Python installation instructions

- Please have the most up to date version of Python (at the time of this course, it’s 3.9) installed.

- If you’ve never installed Python before, we will install Python through the installer Anaconda (which automatically installs several dependents)
  + Choose the the Anaconda installer that matches your computer build, which have been included on your flash drive in the Python Datamangement&Visualization>Installers folder (Windows, Mac, and here is a link for the Linux installation: https://www.anaconda.com/products/distribution

- We will be using Jupyter Lab and Jupyter Notebook to write and run our script. The following pages are instruction on how to open and run script in Jupyter lab and Notebook.

Running and Quitting

Overview

Teaching: 15 min
Exercises: 0 min

Questions

- How can I run Python programs?

Objectives

- Launch the JupyterLab server.
- Create a new Python script.
- Create a Jupyter notebook.
- Shutdown the JupyterLab server.
- Understand the difference between a Python script and a Jupyter notebook.
- Create Markdown cells in a notebook.
- Create and run Python cells in a notebook.

Many software developers will often use an integrated development environment (IDE) or a text editor to create and edit their Python programs which can be executed through the IDE or command line directly. While this is a common approach, we are going to use the Jupyter Notebook (https://jupyter.org/) via JupyterLab (https://jupyter.org/) for the remainder of this workshop.

This has several advantages:

- You can easily type, edit, and copy and paste blocks of code.
- Tab complete allows you to easily access the names of things you are using and learn more about them.
- It allows you to annotate your code with links, different sized text, bullets, etc. to make it more accessible to you and your collaborators.
- It allows you to display figures next to the code that produces them to tell a complete story of the analysis.

Each notebook contains one or more cells that contain code, text, or images.

Getting Started with JupyterLab

JupyterLab is an application with a web-based user interface from Project Jupyter (https://jupyter.org/) that enables one to work with documents and activities such as Jupyter notebooks, text editors, terminals, and even custom components in a flexible, integrated, and extensible manner. JupyterLab requires a reasonably up-to-date browser (ideally a current version of Chrome, Safari, or Firefox); Internet Explorer versions 9 and below are *not* supported.

JupyterLab is included as part of the Anaconda Python distribution. If you have not already installed the Anaconda Python distribution, see the setup instructions (../setup.html) for installation instructions.

Even though JupyterLab is a web-based application, JupyterLab runs locally on your machine and does not require an internet connection.

- The JupyterLab server sends messages to your web browser.
- The JupyterLab server does the work and the web browser renders the result.
- You will type code into the browser and see the result when the web page talks to the JupyterLab server.

JupyterLab? What about Jupyter notebooks?

JupyterLab is the next stage in the evolution of the Jupyter Notebook (https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html#overview). If you have prior experience working with Jupyter notebooks, then you will have a good idea of what to expect from JupyterLab.

Experienced users of Jupyter notebooks interested in a more detailed discussion of the similarities and differences between the JupyterLab and Jupyter notebook user interfaces can find more information in the JupyterLab user interface documentation (https://jupyterlab.readthedocs.io/en/stable/user/interface.html).

Starting JupyterLab

You can start the JupyterLab server through the command line or through an application called `Anaconda Navigator`. Anaconda Navigator is included as part of the Anaconda Python distribution.

macOS - Command Line

To start the JupyterLab server you will need to access the command line through the Terminal. There are two ways to open Terminal on Mac.

1. In your Applications folder, open Utilities and double-click on Terminal
2. Press Command + spacebar to launch Spotlight. Type `Terminal` and then double-click the search result or hit Enter

After you have launched Terminal, type the command to launch the JupyterLab server.

Code

```
$ jupyter lab
```

Windows Users - Command Line

To start the JupyterLab server you will need to access the Anaconda Prompt.

Press Windows Logo Key and search for `Anaconda Prompt`, click the result or press enter.

