Module 1: Getting started with Data Science and Python

1.0. Discovering the match between data science and Python:

1.0.1. Discovering the Wonders of Data Science

Data Science is transforming the world by turning raw data into meaningful insights. It powers recommendations on Netflix, personalized ads on social media, weather predictions, and even financial risk analysis. At its core, data science is about asking the right questions, collecting relevant data, and using computational techniques to extract knowledge and make informed decisions.

Some of the *wonders* of data science include:

- **Predictive analytics** to forecast trends and behaviours.
- **Natural language processing (NLP)** to understand and generate human language.
- **Computer vision** to analyze and interpret visual information.
- **Automation and decision-making** powered by intelligent algorithms.

These capabilities empower businesses, governments, healthcare, and nearly every other industry to innovate and improve decision-making.

1.0.2. Exploring How Data Science Works

Data Science is not a single tool or algorithm—it's a **process**. The data science workflow typically includes the following steps:

1. **Data Collection**
   Gathering structured or unstructured data from databases, APIs, sensors, or web scraping.
2. **Data Cleaning & Preparation**
   Removing inconsistencies, handling missing values, formatting data, and converting it into a usable form.
3. **Exploratory Data Analysis (EDA)**
   Using statistics and visualization to uncover patterns, trends, and relationships.
4. **Model Building**
   Applying machine learning algorithms to make predictions or classifications.
5. **Evaluation & Optimization**
   Testing the model's accuracy and refining it to improve performance.
6. **Communication**
   Presenting results using dashboards, visualizations, or reports that stakeholders can understand and act upon.

This lifecycle helps transform raw data into actionable insights that drive strategic decisions.

1.0.3. Creating the Connection Between Python and Data Science

Python has become the **de facto language** of data science, and for good reasons:

- **Simplicity and readability** make it accessible to both beginners and professionals.
- **Rich libraries** such as:
  - ✓ **NumPy** for numerical operations
  - ✓ **Pandas** for data manipulation
  - ✓ **Matplotlib** and **Seaborn** for data visualization
  - ✓ **Scikit-learn** for machine learning
  - ✓ **TensorFlow** and **PyTorch** for deep learning
- **Community support**: Python has a vast ecosystem of developers and data scientists constantly creating and sharing tools and knowledge.

One of Python's key strengths is its ability to serve as a **unifying language**, allowing seamless integration of data access, cleaning, analysis, modelling, and visualization in a single environment. The book also emphasizes Python's modularity, which means you can start small and gradually expand your toolkit as needed.

1.1. Setting Up Python for Data Science

Python Installation on Windows:

Step 1: Select Python Version

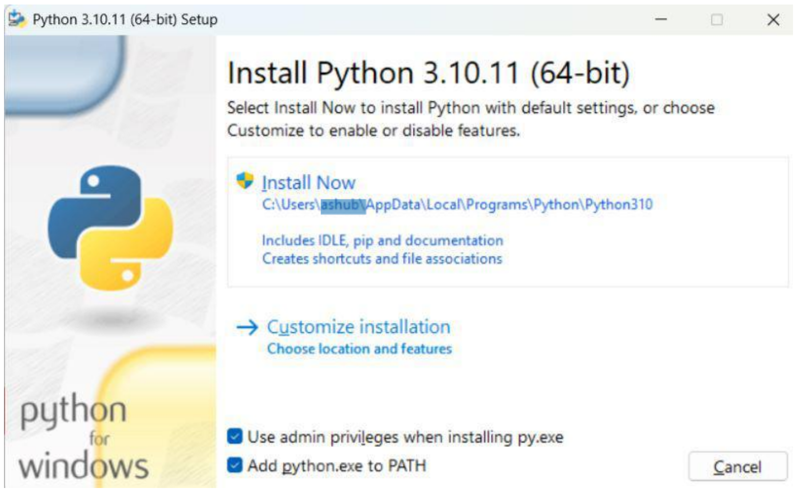
Deciding on a version depends on what you want to do in Python.

The two major versions are Python 2 and Python 3, but Python 2 is outdated and no longer supported.

If you're working on a legacy project, you may need Python 2, but for everything else, Python 3 is the better choice.

Also, choose a stable release over the newest one, as newer versions may have bugs and issues.

Step 2: Downloading the Python Installer


Python 3.10.11 (64-bit) Setup

Install Python 3.10.11 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

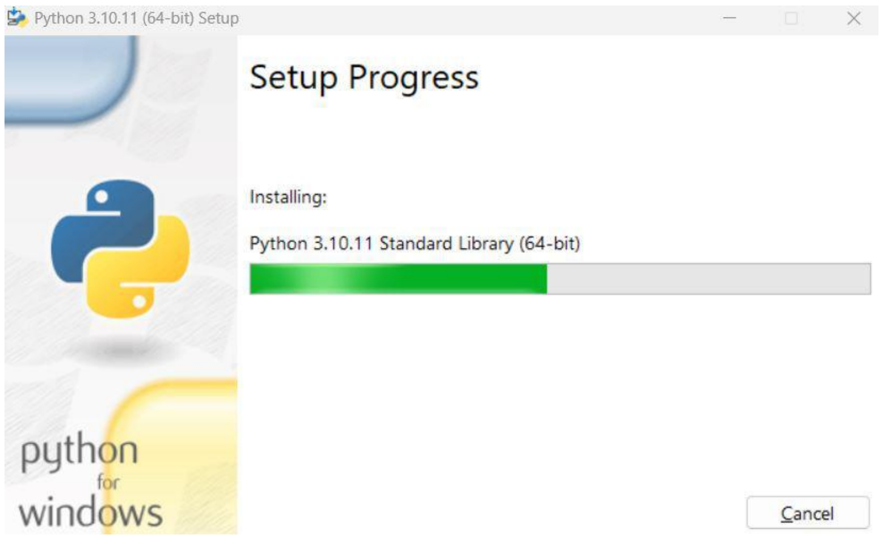
Install Now
C:\Users\ashub\AppData\Local\Programs\Python\Python310

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Use admin privileges when installing py.exe
☑ Add python.exe to PATH

After clicking the **Install Now Button,** the setup will start installing Python on your Windows system. You will see a window like this.



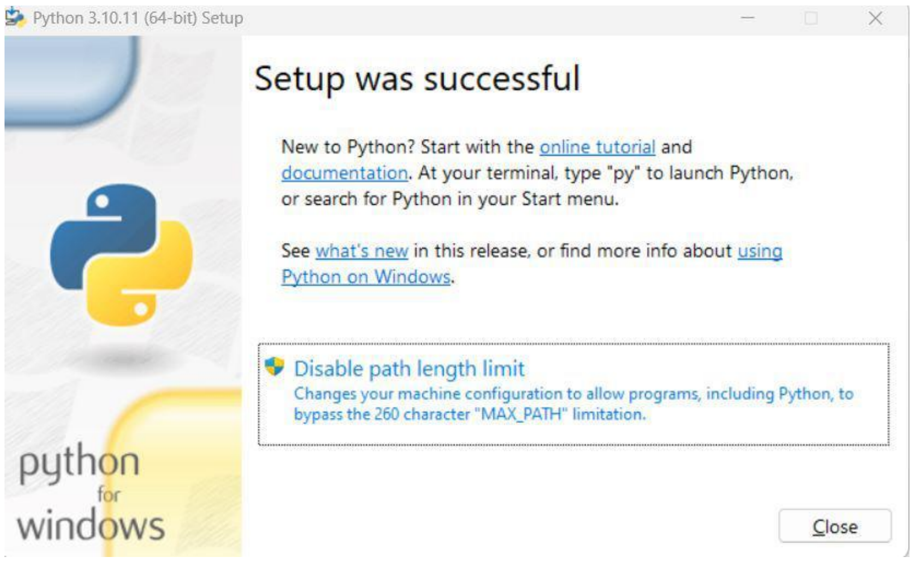
Python 3.10.11 (64-bit) Setup

Setup Progress

Installing:

Python 3.10.11 Standard Library (64-bit)

Step 3: Running the Executable Installer



Python 3.10.11 (64-bit) Setup

Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

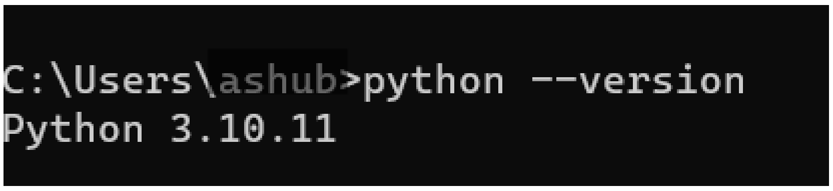
See what's new in this release, or find more info about using Python on Windows.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

Step 4: Verify the Python Installation in Windows

Close the window after the successful installation of Python. You can check if the installation of Python was successful by using either the command line or the **Integrated Development Environment (IDLE)**.

