

# CSC462 Artificial Intelligence

## LAB 2: Jupyter Notebooks

### Introduction

In this lab, you will be introduced to Jupyter notebooks. A Jupyter notebook is an application that can run Python code, display plots, show equations and contain formatted text. Jupyter notebooks are a great tool for problem solvers to write, run, document and share Python code with others.

By the end of this lab you will be able to:

- Explain what a Jupyter notebook is
- Open a Jupyter notebook
- Write Python code in a Jupyter notebook
- Run Python code in a Jupyter notebook
- Write and render markdown text in a Jupyter notebook
- Save and close a Jupyter notebook
- Download Jupyter notebooks in different file formats

### What is a Jupyter Notebook?

A *Jupyter notebook* is an electronic file that contains both programming code and text descriptions. Jupyter notebooks can also contain embedded charts, plots, images, videos, and links. Jupyter notebooks run in a web browser like Firefox or Google Chrome.

Although Jupyter notebooks can contain the code of many different programming languages, many Jupyter notebooks contain Python code. The Python code in a Jupyter notebook is the same type of Python code found in a *.py* file.

The text description sections of Jupyter notebooks contain explanations and clarifications of the programming code in the *markdown* format. *Markdown* files have the extension *.md*. Markdown sections of a Jupyter notebook can include formatting to make text bold, italic, form tables and lists, show code listings and render images.

One way to think of a Jupyter notebook is as a combination of the Python REPL and a Python module *.py* file with a markdown *.md* file thrown in between code sections.

In the Python REPL, only one command can be typed at a time, and only one line of output is shown at a time. In a *.py* file, the entire file is run at one time, line by line. The output of the entire file is produced all at once. Markdown *.md* files contain text in markdown format, but that text is not rendered. In a Jupyter notebook, chunks of code one line or many lines long can be run individually and in any order without running all of the code in the Jupyter notebook. Jupyter notebooks render the markdown sections and display rich text with headings, formatting, and images.

Jupyter notebooks contain three types of cells: *code cells*, *output cells*, and *markdown cells*.

- Code cells: Lines of Python code are run in code cells.

- Output cells: The output from running the code cells is also shown in output cells. Charts, plots, command line output, and images can all be shown in Jupyter notebooks as well.
- Markdown cells: Contain text-like descriptions of what will happen in subsequent code cells. Markdown cells can also contain images and links.

## Why Jupyter Notebooks?

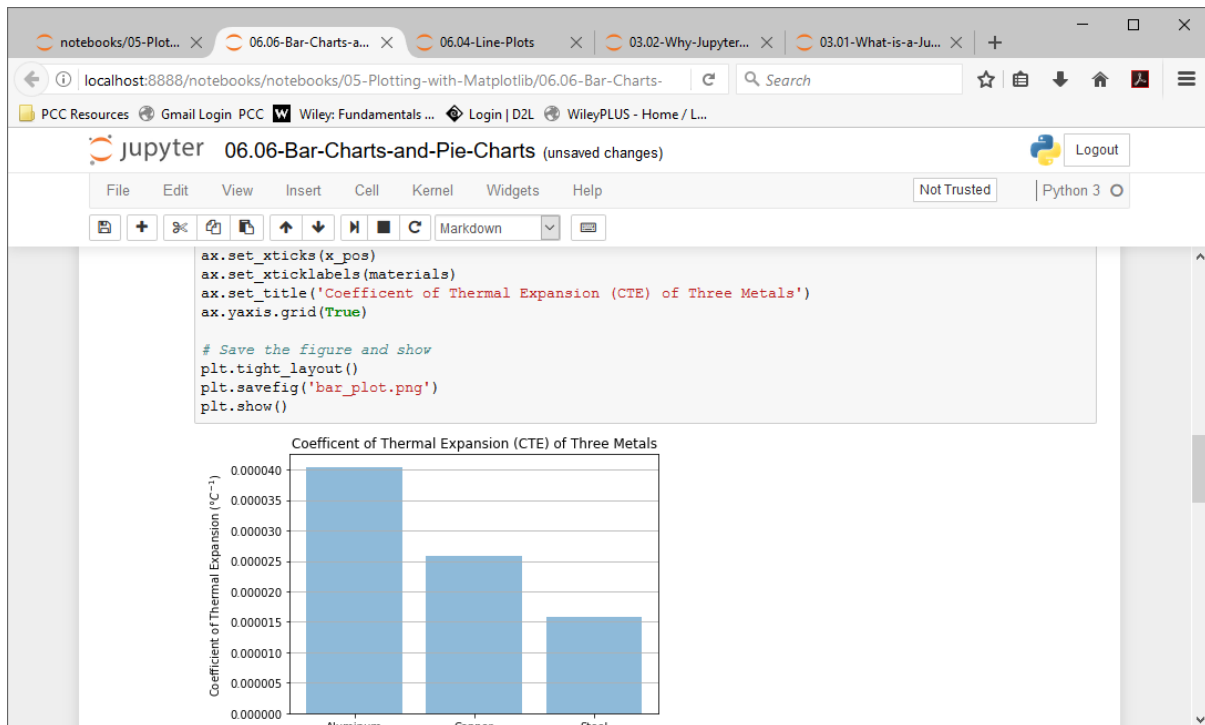
There is a vast array of editors and IDE's (Integrated Development Environments) which can be used to edit and run Python code. Why should problem solvers learn to use Jupyter notebooks?

Below is a table of simple text editors and IDE's which can edit and run Python code:

<b>Python Text Editor or IDE</b>	<b>Description</b>
Notepad	Simple text editor - included with Windows
Idle	Included with Python from Python.org
Sublime Text	Full-featured editor with long-time no-cost license
Spyder	IDE included with the Anaconda Distribution of Python
Visual Studio Code	An multi-language open source IDE
PyCharm	Professional Developer-friendly Python IDE

A Jupyter notebook is neither a simple text editor nor a full-featured IDE. Jupyter notebooks provide a quick and streamlined way for problem-solvers to prototype code and quickly share code.

Jupyter notebooks also provide a way for problem-solvers to share programming solutions with team members, supervisors, and customers.



In a way, Jupyter notebooks strike a balance between simple text editors, which are fast to start and simple and easy to manipulate, and IDE's which tend to start slower and be feature-rich and complex. Simple text editors typically can only edit code, and cannot run the code. A full IDE can edit code, run the code, debug code, provide syntax highlighting and context help.

In the context of problem-solving, Jupyter notebooks are quite handy. Jupyter notebooks open quickly and quickly produce output. Data exploration, data cleaning, and plot building are accomplished in Jupyter notebooks easier and quicker than in a text editor or an IDE.

In the context of sharing solutions to problems, Jupyter notebooks are also useful. Markdown cells render text in different sizes, bold and italic. Tables and images, plots and code can all be shown together in the same Jupyter notebook. Notebooks can be exported to a variety of formats including *.html* and *.pdf*.

## Installing Jupyter

The simplest way to install **Jupyter notebooks** is to download and install the Anaconda distribution of Python. The Anaconda distribution of Python comes with Jupyter notebook included and no further installation steps are necessary.

Below are additional methods to install Jupyter notebooks if you are not using the Anaconda distribution of Python.

### Installing Jupyter on Windows using the Anaconda Prompt

To install Jupyter on Windows, open the **Anaconda Prompt** and type:

```
> conda install jupyter
```

Type y for yes when prompted. Once Jupyter is installed, type the command below into the **Anaconda Prompt** to open the Jupyter notebook file browser and start using Jupyter notebooks.

```
> jupyter notebook
```

## Installing Jupyter on Linux

To install Jupyter on Linux, open a terminal and type:

```
$ conda install jupyter
```

Type y for yes when prompted.

Alternatively, if the Anaconda distribution of Python is not installed, one can use **pip**.

```
$ pip3 install jupyter
```

## Opening a Jupyter Notebook

In this section, you will learn how to open a Jupyter notebook on Windows and MacOS.

One way problem solvers can write and execute Python code is in Jupyter notebooks. Jupyter notebooks contain Python code, the output that code produces and markdown cells usually used to explain what the code means.

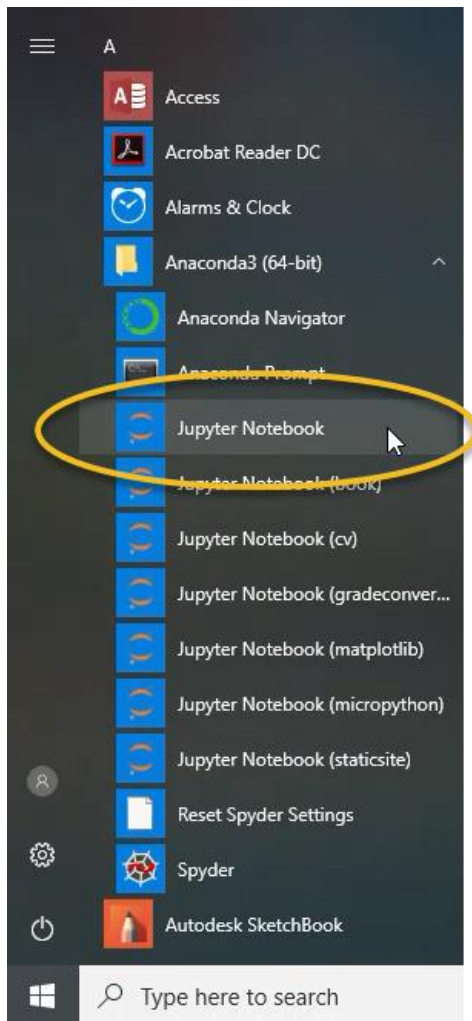
On Windows, a Jupyter notebook can be started from the **Anaconda Prompt**, the Windows start menu and **Anaconda Navigator**.

