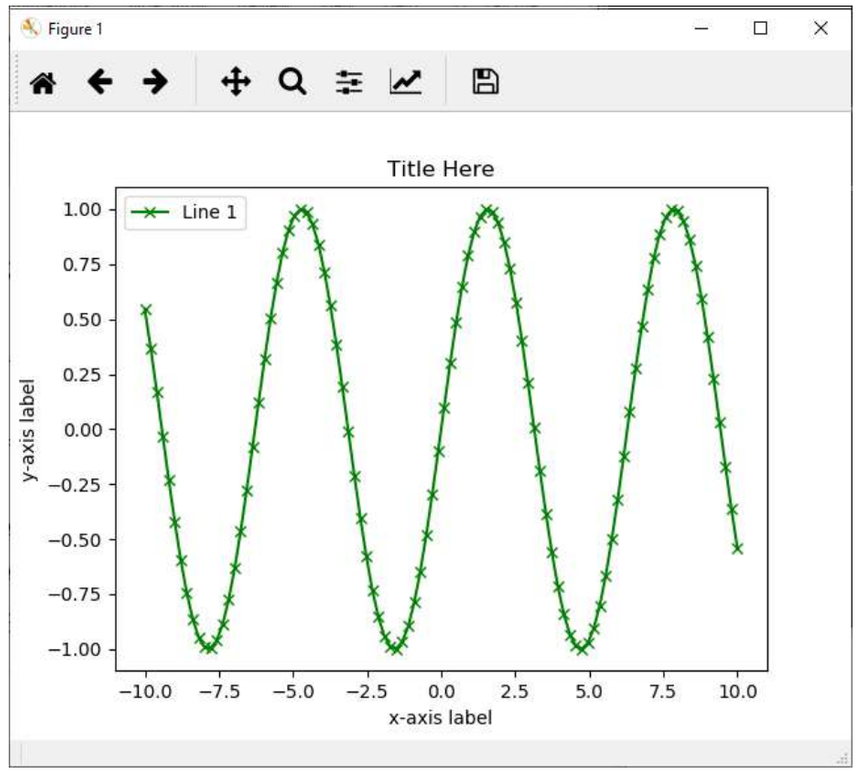
```
x = np.linspace(-10, 10, 100)
y = np.sin(x)

plt.plot(x, y, marker="x", c="g", label="Line 1")
plt.xlabel('x-axis label')
plt.ylabel('y-axis label')
plt.title('Title Here')
plt.legend()
plt.show()
```



