1-Intro-to-Python

August 25, 2016

1 Introduction to Python

For those of you that know python, today's class aims to refresh your memory.

For those of you that don't know python – but do know programming – this lecture is to give you an idea of how python is similar to, and different from, your favorite programming language.

1.1 Environment

There are four ways to run python code:

- put your code in a file (say program.py) and run `python program.py`
- This is least desirable when you are first writing your code
- Later, once your code is debugged, this is the way to go

- type your code into the python interpreter
- This allows you to interact with the interpreter and fix mistakes as they happen
- However you have to type everything by hand

- type, cut/paste, or run your code in `ipython`
- This is a good method
- Allows you to cut/paste from a file you are working on

- run `ipython` in a browser, called `jupyter notebook`
- This is even better
- All the advantages of `ipython` plus interleaved documentation and graphical output
- That is what these slides are in

1.2 Python 2 vs Python 3

There are two versions of Python. I am using Python 3 and that's what I recommend. Here are the main differences:

In Python 2, integer division does *not* return a floating point value:

3/5 = 1

In Python 3, integer division does return a floating point value:

```
In [2]: 3/5

Out[2]: 0.6
```

In Python 2, the print statement does not use parentheses:
print 'a string'
In Python 3, print is a function and it uses parentheses:

```
In [3]: print('a string')

a string
```

1.3 Functions and Methods

Function calls use standard syntax:

```
func(argument1, argument2)
```

However most things you interact with in python are **objects** and they have **methods**. A method is a function that operates on an object:

```
object.method(argument1, argument2)
```

Note that the method might modify the object, or it might return a new, different object. You just have to know the method and keep track of what it does.

```
In [4]: number_list = [1, 3, 5, 7]
        number_list.append(8)

In [5]: number_list

Out[5]: [1, 3, 5, 7, 8]

In [6]: string = 'This is a string'
        string.split()

Out[6]: ['This', 'is', 'a', 'string']

In [7]: string

Out[7]: 'This is a string'
```

1.4 Printing

From the interactive python environment:

```
In [8]: print("Hello World")

Hello World
```

From a file:

```
In [9]: #!/usr/bin/env python

        print("Hello World!")

Hello World!
```

1.5 Data types

Basic data types:

1. Strings
2. Integers
3. Floats
4. Booleans

These are all objects in Python.

```
In [10]: a = "apple"
         type(a)

Out[10]: str

In [11]: b = 3
         type(b)

Out[11]: int

In [12]: c = 3.2
         type(c)

Out[12]: float

In [13]: d = True
         type(d)

Out[13]: bool
```

Python **doesn't require explicitly declared variable types** like C and other languages. Python is dynamically typed.

```
In [14]: myVar = 'I am a string'
         print(myVar)
         myVar = 2.3
         print(myVar)

I am a string
2.3
```

1.6 Strings

String manipulation will be very important for many of the tasks we will do. Here are some important string operations.

A string uses either single quotes or double quotes. Pick one option and be consistent.

```
In [15]: 'This is a string'

Out[15]: 'This is a string'

In [16]: "This is also a string"

Out[16]: 'This is also a string'
```

The '+' operator concatenates strings.

```
In [17]: a = "Hello"
         b = " World"
         a + b

Out[17]: 'Hello World'
```

Portions of strings are manipulated using indexing (which python calls 'slicing').

```
In [18]: a = "World"

         a[0]

Out[18]: 'W'

In [19]: a[-1]

Out[19]: 'd'

In [20]: "World"[0:4]

Out[20]: 'Worl'

In [21]: a[::-1]

Out[21]: 'dlroW'
```

Some important string functions:

```
In [22]: a = "Hello World"
         "-".join(a)

Out[22]: 'H-e-l-l-o- -W-o-r-l-d'

In [23]: a.startswith("Wo")

Out[23]: False
```

```
In [24]: a.endswith("rld")

Out[24]: True

In [25]: a.replace("o","0").replace("d","[]").replace("l","1")

Out[25]: 'He110 W0r1[]'

In [26]: a.split()

Out[26]: ['Hello', 'World']

In [27]: a.split('o')

Out[27]: ['Hell', ' W', 'rld']
```

