Introduction to Python & Jupyter Notebooks

NGP Bootcamp, Day 0

Objectives for this morning

- Introduce Jupyter Notebooks & Python syntax
- 01/02 - Types of variables and data structures
- 03 - Booleans, conditionals, functions, and for loops
- 04 - Object-oriented programming

Corresponding notebooks, all found here: https://github.com/NGP-Bootcamp/Bootcamp2020/tree/master/00-IntroPython

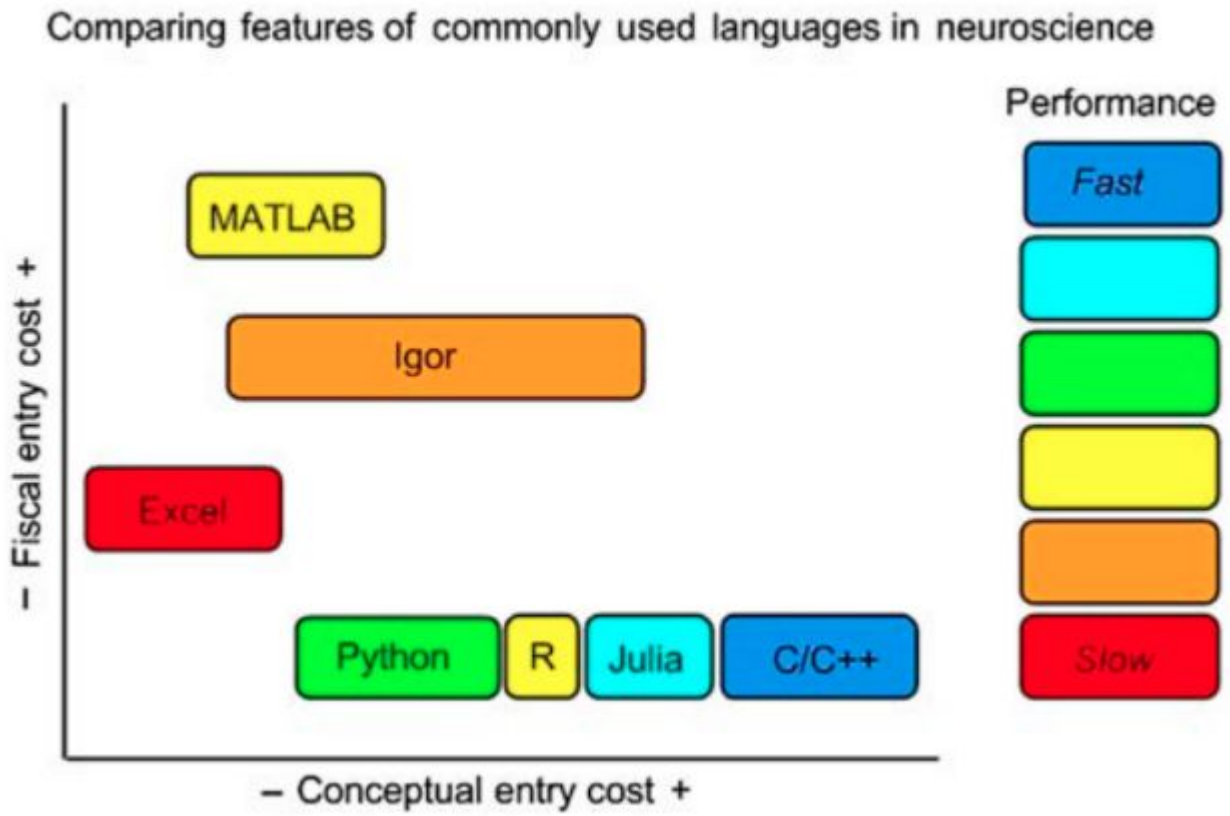
Objectives for this morning

- Introduce Jupyter Notebooks & Python syntax
- 01/02 - Types of variables and data structures
- 03 - Booleans, conditionals, functions, and for loops
- 04 - Object-oriented programming

This is a lot. If this is your first exposure to programming, it'll feel quick. That's okay!

Considerations for choosing a programming language

- Fiscal & conceptual entry
- Usage in particular field or profession



From Wallisch (2017)

Here, we'll use Python

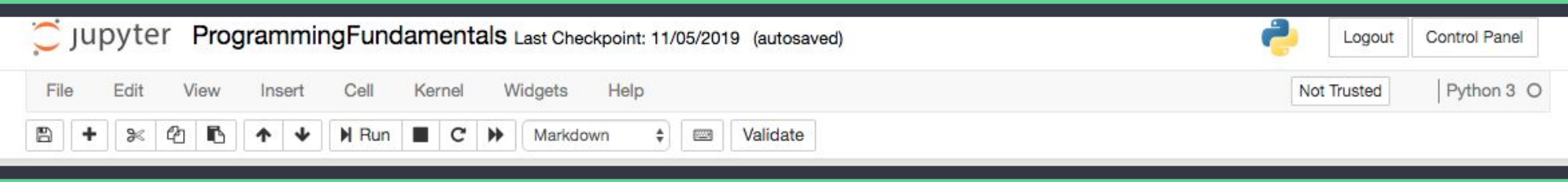
- Programming language, development led by Python Software Foundation (www.python.org)
- Uses concise structure & wording similar to human language
- Can be used for many purposes, from web programming, to creating games, to analyzing & visualizing data
- Works well with free "Notebook" like computing interfaces, such as Jupyter Notebooks, Binder, or Google Colaboratory

we will use these today

python™

colab

Introduction to the UCSD DataHub & Jupyter Notebooks



To accessing the notebooks for this lecture *(and all future lectures)*:

- Click this link: https://datahub.ucsd.edu/hub/user-redirect/git-sync?repo=https://github.com/NGP-Bootcamp/Bootcamp2020
- Open the NEU210 container 💃

About Jupyter Notebooks

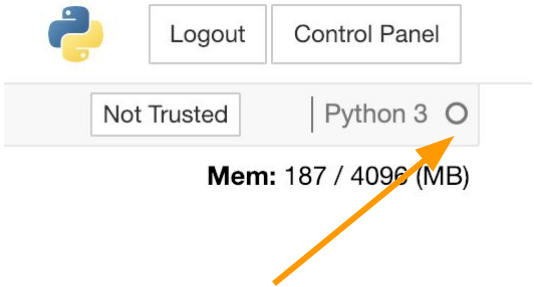
- Jupyter is a loose acronym for Julia, Python, and R
- Run in a web browser!
- Usefully, it will show plots directly in the notebook as you work your way through, performing analyses in real-time
- Two main components:
  - **Kernel**: this is where we actually run the code
  - **Dashboard**: landing page where you can see the notebooks you've created

