

# NIAPythonDay1\_July2017

July 14, 2017

NIA Intro to Python Class - July 17, 2017

## 1 Day 1 - Introduction to Python

### 1.1 About the instructor

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

### 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; basic operators
  - Day 2: Complex data types; reading in data; slicing and sorting
  - Day 3: Statistics
  - Day 4: 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
- Case-sensitive

## 1.4 Help Learning Python

- [Python for Scientists and Engineers](#) - Free Book by Shantnu Tiwari
- Google the error message
  - Questions and answers on StackOverflow.com
- Use Jupyter built-in operator ?
- Use Python help() command to see documentation

## 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, mathematical 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 IDEs

- [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
- Example notebooks [here](#) and [here](#)
- 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.

- Jupyter components

- Do coding inside web browser
- Browser communicates with a "kernel" (on local machine or in the cloud)
- Can optionally "Download as..." 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 easily convertible to HTML
- Headings preceded by #
- unordered lists preceded by a \*
- ordered lists preceded 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 = insert cell above
  - dd = Delete cell
  - Shift + up or down = select/highlight two or more cells
  - M = merge highlighted 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 where I am on the filesystem right now.

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

```
Out[1]: '/Users/colettace/courses/July2017_NIA_Python_Course'
```

### 1.8.2 ls

ls will list files in present working directory.

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

```
NIAPythonDay1_July2017.ipynb  NIAPythonDay4_July2017.ipynb*  
NIAPythonDay2_July2017.ipynb* samplefile.xlsx  
NIAPythonDay3_July2017.ipynb* ~$samplefile.xlsx
```

### 1.8.3 mkdir

mkdir- "make a new folder"

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

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

```
NIAPythonDay1_July2017.ipynb  NewFolder/  
NIAPythonDay2_July2017.ipynb* samplefile.xlsx  
NIAPythonDay3_July2017.ipynb* ~$samplefile.xlsx  
NIAPythonDay4_July2017.ipynb*
```

### 1.8.4 cd

cd foldername - Change Directory of present working directory to foldername

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

```
/Users/colettace/courses/July2017_NIA_Python_Course/NewFolder
```

### 1.8.5 cd ..

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

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

```
/Users/colettace/courses/July2017_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/July2017_NIA_Python_Course
```

```
/Users/colettace/courses/July2017_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 The Python 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.
- Can have multiple statements inside a code cell

```
In [ ]:
```

### 1.10 Comments

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

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

## 1.11 Assignment, i.e., give a value a name

- 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
```

## 1.12 print() function

Use the print function to output one or more values at once.

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

```
42 asdfasdf
```

## 1.13 Code-completion using TAB key

Hit the TAB key to use code completion to help you type faster. Most IDEs have this option. Usually a pop-up menu will appear

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

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

## 1.14 Python Data Types: what are they, and why do we care?

- Different types of data, different data types
- Each type has their own various "superpowers," i.e., functionality.
- Advanced programmers often define their own types with their own functionality
- Here, "simple" means that these are types that are built into core Python, and you can use them right away.
- "Fancy" means simply that you need to use the import command before you use them.

### 1.14.1 What the difference between "scalar" and "iterable"?

- You can't loop over a scalar.

### 1.14.2 Scalar Data Types (simple)

- integer int: counting numbers
- float float: decimal numbers
- boolean bool: true/false

### 1.14.3 Iterable Data Types (simple)

- string str: words
- list list: collection of things (ordered)
- dictionary dict: map one value to another (unordered)
- set set: unique collection of things (unordered)

### 1.14.4 Iterable Data Types (fancy)

- NumPy multi-dimensional array: data, images
- Pandas DataFrame: spreadsheet analog

And many more...

## 1.15 Scalar Data Types: Integer (int)

- A counting number 1, 2, 3, -89 ..., 0

In [16]: -23

Out[16]: -23

## 1.16 Scalar Data Types: Float (float)

- Decimal numbers
- An accurate approximation to many many decimal places, but technically not an EXACT representation
- If you want to know more about why decimal numbers are called "floats", click [here](#).

