Introduction to Python

Python Crash Course

DATA 601

Before We Start

Please follow the directions in the following slides and make sure that you have a healthy python environment to run your own codes while learning!

Some of the early lectures will have some advanced subjects. If you don't understand, please take a note but do not panic. In the following lectures, we will cover those advanced subjects in detail.

Download & Install

Download Python 3.10 (or newer version) from

https://www.python.org/downloads/

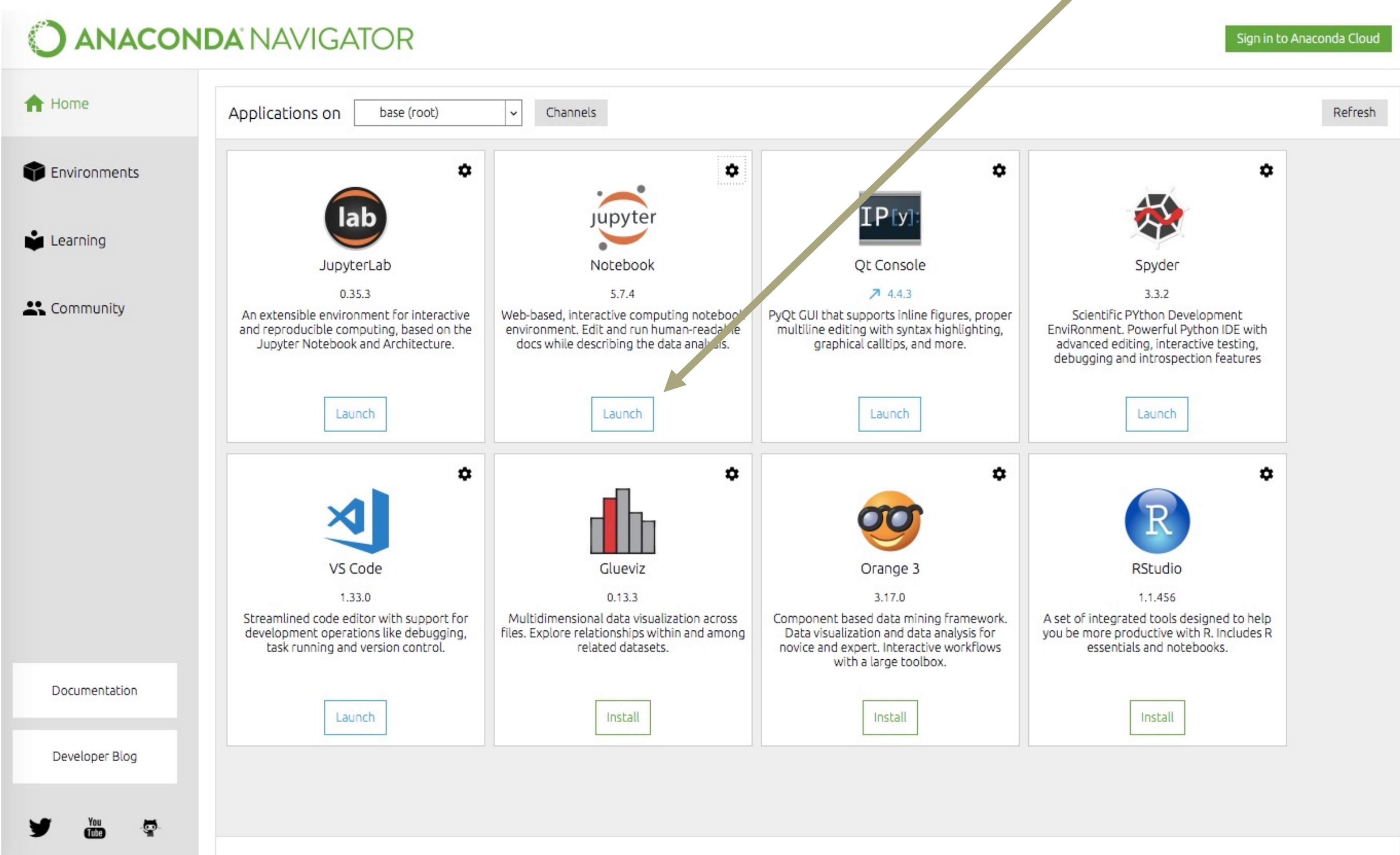
and install it.

Download latest version of Anaconda and install it

https://www.anaconda.com/distribution/

Once the installation complete, run Anaconda

If you see this, lunch "Jupyter Notebook" simply by clicking here



If your internet browser opens a new tab like this, you are ready to go

If somehow it didn't work, backup plan is using Google COLAB

→ Next Slide

Google Colab

If you don't have an UMBC email or another gmail account, first you need to create a Google Account for yourself at

https://accounts.google.com

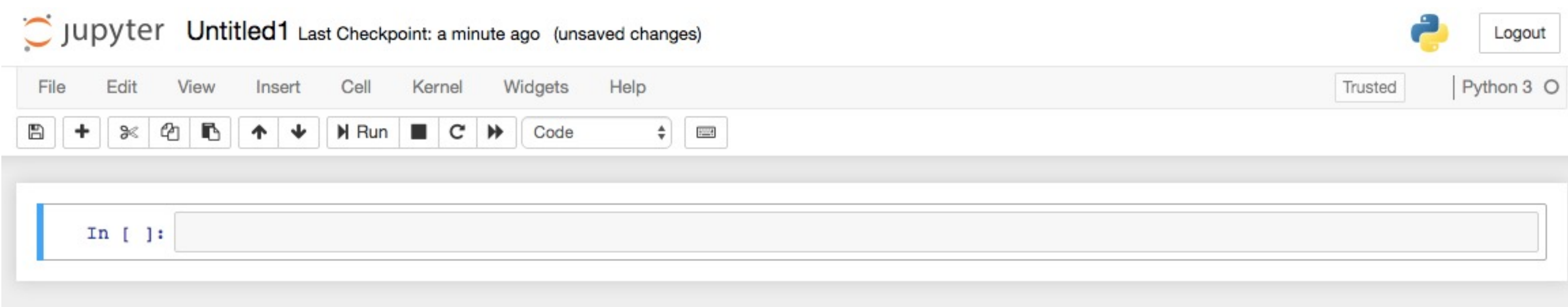
Sign into your google account. Then go to

