Programming With Data (CM2015)

Course Notes

Felipe Balbi

October 19, 2020

Contents

Week 1

Key Concepts

- Set up and run Jupyter Notebook on a Windows, Mac or Linux operating system.
- Write and explain simple Python programs using variables and mathematical operators.

1.01 Welcome to the course

Python was chosen for this course due to its simplicity, ease of use and collection of freely available tools.

A print statement in python is simply:

```
1 print("Welcome to Python!")
```

We can also write our code in a file ending with the extension `.py` and run it through the Python interpreter, like so:

```
1 $ python my_file.py
```

During the course we will rely heavily on Jupyter Notebooks. This will give us a nice interface to work with.

1.06 Introduction to development environments and Python

There are many Development Environments available. Only **emacs** is worth your time. Some folks will swear that *VI* is great, however, remember that *VI VI VI* is the number of the beast (trollface).

Jokes aside, a development environment is a very personal choice. One can visit World Class Text Editor section for a small list of what's available.

1.07 Getting to grips with Python

Read the following introductory reading, which will help you get to grips with Python:

McKinney, W. Python for data analysis: data wrangling with Pandas, NumPy, and IPython. (Sebastopol, CA: O'Reilly, 2017) 2nd edition, Chapter 1 Preliminaries and Chapter 2 Python Language Basics, IPython, and Jupyter Notebooks, pp.1–46.

Available here.

1.103 Jupyter code cells

When we create a new Notebook in Jupyter, it comes with what are referred to as *cells*. Cells can be defined in terms of what they can do and we can change their types too.

Code cells can be used to write code. In our case using Python.

Markdown cells are used to write Markdown, which will serve as documentation of textual input.

Raw NBConvert probably won't be used and won't be discussed.

1.105 Jupyter Notebook basics

Click on the links to below to read about Jupyter Notebook basics and using markdown cells in Jupyter:

- Jupyter Notebook basics Using markdown cells in Jupyter

1.106 Using Python as a Calculator

Click on the link below to read about using Python as a calculator:

- Using Python as a Calculator

Week 3

Key Concepts

- Import Python, NumPy and SciPy modules, and use them to compute basic statistics.
- Use logic and iteration to fill arrays with data, sum array elements and locate array elements with certain characteristics.
- Identify and use correct syntax and explain the purpose of built-in variable types int, float and list.

2.01 Becoming familiar with Python

Python is very strict on indentation. Code blocks are separated by indentation. For example, the following piece of code:

```
def my_function():
    print("Output")

if (x > 10):
    print(x)
```

Is not the same as this one:

```
def my_function():
    print("Output")

    if (x > 10):
        print(x)
```

Python has a set of familiar operators:

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Floor division
**	Exponentiation
%	Modulus

Python also has variables and types. Variables can be virtually any name, except for a few reserved words, such as `and`, `not`, `with`, `else`, `false`, and a few others.

We initialize a variable by assigning a value to it:

```
string_a = "foo"
int_x = 100
truthy = True
falsey = False
```

Regarding types, Python is dynamically typed which means a type is inferred at the time a value is assigned to a variable. Python is unable to implicitly coerce a type mutation, which means Python is a strongly typed language.

Python has a rich collection of built-in data structures. Among them are:

Lists Mutable sequence of comma separated values

Dictionaries Unordered, unique key-value pairs

Tuples Immutable sequence, comma separated

Sets Unordered collections, unique values

Functions are designed around reuse. The basic function definition looks like the one below:

```
def function_name(param1, param2, param3):
    if param1 > 10:
        print("Large value")

    return param2 + param3
```

Parameters to functions are optional.

Python also has the concept of Modules. They are a way to package functionality into a block of code that can be imported elsewhere. Python has a rich set of open source modules ready for use in different applications.

We can import parts of a module, or the entire module. It's convention to use the `from` keyword for the current namespace. A few examples:

```
import random
from fibo import fib
import numpy as np
```

When processing data, we can pull data from a database, scrape the web, or use files in local storage. Data can be stored either in binary format or some form of text. Data representation is important.

