NIAPythonDay1

March 22, 2019

Python Logo

1 Welcome to the Data Analysis Workshop!

1.0.1 Copy course materials to your desktop from:

- If using NIDA Laptops: `T:\TempTransfer\000_NIA_Python_Course`
- If using your own laptop: download from Github
  - Click green clone or download button

1.1 Goals of this course

Students of this course should gain familiarity with:

- Python's capabilities, strengths and weaknesses
- Running Python code within a Jupyter Notebook environment
- Generating tables/figures and doing simple stats
- Keywords/names of concepts to help you Google things yourself

1.2 Target audience

- Bench scientists who want to analyze their own data
- Microsoft Excel users
- For beginners: No prior programming knowledge assumed

1.3 What will be covered

- Day 1: Basic syntax, data types and operators
- Day 2: Fancy data types (arrays and data frames)

- Day 3: Exploratory data analysis
- Day 4: Case study: Gene expression ribbon plot

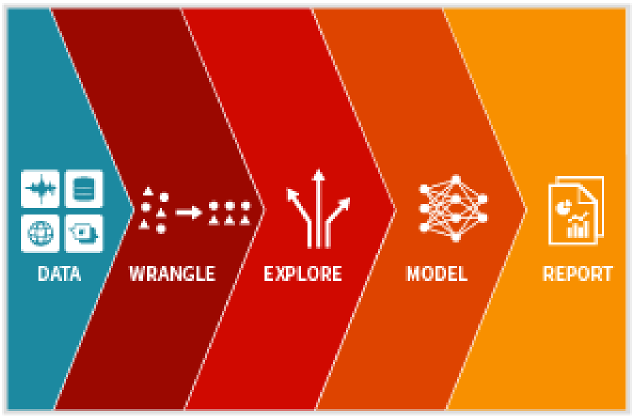


Image Credit: Wolfram

1.4 What will NOT be covered

1.4.1 Statistical inference

- Multiple Regression (Logistic, LASSO, ElasticNet), regression diagnostics
- Generalized linear models (GLM), Generalized Linear Mixed-Effect Models (GLMM)
- Analysis of Variance (ANOVA), Analysis of Covariance (ANCOVA), Post-hoc analysis (Tukey)
- Multivariate Regression (MANOVA, MANCOVA)

1.4.2 Machine learning

- Supervised Learning algorithms: Random Forest, Naive Bayes, Artificial Neural Networks, Ensemble Methods, etc.
- Unsupvervised Learning algorithms: Cluster analysis via PCA, T-SNE, DBSCAN, etc.
- AutoML: Automates tasks like pre-processing, feature engineering, feature selection, algorithm selection, metrics, hyperparameter optimization, etc.
- Image Analysis: Image classification, instance segmentation, semantic segmentation, etc.

1.4.3 Software Engineering

- Analysis of algorithms
- Object oriented programming
- Regular expressions
- Helpful websites: Software Carpentry and Data Carpentry

1.5 Basic data analysis toolkit

1.5.1 Wrangle and Clean

- Missing Data

- Filter
- Group-by operations (Split/apply/combine)
- Merge two spreadsheets (JOIN operations)
- Bin continuous variables into categorical variables
- Simple and complex sort

1.5.2 Exploratory Data Analysis

- Summary statistics
- Pivot table
- Histogram
- Scatter plot
- Box and whiskers
- Pairwise scatterplot matrix

1.5.3 Model

- Linear regression
- Tests for mean (t-test & Wilcoxon rank-sum)
- Nice-to-have: Pass data from Python to R for further analysis using rpy2

1.5.4 Visualize

- Facet plot
- Heatmap
- Manhattan plot

1.6 About the Computational Biology Core (CBC)

Core facility housed in LGG
Room 10C222
Seminar or training every month
Two powerful Windows computers with lots of software and remote access available
Soon: BRC cloud computing (RAM, GPUs, virtual machines)

1.6.1 CBC Staff

Supriyo De, Ph.D., Staff Scientist
Krystina Mazan-Mamczarz, Ph.D., Senior Research Fellow
Qiong (Joan) Meng, Ph.D., Post-doctoral fellow
Christopher Coletta, M.S. expected 2019, Computer Scientist