https://colab.research.google.com/

What you see is free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

1. Click File 2. Choose "Python 3 Notebook"

If you All Have either of these, We are ready to start



Python is an object-oriented, high-level programming language with integrated dynamic semantics primarily for web and app development.

Python is relatively simple, so it's easy to learn since it requires a unique syntax that focuses on readability.

Python supports the use of modules and packages, so that programs can be designed in a modular style and code can be reused across a variety of projects.

What is Python?

Why Python?

Open Source/Free: No need to worry about licences

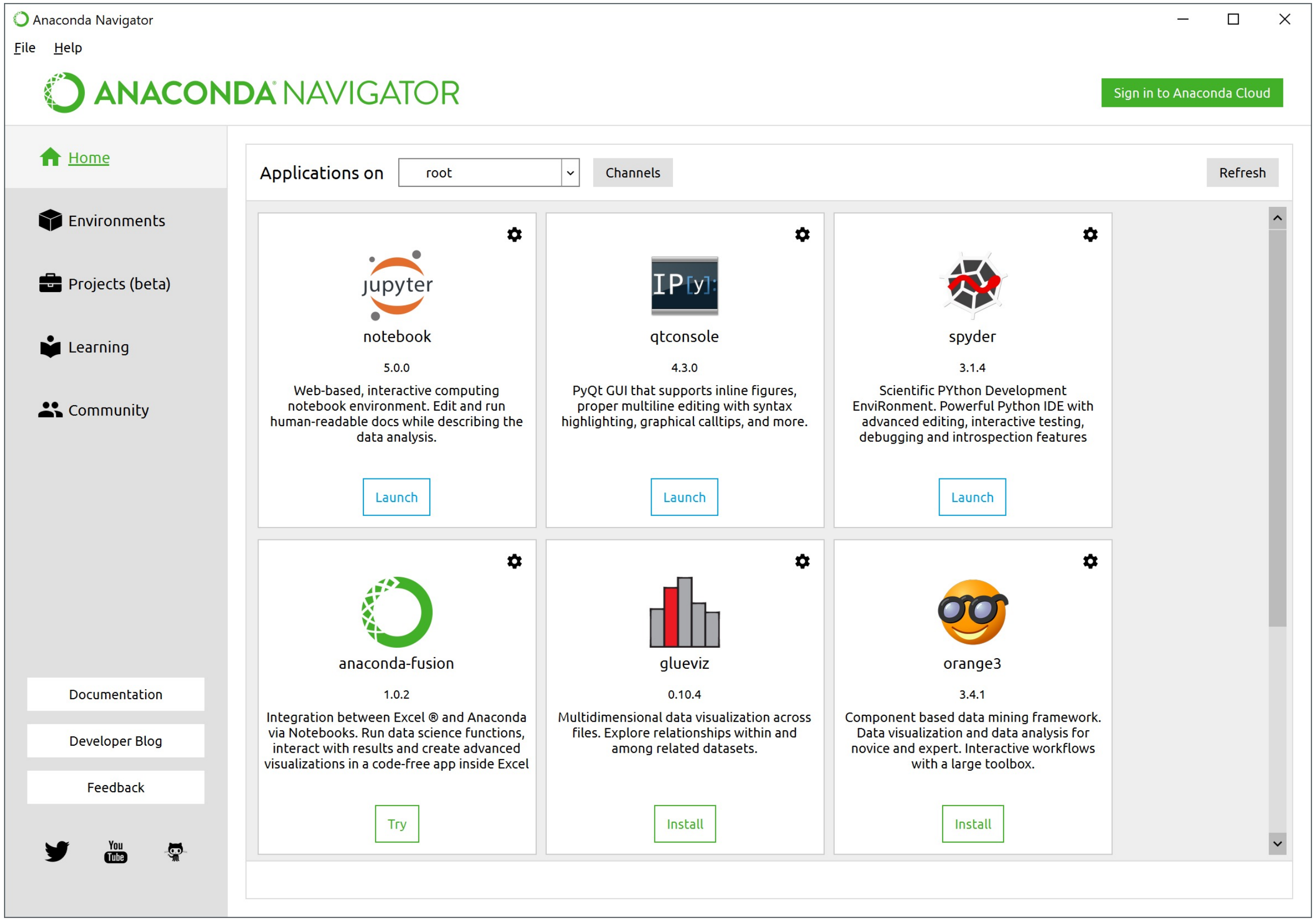
Cross-platform: Can be used with Windows/Macs OS/Linux – even Android and iOS!