Command > python –version



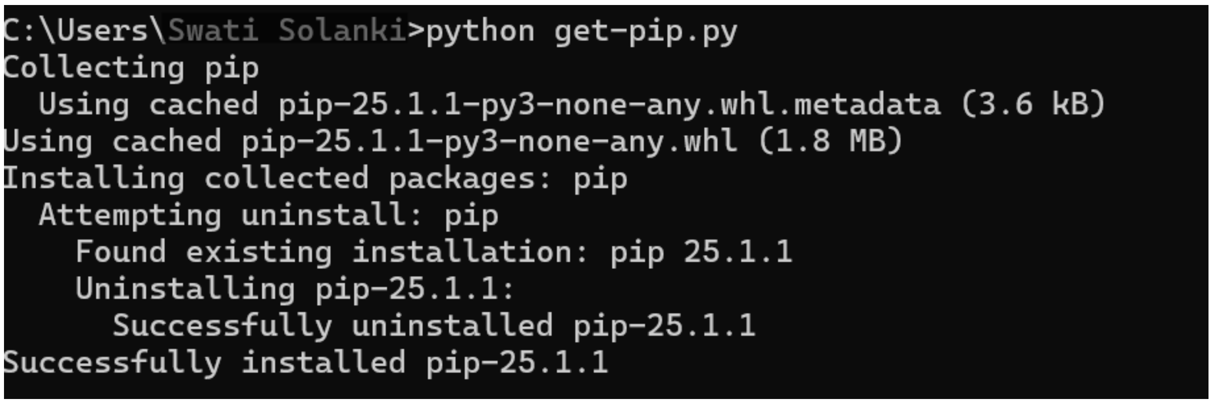
```
C:\Users\ashub>python --version
Python 3.10.11
```

1.2. Understanding the tools:

→ Install PIP (Python's package installer)

Download get-pip.py

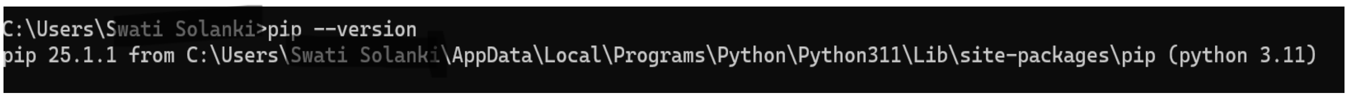
python get-pip.py



```
C:\Users\[redacted]>python get-pip.py
Collecting pip
  Using cached pip-25.1.1-py3-none-any.whl.metadata (3.6 kB)
Using cached pip-25.1.1-py3-none-any.whl (1.8 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 25.1.1
    Uninstalling pip-25.1.1:
      Successfully uninstalled pip-25.1.1
Successfully installed pip-25.1.1
```

Verifying the pip installation on Windows

pip –version



```
C:\Users\[redacted]>pip --version
pip 25.1.1 from C:\Users\[redacted]\AppData\Local\Programs\Python\Python311\Lib\site-packages\pip (python 3.11)
```

To upgrade the existing pip version

python –m pip install - -upgrade pip

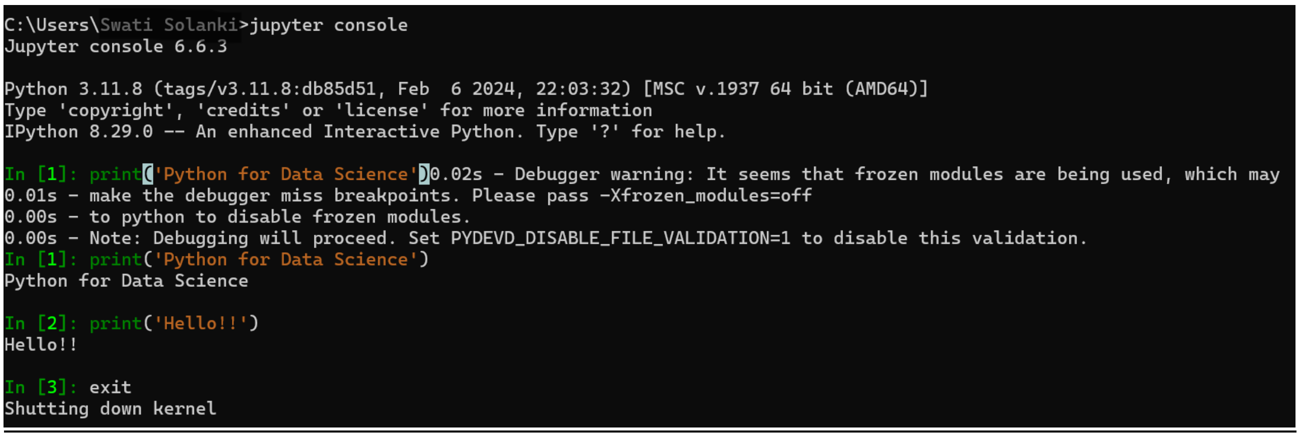
1.2.1. Using the Jupyter Console

The Python console is where you can experiment with data science interactively. You can try things and see the results immediately. If you make a mistake, you can simply close the console and create a new one. The console is for playing around and considering what might be possible.

Installing Jupyter Console

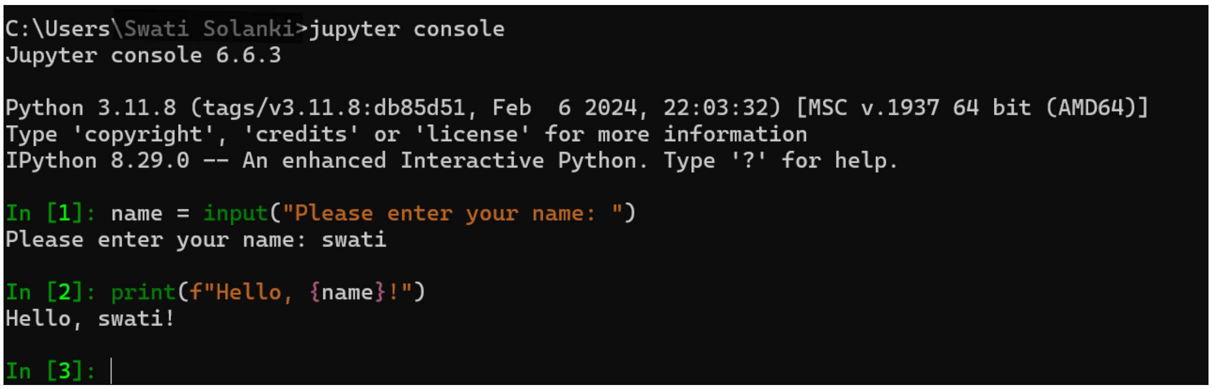
pip install jupyter-console

Starting with Jupyter console:



```
C:\Users\[redacted]>jupyter console
Jupyter console 6.6.3

Python 3.11.8 (tags/v3.11.8:db85d51, Feb  6 2024, 22:03:32) [MSC v.1937 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.29.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print('Python for Data Science')0.02s - Debugger warning: It seems that frozen modules are being used, which may
0.01s - make the debugger miss breakpoints. Please pass -Xfrozen_modules=off
0.00s - to python to disable frozen modules.
0.00s - Note: Debugging will proceed. Set PYDEVD_DISABLE_FILE_VALIDATION=1 to disable this validation.
In [1]: print('Python for Data Science')
Python for Data Science

In [2]: print('Hello!!')
Hello!!

In [3]: exit
Shutting down kernel
```

1.2.1.1. Interacting with screen text



```
C:\Users\Swati Solanki>jupyter console
Jupyter console 6.6.3

Python 3.11.8 (tags/v3.11.8:db85d51, Feb  6 2024, 22:03:32) [MSC v.1937 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.29.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: name = input("Please enter your name: ")
Please enter your name: swati

In [2]: print(f"Hello, {name}!")
Hello, swati!

In [3]: 
```