In [17]: 3.14159

Out[17]: 3.14159

In [18]: 1/3

Out[18]: 0.3333333333333333

## 1.17 PEMDAS operators

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

## 6. Subtraction - -

Example: What is  $9 - 3 \div \frac{1}{3} + 1 = ?$

```
In [19]: 9 - 3 / 1/3 + 1
```

```
Out[19]: 9.0
```

```
In [20]: 9 - 3 / (1/3) + 1
```

```
Out[20]: 1.0
```

### 1.18 Using the type() function

Use this to have Python tell you the data type of any expression or named value.

```
In [21]: type( my_fav_number )
```

```
Out[21]: int
```

```
In [22]: type( 3.14159 )
```

```
Out[22]: float
```

### 1.19 Scalar Data Types: Boolean (bool)

Bools can only have a value of True or False.

```
In [23]: True
```

```
Out[23]: True
```

### 1.20 Boolean operators 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 [24]: True and False
```

```
Out[24]: False
```

```
In [25]: True or False
```

```
Out[25]: True
```

```
In [26]: my_bool_value = True and False  
         print( my_bool_value )
```

```
False
```

```
In [27]: not True
```

```
Out[27]: False
```



## 1.21 Some math operators

- < less than
- <= less than or equal to
- > greater than
- >= greater than or equal to
- == is equal to
- != is not equal to

Note the double equal signs is an operator, not an assignment!!

```
In [28]: 5 < 6
```

```
Out[28]: True
```

```
In [29]: 6 <= 6
```

```
Out[29]: True
```

```
In [30]: -6 <= 6
```

```
Out[30]: True
```

```
In [31]: 6 != 6
```

```
Out[31]: False
```

```
In [32]: not( 6 == 6 )
```

```
Out[32]: False
```

## 1.22 Using whos command to keep track of named values

```
In [33]: whos
```

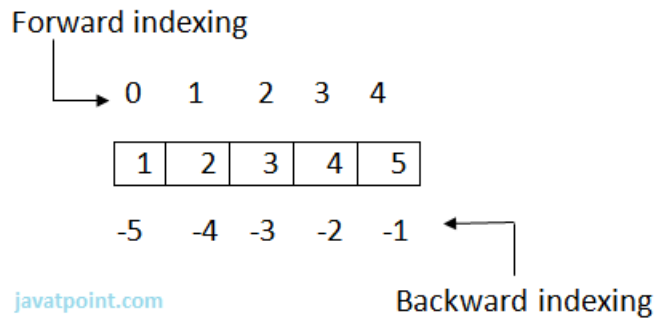
Variable	Type	Data/Info
f00	str	asdfasdf
my_bool_value	bool	False
my_fav_number	int	42

## 1.23 Iterable Data Types: Strings (str)

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

```
In [34]: "Hello, world!"
```

```
Out[34]: 'Hello, world!'
```



<https://www.javatpoint.com/python/images/elementsinalists.png>

```
In [35]: 'Hello, world!'
```

```
Out[35]: 'Hello, world!'
```

I repeat: *No difference between single and double quotes strings!!!!* I promise!

```
In [36]: "Can't"
```

```
Out[36]: "Can't"
```

```
In [37]: '"Really," she said?'
```

```
Out[37]: '"Really," she said?'
```

By the way, I'm *not* talking about the backtick ` , which shares a key with the tilde ~ character. Backtick is *different* than a single quote ' , which shares a key with the double quote " .

## 1.24 Iterable Data Types: Lists (list)

- Container for a collection of values
- Can all be the same type or different, doesn't matter.
- Items delimited by commas, all surrounded by brackets [], not parentheses ()
- The order of the values in the list is remembered

```
In [38]: a_list = [ 1, 2, 3, 1, "a dog" ]
```

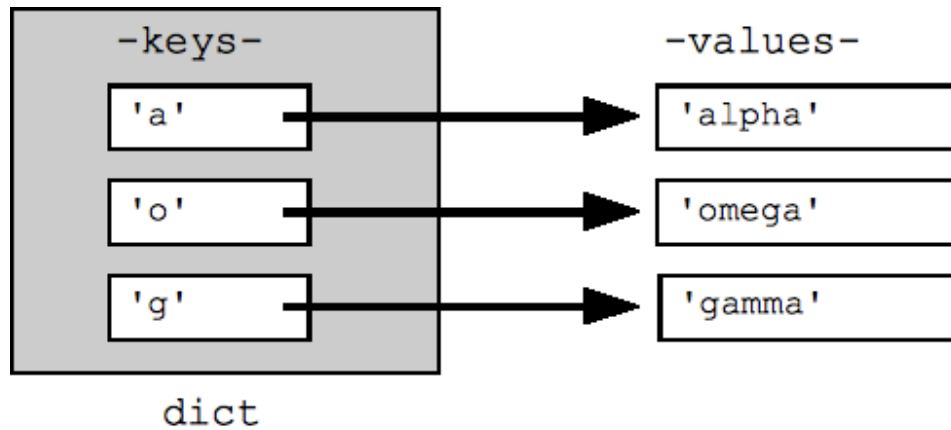
### 1.24.1 Get the ith element from a list using bracket notation

```
In [39]: a_list[0]
```

```
Out[39]: 1
```

```
In [40]: a_list[4]
```

```
Out[40]: 'a dog'
```



<https://developers.google.com/edu/python/images/dict.png>

### 1.24.2 Negative index counts from the back of the list

```
In [41]: a_list[-1]
```

```
Out[41]: 'a dog'
```

## 1.25 Iterable Data Types: Dictionaries (dict)

- A dict is one-way associative array, where "keys" are mapped to "values."
- DOES NOT KEEP TRACK OF ORDER OF KEY-VALUE PAIRS (for that you need `collections.OrderedDict`)

### 1.25.1 Create a dict with stuff in it

The keys are separated by the values by a colon (:), and the key-value pairs are separated by commas.

