Intro to Python for Data Science and Machine Learning

Marc Berghouse and Lazaro Perez

Introduction

- Marc Berghouse, 3rd year Hydrology PhD Student
  - PhD research on microbial motility and particle tracking
  - Mammogram classification with deep learning
- Please introduce yourselves!
  - Name, program, research interests

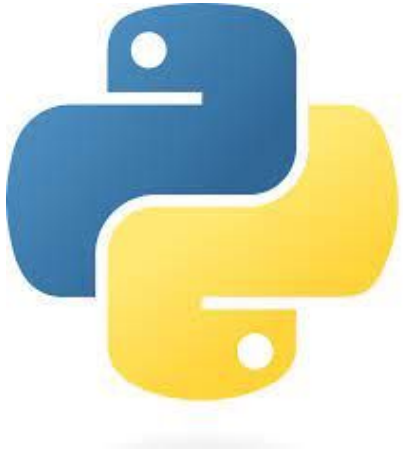
My Teaching Format

- Lecture and Coding Sessions
  - The first couple classes will be mostly lecture based, then time will be split about 50-50
- Teaching goals
  - How to get some basic Python code running
  - How to load and process data
    - We will start off simple, then go up to reading and parsing data from the USGS NWIS and Snotel
  - Basic data analysis and visualization skills
  - Different machine learning algorithms
  - How to use sci-kit learn to create machine learning models in Python
  - How to select the best model for a given dataset
  - How to program the entire machine learning pipeline start to finish
  - How to apply these skills to a variety of hydrologic data sets and develop cutting edge analyses for the field of hydrology
    - Try to design the scope of your project for a future publication. At then end of the semester you should have initial steps and analysis done, which will help you determine if you could actually publish your work later on.

Why Python?

- Most popular programming language for ML
- Relatively easy to read/write
- Huge user base
  - There is a tutorial for everything
  - There is a package for everything
  - If there is not a package for something, there is probably some code on StackOverflow

Basics of Python



- Installation
- Development environment (script vs notebook)
- Libraries
- Anaconda (python environments)
- Data types
- Basic python (math, for loops, functions, if statements, boolean indexing, etc.)

Installation

- I’ve found the best way to install Python is through Anaconda
  - If anyone has issues with Anaconda installation, we can try to figure it out in class during a coding session
  - If you can’t use Anaconda on your personal device for some reason, you can use kaggle. I’ll give a Kaggle tutorial today or next class

Scripts vs Notebooks

- Scripts are just files with Python code, and they must end with the .py extension
  - Scripts can simply be run from the terminal (or in an IDE such as Spyder), which allows for the development of complex applications
- Notebooks are development environments where each “cell” basically acts as its own script that is run individually
  - Notebooks are good for displaying results, which makes them great for teaching and learning
  - Great for exploratory programming, bad for final development
- In this class all of my homework assignments will be programmed in something called Jupyter notebooks, an IPython notebook environment that’s installed with Anaconda.