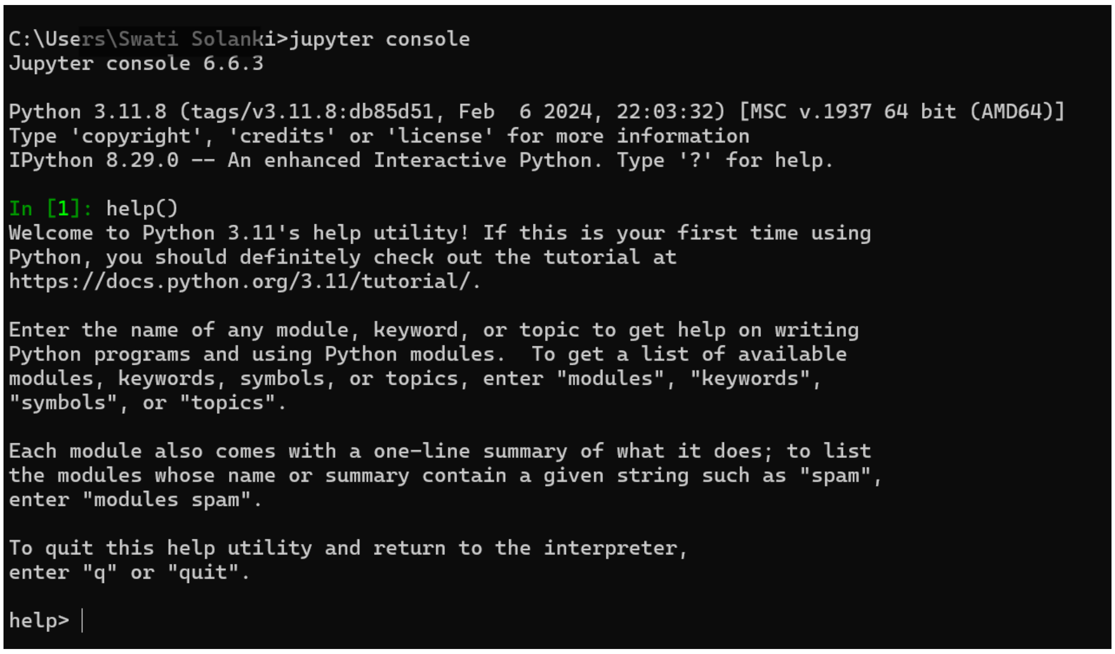
No one can remember absolutely everything about a programming language. Even the best coders have memory lapses. This is why having language-specific help is so important.

The Python portion of the IPython console provides two methods of getting help: help mode and interactive help.

1.2.1.2. Getting Python help

Entering help mode

To enter help mode, type help( ) and press Enter. The console enters a new mode, in which you can type help-related commands as needed to discover more about Python. You can't type Python commands in this mode. The prompt changes to a help> prompt, to remind you that you're in help mode.

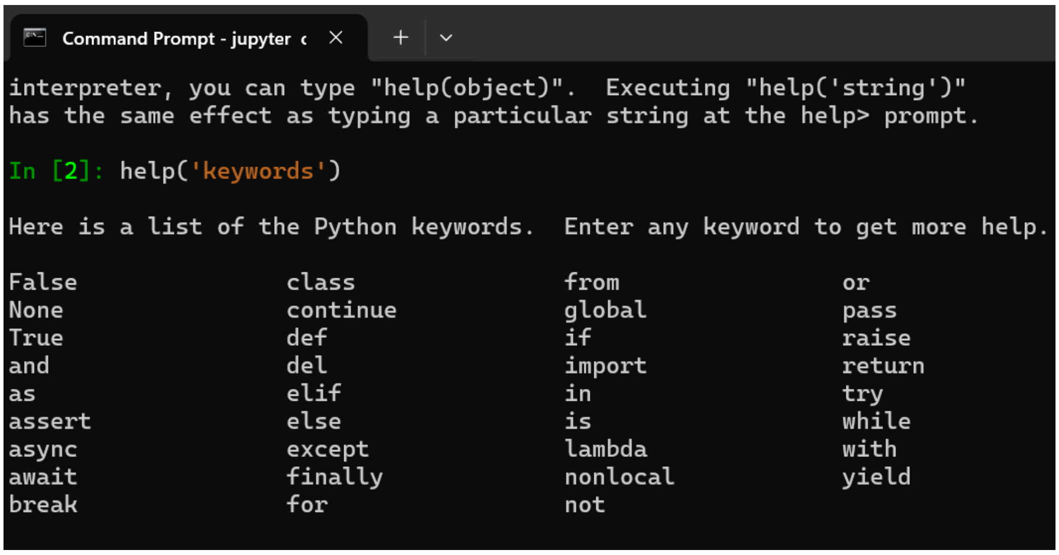


```
C:\Users\Swati Solanki>jupyter console
Jupyter console 6.6.3

Python 3.11.8 (tags/v3.11.8:db85d51, Feb  6 2024, 22:03:32) [MSC v.1937 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.29.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: help()
Welcome to Python 3.11's help utility! If this is your first time using
Python, you should definitely check out the tutorial at
https://docs.python.org/3.11/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To get a list of available
modules, keywords, symbols, or topics, enter "modules", "keywords",
"symbols", or "topics".

Each module also comes with a one-line summary of what it does; to list
the modules whose name or summary contain a given string such as "spam",
enter "modules spam".

To quit this help utility and return to the interpreter,
enter "q" or "quit".

help> 
```

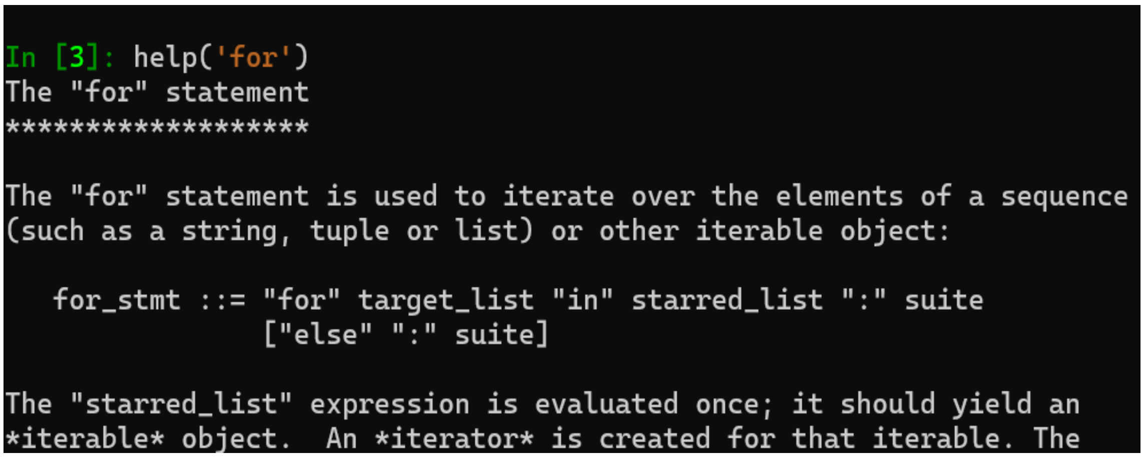
To obtain help about any object or command, simply type the object or command name and press Enter. You can also type any of the following commands to obtain a listing of other topics of discussion.

modules: Compiles a list of the currently loaded modules. This list varies by how your copy of Python (the underlying language) is configured at any given time, so the list won't be the same every time you use this command. The command can take a while to execute, and the output list is usually quite large.

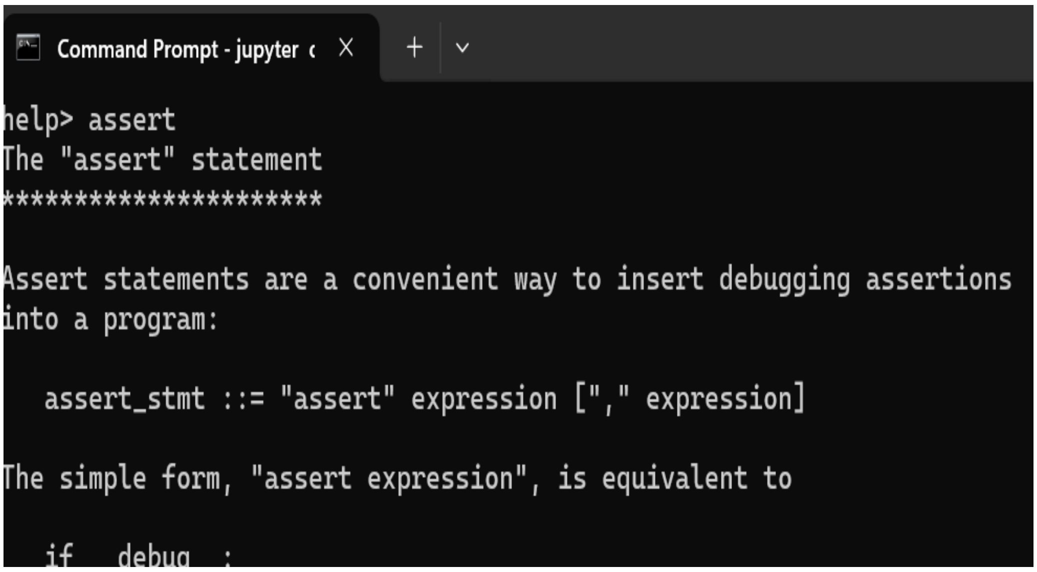
keywords: Presents a list of Python keywords that you can ask about. For example, you can type assert and learn more about the assert keyword.