https://matplotlib.org/2.1.1/api/_as_gen/matplotlib.pyplot.plot.html

Matplotlib: 1-D Plotting

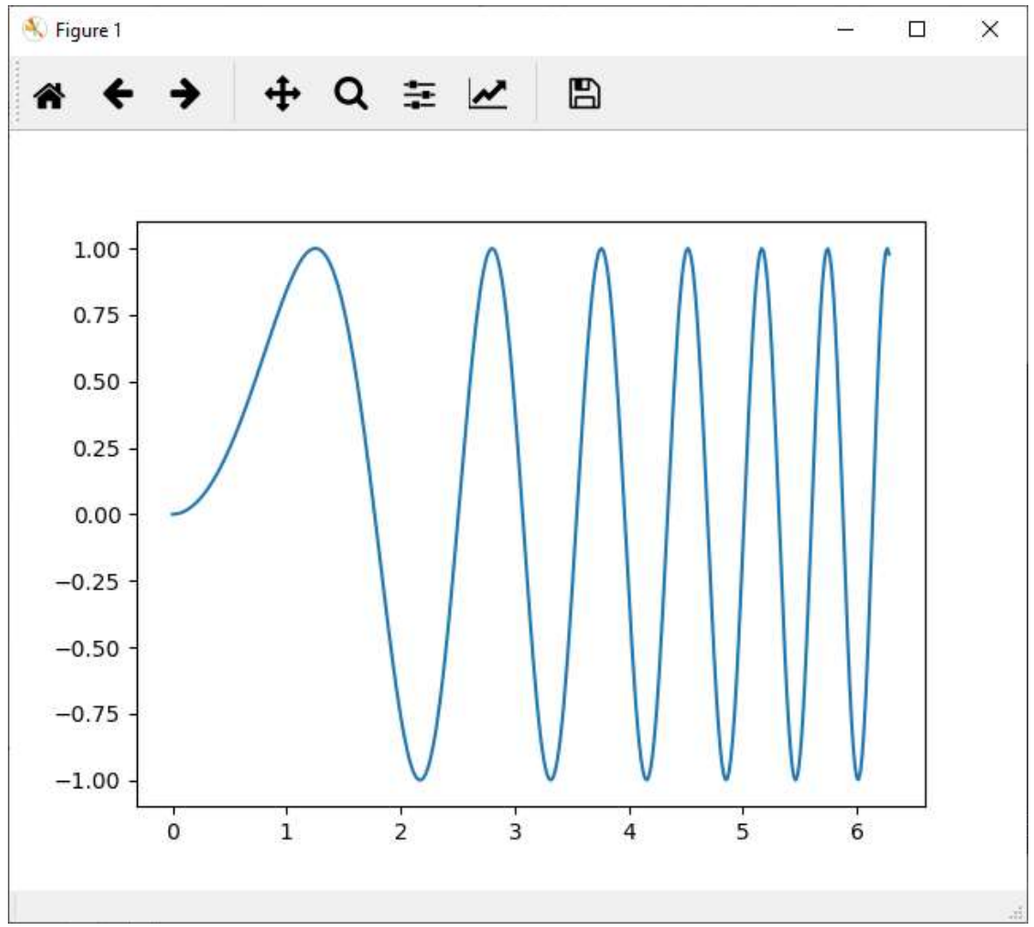
```
x = np.linspace(-10, 10, 100)
y = np.sin(x)

plt.plot(x, y, marker="x", c="g", label="Line 1")
plt.xlabel('x-axis label')
plt.ylabel('y-axis label')
plt.title('Title Here')
plt.legend()
plt.savefig("testplot.png")
plt.show()
```