2.03 Built-in types in Python

Read through the following literature on built-in types in Python:

- Built-in types

2.05 Mapping operators to functions

Read through the following documentation.

- Python 10.3.1 Mapping operators to functions, Python language documentation 10.3. operator – Standard operators as functions.

2.104 Introduction to conditional logic

Python supports our regular `if`, `else`, and `elif` constructs. It also has the expected set of comparison operators `>`, `>=`, `<`, `<=`, `==`, and so on.

2.105 Functions and reuse

We define a function to make it easier to reuse code by changing a few parameters. Once defined, we can call the same function as many times as we want. To call a function, we input its name followed by parenthesis.

```
def my_function(eat):
    print(eat + " for dinner")

def double(x):
    return 2 * x

my_function("Carrots")
my_function("Cheese")
my_function("Bread")

print(double(2))
print(double(3))
print(double(4))
```

2.201 Conditions and logic

We combine logical statements with the connectives `or` and `and`.

```
a = 100
b = 50
c = 1000

if a > b or a > c:
    print("a is greater than b or c")
if a > b and a > c:
    print("a is greater than b and c")
```

2.202 More conditions

In Python, there are quite a few ways to deal with conditions. We might use certain conditions in cases where:

- we want to identify a time when a condition is met
- we want to iterate through a list of items
- we want to do something until something else happens, at which point we stop.

Click on the links below and read through the documentation that covers these conditional situations: Python 4.1, 4.2, 8.2 – 'if' 'for' 'while'.

- Python 4.1. if statements, Python language documentation – 4. Built-in types.
- Python 8.2. The while statement, Python language documentation – 8. Compound statements.
- Python 4.2. for statements, Python language documentation – 4. Built-in types.

Week 4

Key Concepts

- Import Python, NumPy and SciPy modules, and use them to compute basic statistics.
- Use logic and iteration to fill arrays with data, sum array elements and locate array elements with certain characteristics.
- Identify and use correct syntax and explain the purpose of built-in variable types int, float and list.

2.301 Introduction to lists

Lists are akin to *arrays* in other languages. It's a method for storing multiple values in a single variable. We can create a list by placing comma separate values inside square brackets `[]`. Like shown below:

```
1 car_emissions = [1.5, 1.26, 2.6]
2 print(car_emissions)
```

We can access individual elements of the list by indexing the variable. Note that indices start at 0, not 1. So the first element would be accessed like shown below:

```
1 print(car_emissions[0])
```

For other values, we just change the index. An interest peculiarity of python lists is that it works like a *ring*; that is to say that using negative indices allows us to walk backwards. In other words, the last element of a list is always at index `-1`:

```
1 print(car_emissions[-1])
```

We don't have to always index the list using a literal number. We can use a variable as well:

```
1 index = 1
2 print(car_emissions[index])
```

We can easily compute the length of a list using the `len()` method:

```
print(len(car_emissions))
```

Lists can also be indexed by ranges of numbers. If we want to print elements 0 through 2, we can use:

```
print(len(car_emissions[0:2]))
```

As we can see, the interval of the range is not inclusive of the final value. This means that the range `0:1` will only print the 0^{th} element.

2.302 Lists

Click on the link below to read through the documentation on lists:

- Python 4.6.4. Lists, Python language documentation – 4. Built-in types.

2.305 Loops and iteration

Iteration allows us to work through a series of objects. Python has different looping constructs to help us iterate through several objects. The first example is the `while` loop:

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

We can also use a `for` loop which works rather like `while` loop above:

```
for i in range(10):
    print(i)
    i += 1
```

In Python, there are other ways to work through a list. For example:

```
teams = ["red", "blue", "green"]

for x in teams:
    print(x)
```

2.401 Libraries and dependencies

Python has a rich set of libraries available for our use. But how do we use them? We use the `import` keyword. To demonstrate, we import `NumPy` and `scipy`, two important libraries for data crunching.

After importing the library we can start using functions from the library, as shown below:

```
1 import numpy as np
2 from scipy import stats
3
4 x = np.array([1, 2, 3, 4, 5, 6])
5
6 np.mean(x)
7 stats.describe(x)
```