Command Prompt - jupyter

```
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.

In [2]: help('keywords')

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```



```
In [3]: help('for')
The "for" statement
*******************

The "for" statement is used to iterate over the elements of a sequence
(such as a string, tuple or list) or other iterable object:

   for_stmt ::= "for" target_list "in" starred_list ":" suite
                ["else" ":" suite]

The "starred_list" expression is evaluated once; it should yield an
*iterable* object.  An *iterator* is created for that iterable. The
```


Command Prompt - jupyter

```
help> assert
The "assert" statement
**********************

Assert statements are a convenient way to insert debugging assertions
into a program:

   assert_stmt ::= "assert" expression ["," expression]

The simple form, "assert expression", is equivalent to

   if   debug  :
```

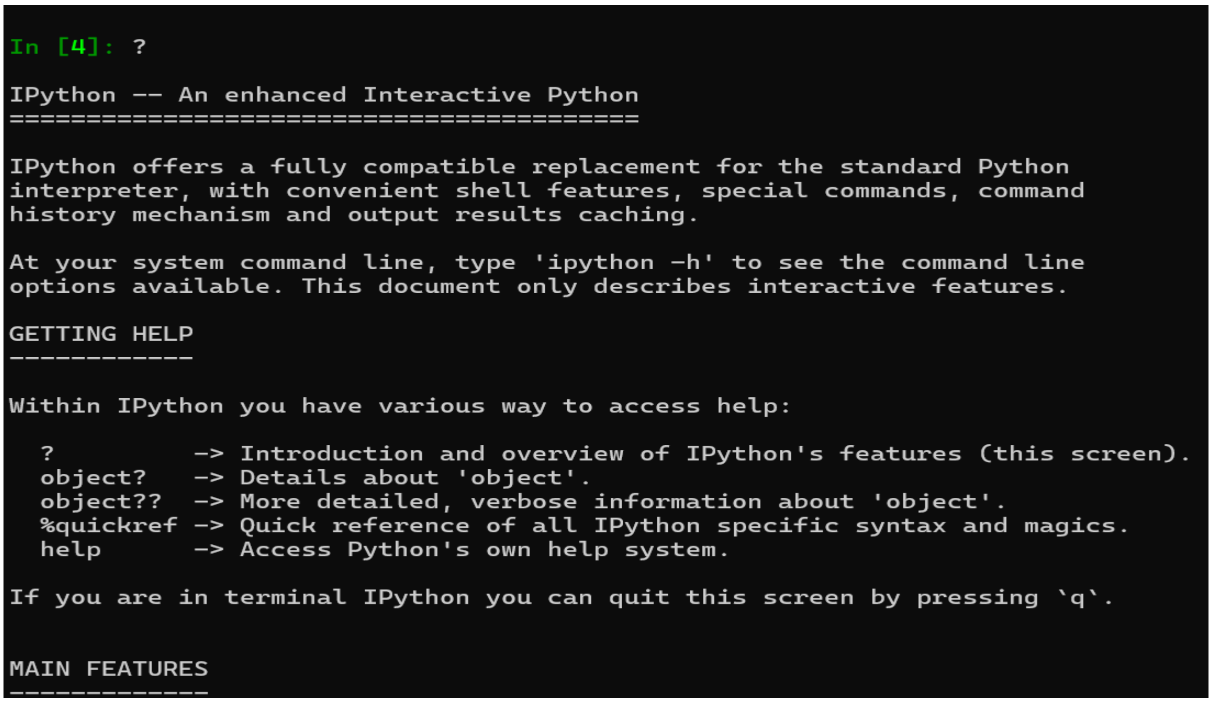
symbols: Shows the list of symbols that have special meaning in Python, such as * for multiplication and << for a left shift.

topics: Displays a list of general Python topics, such as CONVERSIONS. The topics appear in uppercase rather than lowercase

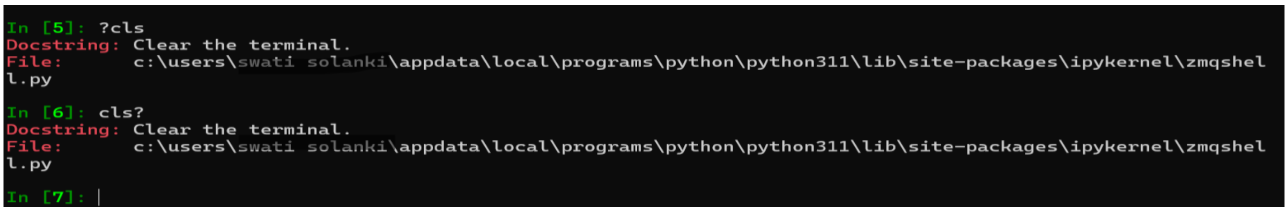
1.2.1.3. Getting IPython help

Getting help with IPython is different from getting help with Python. When you obtain IPython help, you work with the development environment rather than the programming language. To obtain IPython help, type ? and press Enter. You see a long listing of the various ways in which you can use IPython help.

Some of the more essential forms of help rely on typing a keyword with a question mark. For example, if you want to learn more about the cls command, you type cls? or ?cls and press Enter. It doesn't matter whether the question mark appears before or after the command.



```
In [4]: ?

IPython -- An enhanced Interactive Python
=========================================

IPython offers a fully compatible replacement for the standard Python
interpreter, with convenient shell features, special commands, command
history mechanism and output results caching.

At your system command line, type 'ipython -h' to see the command line
options available. This document only describes interactive features.

GETTING HELP
------------

Within IPython you have various way to access help:

  ?         -> Introduction and overview of IPython's features (this screen).
  object?   -> Details about 'object'.
  object??  -> More detailed, verbose information about 'object'.
  %quickref -> Quick reference of all IPython specific syntax and magics.
  help      -> Access Python's own help system.

If you are in terminal IPython you can quit this screen by pressing `q`.


MAIN FEATURES
-------------
```



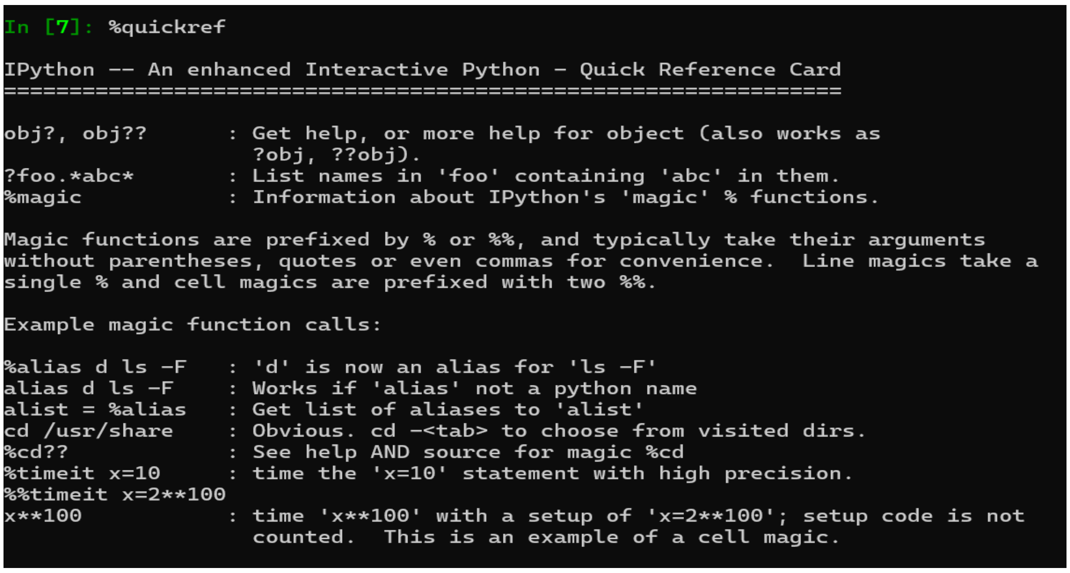
```
In [5]: ?cls
Docstring: Clear the terminal.
File:      c:\users\swati solanki\appdata\local\programs\python\python311\lib\site-packages\ipykernel\zmqshel
l.py

In [6]: cls?
Docstring: Clear the terminal.
File:      c:\users\swati solanki\appdata\local\programs\python\python311\lib\site-packages\ipykernel\zmqshel
l.py

