Python Bootcamp

Preparation for BIOS 534 - January 2019

Preliminaries

What is this Bootcamp about?

This Python Bootcamp is a (very) quick and dirty introduction to Python programming for students enrolled in BIOS 534. It is intended for students with little or no exposure to Python programming, and seeks to get everyone up and running in a Python environment ahead of the programming projects to be assigned in class.

What is covered in this Bootcamp?

1. How to get Python, and which version to use
2. The Python programming environment, and IDEs
3. Basic coding practices in Python, including some Python-specific idioms
4. How to get additional help
5. Resources (recommended reading, websites, etc.)

What’s not covered in this BootCamp?

1. The Fundamentals of Computer Science and Computer Programming
2. Object-oriented Programming
3. Machine Learning algorithms (things covered in BIOS 534)
4. Esoteric features of Python not related to learning the basics
5. Operating Systems (e.g., various versions of Windows, Mac OS X, Linux)

Who am I?

Keven Haynes (“Keven” - neither “Doctor” nor “Professor”)

Manager of High Performance Computing (HPC) , at Rollins School of Public Health

M.S., Mathematics and Computer Science, Emory University

Email: khaynes@emory.edu

Who are you?

If you are here, you should be currently enrolled as a student in BIOS 534 or CS 534 this spring (2019). It is assumed that you have some computing experience, preferably programming in some “current” language, and possess reasonable facility in installing software on your own laptop or home computer.

If you have done extensive programming in Python or other languages before (e.g. Computer Science majors), this Bootcamp may be superfluous and boring to you. The Bootcamp is not required for successful completion of the course, and no grade or class credit is being granted for attending the Bootcamp.

About Python

Why | What | Which Python?

Python is a “high level” general-purpose programming language that focuses on code readability and simplicity.

Python is considered a “scripting language”.

Python is open source and officially supported in two versions: Python 2 (EOL in 2020) and Python 3. There are some significant (and incompatible) differences between the two versions. We’ll focus on Python 3.

Python is extensively used in (Data) Science, Education, and Systems and Web Programming. It differs from other languages such as R (primarily designed for statistics), but has been extended to include a lot of similar features.

Python is an “object oriented” language.

Getting Python

Unix-like operating systems (Mac OS X and Linux, etc.) usually come with a version of Python installed. Typically, it is an older version used in the management of the system.

Windows usually does not have a native version of Python installed.

Python.org is the canonical source of the Python code.

Anaconda Python (anaconda.com) is recommended:

1. It is still free
2. It is a very easy to install and does not require administrative access
3. I has a large number of Data Science packages and other features built in
4. You can run the same version everywhere and on all OSes

NB: It is **large (~5 GB)**. “Miniconda” is smaller.

Install Anaconda Python 3.7 version

1. Go to www.anaconda.com/download
2. Click on “Download” for Python 3.7 Version of your operating system
3. It should default to 64-bit - if not select it
4. Follow instructions for your operating system
5. Allow installer to set default path to Anaconda version of Python (but not as the ‘root’ or ‘admin’ user)

What comes with Anaconda?

1. A complete Python (3.7) distribution compiled for your OS
2. Many prebuilt (data-) scientific packages are included:

   https://docs.anaconda.com/anaconda/packages/py3.7_win-64/

3. iPython and Jupyter Notebooks (IDEs)

4. Anaconda Navigator and *conda* package management system

IDE’s
(Integrated Development Environments)

The Python Interpreter

- It comes standard with the Python distribution
- Simply type ‘python’ at the command line
- Allows for interactive Python programming
- It is pretty minimal, but lightweight

iPython - a better Python Interpreter

- Allows for command-completion and history scrolling
- Line-numbered history
- It has “magic” commands, such as %run (for running a local script from a file), %pwd, %ls, etc., which allow interaction with the local filesystem
- Introspection: a “?” after an object displays information about that object
- It is usually an add-on for most Python installations (i.e., not installed by default - Anaconda is an exception)

Jupyter Notebooks

- Notebooks are an interactive documents for code and text
- Data visualizations can be displayed in-line as well
- Computations run in a “kernel” (can be Python 3, Python 2, Julia, R, et al)
- It functions similarly to iPython for Python users
- Can export interactive sessions as human-readable documents
- Can be used to connect to kernels on remote servers

Python Basics

1. Basic Built-In Data Types

- Booleans (“True”, “False”)
- Integers (whole numbers, both positive and negative, including 0, e.g. “-16”, of arbitrary size)
- Floats (numbers with decimal points, e.g., “‘3.141”)
- Strings (sequences of text characters, e.g., “cheese”)

A note about Python variables

Python variables are names which point to values in memory. These values are objects of one of the aforementioned types.

Variable names may consist of:

- Lowercase letters
- Uppercase letters
- Numbers
- Underscores (“_”)

They may **NOT** begin with a number.

You may also not use **reserved words** for variable names.

