American Concrete Institute (ACI)-UIUC Student Chapter

Python Workshop (Basic and Advanced)

Sushobhan Sen

Doctoral Candidate

April 6, 2019

Urbana, IL

Contents

1 Introduction

1.1 Why Python?

The Python programming language was conceived in the 1980s and the first implementation was deployed in 1991 by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands. Python is designed to be a high-level, interpreted, general-purpose computer programming language with exception-handling and the ability to be extended by users. Crucially, Python is an open-source language, meaning anybody can see the code behind it. Python today is a very popular language is a broad variety of disciplines: machine learning, web development, Geographical Information Systems (GIS), engineering, database management, high performance computing, etc.

Python 2.0, which truly launched the popularity of the language, was released in 2000, with the last version in the series, Python 2.7, set to be phased out in 2020. The current recommended standard as of when I wrote this is Python 3.7.2, which is crucially **not backwards compatible** with Python 2.x. All subversions of Python 3.x are backwards compatible though, so feel free to update regularly.

(The above information was taken from Wikipedia)

The beauty of Python is its ease - the syntax is extremely simple as compared to many other popular programming languages. It is often said, not entirely without truth, that the difference between Python code and pseudocode is merely the indentation. Thus, the language is very easy to learn for programming novices. However, its ease should not be misconstrued to mean that it is only for easy or trivial applications: Python is an extremely powerful language with a wide variety of applications, it has extensive documentation, and its open-source nature ensures that new features are being developed continuously while bugs are also being fixed.

1.2 Learning Objectives

This workshop is broken up into two sessions:

1. A Basic Python session for beginners with zero knowledge of programming in any language

2. An Advanced Python session for those with prior knowledge of Python programming

At the end of the **Basic Python** session, participants will be able to:

1. List the types of Python variables and define them
2. Add control statements (`if-then-else`) and `for` loops to their program
3. Define and use functions
4. Define and use object oriented programming

At the end of the **Advanced Python** session, participants will be able to:

1. Use the `numpy` library to define and use matrices, and read and process data from files
2. Use the `pandas` library to read and process data
3. Use the `scipy` library to solve a system if linear equations and implement other useful scientific functions
4. Use the `matplotlib` library to create publication-quality plots

The lists above ware what I will try to accomplish in our two two-hour long sessions. Depending on how quickly the workshop goes, I may or may not be able to meet all the learning objectives. However, I will get you along far enough so that you can complete any remaining items on your own.

1.3 Installing Python

The easiest and **recommended** way to get all popular Python libraries and IDEs is by downloading and installing the latest Anaconda distribution for your computer. Make sure to download the latest version corresponding to Python 3.x.

Alternatively, you can install and use Python using the terminal:

- On Windows 10, activate Windows Subsystem for Linux

- On Mac OS or any UNIX-like OS (such as Linux), just run your favorite Terminal application

Then install Jupyter with `pip`. With any other version of Windows (which you should not be using for too long on your personal computers anyway for security reasons), Anaconda is your best option.

Once installed, you can now create a new notebook. If you installed Anaconda, open the Anaconda Prompt, navigate to your directory, and use the `jupyter notebook` command. If you choose to use the terminal, follow the same steps on the terminal. Both methods will launch Jupyter Notebooks in your browser. From there, you can create a new Notebook.

If you prefer not to install anything on your computer but would rather run Python remotely from your browser, you can use the Online IDE from repl.it. This doesn't always work very well though. Any other online IDE that you find should be OK too.

2 Variables in Python

2.1 Data Types

Python defines the following data types for variables:

Data Type	**Syntax**	**Description**	**Comments**
Integer	`x = 5`	Signed integer	Use `int` to typecast, if valid
Float	`x = 5.`	IEEE floating point number	Use `float` to typecast, if valid
Complex	`x = 3.1+4.6j`	Complex number	Use `complex` to typecast, if valid
String	`x = 'Hello World!'`	Strings are always enclosed within quotation marks	Use `str` to typecast, slicing operator valid
List	`x = [1, 2.2, 'otter']`	**Mutable** list of variables of any type	Use `list()` to typecast, if valid
Tuple	`x = (1, 2.2, 'otter')`	Immutable tuple of variables of any tupe	Use `tuple()` to typecast, if valid
Dictionary	`x = {'one':1, 'two':2}`	Key-value pairs	Use `get()` to get `value` from `key`, use `dict()` to typecast, if valid
Bool	`x = True`	Boolean value	Python uses keywords `True` and `False`, not 1 and 0