After you have launched the Anaconda Prompt, type the command:

Code

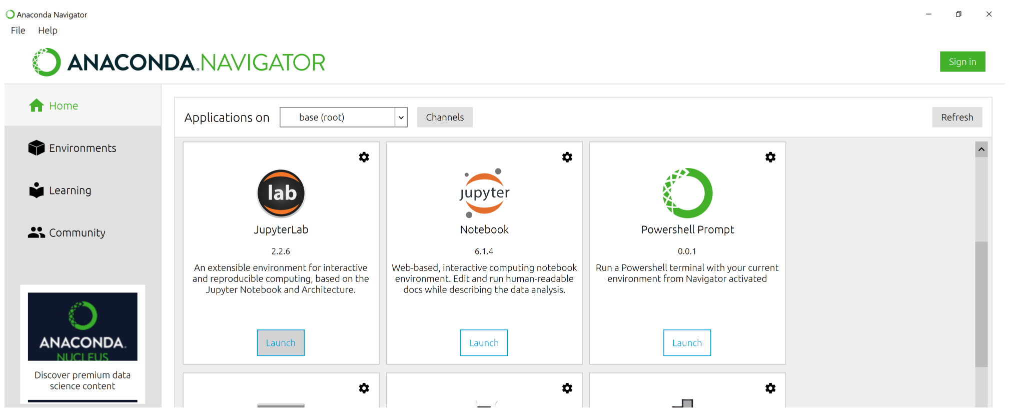
```
$ jupyter lab
```

Anaconda Navigator

To start a JupyterLab server from Anaconda Navigator you must first start Anaconda Navigator (click for detailed instructions on macOS, Windows, and Linux) (https://docs.anaconda.com/anaconda/navigator/getting-started/#starting-navigator). You can search for Anaconda Navigator via Spotlight on macOS (Command + spacebar), the Windows search function (Windows Logo Key) or opening a terminal shell and executing the `anaconda-navigator` executable from the command line.

After you have launched Anaconda Navigator, click the `Launch` button under JupyterLab. You may need to scroll down to find it.

Here is a screenshot of an Anaconda Navigator page similar to the one that should open on either macOS or Windows.

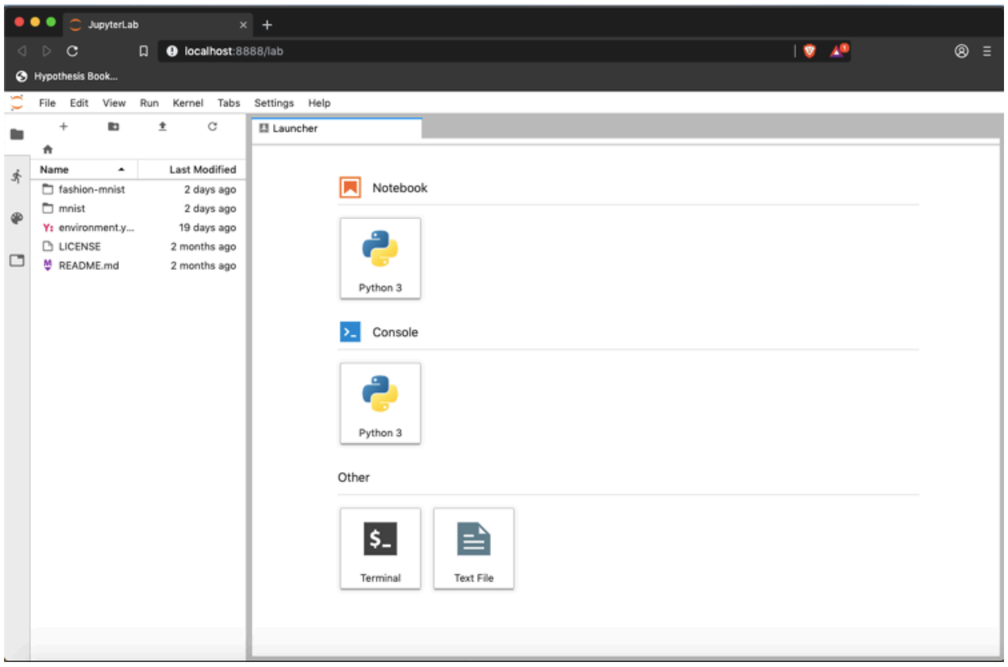


An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Run a Powershell terminal with your current environment from Navigator activated

And here is a screenshot of a JupyterLab landing page that should be similar to the one that opens in your default web browser after starting the JupyterLab server on either macOS or Windows.



