NIAPythonDay1

May 16, 2017

NIA Intro to Python Class - May 15, 2017

1 Day 1

1.1 About the instructor

- Chris Coletta, Computer Scientist
- Human Genetics Section (Schlessinger Lab), LGG
- christopher.coletta@nih.gov
- x8170
- Room 10C222
- LinkedIn, personal webpage, twitter

1.2 Course format

- Bootcamp style - no prior programming knowledge assumed
- 6 total hours of instruction
- No homework
- Goal of this course: Spreadsheet Manipulation
  - Read in and Excel file
  - Do some transformations on the data
  - Visualize the data
- Roadmap
  - Day 1: Background; the IDE; basic syntax & data types
  - Day 2: Iterable data types; Operators
  - Day 3: Controling the flow of your program
  - Day 4: Data manipulation/visualization

1.3 Python fast facts

- General-purpose programming language
- Open-source software
- Free
- Started in 1989 by Guido van Rossum
- Emphasizes code readability => Lower barrier to entry than other programming languages

1.4 Help Learning Python

- Python for Scientists and Engineers - Free Book by Shantnu Tiwari
- Google search!
- Use Python help() command

1.5 Ecosystem of Python Data Analysis Software

Anaconda is one of many Python "distributions" that bundles the following three types of software:

1.5.1 "Core" Python

- The Python interpreter - understands the syntax of the Python language
- Python Standard Library
  - Built-in tools, mathmatical functions, algorithms
  - Organized into sub-units called "packages" that you import

1.5.2 Third-party packages

- Hundreds of them. My favorites:
  - NumPy - Linear algebra/matrices
  - SciPy - Statistics + math
  - statsmodels - Linear models/regression
  - matplotlib - Makes plots/figures
  - Seaborn - Really nice plots/figures
  - Pandas - Spreadsheet replacement/data manipulation
  - Scikit-learn - Machine Learning
  - Scikit-image - Image processing
  - Biopython - Bioinformatics
  - WND-CHARM - NIA in-house image analysis/machine learning

1.5.3 IDE

- Jupyter Notebook - Creates sharable documents containing live code, equations, visualizations and explanatory text.
- Spyder - "Scientific PYthon Development EnviRonment"

1.6 IDE Concepts

- Integrated Development Environment - The software app you use to build and test your code
- Compare and contrast how the user interfaces with Python and Excel
  - Excel: Little cubby holes that you can shove data into
  - Python: Give it a command to enter data
  - Excel: You're the customer in the restaurant: All possible operations listed in the MENU
  - Python: You're the chef in the restaurant: Write your own program by following recipes/cookbooks

  - Excel: Don't really talk to other files
  - Python: Input/output to other files is fundamental
  - Excel: Sandbox: input and output to the same place
  - Python & Jupyter Notebook: Clear workflow, like a cooking recipe or driving directions. Good for reproducible science.

- Example notebooks here and here
- Jupyter components
  - Do coding inside web browser
  - Browser communicates with a "kernel" (on local machine or in the cloud)
  - nbconvert to save notebook into a .py, HTML, PDF, LaTeX, etc

1.7 Exploring the Jupyter IDE

- Do the user interface tour

1.7.1 Cell types

Markdown cells

- Markdown Document-formatting style that is easly convertable to HTML
- Headings preceeded by #
- unordered lists preceeded by a *
- ordered lists preceeded by a number
- Math equations go in between two $, example: $t = \frac{\hat{\beta} - \beta_{H_0}}{s.e.(\hat{\beta})}$
- Create links like this

Code Cells

- commands go in here
- tab-completion

1.7.2 Interacting with cells

Command mode

- Press Esc - box turns blue
- Useful shortcuts:
  - b = Insert cell below
  - a = inser cell above
  - dd = Delete cell
  - Shift + up or down = select/highlight two or more cells
  - M = merge highlightes cells into one

Edit mode

- Double click to edit - box turns green
- Useful shortcuts
  - Ctrl + Shift + - = split cell at cursor location
  - Enter = gives you a new line inside the same cell
  - Shift + Enter = Runs the code in this cell and go to the next one
  - Ctrl + Enter = Runs the code in this cell and stay on this one