Full-featured Packages: If there's something you want to do, there's probably a package out there to help you

Code Portability: With most code, it can run unaltered on a plethora of computers so long as all the required modules are supplied

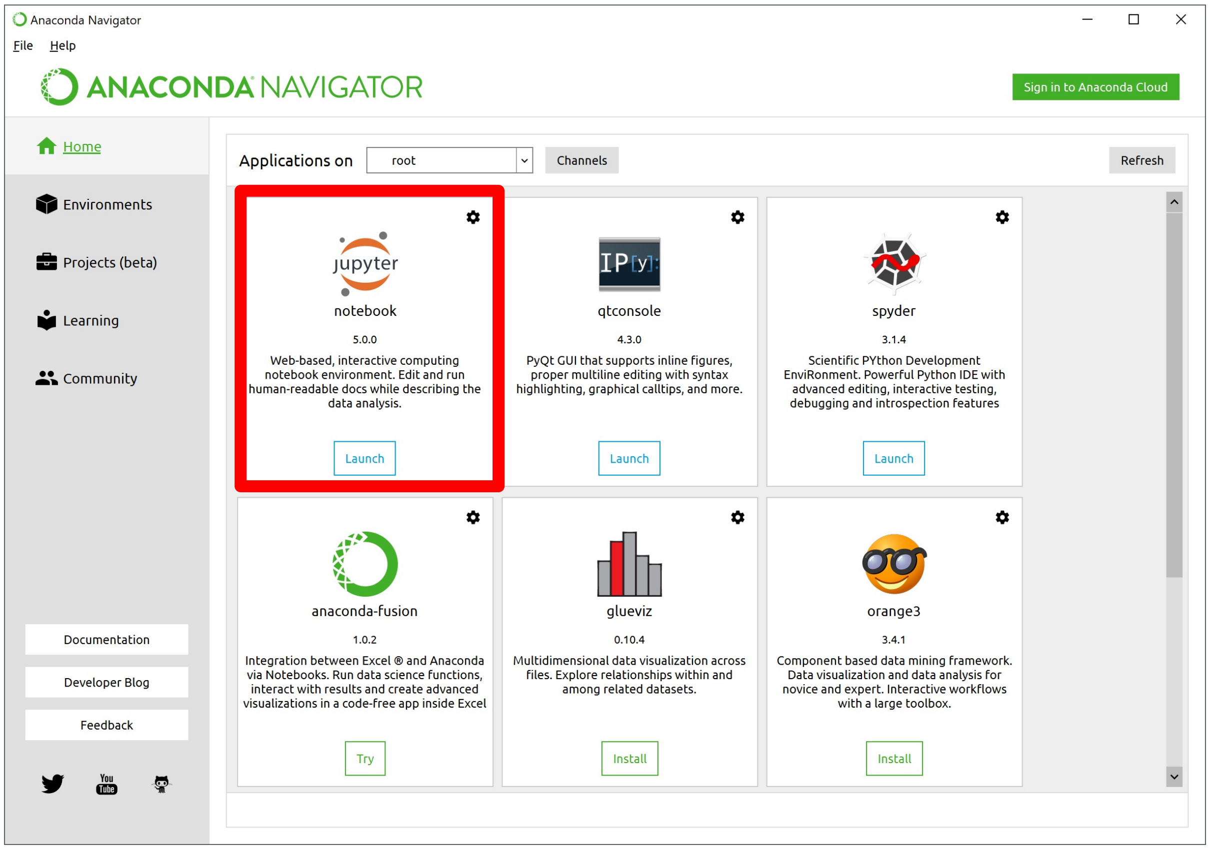
Large and Growing Community: People from all fields from Data Science to Physics are coding in Python, creating a diverse and rich community of experts all over.

Anaconda: What We'll be Using



Anaconda is a *distribution* that uses the *conda* tool to manage packages.

JUPYTER Notebook



PRO TIP:

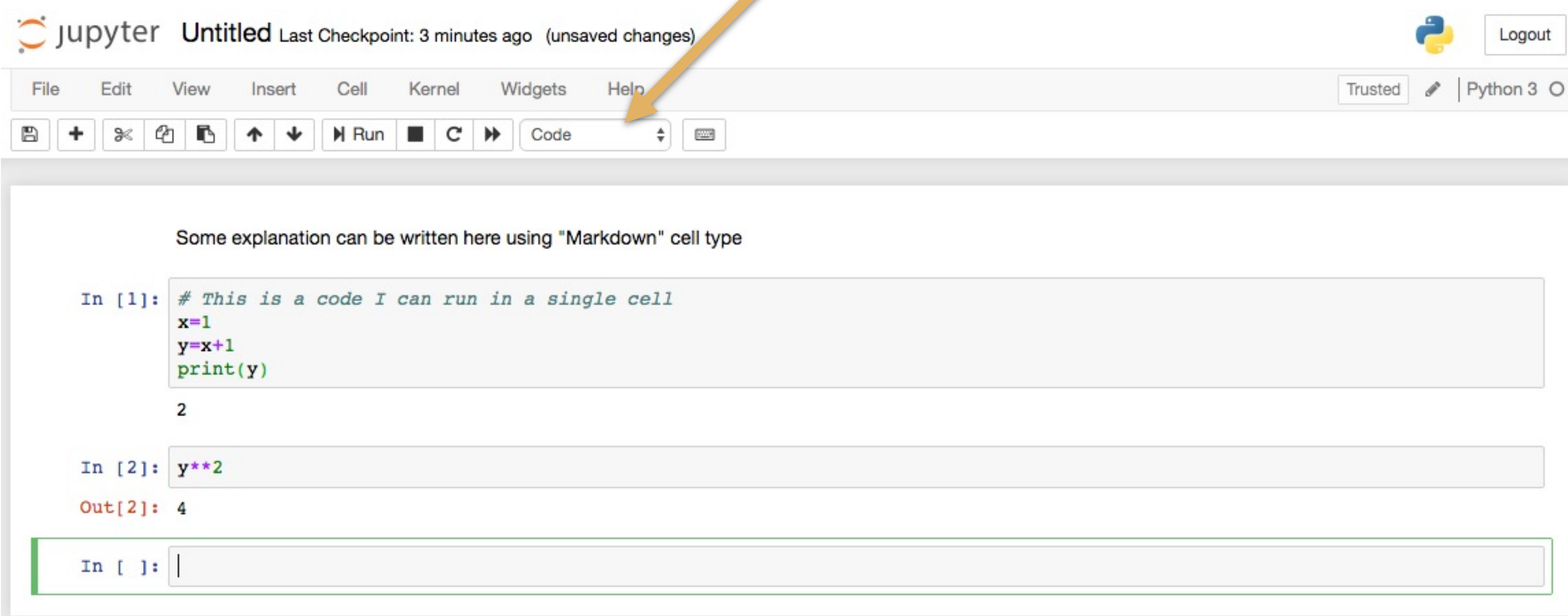
From command line:

```
$ jupyter notebook
```