Strings are an example of an **immutable** data type. Once you instantiate a string you cannot change any characters in it's set.

```
In [28]: string = "string"
         string[-1] = "y"  #Here we attempt to assign the last character in the sbt


        ---------------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-28-cc6a83ef485f> in <module>()
          1 string = "string"
    ----> 2 string[-1] = "y"  #Here we attempt to assign the last character in the


        TypeError: 'str' object does not support item assignment
```

To create a string with embedded objects use the `.format()` method:

```
In [29]: course_name = 'CS505'
         enrollment = 75
         percent_full = 100.0
         'The course {} has an enrollment of {} and is {} percent full.'.format(
             course_name,enrollment,percent_full)

Out[29]: 'The course CS505 has an enrollment of 75 and is 100.0 percent full.'
```

1.7 Code Structure

Python uses indents and whitespace to group statements together. To write a short loop in C, you might use:

`c for (i = 0, i < 5, i++){ printf("Hi! \n"); }`

Python does not use curly braces like C, so the same program as above is written in Python as follows:

```
In [30]: for i in range(5):
             print("Hi")
```

```
Hi
Hi
Hi
Hi
Hi
```

If you have nested for-loops, there is a further indent for the inner loop.

```
In [31]: for i in range(3):
             for j in range(3):
                 print(i, j)

             print("This statement is within the i-loop, but not the j-loop")
```

```
0 0
0 1
0 2
This statement is within the i-loop, but not the j-loop
1 0
1 1
1 2
This statement is within the i-loop, but not the j-loop
2 0
2 1
2 2
This statement is within the i-loop, but not the j-loop
```

1.8 File I/O

`open()` and `close()` are used to access files. However if you use the `with` statement the file close is automatically done for you.

You should use `with`.

```
In [32]: with open("example.txt", "w") as f:
             f.write("Hello World! \n")
             f.write("How are you? \n")
             f.write("I'm fine. OK.\n")
```

Reading from a file:

```
In [33]: with open("example.txt", "r") as f:
             data = f.readlines()
             for line in data:
                 words = line.split()
                 print(words)

['Hello', 'World!']
['How', 'are', 'you?']
["I'm", 'fine.', 'OK.']
```

Here is an example of counting the number of lines and words in a file:

```
In [34]: lines = 0
         words = 0
         the_file = "example.txt"

         with open(the_file, 'r') as f:
             for line in f:
                 lines += 1
                 words += len(line.split())
         print("There are {} lines and {} words in the %s file.".format(
                 lines, words, the_file))

There are 3 lines and 8 words in the %s file.
```

1.9 Lists, Tuples, Sets and Dictionaries

Number and strings alone are not enough! We need data types that can hold multiple values.

1.9.1 Lists:

A list is a collection of data items, which can be of differing types.
Here is an empty list:

```
In [35]: groceries = []
```

A list is **mutable**, meaning that it can be altered.
Adding to the list:

```
In [36]: groceries.append("oranges")
         groceries.append("meat")
         groceries.append("asparagus")
         groceries

Out[36]: ['oranges', 'meat', 'asparagus']
```

Accessing list items by index:

```
In [37]: groceries[0]

Out[37]: 'oranges'

In [38]: groceries[2]

Out[38]: 'asparagus'

In [39]: len(groceries)

Out[39]: 3
```

Sort the items in the list:

```
In [40]: groceries.sort()
         groceries

Out[40]: ['asparagus', 'meat', 'oranges']
```

Remove an item from a list:

```
In [41]: groceries.remove('asparagus')
         groceries

Out[41]: ['meat', 'oranges']

In [42]: groceries[0] = 'peanut butter'
         groceries

Out[42]: ['peanut butter', 'oranges']
```