1.8 Linux-style file system commands

Here are some useful Linux file commands that Jupyter notebook understands:

1.8.1 pwd

pwd stands for "Present working directory." Tell me whare I am on the filesystem right now.

```
In [1]: pwd
```

```
Out[1]: '/Users/colettace/courses/May2017_NIA_Python_Course'
```

1.8.2 ls

ls will list files in present working directory.

```
In [2]: ls
```

```
MicroarrayAnalysisUsingPython.ipynb  NIAPythonDay1.synctex.gz
NIAPythonDay1.aux                    NIAPythonDay1.tex
NIAPythonDay1.ipynb                  NIAPythonDay2.ipynb
NIAPythonDay1.log                    NIAPythonDay3.ipynb
NIAPythonDay1.out                    README.md
NIAPythonDay1.pdf                    samplefile.xlsx
```

1.8.3 mkdir

mkdir- "make a new folder"

```
In [3]: mkdir NewFolder
```

```
In [4]: ls
```

```
MicroarrayAnalysisUsingPython.ipynb  NIAPythonDay1.tex
NIAPythonDay1.aux                    NIAPythonDay2.ipynb
NIAPythonDay1.ipynb                  NIAPythonDay3.ipynb
NIAPythonDay1.log                    NewFolder/
NIAPythonDay1.out                    README.md
NIAPythonDay1.pdf                    samplefile.xlsx
NIAPythonDay1.synctex.gz
```

1.8.4 cd

cd foldername - Change Directory of present working directory to foldername

```
In [5]: cd NewFolder
```

```
/Users/colettace/courses/May2017_NIA_Python_Course/NewFolder
```

1.8.5 cd ..

cd .. means make the current working directory one folder up.

```
In [6]: cd ..
```

```
/Users/colettace/courses/May2017_NIA_Python_Course
```

1.8.6 cd ~

cd ~ (tilde character) means make the current working directory your home folder.

```
In [7]: cd ~
```

```
/Users/colettace
```

1.8.7 rmdir

rmdir foldername means delete the folder in the current working directory named foldername.

```
In [8]: cd /Users/colettace/courses/May2017_NIA_Python_Course
```

```
/Users/colettace/courses/May2017_NIA_Python_Course
```

```
In [9]: rmdir NewFolder
```

1.8.8 Some other commands

- cp which copies a file from one place to another
- mv which moves a file from one place to another, or changes the name of a file.
- more...

1.9 First steps with Python syntax

FYI: Python is a case sensitive language, so True is not the same as true.

1.9.1 Statement

- Your Python code is broken up into statements
- One statement per line, except:
  - You can put two statements on one line if they are separated by a semi-colon ;
  - You can break up one statement over multiple lines using a backslash, which is called the "continuation" character.

1.9.2 Comments

Lines preceededed by a hash symbol "#" are ignored by the Python interpreter

```
In [10]: # Run me! nothing happens!!!
```

1.9.3 Assignment

- An assignment is the name on the left side of an equal sign.
- It gives a name to a value.
- Names can have upper and lowercase letters, numbers (as long as it's not the first character), as well as underscores (Shift + -).
- Don't use a name that is also a Python Syntax keyword

```
In [11]: my_fav_number = 42
```

```
In [12]: f00 = "asdfasdf"
```

See the value attached to the name by typing the name

```
In [13]: my_fav_number
```

```
Out[13]: 42
```

1.9.4 Print function

Use the print function to see multiple values at once

```
In [14]: print( my_fav_number, f00)
```

```
42 asdfasdf
```

1.9.5 Code-completion

Hit the TAB key to use code completion to help you type faster.

```
In [15]: my_fav_number
```

```
Out[15]: 42
```

1.9.6 Scalar Data types

Integer int a counting number 1,2,3,....

```
In [16]: 1
```

```
Out[16]: 1
```

BTW: You can use the Python type() command to have Python tell you the type of any named value.

```
In [17]: type( my_fav_number )
```

```
Out[17]: int
```

