Setting up your Python programming environment

Jupyter Notebook

Jupyter Notebook is an IDE for data science projects. This lab worksheet will walk you through how to set up Jupyter Notebook on your local machine and how to start using it. A notebook integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media.

Installation

To get started, install Anaconda. Libraries wrapped up in Anaconda include NumPy, SciPy, pandas, and many others. [Download](https://www.anaconda.com/download/) the latest version of Anaconda for Python 3.7 (chose the appropriate version 64/32 bits for your own machine); https://www.anaconda.com/download/

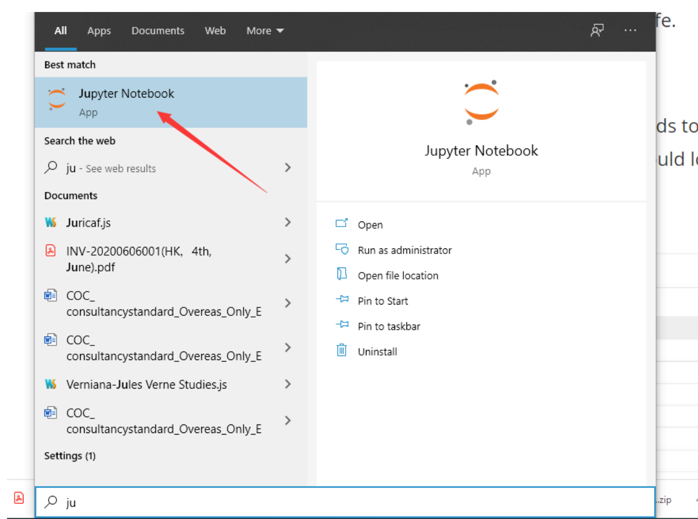
Install Anaconda by following the instructions on the download page and/or in the executable.

Under Linux, it may instead be best to proceed as follows:

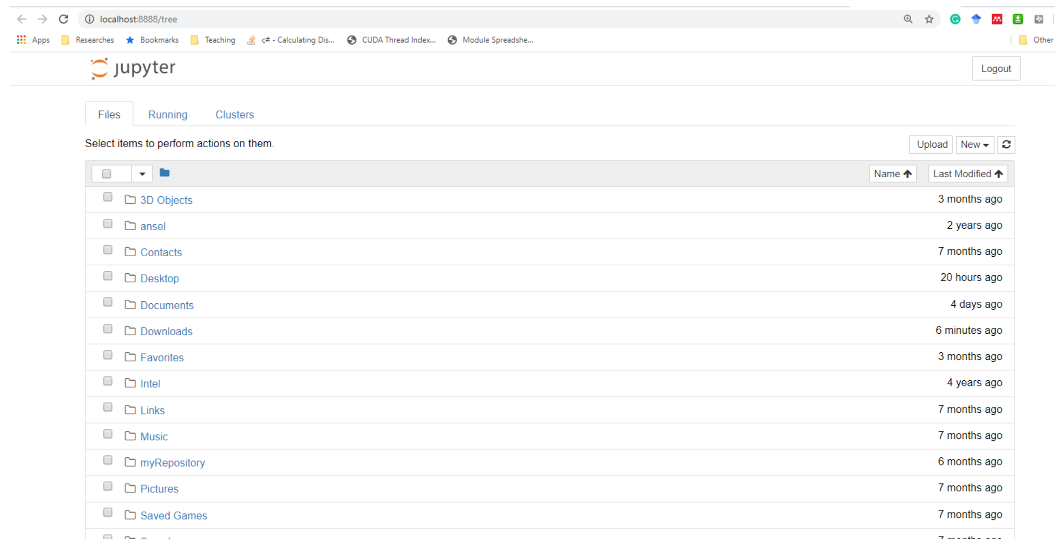
- Install conda with your package manager. For example, if dnf is your package manager, the command would be **sudo dnf install conda**. (You could then have to restart the terminal.)
- Create an environment, e.g., by the command **conda create -n local**, where “local” is the name of the environment, followed by an initialization command depending on the shell, such as **conda init bash** if it is bash. (Afterwards, it may be necessary to restart the terminal.)
- Activate the environment (command, e.g., **conda activate local**) and then install jupyter using conda with the command **conda install -c conda-forge jupyterlab**.
- If this was successful, the Jupyter Notebook dashboard can be launched with the command **jupyter-notebook**. By default, it is accessible through the URL http://localhost:8888/tree.