Python Reserved Words

False	None	True	and	as	assert	break
class	continue	def	del	elif	else	except
finally	for	from	global	if	import	in
is	lambda	not	or	pass	raise	return
try	while	with	yield			

Variable Assignment

Variables are assigned values using the assignment operator, “=”.

“=” does NOT mean “equal” in Python. It means “variable on the left gets value on the right”.

“==” is the equality comparison operator:

```
if sky == “blue”:

    print(“Winter is over!”)
```

type() command

Use the **type()** command to determine the type of object a variable points to. This is used often.

You can do type conversions (called ‘type casting’ sometimes in other languages) with **int()**, **float()**, and **bool().**

You can also cast from numeric to string with **str().** The reverse operation only works if it makes sense (e.g., str(9) returns ‘9’, int(“9”) produces 9, but int(“horse”) would produce a value error).

Python strings

You make a python string by enclosing characters in either pairs of single quotes (‘horse’) or double quotes (“horse”), or even *triple* quotes (e.g. ‘’’horse’’’ or “””horse”””).

Triple quotes allow you to create multiline strings:

```
‘’’ I think that I shall never see

    A poem as lovely as a tree‘’’
```

Working with strings

- You can combine strings with “+” operator
- Duplicate strings with “*” operator
- Escape characters with “\” (e.g., the newline ‘**\n**’, or tab ‘**\t**’)
- Take string slices
- Use string object “methods”: **replace, split, join, len, upper, lower**

Remember: Strings are *immutable*. Items in a string cannot be replaced without creating a new string.

2. Built-In Data Structures

- Lists
- Tuples
- Dictionaries
- Sets

Ordered Collections: Lists and Tuples

Lists: sequences of objects indexed by their integer position

Tuples: also sequences of objects indexed by their integer position

Difference? *Mutability*

Lists are *mutable* - elements in a list can be inserted, deleted, reordered, etc.

Tuples are *immutable*. Once created they can't be changed. More on why that is useful later...

About Lists

Lists are good for collections that need to be kept in order. Unlike strings, they can be made up of any Python object, including other Lists or other data structures. Lists can be created by assigning a variable the empty list '[ ]', comma-separated items enclosed within brackets, or using the list() function. Elements are accessed by their index or offset:

```
In [18]: fruit = [ "apple", 'pear', 'watermelon', 'kiwi' ]

In [19]: fruit[1]

Out[19]: 'pear'
```

List particulars

- Individual elements and sublists can be accessed via slices *list[index1:index2]*
- Use **append()** to add an element to the end of a list
- Delete a item by value with **remove()**
- Use **in** to test for a value in a list
- Delete an item by index with **del**
- Convert to and from strings with **join()** and **split()**
- **sort()** and **sorted()** do different things: **sort()** sorts the original list and **sorted()** returns a sorted copy
- Use **copy()** instead of assignment operator '=' to copy a list

Tuples

- Tuples can be created by assigning the empty tuple set **()** to a variable, or by setting it to one or more elements separated by a comma
- Use **tuple()** to convert objects to tuples

Why use tuples at all?

1. Tuples use less space in memory
2. Elements of tuples can't be clobbered
3. Arguments of functions are passed as tuples
4. Tuples are immutable, and therefore can be used as *dictionary keys*...

Dictionaries

Dictionaries are an unordered collection of objects indexed by *keys* instead of integers (as in lists). Every element of a dictionary is a *key:value* pair, with the key being an immutable item and the value being any Python object.

Dictionaries are similar *hashes* or *associative arrays* in other programming languages. A Python dictionary is often referred to as a *dict* for brevity.

```
Dictionary = {key1:value1, key2:value2, …}
```

More about dictionaries

- Create dictionaries using the empty dictionary **{}**
- Use **dict()** to convert a list of two-value sequences to a dictionary
- Add elements to a dictionary by key, e.g., *dictionary[key]=value*
- Delete an item using **del**
- Get an item by key: *dictionary[key]* returns value
- Get all keys via **keys()** method (*dict_keys* type returned - use **list()** to convert)
- Get all values via **values()** method (list returned)
- **items()** returns all key-value pairs as a list of tuples
- Copy dictionaries with **copy()**

3. Control Structures

- Code comments (#)
- **if/elif/else** comparisons
- **while** loops
- **for** loops
- Functions

For each of these structures (except comments), proper indentation must be maintained. Python uses whitespace to define blocks of code. The convention is four spaces per block.

if/else statements

```
if condition:
    statements  # condition is True.  This block must be indented!
else:
    statements  # condition is False.  This block must also be indented!
```

if/elif/else

```
if condition1:
    statements
elif condition2:
    statements
else:
    statements
```

Things that are False

- Boolean False
- Integer 0
- Float 0.0
- String ``
- List []
- Tuple ()
- Dict {}
- null None

Things that are True

Anything else that is not False.