In [7]: |
```

1.2.1.4. Using magic functions

Amazingly, you really can get magic on your computer! Jupyter provides a special feature called magic functions. The functions let you perform all sorts of amazing tasks with your Jupyter console.

Obtaining the magic functions list

The best way to start working with magic functions is to obtain a list of them by typing **%quickref** and pressing Enter.

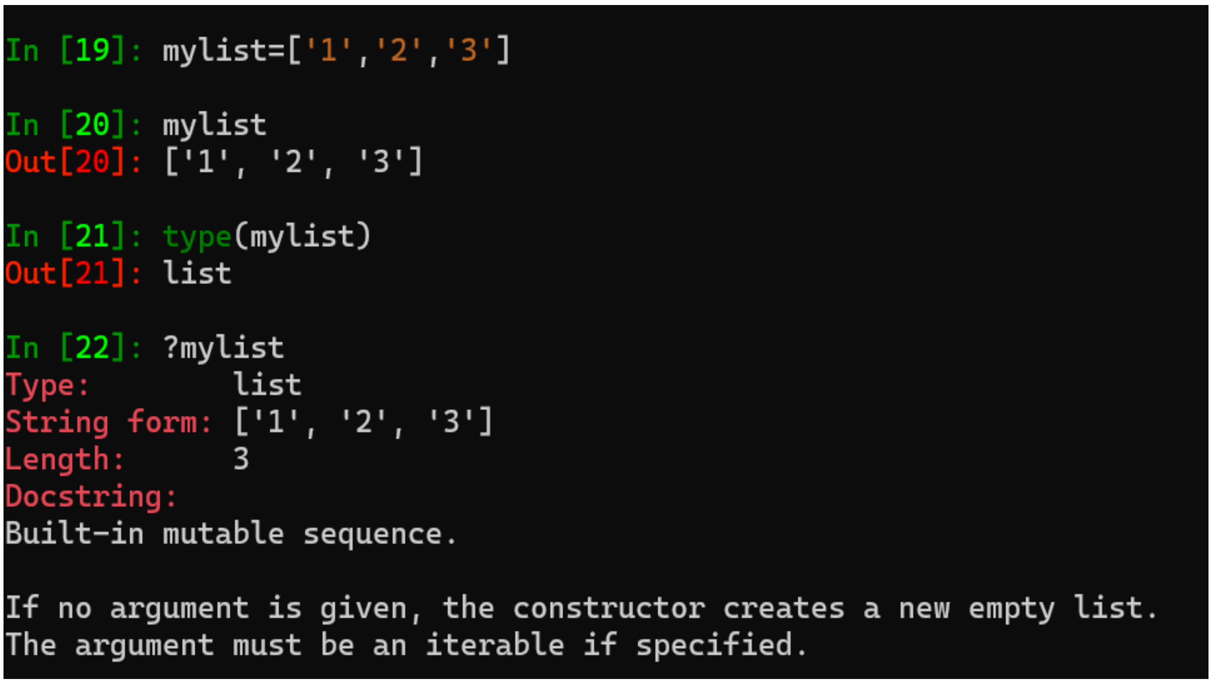


```
In [7]: %quickref

IPython -- An enhanced Interactive Python - Quick Reference Card
================================================================

obj?, obj??      : Get help, or more help for object (also works as
                   ?obj, ??obj).
?foo.*abc*       : List names in 'foo' containing 'abc' in them.
%magic           : Information about IPython's 'magic' % functions.

Magic functions are prefixed by % or %%, and typically take their arguments
without parentheses, quotes or even commas for convenience.  Line magics take a
single % and cell magics are prefixed with two %%.

Example magic function calls:

%alias d ls -F   : 'd' is now an alias for 'ls -F'
alias d ls -F    : Works if 'alias' not a python name
alist = %alias   : Get list of aliases to 'alist'
cd /usr/share    : Obvious. cd -<tab> to choose from visited dirs.
%cd??            : See help AND source for magic %cd
%timeit x=10     : time the 'x=10' statement with high precision.
%%timeit x=2**100
x**100           : time 'x**100' with a setup of 'x=2**100'; setup code is not
                   counted.  This is an example of a cell magic.
```

1.2.1.5. Discovering objects

With IPython, you can request information about specific objects using the object name and a question mark (?).

For example, if you want to know more about a list object named mylist, simply type mylist?



```
In [19]: mylist=['1','2','3']

In [20]: mylist
Out[20]: ['1', '2', '3']

In [21]: type(mylist)
Out[21]: list

In [22]: ?mylist
Type:        list
String form: ['1', '2', '3']
Length:      3
Docstring:
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.
```

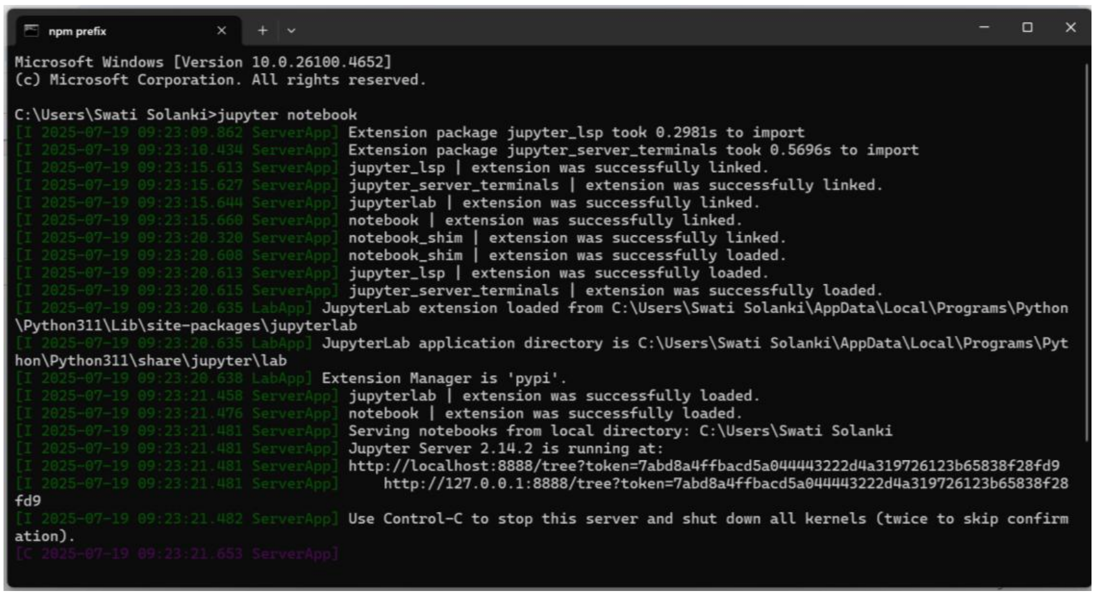
1.2.2. Using Jupyter Notebook

Installing Jupyter Notebook

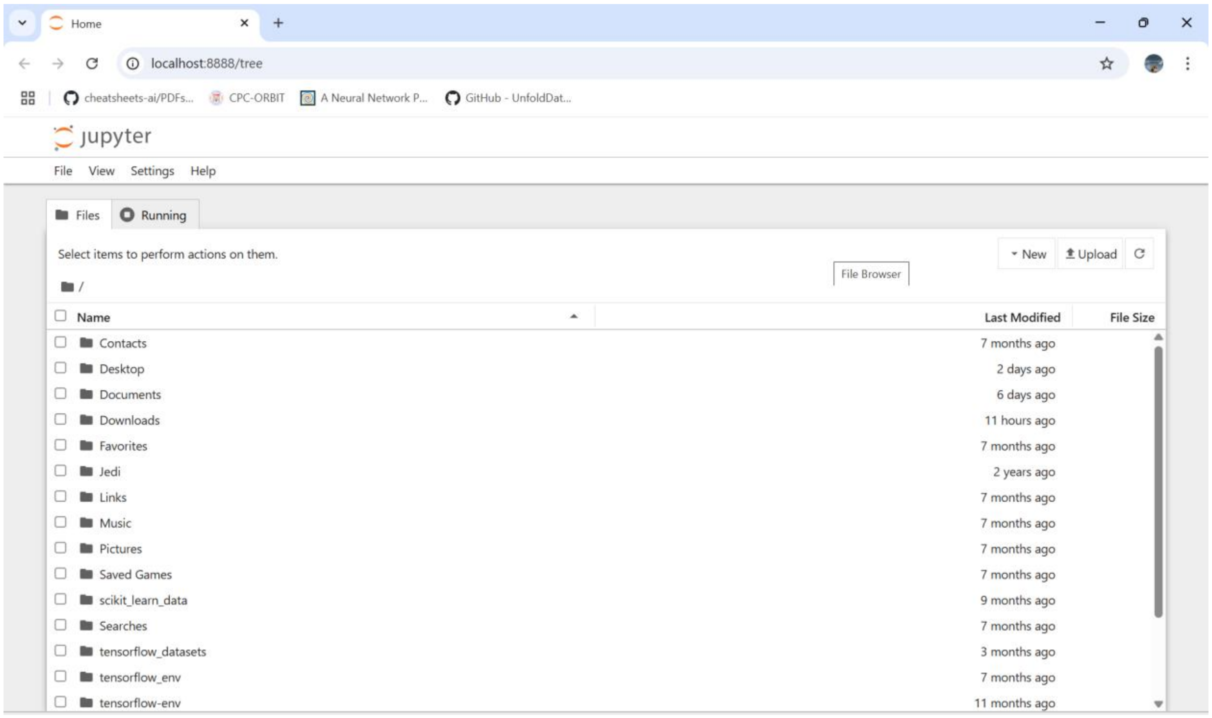
Pip install jupyter

Starting with Jupyter notebook

Open Command Prompt and type> jupyter notebook