The JupyterLab Interface

JupyterLab has many features found in traditional integrated development environments (IDEs) but is focused on providing flexible building blocks for interactive, exploratory computing.

The JupyterLab Interface (https://jupyterlab.readthedocs.io/en/stable/user/interface.html) consists of the Menu Bar, a collapsable Left Side Bar, and the Main Work Area which contains tabs of documents and activities.

Menu Bar

The Menu Bar at the top of JupyterLab has the top-level menus that expose various actions available in JupyterLab along with their keyboard shortcuts (where applicable). The following menus are included by default.

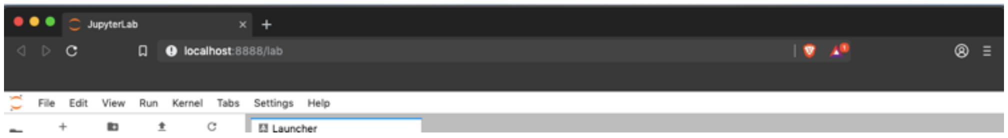
- **File:** Actions related to files and directories such as *New*, *Open*, *Close*, *Save*, etc. The *File* menu also includes the *Shut Down* action used to shutdown the JupyterLab server.
- **Edit:** Actions related to editing documents and other activities such as *Undo*, *Cut*, *Copy*, *Paste*, etc.
- **View:** Actions that alter the appearance of JupyterLab.
- **Run:** Actions for running code in different activities such as notebooks and code consoles (discussed below).
- **Kernel:** Actions for managing kernels. Kernels in Jupyter will be explained in more detail below.
- **Tabs:** A list of the open documents and activities in the main work area.
- **Settings:** Common JupyterLab settings can be configured using this menu. There is also an *Advanced Settings Editor* option in the dropdown menu that provides more fine-grained control of JupyterLab settings and configuration options.
- **Help:** A list of JupyterLab and kernel help links.

📌 Kernels

The JupyterLab docs (https://jupyterlab.readthedocs.io/en/stable/user/documents_kernels.html) define kernels as "separate processes started by the server that run your code in different programming languages and environments." When we open a Jupyter Notebook, that starts a kernel - a process - that is going to run the code. In this lesson, we'll be using the Jupyter ipython kernel which lets us run Python 3 code interactively.

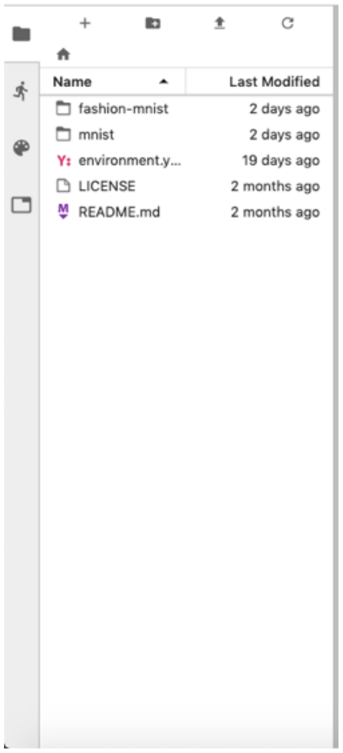
Using other Jupyter kernels for other programming languages (https://github.com/jupyter/jupyter/wiki/Jupyter-kernels) would let us write and execute code in other programming languages in the same JupyterLab interface, like R, Java, Julia, Ruby, JavaScript, Fortran, etc.

A screenshot of the default Menu Bar is provided below.



Left Sidebar

The left sidebar contains a number of commonly used tabs, such as a file browser (showing the contents of the directory where the JupyterLab server was launched), a list of running kernels and terminals, the command palette, and a list of open tabs in the main work area. A screenshot of the default Left Side Bar is provided below.