Using Jupyter Notebooks

- **Cell**: the main organizational structure of the notebook
  - Use **Shift+Enter** to run a cell (or press Run)
  - You can run cells out of order, and move cells around!
  - Cells can be **code** (the default) or **markdown** (descriptive text or images)
    - Code cells have `In[ ]:`
      - If there is a star (`In[]*:`), that means your cell is running
    - Change between code & markdown using dropdown menu (or keyboard shortcuts)
    - Turns **green** in edit mode

Using Jupyter Notebooks (continued)

- Processing-intensive cells will take > 10 seconds to run, but your code may also get stuck in a cell.
  - Interrupt a stuck cell using Kernel > Interrupt
- **If you change anything in the cell, you need to re-run it.**
- For help:
  - Help > User Interface tour
  - Help > Keyboard Shortcuts



You can tell if the kernel is busy by whether or not the circle next to Python 3 (upper right corner) is filled or not. (filled = busy)

Syntax Rules in Python

- Python is ***case sensitive*:** letsroll ≠ LetsRoll
- White space does not matter (e.g., line 9 or 11 below)
- Indentation *does* matter — use **tab** to indent your code
- **Indexing** (we'll come back to this later)
    - Python starts indexing at 0. So if you have a list of numbers:

      `list = [ 2 , 5 , 7 , 1 , 9 , 2 ]`

      and you ask for `list[1]`, you'll get 5.
- Code that follows # is not read by the interpreter:

```
 8 # this is a commented line!
 9
10 print('this line will totally run.')
11
```

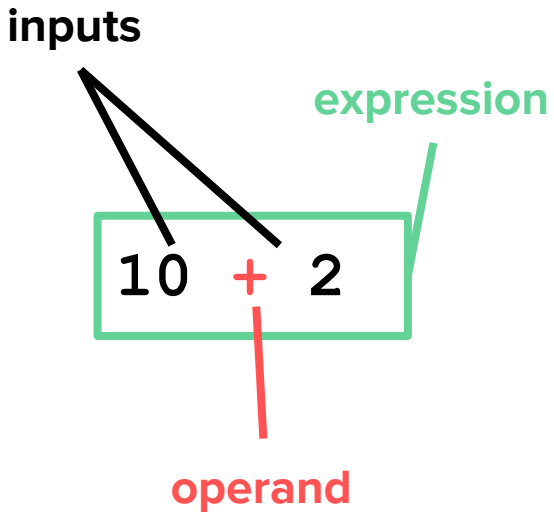
MATLAB vs Python Syntax

	Matlab	Python
Element index	1	0
Comment	%	#
Print variable contents to screen	disp(x)	print(x)
Print string	'Hello Everyone!'	print "hello Everyone!"
Find help on a function	help func	Help(func)
Script file extension	.m	.py
Import library functions	Must be in MATLABPATH	from func import *
Matrix dimensions	size(x)	x.shape
Line continuation	...	\

From this slide deck (UC Boulder)

Basic arithmetic operators in Python

Symbol	Operation	Usage
+	Addition	`10+2`
-	Subtraction	`10-2`
*	Multiplication	`10*2`
/	Division	`10/2`
**	Exponent	`10**2`
%	Modulo	`10%2`



If you want a whole number (floor division), use // instead.

Basic arithmetic operators in Python

- The default order of operations is the same as in mathematics! (PEMDAS)
- Use parentheses to specify that you want an operation to happen first.

Storing values

We can store values in variables, e.g.:

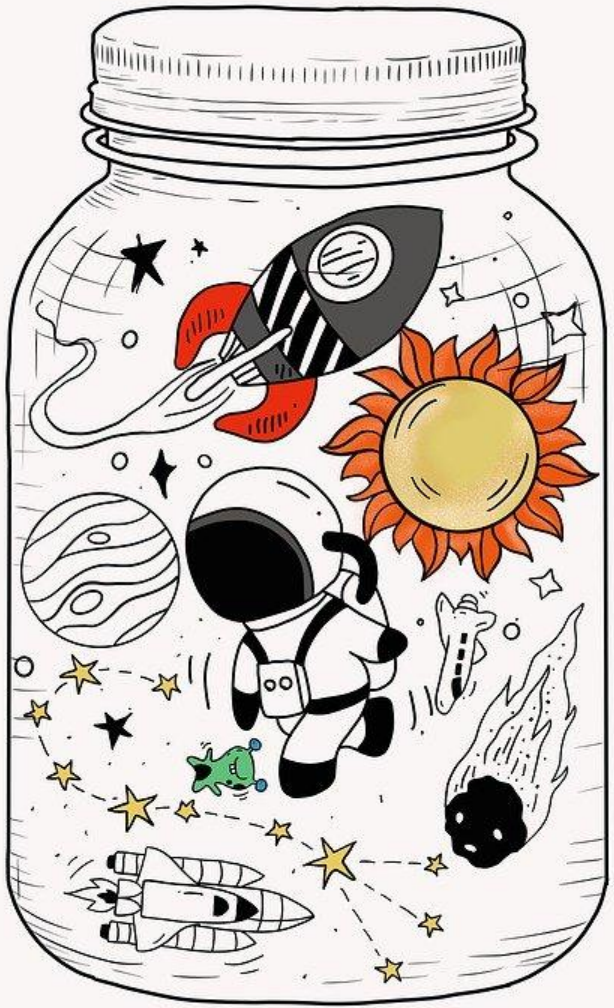
```
variable_1 = 48
```

name

value

Variables can be text, integers, or floats (with decimals), e.g.:

```
text_string = “hello”
```



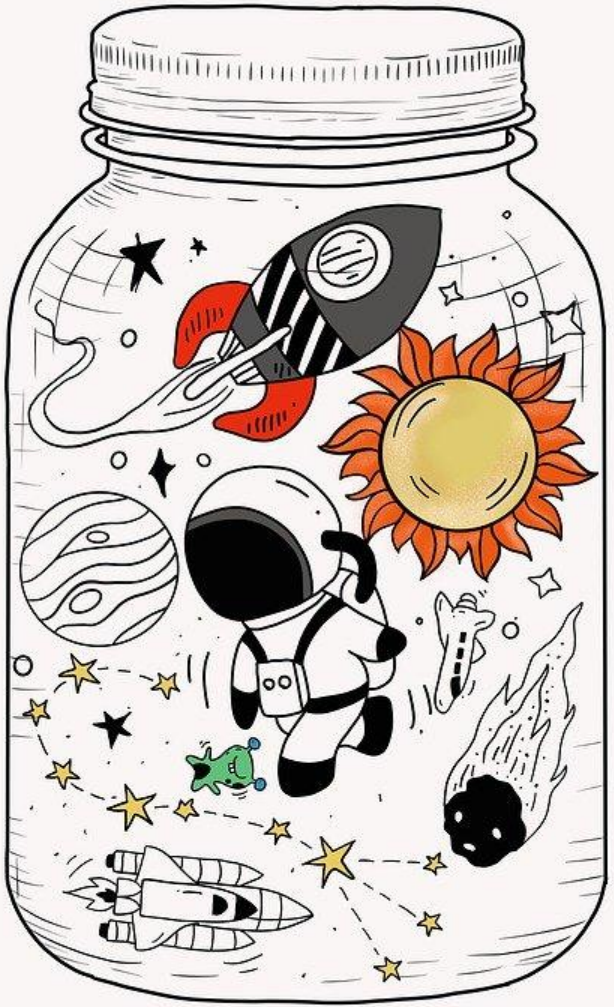
Storing values

We can store values in variables, e.g.:

```
variable_1 = 48
```

We use an equal sign to *assign* the value to a name, but it's not the same thing as saying they are equal.

