TOPICS Jupyter笔记本电脑支持 ▼

# 在Visual Studio Code中使用Jupyter Notebook

(https://github.com/Microsoft/vscode-docs/blob/master/docs/python/jupyter-support.md)

Jupyter (http://jupyter-notebook.readthedocs.io/en/latest/)（以前称为IPython Notebook）是一个开源项目，可让您轻松地在一个名为**Notebook的**画布上组合Markdown文本和可执行的Python源代码。Visual Studio Code支持本地使用Jupyter Notebook，以及通过Python代码文件 (/docs/python/jupyter-support-py)。本主题介绍了Jupyter Notebook可用的本机支持，并演示了如何：

- 创建，打开和保存Jupyter Notebook
- 使用Jupyter代码单元
- 使用变量浏览器和数据查看器查看，检查和过滤变量
- 连接到远程Jupyter服务器
- 调试Jupyter笔记本

## 设置环境

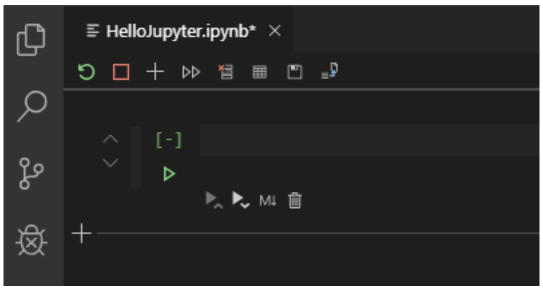
要使用Jupyter笔记本，您必须在VS Code中激活Anaconda环境，或在其中安装Jupyter软件包的 (https://pypi.org/project/jupyter/)另一个Python环境中激活 (https://pypi.org/project/jupyter/)。要选择环境，请使用**Python：**从命令面板中**选择"解释器"**命令（ Ctrl + Shift + P ）。

激活适当的环境后，您可以创建并打开Jupyter Notebook，连接到用于运行代码单元的远程Jupyter服务器，并将Jupyter Notebook导出为Python文件。

> **注意：**默认情况下，Visual Studio Code Python扩展将在笔记本编辑器中打开Jupyter笔记本（.ipynb）。如果要禁用此行为，可以在设置中将其关闭。（Python>数据科学：使用笔记本编辑器）。

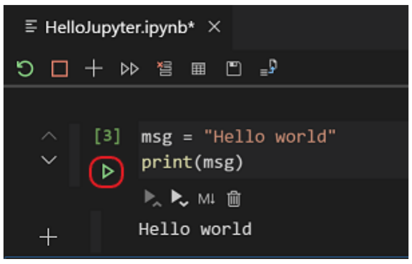
## 创建或打开Jupyter笔记本

您可以通过运行**Python：**从命令面板（ Ctrl + Shift + P ）**创建空白的新Jupyter Notebook**命令或在工作区中创建新的.ipynb文件来创建Jupyter Notebook。选择文件后，将启动笔记本编辑器，使您可以编辑和运行代码单元。



如果您有现有的Jupyter笔记本，则可以在笔记本编辑器中通过双击该文件并使用Visual Studio Code，通过Visual Studio Code或使用"命令面板**Python：在笔记本编辑器中打开** "命令来打开它来打开它。

创建笔记本后，您可以使用单元格旁边的绿色运行图标运行代码单元，输出将直接显示在该代码单元下方。

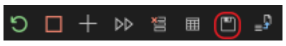


```
msg = "Hello world"
print(msg)
```

```
Hello world
```

## 保存您的Jupyter笔记本

您可以使用键盘组合键 Ctrl + S 或"笔记本编辑器"工具栏上的"保存"图标来保存Jupyter Notebook 。



> **注意：**目前，您必须使用上述方法来保存笔记本。" **文件**" >" **保存**"菜单不会仅保存工具栏图标或键盘命令来保存笔记本。

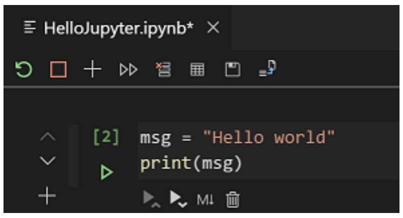
## 在笔记本编辑器中使用代码单元

笔记本编辑器使您可以在Jupyter笔记本中轻松创建，编辑和运行代码单元。

### 创建一个代码单元

默认情况下，空白的Notebook将有一个空的代码单元供您开始使用，而现有的Notebook将在底部放置一个。将您的代码添加到空代码单元以开始。

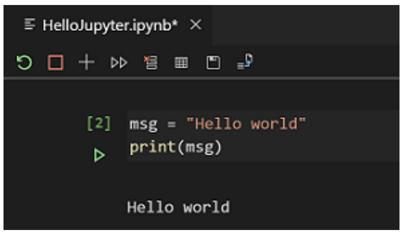
```
msg = "Hello world"
print(msg)
```



```
[2] msg = "Hello world"
    print(msg)
```

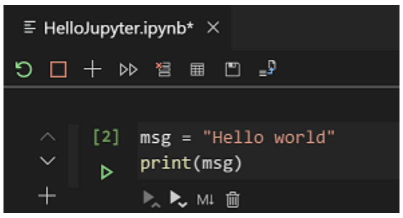
## 代码单元模式 #

使用代码单元时，单元可以处于三种状态：未选择，命令模式和编辑模式。单元格的当前状态由代码单元格左侧的竖线表示。如果看不到任何条形，则表示该单元格未被选中。



```
[2] msg = "Hello world"
    print(msg)

Hello world
```

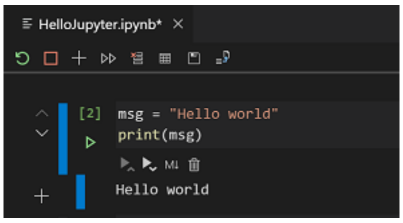
未选择的单元格不可编辑，但是您可以将鼠标悬停在其上方以显示其他特定于单元格的工具栏选项。这些其他工具栏选项直接显示在单元格的下方和左侧。当将鼠标悬停在一个单元格上时，您还将看到左侧有一个空的竖线。



```
[2] msg = "Hello world"
    print(msg)
```

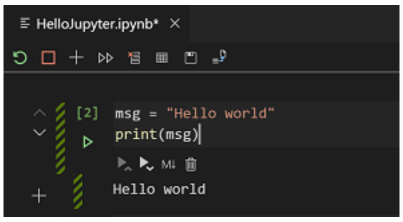
选择一个单元格时，它可以处于两种不同的模式。它可以处于命令模式或编辑模式。当单元处于命令模式时，可以对其进行操作并接受键盘命令。当单元处于编辑模式时，可以修改单元的内容（代码或降价）。

当单元处于命令模式时，单元左侧的竖线将一直显示以指示其已被选中。



```
[2] msg = "Hello world"
    print(msg)

Hello world
```

在编辑模式下，竖线将带有对角线。

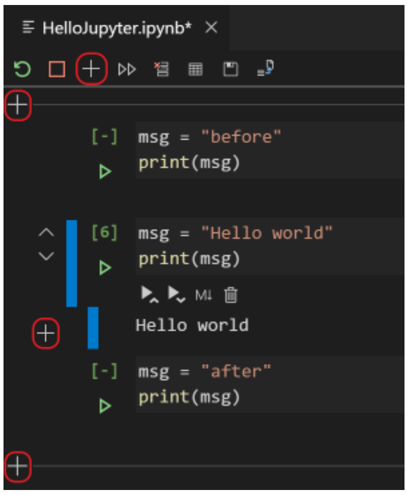


```
[2] msg = "Hello world"
    print(msg)

Hello world
```

要从编辑模式切换到命令模式，请按 ESC 键。要从命令模式切换到编辑模式，请按 Enter 键。您也可以使用鼠标来更改模式，方法是在代码单元格中的代码/降价区域中单击或移出。

## 添加其他代码单元

可以使用主工具栏，代码单元的垂直工具栏，笔记本底部的"添加代码单元"图标，笔记本顶部的"添加代码单元"图标（通过鼠标悬停），将代码单元添加到笔记本中。键盘命令。



```
[-] msg = "before"
    print(msg)

[6] msg = "Hello world"
    print(msg)

Hello world

[-] msg = "after"
    print(msg)
```

使用主工具栏中的加号图标将在当前所选单元格的正下方添加一个新单元格。使用Jupyter Notebook顶部和底部的添加单元格图标，将分别在顶部和底部添加一个代码单元格。使用代码单元格工具栏中的添加图标，将在其下方直接添加一个新的代码单元格。

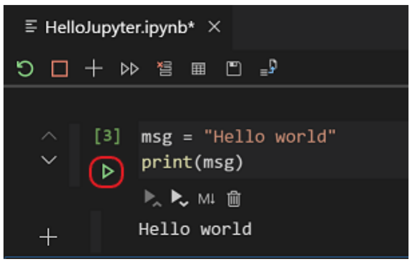
当代码单元格处于命令模式时， `A` 键可用于在所选单元格上方添加一个单元格，而 `B` 键可用于在所选单元格下方添加一个单元格。

### 选择一个代码单元

可以使用鼠标，键盘上的向上/向下箭头键以及 `J` （向下）和 `K` （向上）键更改所选的代码单元。要使用键盘，单元必须处于命令模式。

### 运行单个代码单元

Once your code is added, you can run a cell using the green run arrow and the output will be displayed below the code cell.

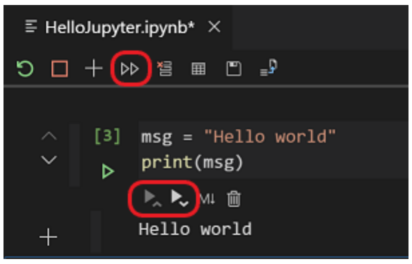


```
[3] msg = "Hello world"
    print(msg)

Hello world
```

You can also use key combos to run a selected code cell. `Ctrl+Enter` runs the currently selected cell, `Shift+Enter` runs the currently selected cell and inserts a new cell immediately below (focus moves to new cell), and `Alt+Enter` runs the currently selected cell and inserts a new cell immediately below (focus remains on current cell). These keyboard combos can be used in both command and edit modes.

### Run multiple code cells

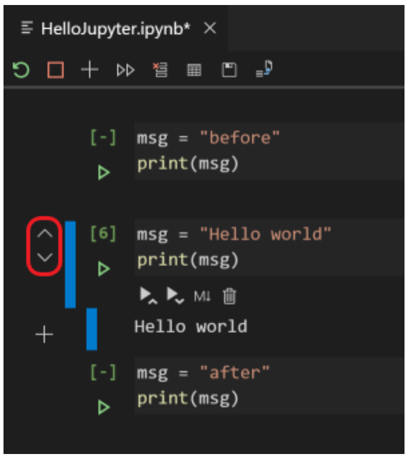
Running multiple code cells can be accomplished in a number of ways. You can use the double arrow in the toolbar of the Notebook Editor to run all cells within the Notebook or the hover toolbar arrows to run all cells above or below the current code cell.



```
[3] msg = "Hello world"
    print(msg)

Hello world
```

### Move a code cell

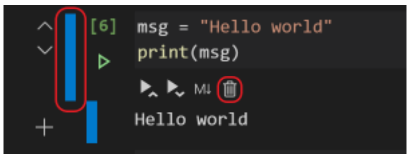
Moving code cells up or down within a Notebook can be accomplished using the vertical arrows beside each code cell. Hover over the code cell and then click the up arrow to move the cell up and the down arrow to move the cell down.



```
[-] msg = "before"
    print(msg)

[6] msg = "Hello world"
    print(msg)

Hello world

[-] msg = "after"
    print(msg)
```

Delete a code cell

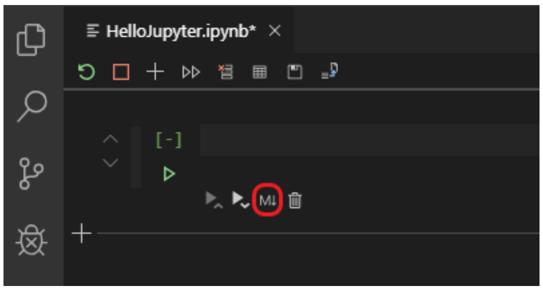
Deleting a code cell can be accomplished by hovering over a code cell and using the delete icon in the code cell toolbar or through the keyboard combo `dd` when the selected code cell is in command mode.



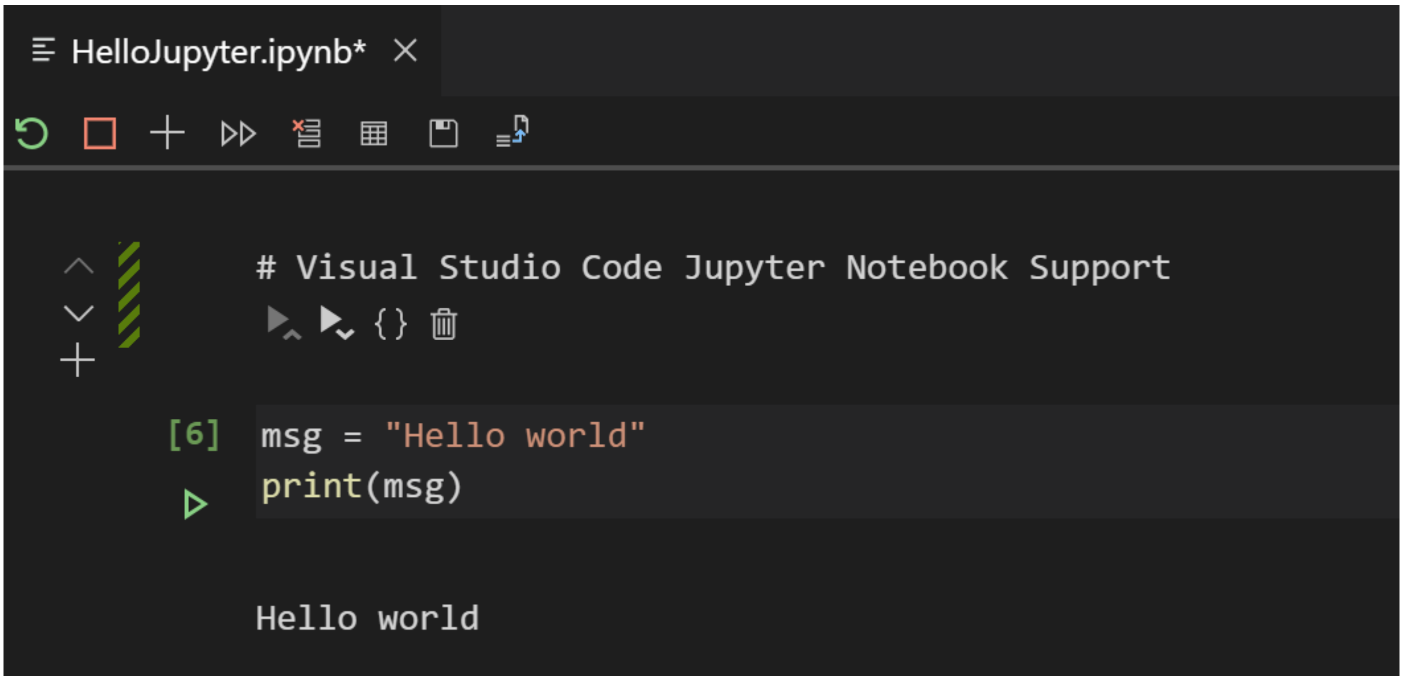
```
[6] msg = "Hello world"
    print(msg)

Hello world
```

Switch between code and Markdown

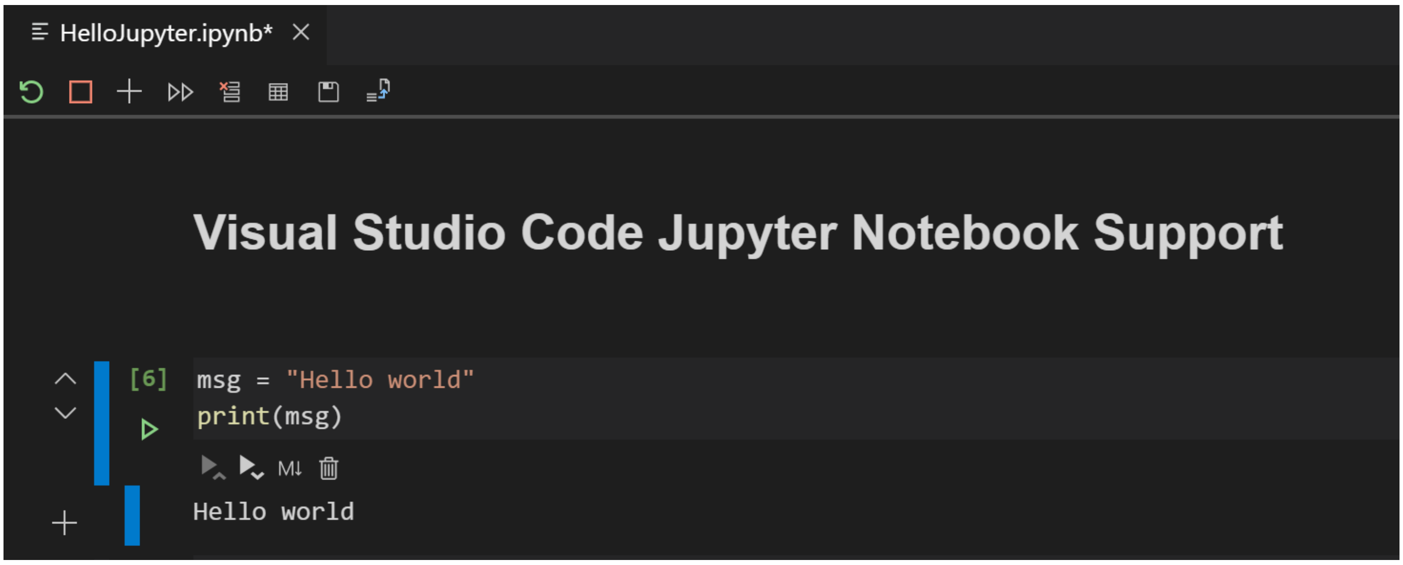
The Notebook Editor allows you to easily change code cells between Markdown and code. By default a code cell is set for code, but just click the Markdown icon (or the code icon, if Markdown was previously set) in the code cell's toolbar to change it.



Once Markdown is set, you can enter Markdown formatted content to the code cell. Once you select another cell or toggle out of the content selection, the Markdown content is rendered in the Notebook Editor.



```
# Visual Studio Code Jupyter Notebook Support

[6] msg = "Hello world"
    print(msg)

Hello world
```



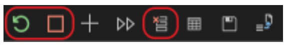
```
msg = "Hello world"
print(msg)
```

```
Hello world
```

You can also use the keyboard to change the cell type. When a cell is selected and in command mode, the `M` key switches the cell type to Markdown and the `Y` key switches the cell type to code.

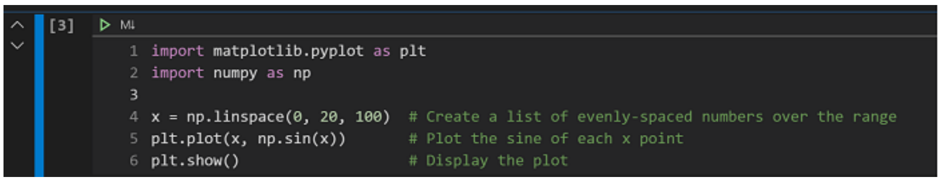
### Clear output or restart/interrupt the kernel

If you'd like to clear the code cell output or restart/interrupt the kernel, you can accomplish that using the main Notebook Editor toolbar.



### Enable/Disable line numbers

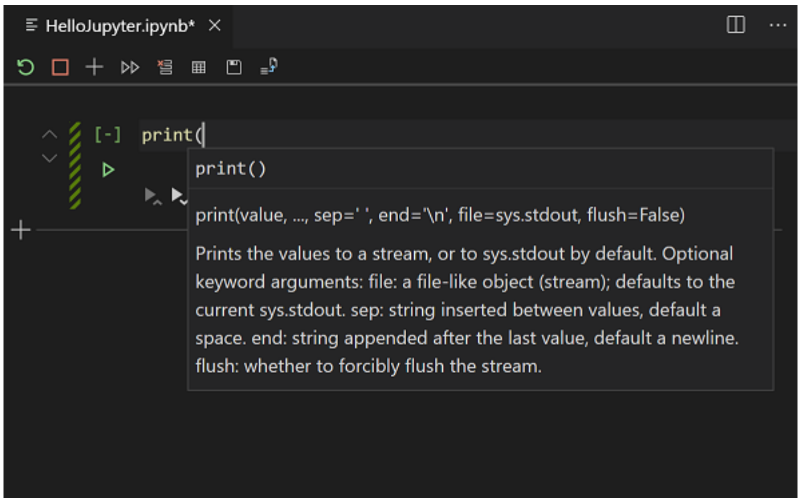
You can enable or disable line numbering within a code cell using the `L` key.



```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  x = np.linspace(0, 20, 100)  # Create a list of evenly-spaced numbers over the range
5  plt.plot(x, np.sin(x))       # Plot the sine of each x point
6  plt.show()                   # Display the plot
```

## IntelliSense support in the Jupyter Notebook Editor

The Python Jupyter Notebook Editor window has full IntelliSense – code completions, member lists, quick info for methods, and parameter hints. You can be just as productive typing in the Notebook Editor window as you are in the code editor.



```
print(
```

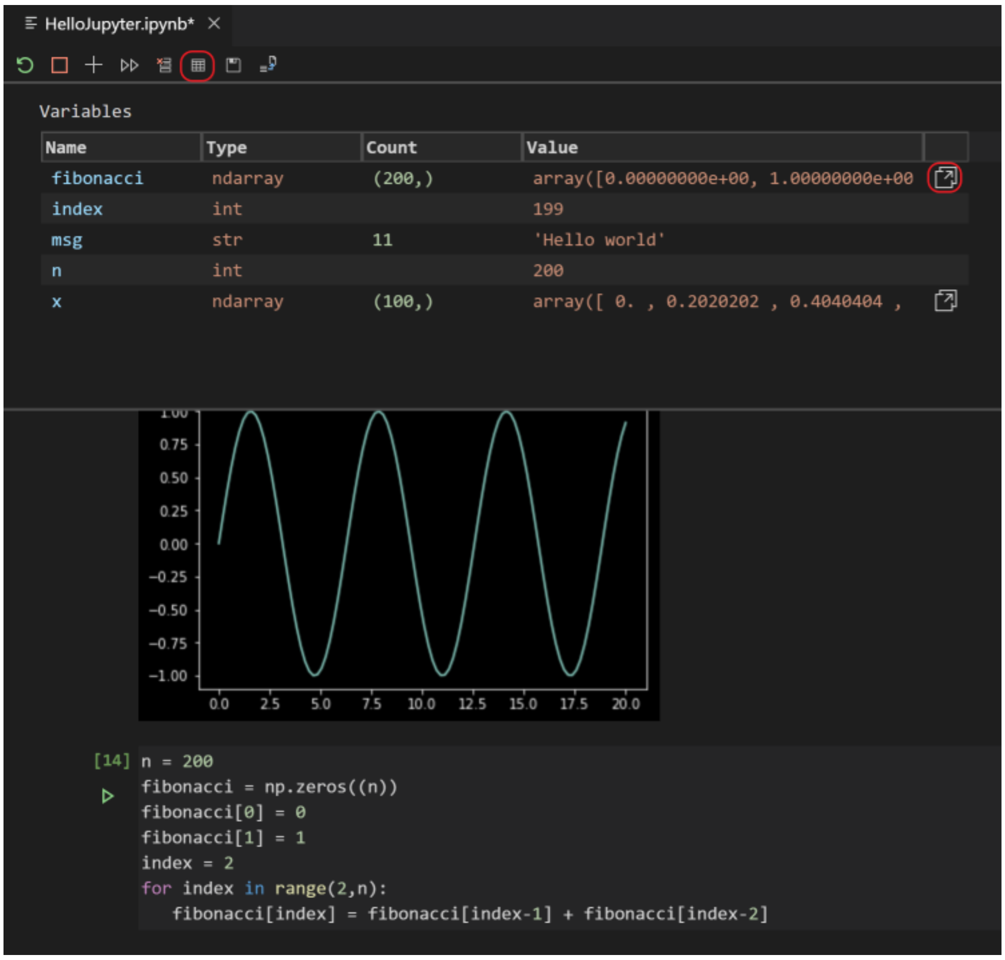
print()

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default. Optional keyword arguments: file: a file-like object (stream); defaults to the current sys.stdout. sep: string inserted between values, default a space. end: string appended after the last value, default a newline. flush: whether to forcibly flush the stream.

## Variable explorer and data viewer

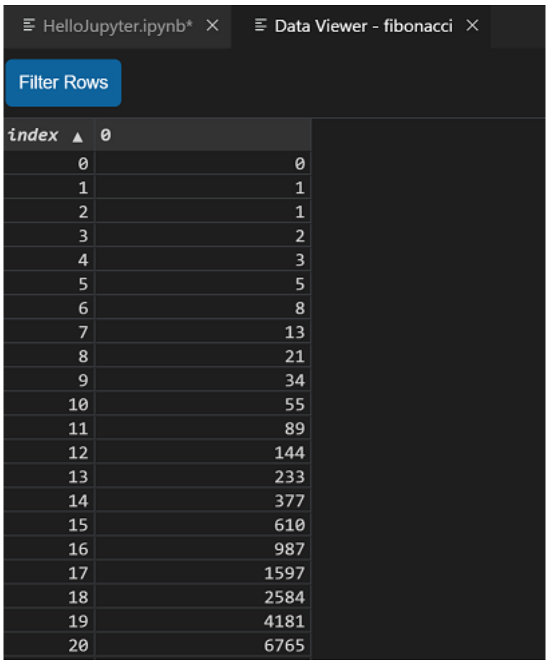
Within the Python Notebook Editor, it's possible to view, inspect, and filter the variables within your current Jupyter session. By clicking the **Variables** icon in the top toolbar after running code and cells, you'll see a list of the current variables, which will automatically update as variables are used in code.



| Name | Type | Count | Value |
|---|---|---|---|
| fibonacci | ndarray | (200,) | array([0.00000000e+00, 1.00000000e+00 |
| index | int | | 199 |
| msg | str | 11 | 'Hello world' |
| n | int | | 200 |
| x | ndarray | (100,) | array([ 0. , 0.2020202 , 0.4040404 , |

```
[14] n = 200
     fibonacci = np.zeros((n))
     fibonacci[0] = 0
     fibonacci[1] = 1
     index = 2
     for index in range(2,n):
         fibonacci[index] = fibonacci[index-1] + fibonacci[index-2]
```

For additional information about your variables, you can also double-click on a row or use the **Show variable in data viewer** button next to the variable to see a more detailed view of a variable in the Data Viewer. Once open, you can filter the values by searching over the rows.



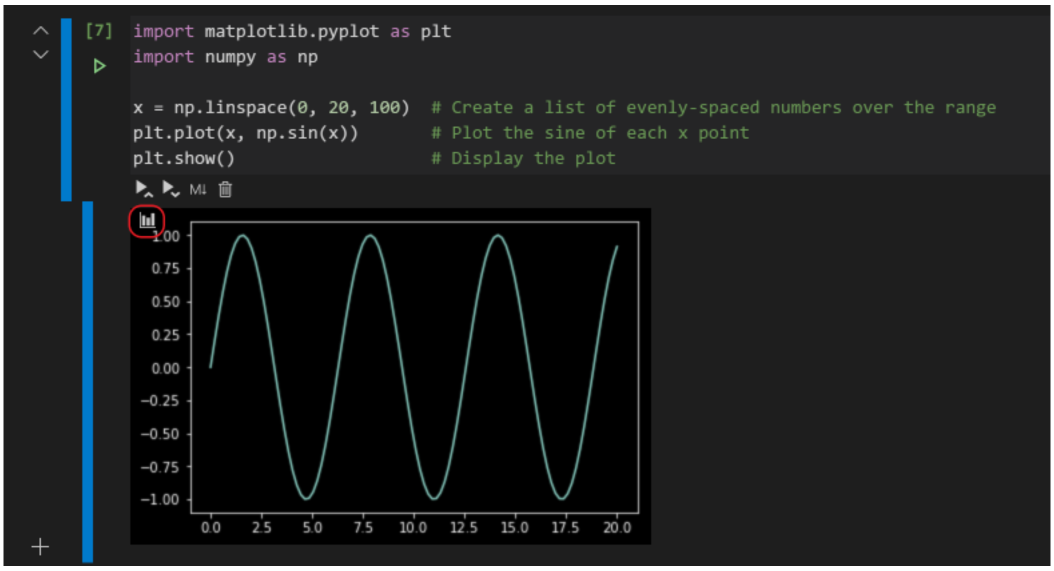
| index ▲ | 0 |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |
| 5 | 5 |
| 6 | 8 |
| 7 | 13 |
| 8 | 21 |
| 9 | 34 |
| 10 | 55 |
| 11 | 89 |
| 12 | 144 |
| 13 | 233 |
| 14 | 377 |
| 15 | 610 |
| 16 | 987 |
| 17 | 1597 |
| 18 | 2584 |
| 19 | 4181 |
| 20 | 6765 |

**Note:** Variable explorer is enabled by default, but can be turned off in settings (Python > Data Science: Show Jupyter Variable Explorer).

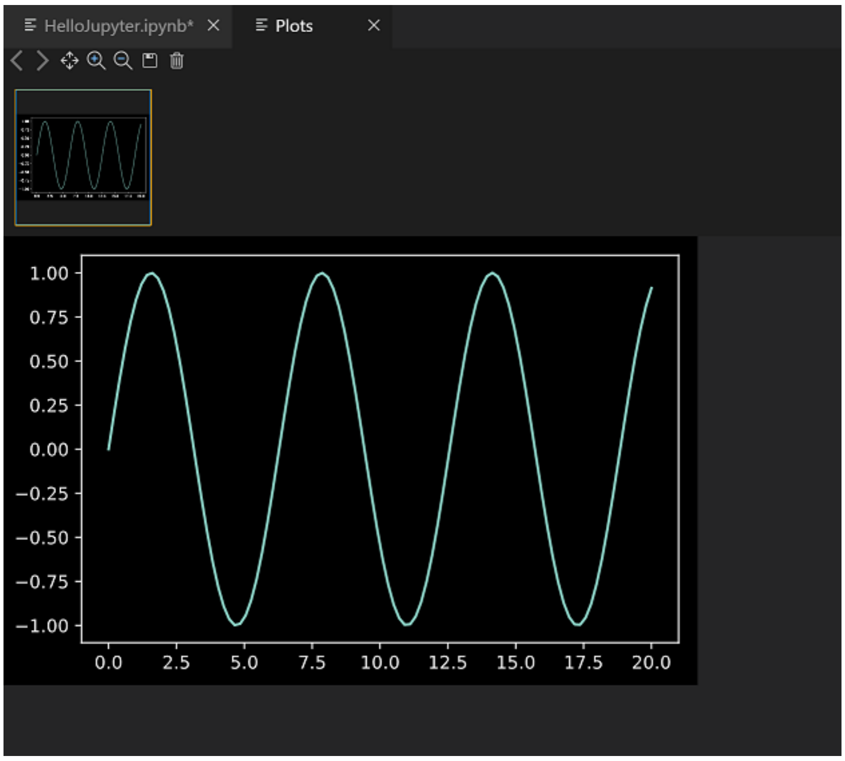
## Plot viewer

The Plot Viewer gives you the ability to work more deeply with your plots. In the viewer you can pan, zoom, and navigate plots in the current session. You can also export plots to PDF, SVG, and PNG formats.

Within the Notebook Editor window, double-click any plot to open it in the viewer, or select the plot viewer button on the upper left corner of the plot (visible on hover).



```
[7] import matplotlib.pyplot as plt
    import numpy as np

    x = np.linspace(0, 20, 100)  # Create a list of evenly-spaced numbers over the range
    plt.plot(x, np.sin(x))       # Plot the sine of each x point
    plt.show()                   # Display the plot
```

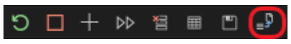


**Note:** There is support for rendering plots created with matplotlib (https://matplotlib.org/) and Altair (https://altair-viz.github.io/index.html).

## Debug a Jupyter Notebook

Currently, to debug a Jupyter Notebook you will need to first export it as a Python file. Once exported as a Python file, the Visual Studio Code debugger lets you step through your code, set breakpoints, examine state, and analyze problems. Using the debugger is a helpful way to find and correct issues in notebook code. To debug your Python file:

1. In VS Code, if you haven't already, activate a Python environment in which Jupyter is installed.
2. From your Jupyter Notebook (.ipynb) select the convert button in the main toolbar.

   Once exported, you'll have a .py file with your code that you can use for debugging.
3. After saving the .py file, to start the debugger, use one of the following options:
   - For the whole Notebook, open the Command Palette ( `Ctrl+Shift+P` ) and run the **Python: Debug Current File in Python Interactive Window** command.
   - For an individual cell, use the **Debug Cell** adornment that appears above the cell. The debugger specifically starts on the code in that cell. By default, **Debug Cell** just steps into user code. If you want to step into non-user code, you need to uncheck **Data Science: Debug Just My Code** in the Python extension settings ( `Ctrl+,` ).
4. To familiarize yourself with the general debugging features of VS Code, such as inspecting variables, setting breakpoints, and other activities, review VS Code debugging (/docs/editor/debugging).
5. As you find issues, stop the debugger, correct your code, save the file, and start the debugger again.
6. When you're satisfied that all your code is correct, use the Python Interactive window to export the Python file as a Jupyter Notebook (.ipynb).