1.7 Why Python (vs. Excel, Matlab, R, etc)

- Free (no cost) and free (open-source)
- General-purpose: data, web, apps
- Readible: simple, non-cluttered syntax
- Expressive: do more with less lines of code (relative to C)
- Popular: #3 language in TIOBE Index

1.7.1 Relative Strengths

Extensive, mature libraries for: * Image analysis * Video analysis (self driving cars platform) * Machine learning, especially artificial neural networks * Natural language processing (sentiment analysis)

1.7.2 Relative Weaknesses

- Mobile apps

1.8 Python Learning Resources

- SoloLearn phone app: Python 3 tutorial
- Python for Scientists and Engineers - Free Book by Shantnu Tiwari
- PyData YouTube channel
- Questions and answers on StackOverflow.com
- Use Jupyter built-in operator ?

1.9 Ecosystem of Python Data Analysis Software

Anaconda is one of many Python "distributions" that bundles core Python, essential 3rd party packages, and various IDEs.

2 Integrated Development Environment

The software app you use to build and test your code.

2.1 Python IDE Choices

- Spyder
- PyCharm
- Jupyter - IDE for this workshop

2.2 Jupyter

- IDE for Python, R, Bash, many others
- Web browser interface: communicates with either local or remote back-end ("kernel")
- Creates a sharable document called a notebook.
- Notebook divided up into cells that contain code, output and documentation ("Markdown cell").

2.3 Jupyter Markdown Cell

- 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 Dollar signs, example: $t = \frac{\hat{\beta}-\beta_{H_0}}{s.e.(\hat{\beta})}$
- Create links like this

2.4 Code Cells

- Python code goes in here
- Shift+Enter to run and goto the next cell
- Ctrl+Enter to run code and stay on current cell
- Upon execution, a number shows up on the left indicating order of execution.
- You don't have to run code cells in the order they appear in the notebook.

```
In [1]: print( "Hello, world!" )
```

```
Hello, world!
```

2.5 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 highlightes cells into one

2.6 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

3 Basic Python Syntax

3.1 Comments

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

```
In [2]: # Run me! nothing happens!!!
        # askfdjhdsakfadhsfadsk
```

3.2 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 [3]: my_fav_number = 42

In [4]: my_fav_number

Out[4]: 42

In [5]: f00 = "asdfasdf"

In [6]: f00

Out[6]: 'asdfasdf'

In [7]: my_fav_animal = "zebra"

In [8]: zebra = "Lawrence"

In [9]: my_fav_animal = zebra

In [10]: my_fav_animal

Out[10]: 'Lawrence'

In [11]: my_fav_animal = 'Lion'

In [12]: zebra

Out[12]: 'Lawrence'
```

See the value attached to the name by typing the name

```
In [13]: my_fav_number

Out[13]: 42
```

3.3 print() function

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

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

42 asdfasdf
```

3.4 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
```

3.5 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.

3.5.1 Scalar Data Types (simple)

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

3.5.2 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)

3.5.3 What the difference between “scalar” and “iterable”?

- You can’t loop over a scalar.

3.5.4 Iterable Data Types (fancy)

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

And many more…

3.6 Scalar Data Types: Integer (int)

- A counting number 1, 2, 3, -89 …, 0

```
In [16]: -23

Out[16]: -23

In [17]: type( my_fav_number)

Out[17]: int

In [18]: type( -23 )

Out[18]: int
```

3.7 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 [19]: type( 3.14159 )
```

```
Out[19]: float
```

```
In [20]: type( 1/3 )
```

```
Out[20]: float
```

3.8 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 [21]: 9 - 3 / 1/3 + 1
```

```
Out[21]: 9.0
```

```
In [22]: 9 - 3 / (1/3) + 1
```

```
Out[22]: 1.0
```

3.9 Using the type() function

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

```
In [23]: type( my_fav_number )
```

```
Out[23]: int
```

```
In [24]: type( 3.14159 )
```

```
Out[24]: float
```

3.10 Scalar Data Types: Boolean (bool)

Bools can only have a value of True or False.

```
In [25]: True

Out[25]: True

In [26]: False

Out[26]: False

In [27]: type( True)

Out[27]: bool
```

3.11 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.

```
In [28]: False and False

Out[28]: False

In [29]: True and True

Out[29]: True

In [30]: True or False

Out[30]: True

In [31]: True or True

Out[31]: True

In [32]: False or False

Out[32]: False

In [33]: my_bool_value = True and False
         print( my_bool_value )

False
```

not is a unary operator that negates the value after it.