2.2 Creating a variable

Variables are usually given a name, which is any string of characters you use to call it. Any string of characters is a valid name, although there are a couple of rules to follow:

1. The name **cannot** contain spaces - consider using an underscore character or chamelCase instead for readability
2. The name cannot start with a number, but can contain a number anywhere
3. The name is case-sensitive, so `temp` and `Temp` are different variables
4. Python has a number of reserved keywords (see documentation, of just follow along and you'll get the hang of it), which cannot be used as names

Creating a variable is as simple as giving it a name, and assigning a value to it. For example, run this piece of code:

```
x = 1.0
y = 'Hello, World!'
z = (x==y)
print(type(x),type(y),type(z),z)
```

Here, we defined three variables: `x` of type `int`, `y` of type `str`, and `z` of type `bool`. Note that the `==` is a comparison operator that compares two values, while `=` is an assignment operator that assigns a value to a variable. The `print()` function, as the name suggests, prints the comma-separated inputs as a string , while the `type()` function returns the data type of the input. Note that every new line of code in Python starts on a new line, and **there is no end of line character** (such as a semi-colon in many languages).

2.3 Mutability

Some Python data types are **immutable**. This means that, once defined, ibjects of those types cannot be changed without creating a new object entirely. Most objects are immutable: `int, float, complex, str, bool, tuple` are immutable. If you try to change their values, you will either get an error or a new object containing the new value will be created. For example, run the following code, where `id()` is a function that returns the memory location of the object passed to it:

```
x = 1
print(id(x))
x = 2.0
print(id(x))
```

You'll see that the memory location of `x` has changed entirely, instead of just the value being changed. This is because `x` was defined as type `int`, which is immutable. Similarly, try changing the value inside a tuple and see what happens (note that `x[0]` is a way to reference the first element of `x` - **Python is zero-indexed**):

```
x = (1,2,3)
x[0] = 10
print(x)
```

Lists and dictionaries are mutable, which means their values can be changed at any time. Try running the following code and compare it to the last one:

```
x = [1,2,3]
print(id(x))
x[0] = 10
print(id(x))
print(x)
```

2.4 Operations on Variables

Python provides a number of operations that can be performed between variables:

- **Arithmetic operators:** + (add), - (subtract), * (multiply), / (divide), % (modulus or remainder), // (floor division), ** (exponent)
- **Comparison operators:** > (grater than), < (less than), == (equal to), != (not equal to), >= (greater than or equal to), <= (less than or equal to)
- **Logical operators:** `and`, `or`, and `not`

- **Bitwise operators:** & (bitwise AND), | (bitwise OR), ∼ (bitwise NOT), (bitwise XOR), >> (bitwise right shift), << (bitwise left shift)
- **Assignment operators:** A combination of the basic assignment operator (=) and an optional arithmetic or bitwise operator. For example, `x **= 5` is the same as `x = x**5`
- **Special operators:** `is` or `is not` check if two variables have the same memory address, `in` or `not in` check if a variable is in a sequences of variables

Operators are mostly self-explanatory, but their behavior can be different based on the data type of the input variables. Consider adding two integers, adding an integer to a float, adding two strings, and adding an integer to a string, all of which use the same + operator:

```
x = 2
print(x+2)
print(x+2.5)
print('Goodbye'+'Hello')
print(x+'Hello')
```

The first operation is as expected and returns an integer. In the second operation, `x` is first typecast to `float` *implicitly* (the program does it by itself) and then added to another float to return a float. In the third operation, two strings are simply concatenated together to return another string.

However, the last operation of adding an integer to a string returns an error, because Python is unable to figure out which variable to cast to which type. Of course, casting a string to an integer makes no sense, but Python doesn't even try, because it's unsure about what to do. You can help it by *explicitly* casting `x` from an integer to string, and then see what happens:

```
x = 2
print(str(x)+'Hello')
```

3 Control Statements

Control statements are blocks of code that are used to control the flow of the program.