In other words, we're storing that value in the variable. (Think of them like cookie jars)



Creating new variables

- Names are always on the left of the `=`, values are always on the right
- Pick names that describe the data / value that they store
- Make variable names as **descriptive** and **concise** as possible (this is an art!)
- Variables cannot be Python keywords:

```
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def',
 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lamb
da', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
>>>
```

- They also cannot start with a number or contain special characters except for an underscore

Objectives for this morning

- Introduce Jupyter Notebooks & Python syntax
- 01/02 - Types of variables and data structures
- 03 - Booleans, conditionals, functions, and for loops
- 04 - Object-oriented programming

Python has many variable types, and each function a little bit differently.

Understanding your variable type is crucial for working with it.

Built-in simple variable types in Python

Type	Example	Description
`int`	x = 1	integers (i.e., whole numbers)
`float`	x = 1.0	floating-point numbers (i.e., real numbers)
`complex`	x = 1 + 2j	Complex numbers (i.e., numbers with real and imaginary part)
`bool`	x = True	Boolean: True/False values
`str`	x = 'abc'	String: characters or text
`NoneType`	x = None	Special object indicating nulls

Integers, strings, floats

- **Integers** (`int`): any whole number
- **Float** (`float`): any number with a decimal point (floating point number)
- **String** (`str`): letters, numbers, symbols, spaces
  - Represented by matching beginning & ending quotes
  - Quotes can be single or double; use single *within* double
  - Use \ to ignore single quote
  - Concatenate strings with +

Checking variable types

This is a very useful troubleshooting step!

- You can check what type your variable (**a**) is by using **`type(a)`**
  - Alternatively, we can use:

    >>> **`type(a) is float`**

    or

    >>> **`isinstance(x,float)`**
- Python lets you change the type of variables, however, ***you cannot combine types.***
- Use **`del`** to delete variables

Python has different ways to store data: **lists**, **tuples**, **dictionaries**, and **sets.**

These differ in their **syntax, mutability,** and use cases.

Sets are a mutable collection of distinct (unique) immutable values that are unordered. *(We're not going to work with sets.)*

Lists are flexible & efficient containers for heterogeneous data

- Lists are **mutable**: we can change individual elements of the list
- Denoted by brackets & elements are separated by commas

```
my_list = ['apples', 'bananas', 'oranges']
```

Let's do this in the Jupyter Notebook!

Corresponding notes are here for your reference.

- Check the length of your list by using `len(my_list)`
- Use `my_list.append()` to add elements to a list
- Remove elements by index using `del my_list[2]`
- Remove elements by value by using `my_list.remove('oranges')`
- Sort by using `my_list.sort()`

Lists are flexible & efficient containers for heterogeneous data

- Lists are **mutable**: we can change individual elements of the list
- Denoted by brackets & elements are separated by commas

```
my_list = ['apples', 'bananas', 'oranges']
```

- Check the length of your list by using **`len(my_list)`**
- Use **`my_list.append()`** to add elements to a list
- Remove elements by index using **`del my_list[2]`**
- Remove elements by value by using **`my_list.remove('oranges')`**
- Sort by using **`my_list.sort()`**

Indexing lists

```
my_list = [1,2,5,2,3]

my_list[1] = 2

my_list[-1] = 3

my_list[5] =

    IndexError
```

Index number

Allows you to count from the end
(could be -2, etc.)

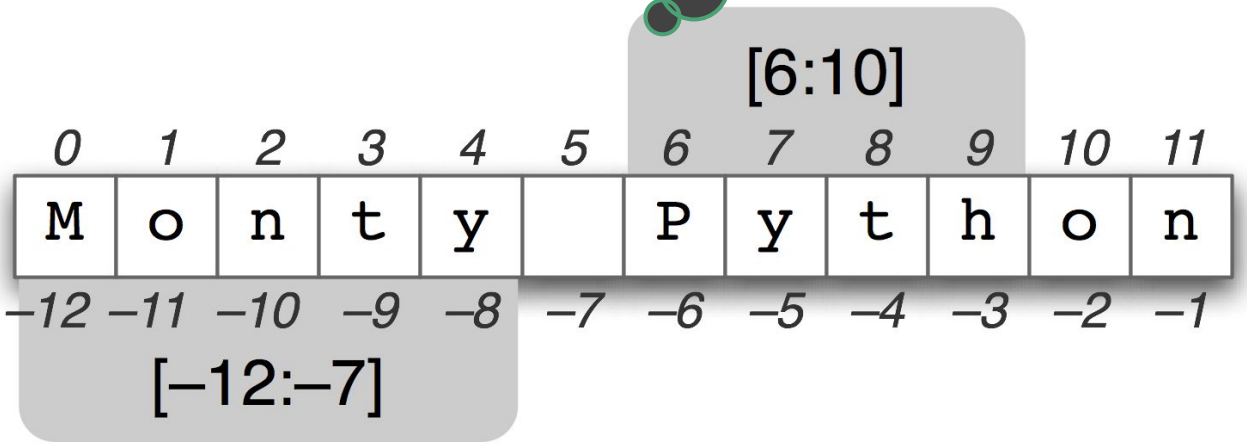
Shown if you try to get an index
that doesn't exist

Slicing lists

```
my_list[0:2]

my_list[1:3]

my_list[:3]

my_list[3:]

my_list[:]
```

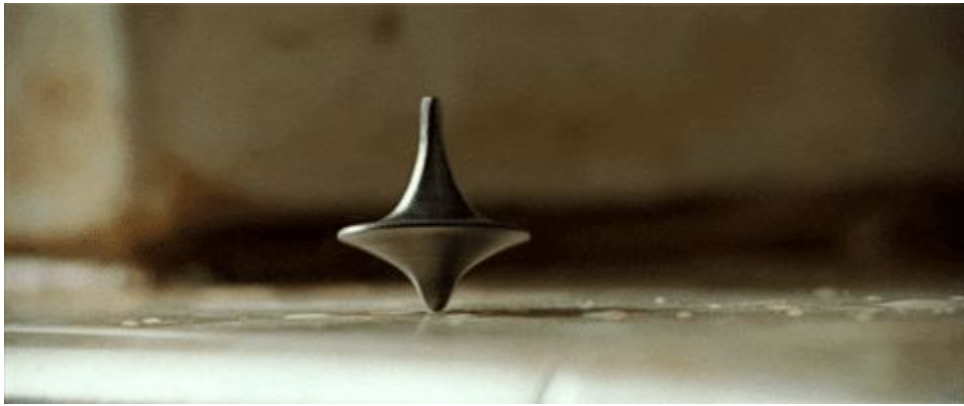
`[included:excluded]`
It doesn't show you the stop element (it shows you elements with indices 0 & 1)

One way to remember how slices work is to think of the indices as pointing between characters, with the left edge of the first character numbered 0. Then the right edge of the last character of a string of n characters has index n.