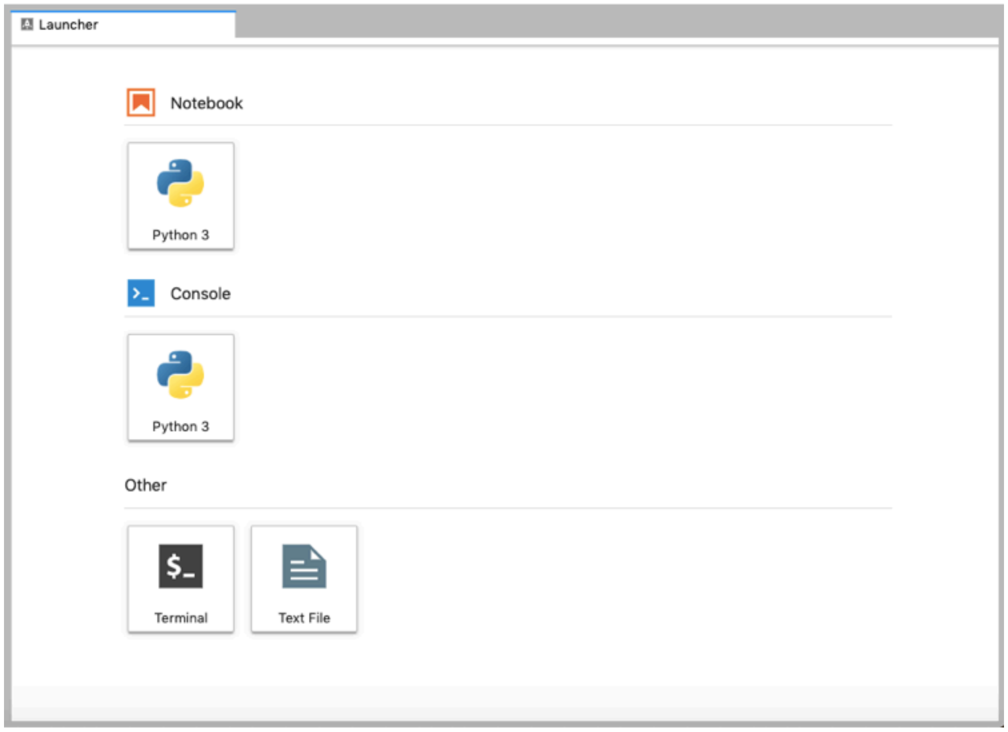


Name	Last Modified
fashion-mnist	2 days ago
mnist	2 days ago
environment.y...	19 days ago
LICENSE	2 months ago
README.md	2 months ago

The left sidebar can be collapsed or expanded by selecting “Show Left Sidebar” in the View menu or by clicking on the active sidebar tab.

Main Work Area

The main work area in JupyterLab enables you to arrange documents (notebooks, text files, etc.) and other activities (terminals, code consoles, etc.) into panels of tabs that can be resized or subdivided. A screenshot of the default Main Work Area is provided below.



Drag a tab to the center of a tab panel to move the tab to the panel. Subdivide a tab panel by dragging a tab to the left, right, top, or bottom of the panel. The work area has a single current activity. The tab for the current activity is marked with a colored top border (blue by default).

Creating a Python script

- To start writing a new Python program click the Text File icon under the *Other* header in the Launcher tab of the Main Work Area.
    - You can also create a new plain text file by selecting the *New* -> *Text File* from the *File* menu in the Menu Bar.
- To convert this plain text file to a Python program, select the *Save File As* action from the *File* menu in the Menu Bar and give your new text file a name that ends with the `.py` extension.
    - The `.py` extension lets everyone (including the operating system) know that this text file is a Python program.
    - This is convention, not a requirement.

