

科研经验分享

为交互式计算而生——Jupyter项目

赵相杰

2023年2月27日

In 2021, Nature named Jupyter as one of ten computing projects that transformed science

TEN COMPUTER CODES THAT TRANSFORMED SCIENCE

From Fortran to preprint archives, these advances in programming and platforms saved biology, climate science and physics to new heights. By Jeffrey M. Perkel

In 2019, the Event Horizon Telescope team gave the world the first glimpse of what a black hole actually looks like. But the image of a glowing, ring-shaped object that the group unveiled wasn't a conventional photograph. It was computed – a mathematical transformation of data captured by radio telescopes in the United States, Mexico, Chile, Spain and the South Pole*. The team released the programming code it used to accomplish that feat alongside the articles that documented its findings, so the scientific community could see – and build on – what it had done.

It's an increasingly common pattern. From astronomy to zoology, behind every great scientific finding of the modern age, there is a computer. Michael Levitt, a computational biologist at Stanford University in California who won a share of the 2013 Nobel Prize in Chemistry for his work on computational strategies for modelling chemical structure, notes that today's laptops have about

10,000 times the memory and clock speed that his lab-built computer had in 1967, when he began his prizewinning work. "We really do have quite phenomenal amounts of computing at our hands today," he says. "Trouble is, it still requires thinking."

Enter the scientist-coder. A powerful computer is useless without software capable of tackling research questions – and researchers who know how to write it and use it. "Research is now fundamentally connected to software," says Neil Chue Hong, director of the Software Sustainability Institute, headquartered in Edinburgh, UK, an organization dedicated to improving the development and use of software in science. "It permeates every aspect of the conduct of research."

Scientific discoveries rightly get top billing in the media. But *Nature* this week looks behind the scenes, at the key pieces of code that have transformed research over the past few decades. "It permeates every aspect of the conduct of research."

Scientific discoveries rightly get top billing in the media. But *Nature* this week looks behind the scenes, at the key pieces of code that have transformed research over the past few decades. "It permeates every aspect of the conduct of research."

Scientific discoveries rightly get top billing in the media. But *Nature* this week looks behind the scenes, at the key pieces of code that have transformed research over the past few decades. "It permeates every aspect of the conduct of research."

we polled dozens of researchers over the past year to develop a diverse line-up of ten software tools that have had a big impact on the world of science.

Language pioneer: the Fortran compiler (1957)

The first modern computers weren't user-friendly. Programming was literally done by hand, by connecting banks of circuits with wires. Subsequent machine and assembly languages allowed users to program computers in code, but both still required an intimate knowledge of the computer's architecture, putting the languages out of reach of many scientists.

That changed in the 1950s with the development of symbolic languages – in particular the 'formula translation' language Fortran, developed by John Backus and his team at IBM in San Jose, California. Using Fortran, users could program computers using human-readable instructions, such as $x = 3 + 5$. A compiler then turned such directions into fast, efficient machine code.

It still wasn't easy: in the early days, programmers used punch cards to input code, and a complex simulation might require tens of thousands of them. Still, says Syukuro Manabe, a climatologist at Princeton University in New Jersey, Fortran made programming accessible to researchers who weren't computer scientists. "For the first time, we were able to program (the computer) by ourselves," Manabe says. He and his colleagues used the language to develop one of the first successful climate models.

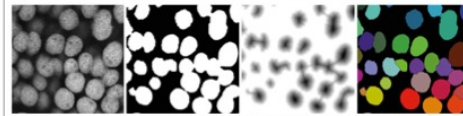
Now in its eighth decade, Fortran is still widely used in climate modelling, fluid dynamics, computational chemistry – any discipline that involves complex linear algebra and requires powerful computers to crunch numbers quickly. The resulting code is fast, and there are still plenty of programmers who know how to write it. Vintage Fortran code bases are still alive and kicking in labs and on supercomputers worldwide. "Old time programmers knew what they were doing," says Frank Giraldo, an applied mathematician and climate modeller at the Naval Postgraduate School in Monterey, California. "They were very mindful of memory, because they had so little of it."

Signal processor: fast Fourier transform (1965)

When radioastronomers scan the sky, they capture a cacophony of complex signals changing with time. To understand the nature of those radio waves, they need to see what those signals look like as a function of frequency. A mathematical process called a Fourier transform allows researchers to do that. The problem is that it's inefficient, requiring N^2 calculations for a data set of size N .

In 1965, US mathematicians James Cooley and John Tukey worked out a way to accelerate the process. Using recursion, a divide and

Feature



The ImageJ tool can analyse microscope images and automatically identify cell nuclei, as here.

con more generally viewed as obvious and natural," Ginsparg says. "In that sense, it's like a successful research project."

Preprint powerhouse: arXiv.org (1991)

In the late 1980s, high-energy physicists routinely sent physical copies of their submitted manuscripts to colleagues by post for comment and as a courtesy – but only to a select few. "Those lower in the food chain relied on the beneficence of those on the A-list, and aspiring researchers at non-elite institutions were frequently out of the privileged loop entirely," wrote physicist Paul Ginsparg in 2011 (ref. 7).

In 1991, Ginsparg, then at Los Alamos National Laboratory in New Mexico, wrote an e-mail autoresponder to level the playing field. Subscribers received daily lists of preprints, each associated with an article identifier. With a single e-mail, users across the world could submit or retrieve an article from the lab's computer system, get lists of new articles or search by author or title.

Ginsparg's plan was to retain articles for three months, and to limit content to the high-energy physics community. But a colleague convinced him to retain the articles indefinitely. "That was the moment it transitioned from bulletin board to archive," he says. And papers flooded in from much farther afield than Ginsparg's own discipline. In 1993, Ginsparg migrated the system to the World Wide Web, and in 1998 he gave it the name it goes by today: arXiv.org.

Now in its thirtieth year, arXiv houses some 1.8 million preprints – all available for free – and attracts more than 15,000 submissions and some 30 million downloads per month. "It's not hard to see why the arXiv is such a popular service," the editors of *Nature Photonics* wrote* a decade ago on the occasion of the site's twentieth anniversary. "The system provides researchers with a fast and convenient way to plant a flag that shows what they did and when, avoiding the hassle and time required for peer review at a conventional journal."

The site's success catalysed a boom in sister archives in biology, medicine, sociology and other disciplines. The impact can be seen today in tens of thousands of preprints that have been published on the virus SARS-CoV-2. "It's gratifying to see a methodology, considered heterodox outside of the particle-physics community 30 years ago,

Data explorer: IPython Notebook (2011)

Fernando Pérez was a graduate student "in search of procrastination" in 2001 when he decided to take on a core component of Python.

Python is an interpreted language, which means programs are executed line by line. Programmers can use a kind of computational call-and-response tool called a read-evaluate-print loop (REPL), in which they type code and a program called an interpreter executes it. A REPL allows for quick exploration and iteration, but Pérez noted that Python's wasn't built for science. It didn't allow users to easily preload modules of code, for instance, or keep data visualizations open. So Pérez wrote his own version.

The result was IPython, an 'interactive' Python interpreter that Pérez unveiled in December 2001. A decade later, physicist Brian Granger, working with Pérez and others, migrated that tool to the web browser, launching the IPython Notebook and kick-starting a data-science revolution.

Like other computational notebooks, IPython Notebook combined code, results, graphics and text in a single document. But unlike other such projects, IPython Notebook was open-source, inviting contributions from a vast developer community. And it supported Python, a popular language for scientists. In 2014, IPython evolved into Project Jupyter, supporting some 100 languages and allowing users to explore data on remote supercomputers as easily as on their own laptops.

"For data scientists, Jupyter has emerged as a de facto standard," *Nature* wrote in 2018 (ref. 9). At the time, there were 2.5 million Jupyter notebooks on the GitHub code-sharing platform; today, there are nearly 10 million, including the ones that document the 2016 discovery of gravitational waves and the 2019 imaging of a black hole. "That we made a small contribution to those projects is extremely rewarding," Pérez says.

Fast learner: AlexNet (2012)

Artificial intelligence (AI) comes in two flavours. One uses codified rules, the other enables a computer to 'learn' by emulating the

neural structure of the brain. For decades, says Geoffrey Hinton, a computer scientist at the University of Toronto, Canada, AI researchers dismissed the latter approach as "nonsense". In 2012, Hinton's graduate students Alex Krizhevsky and Ilya Sutskever proved otherwise.

The venue was ImageNet, an annual competition that challenges researchers to train an AI on a database of one million images of everyday objects, then test the resulting algorithm on a separate image set. At the time, the best algorithms miscategorized about one-quarter of them, Hinton says. Krizhevsky and Sutskever's AlexNet, a 'deep-learning' algorithm based on neural networks, reduced that error rate to 16% (ref. 10). "We basically halved the error rate, or almost halved it," says Hinton.