Running Jupyter

Under Linux, activate the environment (e.g., **conda activate local**; it may be convenient to add the activation command to your ~/.bashrc or equivalent file), and then start Jupyter with **jupyter-notebook**. On Windows, you can run Jupyter via a shortcut Anaconda adds to your start menu,



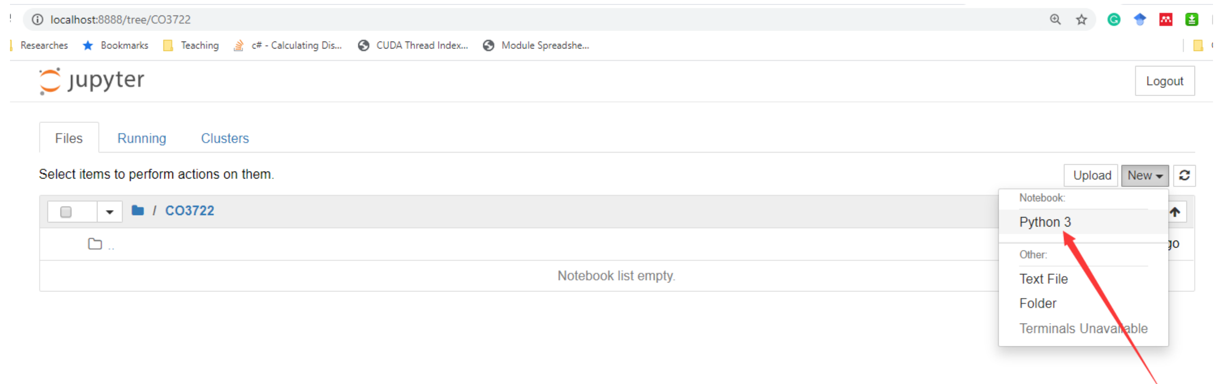
which will open a new tab in your default web browser that should look something like the following screenshot.



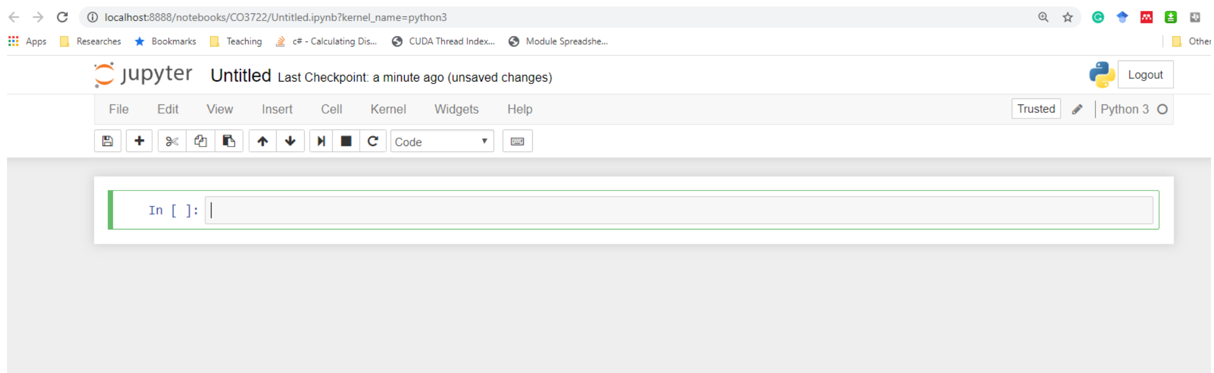
This is the Notebook Dashboard, designed for managing Jupyter Notebooks. Think of it as the launchpad for exploring, editing and creating your notebooks. The dashboard will give you access only to the files and sub-folders contained within Jupyter's start-up directory.