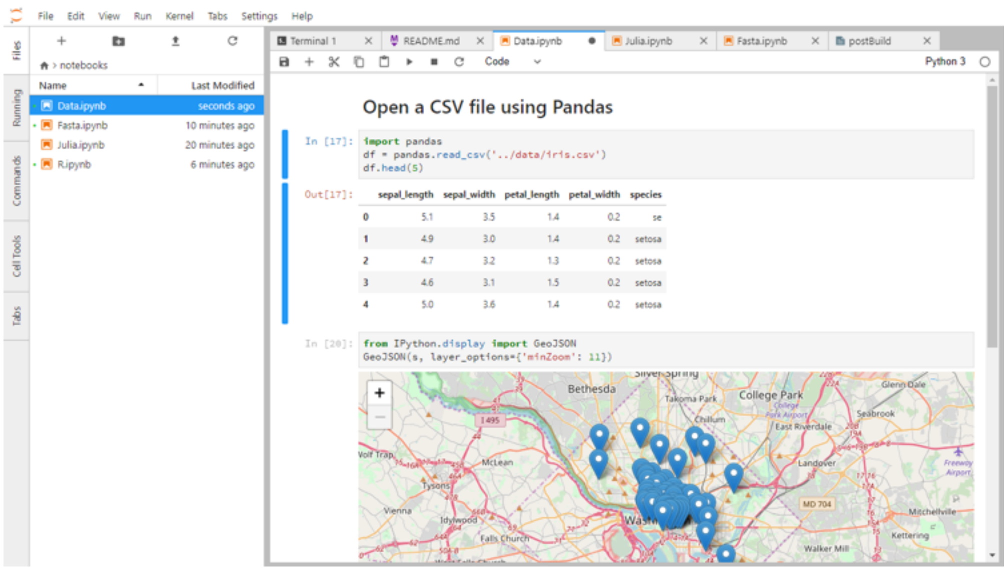
Creating a Jupyter Notebook

To open a new notebook click the Python 3 icon under the *Notebook* header in the Launcher tab in the main work area. You can also create a new notebook by selecting *New* -> *Notebook* from the *File* menu in the Menu Bar.

Additional notes on Jupyter notebooks.

- Notebook files have the extension `.ipynb` to distinguish them from plain-text Python programs.
- Notebooks can be exported as Python scripts that can be run from the command line.

Below is a screenshot of a Jupyter notebook running inside JupyterLab. If you are interested in more details, then see the official notebook documentation (https://jupyterlab.readthedocs.io/en/stable/user/notebook.html).



Name	Last Modified
Data.ipynb	seconds ago
Fasta.ipynb	10 minutes ago
Julia.ipynb	20 minutes ago
R.ipynb	6 minutes ago

Open a CSV file using Pandas

```
In [17]: import pandas
         df = pandas.read_csv('../data/iris.csv')
         df.head(5)
```

Out[17]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	se
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

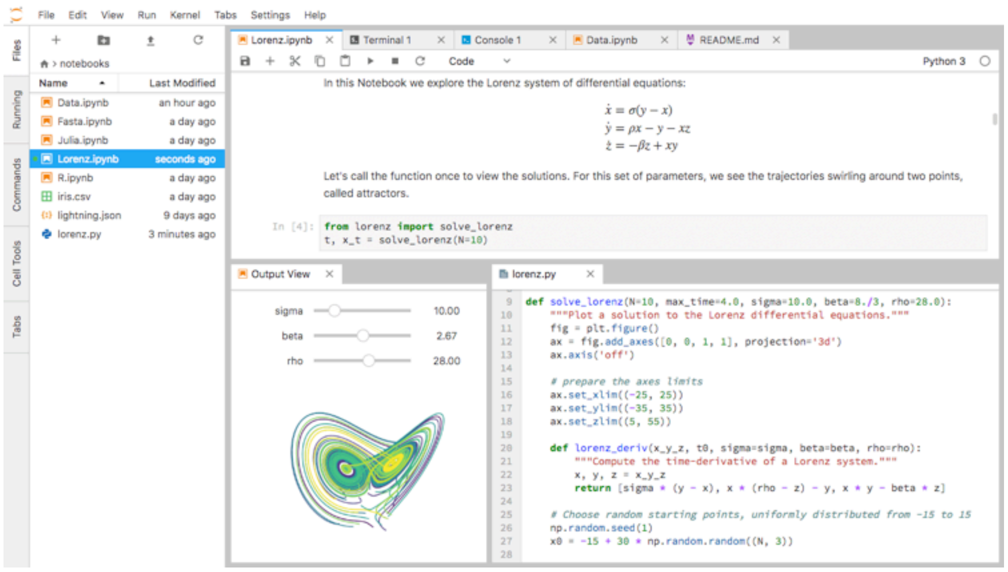
```
In [20]: from IPython.display import GeoJSON
         GeoJSON(s, layer_options={'minZoom': 11})
```