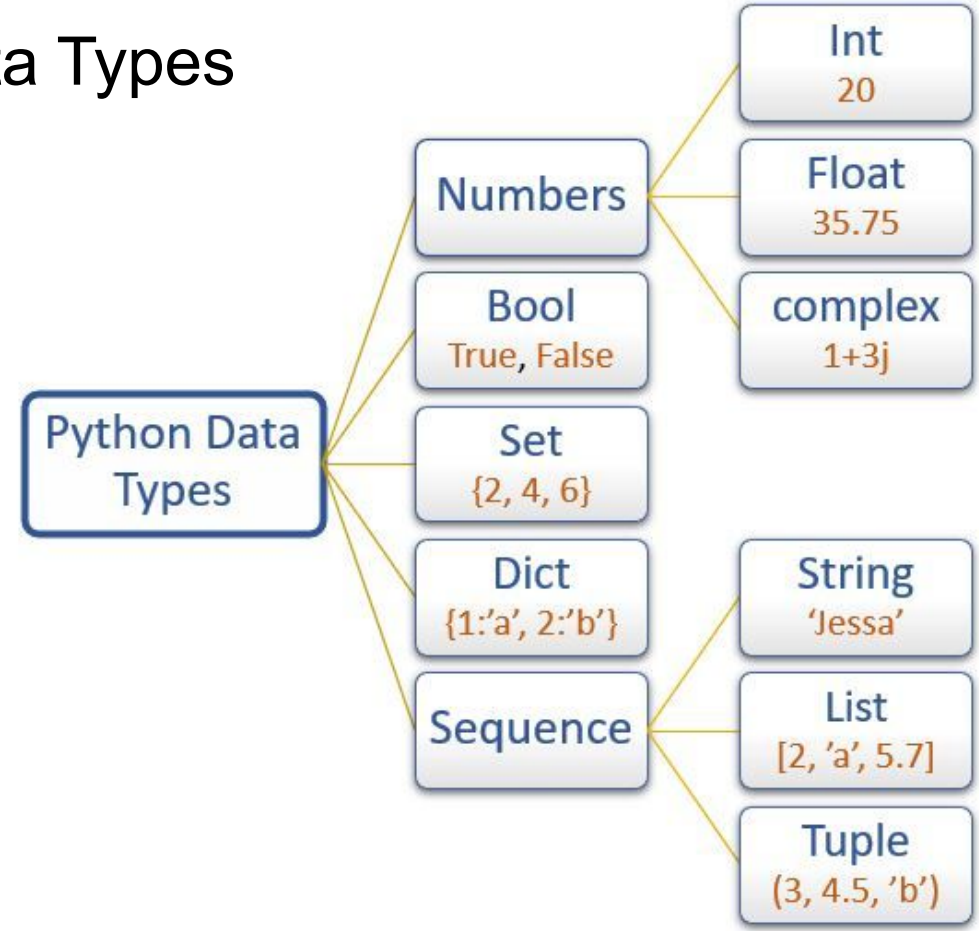
Libraries

- Pandas - Dataframes, data parsing, data analysis
- Numpy - Data arrays, basic statistics
- Scipy - Advanced statistics
- Os - miscellaneous operating system interfaces: read, write, rename, move, list files and directories
- Matplotlib - Plots and figures
- Sci-kit learn - Machine learning
- Open-CV - Image analysis
- To install most libraries, just do !pip install “package_name” in a notebook cell, or the same command in terminal but without an exclamation point
    - I would look up the libraries first so you don’t accidentally install the wrong thing

Anaconda (Python Environments)

- Anaconda is a program that maintains your Python environments and allows easy switching between different environments
- The Python environment is essentially the collection of libraries installed on your computer that are associated with a particular version of Python
    - For example, say we need to use two programs but one is written in Python 2.5 and another is written in Python 3.8. You can't run both these programs on a single environment, since they depend on different versions of Python. If the programs are too long to rewrite, then it makes sense to create 2 different environments (one with Python 2.5 and another with Python 3.8)
- Anaconda gives you a number of tools to write and run Python code. I will only use Jupyter notebooks, but I know some people prefer Spyder

Python Data Types



https://pynative.com/python-data-types/

Basics of Python Programming

- Syntax and Variables
- Lists
- Arrays
- If/Else statements
- For loops
- While loops
- Functions
- Boolean indexing
- Best practices

Syntax

- Indentations and spaces matter (use tab indent for all if/else, for loops, while loops, and function definitions)
- Case sensitive
- For file paths use forward slashes
- Single, double or triple quotes can be used for strings
- Use # to comment out lines of code
- 0 indexing

Variables

- Store data values
- x=5
- name=”Marc”

Lists

- The most “Pythonic” way to store data in Python
  - Essential for data manipulation
- Allows for organization of multi-dimensional data
- Say you have a bunch of images of different sizes
  - You can’t use a numpy array, because these have predefined dimensions
  - You must use lists to store the data
- Similar to cells in Matlab

```
l=[]
for i in range(10):
    l.append(i)
print ('list: ',l,'\nelement 0 of the list: ',
       l[0],'\nlength of the list: ',len(l))
```

```
list:  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
element 0 of the list:  0
length of the list:  10
```