You may have noticed that the URL for the dashboard is something like http://localhost:8888/tree. Localhost indicates that the content is being served from your local machine. Jupyter starts up a local Python server, accessible through your web browser, making it platform independent and opening the door to easier sharing on the web. The dashboard's interface is mostly self-explanatory.

Now, browse to the folder in which you would like to create your first notebook, click the "New" drop-down button in the top-right and select "Python 3" (or the version of your choice).



Each notebook uses its own tab because you can open multiple notebooks simultaneously. If you switch back to the dashboard, you will see the new file Untitled.ipynb and you should see some green text that tells you your notebook is running. Our first Jupyter Notebook will open in a new tab:



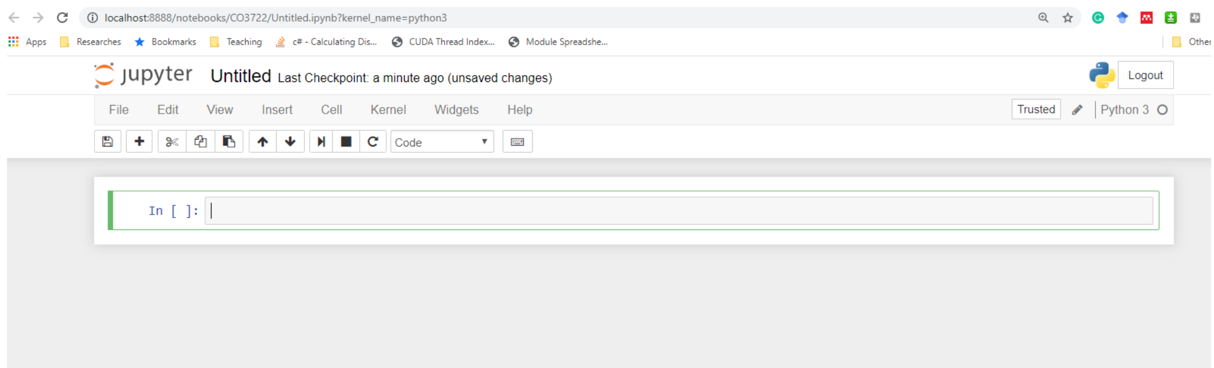
What is an ipynb File?

It will be useful to understand what this file really is. Each .ipynb file is a text file that describes the contents of your notebook in JSON format (i.e., JavaScript Object Notation). Each cell and its contents, including image attachments that have been converted into strings of text, is listed therein along with some metadata. You can edit this yourself — if you know what you are doing! — by selecting "Edit > Edit Notebook Metadata" from the menu bar in the notebook.

You can also view the contents of your notebook files by selecting "Edit" from the controls on the dashboard, but the keyword here is "can"; there's no reason other than curiosity to do so for this lab, unless you really know what you are doing.

The Notebook Interface

Now that you have an open notebook in front of you, its interface will hopefully not look entirely alien; it is essentially an advanced word processor. Check out the menus to get a feel for it, especially take a few moments to scroll down the list of commands in the command palette, which is the small button with the keyboard icon (or Ctrl + Shift + P).

