

# Python Journey

## Session 1

# Welcome!



Welcome to the **Python Journey** Series for Apple!

## About the Program

### 10 Sessions

- 1: Getting Started with Python
- 2: Applying Python – The Basics
- 3: Exploring Python Files, Dictionaries, Sets & Functions
- 4: Expanding Python – Functions, Error Handling, Importing and OO Classes

Quick Logistics: Format, Q&A and Follow-On Materials / Hand-Outs

About Me: Dr. Ernesto Lee

# Today's Agenda: Session 1



## Get started with Python!

**Leave with an understanding of how to create a sandbox environment so that you can practice Python at any time!**

- How to get started with Python
- Tools and environments for writing Python code
- How and Why we test software

## Topics We'll Start Exploring Today:

### An Overview of Python

- What is python?
- Python Timeline
- Advantages/Disadvantages of Python
- Getting help with pydoc

### The Python Environment

- Starting Python
- Using the interpreter
- Running a Python script
- Python scripts on Unix/Windows
- Editors and IDEs

### Getting Started

- Using variables
- Builtin functions
- Strings
- Numbers
- Converting among types
- Writing to the screen
- Command line parameters

The background image shows a person's hands typing on a laptop keyboard. The laptop screen displays a web browser with multiple tabs. A semi-transparent grey diagonal bar covers the upper left portion of the image, and a solid orange vertical bar is on the right side. The title 'Python Overview' is centered in a white serif font.

# Python Overview

# About Python

This lesson covers

- Why and what is Python?
- What Python does really well
- What Python doesn't do as well
- Why learn Python 3?



# Why use Python?

- Choosing a language to learn is difficult.
- Python is a good choice for many programming problems, (especially data analytics)

```
1 a = 1
2 while a < 7 :
3     if(a % 2 == 0):
4         print(a, "is even")
5     else:
6         print(a, "is odd")
7     a += 1
```

code

output

variables

[www.penjee.com](http://www.penjee.com)



A portrait of Guido van Rossum, the creator of Python. He is a middle-aged man with grey hair, a grey beard, and black-rimmed glasses. He is wearing a blue t-shirt. The background is dark with some green foliage and blurred lights.

**GUIDO** VAN ROSSUM  
**Creator of Python**

# Python is easy to use

```
1 numbers = [12, 37, 5, 42, 8, 3]
2 even = []
3 odd = []
4 while len(numbers) > 0 :
5     number = numbers.pop()
6     if(number % 2 == 0):
7         even.append(number)
8     else:
9         odd.append(number)
```

---

[www.penjee.com](http://www.penjee.com)

Programmers familiar with traditional languages will find it easy to learn Python. All of the familiar constructs—loops, conditional statements, arrays, and so forth—are included, but many are easier to use in Python.



# Python is expressive

- To get an idea of how Python's expressiveness can simplify code, consider swapping the values of two variables, `var1` and `var2`.
- In a language like Java, this requires three lines of code and an extra variable:

```
int temp = var1;  
var1 = var2;  
var2 = temp;
```



# Python is expressive

- Python lets you make the same swap in one line and in a way that makes it obvious that a swap of values has occurred:

```
var2, var1 = var1, var2
```

- Of course, this is a very simple example, but you find the same advantages throughout the language.

# Python is readable

```
# Perl version.
sub pairwise_sum {
    my($arg1, $arg2) = @_;
    my @result;
    for(0 .. $#arg1) {
        push(@result, $arg1->[$_] + $arg2->[$_]);
    }
    return(\@result);
}
```

```
# Python version.
def pairwise_sum(list1, list2):
    result = []

    for i in range(len(list1)):
        result.append(list1[i] + list2[i])
    return result
```



# Python is complete — “batteries included”

- For example, with Python, you can write a web server to share the files in a directory with just two lines of code:

```
import http.server  
http.server.test(HandlerClass=http.server.SimpleHTTPRequestHandler)
```

- There's no need to install libraries to handle network connections and HTTP; it's already in Python, right out of the box

# Python is cross-platform

- Python is also an excellent cross-platform language. Python runs on many platforms: Windows, Mac, Linux, UNIX, and so on.
- Because it's interpreted, the same code can run on any platform that has a Python interpreter, and almost all current platforms have one.





# Python is free

- Python is free.
- Python was originally, and continues to be, developed under the open source model, and it's freely available.

The word "FREE" is rendered in a bold, green, 3D block font. The letters are thick and have a slight shadow underneath, giving them a three-dimensional appearance. The font is sans-serif and very stylized.

# What Python doesn't do as well



- Although Python has many advantages, no language can do everything, so Python isn't the perfect solution for all your needs.

# Python isn't the fastest language

- A possible drawback with Python is its speed of execution.
- It isn't a fully compiled language. Instead, it's first compiled to an internal bytecode form, which is then executed by a Python interpreter.



# Python doesn't have the most libraries



- Although Python comes with an excellent collection of libraries, and many more are available, Python doesn't hold the lead in this department.
- Languages like C, Java, and Perl have even larger collections of libraries available

# Python doesn't check variable types at compile time

- It's possible to use the variable `x` to refer to a string in one line and an integer in another:

```
>>> x = "2"
```

```
>>> x
```

```
'2'
```

```
>>> x = int(x)
```

```
>>> x
```

```
2
```

← **x is string "2"**

← **x is now integer 2**



# Python doesn't have as much mobile support

- In the past decade the numbers and types of mobile devices have exploded, and smartphones, tablets, phablets, Chrome courses, and more are everywhere, running on a variety of operating systems.
- Python isn't as strong player in this space.