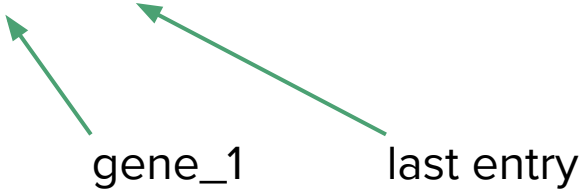
Lists of lists

```
>>> gene_1 = [‘gene1’,0.48,0.55]
>>> gene_2 = [‘gene2’,0.38,0.85]
>>> gene_3 = [‘gene3’,0.21,0.81]
>>> all_genes = [gene_1, gene_2, gene_3]
>>> print(all_genes[0][-1])
```



Lists of lists

```
>>> gene_1 = ['gene1',0.48,0.55]
>>> gene_2 = ['gene2',0.38,0.85]
>>> gene_3 = ['gene3',0.21,0.81]
>>> all_genes = [gene_1, gene_2, gene_3]
>>> print(all_genes[0][-1])


>>> 0.55
```



Tuples

- A tuple is an **immutable** collection of ordered items, that can be of mixed type.
- Tuples are created using parentheses.
- Indexing works similar to lists.

```
>>> my_tuple = ( 3, ‘blue’, 54.1)
```

Dictionaries link names to values

- Denoted by **curly braces** and elements are separated by **commas**. Assignments are done using **colons**.

```
>>> capitals =  {'US' : 'DC' , 'Spain' :
'Madrid' , 'Italy:'Rome'}

>>> capitals['US']

>>> 'DC'
```

- You'll get a Key Error if you ask for a key that doesn't exist
    - Use **`'Germany' in capitals`** to check

Working with dictionaries in Python

- Use **`capitals.update(morecapitals)`** to add another dictionary
- Use **`del capitals['US']`** to delete entries
- Loop by key or values, or both

When dictionaries are useful

1. Flexible & efficient way to associate labels with heterogeneous data
2. Use where data items have, or can be given, labels
3. Appropriate for collecting data of different kinds (e.g., name, addresses, ages)

Objectives for this morning

- Introduce Jupyter Notebooks & Python syntax
- 01/02 - Types of variables and data structures
- 03 - Booleans, conditionals, functions, and for loops
- 04 - Object-oriented programming

Basic conditional operators in Python

Symbol	Operation	Usage	Outcome
==	Is equal to	`10==5*2`	`True`
!=	Is not equal to	`10 != 5*2`	`False`
>	Is greater than	`10 > 2`	`True`
<	Is less than	`10 < 2`	`False`
>=	Greater than *or* equal to	`10 >= 10`	`True`
<=	Less than *or* equal to	`10 <= 10`	`True`

Boolean variables store `True` (1) or `False` (0) and are the basis of all computer operations.

Syndey Padua:
https://sydneypadua.com/2dgoggles/happy-200th-birthday-george-boole/

`if` statements syntax

```
if condition:
    print('condition met')
    print('nice work.')
print('not in the block')
```

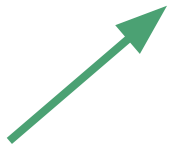
you need a colon here!

indented
by 4 spaces
(or tab)

block

`if/else` statement syntax

```
if condition:

        print('condition met')

        print('nice work.')

else:

        print('condition not met')
```



you need a colon here!

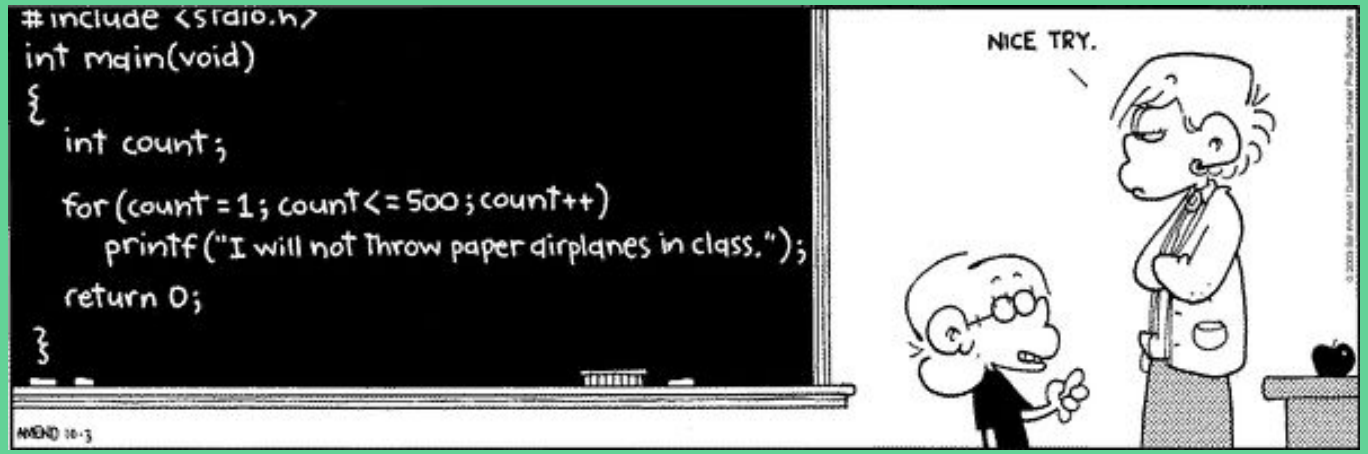
One more function: **`elif`**

- Short for “else if”
- Enables you to check for additional conditions.

```
condition_1 = False
condition_2 = True

if condition_1:
    print('Condition 1 is true.')
elif condition_2:
    print('Condition 2 is true.')
else:
    print('Both Condition 1 and 2 are false.')
```