Can launch from Anaconda: runs a python session in the background

JUPYTER Notebook

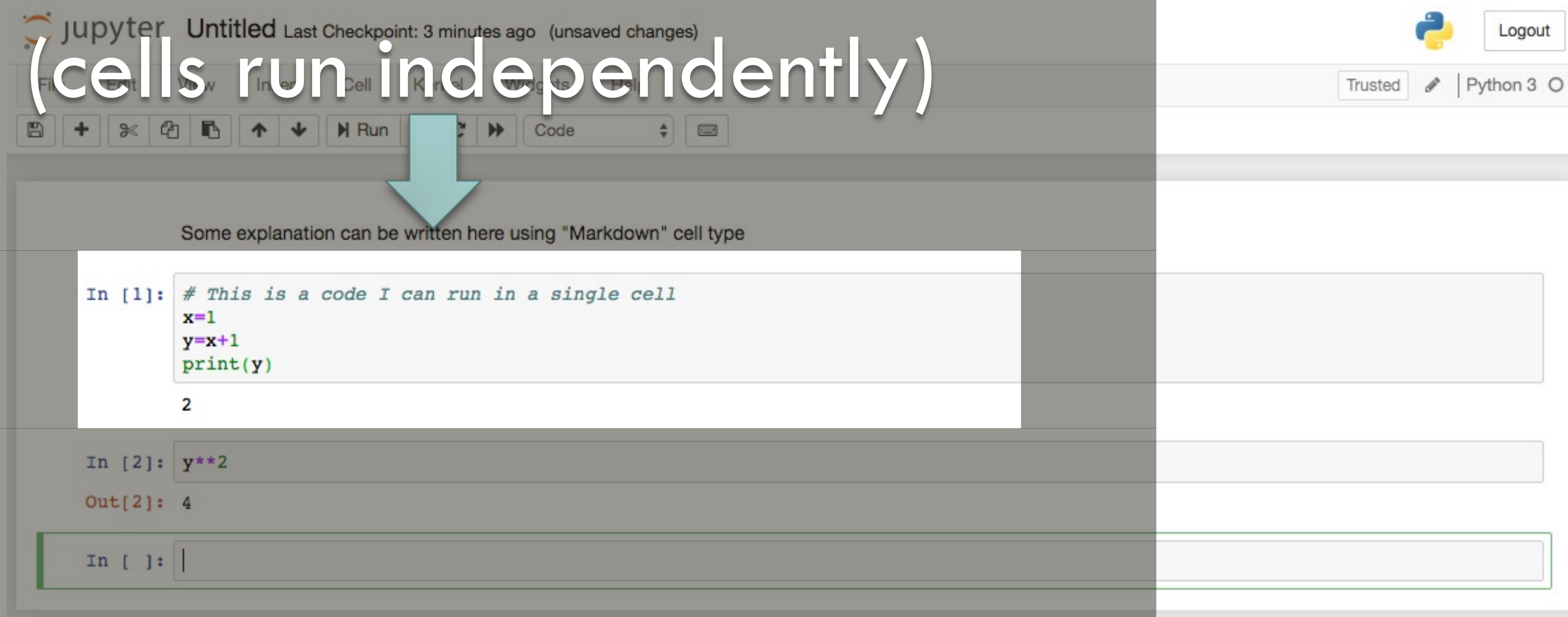
PRO TIP:
Change "Code" to "Markdown" for explanations