```
In [42]: simple = { 1 : 'a', 2 : 'b', 3: 'c'}
```

```
In [43]: simple
```

```
Out[43]: {1: 'a', 2: 'b', 3: 'c'}
```

```
In [44]: simple[1]
```

```
Out[44]: 'a'
```

### 1.25.2 Create an empty dict

Declare empty dict with {}, or `dict()`.

```
In [45]: {}
```

```
Out[45]: {}
```

```
In [46]: simple['new_key'] = 'new_value'
```

```
In [47]: simple
```

```
Out[47]: {1: 'a', 2: 'b', 3: 'c', 'new_key': 'new_value'}
```

## 1.26 Iterable Data Types: Sets (set)

- Similar to math concept of sets; has operations like union, intersection, etc.
- Sets are unindexed, unordered, and contains no duplicates.
- My personal favorite of the Python standard types!

### 1.26.1 Create a set with stuff in it

Declare a set by putting values inside braces.

```
In [48]: a_set = {'set', 'of', 'words'}
```

### 1.26.2 Create an empty set

Make an empty using set().

```
In [49]: empty_set = set() # not {}, that would be an empty dict
```

```
In [50]: empty_set
```

```
Out[50]: set()
```

```
In [51]: empty_set.add('hi')
```

```
In [52]: empty_set
```

```
Out[52]: {'hi'}
```

```
In [53]: first = {1,2,3,4,5}
         second = {4,5,6,7,8}
```

### 1.26.3 Set Union Operator (|) - "or"

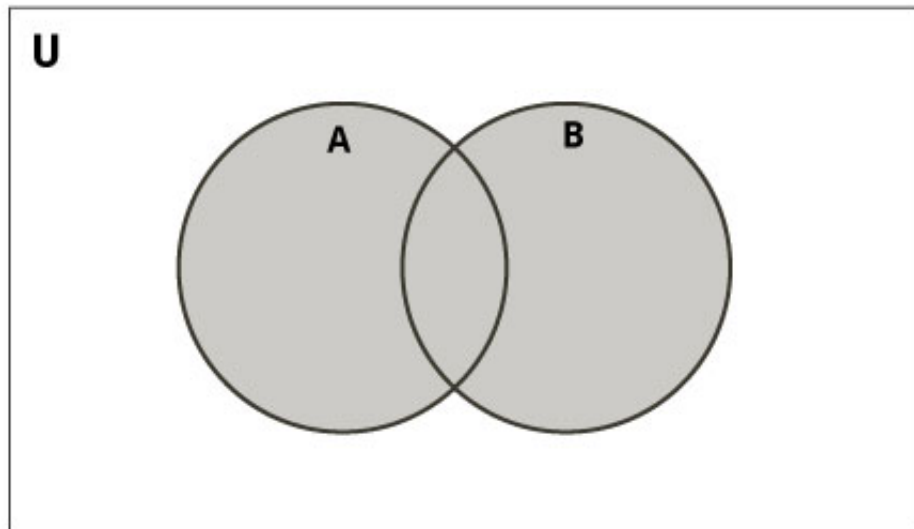
```
In [54]: first | second
```

```
Out[54]: {1, 2, 3, 4, 5, 6, 7, 8}
```

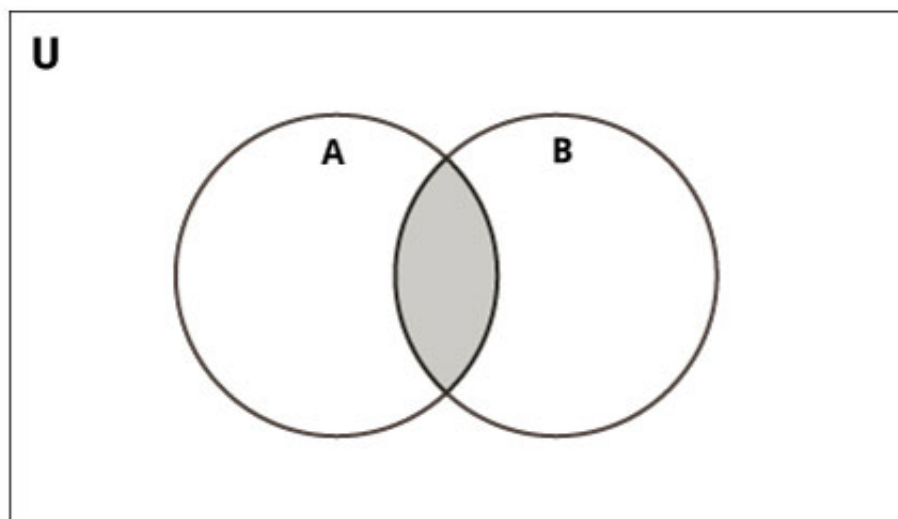
### 1.26.4 Set Intersection Operator (&) - "and"