```
Microsoft Windows [Version 10.0.26100.4652]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Swati Solanki>jupyter notebook
[I 2025-07-19 09:23:09.862 ServerApp] Extension package jupyter_lsp took 0.2981s to import
[I 2025-07-19 09:23:10.434 ServerApp] Extension package jupyter_server_terminals took 0.5696s to import
[I 2025-07-19 09:23:15.613 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2025-07-19 09:23:15.627 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2025-07-19 09:23:15.644 ServerApp] jupyterlab | extension was successfully linked.
[I 2025-07-19 09:23:15.660 ServerApp] notebook | extension was successfully linked.
[I 2025-07-19 09:23:20.320 ServerApp] notebook_shim | extension was successfully linked.
[I 2025-07-19 09:23:20.608 ServerApp] notebook_shim | extension was successfully loaded.
[I 2025-07-19 09:23:20.613 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2025-07-19 09:23:20.615 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2025-07-19 09:23:20.635 LabApp] JupyterLab extension loaded from C:\Users\Swati Solanki\AppData\Local\Programs\Python
\Python311\Lib\site-packages\jupyterlab
[I 2025-07-19 09:23:20.635 LabApp] JupyterLab application directory is C:\Users\Swati Solanki\AppData\Local\Programs\Pyt
hon\Python311\share\jupyter\lab
[I 2025-07-19 09:23:20.638 LabApp] Extension Manager is 'pypi'.
[I 2025-07-19 09:23:21.458 ServerApp] jupyterlab | extension was successfully loaded.
[I 2025-07-19 09:23:21.476 ServerApp] notebook | extension was successfully loaded.
[I 2025-07-19 09:23:21.481 ServerApp] Serving notebooks from local directory: C:\Users\Swati Solanki
[I 2025-07-19 09:23:21.481 ServerApp] Jupyter Server 2.14.2 is running at:
[I 2025-07-19 09:23:21.481 ServerApp] http://localhost:8888/tree?token=7abd8a4ffbacd5a044443222d4a319726123b65838f28fd9
[I 2025-07-19 09:23:21.481 ServerApp]     http://127.0.0.1:8888/tree?token=7abd8a4ffbacd5a044443222d4a319726123b65838f28
fd9
[I 2025-07-19 09:23:21.482 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirm
ation).
[C 2025-07-19 09:23:21.653 ServerApp]
```



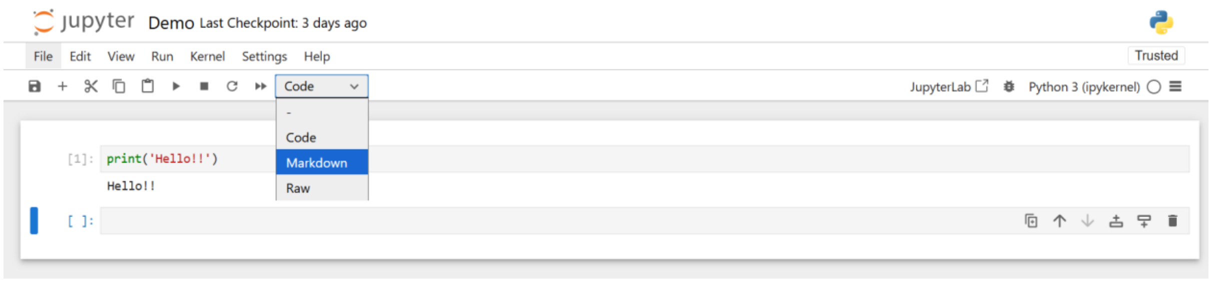
Select items to perform actions on them.

/

```
Name                      Last Modified   File Size
Contacts                  7 months ago
Desktop                   2 days ago
Documents                 6 days ago
Downloads                 11 hours ago
Favorites                 7 months ago
Jedi                      2 years ago
Links                     7 months ago
Music                     7 months ago
Pictures                  7 months ago
Saved Games               7 months ago
scikit_learn_data         9 months ago
Searches                  7 months ago
tensorflow_datasets       3 months ago
tensorflow_env            7 months ago
tensorflow-env            11 months ago
```

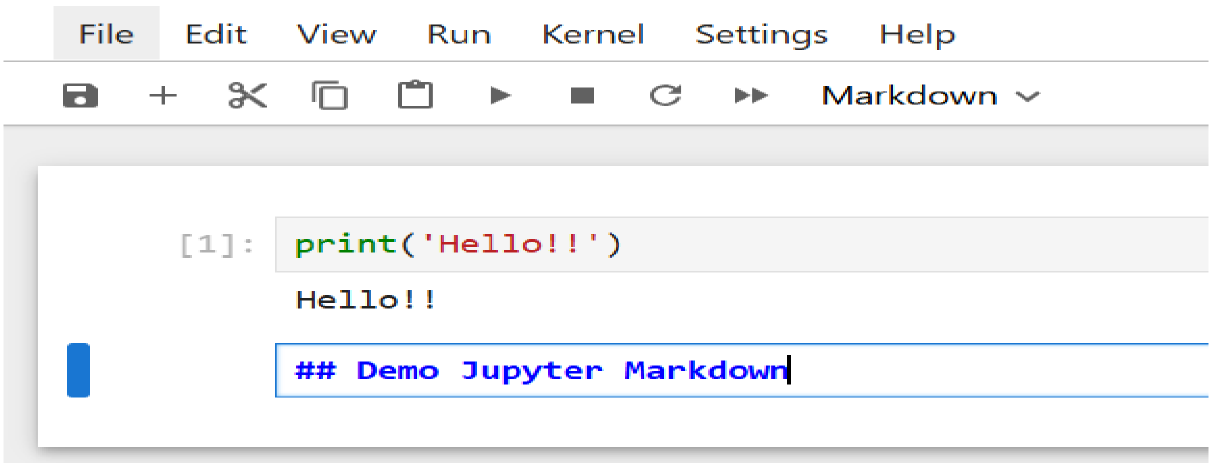
1.2.2.1. Working with styles

Here's one of the ways in which Notebook excels over just about any other IDE that you'll ever use: It helps you to create nice-looking output. Rather than have a screen full of a whole bunch of plain-old code, you can use Notebook to create sections and add styles so that the output is nicely formatted.



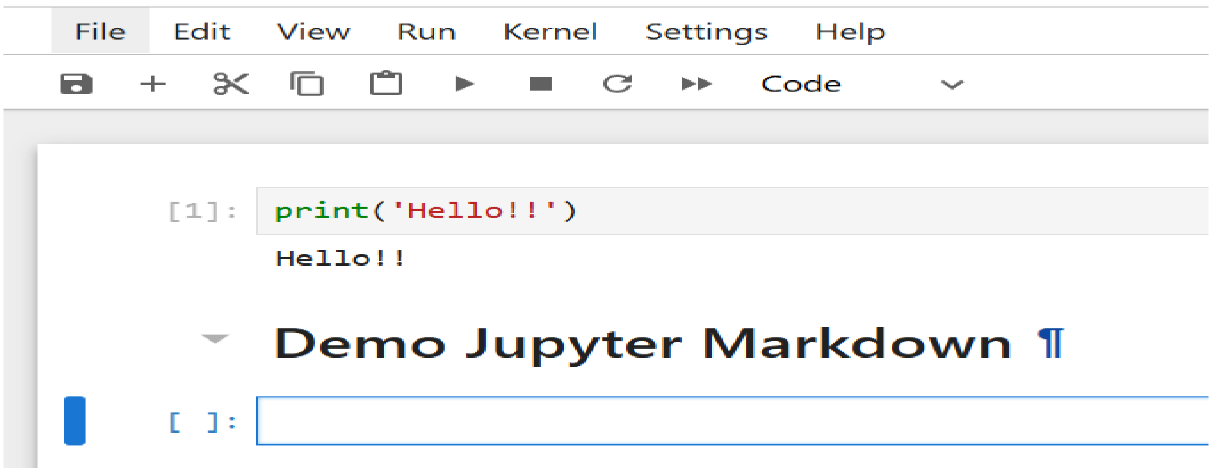
jupyter Demo Last Checkpoint: 3 days ago
File Edit View Run Kernel Settings Help
Trusted
Code
-
Code
Markdown
Raw
JupyterLab
Python 3 (ipykernel)