# Python doesn't use multiple processors well

- The standard implementation of Python isn't designed to use multiple cores, due to a feature called the global interpreter lock (GIL).



# Why learn Python 3?

- In earlier versions of Python, for example, the print statement didn't require parentheses around its arguments:

`print "hello"`

- In Python 3, print is a function and needs the parentheses:

`print("hello")`



# Quick Review

- Python is a modern, high-level language with dynamic typing and simple, consistent syntax and semantics.
- Python is multiplatform, highly modular, and suited for both rapid development and large-scale programming.





# The Python Environment



# Getting started

This lesson covers

- Starting Python
- Using the interpreter
- Running a Python script
- Python scripts on Unix/Windows
- Editors and IDEs



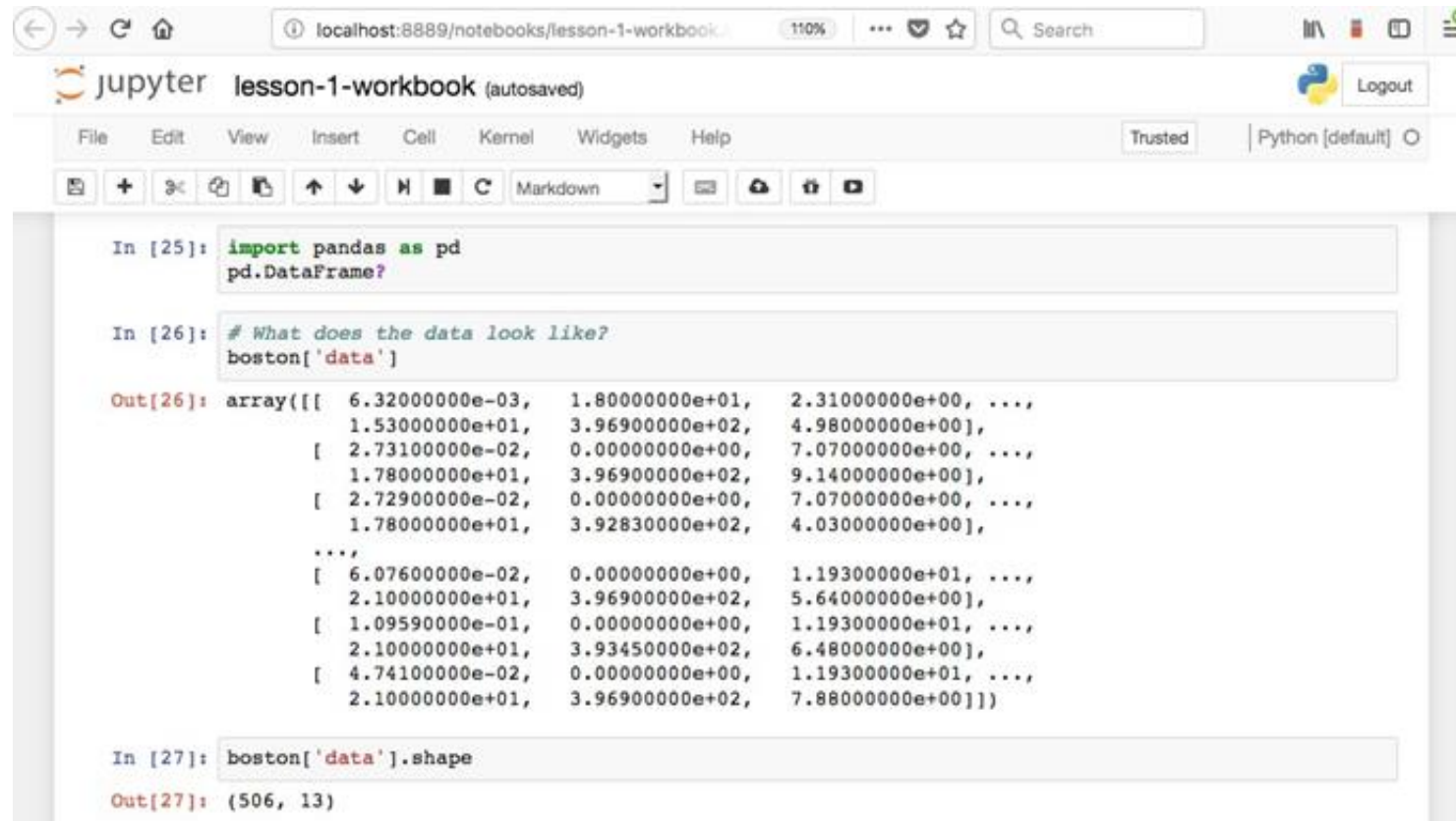
# Installing Python



- Installing Python is a simple matter, regardless of which platform you're using.
- You'll need Python v3.6 or better for this course.
- To obtain the latest approved version, please reach out to your Learning Support Team.

# Jupyter Notebooks

- You have many options for accessing interactive Python and one of the more popular IDEs is: Jupyter.