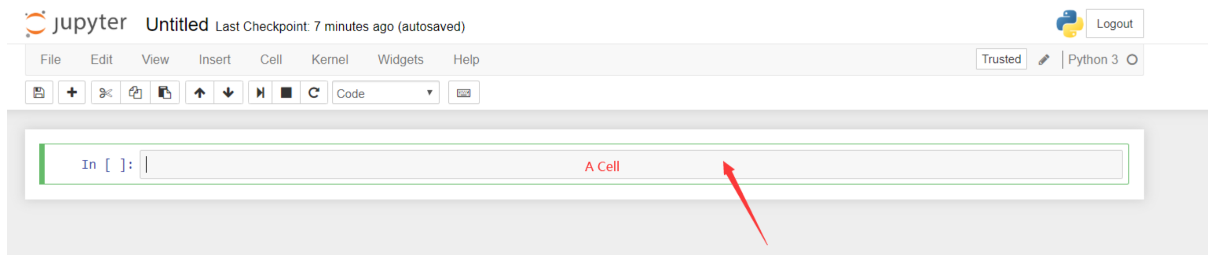


There are two fairly prominent terms that you should notice, which are probably new to you: *cells* and *kernels* are key to understanding Jupyter and to what makes it more than a word processor.

- A **kernel** is a "computational engine" that executes the code contained in a notebook document.
- A **cell** is a container for text to be displayed in the notebook or code to be executed by the notebook's kernel.

Cells

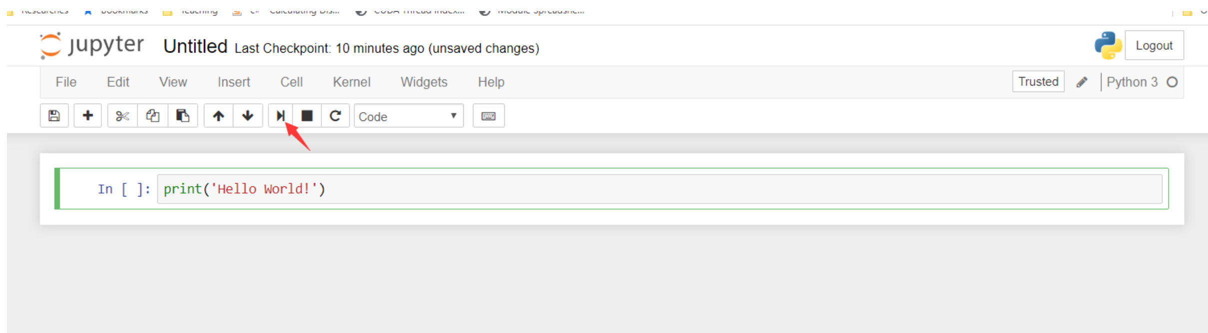
We'll return to kernels a little later, but first let's come to grips with cells. Cells form the body of a notebook. In the screenshot of a new notebook in the section above, that box with the green outline is an empty cell.



There are two main cell types that we will cover:

- A **code cell** contains code to be executed in the kernel and displays its output below.
- A **Markdown cell** contains text formatted using Markdown and displays its output in-place when it is run.

The first cell in a new notebook is always a **code cell**. Let's test it out with a classic hello world example. Type print('Hello World!') into the cell and click the run button Notebook Run Button in the toolbar above or press Ctrl + Enter.



The result should look like this:

When you ran the cell, its output will have been displayed below and the label to its left will have changed from In [ ] to In [1]. The output of a code cell also forms part of the document, which is why you can see it. You can always tell the difference between code and Markdown cells because code cells have that label on the left and Markdown cells do not.

The "In" part of the label is simply short for "Input," while the label number indicates when the cell was executed on the kernel — in this case the cell was executed first. Run the cell again and the label will change to In [2] because now the cell was the second to be run on the kernel. It will become clearer why this is so useful later on when we take a closer look at kernels.

From the menu bar, click Insert and select Insert Cell Below to create a new code cell underneath your first and try out the following code to see what happens. Do you notice anything different?

This cell doesn't produce any output, but it does take three seconds to execute. Notice how Jupyter signifies that the cell is currently running by changing its label to In [*].

In general, the output of a cell comes from any text data specifically printed during the cells execution, as well as the value of the last line in the cell, be it a lone variable, a function call, or something else. For example:

You'll find yourself using this almost constantly in your own projects, and we'll see more of it later on.