```
[1]: print('Hello!!')
     Hello!!
[ ]:
```



```
[1]: print('Hello!!')
Hello!!
```

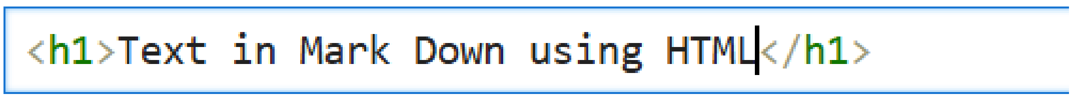
```
## Demo Jupyter Markdown
```



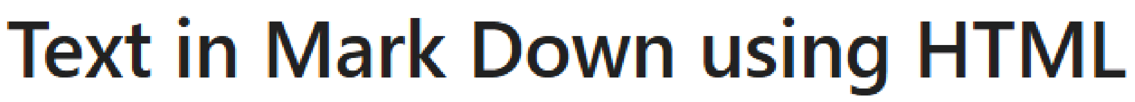
```
[1]: print('Hello!!')
Hello!!
```

Demo Jupyter Markdown ¶

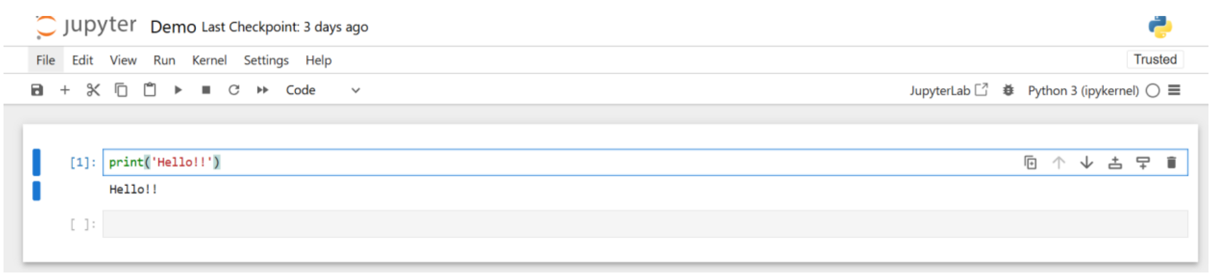
```
[ ]:
```



```
<h1>Text in Mark Down using HTML</h1>
```



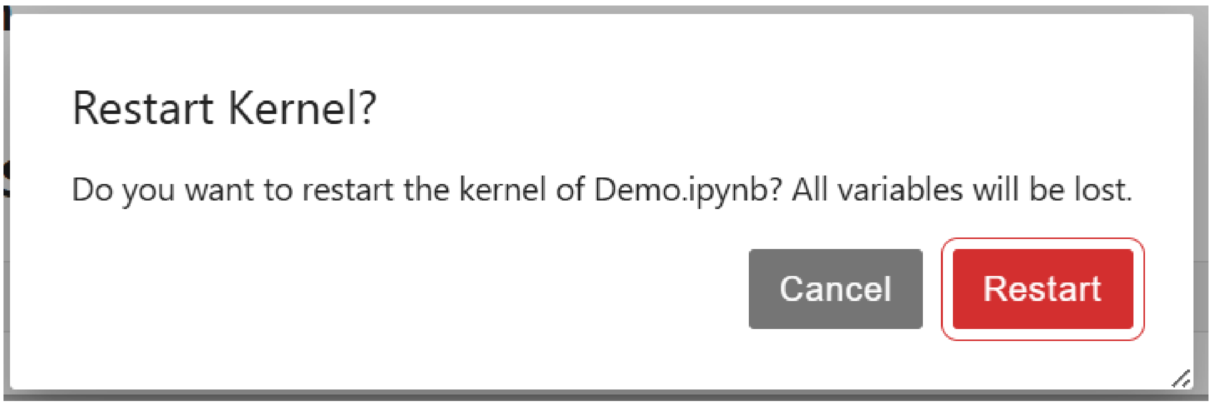
Text in Mark Down using HTML



```
[1]: print('Hello!!')
Hello!!
[ ]:
```

1.2.2.2. Restarting the kernel

Every time you perform a task in your notebook, you create variables, import modules, and perform a wealth of other tasks that corrupt the environment. At some point, you can't really be sure that something is working as it should. To overcome this problem, you click Restart Kernel (the button with an open circle with an arrow at one end) after saving your document by clicking Save and Checkpoint (the button containing a floppy disk symbol). You can then run your code again to ensure that it does work as you thought it would.



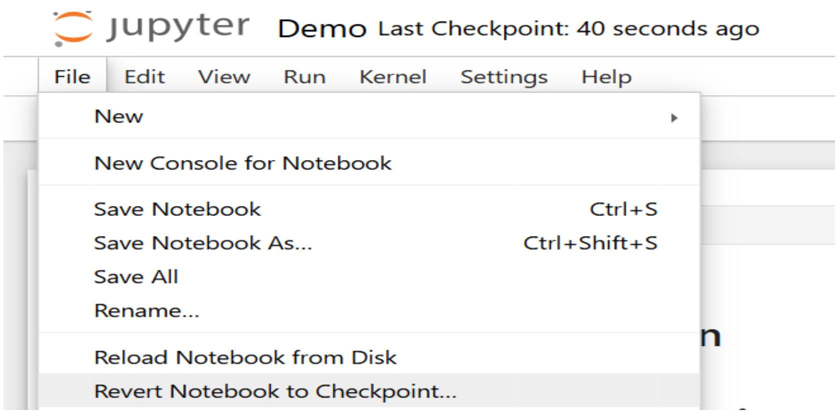
Restart Kernel?

Do you want to restart the kernel of Demo.ipynb? All variables will be lost.

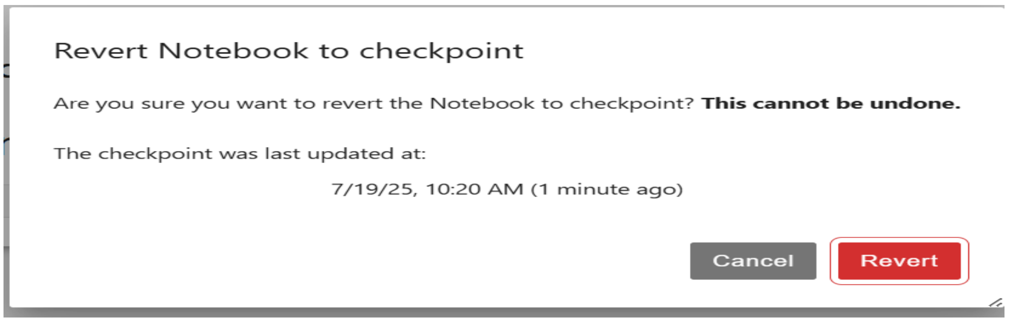
1.2.2.3. Restoring a checkpoint

At some point, you may find that you made a mistake. Notebook is notably missing an Undo button: You won't find one anywhere. Instead, you create checkpoints each time you finish a task. Creating checkpoints when your document is stable and working properly helps you recover faster from mistakes.

To restore your setup to the condition contained in a checkpoint, choose File Revert to Checkpoint. You see a listing of available checkpoints.



jupyter Demo Last Checkpoint: 40 seconds ago
File
Edit
View
Run
Kernel
Settings
Help
New
New Console for Notebook
Save Notebook Ctrl+S
Save Notebook As... Ctrl+Shift+S
Save All
Rename...
Reload Notebook from Disk
Revert Notebook to Checkpoint...



Revert Notebook to checkpoint

Are you sure you want to revert the Notebook to checkpoint? **This cannot be undone.**

The checkpoint was last updated at:

7/19/25, 10:20 AM (1 minute ago)

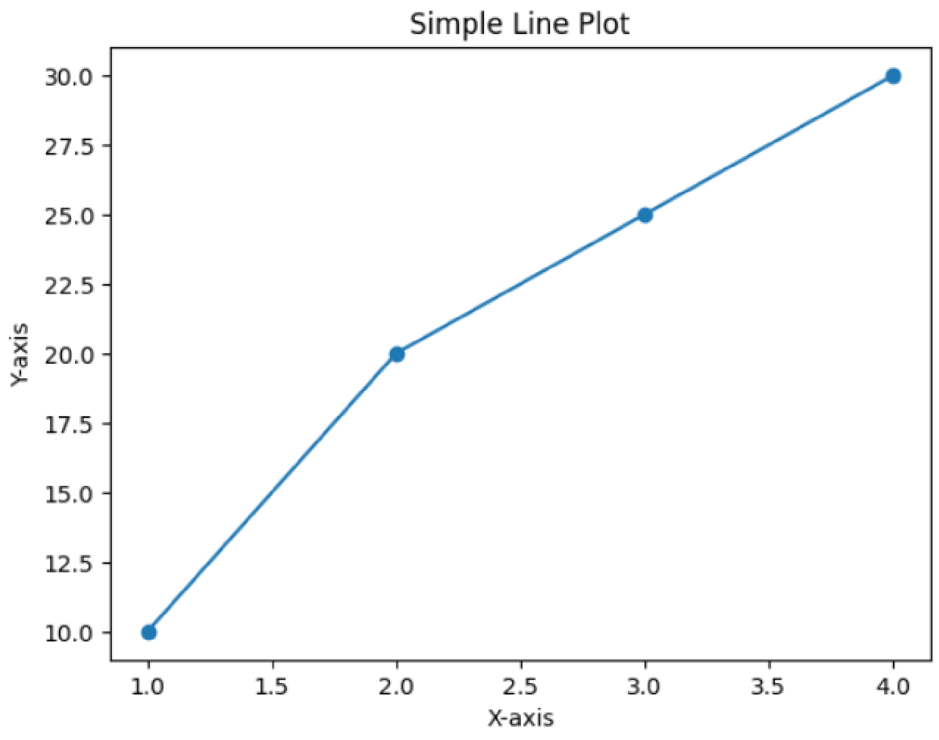
1.2.2.4. Performing Multimedia and Graphic Integration

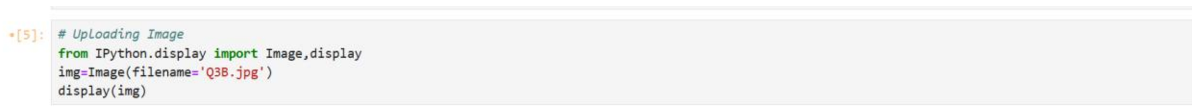
Pictures say a lot of things that words can't say (or at least they do it with far less effort). Notebook is both a coding platform and a presentation platform. You may be surprised at just what you can do with it.

1.2.2.4.1. Embedding plots and other images

At some point, you might have spotted a notebook with multimedia or graphics embedded into it and wondered why you didn't see the same effects in your own files. In fact, all the graphics examples in the book appear as part of the code. Fortunately, you can perform some more magic by using the %matplotlib magic function. The possible values for this function are: 'gtk', 'gtk3', 'inline', 'nbagg', 'osx', 'qt', 'qt4', 'qt5', 'tk', and 'wx', each of which defines a different plotting backend (the code used to actually render the plot) used to present information onscreen.