https://matplotlib.org/2.1.1/api/_as_gen/matplotlib.pyplot.plot.html

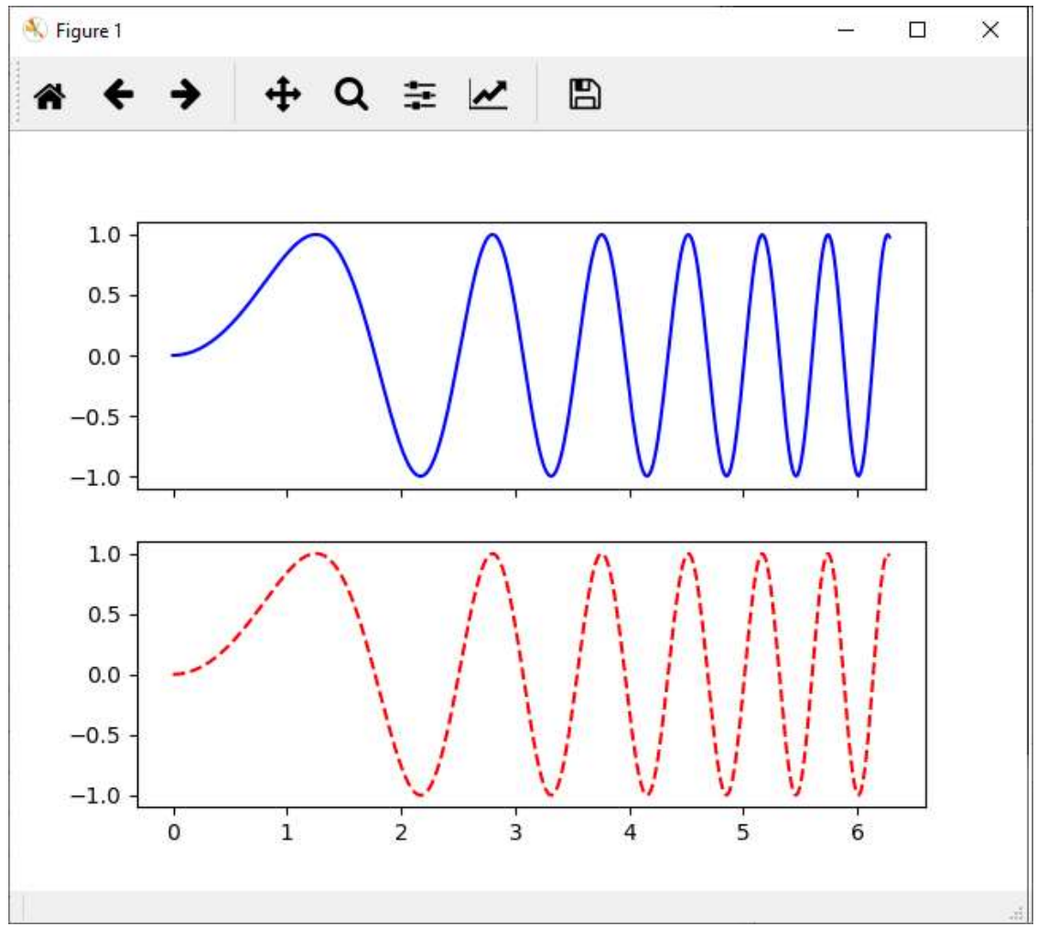
Subplots

```
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)

f, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```



Subplots

```
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)

f, ax = plt.subplots(2, sharex=True)
ax[0].plot(x,y,'b-')
ax[1].plot(x,y,'r--')
plt.show()
```



Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

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

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

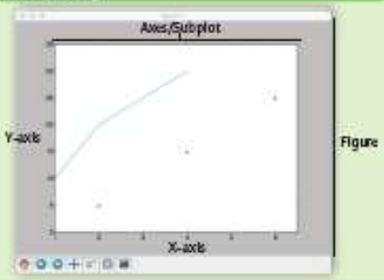
Axes

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]                      Step 1
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)  Step 3,4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                 Add padding to a plot
>>> ax.axis('equal')                        Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])   Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                     Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',         Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                   No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),          Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,        Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                      Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)    Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))  Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> lines = ax.plot(x,y)                    Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                         Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])          Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])     Plot horizontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                 Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                 Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')               Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')     Fill between y-values and 0
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)            Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                   Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)           Plot 2D vector fields
```

Data Distributions

```
>>> ax1.hist(y)                             Plot a histogram
>>> ax3.boxplot(y)                          Make a box and whisker plot
>>> ax3.violinplot(z)                       Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                     Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                  Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)               Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                 Plot contours
>>> axes2[2].contourf(data1)                Plot filled contours
>>> axes2[2]= ax.clabel(CS)                 Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                               Clear an axis
>>> plt.clf()                               Clear the entire figure
>>> plt.close()                             Close a window
```

DataCamp
Learn Python for Data Science Interactively

Functions

- def function_name(inputA, inputB, inputC):

    do some stuff

    do more stuff

    return (x,y)

- def square_of_x(x):

    return x**2

Variables in Functions

- Variables used in functions are completely separate from variables outside the function
- X inside the function is NOT the same as X in the rest of your script
- To change the value of a variable outside the function you must declare
  - global x
  - x = 10

References

- *AOSC458J Scientific Programming: Python*, Jeffrey Henrikson
  - https://www.coursicle.com/umd/courses/AOSC/247/
- *Python Programming and Visualization for Scientists*, Alex DeCarla 2016
- *Numpy*
  - https://www.numpy.org/
- *Matplotlib*
  - https://matplotlib.org/