Keyboard Shortcuts

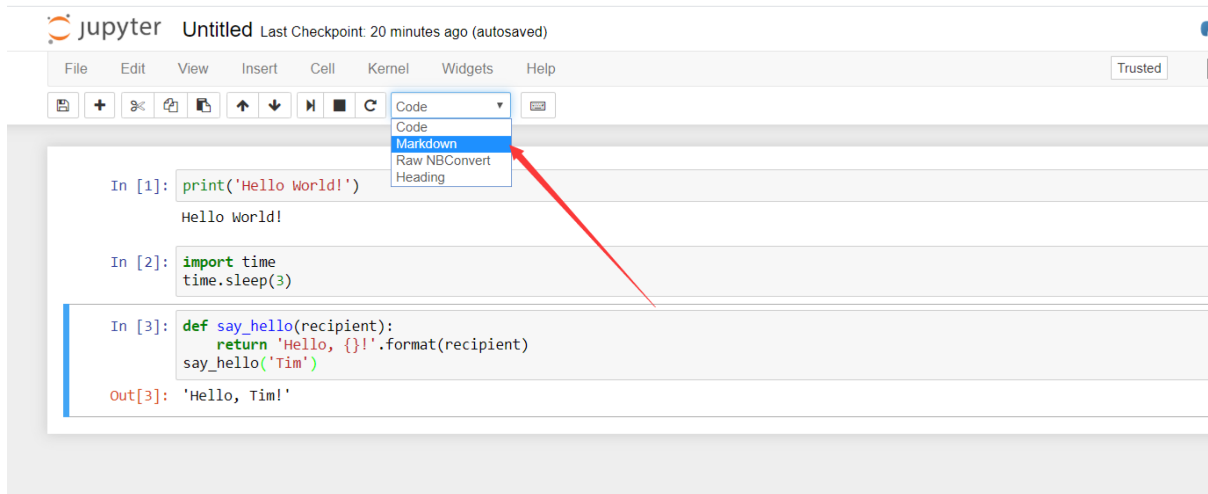
One final thing you may have observed when running your cells is that their border turned blue, whereas it was green while you were editing. There is always one "active" cell highlighted with a border whose color denotes its current mode, where green means "edit mode" and blue is "command mode."

So far we have seen how to run a cell with Ctrl + Enter, but there are plenty more. Keyboard shortcuts are a very popular aspect of the Jupyter environment because they facilitate a speedy cell-based workflow. Many of these are actions you can carry out on the active cell when it's in command mode.

Below, you'll find a list of some of Jupyter's keyboard shortcuts. You're not expected to pick them up immediately, but the list should give you a good idea of what's possible.

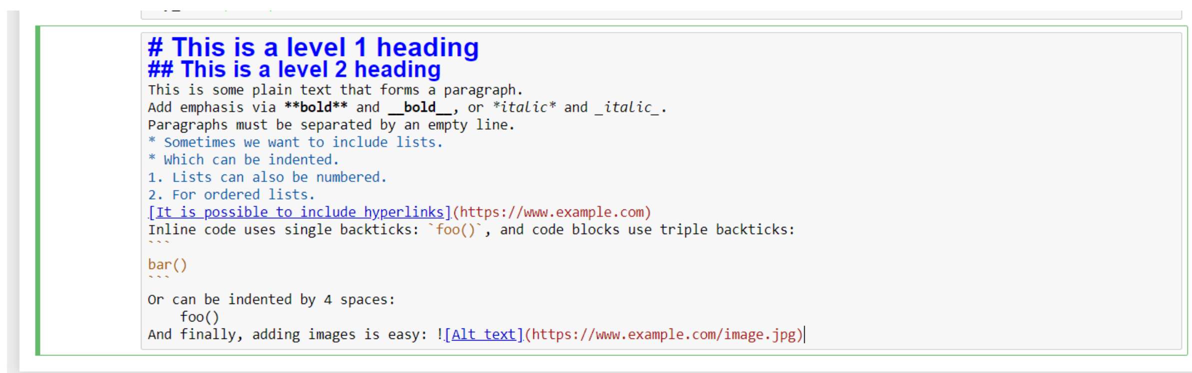
- Toggle between edit and command mode with Esc and Enter, respectively.
- Once in command mode:
    - Scroll up and down your cells with your Up and Down keys.
    - Press A or B to insert a new cell above or below the active cell.
    - M will transform the active cell to a Markdown cell.
    - Y will set the active cell to a code cell.
    - D + D (D twice) will delete the active cell.
    - Z will undo cell deletion.
    - Hold Shift and press Up or Down to select multiple cells at once.
        - With multiple cells selected, Shift + M will merge your selection.
- Ctrl + Shift + -, in edit mode, will split the active cell at the cursor.
- You can also click and Shift + Click in the margin to the left of your cells to select them.