### 3 ways to open a Jupyter notebook:

- Windows Start Menu
- **Anaconda Prompt**
- Anaconda Navigator

### Open a Jupyter notebook with the Windows Start Menu

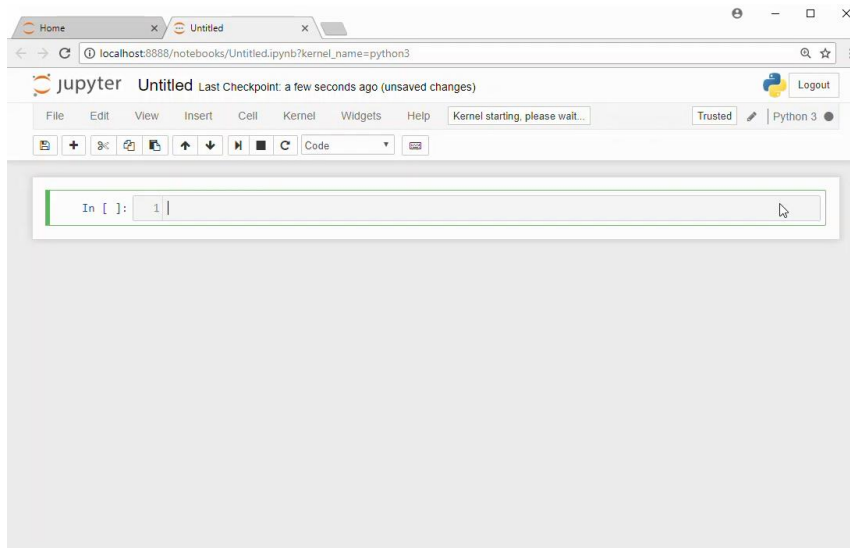
One way to open a Jupyter notebook is to use the Windows Start Menu. Note that the Anaconda distribution of Python must be installed to use the Windows Start Menu to open a Jupyter notebook. Download **Anaconda** at the following link: [Anaconda.com/distribution](https://anaconda.com/distribution)  
Open the Windows start menu and select [**Anaconda3(64 bit)**] --> [**Jupyter Notebook**]



This action opens the **Jupyter file browser** in a web browser tab.  
In the upper right select [New] --> [Python 3]

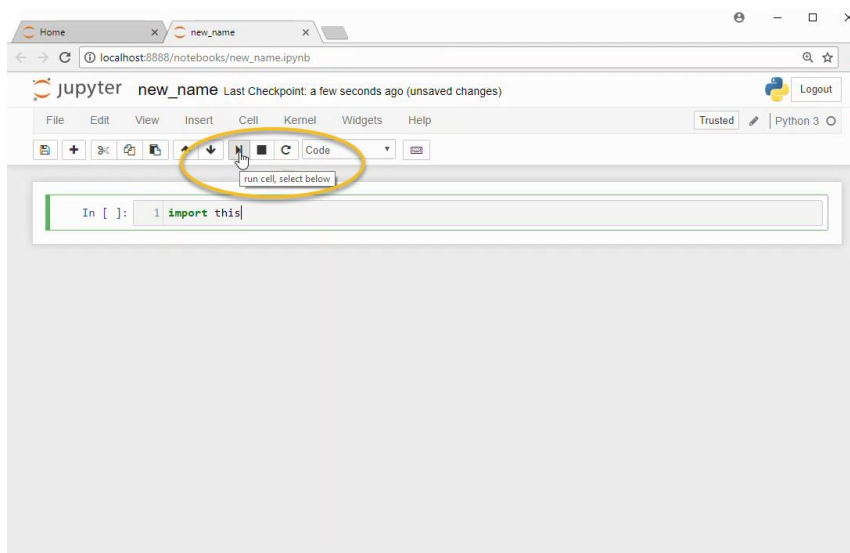


A new **notebook** will open as a new tab in your web browser.



Try typing the code below in the first cell in the notebook to the right of the In [ ]: prompt:  
**import this**

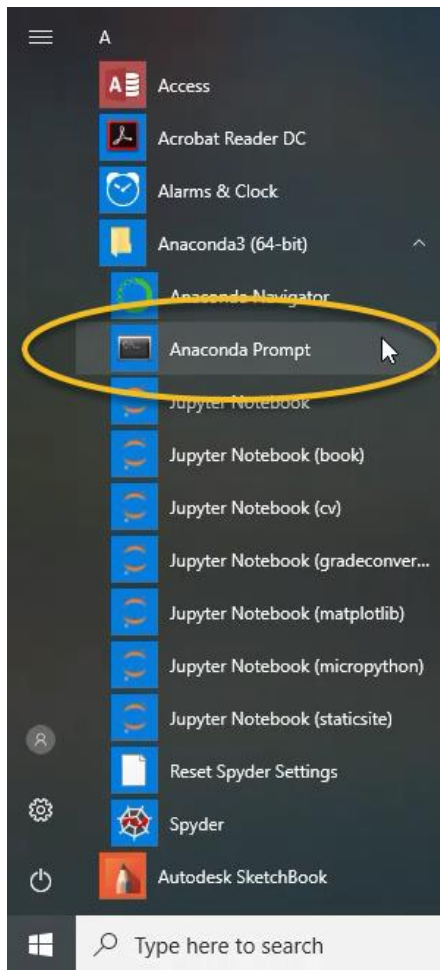
Then click the run button in the middle of the menu at the top of the notebook.



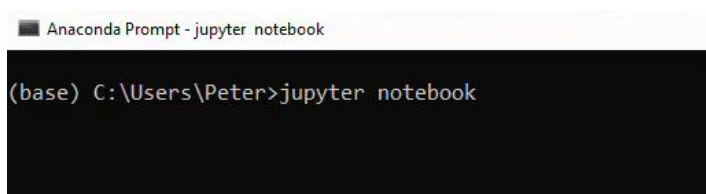
## Open a Jupyter Notebook with the Anaconda Prompt

Another method to open a Jupyter notebook is to use the **Anaconda Prompt**.

Go to the Windows start menu and select **[Anaconda Prompt]** under **[Anaconda3]**.



If you don't see the **Anaconda Prompt** in the Windows Start Menu, then you need to install the Anaconda distribution of Python. Download **Anaconda** at the following link: [Anaconda.com/distribution](https://anaconda.com/distribution)  
The **Anaconda Prompt** window should look something like the image below.



At the **Anaconda Prompt** type:

> jupyter notebook

This command starts the **Jupyter notebook** server. The output in the **Anaconda Prompt** will look something like the output shown below:

Copy/paste this URL into your browser when you connect ...

to login with a token:

`http://localhost:8888/?token=6bdef677d3503fbb2 ...`

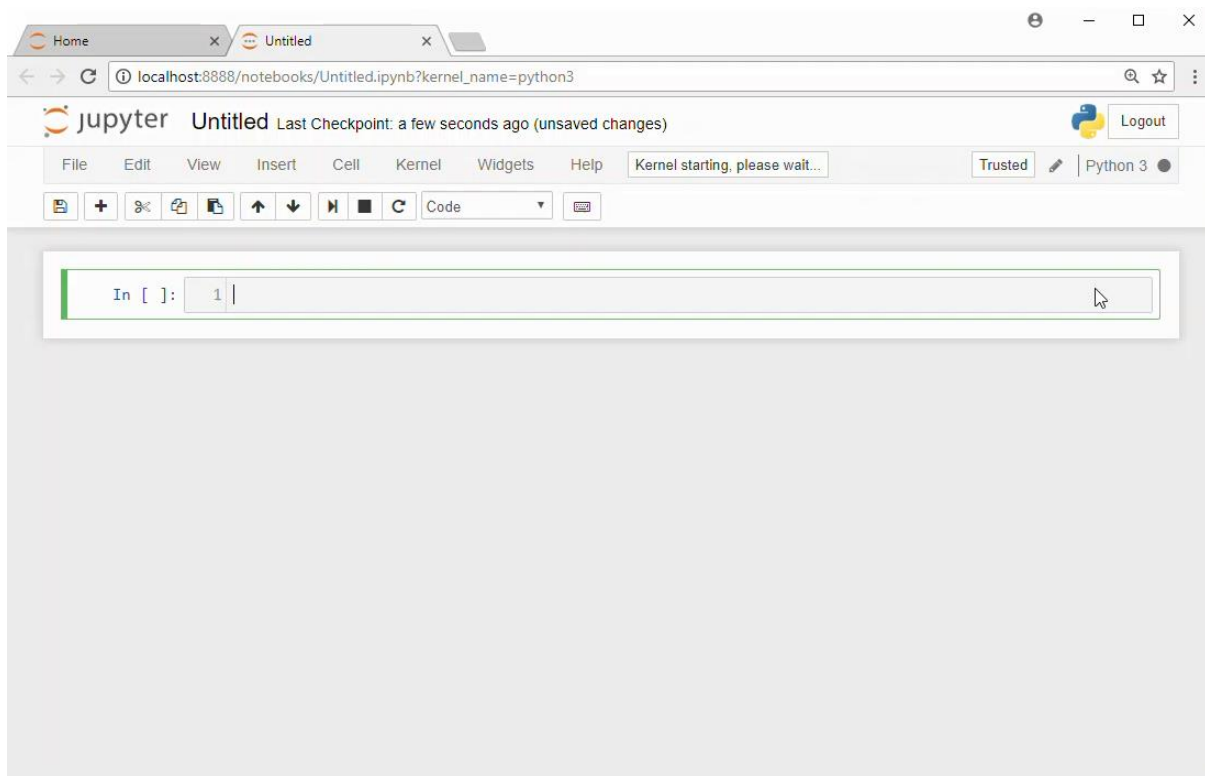
[I 16:14:12.661 NotebookApp] Accepting one-time-token ...