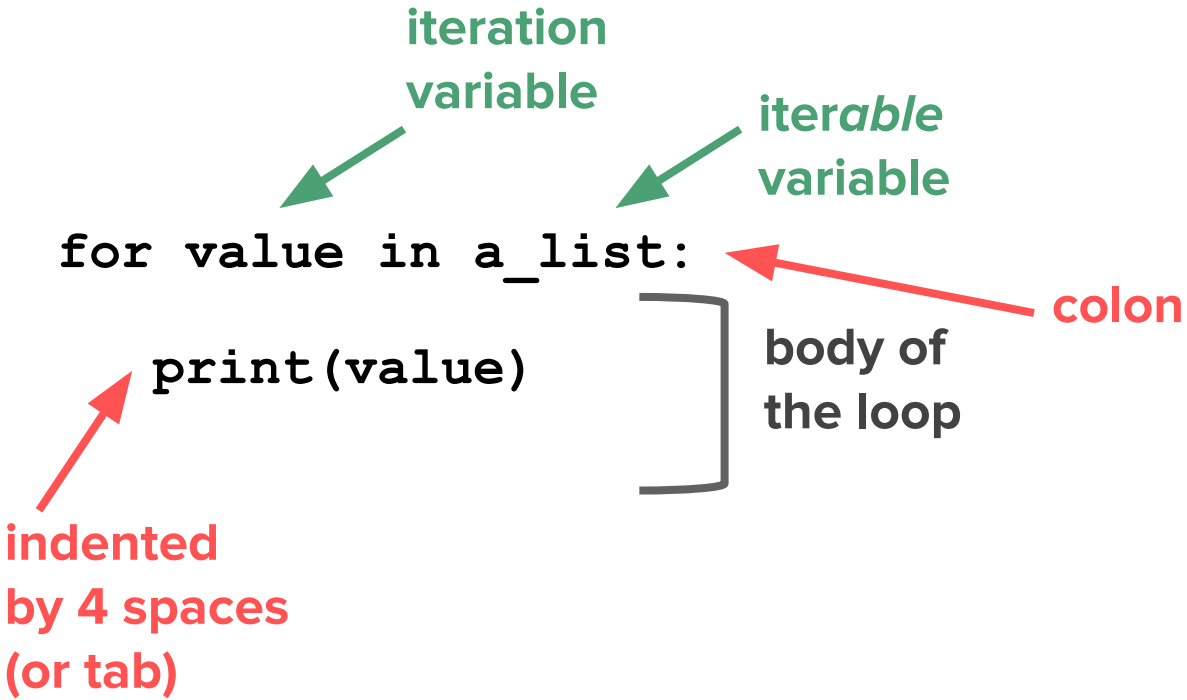
Loops enable you to re-run blocks of code for as many times as you need.

Python has two main ways to run loops: **`while`** & **`for`**



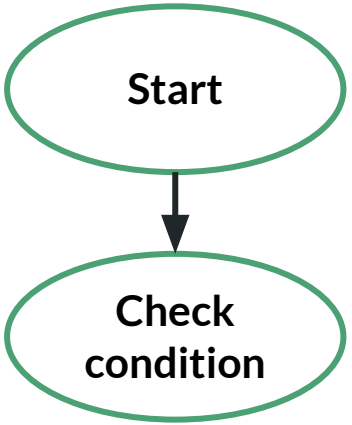
Bill Amend, FoxTrot, October 3, 2003

`for` loop syntax



`for` loop syntax

```
a_list = [1,2,3]

for value in a_list:

    print(value)
```

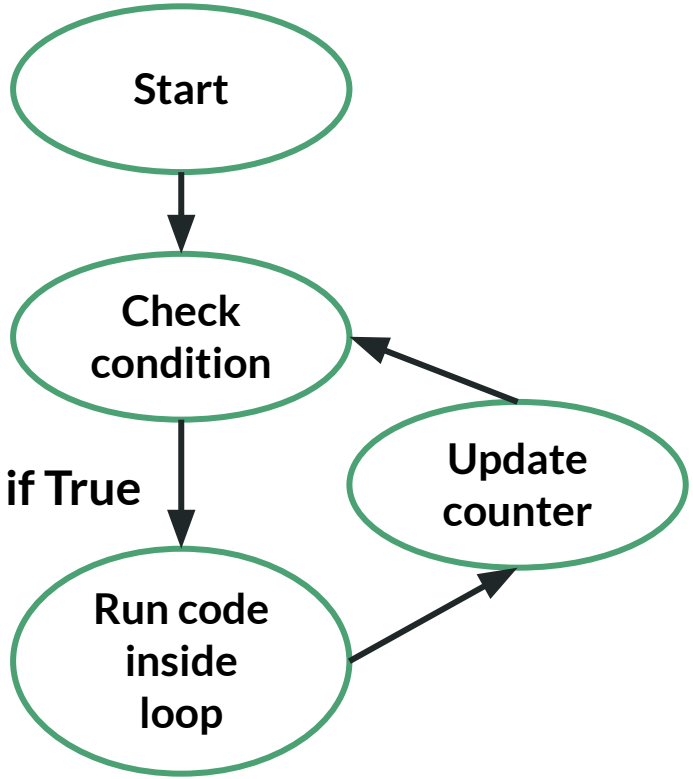


`for` loop syntax

```
a_list = [1,2,3]

for value in a_list:

    print(value)
```

```
1
```

output

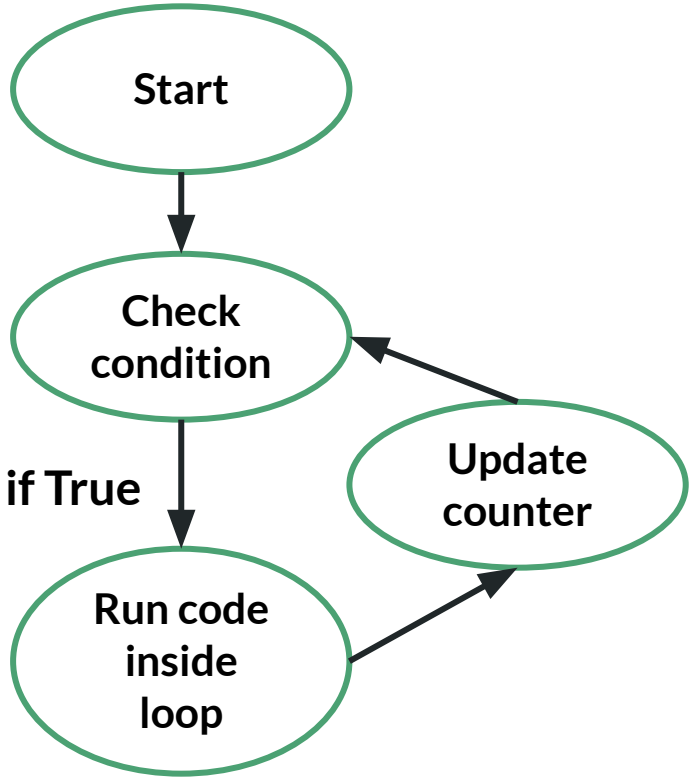


`for` loop syntax

```
a_list = [1,2,3]

for value in a_list:

    print(value)
```

```
1
2
```