Arrays

- Use numpy to create arrays

```
import numpy as np
arr=np.array([1,2,3,4,5,6])
print (arr, arr.shape)
```

```
[1 2 3 4 5 6] (6,)
```

```
import numpy as np
arr=np.array([[1,2,3],[4,5,6]])
print (arr, arr.shape)
```

```
[[1 2 3]
 [4 5 6]] (2, 3)
```

```
arr=np.zeros((5,5))
print (arr, arr.shape)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]] (5, 5)
```

```
arr=np.zeros((5,5))
arr[0,:]=1
print (arr, arr.shape)
```

```
[[1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]] (5, 5)
```

```
arr=np.zeros((5,5))
arr[:,0]=1
print (arr, arr.shape)
```

```
[[1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]] (5, 5)
```

```
#np.zeros((rows,columns))
arr1=np.zeros((5,5))
arr2=np.zeros((5,2))
arr=np.hstack((arr1,arr2))
print (arr, arr.shape)
```

```
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]] (5, 7)
```

```
#np.zeros((rows,columns))
arr1=np.zeros((5,5))
arr2=np.zeros((2,5))
arr=np.vstack((arr1,arr2))
print (arr, arr.shape)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]] (7, 5)
```

```
arr = np.array([0,1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
```

```
[4 5 6 7]
```

```
arr = np.array([0,1, 2, 3, 4, 5, 6, 7])
print(arr[:4])
```

```
[0 1 2 3]
```

```
arr = np.array([0,1, 2, 3, 4, 5, 6, 7])
print(arr[0:6:2], arr[6:0:-2], arr[4:-2],arr[-6:-4])
```

```
[0 2 4] [6 4 2] [4 5] [2 3]
```

Array Math

```
arr=np.random.randint(0, high=10, size=(3,3))
arr
```

```
array([[2, 5, 7],
       [2, 3, 7],
       [0, 6, 5]])
```

```
arr+1
```

```
array([[3, 6, 8],
       [3, 4, 8],
       [1, 7, 6]])
```

```
arr*2
```

```
array([[ 4, 10, 14],
       [ 4,  6, 14],
       [ 0, 12, 10]])
```

```
print (np.mean(arr), np.std(arr), np.sum(arr))
```

```
4.111111111111111 2.3306863292670035 37
```

```
print ("mean of columns: ",np.mean(arr,axis=0),
       "\nmean of rows: ", np.mean(arr,axis=1))
```

```
mean of columns:  [1.33333333 4.66666667 6.33333333]
mean of rows:  [4.66666667 4.         3.66666667]
```

```
arr2=np.random.randint(0, high=10, size=(3,3))
arr2
```

```
array([[9, 3, 3],
       [7, 8, 8],
       [7, 2, 2]])
```

```
#Not dot product. This is element-wise multiplication
arr*arr2
```

```
array([[18, 15, 21],
       [14, 24, 56],
       [ 0, 12, 10]])
```

```
arr2=np.random.randint(0, high=10, size=(2,2))
arr2
```

```
array([[1, 3],
       [7, 1]])
```

```
arr*arr2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [191], in <cell line: 1>()
----> 1 arr*arr2

ValueError: operands could not be broadcast together with shapes (3,3) (2,2)
```

If/Else Statements

```
arr = np.array([0,1, 2, 3])
if 1 in arr:
    print ('True')
if arr[0]==0:
    print ('True')
```

```
True
True
```

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

```
Both conditions are True
```

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

```
=
```

```
arr = np.array([0,1, 2, 3])
if arr[0]>arr[1]:
    print ('Greater')
elif arr[0]==arr[1]:
    print ('Equal')
else:
    print ('Less')
```

```
Less
```

```
arr = np.array([2,1, 2, 3])
if arr[0]>arr[1]:
    print ('Greater')
elif arr[0]==arr[1]:
    print ('Equal')
else:
    print ('Less')
```

```
Greater
```

```
arr = np.array([0,1, 2, 3])
if (1 in arr) or (2 in arr) or (3 in arr):
    if arr[3]>0:
        arr[3]=arr[2]
    if arr[2]>0:
        arr[2]=arr[1]
    if arr[1]>0:
        arr[1]=arr[0]
arr
```

```
array([0, 0, 1, 2])
```