```
In [55]: first & second
```

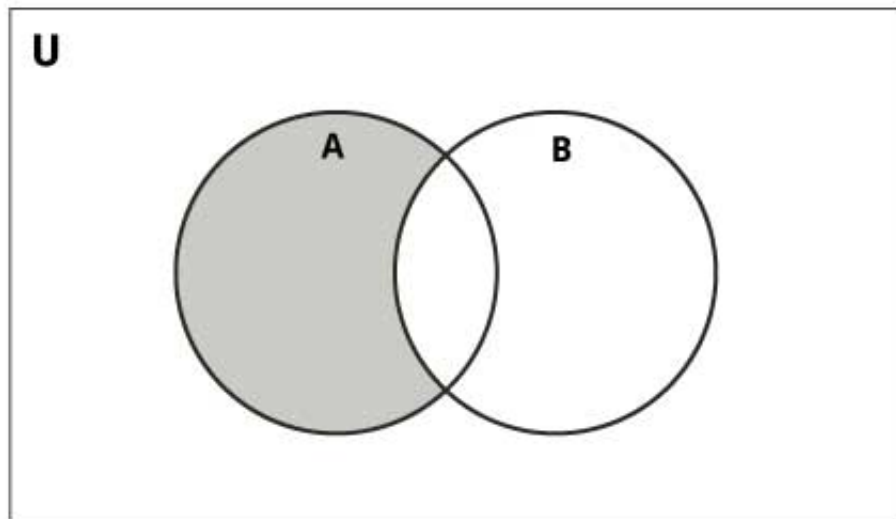
```
Out[55]: {4, 5}
```



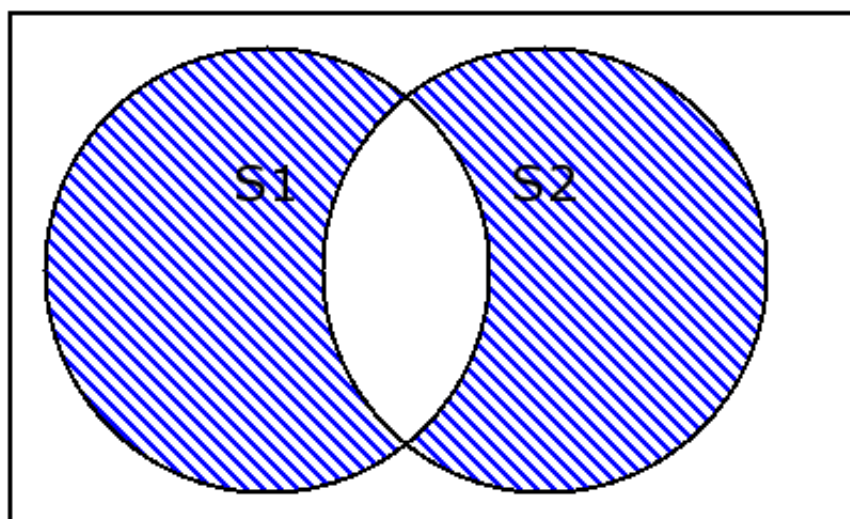
<https://cdn.programiz.com/sites/tutorial2program/files/set-union.jpg>



<https://cdn.programiz.com/sites/tutorial2program/files/set-intersection.jpg>



<https://cdn.programiz.com/sites/tutorial2program/files/set-difference.jpg>



[http://www.itmaybeahack.com/book/python-2.6/html/\\_images/p2c6-symmdiff.png](http://www.itmaybeahack.com/book/python-2.6/html/_images/p2c6-symmdiff.png)

### 1.26.5 Set Difference Operator (-)

```
In [56]: first - second
```

```
Out[56]: {1, 2, 3}
```

### 1.26.6 Set Symmetrical Difference Operator (^)

```
In [57]: first ^ second
```

```
Out[57]: {1, 2, 3, 6, 7, 8}
```