How It's Stored

- The notebook file is stored in a format called JSON.
- Just like a webpage, what's saved looks different from what you see in your browser.
- But this format allows Jupyter to mix source code, text, and images, all in one file.

Arranging Documents into Panels of Tabs

In the JupyterLab Main Work Area you can arrange documents into panels of tabs. Here is an example from the official documentation (https://jupyter.org/).



Name	Last Modified
Data.ipynb	an hour ago
Fasta.ipynb	a day ago
Julia.ipynb	a day ago
Lorenz.ipynb	seconds ago
R.ipynb	a day ago
iris.csv	a day ago
lightning.json	9 days ago
lorenz.py	3 minutes ago

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

Let's call the function once to view the solutions. For this set of parameters, we see the trajectories swirling around two points, called attractors.

```
In [4]: from lorenz import solve_lorenz
        t, x_t = solve_lorenz(N=10)
```

sigma		10.00
beta		2.67
rho		28.00

```
def solve_lorenz(N=10, max_time=4.0, sigma=10.0, beta=8./3, rho=28.0):
    """Plot a solution to the Lorenz differential equations."""
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.axis('off')

    # prepare the axes limits
    ax.set_xlim((-25, 25))
    ax.set_ylim((-35, 35))
    ax.set_zlim((5, 55))

    def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
        """Compute the time-derivative of a Lorenz system."""
        x, y, z = x_y_z
        return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]

    # Choose random starting points, uniformly distributed from -15 to 15
    np.random.seed(1)
    x0 = -15 + 30 * np.random.random((N, 3))

```

First, create a text file, Python console, and terminal window and arrange them into three panels in the main work area. Next, create a notebook, terminal window, and text file and arrange them into three panels in the main work area. Finally, create your own combination of panels and tabs. What combination of panels and tabs do you think will be most useful for your workflow?

Solution

Code vs. Text

Jupyter mixes code and text in different types of blocks, called cells. We often use the term “code” to mean “the source code of software written in a language such as Python”. A “code cell” in a Notebook is a cell that contains software; a “text cell” is one that contains ordinary prose written for human beings.

The Notebook has Command and Edit modes.

- If you press Esc and Return alternately, the outer border of your code cell will change from gray to blue.
- These are the **Command** (gray) and **Edit** (blue) modes of your notebook.
- Command mode allows you to edit notebook-level features, and Edit mode changes the content of cells.
- When in Command mode (esc/gray),
  - The b key will make a new cell below the currently selected cell.
  - The a key will make one above.
  - The x key will delete the current cell.
  - The z key will undo your last cell operation (which could be a deletion, creation, etc).
- All actions can be done using the menus, but there are lots of keyboard shortcuts to speed things up.

Command Vs. Edit

In the Jupyter notebook page are you currently in Command or Edit mode?
Switch between the modes. Use the shortcuts to generate a new cell. Use the shortcuts to delete a cell. Use the shortcuts to undo the last cell operation you performed.

Solution

Use the keyboard and mouse to select and edit cells.