For and While Loops

```
arr = np.zeros((3,5))
count=0
icount=0
for i in range(len(arr[:,])):
    icount=icount+1
    for j in range(len(arr[0,:])):
        arr[i,j]=count
        count=count+1
print (arr, count, icount)
```

```
[[ 0.  1.  2.  3.  4.]
 [ 5.  6.  7.  8.  9.]
 [10. 11. 12. 13. 14.]] 15 3
```

```
arr = np.zeros((3,5))
for i in range(len(arr[:,])):
    for j in range(len(arr[0,:])):
        if i==j:
            arr[i,j]=count
        else:
            arr[i,j]=1
arr
```

```
array([[15.,  1.,  1.,  1.,  1.],
       [ 1., 15.,  1.,  1.,  1.],
       [ 1.,  1., 15.,  1.,  1.]])
```

```
x=10
count=0
while x>2:
    x=x-1
    count=count+1
print (x,count)
```

```
2 8
```

```
arr = np.array([0,1, 2, 3])
count = 0
for i in range(len(arr)):
    if np.any(arr)>0:
        if arr[3]>0:
            arr[3]=arr[2]
            if arr[2]==0:
                print ("it's all 0's now")
                break
        if arr[2]>0:
            arr[2]=arr[1]
        if arr[1]>0:
            arr[1]=arr[0]
    count=count+1
print (arr, count)
```

```
it's all 0's now
[0 0 0 0] 2
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
        break
```

```
apple
banana
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)
```

```
apple
cherry
```

Functions

```
def my_function():
    print("Hello from a function")
my_function()
```

```
Hello from a function
```

```
a=np.array([1,1,0,1,0])
b=np.array([1,1,10,1,0])
def mean_square_error(x,x_pred):
    mse=(1/len(x)*np.sum((x-x_pred)**2))
    return mse
mean_square_error(a,b)
```

```
20.0
```

```
a=np.array([1,1,0,1,0])
b=np.array([1,1,1,0,1])
def mean_square_error(x,x_pred):
    mse=(1/len(x)*np.sum((x-x_pred)**2))
    return mse
mean_square_error(a,b)
```

```
0.6000000000000001
```

```
def square(x):
    sq=x**2
    return sq
square(5)
```

```
25
```

```
def tri_recursion(k):
    if(k > 0):
        result = k + tri_recursion(k - 1)
        print(result)
    else:
        result = 0
    return result

print("Recursion Example Results")
tri_recursion(3)
```

```
Recursion Example Results
1
3
6
```

Boolean Indexing

```
arr=np.random.normal(0,1,size=(3,3))
arr
```

```
array([[-0.35777573, -0.01681105,  1.59329057],
       [-0.50261512,  1.02920702,  1.56725883],
       [ 1.93291742, -0.98512758,  0.18921954]])
```

```
arr[arr<0]
```

```
array([-0.35777573, -0.01681105, -0.50261512, -0.98512758])
```

```
print(np.where(arr<0))
```

```
(array([0, 0, 1, 2]), array([0, 1, 0, 1]))
```

```
print(np.where(arr<0)[0],np.where(arr<0)[1])
```

```
[0 0 1 2] [0 1 0 1]
```

Boolean Indexing

```
arr[arr<0 | arr>1.58]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [173], in <cell line: 1>()
----> 1 arr[arr<0 | arr>1.58]

TypeError: ufunc 'bitwise_or' not supported for the input types, and the inputs could not be
pported types according to the casting rule ''safe''
```

```
arr[(arr<0)|(arr>1.58)]
```

```
array([-0.35777573, -0.01681105,  1.59329057, -0.50261512,  1.93291742,
       -0.98512758])
```

```
arr[(arr<0) or (arr>1.58)]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [177], in <cell line: 1>()
----> 1 arr[(arr<0) or (arr>1.58)]

ValueError: The truth value of an array with more than one element is ambiguous. Use a.any()
```