The screenshot displays a Jupyter Notebook interface in a web browser. The address bar shows the URL `localhost:8889/notebooks/lesson-1-workbook` at 110% zoom. The notebook title is `lesson-1-workbook (autosaved)`. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for adding cells, undo, redo, and other actions. The code area shows three input cells:

```
In [25]: import pandas as pd
pd.DataFrame?
```

```
In [26]: # What does the data look like?
boston['data']
```

The output for cell [26] is a large array of numerical data in scientific notation:

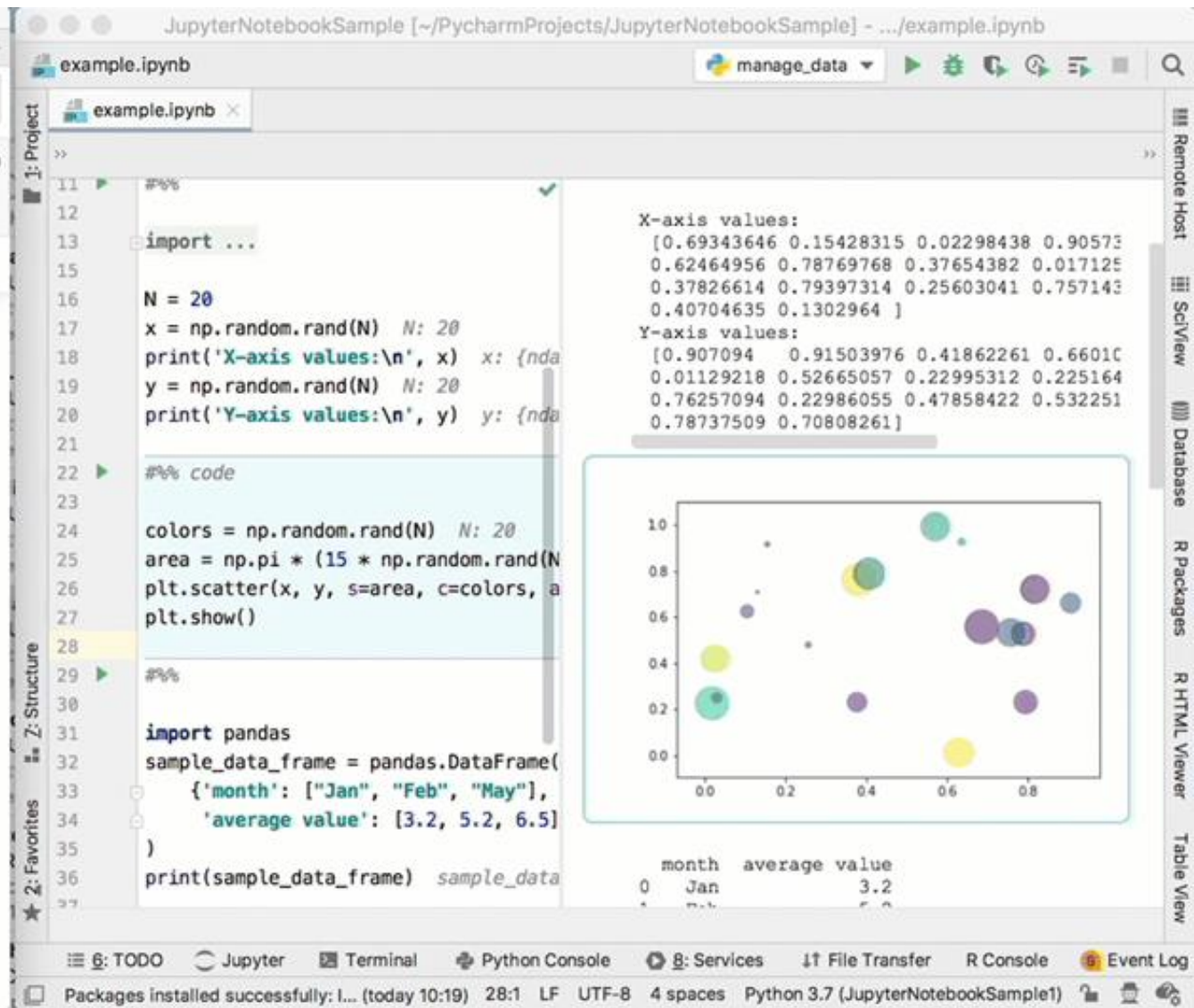
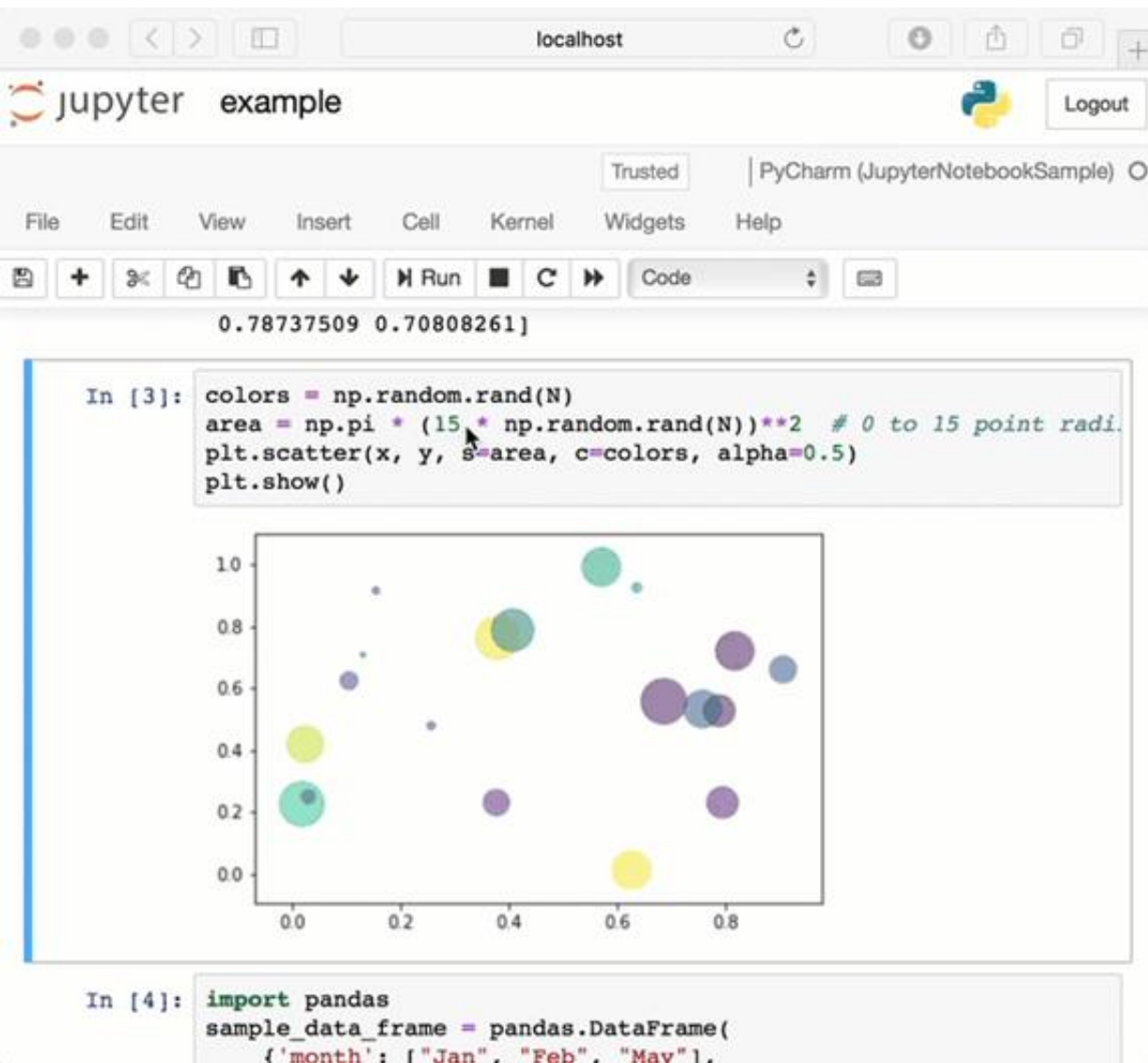
```
Out[26]: array([[ 6.32000000e-03,  1.80000000e+01,  2.31000000e+00, ...,
                  1.53000000e+01,  3.96900000e+02,  4.98000000e+00],
                 [ 2.73100000e-02,  0.00000000e+00,  7.07000000e+00, ...,
                  1.78000000e+01,  3.96900000e+02,  9.14000000e+00],
                 [ 2.72900000e-02,  0.00000000e+00,  7.07000000e+00, ...,
                  1.78000000e+01,  3.92830000e+02,  4.03000000e+00],
                 ...,
                 [ 6.07600000e-02,  0.00000000e+00,  1.19300000e+01, ...,
                  2.10000000e+01,  3.96900000e+02,  5.64000000e+00],
                 [ 1.09590000e-01,  0.00000000e+00,  1.19300000e+01, ...,
                  2.10000000e+01,  3.93450000e+02,  6.48000000e+00],
                 [ 4.74100000e-02,  0.00000000e+00,  1.19300000e+01, ...,
                  2.10000000e+01,  3.96900000e+02,  7.88000000e+00]])
```

```
In [27]: boston['data'].shape
```