- Pressing the Return key turns the border blue and engages Edit mode, which allows you to type within the cell.
- Because we want to be able to write many lines of code in a single cell, pressing the Return key when in Edit mode (blue) moves the cursor to the next line in the cell just like in a text editor.
- We need some other way to tell the Notebook we want to run what’s in the cell.
- Pressing Shift + Return together will execute the contents of the cell.
- Notice that the Return and Shift keys on the right of the keyboard are right next to each other.

The Notebook will turn Markdown into pretty-printed documentation.

- Notebooks can also render Markdown (https://en.wikipedia.org/wiki/Markdown).
  - A simple plain-text format for writing lists, links, and other things that might go into a web page.
  - Equivalently, a subset of HTML that looks like what you’d send in an old-fashioned email.
- Turn the current cell into a Markdown cell by entering the Command mode (Esc/gray) and press the M key.
- `In [ ]:` will disappear to show it is no longer a code cell and you will be able to write in Markdown.
- Turn the current cell into a Code cell by entering the Command mode (Esc/gray) and press the y key.

Markdown does most of what HTML does.

Code

```
*   Use asterisks
*   to create
*   bullet lists.
```

- Use asterisks
- to create
- bullet lists.

Code

```
1.  Use numbers
1.  to create
1.  numbered lists.
```

1. Use numbers
2. to create
3. numbered lists.

Code

```
*  You can use indents
	*  To create sublists
	*  of the same type
*  Or sublists
	1. Of different
	1. types
```

- You can use indents
  - To create sublists
  - of the same type
- Or sublists
  1. Of different
  2. types

Code

```
# A Level-1 Heading
```

A Level-1 Heading

Code

```
## A Level-2 Heading (etc.)
```

A Level-2 Heading (etc.)

Code

```
Line breaks
don't matter.

But blank lines
create new paragraphs.
```

Line breaks don't matter.

But blank lines create new paragraphs.

Code

```
[Create links](http://software-carpentry.org) with `[...](...)`.
Or use [named links][data_carpentry].

[data_carpentry]: http://datacarpentry.org
```

Create links (http://software-carpentry.org) with `[...](...)`. Or use named links (http://datacarpentry.org).

Creating Lists in Markdown

Create a nested list in a Markdown cell in a notebook that looks like this:

1. Get funding.
2. Do work.
   - Design experiment.
   - Collect data.
   - Analyze.
3. Write up.
4. Publish.

Solution

More Math

What is displayed when a Python cell in a notebook that contains several calculations is executed? For example, what happens when this cell is executed?

Python

```
7 * 3
2 + 1
```

Solution

Change an Existing Cell from Code to Markdown

What happens if you write some Python in a code cell and then you switch it to a Markdown cell? For example, put the following in a code cell:

Python

```python
x = 6 * 7 + 12
print(x)
```

And then run it with Shift + Return to be sure that it works as a code cell. Now go back to the cell and use Esc then m to switch the cell to Markdown and "run" it with Shift + Return. What happened and how might this be useful?

Solution

Equations

Standard Markdown (such as we're using for these notes) won't render equations, but the Notebook will. Create a new Markdown cell and enter the following:

Code

```
$\sum_{i=1}^{N} 2^{-i} \approx 1$
```

(It's probably easier to copy and paste.) What does it display? What do you think the underscore, `_`, circumflex, `^`, and dollar sign, `$`, do?

Solution

Closing JupyterLab

- From the Menu Bar select the "File" menu and then choose "Shut Down" at the bottom of the dropdown menu. You will be prompted to confirm that you wish to shutdown the JupyterLab server (don't forget to save your work!). Click "Shut Down" to shutdown the JupyterLab server.
- To restart the JupyterLab server you will need to re-run the following command from a shell.

Code

```
$ jupyter lab
```

Closing JupyterLab

Practice closing and restarting the JupyterLab server.

Key Points

- Python scripts are plain text files.
- Use the Jupyter Notebook for editing and running Python.
- The Notebook has Command and Edit modes.
- Use the keyboard and mouse to select and edit cells.
- The Notebook will turn Markdown into pretty-printed documentation.
- Markdown does most of what HTML does.

(/python-novice-gapminder/)

(/pyth
novice