Go ahead and try these out in your own notebook. Once you've had a play, create a new Markdown cell and we'll learn how to format the text in our notebooks.

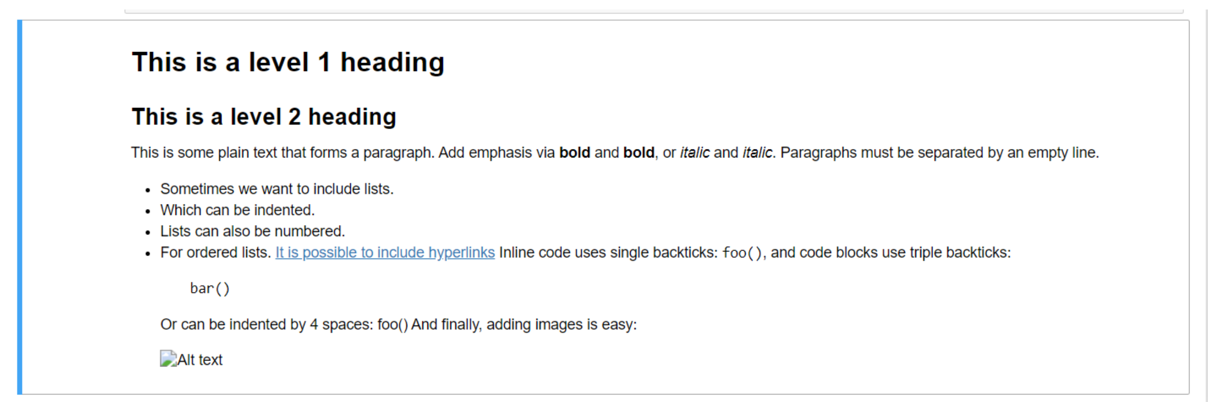


Markdown

Markdown is a lightweight, easy to learn markup language for formatting plain text. Its syntax has a one-to-one correspondance with HTML tags, so some prior knowledge here would be helpful but is definitely not a prerequisite. Remember that this article was written in a Jupyter notebook, so all of the narrative text and images you have seen so far was achieved in Markdown. Let's cover the basics with a quick example.



Run the above cell and you will see the following:



When attaching images, you have three options:

- Use a URL to an image on the web.
- Use a local URL to an image that you will be keeping alongside your notebook, such as in the same git repo.

- Add an attachment via “Edit > Insert Image”; this will convert the image into a string and store it inside your notebook .ipynb file.

Note that this will make your .ipynb file much larger!

There is plenty more detail to Markdown, especially around hyperlinking, and it’s also possible to simply include plain HTML. Once you find yourself pushing the limits of the basics above, you can refer to the official guide.

Kernels

Behind every notebook runs a kernel. When you run a code cell, that code is executed within the kernel and any output is returned back to the cell to be displayed. The kernel’s state persists over time and between cells — it pertains to the document as a whole and not individual cells.