The output for cell [27] is:

```
Out[27]: (506, 13)
```

# What EXACTLY are Jupyter Notebooks



# Markdown

# Markdown!

This is a basic [Markdown]  
(<https://en.wikipedia.org/wiki/Markdown>) document.

### Sub heading

It's *\*simple\**, but **\*\*powerful\*\***.

## Markdown!

This is a basic [Markdown](#) document.

### Sub heading

It's *simple*, but **powerful**.

#### The Beginner's Guide to Markdown

Imagine typing on a mechanical typewriter, with only letters and punctuation at your fingertips. There's no italics, no color options, no larger typeface for headers. How do you emphasize words, set apart quotes, and both make your document nicer looking and easier to read at the same time?

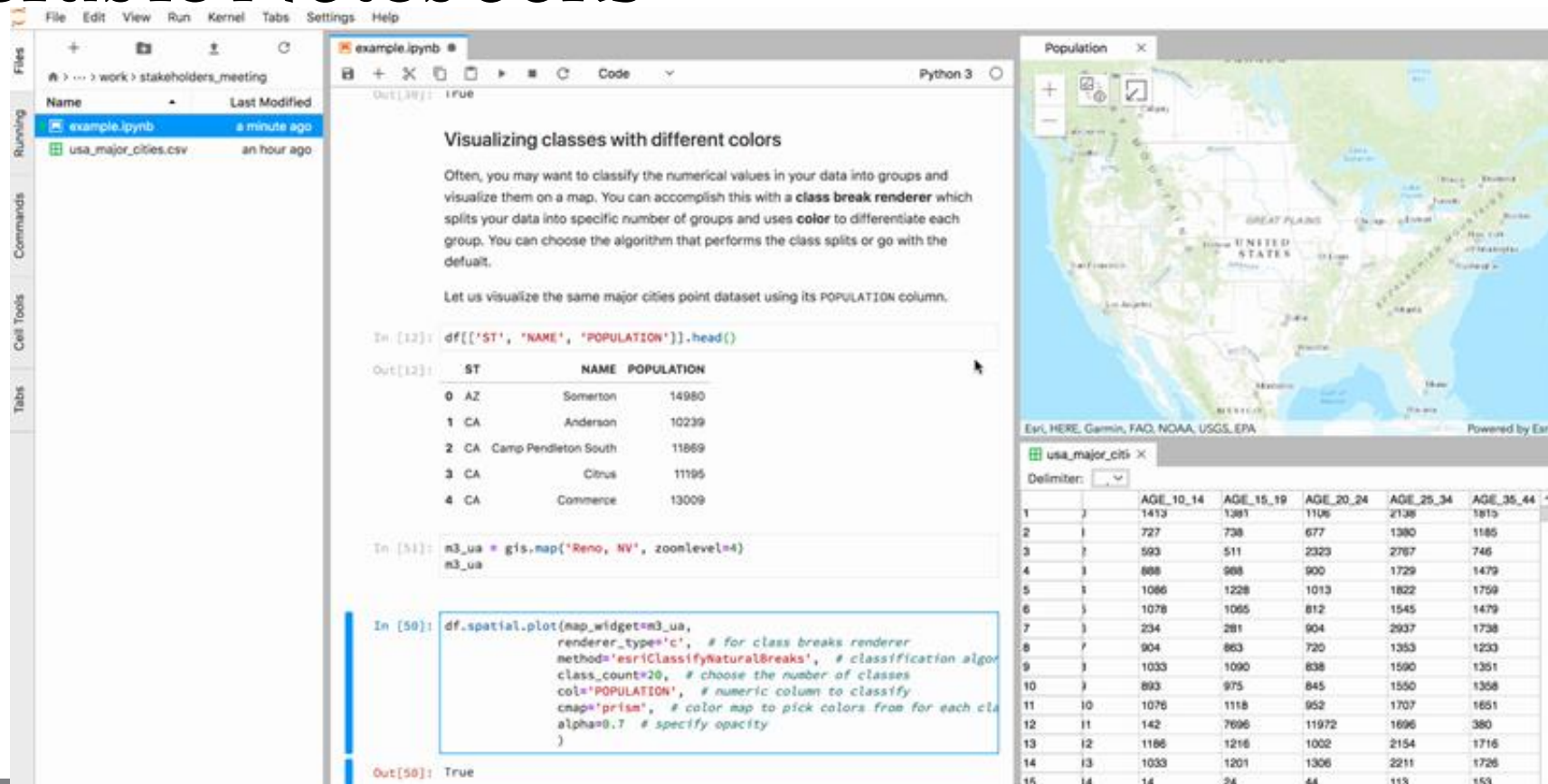
Markdown, that's how.

"The overriding design goal for Markdown's formatting syntax is to make it as readable as possible." - John Gruber