output

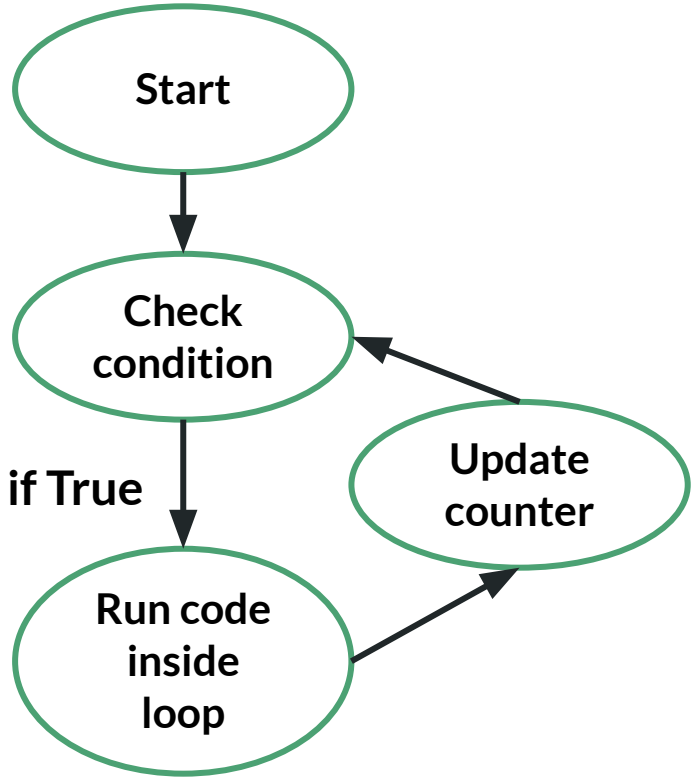


`for` loop syntax

```
a_list = [1,2,3]

for value in a_list:

    print(value)
```

```
1
2
3
```

output

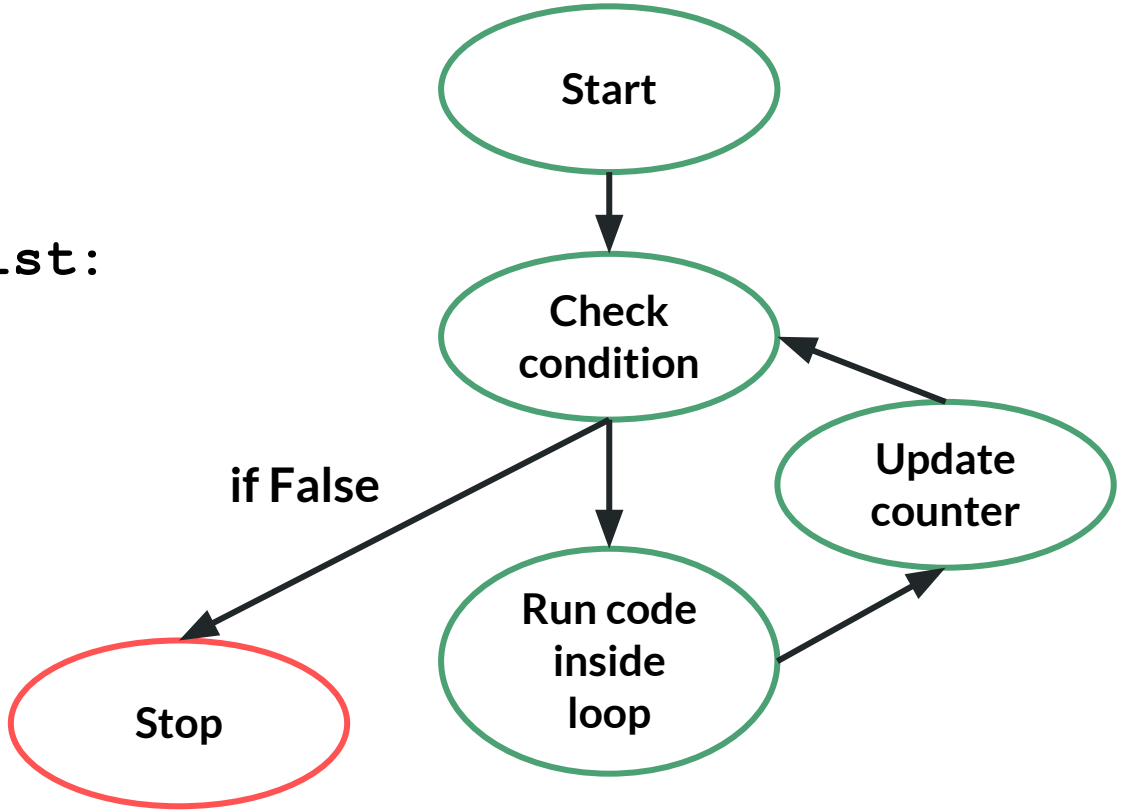


`for` loop syntax

```
a_list = [1,2,3]

for value in a_list:

    print(value)
```

```
1
2
3
```

output



efficiency benefit of `for` loops

Each of these would accomplish the same thing:

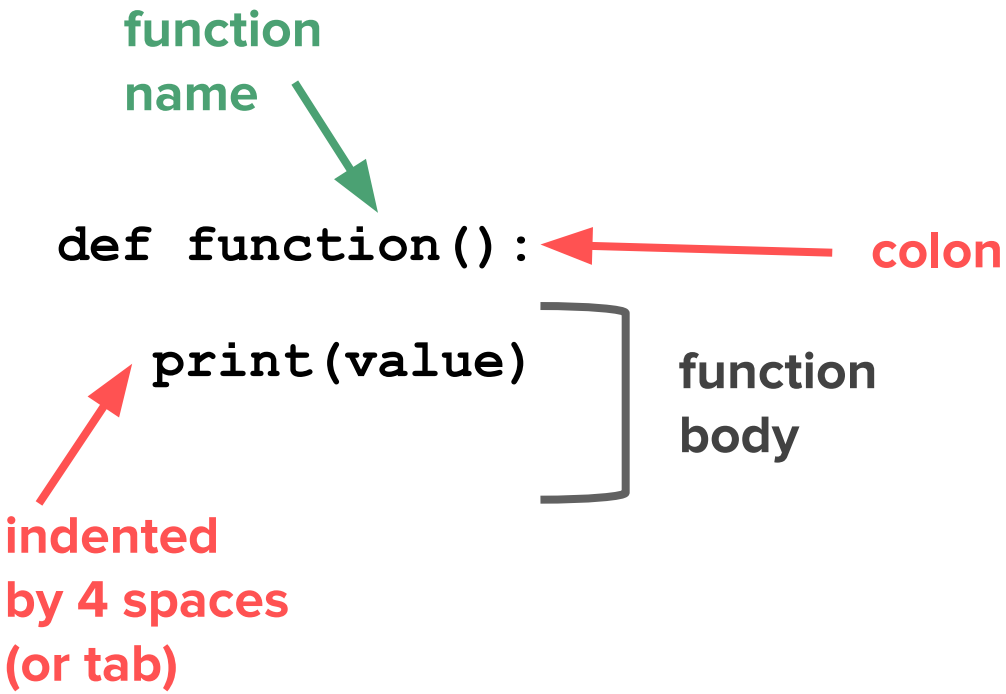
Option #1: 2+ lines of code

```
for value in a_list:

    print(value)
```

