

Tools for Data Analysis

Welcome to class!

Introductions

Dusty White, PhD (Economics)

University of Nebraska at Omaha

Introductions

Data is my passion, and I use combined with Economics to

- Study risky/dangerous behaviors
- Explore the relationship between remote work and pay
- Learn about strategic choices in sports

What do YOU want to get out of this class?

MyCourses

A quick tour

Github

All the other files (the majority of the content) -

<https://github.com/dustywhite7/pythonMikkeli>

PyCharm Edu

Your workspace. We will learn more as we go through class!

- Install
- Load the [zip file](#) into PyCharm as a course
 - I'll show you how
- Read the lessons, and do the practice problems!

OR Google Colab

Your alternate workspace (although honestly I think you might all prefer it...).

- Install
- Download the notebooks from [github](#) and upload into Google Drive, so that you can launch them in Colab
 - I'll show you how
- Same lessons, same practice problems!

Let's get started!

Introducing... Python!

- A dynamically typed language
- High-level
- Widely adopted in data science
- **General-purpose** language!
 - This means that we can use it for anything, not just for data analysis
- Emphasizes readability (you'll see what I mean)

Getting Started in Python

- Open PyCharm or Colab, and let's explore
 - PyCharm is called an IDE, or Integrated Development Environment, and contains all the tools we will need to work with Python (and to help this class go smoothly!)
 - Colab is a notebook environment, and is very popular for preliminary analysis and for use among data scientists.
- Let's write some Python!

A simple program in Python

Copy this program into your Python interpreter (follow my lead):

```
import numpy as np

def manhattanDistance(coord1, coord2):
    dist = 0
    errorstring = "Coordinate dimension mismatch."
    if len(coord1)==len(coord2):
        for i in range(len(coord1)):
            dist+=np.abs(coord1[i]-coord2[i])
        return dist
    else:
        raise RuntimeError(errorstring)
```

A simple program in Python - Explained

```
import numpy as np
```

`import` statements allow us to use pre-written (and typically optimized) code within our own programs

- `numpy` itself is an excellent mathematics library (NUM-eric PY-thon)
- imported libraries are often written in languages like C++ and Fortran, giving a tremendous speed advantage, as well!

A simple program in Python - Explained

```
def manhattanDistance(coord1, coord2):  
    ...  
    return dist
```

Using the `def` keyword allows us to define **functions**, or reusable bits of code that perform some specific task.

Functions accept arguments, and can be made to **return** values, as well.

A simple program in Python - Explained

```
if len(coord1)==len(coord2):  
    ...  
else:  
    ...
```

We can easily incorporate different kinds of conditions into our code using `if` statements. Here, we test for equality between two values and condition our response on the result of that test.

A simple program in Python - Explained

```
for i in range(len(coord1)):  
    ...
```

For loops allow us to repeat code multiple times with minor variations, so that we can reduce the amount of code we need to write.

A simple program

Every piece of code that we write will contain **logical statements** that allow the computer to perform tasks that we describe

- **The computer/Python will only be as careful as our code!**

Core Data Types in Python

Core types are the base types that everything else in Python will be built upon:

1. Numbers, Strings, Booleans, None
2. Lists, Dictionaries, Tuples, Sets
3. Functions, Modules, Classes

Numbers

Common

1. Integers: `int()`

2. Floating-point numbers: `float()`

Not so common

3. Complex numbers

4. Irrational numbers

Numbers

Numbers support basic arithmetic like we are familiar with:

- Addition and subtraction: `15+3` , `0-4`
- Multiplication and division: `2*4` , `3/5`
- Exponentiation: `2**4` denotes 2^4

We will also be able to import greater functionality from modules like `numpy` .

Strings

Strings are collections of characters with defined positions. Strings are also **immutable**, meaning that they cannot be modified, only replaced.

```
>>> myStr = 'DataScience!'
>>> len(myStr)
12
>>> myStr[0] # Using index values to select elements
'D'
```

Note: the first character in the string has position 0!