Reading/Writing Data

pandas.DataFrame.to_csv

```
DataFrame.to_csv(path_or_buf=None, sep=',', na_rep='', float_format=None,
columns=None, header=True, index=True, index_label=None, mode='w',
encoding=None, compression='infer', quoting=None, quotechar='"',
lineterminator=None, chunksize=None, date_format=None, doublequote=True,
escapechar=None, decimal='.', errors='strict', storage_options=None)
```

Write object to a comma-separated values (csv) file. [source]

pandas.read_csv

```
pandas.read_csv(filepath_or_buffer, *, sep=_NoDefault.no_default,
delimiter=None, header='infer', names=_NoDefault.no_default,
index_col=None, usecols=None, squeeze=None, prefix=_NoDefault.no_default,
mangle_dupe_cols=True, dtype=None, engine=None, converters=None,
true_values=None, false_values=None, skipinitialspace=False,
skiprows=None, skipfooter=0, nrows=None, na_values=None,
keep_default_na=True, na_filter=True, verbose=False,
skip_blank_lines=True, parse_dates=None, infer_datetime_format=False,
keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True,
```

pandas.read_excel

```
pandas.read_excel(io, sheet_name=0, *, header=0, names=None,
index_col=None, usecols=None, squeeze=None, dtype=None, engine=None,
converters=None, true_values=None, false_values=None, skiprows=None,
nrows=None, na_values=None, keep_default_na=True, na_filter=True,
verbose=False, parse_dates=False, date_parser=None, thousands=None,
decimal='.', comment=None, skipfooter=0, convert_float=None,
mangle_dupe_cols=True, storage_options=None) #
```

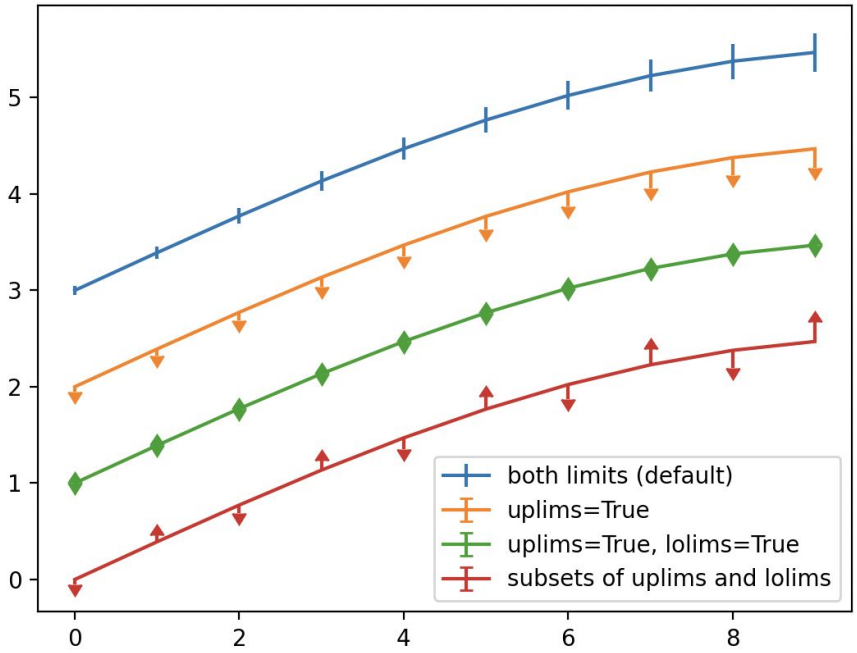
```
f = open("demofile.txt", "r")
print(f.read())
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [180], in <cell line: 1>()
----> 1 f = open("demofile.txt", "r")
      2 print(f.read())

FileNotFoundError: [Errno 2] No such file or directory: 'demofile.txt'
```

The first argument is a string containing the filename. The second argument is another string containing a few characters describing the way in which the file will be used. *mode* can be `'r'` when the file will only be read, `'w'` for only writing (an existing file with the same name will be erased), and `'a'` opens the file for appending; any data written to the file is automatically added to the end. `'r+'` opens the file for both reading and writing. The *mode* argument is optional; `'r'` will be assumed if it's omitted.

