ECON 8320

Tools for Data Analysis

Instructor:

Dustin White (he/him/his)
Mammel Hall 332M

Office Hours:

By appointment

Contact Info:

drwhite@unomaha.edu

Quick Note

This class does not require that you **already** know how to program, but you will know how by the end of the term. The class was actually designed **because** I do not expect that you already know how to program.

- Spend time outside class practicing (by time I mean hours, and not just a few)

Quick Note

Your ability to use code to solve problems will be the basis for your grade in this course, so if you cannot commit the time to practice coding, you are not likely to pass this class.

You wouldn't study French and not set aside time to practice. The same applies here!

Grade Details

Score	Grade	Score	Grade
>94%	A	72.5-77.4	C
90-93.9	A-	70-72.4	C-
87.5-89.9	B+	62.5-69.9	D
82.5-87.4	B	60-62.5	D-
80-82.4	B-	<60	F
77.5-79.9	C+		

Grade Details

Assignment	Percent of Grade
Lab Work/Homework	50%
Final Project	30%
Participation	20%

Note: No matter what Codio says an assignment is worth, all homework assignments will have equal impact on your final grade

My Expectations

- You will be expected to learn to program during this course
- Take charge of your assignments; they will be open-ended, and will often require **significant** effort

Expectations of Me

- I will work through examples of code in class
- I will be available to help you with assignments
- I will revise the course material as needed to suit your interests

Week 1 - Data Types and Documentation

Introducing... Python!

- A dynamically typed language
- High-level
- Widely adopted in data analysis
- **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 the Codio IDE from the Canvas home page
  - Codio allows us to execute code on a remote server, processing the code on uniform machines
- Let's write some Python!

A simple program in Python

```
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

```
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

```
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

```
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

```
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.

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. Rational 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` is 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
```

Note: ALL OF PYTHON is case sensitive!!

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:value` pair 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
```

This prints the `keys`

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

This prints the `values` assigned to the keys

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.

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.

Using the help tools

We can quickly see information about a Python object by using the `help()` function:

```
help(np.random) # or help(np.random.random)
```

Documentation

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!)
- Don't Panic!

Lab Time!