A **list comprehension** makes a new list from an old list. It is incredibly useful (learn how to use it!)

```
In [43]: groceries = ['asparagus', 'meat', 'oranges']
         veggie = [x for x in groceries if x is not "meat"]
         veggie

Out[43]: ['asparagus', 'oranges']
```

This is the same as:

```
In [44]: newlist = []
         for x in groceries:
             if x is not 'meat':
                 newlist.append(x)
         newlist

Out[44]: ['asparagus', 'oranges']
```

Notice how much more concise and clear the list comprehension is. It's more efficient too.

1.9.2 Sets:

A set is a collecton of items that cannot contain duplicates. Sets handle operations like sets in mathematics.

```
In [45]: numbers = range(10)
         numbers = set(numbers)

         evens = {0, 2, 4, 6, 8}

         # Use difference to find the odds
         odds = numbers - evens
         odds

Out[45]: {1, 3, 5, 7, 9}
```

Sets also support the use of union (|), and intersection (&)

1.9.3 Dictionaries:

A dictionary is a map of keys to values. **Keys must be unique**.

```
In [46]: # A simple dictionary
         simple_dic = {'cs505': 'data-mining tools'}

         # Access by key
         print simple_dic['cs591']


           File "<ipython-input-46-bb1d697223e1>", line 5
        print simple_dic['cs591']
                       ^
    SyntaxError: Missing parentheses in call to 'print'



In [ ]: # A longer dictionary
        classes = {
            'cs591': 'data-mining tools',
            'cs565': 'data-mining algorithms'
        }

        # Check if item is in dictionary
        print 'cs530' in classes

        # Add new item
        classes['cs530'] = 'algorithms'
        print classes['cs530']
```

```
         # Print just the keys
         print classes.keys()

         # Print just the values
         print classes.values()

         # Print the items in the dictionary
         print classes.items()

         # Print dictionary pairs another way
         for key, value in classes.items():
             print key, value
```

```
In [ ]: # Complex Data structures
        # Dictionaries inside a dictionary!

        professors = {
            "prof1": {
                "name": "Evimaria Terzi",
                "department": "Computer Science",
                "research interests": ["algorithms", "data mining", "machine learni
            },
            "prof2": {
                "name": "Chris Dellarocas",
                "department": "Management",
                "interests": ["market analysis", "data mining", "computational educ
            }
        }

        for prof in professors:
            print professors[prof]["name"]
```

1.9.4 Tuples:

Tuples are an **immutable** type. Like strings, once you create them, you cannot change them.
Because they are immutabile you can use them as keys in dictionaries.
However, they are similar to lists in that they are a collection of data and that data can be of differing types.

```
In [ ]: # Tuple grocery list

        groceries = ('orange', 'meat', 'asparangus', 2.5, True)

        print groceries

        #print groceries[2]

        #groceries[2] = 'milk'
```

1.10 Creating Functions

```
In [ ]: def displayperson(name,age):
            print("My name is {} and I am {} years old.".format(name,age))
            return

        displayperson("Larry","40")
```

1.11 Libraries

Python is a high-level open-source language. But the *Python world* is inhabited by many packages or libraries that provide useful things like array operations, plotting functions, and much more. We can (and we should) import libraries of functions to expand the capabilities of Python in our programs.

```
In [ ]: import random
        myList = [2, 109, False, 10, "data", 482, "mining"]
        random.choice(myList)
```

```
In [ ]: from random import shuffle
        x = [[i] for i in range(10)]
        shuffle(x)
        print x
```

1.12 APIs

```
In [ ]: # Getting data from an API

        import requests

        width = '200'
        height = '300'
        response = requests.get('http://loremflickr.com/' + width + '/' + height)

        print response

        with open('img.jpg', 'wb') as f:
            f.write(response.content)
```

```
In [ ]: from IPython.display import Image
        Image(filename="img.jpg")
```