A web browser should open, and you should be able to see the **Jupyter file browser**. If a web browser doesn't open automatically, you can copy the web address from the **Anaconda Prompt** and paste it into a web browser's address bar.



In the upper right select **[New]** --> **[Python 3]**

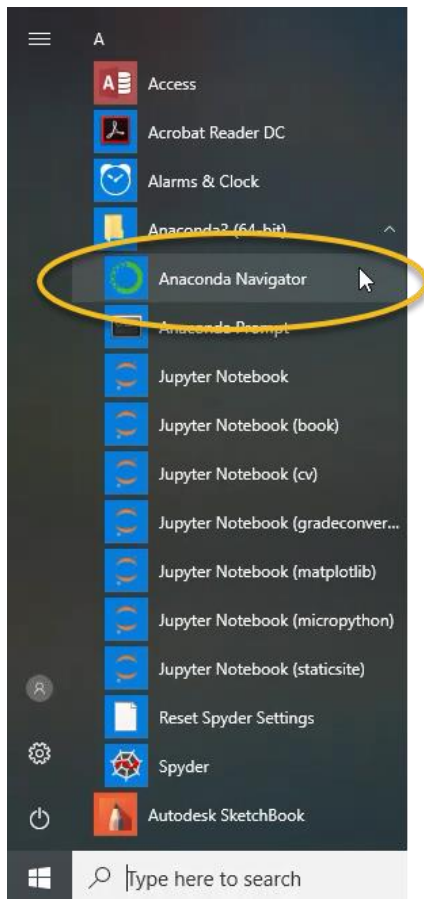
You will see a new tab open in your web browser. This web browser page is a **Jupyter notebook**.



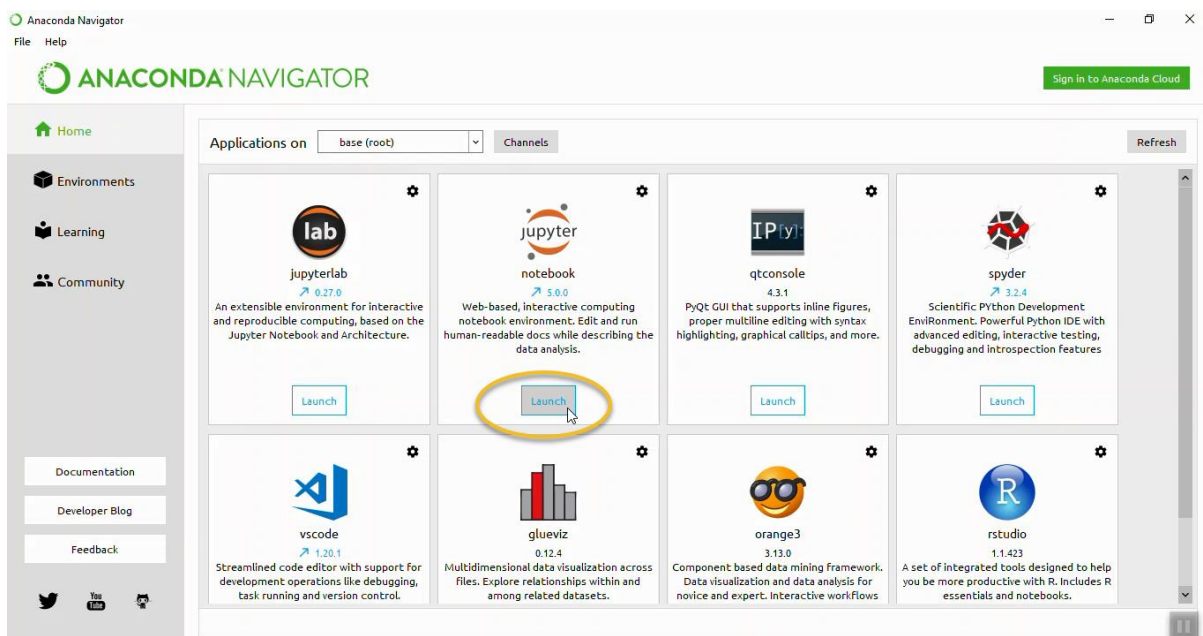
### Open a Jupyter Notebook with Anaconda Navigator

One additional way to open a Jupyter notebook is to use **Anaconda Navigator**. Anaconda Navigator comes with the Anaconda distribution of Python. Open **Anaconda Navigator** using the Windows start menu and select **[Anaconda3(64-bit)]** --> **[Anaconda Navigator]**.





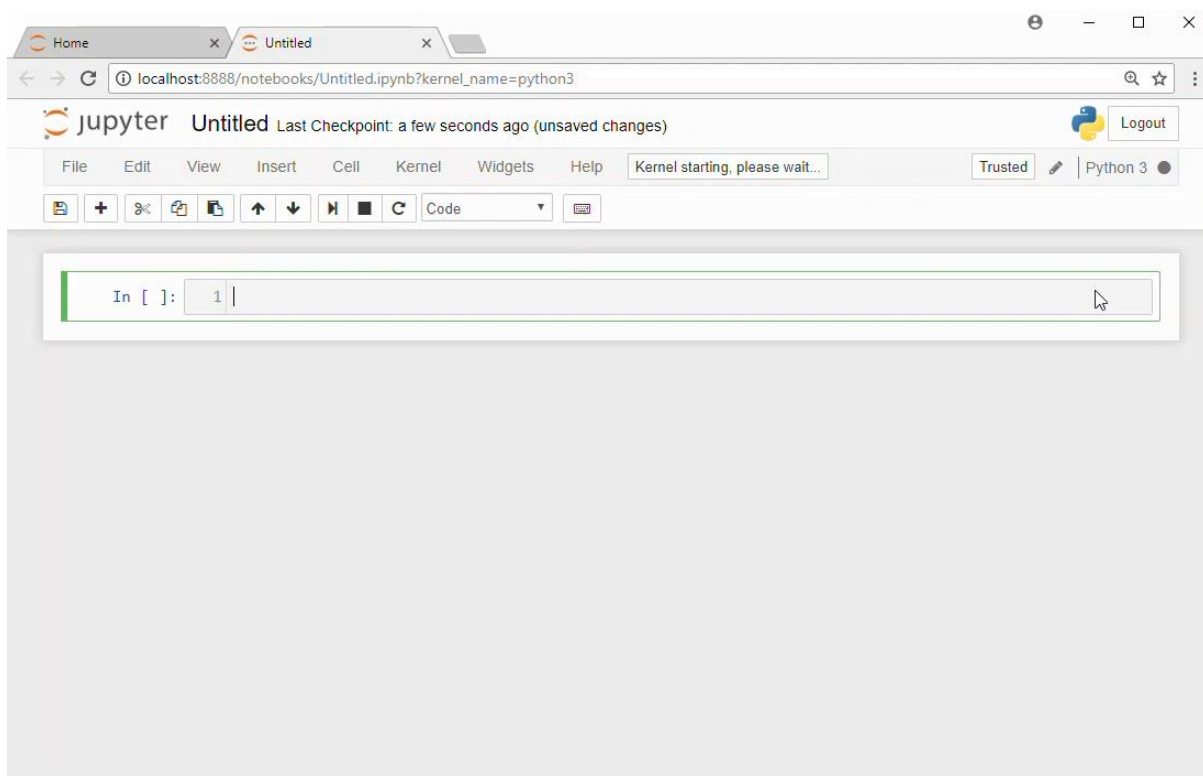
An **Anaconda Navigator** window will open. In the middle of the page, in the **Jupyter notebook** tile, click **[Launch]**



A **Jupyter file browser** will open in a web browser tab.  
In the upper right select **[New]** --> **[Python 3]**



A new notebook will open as a new tab in your web browser.