Float floats are decimal numbers

PEMDAS operators

1. Parentheses - ()
2. Exponent - **
3. Multiplication - *
4. Division - /
5. Addition - +
6. Subtraction - -

1.9.7 Boolean Expressions

A bool can only have a value of True or False.

```
In [18]: True
```

```
Out[18]: True
```

1.9.8 and, or, and not

and and or are "binary operators", meaning you slap them in between two truth values to make one value. not is a unary operator that negates the value after it.

```
In [19]: True and False
```

```
Out[19]: False
```

```
In [20]: True or False
```

```
Out[20]: True
```

```
In [21]: my_bool_value = True and False
         print( my_bool_value )
```

```
False
```

```
In [22]: not True
```

```
Out[22]: False
```

1.10 Keeping track of your named values

```
In [23]: whos
```

```
Variable        Type    Data/Info
---------------------------------
f00             str     asdfasdf
my_bool_value   bool    False
my_fav_number   int     42
```

```
In [24]: whos
```

```
Variable        Type    Data/Info
---------------------------------
f00             str     asdfasdf
my_bool_value   bool    False
my_fav_number   int     42
```

```
In [25]: ?whos
```

1.10.1 Strings

- A string is a data type that contains one or more characters
- Strings are surrounded by matching single or double quotes
- You choose whether to use single or double quotes based on what's in the string.

```
In [26]: "Hello, world!"

Out[26]: 'Hello, world!'

In [27]: 'Hello, world!'

Out[27]: 'Hello, world!'
```

1.10.2 Escape characters

- Escape characters use a backslash followed by a single letter
  - `'\n'` - a newline character
  - `'\t'` - a tab character
  - '\' - a backslash character (escape the escape character)
  - """ - a single quote (but why wouldn't you just use '"'?

```
In [28]: some_escape_chars = 'line one\nline two'
         print( some_escape_chars )

line one
line two
```

Triple double quotes capture the newlines.

```
In [29]: no_need_for_escape_chars = """ hello
         what's up? nuttin'
         "whatchu say to me?"
         word.
         here are some backslashes: \ \\ \\\ \\\\"""
         print (no_need_for_escape_chars)
```

```
 hello
what's up? nuttin'
"whatchu say to me?"
word.
here are some backslashes: \ \ \\ \\
```

1.10.3 String Operations

Repeat

```
In [30]: n = "hello"
         echo = n * 5 # Repeat the string 5 times
         echo
```

```
Out[30]: 'hellohellohellohellohello'
```

Concatenate

```
In [31]: # Concatenate with +
         boast = "I am the very model of a "
         occupation = "modern major general."
         a_string = boast + occupation
         a_string
```

```
Out[31]: 'I am the very model of a modern major general.'
```

To concatenate strings, everything must already be a string. See the problem here?

```
In [32]: profound_statement = "The answer to life, the universe, and everything is " + 42
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-32-b2f14b3f668f> in <module>()
  1 # To concatenate strings, everything must already be a string
  2 # See the problem here?
----> 3 profound_statement = "The answer to life, the universe, and everything is " + 42
```

```
TypeError: must be str, not int
```

Try converting the value to a string:

```
In [ ]: profound_statement = \
           "The answer to life, the universe, and everything is " + str(42)
        profound_statement
```

Slicing strings into substrings

```
In [ ]: # Return a substring using a brackets separated by a colon
        profound_statement[3:22]

In [ ]: # Just because you just returned a substring from a string
        # doesn't mean you changed the original string.
        profound_statement

In [ ]: # [begin index:end index:step]
        profound_statement[3:32:3] # take every 3rd letter

In [ ]: # Leave the index blank to default to the beginning or end of the string
        profound_statement[:25]

In [ ]: profound_statement[25:]

In [ ]: # If i is negative, index is relative to end of string
        profound_statement[-25:]

In [ ]: # Reverse a string by using a negative step value
        "a man, a plan, a canal, panama"[::-1]
```