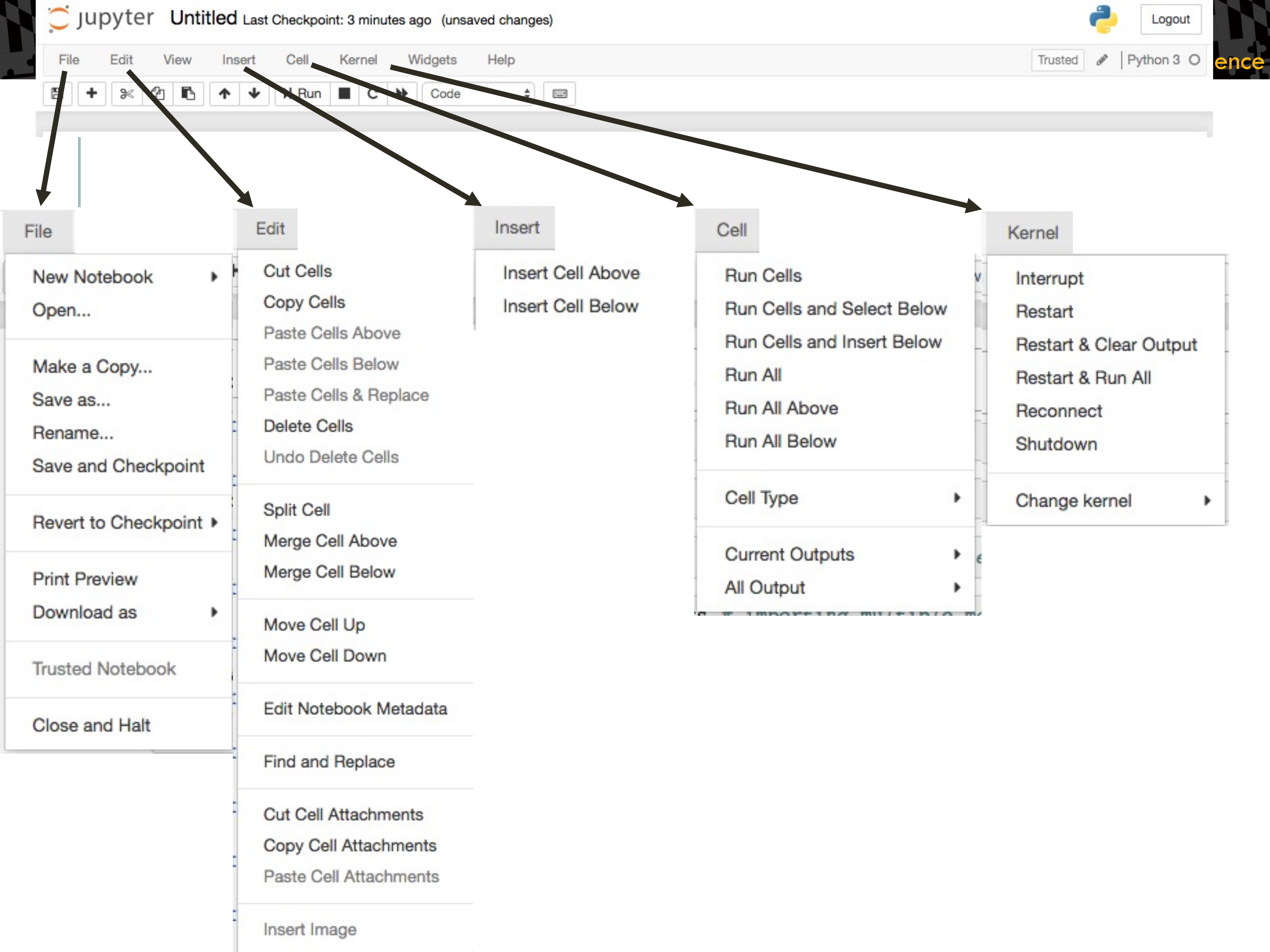


Runs a "notebook" in a web browser. Keeps code and notes together.

JUPYTER Notebook

Code (cells run independently)




jupyter Untitled Last Checkpoint: 3 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
ence
File
New Notebook
Open...
Make a Copy...
Save as...
Rename...
Save and Checkpoint
Revert to Checkpoint
Print Preview
Download as
Trusted Notebook
Close and Halt
Edit
Cut Cells
Copy Cells
Paste Cells Above
Paste Cells Below
Paste Cells & Replace
Delete Cells
Undo Delete Cells
Split Cell
Merge Cell Above
Merge Cell Below
Move Cell Up
Move Cell Down
Edit Notebook Metadata
Find and Replace
Cut Cell Attachments
Copy Cell Attachments
Paste Cell Attachments
Insert Image
Insert
Insert Cell Above
Insert Cell Below
Cell
Run Cells
Run Cells and Select Below
Run Cells and Insert Below
Run All
Run All Above
Run All Below
Cell Type
Current Outputs
All Output
Kernel
Interrupt
Restart
Restart & Clear Output
Restart & Run All
Reconnect
Shutdown
Change kernel

Modules: The Power of Python

The base language of Python is actually quite limited.

Most of its power comes from **Modules** (or sometimes referred to as **Packages**)

Modules must be **imported** before they can be used:

```
In [1]: import math
In [2]: import math, statistics
```

Importing single or multiple modules on a single line

Once imported, you can access functions or variables:

```
In [3]: math.cos(math.pi)
```

Modules: The Power of Python

Sometimes typing the module name all the time can be annoying in which case:

Creating a shorter name or just getting the function you want

```
In [1]: import numpy as np
In [2]: from numpy import zeros
```

Once imported, you can access functions or variables:

```
In [3]: np.ones(5)
Out[3]: array([1., 1., 1., 1., 1.])
In [4]: zeros(5)
Out[4]: array([0., 0., 0., 0., 0.])
```

Modules: The Power of Python

Observe the difference

```
[1]: import math
[2]: math.cos(math.pi)
[2]: -1.0
```

vs.

```
[1]: from math import cos, pi
[2]: cos(pi)
[2]: -1.0
```

Modules: The Power of Python

PRO TIP:

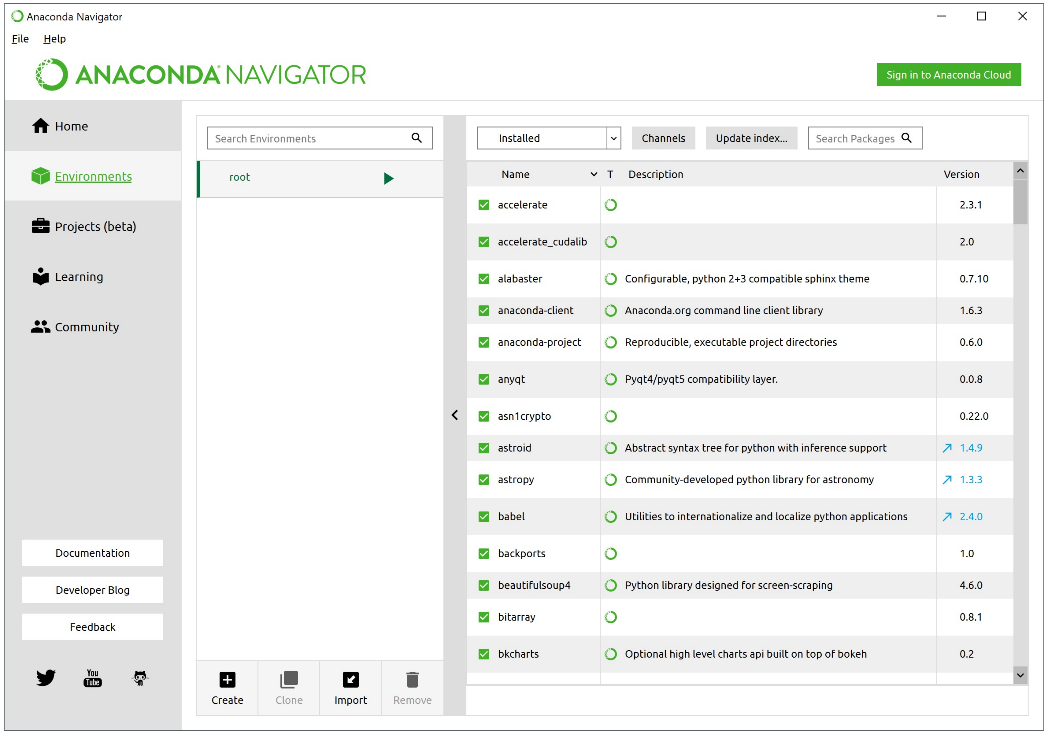
Some places will show examples that involve importing all functions in a module by:

```
from module1 import *
```

While this may seem handy, it is dangerous. **DON'T DO THIS!** It messes with your namespace!

Installing New Modules

Anaconda provides the majority of modules you'll want and/or need automatically. But there are modules that you'll likely want to get. Anaconda makes this easy using the **Environments Tab**



Installing New Modules

Python also makes installing packages easy in general using **conda** on the command line:

```
$ conda install numpy
```

This downloads and installs any package available on the (centralized) Anaconda repository.

To find a package:

```
$ conda search numpy
```

We will talk about modules in great detail later.
For the moment, just know that Python's power comes from modules and they are very easy to install

Basics of a Script: Comments

The most important part of any script

```
In [1]: # This is a comment
In [2]: # This is also a comment
```

For longer comments (in a script for instance):

```
'''
This text is in a comment
So is this text
'''
This text is outside a comment
```

Basics of a Script: Indentation

In the next slide, there is a “for loop” and an “if statement”

We will learn how those loops and statements work later.

The purpose of the next slide is showing you that indentation is important for Python

Basics of a Script: Indentation

Python uses indents to indicate blocks of code – no brackets!

```
# Indentation example
x = 1
y = 2
for i in (1, 2, 3):
    x = x + i
    y = y + i
    if x > y:
        print('Something is wrong!')
    print(x, y)
print('All done.')
```

Let your text editor deal with indenting for you. And when you need to do it yourself, *use spaces not tabs*. Python is picky about it!

Basics of a Script: Variables

What are variables?

Variables are the combination of an identifying label and a value, often a literal like 2 or "hello world". The label/identifier is attached to the value thus:

```
In [1]: a = 10
```

This is called variable assignment.

Once assigned, when you use the label, you get the value:

```
In [2]: print(a)
Out[2]: 10
```

Note that this is not the same as:

```
In [3]: print("a")
Out[2]: a
```

Basics of a Script: Variables (Cont…)

There is no need to declare any variable type before setting it

```
In [1]: x = 1        # No need to declare x is an integer
In [2]: y, z = 100, 200
```

Anything can be a variable in python: numbers, strings, functions, modules, *et cetera*. You can check out what type the variable is by:

```
In [3]: type(x)
Out[3]: int
```

They are called variables because you can change the value:

```
In [4]: x=1
In [5]: print(x)
Out[5]: 1
In [6]: x=2.1
In [7]: print(x)
Out[7]: 2.1
```

Basics of a Script: Variables (Cont…)

The label is just pointing to the value, which is stored somewhere in the computer memory.

```
In [1]: a=10
In [2]: b=a
In [3]: b
Out[3]: 10
In [4]: b=20
In [5]: a
Out[5]: 10
In [6]: b
Out[6]: 20
```

Variable identifiers

Names can be any continuous word/s, but must start with a letter or underscores.

There is no relationship between a variable's value or use and its name. In the last simple examples, a and b were fine, but generally you should name variables so you can tell what they are used for:

```
radius = 10
```

is more meaningful than

```
a = 10
```

or

```
bob = 10
```

In general, the more meaningful your names, the easier it will be to understand the code, when you come back to it, and the less likely you are to use the wrong variable.

Name Style

Style conventions aren't syntax, but allow all coders to recognise what an element is.

There's a styleguide for Python at:

https://www.python.org/dev/peps/pep-0008/

But it goes out of its way to avoid talking about variable names.

The community preference seems to be for lowercase words to be joined with underscores; what is called (coincidentally) snake_case:

```
perimeter_of_a_square
```

Though where Python is working with C or other code, the more conventional camelCase is sometimes used:

```
perimeterOfASquare
```

Either way, start with a lowercase letter, as other things start uppercase.

More on variables

You may see variables described as containers for values, but this isn't true and isn't helpful. Think of them as labels and values.

As we'll see later on, it is quite easy to get confused about what a variable is referring to, and thinking about them as a identifier/label and value helps.

Why we use variables?

Variables are generally used to hold the result of calculations and user inputs. These are things we can't predict before the code is run.