Hinton says the team's success in 2012 reflected the combination of a big-enough training data set, great programming and the newly emergent power of graphical processing units – the processors that were originally designed to accelerate computer video performance. "Suddenly we could run [the algorithm] 30 times faster," he says, "or learn on 30 times as much data."

The real algorithmic breakthrough, Hinton says, actually occurred three years earlier, when his lab created a neural network that could recognize speech more accurately than could conventional AIs that had been refined over decades. "It was only slightly better," Hinton says. "But already that was the writing on the wall."

Those victories heralded the rise of deep learning in the lab, the clinic and more. They're why mobile phones are able to understand spoken queries and why image-analysis tools can readily pick out cells in photomicrographs. And they are why AlexNet takes its place among the many tools that have fundamentally transformed science, and with them, the world.

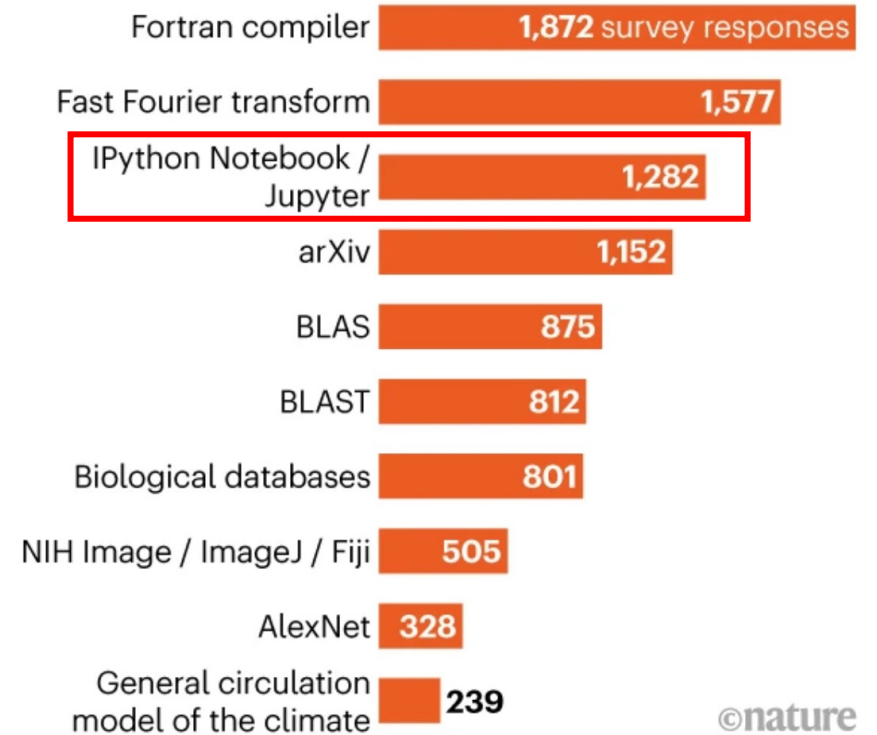
Jeffrey M. Perkel is technology editor at *Nature*.

Take a survey at go.nature.com/10-computer-codes to weigh in on our code selections.

1. The Event Horizon Telescope Collaboration et al. *Astronomy & Astrophysics*, **612**, L1 (2019).
2. Bragg, K., Adams, P. D. & Bringer, A. T. *Nature Struct. Biol.* **4**, 1083–1094 (1995).
3. Strassen, B. J. *J. Heur. Biol.* **43**, 623–660 (2010).
4. Newmark, P. *Nature* **264**, 168 (1983).
5. Manabe, S. & Bryun, K. J. *Annals. Sol.* **26**, 786–789 (1969).
6. Lawson, C. L., Hanson, R. J., Kincard, D. R. & Krogh, F. T. *ACM Trans. Math. Software* **8**, 308–323 (1978).
7. Ginsparg, P. Preprint at <http://arxiv.org/abs/hep-th/9802001>.
8. *Nature Photon.* **6**, 1 (2012).
9. *Nature* **563**, 445–446 (2018).
10. Krizhevsky, A., Sutskever, I. & Hinton, G. E. in Proc. 25th Int. Conf. Neural Information Processing Systems (eds Pereira, F., Burges, C. J. C., Bottou, L. & Weinberger, K. Q.) 1097–1105 (Curran Associates, 2012).

TOP CHOICES FOR SCIENCE CODE

Readers voted on which of the ten software codes in this article had the biggest impact on their work. They could choose up to three. Here are the results.



数据科学岗位任职要求

< 高级机器视觉工程师 ☆ ↻

任职要求：

- 1、硕士及以上学历，具有处理深度学习或工业项目的经验者优先；
- 2、掌握以下其中一种编程能力：
 - 1) 掌握 C++ 或 C#，有实际编码经验，熟练使用 VS 或 QT 等编译器；
 - 2) 掌握 python，有实际编码经验，熟练使用 jupyter notebook, PyCharm 等编译器；
- 3、具备数字图像处理及模式识别相关的基础理论和算法知识优先；
- 4、使用任一种视觉软件库 (Halcon/OpenCv)，深度框架 (如 PyTorch) 进行视觉系统开发，；
- 5、具备较强的自学能力和独立解决问题的能力，善于团队协作与沟通。
- 6、能接受出差工作安排。

< 数据分析师 ☆ ↻

职位描述

hive python 数据分析 运营

任职要求：

- 1、统招本科及以上学历学信网可查学历；
- 2、3年及以上数据分析相关工作经验，有运营经验优先；
- 3、熟练使用 Python 及 jupyter、sklearn 等工具及包，根据需求进行数据分析。
- 4、熟练编写 HSQL，熟悉 hive、hadoop 等基本操作。
- 5、熟练编写 shell 和 Python 脚本。
- 6、会写 hive+Python 即可，会写 map reduce 任务加分项

< 数据分析师 ☆ ↻

任职要求：

学历及专业：

学历：本科及以上。

专业：计算机、软件工程、通信、数学等相关专业。

专业技能：

1. 熟练掌握 Linux 下的编程，掌握 Python 语言，熟练使用 Jupyter 等工具；
2. 熟练掌握数据分析、探索、BI 等技术；
3. 熟悉数据库操作，SQL 语言，熟悉大数据技术，包括 Hive 等；
4. 深入了解机器学习、人工智能相关算法，且有一定广度者优先；
5. 熟悉人工智能领域当前热点和前沿技术，对至少一个应用领域有系统的认识；
6. 熟悉业界通用的人工智能和机器学习方案和应用流程，具有根据需求制定算法方案并完成工程开发的能力；
7. 有系统学习和应用机器学习/人工智能技术经验者优先。

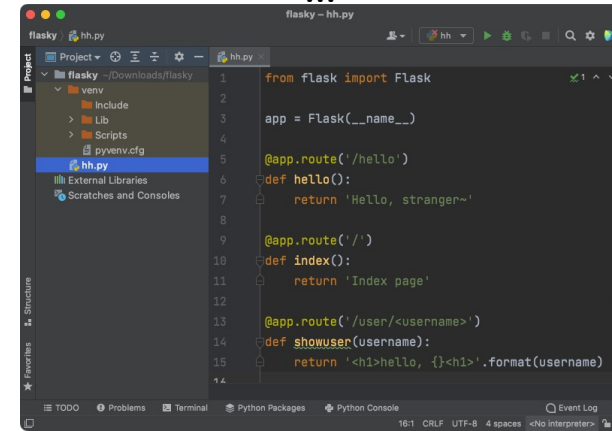


Free software, open standards, and web services for **interactive computing** across all programming languages

Pycharm Visual Studio Code

scripts (.py files)

```
Users > xiangjie > Documents > 基因组 > omics-pipeline-master > omicspipeline > bulkRNA-seq-pipeline.py
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 """
4 Created on Sun Jul 21 2019
5
6 @author: Xiangjie Zhao
7 @email: xj.zhao@genetics.ac.cn
8
9 A pipeline for bulk RNA-seq analysis
10 """
11
12 #
13 # module compatibility
14
15 from __future__ import (division, print_function, absolute_import, unicode_literals)
16 from os.path import join as join_path
17 from datetime import datetime
18
19 import os, subprocess
20 import argparse
21
22 try:
23     from future_builtins import ascii, filter, hex, map, oct, zip
24 except:
25     pass
26
27 if os.sys.version_info.major > 2:
28     xrange = range
29
30 #
31 # utils
32
33 #
34 # general
35
36 class OmicsException(Exception):
37     """custom exception"""
38     pass
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```



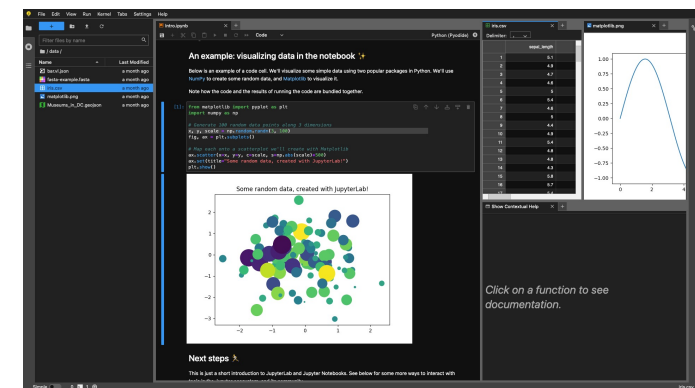
软件工程

Python

数据科学

```
[In [1]: import numpy as np
[In [2]: x = np.random.randn(1, 4)
[In [3]: x
Out[3]: array([[ -1.25920396,  0.31454629,  1.38824689, -1.30699719]])
[In [4]: x = np.random.randn(1, 4)
[In [5]: x
Out[5]: array([[ -0.62501251, -0.32113616,  0.4738682 , -1.05463289]])
```

IPython

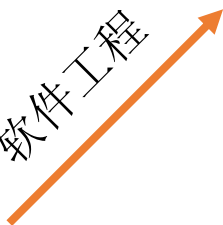


Jupyter

scripts (.py files)

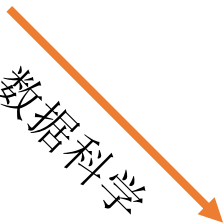
Pycharm
Visual Studio Code

软件工程



Python

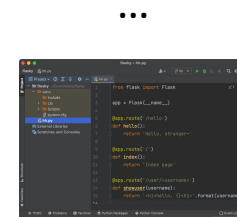
数据科学



```

Users > xiangjie > Documents > 备忘 > omics-pipeline-master > omxpipeline > bulkRNA-pipeline.py
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  """
4  Created on Sun Jul 21 2019
5
6  @author: Xiangjie Zhao
7  @email: xjzhao@genetics.ac.cn
8
9  A pipeline for bulk RNA-seq analysis
10 """
11
12 # -----
13 # module, compatibility
14
15 from __future__ import (division, print_function, absolute_import, unicode_literals)
16 from os.path import join as join_path
17 from datetime import datetime
18
19 import os, subprocess
20 import argparse
21
22 try:
23     from future_builtins import ascii, filter, hex, map, oct, zip
24 except:
25     pass
26
27 if os.sys.version_info.major > 2:
28     xrange = range
29
30 # -----
31 # utils
32
33 # -----
34 # general
35
36 class OmicsException(Exception):
37
38     """custom exception
39     """
40     pass

```

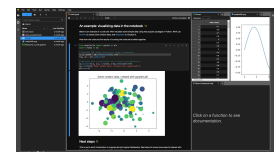


```

In [1]: import numpy as np
In [2]: x = np.random.randn(1, 1)
In [3]: x
Out[3]: array([[ 0.25528794, - 0.31454619,  1.38246054, -1.30597121]])
In [4]: x = np.random.randn(1, 1)
In [5]: x
Out[5]: array([[ 0.82081251, - 0.30218014,  0.4728882 , -1.04612851]])

```

IPython

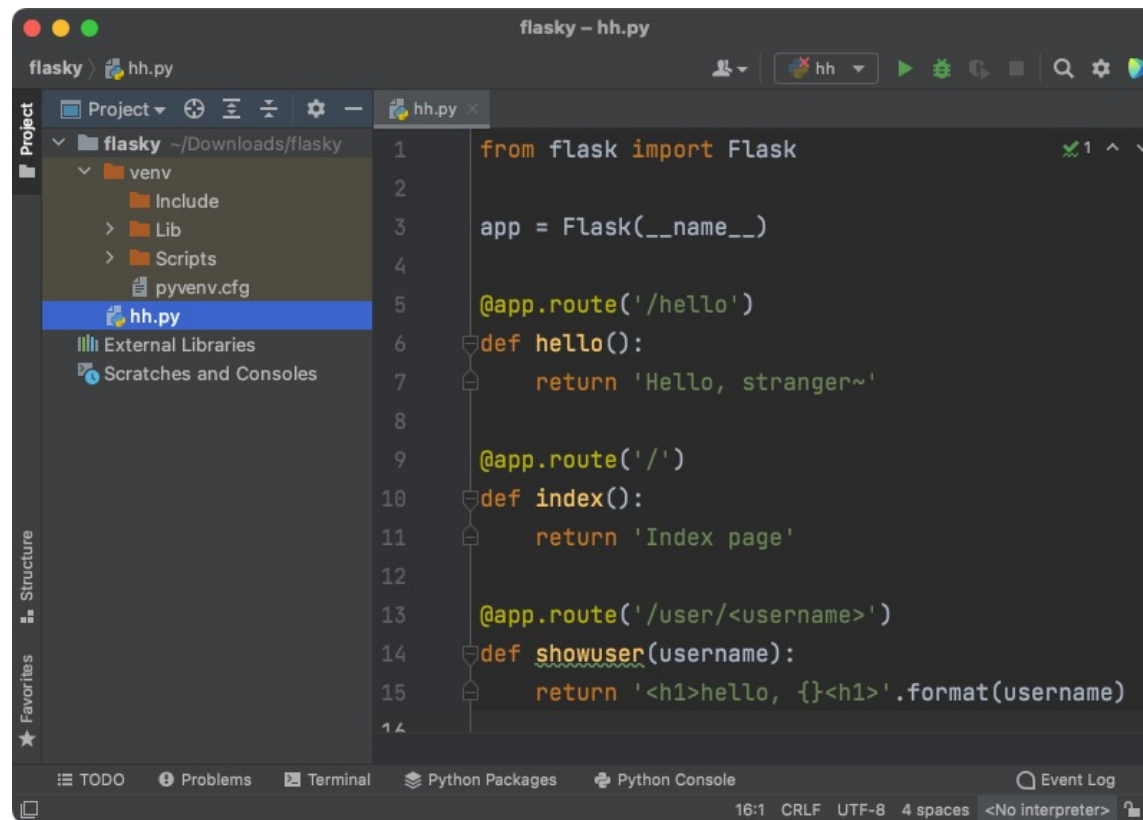


Jupyter

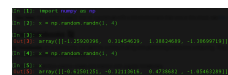
scripts (.py files)

Pycharm
Visual Studio Code

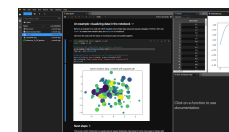
...



Python



IPython

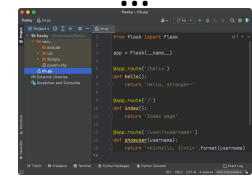


Jupyter

scripts (.py files)



Pycharm
Visual Studio Code



软件工程

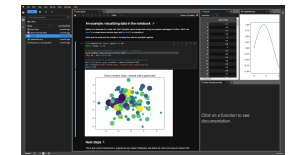
Python

数据科学

```

[In [1]: import numpy as np
[In [2]: x = np.random.randn(1, 4)
[In [3]: x
Out[3]: array([[ -1.25920396,  0.31454629,  1.38824689, -1.30699719]])
[In [4]: x = np.random.randn(1, 4)
[In [5]: x
Out[5]: array([[ -0.62501251, -0.32113616,  0.4738682 , -1.05463289]])

```



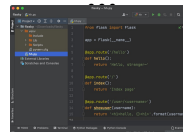
Ipython (2001年)

Jupyter

scripts (.py files)



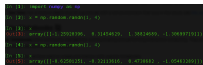
Pycharm
Visual Studio Code



软件工程
↑

Python

↓
数据科学



IPython

The screenshot shows a Jupyter Notebook with the following content:

An example: visualizing data in the notebook

Below is an example of a code cell. We'll visualize some simple data using two popular packages in Python. We'll use NumPy to create some random data, and Matplotlib to visualize it.

Note how the code and the results of running the code are bundled together.

```
[1]: from matplotlib import pyplot as plt
import numpy as np

# Generate 100 random data points along 3 dimensions
x, y, scale = np.random.randn(3, 100)
fig, ax = plt.subplots()

# Map each onto a scatterplot we'll create with Matplotlib
ax.scatter(x=x, y=y, c=scale, s=np.abs(scale)*500)
ax.set(title="Some random data, created with JupyterLab!")
plt.show()
```

Some random data, created with JupyterLab!

Next steps

This is just a short introduction to JupyterLab and Jupyter Notebooks. See below for some more ways to interact with...

iris.csv

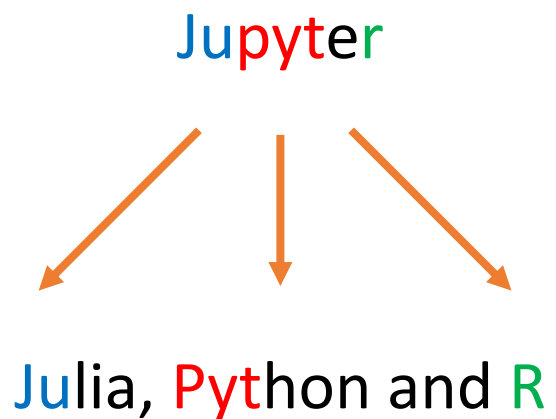
	sepal_length
1	5.1
2	4.9
3	4.7
4	4.6
5	5
6	5.4
7	4.6
8	5
9	4.4
10	4.9
11	5.4
12	4.8
13	4.8
14	4.3
15	5.8
16	5.7
17	5.4

Click on a function to see documentation.

Jupyter (2014年)

Jupyter名字的由来

Jupiter (木星)



JupyterLab界面介绍

Filter files by name

/ ... / HPA-single_cell / figure4 /

Name	Last Modified
gene_clust...	8 minutes ago
gene_clust...	4 days ago
umap_clus...	8 minutes ago

目录/文件 (command+b)

figure4-geneClustering.ipynb Python 3 (ipykernel)

```
[7]: pca.components_.shape
```

```
[7]: (60, 79)
```

```
[8]: fig, ax = plt.subplots()
x = [i+1 for i in range(60)]
y = pca.explained_variance_ratio_
ax.bar(x,y)
ax.scatter(x,y,alpha=0.8, s=7)
plt.show()
```

Notebook (写代码的地方)

Show Contextual Help

Docstring:
A scatter plot of `**y*` vs. `**x*` with varying marker size and/or color.

Parameters

```
x, y : float or array-like, shape (n, )
The data positions.
```

```
s : float or array-like, shape (n, ), optional
The marker size in points**2.
Default is ``rcParams['lines.markersize'] ** 2``.
```

```
c : array-like or list of colors or color, optional
The marker colors. Possible values:
- A scalar or sequence of n numbers to be mapped to colors using
  *cmap* and *norm*.
```

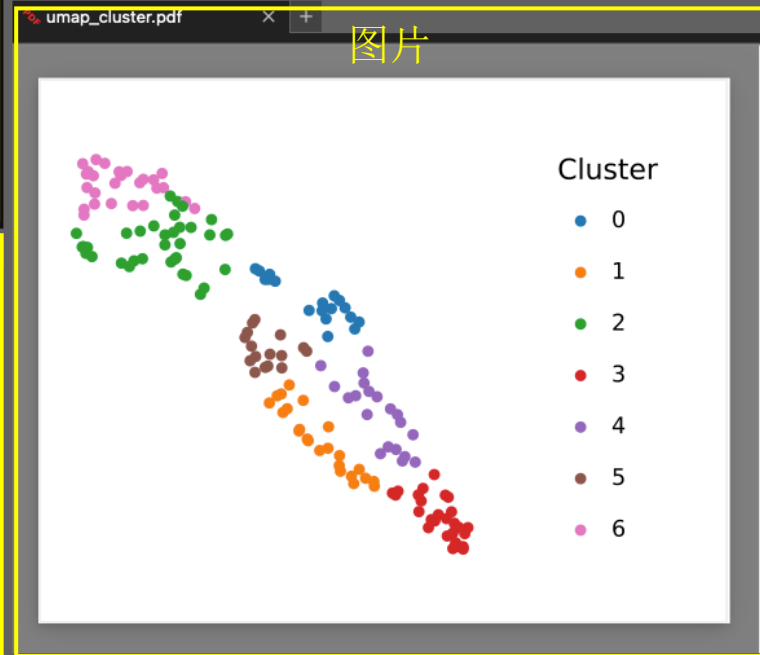
函数说明文档 (command+i)

gene_clustering.tsv 表格

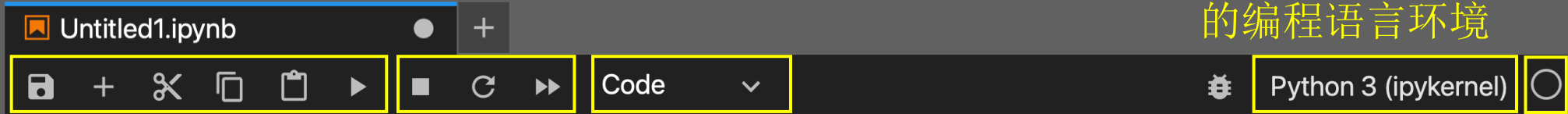
Gene	UMAP
1 ABHD2	9.60
2 ADCYAP1R1	13.29
3 ADIPOR1	9.75
4 ADIPOR2	9.8
5 ADRA1A	12.58
6 ADRA1B	13.31
7 ADRA1D	15.32
8 ADRA2A	13.33
9 ADRA2B	14.20
10 ADRA2C	12.81
11 ADRB1	12.21
12 ADRB2	10.85
13 ADRB3	16.14
14 AGTR1	12.16
15 AGTR2	16.01
16 ALDH1A1	9.48
17 ALDH1A3	10.46

Terminal 1 终端

```
42.99 12/1 2844% user,33.84% 66 97 3
Sys,7:0210 3.29, 2.63, 2.58
530* core15.87% user, 3.59% sys, 0
.53% idler 8903 80:14.50 1
72982* com.apple.IOD 19.3 01678.37 5
93.20 lcom.apple.We 88116 00438.44 2
13040*0top 1 6.6 00:33.98 2
08.99 2/1 0iod 52601 09637.27 9
18315 Wind931erter 5.5 80836.27 3
20.11 22/1 6ft Po 48.1 10701.40 6
72982* com.apple.We 25.9 01:2
394* WindowServer 100.2 80: 5:47
1* launchd 8.2 01:49.30 3
73040* top 6.8 00:12.73 2
/1 0com.apple.We 4.9 01:50.58 1
430* coreaudiod 4.3 09:27.52 2
72782* com.apple.We 4.2 00:38.35 6
0* kernel_task 3.8 36:05.51 5
394* WindowServer 2.6 80:27.06 2
389* AirPlayXPCHe 2.1 00:02.49 1
2487* Microsoft Po 2.0 10:09.51 3
385* bluetoothd 1.8 05:16.12 1
6 77* NeteaseMusic 1.5 00:39.39 2
1956* com.apple.We 1.4 00:34.76 2
620* donotdisturb 1.0 11:01.89 6
72720* VTDecoderXPC 1.0 00:26.83 3
674* Notification 0.8 06:18.07 5
```

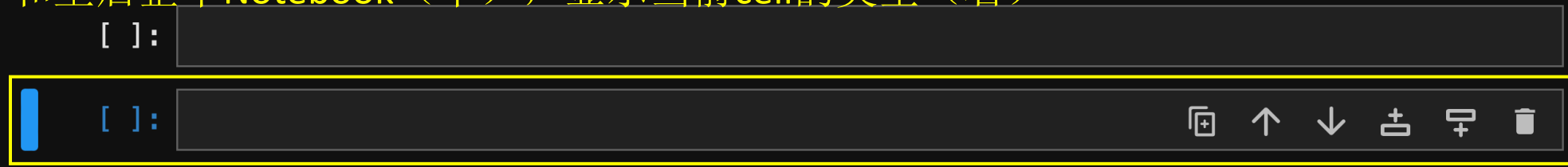


显示和选择当前Notebook的编程语言环境



对cell进行添加、复制、粘贴、运行等操作（左），中断和重启整个Notebook（中），显示当前cell的类型（右）

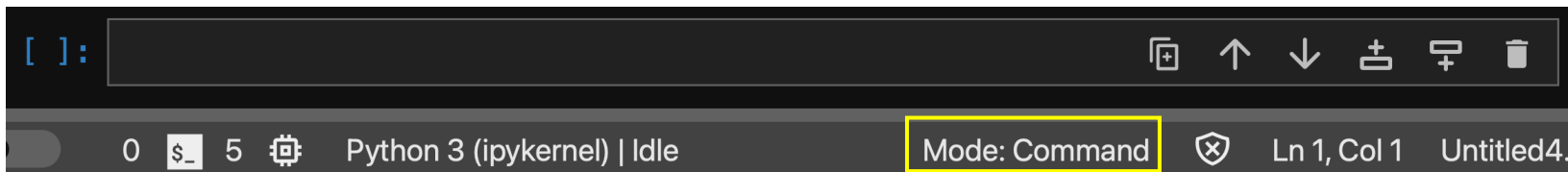
运行状态



这叫一个cell，或者说代码块，运行的结果会直接展示在下方

用Jupyter的Hello World介绍Notebook的两种操作模式

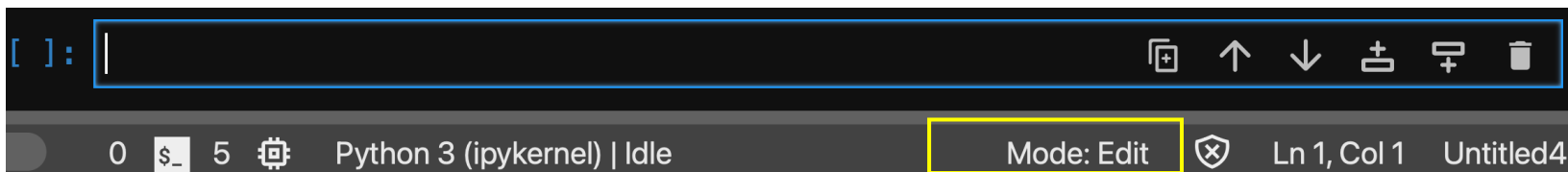
新建一个Notebook



```
[ ]: 
```

Python 3 (ipykernel) | Idle Mode: Command Ln 1, Col 1 Untitled4.

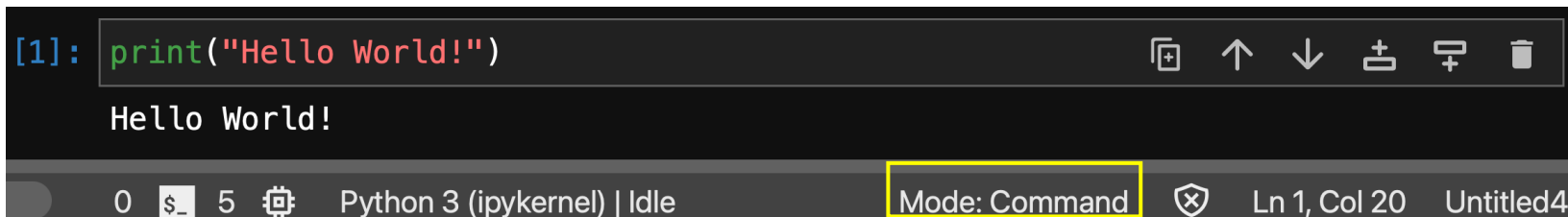
默认是**命令模式**



```
[ ]: |
```

Python 3 (ipykernel) | Idle Mode: Edit Ln 1, Col 1 Untitled4.

按下ENTER进入**编辑模式**



```
[1]: print("Hello World!")  
Hello World!
```

Python 3 (ipykernel) | Idle Mode: Command Ln 1, Col 20 Untitled4.

编辑模式下输入代码并运行



```
[ ]: print("Hello World!")  
Hello World!
```

```
[ ]: print("Hello World!")  
Hello World!
```

Python 3 (ipykernel) | Idle Mode: Command Ln 1, Col 1 Untitled4.i

命令模式下（按下ESC进入）
的输入视为各种命令，比如
这里依次键入c和v的效果是
复制并粘贴了一个cell

Notebook的两种操作模式

编辑模式（按下ENTER进入）：写代码、在一个cell内部进行编辑。

命令模式（按下ESC进入）：进入后无法写代码，而是对整个cell进行操作，比如复制、删除、改变cell类型等。

三种cell类型

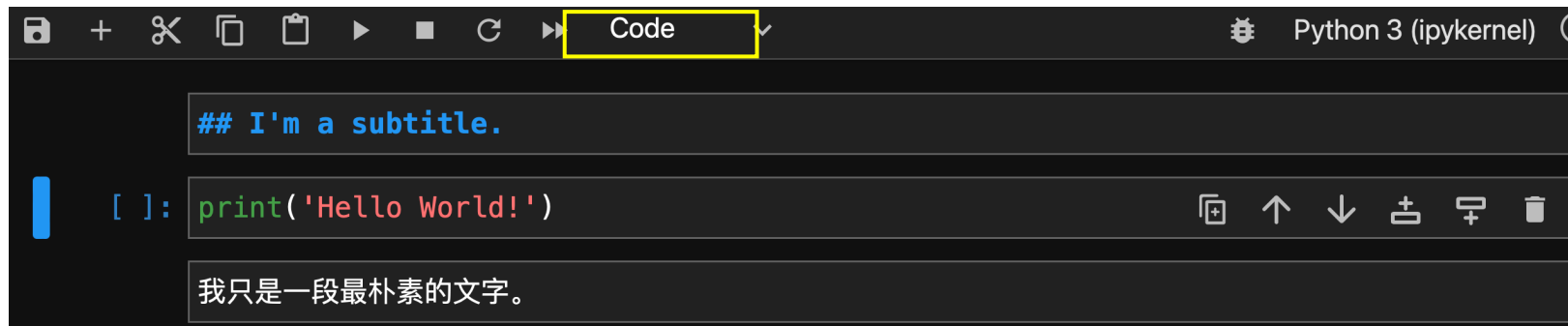


The screenshot shows a Jupyter Notebook interface with the 'Markdown' mode selected in the top toolbar. The cell contains a subtitle and a code block. The subtitle is rendered as a blue heading, and the code block is rendered as a pre-formatted text block. The output of the code block is displayed below it.

```
## I'm a subtitle.
```

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

我只是一段最朴素的文字。

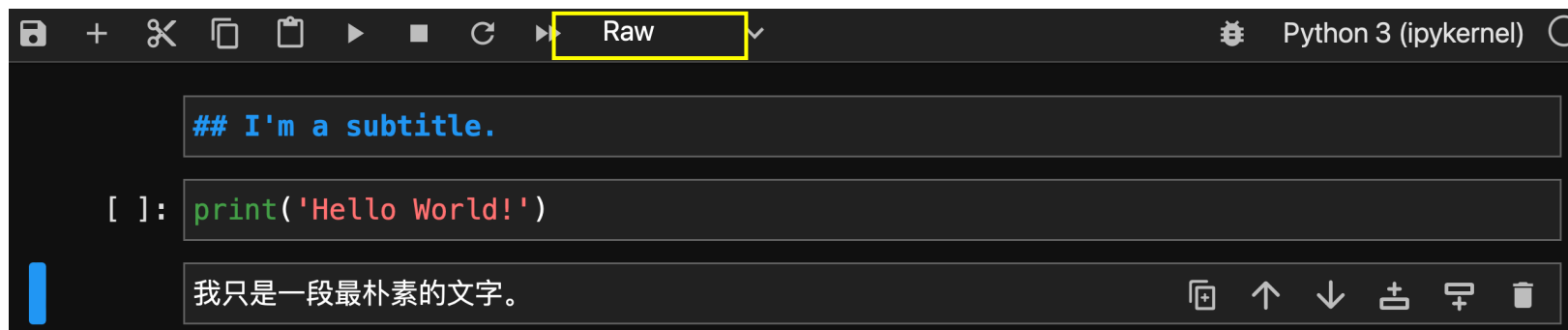


The screenshot shows a Jupyter Notebook interface with the 'Code' mode selected in the top toolbar. The cell contains a subtitle and a code block. The subtitle is rendered as a blue heading, and the code block is rendered as a pre-formatted text block. The output of the code block is displayed below it.

```
## I'm a subtitle.
```

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

我只是一段最朴素的文字。



The screenshot shows a Jupyter Notebook interface with the 'Raw' mode selected in the top toolbar. The cell contains a subtitle and a code block. The subtitle is rendered as a blue heading, and the code block is rendered as a pre-formatted text block. The output of the code block is displayed below it.

```
## I'm a subtitle.
```

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







我只是一段最朴素的文字。

三种cell类型

<code>## I'm a subtitle.</code>	Markdown (命令模式下输入m)
<code>[]: print('Hello World!')</code>	Code (命令模式下输入y)
我只是一段最朴素的文字。	Raw (命令模式下输入r)      

Run



I'm a subtitle.	以Markdown的语法显示
<code>[3]: print('Hello World!')</code> Hello World!	以当前选择的语言 (Python) 运行并将结果直接显示在下方
我只是一段最朴素的文字。	无变化      

File Edit View Run Kernel Tabs Settings Help

UNTITLED.IPYNB

Introduction to the JupyterLab

An example: visualizing data in the notebook ✨

Next steps 🏃

Other notebooks in this demo

Other sources of information in Jupyter

Untitled.ipynb

Python 3 (ipykernel)

Introduction to the JupyterLab

JupyterLab is a next-generation web-based user interface for Project Jupyter. It enables you to work with documents and activities such as Jupyter notebooks, text editors, terminals, and custom components in a flexible, integrated, and extensible manner. It is the interface that you're looking at right now.

For an overview of the JupyterLab interface, see the **JupyterLab Welcome Tour** on this page, by going to `Help -> Welcome Tour` and following the prompts.

See Also: For a more in-depth tour of JupyterLab with a full environment that runs in the cloud, see [\[the JupyterLab introduction on Binder\]](https://mybinder.org/v2/gh/jupyterlab/jupyterlab-demo/HEAD?urlpath=lab/tree/demo).

An example: visualizing data in the notebook ✨

Below is an example of a code cell. We'll visualize some simple data using two popular packages in Python. We'll use [\[NumPy\]](https://numpy.org/) to create some random data, and [\[Matplotlib\]](https://matplotlib.org) to visualize it.

Note how the code and the results of running the code are bundled together.

```
[3]: from matplotlib import pyplot as plt
import numpy as np

# Generate 100 random data points along 3 dimensions
x, y, scale = np.random.randn(3, 100)
fig, ax = plt.subplots()

# Map each onto a scatterplot we'll create with Matplotlib
ax.scatter(x=x, y=y, c=scale, s=np.abs(scale)*500)
ax.set(title="Some random data, created with JupyterLab!")
plt.show()

...
```

Next steps 🏃

This is just a short introduction to JupyterLab and Jupyter Notebooks. See below for some more ways to interact with tools in the Jupyter ecosystem, and its community.

Other notebooks in this demo

Here are some other notebooks in this demo. Each of the items below corresponds to a file or folder in the **file browser to the left**.

- `[Lorenz.ipynb]` uses Python to demonstrate interactive visualizations and computations around the [\[Lorenz system\]](https://en.wikipedia.org/wiki/Lorenz_system). It shows off basic Python functionality, including more visualizations.

Simple 0 5 Python 3 (ipykernel) | Idle Mode: Command Ln 4, Col 1 Untitled.ipynb

File Edit View Run Kernel Tabs Settings Help

UNTITLED.IPYNB

Introduction to the JupyterLab

An example: visualizing data in the notebook ✨

Next steps 🏃

Other notebooks in this demo

Other sources of information in Jupyter

Untitled.ipynb

Python 3 (ipykernel)

Introduction to the JupyterLab

JupyterLab is a next-generation web-based user interface for Project Jupyter. It enables you to work with documents and activities such as Jupyter notebooks, text editors, terminals, and custom components in a flexible, integrated, and extensible manner. It is the interface that you're looking at right now.

For an overview of the JupyterLab interface, see the **JupyterLab Welcome Tour** on this page, by going to `Help -> Welcome Tour` and following the prompts.

See Also: For a more in-depth tour of JupyterLab with a full environment that runs in the cloud, see [the JupyterLab introduction on Binder](#).

An example: visualizing data in the notebook ✨


Below is an example of a code cell. We'll visualize some simple data using two popular packages in Python. We'll use [NumPy](#) to create some random data, and [Matplotlib](#) to visualize it.

Note how the code and the results of running the code are bundled together.

```
[1]: from matplotlib import pyplot as plt
import numpy as np

# Generate 100 random data points along 3 dimensions
x, y, scale = np.random.randn(3, 100)
fig, ax = plt.subplots()

# Map each onto a scatterplot we'll create with Matplotlib
ax.scatter(x=x, y=y, c=scale, s=np.abs(scale)*500)
ax.set(title="Some random data, created with JupyterLab!")
plt.show()
```



Simple 0 5 Python 3 (ipykernel) | Idle Mode: Command Ln 1, Col 1 Untitled.ipynb

File Edit View Run Kernel Tabs Settings Help

UNTITLED.IPYNB

Introduction to the JupyterLab

An example: visualizing data in the notebook ✨

Next steps 🏃

Other notebooks in this demo

Other sources of information in Jupyter

Untitled.ipynb

```
ax.scatter(x=x, y=y, c=scale, s=np.abs(scale)*500)
ax.set(title="Some random data, created with JupyterLab!")
plt.show()
```

...

Next steps 🏃

This is just a short introduction to JupyterLab and Jupyter Notebooks. See below for some more ways to interact with tools in the Jupyter ecosystem, and its community.

Other notebooks in this demo

Here are some other notebooks in this demo. Each of the items below corresponds to a file or folder in the **file browser to the left**.

- `Lorenz.ipynb` (Lorenz.ipynb) uses Python to demonstrate interactive visualizations and computations around the [Lorenz system](https://en.wikipedia.org/wiki/Lorenz_system). It shows off basic Python functionality, including more visualizations, data structures, and scientific computing libraries.
- `sqlite.ipynb` (sqlite.ipynb) demonstrates how an in-browser sqlite kernel to run your own SQL commands from the notebook. It uses the [jupyterlite/xeus-sqlite-kernel](https://github.com/jupyterlite/xeus-sqlite-kernel).

Other sources of information in Jupyter

- **More on using JupyterLab**: See [the JupyterLab documentation](https://jupyterlab.readthedocs.io/en/stable/) for more thorough information about how to install and use JupyterLab.
- **More interactive demos**: See try.jupyter.org for more interactive demos with the Jupyter ecosystem.
- **Learn more about Jupyter**: See [the Jupyter community documentation](https://docs.jupyter.org) to learn more about the project, its community and tools, and how to get involved.
- **Join our discussions**: The [Jupyter Community Forum](https://discourse.jupyter.org) is a place where many in the Jupyter community ask questions, help one another, and discuss issues around interactive computing and our ecosystem.

We explore the Lorenz system of differential equations:

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

Let's change (σ), (β), (ρ) with ipywidgets and examine the trajectories.

Simple 0 5 Python 3 (ipykernel) | Idle Mode: Command Ln 7, Col 1 Untitled.ipynb

File Edit View Run Kernel Tabs Settings Help

UNTITLED.IPYNB

Introduction to the JupyterLab

An example: visualizing data in the notebook ✨

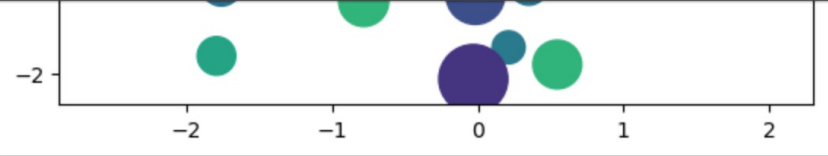
Next steps 🏃

Other notebooks in this demo

Other sources of information in Jupyter

Untitled.ipynb

Python 3 (ipykernel)



Next steps 🏃

This is just a short introduction to JupyterLab and Jupyter Notebooks. See below for some more ways to interact with tools in the Jupyter ecosystem, and its community.

Other notebooks in this demo

Here are some other notebooks in this demo. Each of the items below corresponds to a file or folder in the **file browser to the left**.

- **Lorenz.ipynb** uses Python to demonstrate interactive visualizations and computations around the [Lorenz system](#). It shows off basic Python functionality, including more visualizations, data structures, and scientific computing libraries.
- **sqlite.ipynb** demonstrates how an in-browser sqlite kernel to run your own SQL commands from the notebook. It uses the [jupyterlite/xeus-sqlite-kernel](#).

Other sources of information in Jupyter

- **More on using JupyterLab:** See [the JupyterLab documentation](#) for more thorough information about how to install and use JupyterLab.
- **More interactive demos:** See [try.jupyter.org](#) for more interactive demos with the Jupyter ecosystem.
- **Learn more about Jupyter:** See [the Jupyter community documentation](#) to learn more about the project, its community and tools, and how to get involved.
- **Join our discussions:** The [Jupyter Community Forum](#) is a place where many in the Jupyter community ask questions, help one another, and discuss issues around interactive computing and our ecosystem.

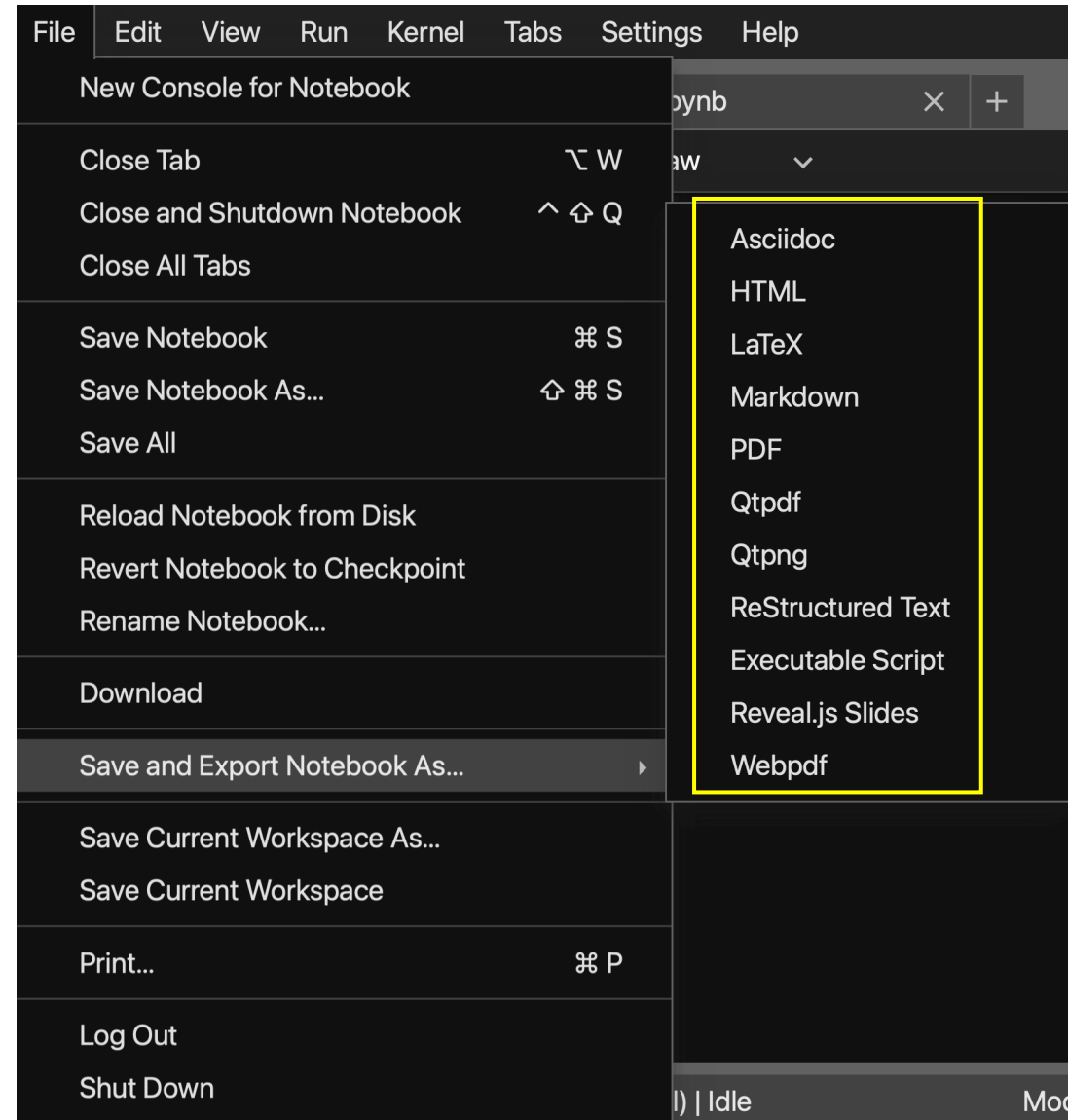
We explore the Lorenz system of differential equations:

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

Let's change (σ, β, ρ) with ipywidgets and examine the trajectories.

Simple 0 5 Python 3 (ipykernel) | Idle Mode: Command Ln 1, Col 1 Untitled.ipynb

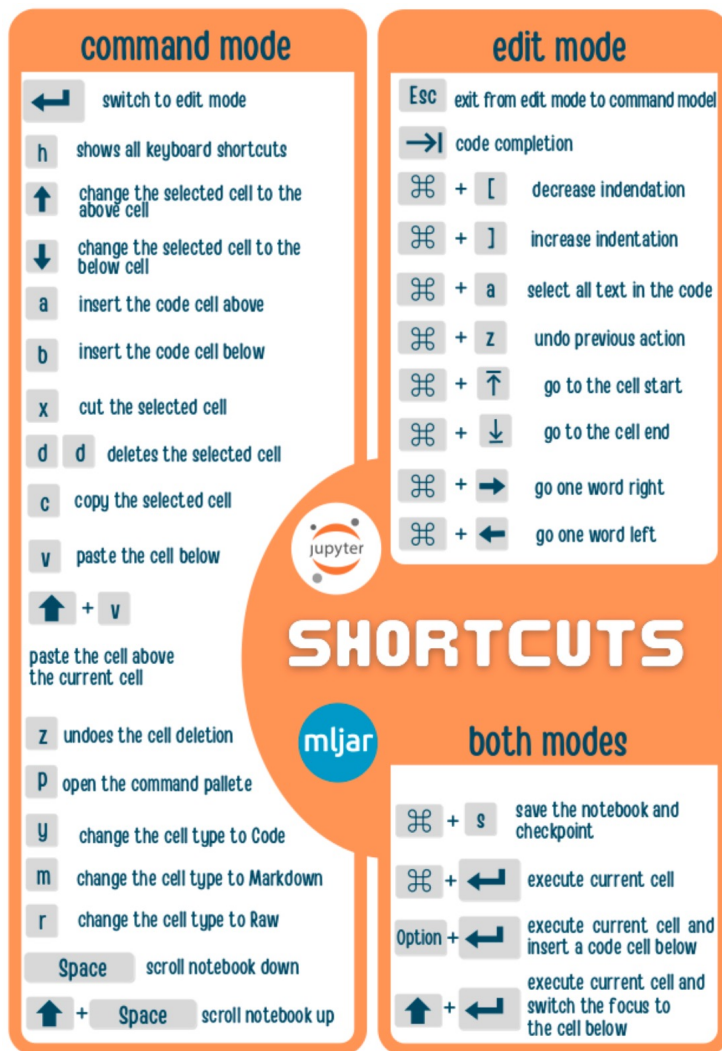
Jupyter可以导出为多种格式去分享



一些常用的快捷键

For Mac users

For Windows users



command mode

- ⌘ ← switch to edit mode
- h shows all keyboard shortcuts
- ↑ change the selected cell to the above cell
- ↓ change the selected cell to the below cell
- a insert the code cell above
- b insert the code cell below
- x cut the selected cell
- d d deletes the selected cell
- c copy the selected cell
- v paste the cell below
- ↑ + v paste the cell above the current cell
- z undoes the cell deletion
- P open the command palette
- Y change the cell type to Code
- m change the cell type to Markdown
- r change the cell type to Raw
- Space scroll notebook down
- ↑ + Space scroll notebook up

edit mode

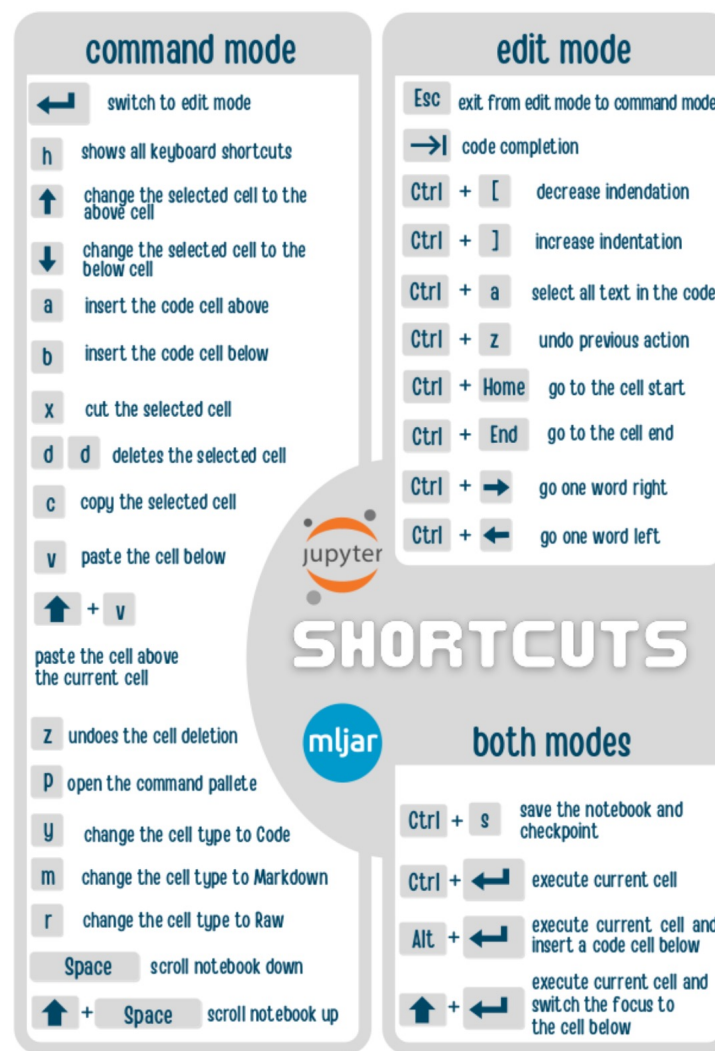
- Esc exit from edit mode to command model
- | code completion
- ⌘ + [decrease indentation
- ⌘ +] increase indentation
- ⌘ + a select all text in the code
- ⌘ + z undo previous action
- ⌘ + ↑ go to the cell start
- ⌘ + ↓ go to the cell end
- ⌘ + → go one word right
- ⌘ + ← go one word left

both modes

- ⌘ + s save the notebook and checkpoint
- ⌘ + ⌘ execute current cell
- Option + ⌘ execute current cell and insert a code cell below
- ⌘ + ⌘ execute current cell and switch the focus to the cell below

SHORTCUTS

mljar



command mode

- ⌵ switch to edit mode
- h shows all keyboard shortcuts
- ↑ change the selected cell to the above cell
- ↓ change the selected cell to the below cell
- a insert the code cell above
- b insert the code cell below
- x cut the selected cell
- d d deletes the selected cell
- c copy the selected cell
- v paste the cell below
- ↑ + v paste the cell above the current cell
- z undoes the cell deletion
- P open the command palette
- Y change the cell type to Code
- m change the cell type to Markdown
- r change the cell type to Raw
- Space scroll notebook down
- ↑ + Space scroll notebook up

edit mode

- Esc exit from edit mode to command model
- | code completion
- Ctrl + [decrease indentation
- Ctrl +] increase indentation
- Ctrl + a select all text in the code
- Ctrl + z undo previous action
- Ctrl + Home go to the cell start
- Ctrl + End go to the cell end
- Ctrl + → go one word right
- Ctrl + ← go one word left

both modes

- Ctrl + s save the notebook and checkpoint
- Ctrl + ⌘ execute current cell
- Alt + ⌘ execute current cell and insert a code cell below
- ⌘ + ⌘ execute current cell and switch the focus to the cell below

SHORTCUTS

mljar

命令模式下一些常用的快捷键跟vim的类似

Running JupyterLab remotely

The screenshot shows the Jupyter Server documentation website. The page title is "Running a public Jupyter Server". The main content area contains a "Note" box stating that the server runs locally at 127.0.0.1:8888 and is accessible from localhost. It also includes an "Important" box warning that this is not a multi-user server and that remote access requires a thorough understanding of security implications. The page also has a "Section Navigation" sidebar on the left and an "On this page" sidebar on the right.

jupyter server Users [Operators](#) [Developers](#) [Contributors](#) [Other](#) 🔍 🌙

Running a public Jupyter Server

The Jupyter Server uses a [two-process kernel architecture](#) based on [ZeroMQ](#), as well as [Tornado](#) for serving HTTP requests.

Note

By default, Jupyter Server runs locally at 127.0.0.1:8888 and is accessible only from `localhost`. You may access the server from the browser using `http://127.0.0.1:8888`.

This document describes how you can [secure a Jupyter server](#) and how to [run it on a public interface](#).

Important

This is not the multi-user server you are looking for. This document describes how you can run a public server with a single user. This should only be done by someone who wants remote access to their personal machine. Even so, doing this requires a thorough understanding of the set-ups limitations and security implications. If you allow multiple users to access a Jupyter server as it is described in this document, their commands may collide, clobber and overwrite each other.

If you want a multi-user server, the official solution is [JupyterHub](#). To use JupyterHub, you need a Unix server (typically Linux) running somewhere that is accessible to your users on a network. This may run over the public internet, but doing so introduces additional [security concerns](#).

Securing a Jupyter server

You can protect your Jupyter server with a simple single password. As of notebook 5.0 this can be done automatically. To set up a password manually you can configure the `ServerApp.password` setting in `jupyter_server_config.py`.

On this page

- Securing a Jupyter server
 - Prerequisite: A Jupyter server configuration file
 - Automatic Password setup
 - Preparing a hashed password
 - Adding hashed password to your notebook configuration file
 - Using SSL for encrypted communication
- Running a public notebook server
- Running the notebook with a customized URL prefix
- Embedding the notebook in another website
- Using a gateway server for kernel management
- Known issues

[Show Source](#)

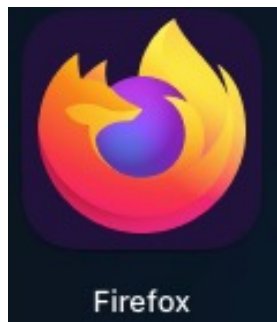
<https://jupyter-server.readthedocs.io/en/stable/operators/public-server.html>

我的JupyterLab工作流的日常

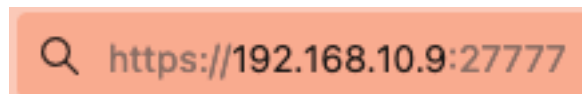
1. 打开VPN



2. 打开浏览器



3. 输入Jupyter Server运行的地址和密码
(上一张slide中你设置的)



4. 开始/继续工作

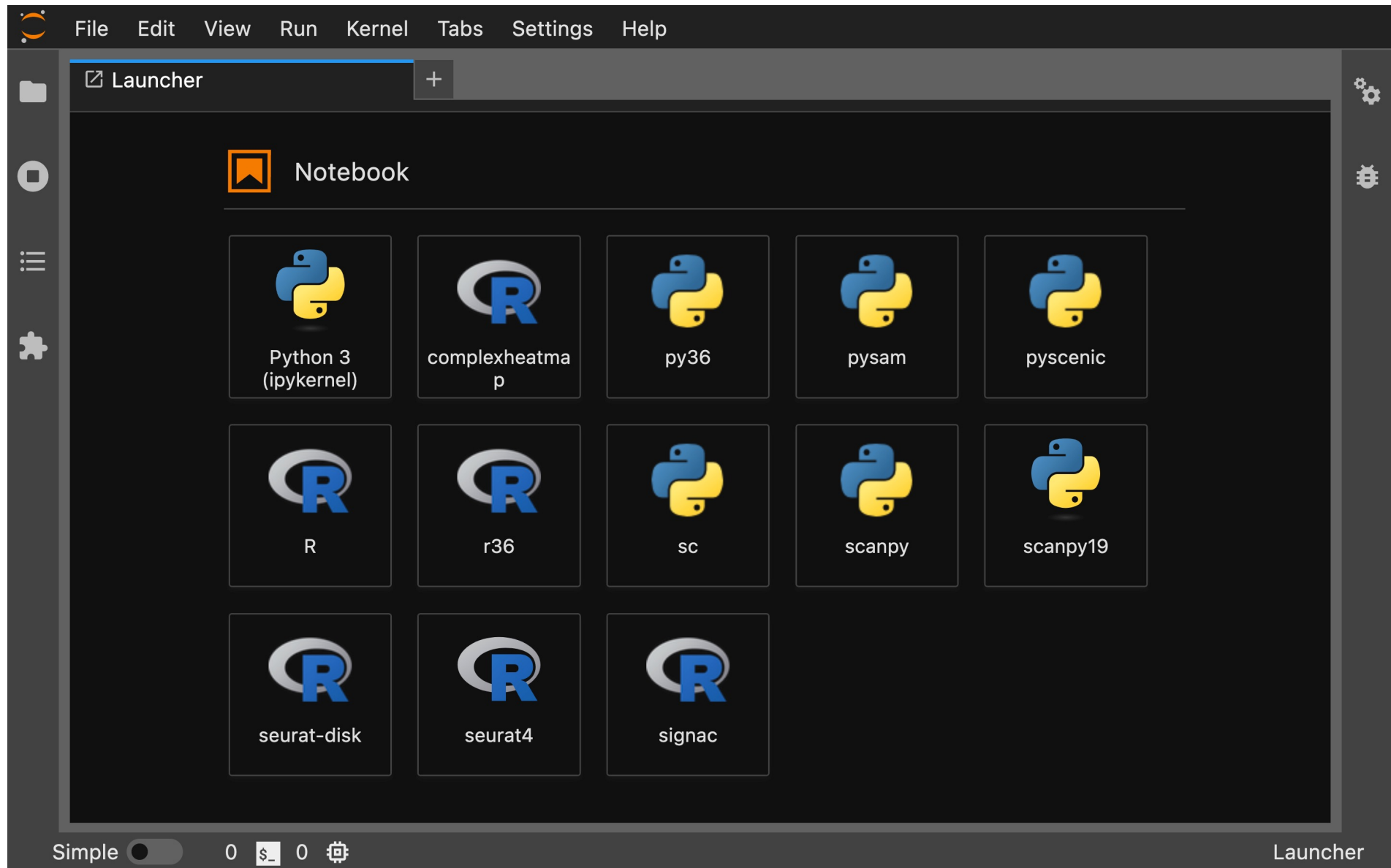
5. 保存Notebook

6. 关掉JupyterLab网页，
结束工作

JupyterLab的多编程语言支持

1. 安装**，建议给每一种语言单独建立一个conda环境。
2. Google或Bing搜索“i**kernel”，然后点进第一个查看下载和安装方法。
3. **代表编程语言的名字，比如Julia、R、Java。

我用不同的conda环境来解决软件之间的冲突



Thanks for your attention!