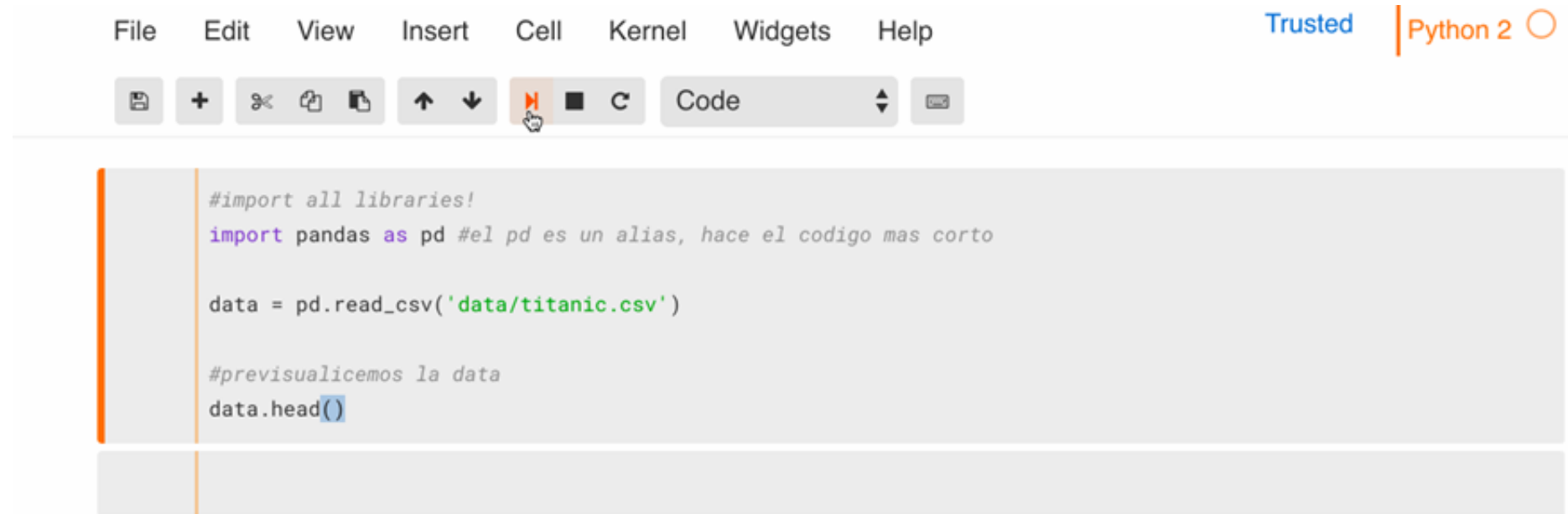
# Matter of Perspective

- Lab Style Notebooks
- Deliverable Notebooks



# Jupyter Extensibility (through Python Libs)

- NumPy
- Pandas
- Matplotlib
- Seaborn
- Scikit-learn
- Requests
- Bokeh

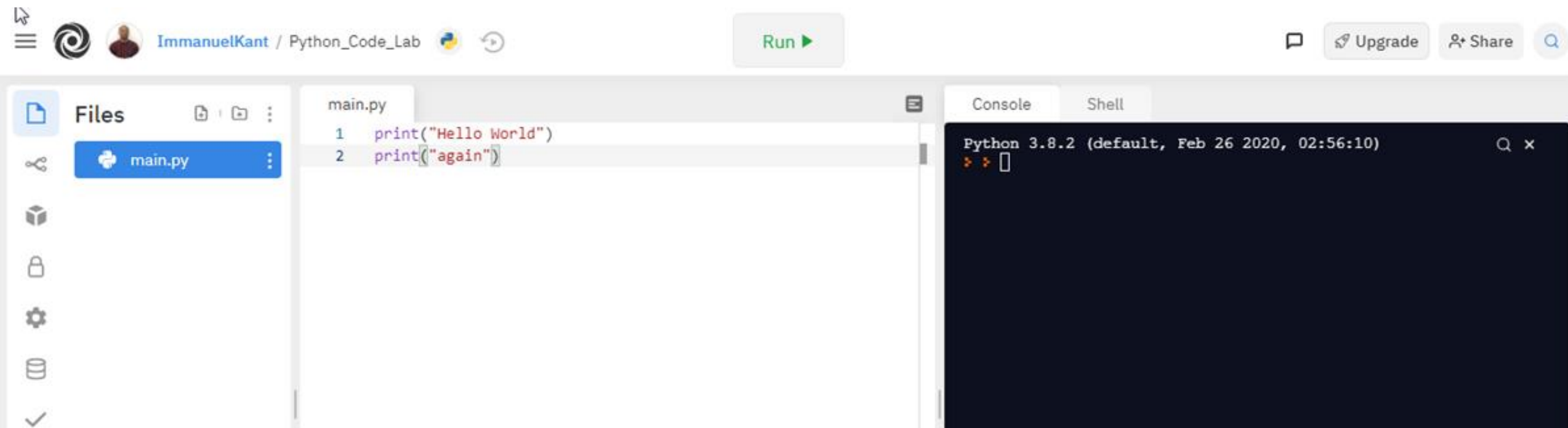


The screenshot shows a Jupyter Notebook interface. At the top, there is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar, it says "Trusted" and "Python 2" with a small orange circle icon. Below the menu bar is a toolbar with various icons for file operations, cell navigation, and execution. The main area of the notebook contains a code cell with the following Python code:

```
#import all libraries!  
import pandas as pd #el pd es un alias, hace el codigo mas corto  
  
data = pd.read_csv('data/titanic.csv')  
  
#previsualicemos la data  
data.head()
```

# The basic interactive mode

The basic interactive mode is a rather primitive environment, but the interactive examples in this course are generally small. (Jupyter will be the interface of choice)



# Hello, world

- Use Jupyter or JupyterLab for this (and all exercises and demos)
- Start with the obligatory “Hello, World” program, which is a oneliner in Python (ending each line you type with a hard return):

```
>>> print("Hello, World")  
Hello, World
```

# Using the interactive prompt to explore Python

```
>>> x = 2
>>> help(x)
Help on int object:
```

```
class int(object)
|   int(x=0) -> integer
|   int(x, base=10) -> integer
|
|   Convert a number or string to an integer, or return 0 if no arguments
|   are given.  If x is a number, return x.__int__().  For floating point
|   numbers, this truncates towards zero.
|
|   If x is not a number or if base is given, then x must be a string,
|   bytes, or bytearray instance representing an integer literal in the...
|   (continues with the documentation for an int)
```



# Using the interactive prompt to explore Python

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
  '__package__', '__spec__', 'x']
>>> dir(int)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
  '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__',
  '__float__', '__floor__', '__floordiv__', '__format__', '__ge__',
  '__getattribute__', '__getnewargs__', '__gt__', '__hash__', '__index__',
  '__init__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__',
  '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__',
  '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__',
  '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__',
  '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__',
  '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__',
  '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
  '__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_length',
  'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real',
  'to_bytes']
>>>
```



# Using the interactive prompt to explore Python

- Unlike `dir`, both `globals` and `locals` show the values associated with the objects.
- In the current situation, both functions return the same thing, so we have only shown the output from `globals()`:

```
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
  <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None,
  '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>,
  'x': 2}
```



# The Zen of Python

- In your console – type:

`import this`

# Quick Review



- Installing Python 3 on Windows systems is as simple as downloading the latest installer from [www.python.org](http://www.python.org) and running it.
- Installation on Linux, UNIX, and Mac systems will vary
- Refer to installation instructions on the Python website, and use your system's software package installer where possible.

A person's hands are shown typing on a laptop keyboard. The laptop screen displays a web application with a sidebar of colorful icons. A semi-transparent grey diagonal overlay covers the left and center portions of the image, and a solid orange vertical bar is on the right. The text "Getting Started!" is centered in a white serif font over the grey area.

# Getting Started!

# The absolute basics

This lesson covers

- Indenting and block structuring
- Differentiating comments
- Assigning variables
- Evaluating expressions
- Using common data types
- Getting user input
- Using correct Pythonic style

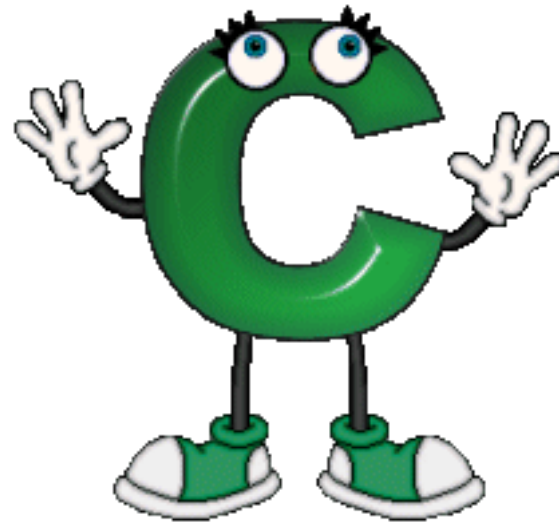


# Indentation and block structuring

- C code that calculates the factorial of 9, leaving the result in the variable r:

```
/* This is C code */
```

```
int n, r;  
n = 9;  
r = 1;  
while (n > 0) {  
    r *= n;  
    n--;  
}
```



# Indentation and block structuring

/\* And this is C code with arbitrary indentation \*/

```
int n, r;  
n = 9;  
r = 1;  
while (n > 0) {  
    r *= n;  
    n--;  
}
```





# Indentation and block structuring

- The code still would execute correctly, even though it's rather difficult to read.
- Here's the Python equivalent:

```
# This is Python code. (Yea!)
```

```
n = 9
```

```
r = 1
```

```
while n > 0:
```

```
    r = r * n
```

```
    n = n - 1
```

← Python also supports  
C-style `r *= n`

← Python also  
supports `n -= 1`

# Differentiating comments

- For the most part, anything following a # symbol in a Python file is a comment and is disregarded by the language.
- The obvious exception is a # in a string, which is just a character of that string:

```
# Assign 21 to x
```

```
x = 21
```

```
x = 7 # Now x is 7
```

```
x = "# This is not a comment"
```

# Variables and assignments

- The most commonly used command in Python is assignment, which looks pretty close to what you might've used in other languages.
- Python code to create a variable called `x` and assign the value 21 to that variable is

```
x = 21
```

# Variables and assignments

- Python variables can be set to any object, whereas in C and many other languages, variables can store only the type of value they're declared as.
- The following is perfectly legal Python code:

```
>>> x = "Hello Ernesto"
```

```
>>> print(x)
```

```
Hello Ernesto
```

```
>>> x = 21
```

```
>>> print(x)
```

```
21
```

- A new assignment overrides any previous assignments.
- The del statement deletes the variable.

```
>>> x = 21
```

```
>>> print(x)
```

```
21
```

```
>>> del x
```

```
>>> print(x)
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
NameError: name 'x' is not defined
```

```
>>>
```



# Expressions

- Python supports arithmetic and similar expressions; these expressions will be familiar to most readers.

$x = 13$

$y = 15$

$z = (x + y) / 2$





# Strings

- You've already seen that Python, like most other programming languages, indicates strings through the use of double quotes.
- This line leaves the string "Hello, World" in the variable x:

```
x = "Hello, World"
```

# Strings

- Backslashes can be used to escape characters, to give them special meanings.

    \n means the newline character

    \t means the tab character

    \\ means a single normal backslash character

    \" is a plain double-quote character

        It doesn't end the string

`x = "\tThis string starts with a \"tab\"."`

`x = "This string contains a single backslash(\\)."`

You can use single quotes instead of double quotes. The following two lines do the same thing:

```
x = "Hello, World"  
x = 'Hello, World'
```

The only difference is that you don't need to backslash " characters in single-quoted strings or ' characters in double-quoted strings:

```
x = "Don't need a backslash"  
x = 'Can\'t get by without a backslash'  
x = "Backslash your \" character!"  
x = 'You can leave the " alone'
```

You can't split a normal string across lines. This code won't work:

```
# This Python code will cause an ERROR -- you can't split the string  
across two lines.  
x = "This is a misguided attempt to  
put a newline into a string without using backslash-n"
```

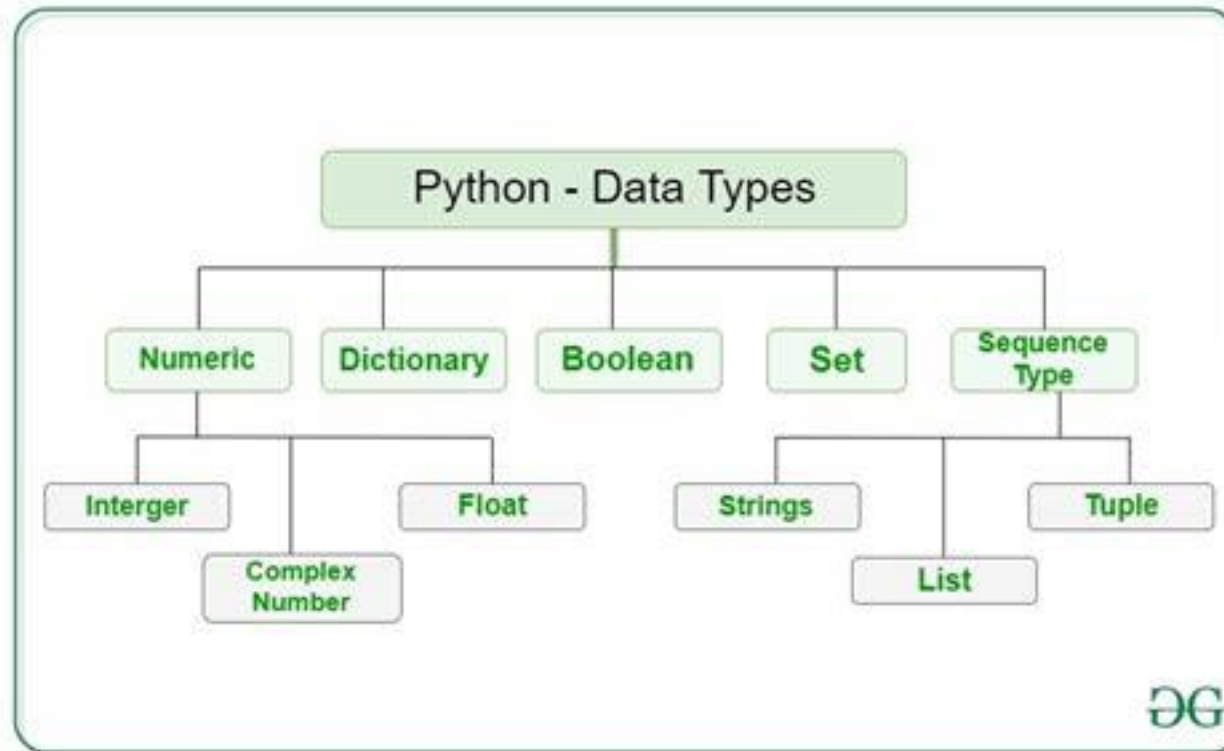
# Strings

- But Python offers triple-quoted strings, which let you do this and include single and double quotes without backslashes:

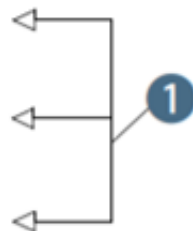
```
x = """Starting and ending a string with triple " characters  
permits embedded newlines, and the use of " and ' without  
backslashes"""
```

# Numbers

- Python offers four kinds of numbers: integers, floats, complex numbers, and Booleans



```
>>> 5 + 2 - 3 * 2
1
>>> 5 / 2          # floating-point result with normal division
2.5
>>> 5 / 2.0        # also a floating-point result
2.5
>>> 5 // 2         # integer result with truncation when divided using '//'
2
>>> 30000000000    # This would be too large to be an int in many languages
30000000000
>>> 30000000000 * 3
90000000000
>>> 30000000000 * 3.0
90000000000.0
>>> 2.0e-8         # Scientific notation gives back a float
2e-08
>>> 3000000 * 3000000
9000000000000
>>> int(200.2)
200
>>> int(2e2)
200
>>> float(200)
200.0
```





# Built-in numeric functions

- Python provides the following number-related functions as part of its core:

abs, divmod, float, hex, int, max, min, oct, pow, round

# Advanced numeric functions

```
from math import *
```

The `math` module provides the following functions and constants:

```
acos, asin, atan, atan2, ceil, cos, cosh, e, exp, fabs, floor, fmod,  
frexp, hypot, ldexp, log, log10, mod, pi, pow, sin, sinh, sqrt, tan,  
tanh
```

See the documentation for details.



# Numeric computation

- The core Python installation isn't well suited to intensive numeric computation because of speed constraints.
- But the powerful Python extension NumPy provides highly efficient implementations of many advanced numeric operations
- (See: [scipy.org](http://scipy.org) and pandas – great for analytics!)

# The None value

- None is used to represent an empty value.

```
In [ ]: import ipywidgets as widgets
        #BUG: dropdown does not start with empty value
        my_dropdown = widgets.Dropdown(options=['option 1', 'option 2'], value=None)
        my_dropdown
```

```
In [ ]: #you can set it afterwards, but not beforehand
        my_dropdown.value = None
```

```
In [ ]:
```

# Getting input from the user

- You can also use the `input()` function to get input from the user.
- Use the prompt string you want to display to the user as `input`'s parameter:

```
>>> name = input("Name? ")
Name? Jane
>>> print(name)
Jane
>>> age = int(input("Age? "))
Age? 28

>>> print(age)
28
>>>
```

← Converts input  
from string to int

# Built-in operators

- Python provides various built-in operators, from the standard (+, \*, and so on) to the more esoteric, such as operators for performing bit shifting, bitwise logical functions, and so forth.
- Most of these operators are no more unique to Python than to any other language; hence, I won't explain them in the main text.

# Basic Python style

Situation	Suggestion	Example
Module/package names	Short, all lowercase, underscores only if needed	<code>imp, sys</code>
Function names	All lowercase, underscores_for_readability	<code>foo(), my_func()</code>
Variable names	All lowercase, underscores_for_readability	<code>my_var</code>
Class names	CapitalizeEachWord	<code>MyClass</code>
Constant names	ALL_CAPS_WITH_UNDERSCORES	<code>PI, TAX_RATE</code>
Indentation	Four spaces per level, no tabs	
Comparisons	Don't compare explicitly to True or False	<code>if my_var: if not my_var:</code>



# Basic Python style

- **QUICK CHECK: PYTHONIC STYLE** Which of the following variable and function names do you think are not good Pythonic style? Why?

bar(),  
varName,  
VERYLONGVARNAME,  
foobar,  
longvarname,  
foo\_bar(),  
really\_very\_long\_var\_name



# Quick Review

- The basic syntax summarized above is enough to start writing Python code.
- Python syntax is predictable and consistent.
- Because the syntax offers few surprises, many programmers can get started writing code surprisingly quickly.





# Thanks for Your Time Today!

Recommended Reading Reminder for the Next Session:

**Python for Everybody** by Charles Severance

Please read Chapters 4 & 8