Strings

We can access elements of strings using index values beginning at 0, or we can access them by giving negative index values to indicate that we are counting from the end of the string to the front. An index of `-1` refers to the last element in the string.

```
>>> myStr = 'DataScience!'
>>> myStr[-1]
'!'
>>> myStr[-12]
'D'
```

Strings

We can **slice** a string, selecting a series of elements from within the string together.

```
>>> myStr = 'DataScience!'
>>> myStr[4:11]
'Science'
>>> myStr[4:11:2] # Only taking every other character
'Sine'           # 'step size of two'
```

We can also **concatenate** strings:

```
>>> myStr + 'YESSSS'
'DataScience!YESSSS'
```

Booleans

Booleans are data types that only permit storage of a binary value:

```
if lightsOff==True:  
    ...
```

The two boolean values are `True` and `False` (case sensitive).

```
>>> 3==(2+1)  
True  
>>> 3==2  
False
```


None

Python also has a `None` type that is frequently used to initialize objects. It can also be used to serve functions like determining whether or not information has been received

```
data = None
if data==None:
    raise RuntimeError('No data yet!')
else:
    ...
```

Lists

Like strings, lists contain multiple elements. Unlike strings, these can be any type of data. Lists can also be modified in place (**mutable**).

```
>>> myList = [2, 3, 4, 5]
>>> myList[-2]
4
>>> myList[-2] = 10
>>> myList[-2]
10
```

Lists

Lists can be **iterated** on:

```
>>> myList = [2, 3, 4, 5]
>>> for i in myList:
...     print(i**2)
4
9
16
25
```

They can be appended to:

```
>>> myList.append(6)
>>> myList
[2, 3, 4, 5, 6]
```

Lists

Lists can be "popped":

```
>>> myList = [2, 3, 4, 5]
>>> myList.pop()
5
```

Lists can be sorted:

```
myList.sort()
```

Or reversed:

```
myList.reverse()
```

Lists

Lists can also have lists as elements, and are then referred to as "a list of lists"

```
>>> listOfLists = [[2,3,4,5],[6,7,8,9]]  
>>> listOfLists  
[[2, 3, 4, 5], [6, 7, 8, 9]]
```

We can embed lists infinitely deep (list of lists of lists...), allowing us to create n -dimensional objects

- This becomes especially helpful when doing matrix computations, or in more advanced machine learning techniques

Tuples

Tuples are **immutable** lists. They cannot be modified in place, and are useful when you don't want to accidentally change any values.

```
>>> myTuple = (2, 3, 4, 5)
>>> myTuple[0]=10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Dictionaries

While strings, lists and tuples have specific orders, dictionaries approach the organization of data differently, using a `key:pair` combination to store data that can be found using the index provided by the programmer to the dictionary.

```
>>> myDict = {"first": "Dusty", "last": "White"}
>>> myDict['first']
'Dusty'
>>> myDict[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
```

Dictionaries

Like lists, dictionaries can be nested, iterated and are mutable.

```
>>> myDict = {"first": "Dusty", "last": "White",  
...          "hobbies": {1: "Checkers", 2: "Food"}}  
>>> myDict["hobbies"]  
{1: 'Checkers', 2: 'Food'}  
>>> myDict["hobbies"][1]  
'Checkers'  
>>> myDict["hobbies"][1] = "Sleeping"  
>>> myDict["hobbies"][1]  
'Sleeping'
```


Dictionaries

Two examples of iterating on a dictionary:

```
>>> for i in myDict:
...     print(i)
first
last
hobbies
```

```
>>> for i in myDict:
...     print(myDict[i])
Dusty
White
{1: 'Sleeping', 2: 'Food'}
```

Modules

Modules are pre-written code that can be imported to make your life easier.

```
import numpy as np
```

```
np.random.random(10) # Would generate an array of 10  
# random numbers – EASY!
```

In this case, the module is `numpy`, a numeric library already mentioned.

What if I can't remember all this?

DON'T PANIC!

This is a LOT of information! Fortunately, we have

DOCUMENTATION to help us make sure that we are doing the right thing.

Reading Documentation

To get started, let's look at the [Numpy Random Sampling Documentation](#)

Keep in mind, [StackOverflow](#) is a great website to help us figure out what to do when we have an error.

Documentation Matters!

Learning to read documentation is a critical component of becoming a programmer, or using programming for pretty much any purpose.

- Take your time
- Follow this [link](#) (Google is your friend!)
- REMEMBER: **Don't Panic!**

For Lab Today

Complete the **Solve it** exercises in PyCharm/Google Colab!

- Experiment! The more you try different things, the quicker you will learn.
- You WILL break things!
- You CANNOT learn to program by just attending lecture!

Submit a screenshot of the completed tasks verified in PyCharm, or the notebook if you complete in Google Colab (let's walk through how...)