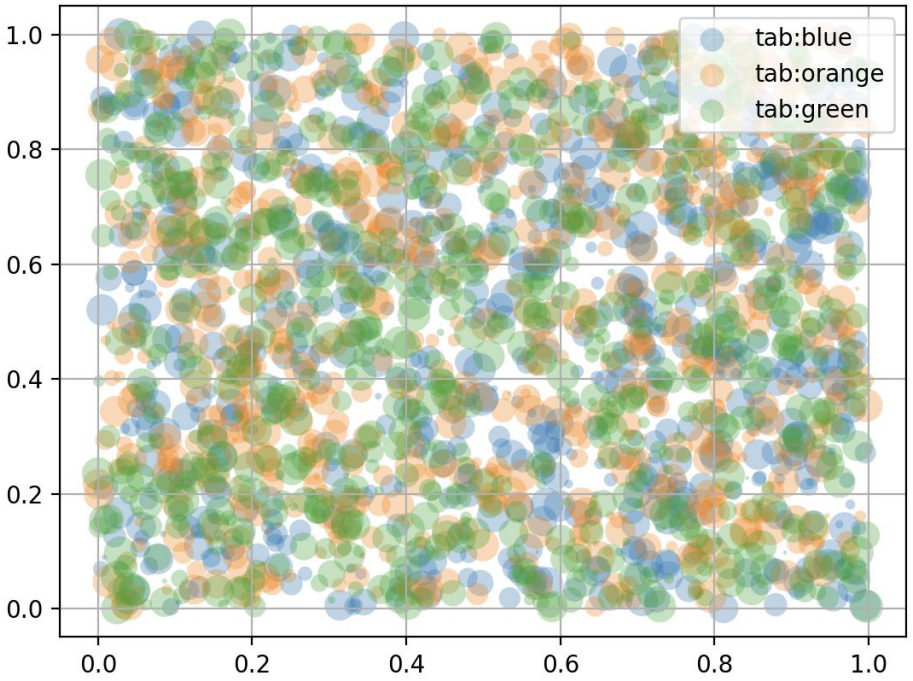
Plotting (Matplotlib)

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


fig = plt.figure()
x = np.arange(10)
y = 2.5 * np.sin(x / 20 * np.pi)
yerr = np.linspace(0.05, 0.2, 10)

plt.errorbar(x, y + 3, yerr=yerr, label='both limits (default)')

plt.errorbar(x, y + 2, yerr=yerr, uplims=True, label='uplims=True')

plt.errorbar(x, y + 1, yerr=yerr, uplims=True, lolims=True,
             label='uplims=True, lolims=True')

upperlimits = [True, False] * 5
lowerlimits = [False, True] * 5
plt.errorbar(x, y, yerr=yerr, uplims=upperlimits, lolims=lowerlimits,
             label='subsets of uplims and lolims')

plt.legend(loc='lower right')
```



Plotting (Matplotlib)

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

np.random.seed(19680801)


fig, ax = plt.subplots()
for color in ['tab:blue', 'tab:orange', 'tab:green']:
    n = 750
    x, y = np.random.rand(2, n)
    scale = 200.0 * np.random.rand(n)
    ax.scatter(x, y, c=color, s=scale, label=color,
               alpha=0.3, edgecolors='none')

ax.legend()
ax.grid(True)

plt.show()
```



Best Practices

- PEP 8 style guidelines: https://peps.python.org/pep-0008/
- Write frequent comments to explain what your code is doing
- Readability is the most important thing
  - Don’t write overly complex code just to save a line
  - Be consistent and clean
- Always try to use functions from libraries before defining your own
  - Published libraries have highly optimized code
- Google and StackOverflow are your best friends
  - I’ll try trial and error for 10-30 minutes then start the Googling
- https://peps.python.org/pep-0020/