1

True

Any non-empty set

...

while loops

while condition:

    statements # the statements in this block run while the condition is True

for loops

for item **in** collection:

    Statements

For loop statement blocks are useful for iterating over a collection of items, like in a list or tuple.

Functions

```
def function_name():
    statement_block
    return      #  sometimes with argument, sometimes omitted
function_name()         # function call
```

HELP!?!

What do you do if you get stuck?

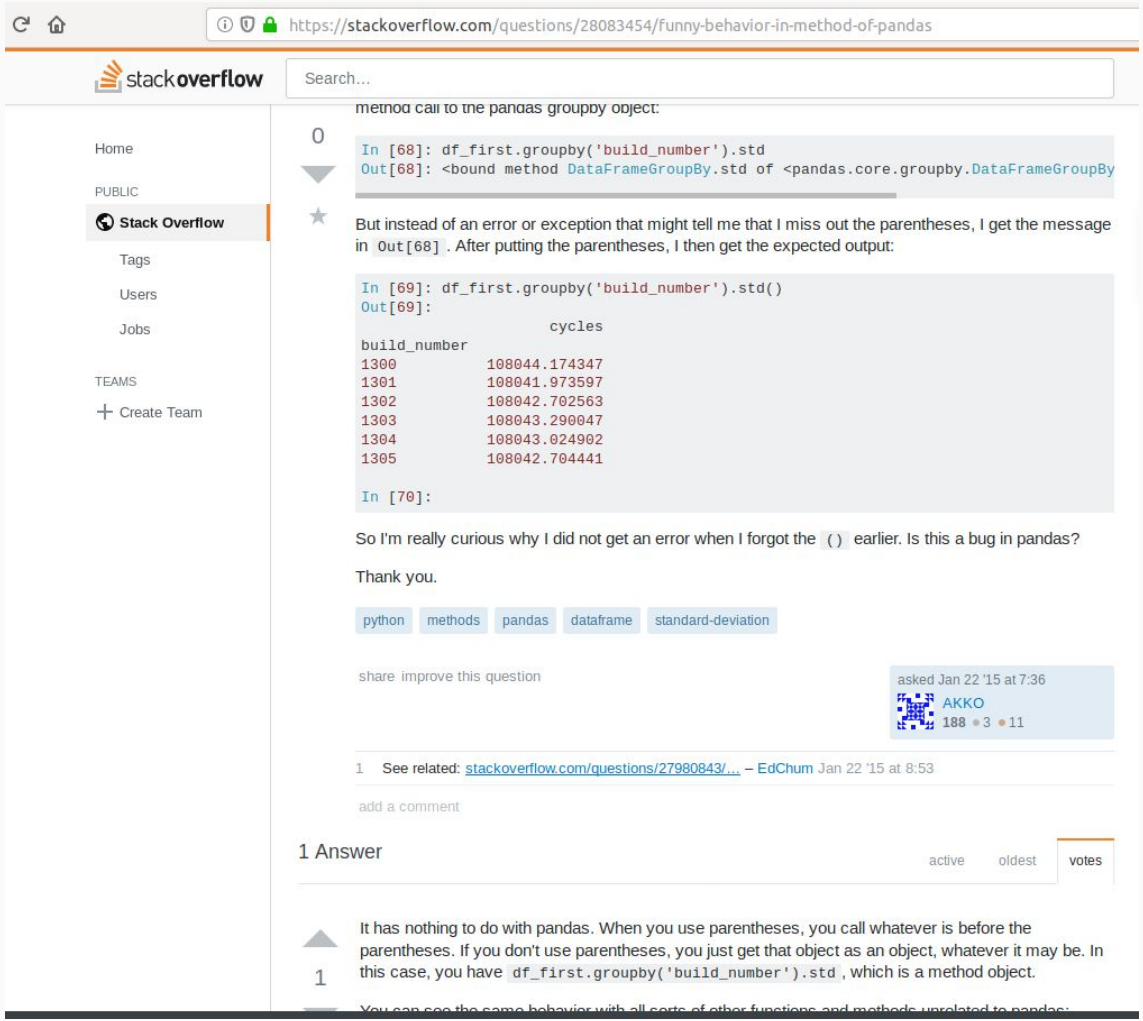
Google search is your friend

Stack OverFlow

A forum for code hackers, Stack Overflow is a great resource asking general questions about any programming or IT project.

Be forewarned: How you ask a question makes a big difference in the kind of response you get.

Also: Not everyone is an expert. But it is a good place for suggestions if stuck.



Additional Sources of Python info

Some recommended books:

- Wes McKinney, Python For Data Analysis, 2nd ed. (covers Python 3, author wrote Pandas Library)
- Bill Lubanovic, Introducing Python (covers Python 3)

Python.org list of tutorials:

https://wiki.python.org/moin/BeginnersGuide/Programmers

Coursera.org also has a large number of Python courses

Happy Hacking!

Keven Haynes
Emory University
Rollins School of Public Health

khaynes@emory.edu
khaynesemory.github.io