```
In [34]: not True

Out[34]: False

In [35]: (True or False) and False

Out[35]: False

In [36]: True and False or True

Out[36]: True
```

3.12 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 [37]: 5 < 6

Out[37]: True

In [38]: 6 <= 6

Out[38]: True

In [39]: -6 <= 6

Out[39]: True

In [40]: 6 != 6

Out[40]: False

In [41]: not( 6 == 6 )

Out[41]: False
```

3.13 Using whos command to keep track of named values

```
In [42]: whos

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

3.14 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.
- Escape characters: Backslash followed by a letter to render special characters

  - `\n`: New line
  - `\t`: Tab
  - `\"`: Quote character (not end of string)

```
In [43]: "Hello, world!"

Out[43]: 'Hello, world!'

In [44]: 'Hello, world!'

Out[44]: 'Hello, world!'
```

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

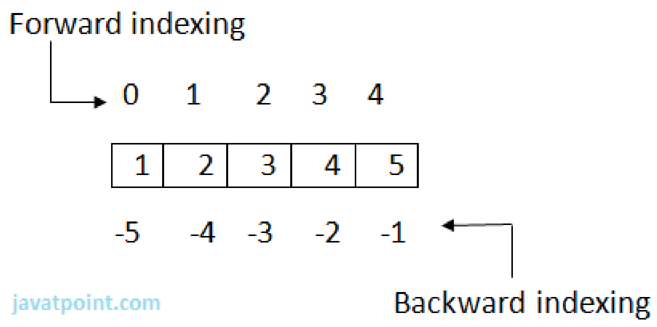
```
In [45]: "Can't"

Out[45]: "Can't"

In [46]: '"Really," she said?'

Out[46]: '"Really," she said?'

In [47]: "I said, \"Hi my name is Chris\""

Out[47]: 'I said, "Hi my name is Chris"'

In [48]: " First line\n Second line"

Out[48]: ' First line\n Second line'

In [49]: print( " First line\t Second line" )

 First line          Second line
```

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 ".

3.15 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 [50]: a_list = [ 1, 2, 3, 1, "a dog", 'a cat' ]
```



list indexing in python

3.15.1 Get the ith element from a list using bracket notation

```
In [51]: a_list[0]

Out[51]: 1

In [52]: a_list[4]

Out[52]: 'a dog'
```

3.15.2 Negative index counts from the back of the list

```
In [53]: a_list[-1]

Out[53]: 'a cat'
```

3.15.3 Get position of a value within a list using .index()

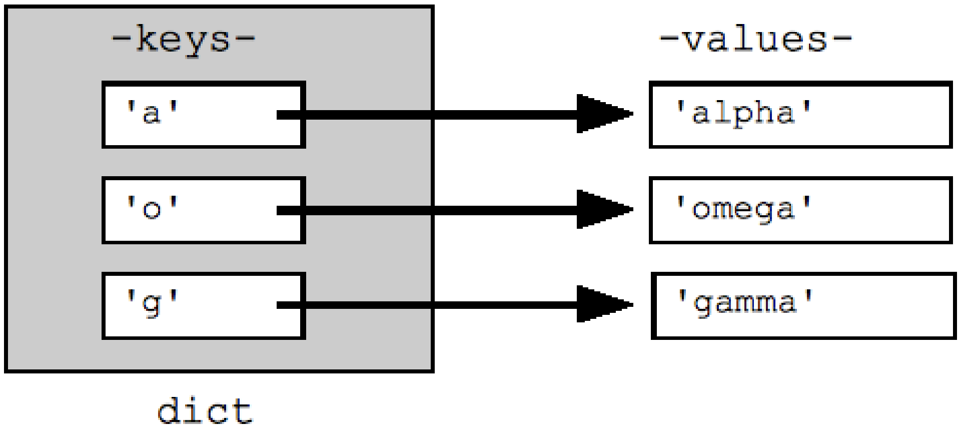
```
In [54]: a_list.index('a cat')

Out[54]: 5
```

3.15.4 Use the “unpacking” syntax to get values out of small lists

```
In [55]: a_few_things = [ "hello", "goodbye", 42 ]

In [56]: a_few_things

Out[56]: ['hello', 'goodbye', 42]

In [57]: first, second, third = a_few_things

In [58]: first

Out[58]: 'hello'

In [59]: second

Out[59]: 'goodbye'

In [60]: third

Out[60]: 42
```