For example, if you import libraries or declare variables in one cell, they will be available in another. In this way, you can think of a notebook document as being somewhat comparable to a script file, except that it is multimedia. Let’s try this out to get a feel for it. First, we’ll import a Python package and define a function.

```
In [5]: import numpy as np        np is a global variable here
        def square(x):
            return x * x
```

Once we’ve executed the cell above, we can reference np and square in any other cell.

```
In [6]: x = np.random.randint(1, 10)
        y = square(x)
        print('%d squared is %d' % (x, y))

        6 squared is 36
```

This will work regardless of the order of the cells in your notebook. You can try it yourself, let’s print out our variables again.

```
In [6]: x = np.random.randint(1, 10)
        y = square(x)
        print('%d squared is %d' % (x, y))

        6 squared is 36

In [7]: print('Is %d squared is %d?' % (x, y))

        Is 6 squared is 36?
```

No surprises here! But now let’s change y.

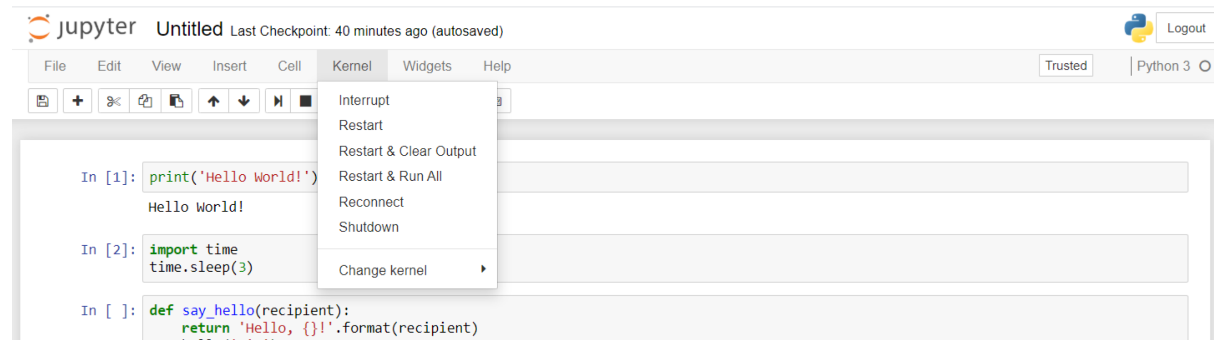
```
In [9]: x = np.random.randint(1, 10)
        y = 10
        print('%d squared is %d' % (x, y))

        1 squared is 10

In [10]: print('Is %d squared is %d?' % (x, y))

         Is 1 squared is 10?
```

What do you think will happen if we run the cell containing our print statement again? We will get the output Is 1 squared is 10?!

Most of the time, the flow in your notebook will be top-to-bottom, but it’s common to go back to make changes. And if you ever wish to reset things, there are several incredibly useful options from the Kernel menu:

- Restart: restarts the kernel, thus clearing all the variables etc that were defined.
- Restart & Clear Output: same as above but will also wipe the output displayed below your code cells.
- Restart & Run All: same as above but will also run all your cells in order from first to last.

If your kernel is ever stuck on a computation and you wish to stop it, you can choose the Interupt option.



Choosing a Kernel

You may have noticed that Jupyter gives you the option to change kernel, and in fact there are many different options to choose from. Back when you created a new notebook from the dashboard by selecting a Python version, you were actually choosing which kernel to use.

Not only are there kernels for different versions of Python, but also for over 100 languages including Java, C, and even Fortran. Data scientists may be particularly interested in the kernels for R and Julia, as well as both imatlab and the Calysto MATLAB Kernel for Matlab. The SoS kernel provides multi-language support within a single notebook. Each kernel has its own installation instructions, but will likely require you to run some commands on your computer.

Testing

Do test a few examples such as those given above, in particular in order to test whether libraries are properly installed and can be included; e.g., by the example

```
import numpy as np

def randomlist(n, sig):

  data = []

  for i in range(n):

    data.append(sig * np.random.randn())

  return data

print(randomlist(8, 3))
```

It may be necessary to specifically install libraries, e.g., by **conda install -c conda-forge numpy**.

Remark

This document is based on the “Setting up Jupyter” file included in the Jupyter Notebook distribution, with some minor modifications.