Option #2: as many lines of code as there are list entries

```
print(a_list[0])
print(a_list[1])
print(a_list[2])
...
```

`function` syntax



(advanced) **`function`** syntax

arguments (these can be variables or default arguments)

```
def function(b):

    a = b**2

    return a


a = function(6)
```

`return` to retrieve a variable outside of a function *(what happens in the function stays in the function)*

call to function giving it the argument and saving the returned variable as a

Objectives for this morning

- Introduce Jupyter Notebooks & Python syntax
- 01/02 - Types of variables and data structures
- 03 - Booleans, conditionals, functions, and for loops
- 04 - Object-oriented programming

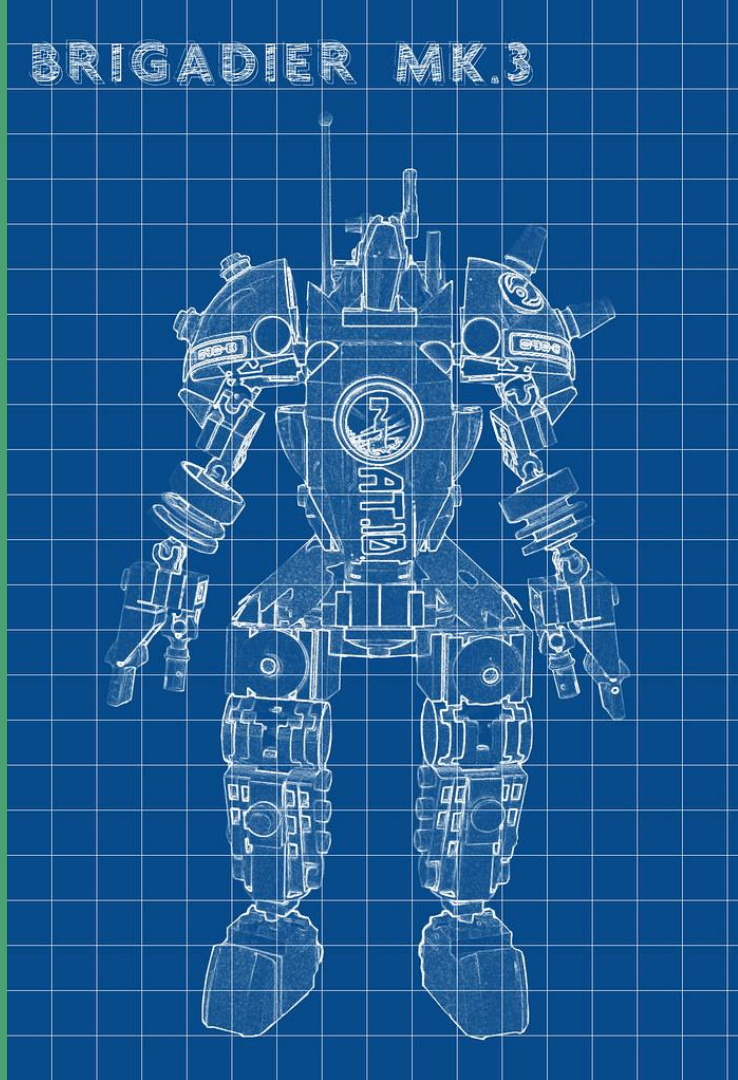
Everything in Python is an **object** (even functions!)

Objects come in different **classes**.*

- An **object** is an entity that stores data.
- An object's **class** defines specific properties objects of that class will have.
- An **instance** is a separate object of a certain **class**

* We've been referring to different "**types**" (e.g., integers, tuples, dictionaries) but even these can be called **classes**.

Think of **classes** as the blueprint for creating and defining objects and their properties (methods, attributes, etc.). They keep related things together and organized.



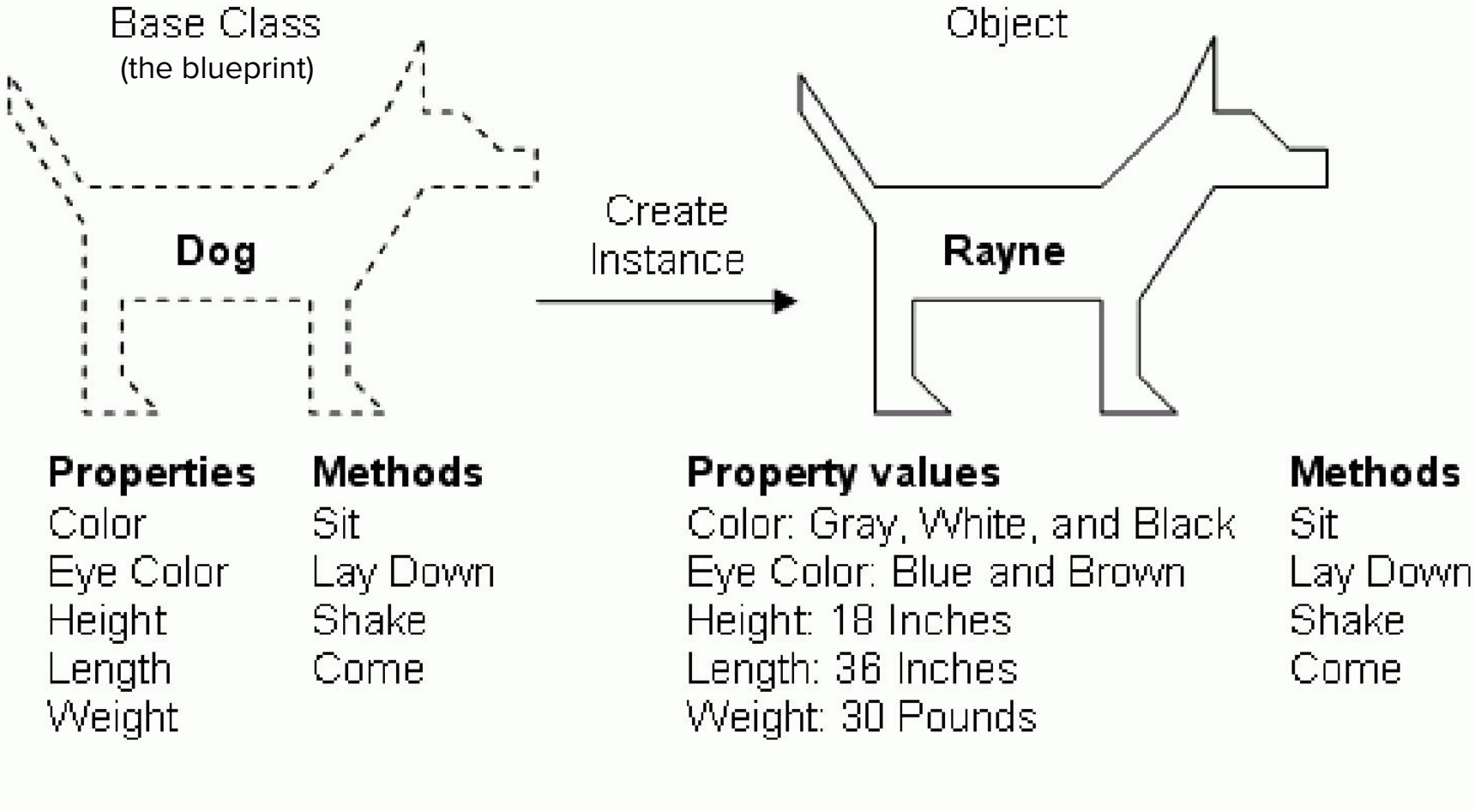


Image: http://justsajid.com/skills/objects-everything-is-an-object-in-csharp/

Objects are an organization of data (**attributes**), with associated code to operate on that data (functions defined on the objects, called **methods**).

Classes

A class is defined almost like a function, but using the **`class`** keyword.

The class definition usually contains a number of class method definitions (a function in a class).

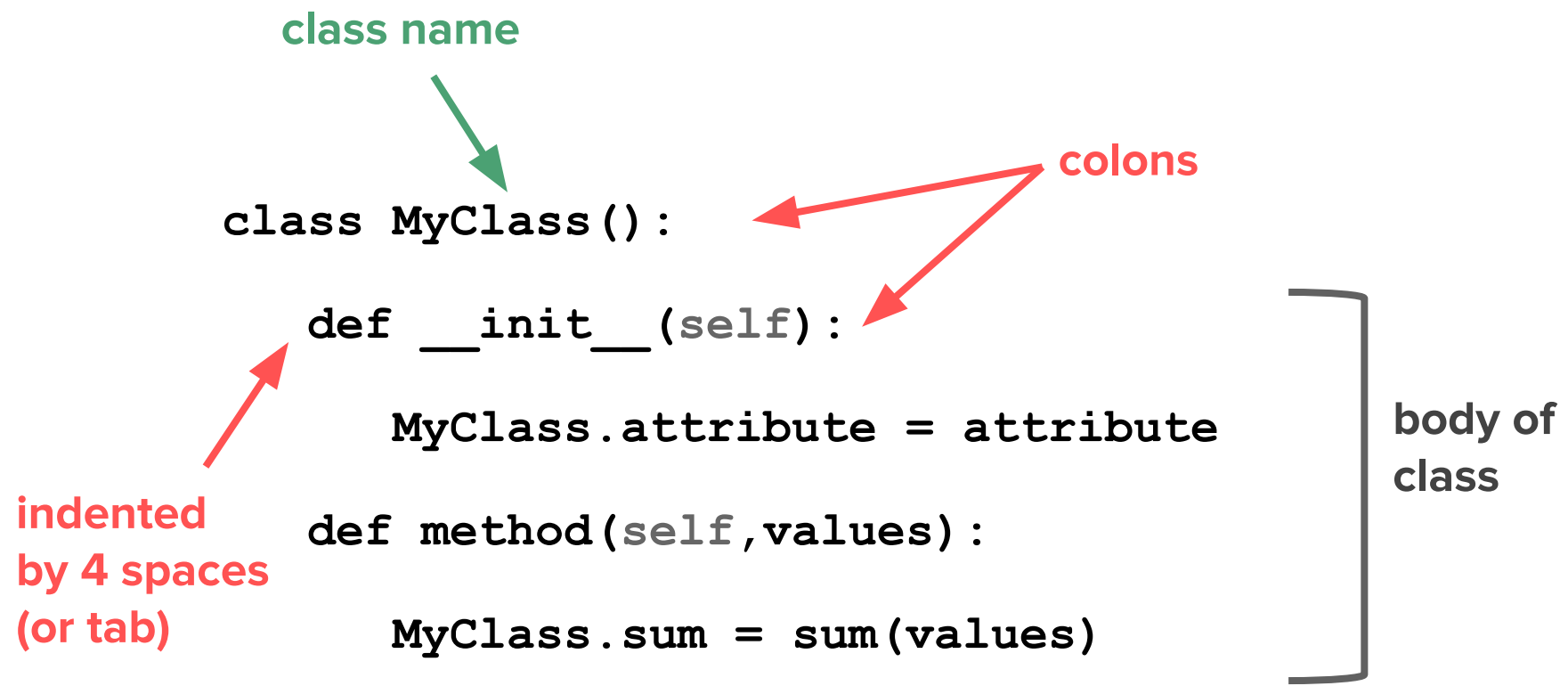
- Each class method should have an argument **`self`** as its first argument. This object is a self-reference.
- Some class method names have special meaning, for example:
  - **`__init__`**: The name of the method that is invoked when the object is first created.
  - (Full list here)

Case conventions in Python

- Style conventions (often called **style guides**) are useful ways to recognize different types of objects in Python, and can help you understand other people's codes
- Variables and functions are typically in **snake_case** (e.g., **`my_variable`**)
- Classes are in **PascalCase** (e.g. **`MyClass`**)
    - Sometimes called camel case, but more accurately, camel case is: **camelCase**

Full Python style guide here: https://www.python.org/dev/peps/pep-0008/

`class` syntax



Resources

DataQuest.Io: Free, solid introductory tutorials

Software Carpentries Lists

Python 101: Lists, Tuples, and Dictionaries

Whirlwind Tour of Python: Built-In Data Structures

Differences between MATLAB & Python

Transitioning from MATLAB to Python

Slides on Python, with an emphasis on transitioning from MATLAB:
https://www.fz-juelich.de/ias/jsc/EN/Expertise/Services/Documentation/presentations/presentation-matlab2python_table.html?nn=362392

In depth paper with descriptions on many practical applications:
https://www.enthought.com/wp-content/uploads/Enthought-MATLAB-to-Python-White-Paper.pdf