## The Jupyter Notebook Interface

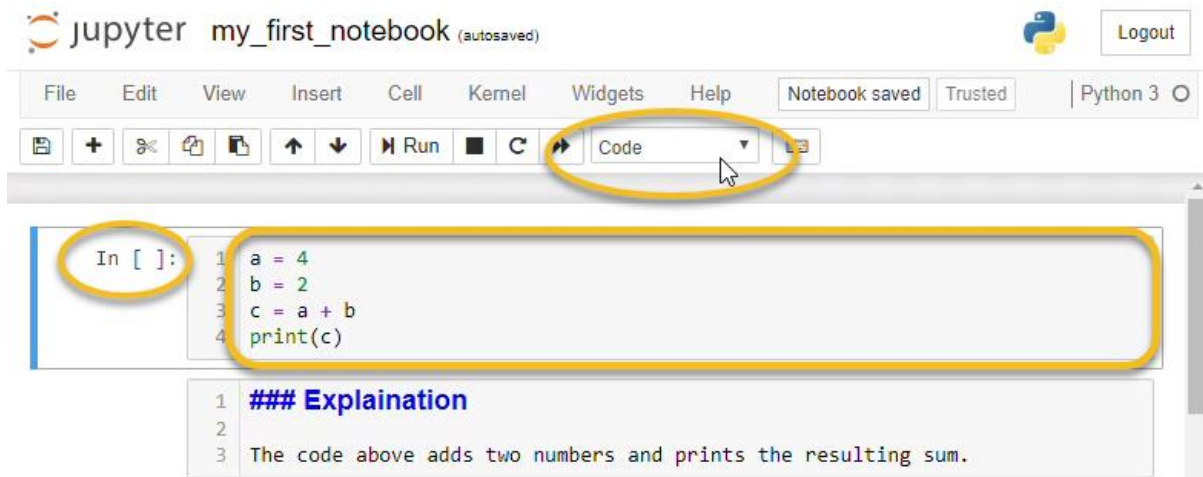
When a new Jupyter notebook opens, you will see the Jupyter notebook interface. Across the top of the notebook you see the Jupyter icon and the notebook name. You can click on the notebook name field and change the name of the notebook. Note that the file extension `.ipynb` is not printed in the file name field, but if you look in the Home tab, you will see that the notebook is saved with the `.ipynb` extension.

### Menus and Buttons

A Jupyter notebook is comprised of a bunch of *cells* which are arrayed one after another in boxes below the menu items and buttons. There are three main types of cells: code cells, output cells, and markdown cells.

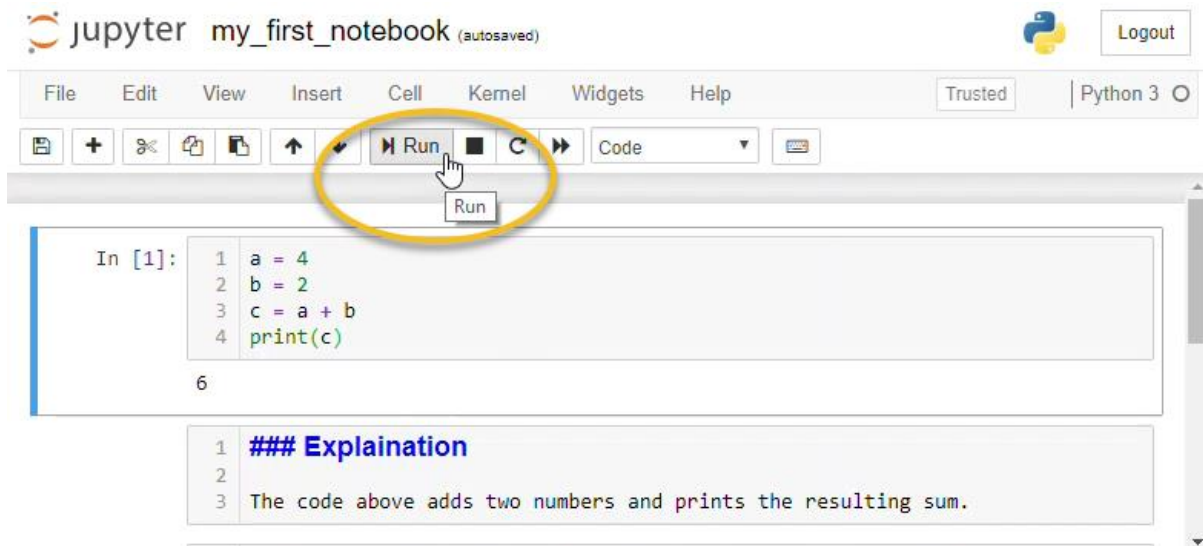
### Code Cells

In code cells, you can write Python code, then execute the Python code and see the resulting output. An example of a code cell is shown below.



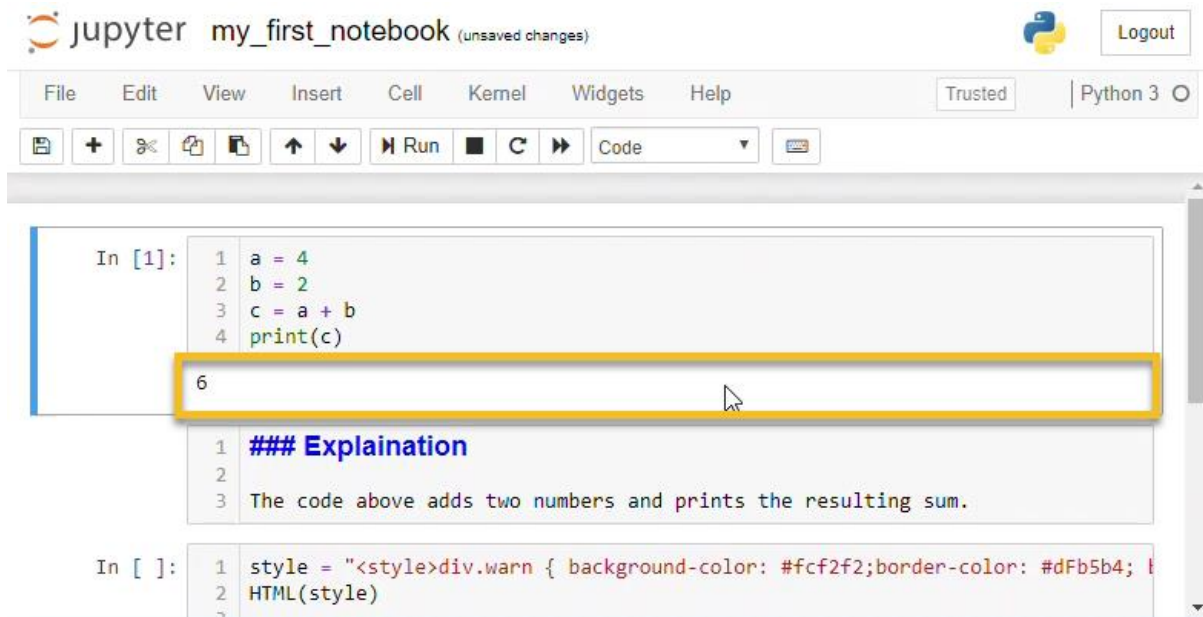
You can tell you are typing in a code cell because In [ ]: is shown to the left of the cell and the cell-type drop-down menu shows **Code**.

To run the Python code in a code cell push the [Run] button or type [Shift]+[Enter]. Hitting [Enter] when the cursor is inside a code cell brings the cursor down to a new line.

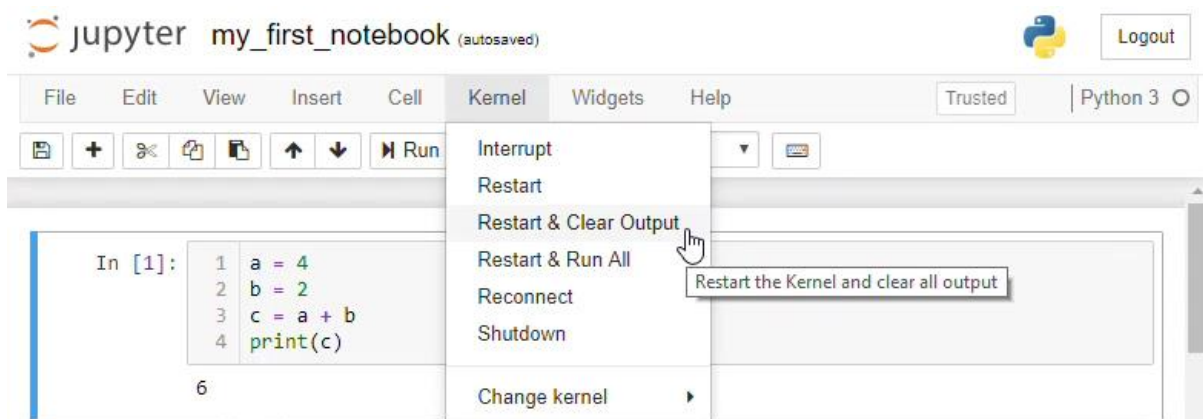


## Output Cells

After a code cell is run, an output cell can be produced below the code cell. The output cell contains the output from the code cell above it. Not all code produces output, so not all code cells produce output cells. The results in output cells can't be edited. If a code cell produces plots, charts or images, these outputs are shown in output cells.

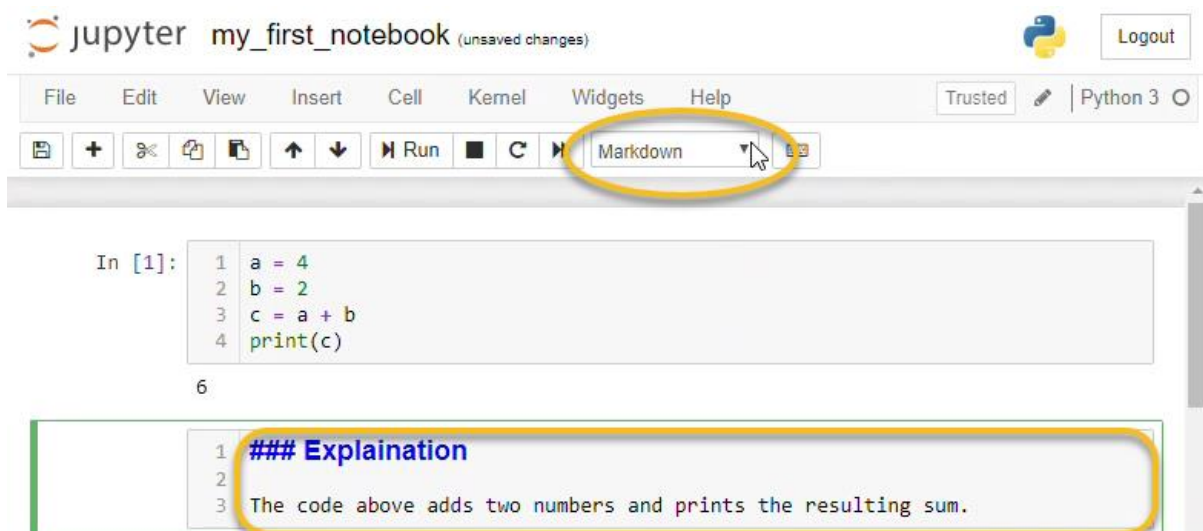


You can clear all the output cells and re-run code cells by selecting **[Kernel] --> [Restart Kernel and Clear Output]**.



## Markdown Cells

Markdown cells don't contain Python code. Markdown cells contain text written in Markdown format. Text in markdown cells can be formatted to show **bold** or *italic* text. Tables, images, and lists can also be included in markdown cells.



Markdown cells are used for documentation and explaining your code. The text in a markdown cell is not executed. Markdown cells can be formatted with a few special characters.

Markdown cells are run like code cells. The difference is that when markdown cells are run, the text is formatted (when code cells run, code is executed). Markdown cells are run by clicking the [Run] button or by pressing [Shift] + [Enter].

Text in markdown cells can be formatted using *markdown syntax*. An example of markdown syntax is putting an underscore before and after a word to cause the word to be formatted in *italics*.

## Headings

Headings are created in markdown cells using the hash symbol #. One # is the largest heading. Four hashes #### is the smallest heading.

```
# H1 Heading
## H2 Heading
### H3 Heading
#### H4 Heading
```

## Code Blocks

Code blocks can be inserted in Jupyter notebook markdown cells. For inline code blocks use the ` left quote character, the character to the left of the number [1] and above [Tab] on most keyboards.

This is inline code: `` `Inline code block` `` within a paragraph

For a separated code block use three ` left quote characters on one line, followed by the code block on separate lines. Terminate the separate code block with a line of three ` left quote characters.

```
```
```

Separated code block

```
```
```

The code in markdown cell code blocks do not execute when the markdown cell is run. A code block in a markdown cell is formatted when the markdown cell executes.

## Bold and Italics

**Bold** and *italic font* is displayed by surrounding text with a double asterisk for ***bold*** and a single underscore for *italics*

***bold*** produces **bold**

*italics* produces *italics*

***bold and italic*** produces ***bold and italic***

## Tables

Tables are displayed using the pipe | character, which is [Shift] + [\\] on most keyboards. Columns are separated by pipes | and rows are separated by lines. After the header row, a row of pipes and dashes --- are needed to define the table.

```
| header1 | header 2 | header 3 |
| --- | --- | --- |
| col 1 | col 2 | col 3 |
| col 1 | col 2 | col 3 |
```

produces:

header1	header 2	header 3
col 1	col 2	col 3
col 1	col 2	col 3

### Bullet Points and Lists

Bullet points are produced using the asterisk character \*

```
* item 1
* item 2
* item 3
```

produces

- item 1
- item 2
- item 3

Numbered lists are produced using sequential numbers followed by a dot. Indent sub-items with two spaces.

```
1. First item
2. Second item
3. Third item
  1. sub item
  2. sub item
    1. sub-sub item
    2. sub-sub item
```

produces

1. First item
2. Second item
3. Third item
  - A. sub item
  - B. sub item
    - a. sub-sub item
    - b. sub-sub item

### Horizontal Rule

A horizontal rule is specified with three asterisks \*\*\* on a single line.

```
***
```

produces



### Links

Hyperlinks are specified using a set of square brackets [ ] followed by a pair of parenthesis ( ) The text inside the square brackets will be the link, the link address goes in the parenthesis.

```
[Python.org] (https://python.org/)
```

produces

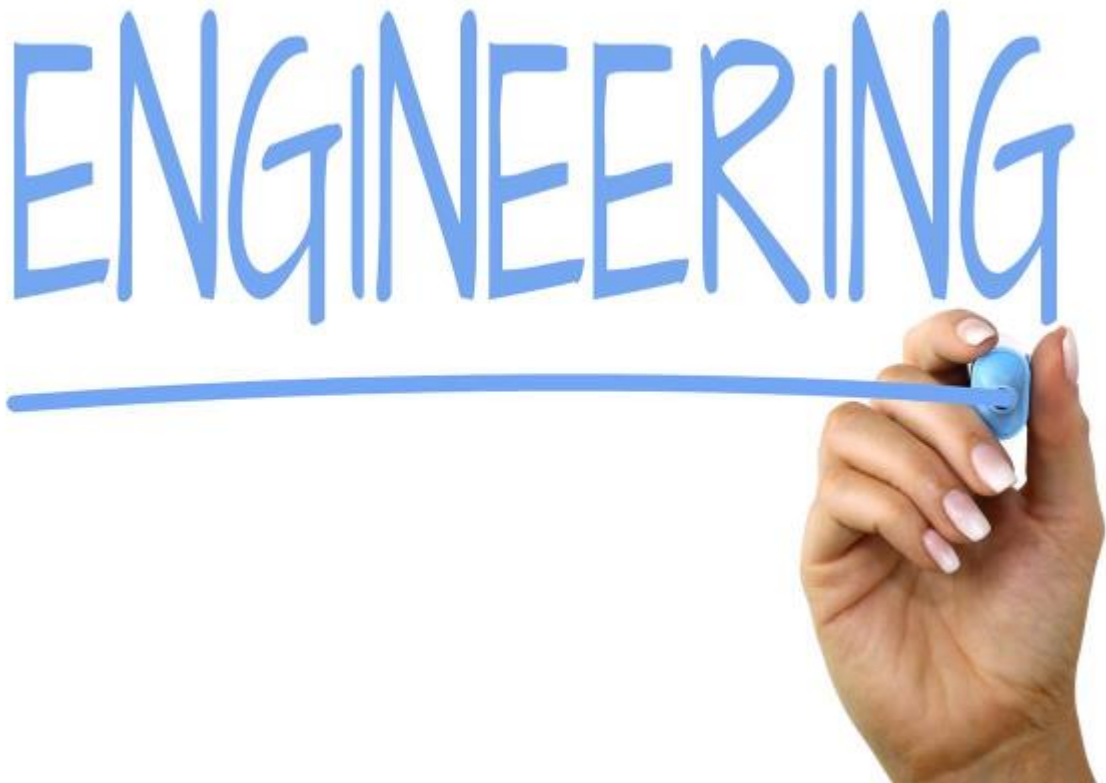
[Python.org](https://python.org/)

## Images

Images are embedded in Jupyter Notebook markdown using the exclamation point and square brackets `![ ]`, followed by the image file path in parenthesis `( )`. If the image can not be displayed, the text in square brackets will be shown. The image can be in the same directory as the notebook, or a relative path can be specified. In this case, the image `engineering.png` is stored in the `images` directory, which is a subdirectory of the directory the notebook is saved in.

```
![Engineering Image](images/engineering.png)
```

displays the image



## LaTeX Math

LaTeX Math equations and symbols are rendered by markdown cells. A more extensive list of LaTeX commands can be found in the appendix.

```
$$ \int_a^b \frac{1}{x^2} dx $$
```

produces

## html

Because Jupyter notebooks are rendered by web browsers, just about any HTML tag can be included in the markdown portion of a notebook. An example of an HTML tag is the `<sup></sup>` tags that surround superscript text.

```
x<sup>2</sup>
```

produces

$x^2$

Text can be colored using html `<font></font>` tags

```
<font color=red>Red Text</font>
```

produces



<font color=red>Red Text</font>

### Warning Boxes

bootstrap style warning boxes can be included in Jupyter notebook markdown using <div> tags

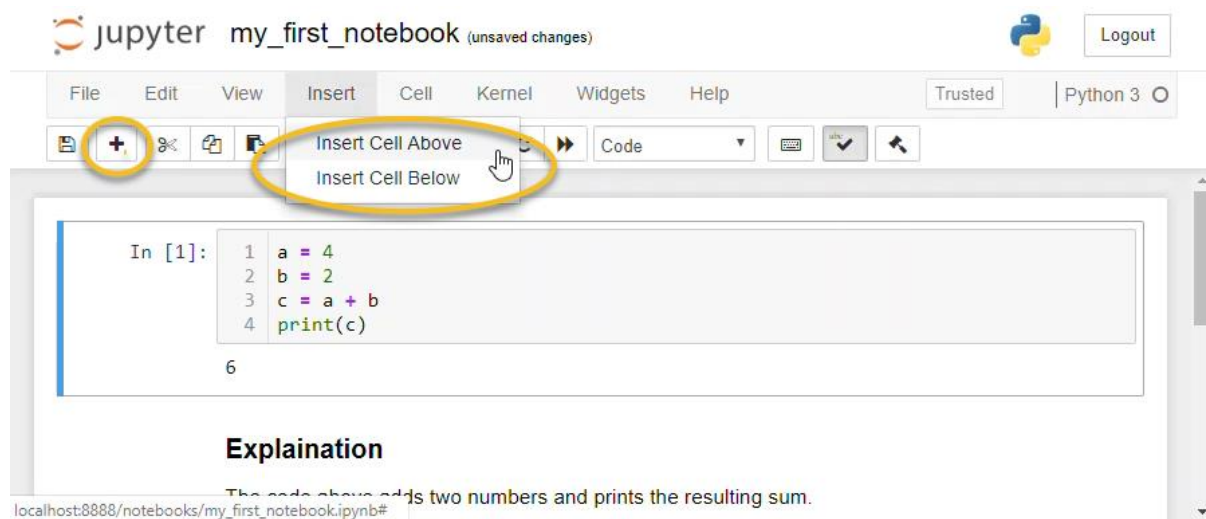
```
<div class="alert alert-danger" role="alert">
  <strong>Warning!</strong> Python lists start at 0
</div>
```

produces

**Warning!** Python lists start at 0

### Creating a new cell

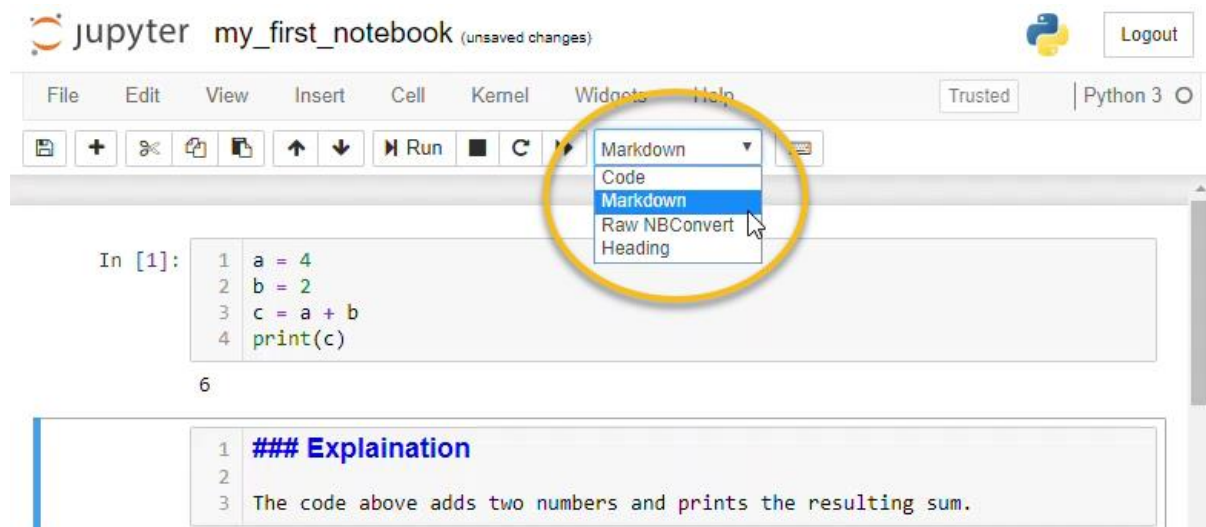
You can create a new cell in a Jupyter Notebook by clicking the [+] button in the upper menu. Clicking the [+] button produces a new code cell below the active cell.



You can also create a new cell using **Insert --> Insert Cell Above** or **Insert Cell Below**. You can choose to insert a cell above or below the active cell.

### Changing the cell type

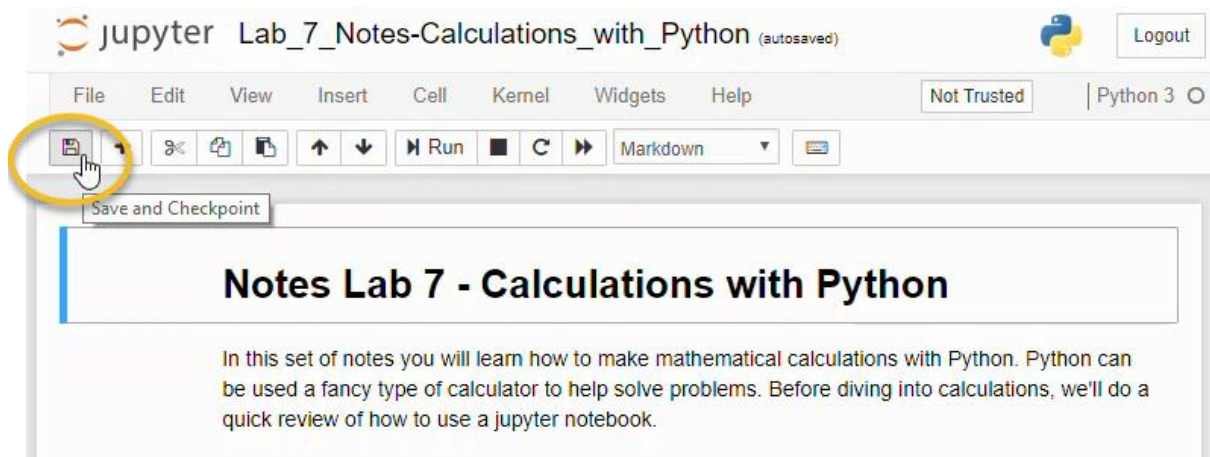
The type of cell: code cell or markdown cell, is changed by clicking on a cell and selecting the cell type from the drop-down menu. Typing [Esc] + [m] changes the cell type to a markdown cell. Typing [Esc] + [y] changes the cell type to a code cell.



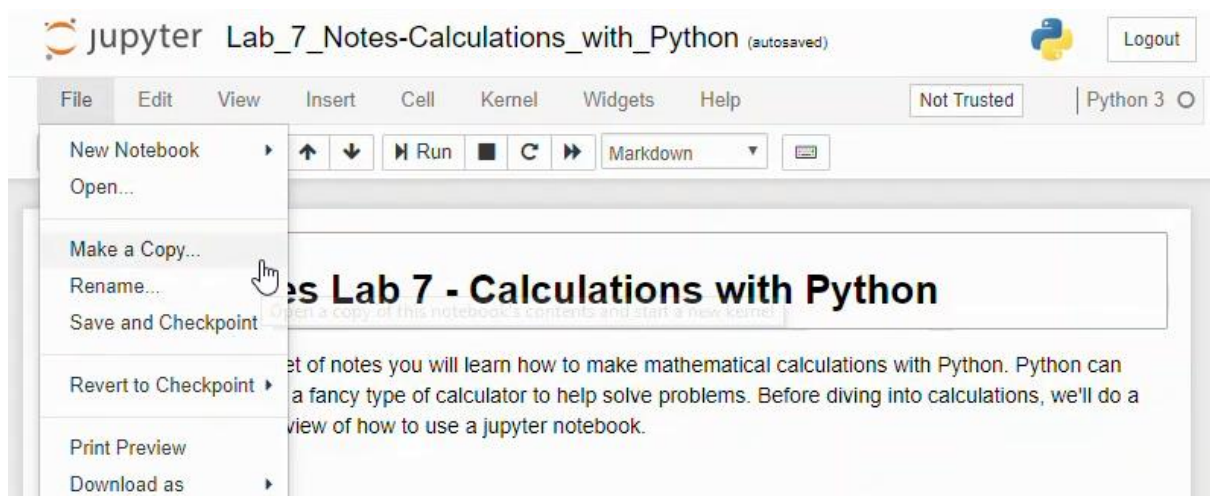
### Saving a Jupyter Notebook

Jupyter notebooks can be saved using the save icon in the upper menu or by pressing [Ctrl] + [s].



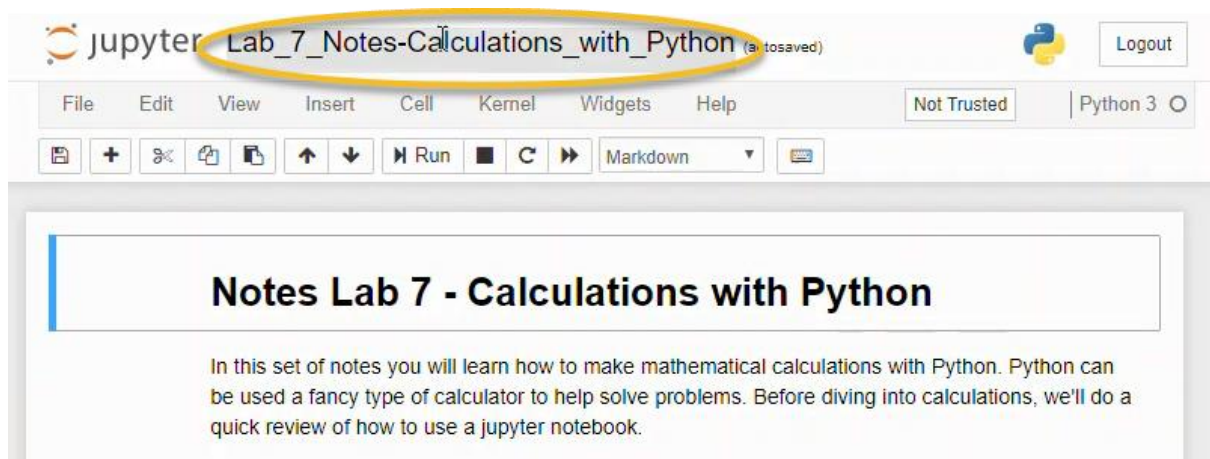


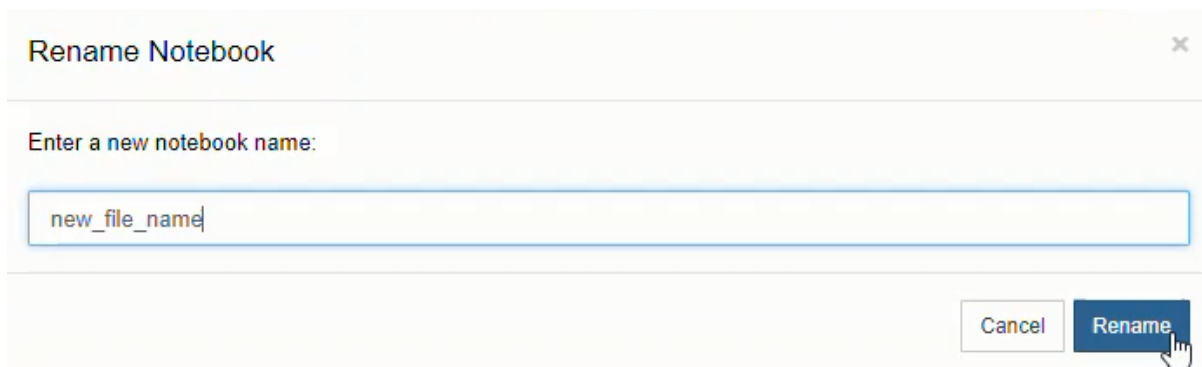
Jupyter notebooks can also be saved as a copy, similar to the Save As command common in many programs. To save a copy of a Jupyter notebook use **File --> Make a Copy...**



### Renaming a Jupyter Notebook

Jupyter notebooks are renamed by clicking on the notebook name above the upper menu and typing a new name into the dialog box.

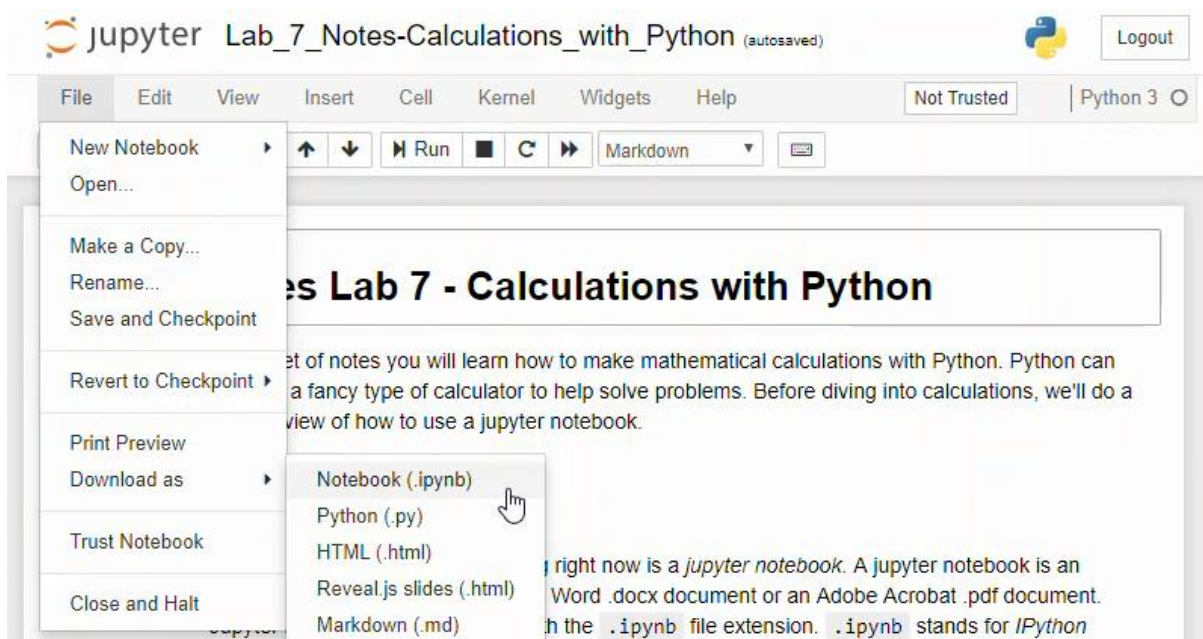




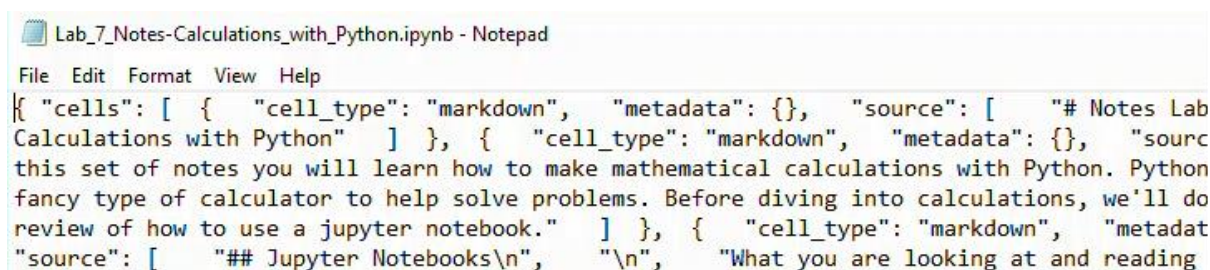
A dialog box titled "Rename Notebook" with a close button (X) in the top right corner. Below the title, it says "Enter a new notebook name:". There is a text input field containing the text "new\_file\_name". At the bottom right, there are two buttons: "Cancel" and "Rename". A mouse cursor is pointing at the "Rename" button.

## Downloading a Jupyter Notebook

Jupyter notebooks can be downloaded and saved using **File --> Download As --> Notebook (.ipynb)**. Selecting this menu option will download the notebook as a .ipynb file.

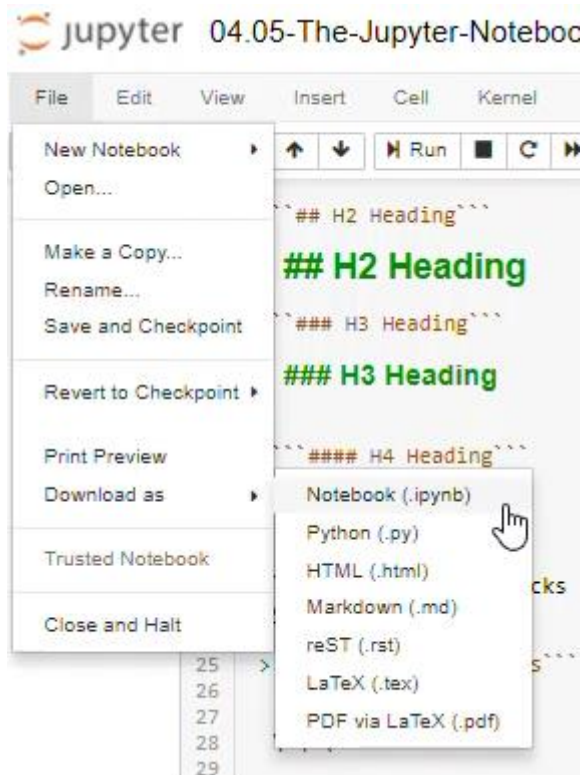


Note that when a .ipynb file is viewed in a text editor like notepad, the notebook is unformatted and looks like a confusing jumble of text. The notebook needs to be opened in a Jupyter notebook file browser in order for the code in the notebook to run and the markdown text to render.



## Saving Jupyter Notebooks in Other Formats

Jupyter notebooks can be saved in other formats besides the native .ipynb format. These formats can be accessed using the **[File] --> [Download As]** menu.



The available file download types are:

- Notebook (.ipynb) - The native jupyter notebook format
- Python (.py) - The native Python code file type.
- HTML (.html) - A web page
- Markdown (.md) - Markdown format
- reST (.rst) - Restructured text format
- LaTeX (.tex) - LaTeX Article format
- PDF via LaTeX (.pdf) - a pdf exported from LaTeX, requires a converter

When a notebook is saved as a `.py` file, all text in markdown cells is converted to comments, and any code cells stay intact as Python code.

```

1  # coding: utf-8
2
3  # ## Tool to remove dashes from names of files and put in a dictionary where:
4  #
5  # Key: Filename-with-dashes-and-extensions, Value: Name with no dashes or extensions
6
7  # In[86]:
8
9
10 import os
11
12 # In[87]:
13
14
15 os.getcwd
16
17 # In[88]:
18
19
20 os.getcwd()
21
22 # In[89]:
23
24
25 os.pardir

```

## Magic Commands

Jupyter notebook code cells can contain special commands which are not valid Python code but affect the behavior of the notebook. These special commands are called *magic commands*.

### %matplotlib inline

One of the most popular magic commands is:

```
%matplotlib inline
```

Entering the %matplotlib inline command at the top of a Jupyter notebook renders Matplotlib plots in cells of the notebook. Without %matplotlib inline, plots may jump out as external windows. A typical start to a Jupyter notebook using **Matplotlib** might start as:

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

```

### %load

The %load command loads a Python module, webpage or file into a Jupyter notebook. If there is a file called **hello.py** in the same directory as the notebook with some Python code written in it, we can load that same code into a Jupyter notebook code cell with the %load command.

Within a Jupyter notebook code cell type the command:

```
%load hello.py
```

The result is the code from the file **hello.py** is copied into the current notebook.

In [1]:

```

# %load hello.py
# hello.py

print('This code was run from a separate Python file')
print('Hello from the file hello.py')

```

This code was run from a separate Python file

Hello from the file hello.py

### %run

If the %run magic command followed by the name of a valid Python file, the Python file runs as a script. Suppose the file **hello.py** is created in the same directory as the running Jupyter notebook. The directory structure will look something like this:

```
| folder
---| notebook.ipynb
---| hello.py
```

In the file **hello.py** is the code:

```
# hello.py
```

```
print('This code was run from a separate Python file')
print('Hello from the file hello.py')
```

Within our Jupyter notebook, if we %run this file, we get the output of the **hello.py** script in a Jupyter notebook output cell.

In [2]:

```
%run hello.py
```

This code was run from a separate Python file

Hello from the file hello.py

### Other useful magic commands

Below is a table of other useful Jupyter notebook magic commands

Magic command	Result
%pwd	Print the current working directory
%cd	Change the current working directory
%ls	List the contents of the current directory
%history	Show the history of the In [ ] : commands

You can list all of the available magic commands by typing and running %lsmagic in a Jupyter notebook code cell:

```
%lsmagic
```

The output shows all the available line magic commands that begin with the percent sign %.

Available line magics:

```
%alias %alias_magic %autocall %automagic %autosave ...
%dhist %dirs %doctest_mode %ed %edit %env %gui ...
dir %more %mv %notebook %page %pastebin %pdb %pdef ...
...
```

Available cell magics:

```
%%! %%HTML %%SVG %%bash %%capture %%debug %%file %%html .
..
%%python %%python2 %%python3 %%ruby %%script %%sh %%svg ..
.
```

# Getting Help in a Jupyter Notebook

There are a couple of different ways to get help when using a Jupyter notebook.

## Get help using dir

Typing `dir()` and passing in a function, method, variable or object shows the possible object, method and function calls available to that object. For example, we can investigate the different functions in the **glob** module, part of Python's Standard Library, by importing `glob`, then calling `dir(glob)`.

In [1]:

```
import glob
dir(glob)
```

Out[1]:

```
['__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '_glob0',
 '_glob1',
 '_glob2',
 '_iglob',
 '_ishidden',
 '_isrecursive',
 '_iterdir',
 '_rlistdir',
 'escape',
 'fnmatch',
 'glob',
 'glob0',
 'glob1',
 'has_magic',
 'iglob',
 'magic_check',
 'magic_check_bytes',
 'os',
 're']
```

## Get help using Tab

After typing the name of a variable, object or function following the `.` character hit the [Tab] key. Typing [Tab] brings up a list of available options. Scroll through the list or type a letter to filter the list to certain starting letters. Use [Enter] to select the option you want.

Tab completion can also be used during module import. Hit [Tab] after typing the module name to see which functions and classes are available in that module.

```
from math import <tab>
```

## Get help using the help() function

After importing a module, you can use the `help()` function to see documentation about the command if it is available.

In [2]:



```
import math
help(math.sin)
Help on built-in function sin in module math:

sin(...)
    sin(x)

    Return the sine of x (measured in radians).
```

After importing a module, you can view help on the imported module by typing the module name followed by a question mark ?

In [3]:

```
import statistics
statistics.mean?
Signature: statistics.mean(data)
Docstring:
Return the sample arithmetic mean of data.

>>> mean([1, 2, 3, 4, 4])
2.8

>>> from fractions import Fraction as F
>>> mean([F(3, 7), F(1, 21), F(5, 3), F(1, 3)])
Fraction(13, 21)

>>> from decimal import Decimal as D
>>> mean([D("0.5"), D("0.75"), D("0.625"), D("0.375")])
Decimal('0.5625')
```

If ``data`` is empty, `StatisticsError` will be raised.  
File: ~/anaconda3/envs/book/lib/python3.6/statistics.py  
Type: function

You can view the source code where a particular function is defined using a double question mark ??

In [4]:

```
import statistics
statistics.mean??
Signature: statistics.mean(data)
Source:
def mean(data):
    """Return the sample arithmetic mean of data.

    >>> mean([1, 2, 3, 4, 4])
    2.8

    >>> from fractions import Fraction as F
    >>> mean([F(3, 7), F(1, 21), F(5, 3), F(1, 3)])
    Fraction(13, 21)

    >>> from decimal import Decimal as D
    >>> mean([D("0.5"), D("0.75"), D("0.625"), D("0.375")])
    Decimal('0.5625')
```

If ``data`` is empty, `StatisticsError` will be raised.  
"""  
if iter(data) is data:

```
data = list(data)
```

## Help online

Help is also available online at in the official Jupyter documentation:

<http://jupyter.readthedocs.io/en/latest/>

You can always try to find help by typing something into Google. The site Stack Overflow is devoted to programming questions and answers. The highest rated answers on Stack Overflow are at the top of each question page.

## Summary

In this chapter, you learned about Jupyter notebooks. You learned what a Jupyter notebook is and why Jupyter notebooks are useful for problem solvers. This chapter showed how to install Jupyter notebooks on Windows, MacOS, and Linux.

Some specific operations with Jupiter notebooks were introduced:

- Open a Jupyter Notebook
- Rename a Jupyter Notebook
- Write Python code in a Jupyter notebook code cells
- Run Python code in a Jupyter notebook code cell
- Write text in Jupyter notebook markdown cells
- Use markdown syntax to produce formatted text, headings, lists, and tables
- Save a Jupyter notebook
- Download a Jupyter notebook in different file types

You also learned about special "magic" commands that can be used in a Jupyter notebook. The final section of the chapter detailed a couple of ways to get help when working with Jupyter notebooks.

## Key Terms and Concepts

Jupyter  
notebook  
Jupyter notebook  
kernel  
iPython  
IDE  
text editor  
markdown  
execute  
Anaconda Prompt  
file browser  
code cell  
markdown cell  
code block  
inline code block  
pipe character  
hyperlink  
LaTeX  
HTML tag  
.ipynb-file  
.py-file  
.md-file  
magic commands



## Python Commands and Functions

### Jupyter Notebook Magic Commands

Command	Description
<code>%matplotlib inline</code>	Display plots in output cells
<code>%run file.py</code>	Run file.py and displays output
<code>%pwd</code>	Print the working directory file path
<code>%ls</code>	List contents of the current working directory
<code>%precision</code>	Set float point precision for pretty printing
<code>%whos</code>	List variables and types in the running kernel session
<code>function?</code>	Display help on a function
<code>function??</code>	Display source code of a function