```
[28]: import matplotlib.pyplot as plt

      # Create a basic plot
      plt.plot([1, 2, 3, 4], [10, 20, 25, 30], marker='o')
      plt.title('Simple Line Plot')
      plt.xlabel('X-axis')
      plt.ylabel('Y-axis')

      # Show the plot
      plt.show()
```



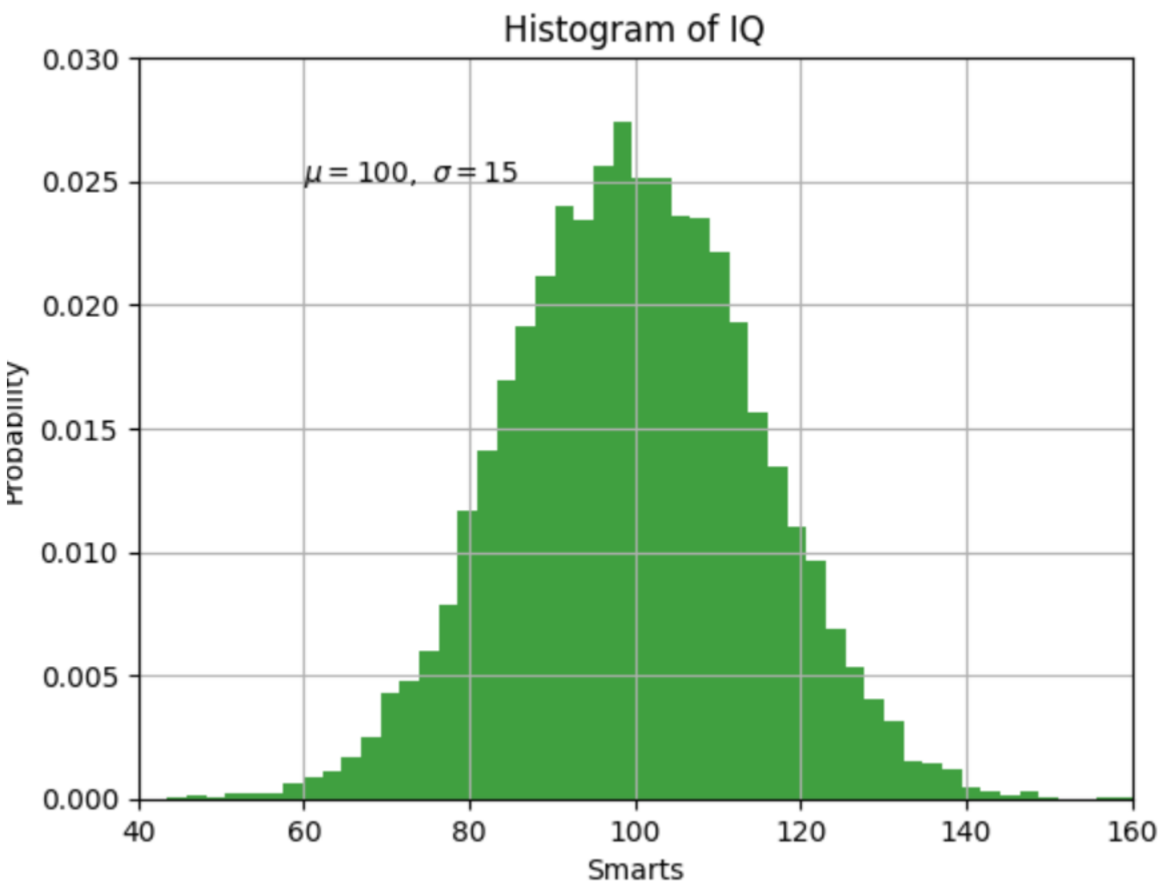


```
[5]: # Uploading Image
from IPython.display import Image,display
img=Image(filename='Q3B.jpg')
display(img)
```

1.2.2.4.2. Loading examples from online sites

Because some examples you see online can be hard to understand unless you have them loaded on your own system, you should also keep the %load magic function in mind. All you need is the URL of an example you want to see on your system. For example, try %load https://matplotlib.org/_downloads/pyplot_text.py. When you click Run Cell, Notebook loads the example directly in the cell and comments the %load call out. You can then run the example and see the output from it on your own system.

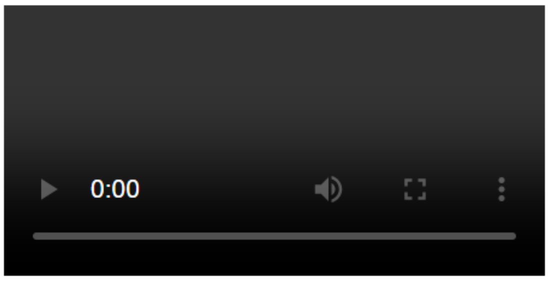
%load https://matplotlib.org/3.0.0/_downloads/pyplot_text.py



1.2.2.4.3. Obtaining online graphics and multimedia

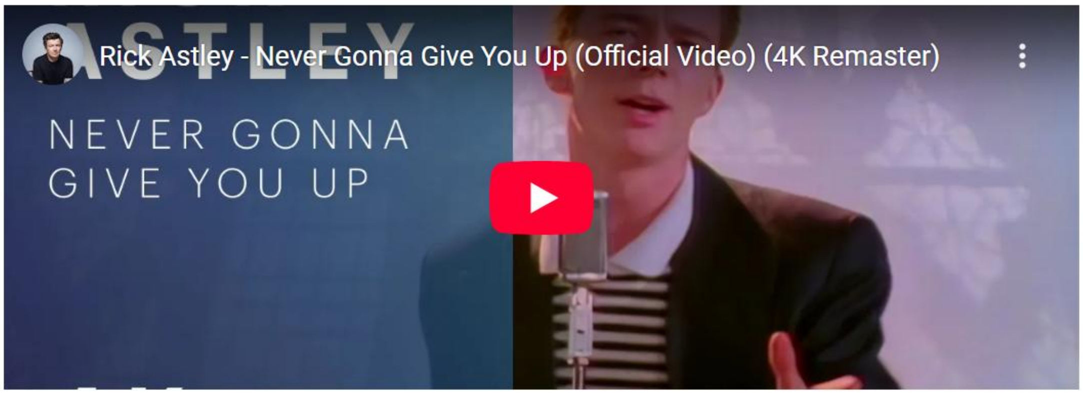
A lot of the functionality required to perform special multimedia and graphics processing appears within IPython.display. By importing a required class, you can perform tasks such as embedding images into your notebook.

```
[1]: from IPython.display import Video
     video=Video('https://samplelib.com/lib/preview/mp4/sample-5s.mp4',embed=False)
     display(video)
```



```
[2]: from IPython.display import YouTubeVideo
     YouTubeVideo("dQw4w9WgXcQ",width=700,height=250)
```

[2]:


Rick Astley - Never Gonna Give You Up (Official Video) (4K Remaster)
ASTLEY
NEVER GONNA
GIVE YOU UP

```
[3]: from IPython.display import Audio
     Audio('file_example_MP3_700KB.mp3')
```

[3]: