



专栏 / 苏州谷歌开发者社区 / 文章详情



DerekGrant RP 113

发布于 苏州谷歌开发者社区

关注专栏

2018-01-27 发布

Jupyter介绍和使用 中文版

原创 jupyter-notebook 6.1k 次阅读 · 读完需要 61 分钟

什么是 Jupyter notebooks?

[Jupyter](#) 是个集成文本，数学公式，代码和可视化的可分享文本。

Notebooks 很快已经成为了数据操作不可或缺的工具。它在 [大数据清理和探究](#), [可视化](#), [机器学习](#), 和 [大数据分析](#)中都有广泛运用。

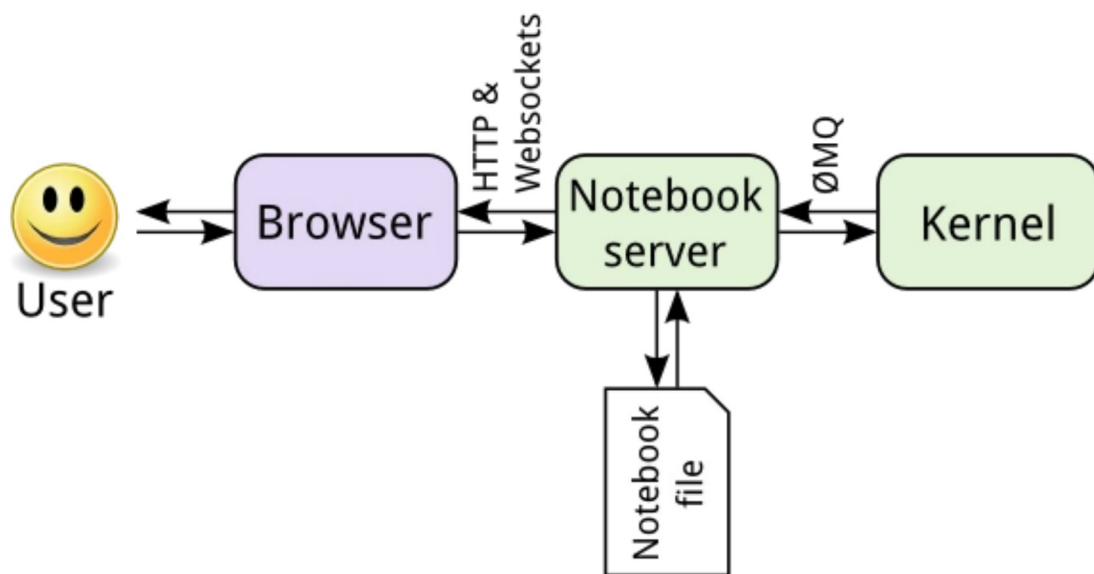
Notebooks 可以直接在github直接被读取. 这是一个非常有用的功能，你可以方便地分享。例如, [自编码器 notebook](#)

文本化编程

Notebooks是Donald Knuth 1984年提出的[文本化编程]的一种形式(<http://www.literateprogramming.com>)。结合文本化编程，文本和代码交错在一起，而不是分成两个独立地本分。

代码文档是给人写的，不是给电脑写的。这会给别人看你的代码或者你自己回头分析你之前的成果都提供了很大的便利。

[Eve](#)真正试图将文本化编程开发成一种完整的语言。



From Jupyter documentation

这个 **中心点** 是 **notebook server**. 你通过浏览器来连接**notebook这个渲染好的网页应用**. 你写的代码通过 server传给kernel。 kernel执行代码再通过server传给浏览器。当你保存文件为 **.ipynb** 后缀的**JSON 文本**。这种架构最好的特点是kernel可以不是基于特定于Python的，你可以运行任何语言。例如你也可以运行R和 **Julia**。所以jupyter不再叫python Notebook。 **Jupyter** 是 **Julia**, **Python**, 和 **R**的组合. 如果你感兴趣，你可以浏览这个 [支持的kernel列表](#).

另外一个好处是你可以通过网络在任何一个地方访问Server。特别的是，你可以在自己的电脑上访问数据存储的地方并运行代码。你可以在一个[远程服务器上执行代码](#)。

这个特征确实很有用。

安装notebook

Anaconda是安装Jupyter最简便的方式，自带Jupyter.

在Conda中安装，可以用 `conda install jupyter notebook` .

Jupyter也可以用pip安装 `pip install jupyter notebook` .

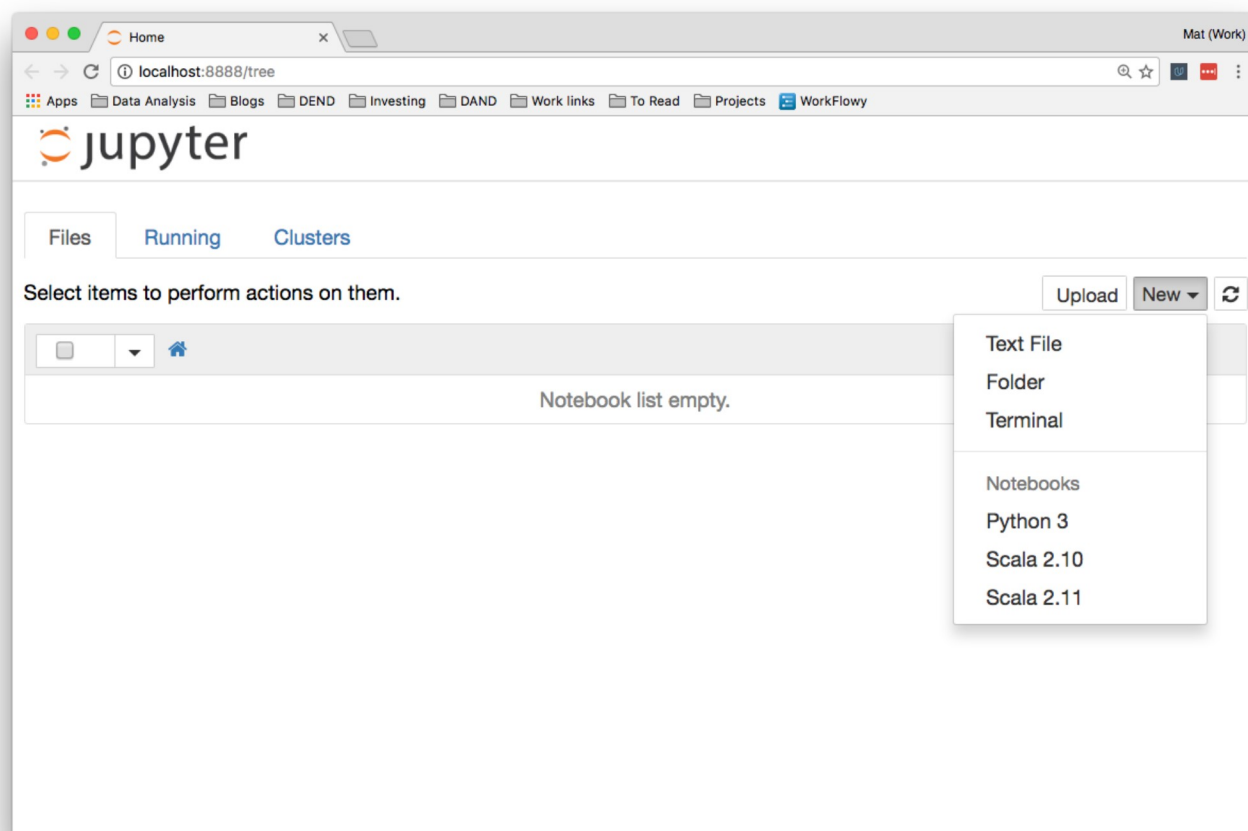
启动Jupyter服务器

在控制台或者terminal里面启动Notebook `jupyter notebook` . Jupyter需要命令行来启动。你在哪个路径下启动jupyter，文本就保存在哪个路径下。

当你启动jupyter，它的主页会在浏览器中自动打开。默认地址为<http://localhost:8888>. `localhost` 指你的本机，而不是网络中其他电脑。`8888` 是你连接的端口. 只要server是在运行的, 你都可以在浏览器中访问<http://localhost:8888>。

如果你另外开了一个server，它会默认想去启动 8888端口,但因为已经占用了，它会去默认开启端口8889. 你可以访问<http://localhost:8889>.

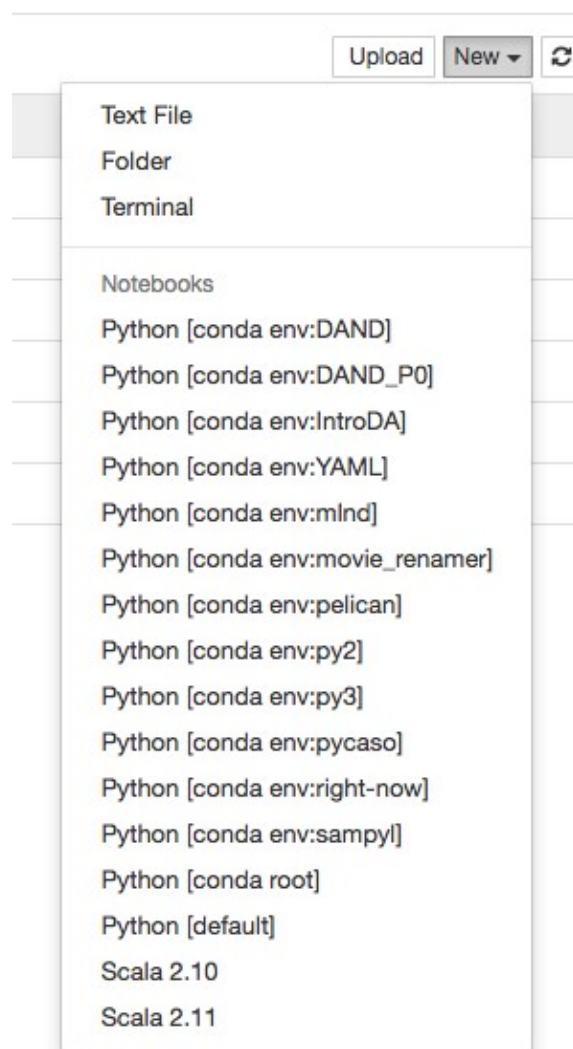
例如这样:



你看到的文件列表取决于你在哪里启动。

我们从点击“新建”开始创建notebook，文本文件，文件夹或terminal。这个按钮下的列表显示了你已经安装的kernel种类。我们现在创建Python 3环境。你可以看到我还装了Scala 2.

如果你安装了kernel，还会涉及到选择哪个环境。例如：



conda environments in Jupyter

顶部的标签有 *Files*, *Running*, 和 *Cluster*. *Files* 现在有哪些文件. *Running* 标签列出了现在运行的 notebooks.

Clusters 指多kernel并行计算. 它由 [ipyparallel]支持(<https://ipyparallel.readthedocs...>). 这里就不多讲了。

如果你在运行Conda环境，那么还有一个"Conda" 标签. 你可以在这里安装管理包和环境。



Files Running Clusters Conda

17 Conda environments



Action	Name	Default?	Directory
	root		/Users/mat/anaconda
	DAND		/Users/mat/anaconda/envs/DAND
	DAND_P0		/Users/mat/anaconda/envs/DAND_P0
	IntroDA		/Users/mat/anaconda/envs/IntroDA
	YAML		/Users/mat/anaconda/envs/YAML
	flask		/Users/mat/anaconda/envs/flask

605 available packages

Search...



62 installed packages in environment "DAND"



Name	Version	Channel
<input type="checkbox"/> _license	1.1	defaults
<input type="checkbox"/> _nb_ext_conf	0.3.0	defaults
<input type="checkbox"/> abstract-rendering	0.5.1	defaults
<input type="checkbox"/> accelerate	2.3.0	defaults
<input type="checkbox"/> accelerate_cudalib	2.0	defaults
<input type="checkbox"/> affine	2.0.0	defaults

Name	Version	Build	Available
<input type="checkbox"/> appnope	0.1.0	py27_0	
<input type="checkbox"/> backports	1.0	py27_0	
<input type="checkbox"/> backports_abc	0.4	py27_0	
<input type="checkbox"/> configparser	3.5.0b2	py27_1	
<input type="checkbox"/> cycler	0.10.0	py27_0	
<input type="checkbox"/> decorator	4.0.10	py27_0	

关闭jupyter

你可以保存文件后点击按钮 "Shutdown"关闭单个notebook。



Files Running Clusters Conda

Duplicate

Shutdown



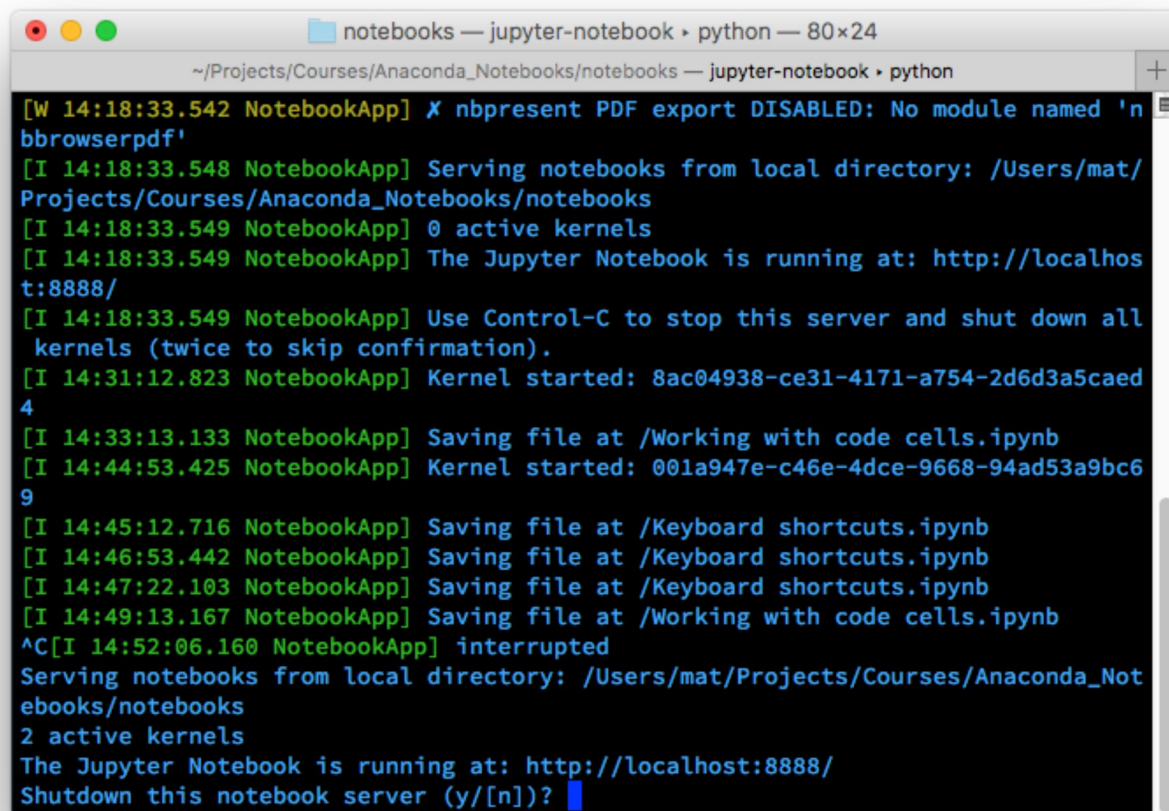
Upload

New



1	
<input checked="" type="checkbox"/> Keyboard shortcuts.ipynb	Running
<input type="checkbox"/> Magic commands.ipynb	
<input type="checkbox"/> Markdown cells.ipynb	
<input type="checkbox"/> Untitled.ipynb	
<input type="checkbox"/> Working with code cells.ipynb	Running
<input type="checkbox"/> Working with code cells.html	

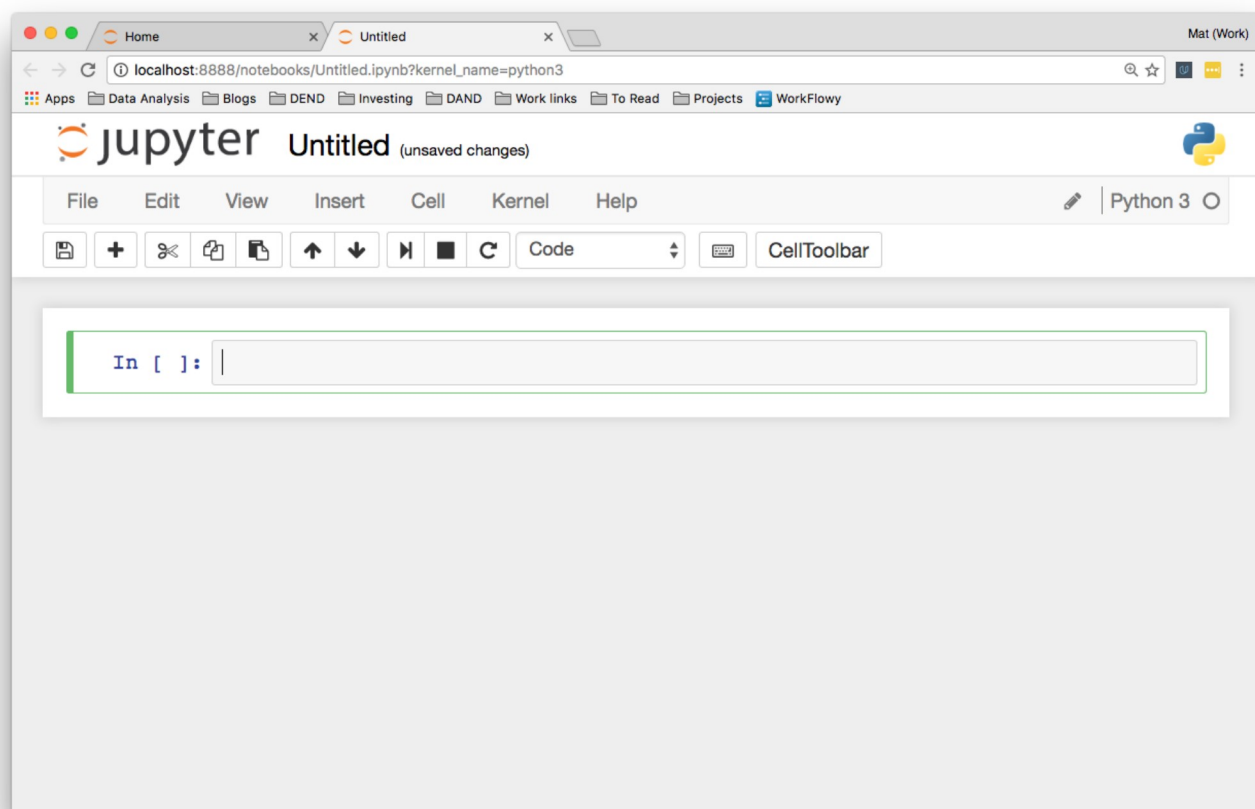
另外你可以在保存后在server里面按ctrl/command + C 来关闭server。



```
notebooks — jupyter-notebook • python — 80x24
~/Projects/Courses/Anaconda_Notebooks/notebooks — jupyter-notebook • python

[W 14:18:33.542 NotebookApp] X nbpresent PDF export DISABLED: No module named 'nbrowserpdf'
[I 14:18:33.548 NotebookApp] Serving notebooks from local directory: /Users/mat/Projects/Courses/Anaconda_Notebooks/notebooks
[I 14:18:33.549 NotebookApp] 0 active kernels
[I 14:18:33.549 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/
[I 14:18:33.549 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[I 14:31:12.823 NotebookApp] Kernel started: 8ac04938-ce31-4171-a754-2d6d3a5caed4
[I 14:33:13.133 NotebookApp] Saving file at /Working with code cells.ipynb
[I 14:44:53.425 NotebookApp] Kernel started: 001a947e-c46e-4dce-9668-94ad53a9bc69
[I 14:45:12.716 NotebookApp] Saving file at /Keyboard shortcuts.ipynb
[I 14:46:53.442 NotebookApp] Saving file at /Keyboard shortcuts.ipynb
[I 14:47:22.103 NotebookApp] Saving file at /Keyboard shortcuts.ipynb
[I 14:49:13.167 NotebookApp] Saving file at /Working with code cells.ipynb
^C[I 14:52:06.160 NotebookApp] interrupted
Serving notebooks from local directory: /Users/mat/Projects/Courses/Anaconda_Notebooks/notebooks
2 active kernels
The Jupyter Notebook is running at: http://localhost:8888/
Shutdown this notebook server (y/[n])? 
```

界面



Feel free to try this yourself and poke around a bit.

绿色的一个方框叫`cell`. Cells是你写代码和执行的地方。你可以修改成文本然后用 Markdown来渲染。

当你运行代码时，左边有数字显示运行的次序 像 `In [1]:`。

工具条

从左向右

- 保存
- 新建 cell +
- 剪切，复制，粘贴
- 启动，停止，重启
- Cell 种类: 代码, Markdown, 文本, 标题
- Command palette (see next)

- Cell toolbar, 可以把Notebook转化为slides , PPT

Command palette

The little keyboard is the command palette. This will bring up a panel with a search bar where you can search for various commands. This is really helpful for speeding up your workflow as you don't need to search around in the menus with your mouse. Just open the command palette and type in what you want to do. For instance, if you want to merge two cells:

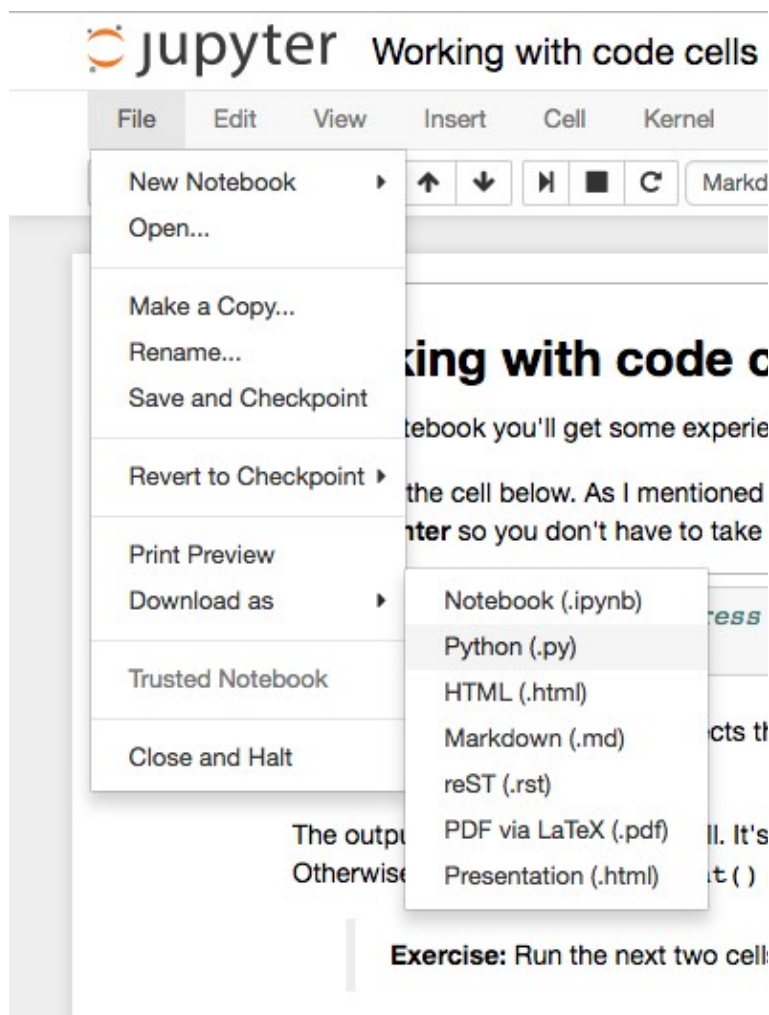
More things

At the top you see the title. Click on this to rename the notebook.

Over on the right is the kernel type (Python 3 in my case) and next to it, a little circle. When the kernel is running a cell, it'll fill in. For most operations which run quickly, it won't fill in. It's a little indicator to let you know longer running code is actually running.

Along with the save button in the toolbar, notebooks are automatically saved periodically. The most recent save is noted to the right of the title. You can save manually with the save button, or by pressing escape then s on your keyboard. The `escape` key changes to command mode and `s` is the shortcut for "save." I'll cover command mode and keyboard shortcuts later.

In the "File" menu, you can download the notebook in multiple formats. You'll often want to download it as an HTML file to share with others who aren't using Jupyter. Also, you can download the notebook as a normal Python file where all the code will run like normal. The `Markdown` and `reST` formats are great for using notebooks in blogs or documentation.



Code cells

Most of your work in notebooks will be done in code cells. This is where you write your code and it gets executed. In code cells you can write any code, assigning variables, defining functions and classes, importing packages, and more. Any code executed in one cell is available in all other cells.

To give you some practice, I created a notebook you can work through. Download the notebook [Working With Code Cells](#) then run it from your own notebook server. (In your terminal, change to the directory with the notebook file, then enter `jupyter notebook`) Your browser might try to open the notebook file without downloading it. If that happens, right click on the link then choose "Save Link As..."

Markdown cells

As mentioned before, cells can also be used for text written in Markdown. Markdown is a formatting syntax that allows you to include links, style text as bold or italicized, and format code. As with code cells, you press **Shift + Enter** or **Control + Enter** to run the Markdown cell, where it will render the Markdown to formatted text. Including text allows you to write a narrative along side your code, as well as documenting your code and the thoughts that went into it.

You can find the [documentation here](#), but I'll provide a short primer.

Headers

You can write headers using the pound/hash/**octothorpe** symbol `#` placed before the text. One `#` renders as an h1 header, two `#` s is an h2, and so on. Looks like this:

```
# Header 1
## Header 2
### Header 3
```

renders as

Header 1

Header 2

Header 3

Links

Linking in Markdown is done by enclosing text in square brackets and the URL in parentheses, like this `[Udacity's home page](https://www.udacity.com)` for a link to Udacity's home page.

Emphasis

You can add emphasis through bold or italics with asterisks or underscores (`*` or `_`). For italics, wrap the text in one asterisk or underscore, `_gelato_` or `*gelato*` renders as *gelato*.

Bold text uses two symbols, `**aardvark**` or `__aardvark__` looks like **aardvark**.

Either asterisks or underscores are fine as long as you use the same symbol on both sides of the text.

Code

There are two different ways to display code, inline with text and as a code block separated from the text. To format inline code, wrap the text in backticks. For example, ``string.punctuation`` renders as `string.punctuation`.

To create a code block, start a new line and wrap the text in three backticks

```
import requests
response = requests.get('https://www.udacity.com')
```

or indent each line of the code block with four spaces.

```
import requests
response = requests.get('https://www.udacity.com')
```

Math expressions

You can create math expressions in Markdown cells using **LaTeX** symbols. Notebooks use MathJax to render the LaTeX symbols as math symbols. To start math mode, wrap the LaTeX in dollar signs `$y = mx + b$` for inline math. For a math block, use double dollar signs,

`$$`

```
y = \frac{a}{b+c}
$$
```

This is a really useful feature, so if you don't have experience with LaTeX [please read this primer](#) on using it to create math expressions.

Wrapping up

Here's [a cheatsheet](#) you can use as a reference for writing Markdown. My advice is to make use of the Markdown cells. Your notebooks will be much more readable compared to a bunch of code blocks.

Keyboard shortcuts

Notebooks come with a bunch of keyboard shortcuts that let you use your keyboard to interact with the cells, instead of using the mouse and toolbars. They take a bit of time to get used to, but when you're proficient with the shortcuts you'll be much faster at working in notebooks. To learn more about the shortcuts and get practice using them, download the notebook [Keyboard Shortcuts](#). Again, your browser might try to open it, but you want to save it to your computer. Right click on the link, then choose "Save Link As..."

Switching between Markdown and code

With keyboard shortcuts, it is quick and simple to switch between Markdown and code cells. To change from Markdown to cell, press **Y**. To switch from code to Markdown, press **M**.

Line numbers

A lot of times it is helpful to number the lines in your code for debugging purposes. You can turn on numbers by pressing **L** (in command mode of course) on a code cell.

Deleting cells

Deleting cells is done by pressing **D** twice in a row so **D**, **D**. This is to prevent accidentally deletions, you have to press the button twice!

The Command Palette

You can easily access the command palette by pressing Shift + Control/Command + **P**.

Note: This won't work in Firefox and Internet Explorer unfortunately. There is already a keyboard shortcut assigned to those keys in those browsers. However, it does work in Chrome and Safari.

This will bring up the command palette where you can search for commands that aren't available through the keyboard shortcuts. For instance, there are buttons on the toolbar that move cells up and down (the up and down arrows), but there aren't corresponding keyboard shortcuts. To move a cell down, you can open up the command palette and type in "move" which will bring up the move commands.

Magic keywords

Magic keywords are special commands you can run in cells that let you control the notebook itself or perform system calls such as changing directories. For example, you can set up matplotlib to work interactively in the notebook with **%matplotlib**.

Magic commands are preceded with one or two percent signs (% or %%) for line magics and cell magics, respectively. Line magics apply only to the line the magic command is written on, while cell magics apply to the whole cell.

NOTE: These magic keywords are specific to the normal Python kernel. If you are using other kernels, these most likely won't work.

Timing code

At some point, you'll probably spend some effort optimizing code to run faster. Timing how quickly your code runs is essential for this optimization. You can use the `timeit` magic command to time how long it takes for a function to run, like so:

```
In [21]: from math import sqrt
```

```
def fibol(n): # Recursive Fibonacci number
    if n == 0:
        return 0
    elif n == 1:
        return 1
    return fibol(n-1) + fibol(n-2)

def fibo2(n): # Closed form
    return ((1+sqrt(5))**n-(1-sqrt(5))**n)/(2**n*sqrt(5))
```

```
In [22]: %timeit fibol(20)
```

100 loops, best of 3: 3.49 ms per loop

```
In [23]: %timeit fibo2(20)
```

The slowest run took 16.75 times longer than the fastest. This could mean that an intermediate result is being cached.
1000000 loops, best of 3: 1.08 μ s per loop

If you want to time how long it takes for a whole cell to run, you'd use `%%timeit` like so:

```
In [24]: import random
```

```
In [97]: %%timeit
prize = 0
for ii in range(100):
    # roll a die
    roll = random.randint(1, 6)
    if roll%2 == 0:
        prize += roll
    else:
        prize -= 1
```

10000 loops, best of 3: 148 μ s per loop

```
In [98]: %%timeit
rolls = (random.randint(1,6) for _ in range(100))
prize = sum(roll if roll%2 == 0 else -1 for roll in rolls)
```

10000 loops, best of 3: 154 μ s per loop

Embedding visualizations in notebooks

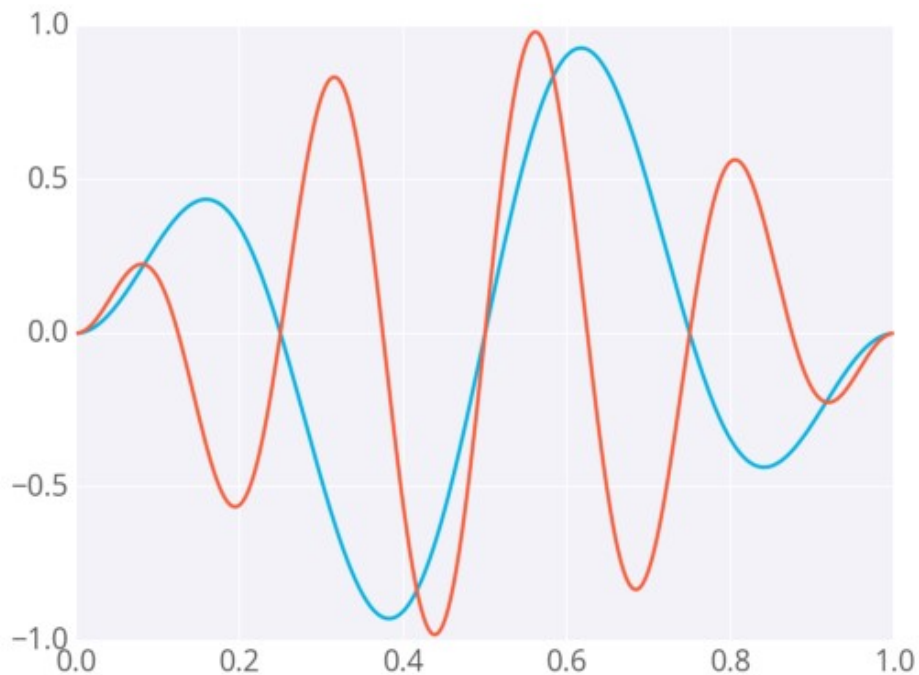
As mentioned before, notebooks let you embed images along with text and code. This is most useful when you're using `matplotlib` or other plotting packages to create visualizations. You can use `%matplotlib` to set up matplotlib for interactive use in the notebook. By default figures will render in their own window. However, you can pass arguments to the command to select a specific "backend", the software that renders the image. To render figures directly in the notebook, you should use the inline backend with the command `%matplotlib inline`.

Tip: On higher resolution screens such as Retina displays, the default images in notebooks can look blurry. Use `%config InlineBackend.figure_format = 'retina'` after `%matplotlib inline` to render higher resolution images.

```
In [103]: %matplotlib inline
          %config InlineBackend.figure_format = 'retina'

          import matplotlib.pyplot as plt
          import numpy as np

In [134]: x = np.linspace(0, 1, 300)
          for w in range(2, 6, 2):
              plt.plot(x, np.sin(np.pi*x)*np.sin(2*w*np.pi*x))
```



Debugging in the Notebook

With the Python kernel, you can turn on the interactive debugger using the magic command `%pdb`. When you cause an error, you'll be able to inspect the variables in the current namespace.

```
In [99]: %pdb
```

Automatic pdb calling has been turned ON

```
In [*]: numbers = 'hello'
sum(numbers)
```

```
-----
-----
TypeError                                Traceback (most recent call
last)
<ipython-input-101-7a179164921f> in <module>()
      1 numbers = 'hello'
----> 2 sum(numbers)

TypeError: unsupported operand type(s) for +: 'int' and 'str'

> <ipython-input-101-7a179164921f>(2)<module>()
      1 numbers = 'hello'
----> 2 sum(numbers)

ipdb> numbers
'hello'

ipdb> 
```

Above you can see I tried to sum up a string which gives an error. The debugger raises the error and provides a prompt for inspecting your code.

Read more about `pdb` in [the documentation](#). To quit the debugger, simply enter `q` in the prompt.

[Python 代码调试技巧](#)

More reading

There are a whole bunch of other magic commands, I just touched on a few of the ones you'll use the most often. To learn more about them, [here's the list](#) of all available magic commands.

Converting notebooks

Notebooks are just big **JSON** files with the extension **.ipynb**.

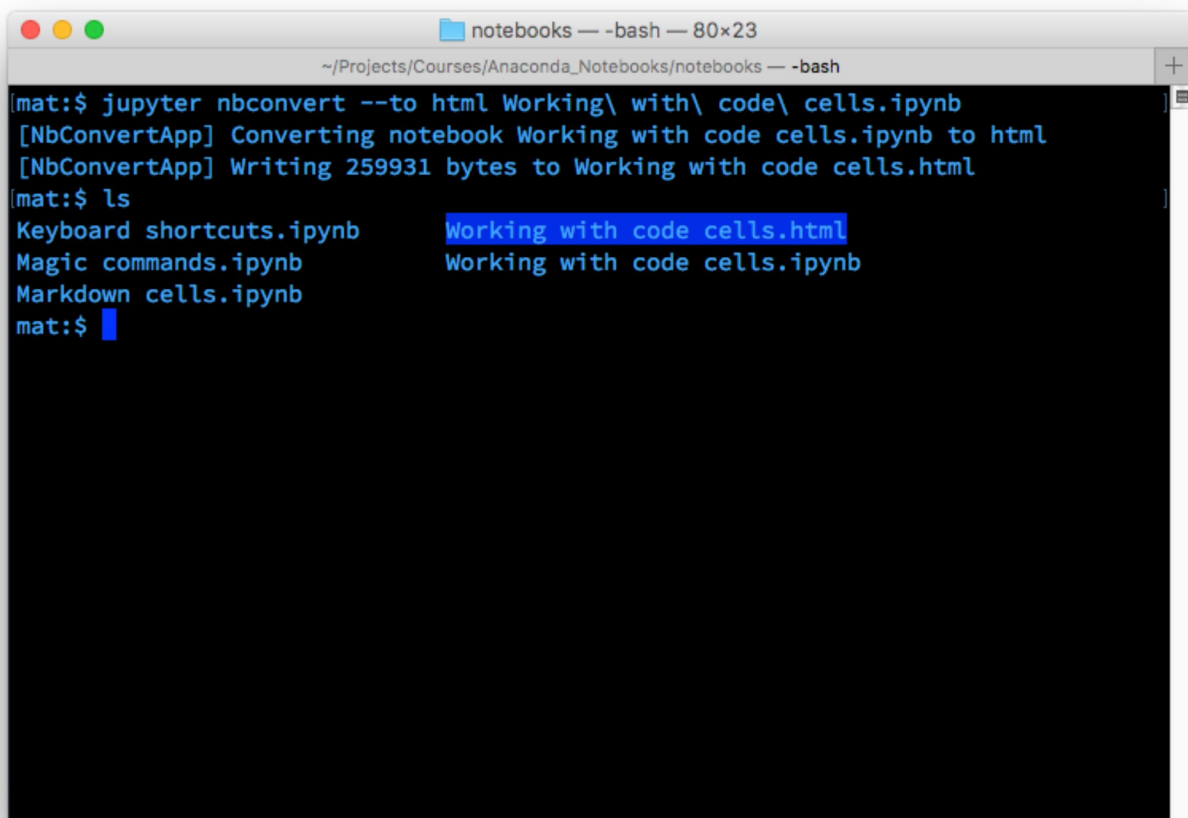
[Notebook file opened in a text editor shows JSON data](#)

Since notebooks are JSON, it is simple to convert them to other formats. Jupyter comes with a utility called **nbconvert** for converting to HTML, Markdown, slideshows, etc.

For example, to convert a notebook to an HTML file, in your terminal use

```
jupyter nbconvert --to html notebook.ipynb
```

Converting to HTML is useful for sharing your notebooks with others who aren't using notebooks. Markdown is great for including a notebook in blogs and other text editors that accept Markdown formatting.

A terminal window titled 'notebooks — -bash — 80x23' with a path of '~/Projects/Courses/Anaconda_Notebooks/notebooks — -bash'. The terminal shows the command 'jupyter nbconvert --to html Working\ with\ code\ cells.ipynb' being executed. The output indicates that the notebook 'Working with code cells.ipynb' is being converted to HTML and 259931 bytes are being written to 'Working with code cells.html'. Following this, the user runs 'ls', which lists several files: 'Keyboard shortcuts.ipynb', 'Magic commands.ipynb', 'Markdown cells.ipynb', and the newly created 'Working with code cells.html'. The prompt 'mat:\$' is visible at the end of the command line.

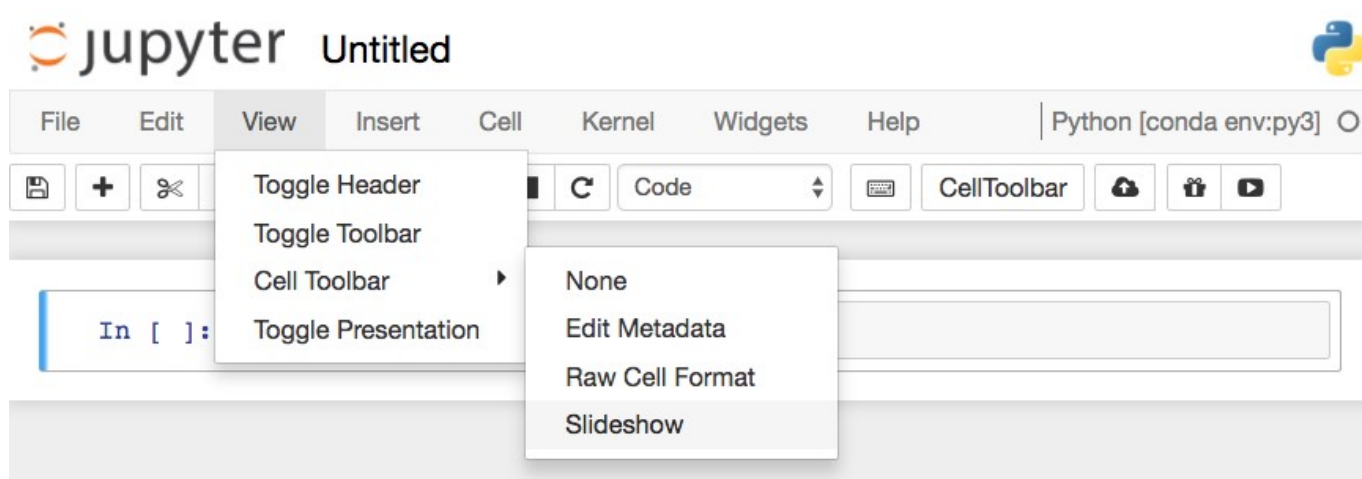
```
notebooks — -bash — 80x23
~/Projects/Courses/Anaconda_Notebooks/notebooks — -bash
mat:$ jupyter nbconvert --to html Working\ with\ code\ cells.ipynb
[NbConvertApp] Converting notebook Working with code cells.ipynb to html
[NbConvertApp] Writing 259931 bytes to Working with code cells.html
mat:$ ls
Keyboard shortcuts.ipynb      Working with code cells.html
Magic commands.ipynb        Working with code cells.ipynb
Markdown cells.ipynb
mat:$
```

As always, learn more about **nbconvert** from the [documentation](#).

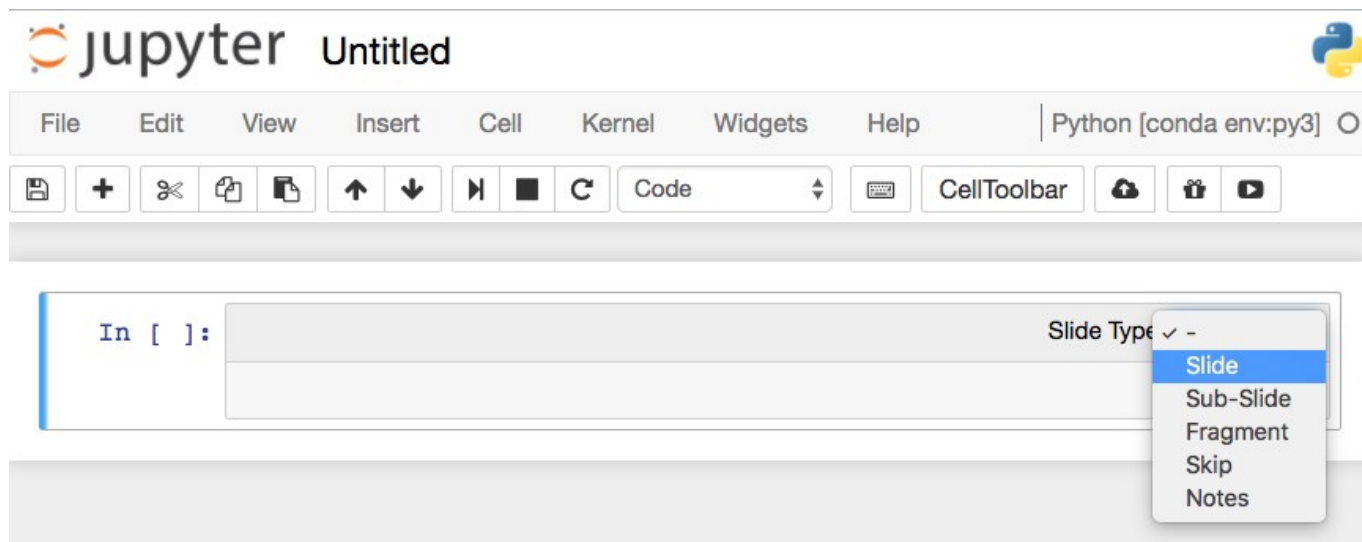
Creating a slideshow

Create slideshows from notebooks is one of my favorite features. You can see [an example of a slideshow](#) here introducing Pandas for working with data.

The slides are created in notebooks like normal, but you'll need to designate which cells are slides and the type of slide the cell will be. In the menu bar, click View > Cell Toolbar > Slideshow to bring up the slide cell menu on each cell.



This will show a menu dropdown on each cell that lets you choose how the cell shows up in the slideshow.



Slides are full slides that you move through left to right. **Sub-slides** show up in the slideshow by pressing up or down. **Fragments** are hidden at first, then appear with a button press. You can skip cells in the slideshow with **Skip** and **Notes** leaves the cell as speaker notes.

Running the slideshow

To create the slideshow from the notebook file, you'll need to use `nbconvert` :

```
jupyter nbconvert notebook.ipynb --to slides
```

This just converts the notebook to the necessary files for the slideshow, but you need to serve it with an HTTP server to actually see the presentation.

To convert it and immediately see it, use

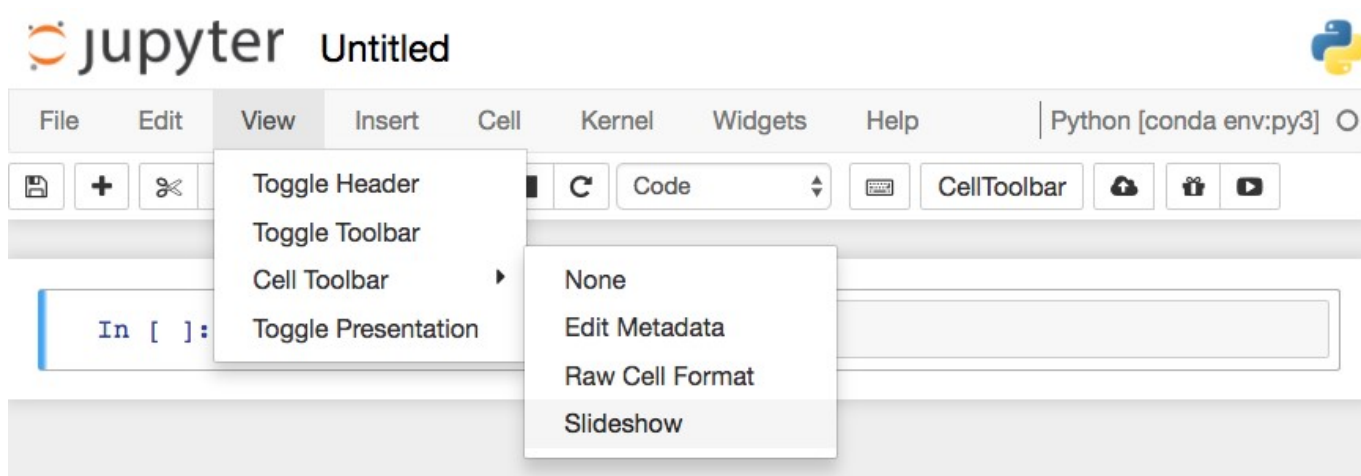
```
jupyter nbconvert notebook.ipynb --to slides --post serve
```

This will open up the slideshow in your browser so you can present it.

Creating a slideshow

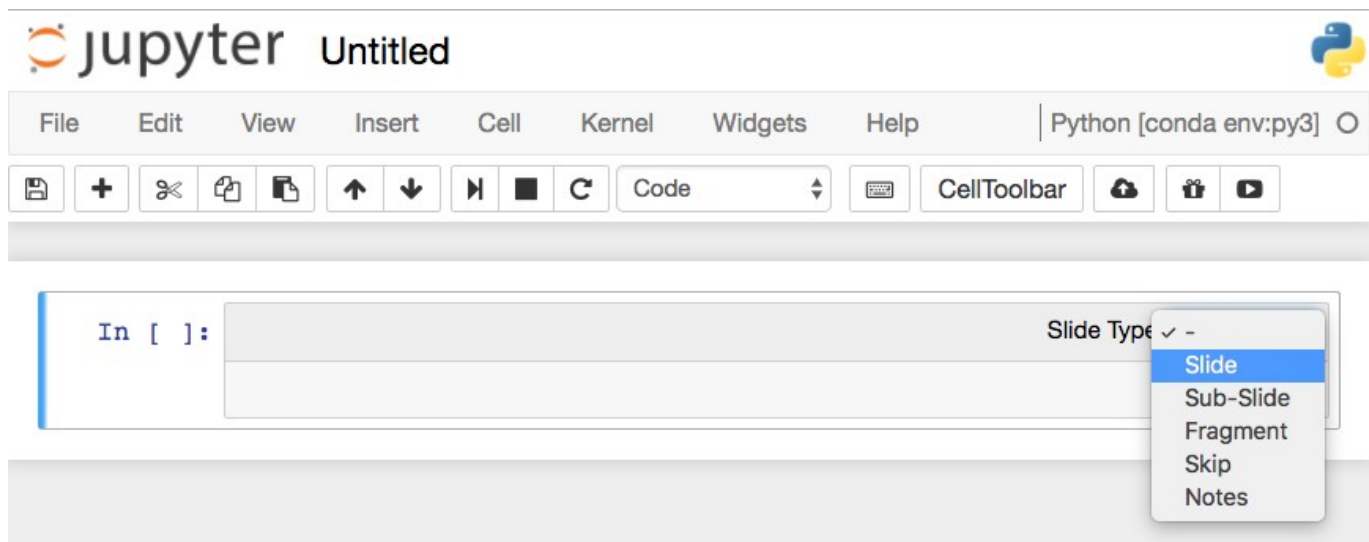
Create slideshows from notebooks is one of my favorite features. You can see [an example of a slideshow](#) here introducing Pandas for working with data.

The slides are created in notebooks like normal, but you'll need to designate which cells are slides and the type of slide the cell will be. In the menu bar, click View > Cell Toolbar > Slideshow to bring up the slide cell menu on each cell.



This will show a menu dropdown on each cell that lets you choose how the cell shows up in the

slideshow.



Slides are full slides that you move through left to right. **Sub-slides** show up in the slideshow by pressing up or down. **Fragments** are hidden at first, then appear with a button press. You can skip cells in the slideshow with **Skip** and **Notes** leaves the cell as speaker notes.

Running the slideshow

To create the slideshow from the notebook file, you'll need to use `nbconvert` :

```
jupyter nbconvert notebook.ipynb --to slides
```

这个命令并不能生成直接可以查看的slides

This just converts the notebook to the necessary files for the slideshow, but you need to serve it with an HTTP server to actually see the presentation.

To convert it and immediately see it, use

```
jupyter nbconvert notebook.ipynb --to slides --post serve
```

This will open up the slideshow in your browser so you can present it.

这样可以直接用Jupyter制作幻灯片，很实用

赞 | 2

收藏 | 1

你可能感兴趣的

- **【架构学习】大数据在线比赛平台架构设计** NicolasHe k8s 架构设计 jupyter-notebook python
- **搭建Python Jupyter Notebook教程** 丹追兵 python jupyter-notebook
- **Jupyter 常见可视化框架选择** 三次方根 可视化 python jupyter-notebook
- **搭建一个jupyter站点做数据分析吧** kamushin233 jupyter-notebook
- **装扮你的Jupyter** 三次方根 jupyter-notebook python ipython matplotlib
- **Windows下安装Jupyter，作为后台服务运行** Kaciras 人工智能 jupyter-notebook python
- **jupyter安装总结** 梦见山 python jupyter-notebook
- **在Centos7上搭建Jupyter Notebook环境** Jinkey centos7 jupyter-notebook

8 条评论

默认排序 | 时间排序



瘦夫子 · 2018年07月26日

你好，看到你发的jupyter的文章，想请你就这方面写书，不知道是否有兴趣？这个工具的使用需求挺大，急需好的入门教程，盼复，瘦夫子，人民邮电出版社

👍 赞 回复

您好，很高兴收到您的邀请。个人感觉Jupyter作为一个工具，大部分人的使用一般限于基本的功能。如果作为一本书的话，您期望包含哪些内容呢，谢谢！

— **DerekGrant** 作者 · 2018年07月26日

我编辑的图书《深入浅出数据科学》这本书，是张星辰翻译的，书上有你的名字啊。加我qq聊吧，130 222 5769。

— **瘦夫子** · 2018年09月27日

哈哈，缘分：)

— **DerekGrant** 作者 · 2018年09月28日

添加回复



纸短情长 · 2018年09月13日

怎么在jupyter列出各种资源的url，如html,css,js,jpg等。

👍 赞 回复

列出是什么意思？切换到markdown本身就支持html

— [DerekGrant](#) 作者 · 2018年09月14日

就是输出有多少个html、css,相当于汇总。

— [纸短情长](#) · 2018年09月14日

我刚接触，可能描述不太准确

— [纸短情长](#) · 2018年09月14日

[添加回复](#)



文明社会，理性评论

发表评论

产品

热门问答
热门专栏
热门讲堂
最新活动
技术圈
酷工作
移动客户端

课程

Java 开发课程
PHP 开发课程
Python 开发课程
前端开发课程
移动开发课程

资源

每周精选
用户排行榜
徽章
帮助中心
声望与权限
社区服务中心

合作

关于我们
广告投放
职位发布
讲师招募
联系我们

关注

产品技术日志
社区运营日志
市场运营日志
团队日志
社区访谈

条款

服务条款
内容许可



扫一扫下载 App

Copyright © 2011-2019 SegmentFault. 当前呈现版本 19.02.27
浙ICP备 15005796号-2 浙公网安备 33010602002000号 杭州堆栈科技有限公司版权所有

CDN 存储服务由 又拍云 赞助提供