Some things we can predict the value of, for example, the 4 in:

```
perimeter = 4 * length_of_side # perimeter of a square
```

Such literals are hardwired in.

Even such literals are often better put in a variable at the top of the file, as changing the variable instantly allows us to change the value throughout:

```
number_of_sides = 4
perimeter = number_of_sides * length_of_side
```

This now works for any regular shape if we change `number_of_sides`.

Values

What kinds of things can we attached to variable labels?

Everything!

Literals like 1, 1.1, "a", "hello world".

But also whole chunks of code.

All the code and values in the computer is held in the form of binary data. We don't usually see this, but it is. It has to be: computers are just very complicated sets of on and off switches.

If values are just spaces in memory with something in them, and all code and values is binary, if we can attach a label to a value in memory, we can attach it to code in memory.

This is the basis of object oriented computing.

Objects

- Objects are chunks of code that are wrapped up in a particular way.
- One thing this format enables is the attaching of labels to them to create variables.
- Objects can have their own functions and variables, so you can have variables inside other variables.
- Objects generally do some particular job.
- We'll talk about them more later.

Values

In Python (but not all other languages), functions themselves are objects that can be given labels:

```
>>> a = print
>>> a("hello world")
hello world
```

This makes it incredibly powerful: for example, we can pass one function into another (the core of functional programming).

```
>>> dir(a)
```

Values

How does the computer know what a variable is?

For Python, it works it out. This takes it a little time, but means it is much more flexible.

This is why Python is more time and energy consuming compared to C, Fortran, etc.

Basics of a Script: Primitive Variables

There are only a few built-in variables in python, with standard operations:

Strings (str)	Integers (int)	Floats (float)

Any form of text. These can be enclosed in single (‘) or double (“) quotes.

```
In [1]: var1 = ‘This is a String’
In [2]: var2 = “This is also a String”
```

When added, they make a longer string:

```
In [3]: var1 + var2
Out[3]: 'This is a StringThis is also a String'
```

Basics of a Script: Primitive Variables

There are only a few built-in variables in python, with standard operations:

Strings (str) | Integers (int) | Floats (float)

They also accept **escape characters** (i.e., \n, \t, \a)

```
In [1]: var1 = 'This is a String \n'
In [2]: var2 = 'This is also a String'
```

When added, they make a longer string:

```
In [3]: print(var1+var2)
Out[3]: This is a String
        This is also a String
```

Basics of a Script: Primitive Variables

There are only a few built-in variables in python, with standard operations:

Strings (str)	Integers (int)	Floats (float)

Any integer (…, -1, 0, 1, …). Mathematical operations are as you expect

```
In [1]: var1, var2 = 1, 2
```

They are subject to floating point math:

```
In [2]: var1 / var2    # Regular division
Out[2]: 0.5
In [3]: var1 // var2   # Integer division - drop
fraction
Out[3]: 0
```

Basics of a Script: Primitive Variables

There are only a few built-in variables in python, with standard operations:

Strings (str)	Integers (int)	Floats (float)

Any integer (…, -1, 0, 1, …). Mathematical operations are as you expect

```
In [1]: var1, var2 = 5, 2
```

They are subject to floating point math:

```
In [2]: var1 / var2    # Regular division
Out[2]: 2.5
In [3]: var1 // var2   # Integer division – drop
fraction
Out[3]: 2
```

Basics of a Script: Primitive Variables

There are only a few built in variables in python:

Strings (str) | Integers (int) | Floats (float)

Any real number (1.0, 2.5, 1e25). Mathematical operations are as you expect

```
In [1]: var1, var2 = 1.0, 2e2
In [2]: var1, var2
Out[2]: (1.0, 200.0)
In [3]: 1/2.0 # int and float results in float
Out[3]: 0.5
```

Basics of a Script: Primitive Variables

There are only a few built in variables in python:

Strings (str) | Integers (int) | Floats (float)

Any real number (1.0, 2.5, 1e25). Mathematical operations are as you expect

```
In [1]: var1, var2 = 1.0, 2e2
In [2]: var1, var2
Out[2]: (1.0, 200.0)
In [3]: 1/2.0 # int and float results in float
Out[3]: 0.5
```

Basics of a Script: Primitive Variables

There are only a few built in variables in python:

Strings (str)	Integers (int)	Floats (float)

Any real number (1.0, 2.5, 1e25). Mathematical operations are as you expect

```
In [1]: var1, var2 = 1.0, 2e2
In [2]: var1, var2
Out[2]: (1.0, 200.0)
In [3]: 1/2.0 # int and float
Out[3]: 0.5
```

NOTE:

There is also a Boolean data type (bool). It can only have the values True or False.

Python and Complex Numbers

Python can handle complex numbers as well

```
In [1]: w = 3+4j
In [2]: print(type(w))
Out[2]: <class 'complex'>
```

Python and Scientific Numbers

Float can also be scientific numbers with an "e" to indicate the power of 10.

```
In [1]: x= 1.1e3
In [2]: print(x)
Out[2]: 1100.0
```

CASTING

There may be times when you want to specify a type on to a variable.

This can be done with casting

int() constructs an integer number from

- an integer literal,
- a float literal (by rounding down to the previous whole number),
- or a string literal (providing the string represents a whole number)

```
In [1]: x = int(1)
In [2]: y = int(2.8)
In [3]: z = int("3")
In [4]: print(x,y,z)
Out[4]: 1 2 3
```

CASTING (Cont…)

float() constructs a float number from

- an integer literal,
- a float literal or
- a string literal (providing the string represents a float or an integer)

```
In [1]: x = float(1)
In [2]: y = float(2.8)
In [3]: z = float("3")
In [4]: w = float("4.2")
In [5]: print(x,y,z,w)
Out[5]: 1.0 2.8 3.0 4.2
```

CASTING (Cont...)

str() constructs a string from a wide variety of data types, including strings, integer literals and float literals

```
In [1]: x = str("s1")
In [2]: y = str(2)
In [3]: z = str(3.0)
In [4]: w = str('Python Rocks')
In [5]: print(x,y,z,w)
Out[5]: s1 2 3.0 Python Rocks
```

More on Strings

Note that Python stores strings as arrays of bytes representing unicode characters. For example

A_String = ‘AbC DeF’

len(‘A_String’) returns 7

A_String[0] returns A

A_String[1] returns b

A_String[-1] returns F

A_String[3] returns an empty space

Some Very Useful String Methods

strip(), removes any whitespace from the beginning or the end

```
In [1]: Name3 = "  Albert Einstein   "
In [2]: print(Name3.strip())
Out[2]: Albert Einstein
```

lower() method returns the string in lower case

```
In [3]: Name3 = "Albert Einstein"
In [4]: print(Name3.lower())
Out[4]: albert einstein
```

upper() method returns the string in upper case:

```
In [5]: print(Name3.upper())
Out[5]: ALBERT EINSTEIN
```

Some Very Useful String Methods (Cont...)

replace() method replaces a string with another string:

```
In [1]: prices = "$100, $200, $87, $500"
In [2]: print(prices.replace("$", "€"))
Out[2]: €100, €200, €87, €500
```

```
In [3]: prices = "$100, $200, $87, $500"
In [4]: print(prices.replace("$", ""))
Out[4]: 100, 200, 87, 500
```

Some Very Useful String Methods (Cont…)

split() method splits the string into substrings if it finds instances of the separator

```
In [1]: prices = "$100, $200, $87, $500"
In [2]: print(prices.split(","))
Out[2]: ['$100', ' $200', ' $87', ' $500']
```

The output is a list of strings. We'll talk about "lists" in the next session

```
In [3]: A_sentence = "I will go shopping today"
In [4]: print(A_sentence.split(" "))
Out[4]: ['I', 'will', 'go', 'shopping', 'today']
```

Combining Numbers and Strings

If you try to combine string and number without proper transformation, Python will give you an error.

Here we convert an integer to a string

```
In [1]: temperature = 74
In [2]: z = "Baltimore"
In [3]: print(z, 'is', str(temperature), "degrees Fahrenheit today.")
Out[3]: Baltimore is 74 degrees Fahrenheit today
```

Arithmetic Operators

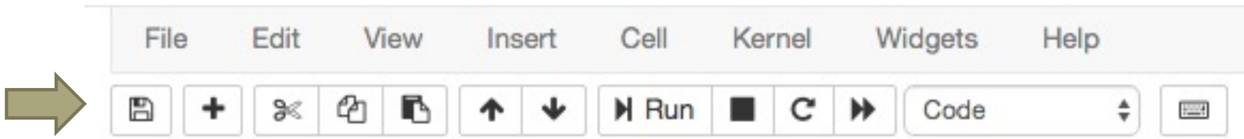
Input	Output
`x = 12.0`	
`y = 2.0`	
`addition = x+y`	14.0
`subtraction = x-y`	10.0
`multiplication = x*y`	24.0
`exponentitation = x**y`	144.0
`division = 12.4/y`	6.2
`floor_division = 12.4//y`	6.0
`modulus = 10%3`	1
`modulus = 10%3.0`	1.0

Augmented Assignment Operators

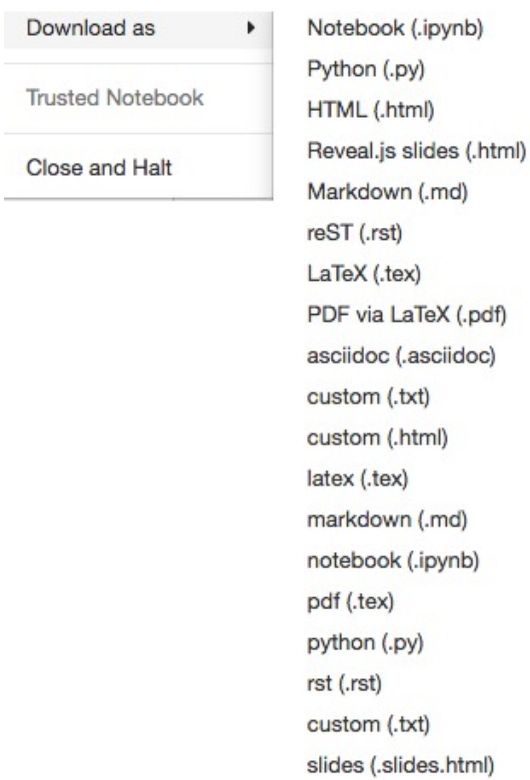
```
x += 1          # same as    x = x + 1
x -= 1          # same as    x = x - 1
x *= 3          # same as    x = x * 3
x /= 3          # same as    x = x / 3
x //= 3         # same as    x = x // 3
x %= 3          # same as    x = x % 3
```

Very useful operators for counting, indexing, etc.

Don’t Forget to Save Your Notebooks



Even though Jupyter saves it every notebook you create, it is always good to give a meaningful name to your notebook, put some explanations at the very top, and save your file before you quit.



Another nice property is that you can download your notebook in various different formats.

HTML is very useful if you want to publish your notebooks.

Merging Notebooks

Merging notebooks is very simple.

First install “nbmerge” with

```
pip install nbmerge
```

Then merge your files with

```
nbmerge file_1.ipynb file_2.ipynb file_3.ipynb > merged.ipynb
```

QUESTIONS?