3.16 Iterable Data Types: Dictionaries (dict)

- A dict is one-way associative array, where "keys" are mapped to "values."
- Note: A dict does not keep track of the order in which you inputted the key-value pairs
  - for that you need collections.OrderedDict



3.16.1 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 [61]: simple = { 1 : 'a', 2 : 'b', 3: 'c'}

In [62]: simple

Out[62]: {1: 'a', 2: 'b', 3: 'c'}
```

3.16.2 Access an element in a dict using its key and bracket notation []

```
In [63]: info = { 'first name': "Chris",
                  "last name" : "Coletta"}

In [64]: info

Out[64]: {'first name': 'Chris', 'last name': 'Coletta'}

In [65]: info['first name']

Out[65]: 'Chris'
```

3.16.3 Keys, not values go into the dict, or you get an error

```
In [66]: info['Chris']
```

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

    KeyError                                  Traceback (most recent call last)

    <ipython-input-66-34f6e1331251> in <module>
----> 1 info['Chris']


    KeyError: 'Chris'
```

3.16.4 Create an empty dict

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

```
In [67]: {}

Out[67]: {}
```

3.16.5 Add a new key-value pair to an existing dict using bracket notation []

```
In [68]: simple['new_key'] = 'new_value'

In [69]: simple

Out[69]: {1: 'a', 2: 'b', 3: 'c', 'new_key': 'new_value'}
```

3.16.6 Get just the keys or just the values

Every dict has the built-in functions (“methods” in Pythonic speak) .keys() and .values()

```
In [70]: simple.keys()

Out[70]: dict_keys([1, 2, 3, 'new_key'])

In [71]: simple.values()

Out[71]: dict_values(['a', 'b', 'c', 'new_value'])
```

3.17 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!

3.17.1 Create a set with stuff in it

Declare a set by putting values inside braces.

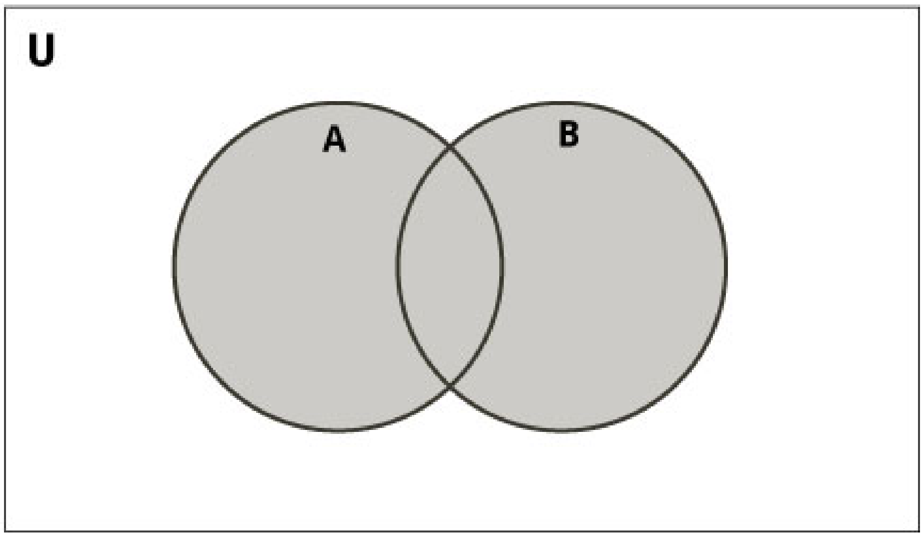
```
In [72]: set('GATTACA')

Out[72]: {'A', 'C', 'G', 'T'}

In [73]: a_set = {'set', 'of', 'words'}

In [74]: a_set

Out[74]: {'of', 'set', 'words'}
```

3.17.2 Create an empty set

Make an empty using set().

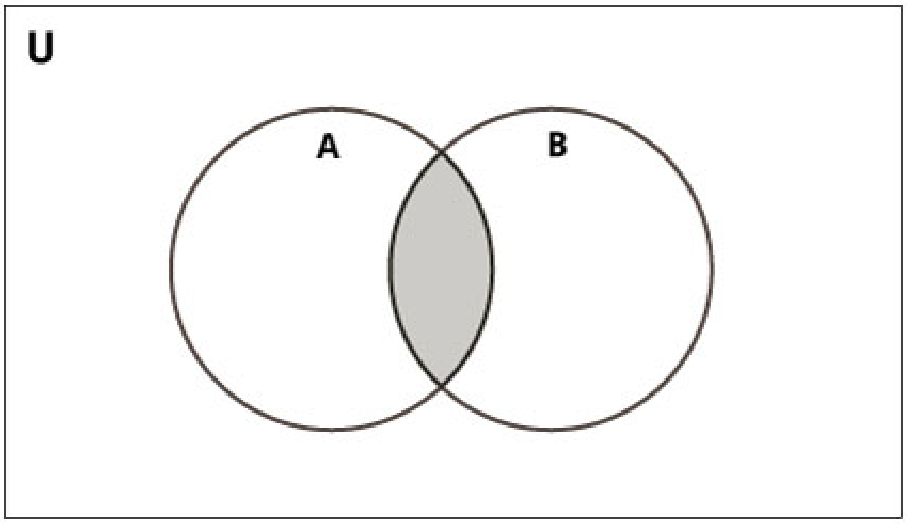
```
In [75]: empty_set = set() # not {}, that would be an empty dict

In [76]: empty_set

Out[76]: set()

In [77]: empty_set.add( 'hi' )

In [78]: empty_set

Out[78]: {'hi'}

In [79]: first = {1,2,3,4,5}
         second = {4,5,6,7,8}
```

3.17.3 Set Union Operator (|) - "or"



```
In [80]: first | second

Out[80]: {1, 2, 3, 4, 5, 6, 7, 8}
```

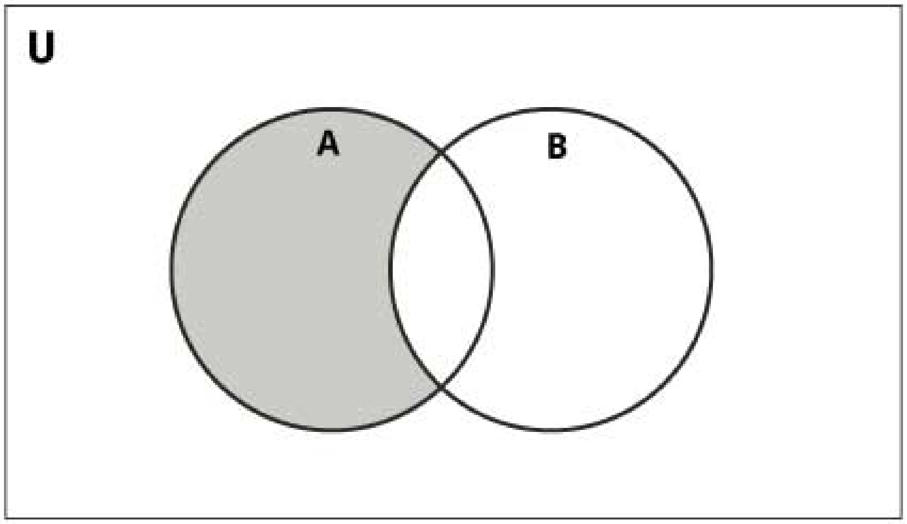
3.17.4 Set Intersection Operator (&) - "and"



```
In [81]: first & second
```

```
Out[81]: {4, 5}
```

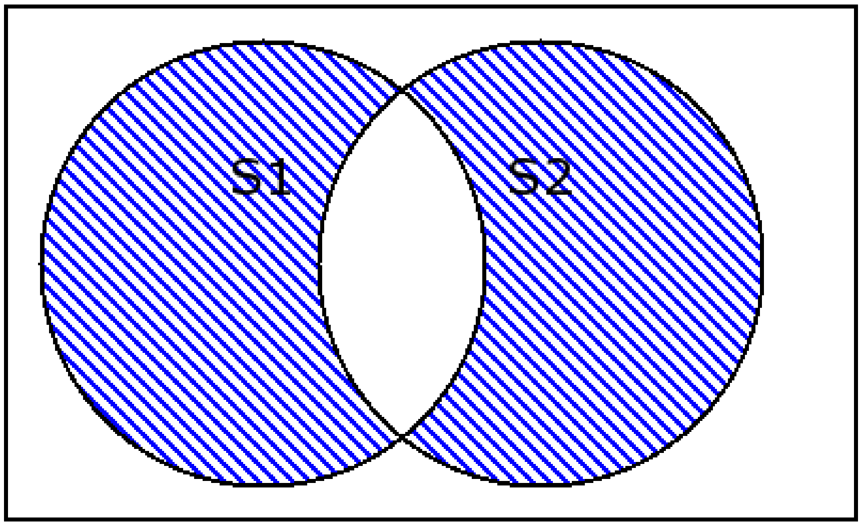
3.17.5 Set Difference Operator (-)



```
In [82]: first - second
```

```
Out[82]: {1, 2, 3}
```

3.17.6 Set Symmetrical Difference Operator (^)



```
In [83]: first ^ second

Out[83]: {1, 2, 3, 6, 7, 8}
```

3.18 How many elements in an iterable? Use len()

```
In [84]: len( first ^ second )

Out[84]: 6

In [85]: a_list

Out[85]: [1, 2, 3, 1, 'a dog', 'a cat']

In [86]: len(a_list)

Out[86]: 6
```

3.19 Can you change a value's type? Yes!

Use these functions to "coerce" a value from one type to another:

- `int()`
- `float()`
- `bool()`
- `list()`
- `dict()`
- `set()`
- et al.

```
In [87]: type( 45 )
```

```
Out[87]: int

In [88]: type( '45' )

Out[88]: str

In [89]: a_string = '45'

In [90]: 56 + a_string


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-90-e1929fed3ecc> in <module>
    ----> 1 56 + a_string


        TypeError: unsupported operand type(s) for +: 'int' and 'str'


In [91]: a_string

Out[91]: '45'

In [92]: int( a_string )

Out[92]: 45

In [93]: 56 + int(a_string)

Out[93]: 101

In [94]: float( '-45.0345' )

Out[94]: -45.0345

In [95]: int( float( '-45.0345' ) )

Out[95]: -45

In [96]: float( 45 )

Out[96]: 45.0

In [97]: str( 56 ) + a_string

Out[97]: '5645'

In [98]: list( "listify me!")
```

```
Out[98]: ['l', 'i', 's', 't', 'i', 'f', 'y', ' ', 'm', 'e', '!']

In [99]: set( "listify me!" )

Out[99]: {' ', '!', 'e', 'f', 'i', 'l', 'm', 's', 't', 'y'}

In [100]: float( 3 )

Out[100]: 3.0

In [101]: int( 3.14159 )

Out[101]: 3

In [102]: int( 4.9 )

Out[102]: 4

In [103]: int( -4.9 )

Out[103]: -4

In [104]: round( 4.9 )

Out[104]: 5

In [105]: bool( "a_string")

Out[105]: True

In [106]: bool( "" )

Out[106]: False

In [107]: len( "" ) != 0

Out[107]: False
```

3.20 Iterating over items in a list using a for loop

- Statements you want to be repeated inside the loop should be *indented* below the first line.
- Use the TAB key to indent.
- In between the for keyword and the in is the placeholder name whose value changes each time through the loop.

```
In [108]: months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'June']

In [109]: for m in months:
              print( m )

Jan
Feb
Mar
Apr
May
June


In [110]: m

Out[110]: 'June'
```

3.21 Iterating over items in a dict using a for loop using .items() syntax

Use the unpacking syntax within the for .. in syntax to directly assign names to the key and value separately.

```
In [111]: num_days_in_month = { 'Jan' : 31,
                               'Feb' : 28,
                               'Mar' : 31,
                               'Apr' : 30 }

In [112]: num_days_in_month

Out[112]: {'Jan': 31, 'Feb': 28, 'Mar': 31, 'Apr': 30}

In [113]: for m, d in num_days_in_month.items():
              print( "There are", d, "days in", m )

There are 31 days in Jan
There are 28 days in Feb
There are 31 days in Mar
There are 30 days in Apr
```

3.22 Day 1 review

1. Python ecosystem of tools
2. Jupyter Notebook is code, output and documentation all in one document
3. Type code into cells, and to run them you press Shift-Enter
4. Tab completion is nice
5. Different data types for different data
6. Operators take one or more input values and turn them into other values *based on the input values type*
7. Converting data from one type to another using the function syntax, e.g., int()
8. Iterating over iterables using a for loop