Help and Documentation in IPython

IPython帮助和文档

> If you read no other section in this chapter, read this one: I find the tools discussed here to be the most transformative contributions of IPython to my daily workflow.

如果本章内容让你仅挑选一节来阅读的话，请你读这一节：本节讨论的工具对于其日常工作中使用IPython有着极大的帮助。

> When a technologically-minded person is asked to help a friend, family member, or colleague with a computer problem, most of the time it's less a matter of knowing the answer as much as knowing how to quickly find an unknown answer. In data science it's the same: searchable web resources such as online documentation, mailing-list threads, and StackOverflow answers contain a wealth of information, even (especially?) if it is a topic you've found yourself searching before. Being an effective practitioner of data science is less about memorizing the tool or command you should use for every possible situation, and more about learning to effectively find the information you don't know, whether through a web search engine or another means.

当一个技术人员在计算机问题上被朋友、家人或同事请求帮助的时候，大多数情况下，直接知道答案和知道如何迅速的找到答案是一样的。在数据科学领域也是同样的情况：可以搜索到的网络资源例如在线文档、邮件列表以及StackOverflow上的问题答案都含有丰富的信息，特别是这方面的内容你之前已经自己搜索过答案的情况下。作为一个高效的数据科学人员，并不需要你能记得每一个可能场景需要用到的工具或命令，更重要的是你能在不知道答案的情况下迅速地找到信息，无论是通过搜索引擎还是其他的方法。

> One of the most useful functions of IPython/Jupyter is to shorten the gap between the user and the type of documentation and search that will help them do their work effectively. While web searches still play a role in answering complicated questions, an amazing amount of information can be found through IPython alone. Some examples of the questions IPython can help answer in a few keystrokes:
>
> - How do I call this function? What arguments and options does it have?
> - What does the source code of this Python object look like?
> - What is in this package I imported? What attributes or methods does this object have?

IPython/Jupyter中最优秀的特性就是弥合了用户与文档以及搜索它们的方式之间的鸿沟，使得用户能够提高他们的工作效率。网络搜索依旧是查找复杂问题答案不可或缺的方式，但是IPython本身就已经提供了许多相关的信息。很多的问题通过在IPython中输入几个字符就可以找到答案，例如：

- 我该如何调用这个函数？它需要怎样的参数和选项？
- 这个Python对象的源码是怎样的？
- 我载入的这个包里面有哪些内容？这个对象有哪些属性或方法？

> Here we'll discuss IPython's tools to quickly access this information, namely the `?` character to explore documentation, the `??` characters to explore source code, and the Tab key for auto-completion.

这里我们会讨论IPython中获取这些信息的方式，即，`?`符号来查看文档，`??`符号来查看源码以及使用制表符进行自动补全。

Accessing Documentation with `?`

使用`?`获取文档

> The Python language and its data science ecosystem is built with the user in mind, and one big part of that is access to documentation. Every Python object contains the reference to a string, known as a *doc string*, which in most cases will contain a concise summary of the object and how to use it. Python has a built-in `help()` function that can access this information and prints the results. For example, to see the documentation of the built-in `len` function, you can do the following:

Python语言和数据科学生态系统始终将用户需求放在重要位置，其中一大部分就是获取文档。每一个Python对象都含有一个字符串的说明，称为*文档字符串*，它是该对象的简要概括以及如何使用的信息。Python中有內建的`help()`函数能够获取这些信息并打印出来。例如，要获得内建函数`len()`的文档字符串，你可以这样做：

```
In [1]: help(len)
Help on built-in function len in module builtins:

len(...)
    len(object) -> integer

    Return the number of items of a sequence or mapping.
```

> Depending on your interpreter, this information may be displayed as inline text, or in some separate pop-up window.

取决于你的解释器，这个信息可能出现在内嵌的输出文本中，也可能出现在一个弹出的窗口中。

> Because finding help on an object is so common and useful, IPython introduces the `?` character as a shorthand for accessing this documentation and other relevant information:

因为查找一个对象的帮助是如此普遍和有用，IPython引入了`?`符号来简化`help()`內建函数的操作：

```
In [2]: len?
Type:        builtin_function_or_method
String form: <built-in function len>
Namespace:   Python builtin
Docstring:
len(object) -> integer

Return the number of items of a sequence or mapping.
```

> This notation works for just about anything, including object methods:

这种写法基本上可以应用于任何对象，包括对象的方法：

```
In [3]: L = [1, 2, 3]
In [4]: L.insert?
Type:        builtin_function_or_method
String form: <built-in method insert of list object at 0x1024b8ea8>
Docstring:   L.insert(index, object) -- insert object before index
```

> or even objects themselves, with the documentation from their type:

或者对象本身，返回的将会是这种类型的文档：

```
In [5]: L?
Type:        list
String form: [1, 2, 3]
Length:      3
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items
```

> Importantly, this will even work for functions or other objects you create yourself! Here we'll define a small function with a docstring:

更为重要的是，这个符号还能应用在你自己创建的对象和其他对象上，下面我们来定义一个很简单的带有docstring的函数：

```
In [6]: def square(a):
  ....:     """Return the square of a."""
  ....:     return a ** 2
  ....:
```

> Note that to create a docstring for our function, we simply placed a string literal in the first line. Because doc strings are usually multiple lines, by convention we used Python's triple-quote notation for multi-line strings.

从上例中我们可以看到，我们可以在函数的第一行放置一个字符串来实现docstring。因为docstring通常是多行的文本，习惯上我们会使用Python的三引号记号来代表它。

> Now we'll use the `?` mark to find this doc string:

下面我们就可以使用`?`符号来查找这个docstring：

```
In [7]: square?
Type:        function
String form: <function square at 0x103713cb0>
Definition:  square(a)
Docstring:   Return the square of a.
```

> This quick access to documentation via docstrings is one reason you should get in the habit of always adding such inline documentation to the code you write!

因为有了这么简便的查找docstring的方式，因此这也是你需要养成在每个对象中加入docstring文档习惯的原因之一。

Accessing Source Code with `??`

使用`??`获取源代码

> Because the Python language is so easily readable, another level of insight can usually be gained by reading the source code of the object you're curious about. IPython provides a shortcut to the source code with the double question mark (`??`):

因为Python非常容易阅读，所以你也可以通过阅读你感兴趣的对象的源代码来获取你需要的帮助信息。IPython提供了使用双问号`??`的方式获取对象源代码：

```
In [8]: square??
Type:        function
String form: <function square at 0x103713cb0>
Definition:  square(a)
Source:
def square(a):
    "Return the square of a"
    return a ** 2
```

> For simple functions like this, the double question-mark can give quick insight into the under-the-hood details.

对于像这样简单的函数来说，双问号语法可以快速地给你提供对象的内部详细实现机制。

> If you play with this much, you'll notice that sometimes the `??` suffix doesn't display any source code: this is generally because the object in question is not implemented in Python, but in C or some other compiled extension language. If this is the case, the `??` suffix gives the same output as the `?` suffix. You'll find this particularly with many of Python's built-in objects and types, for example `len` from above:

如果你经常使用`??`，你会发现有些情况下`??`并不能显示任何源代码：这主要是因为某些对象并不是用Python语言实现的，而是使用C或者其他一个需编译的语言实现的。如果出现这种情况，那么`??`的作用就与`?`一致。你可以在很多Python的內建对象和类型中发现这个问题，例如前面的`len()`函数：

```
In [9]: len??
Type:        builtin_function_or_method
String form: <built-in function len>
Namespace:   Python builtin
Docstring:
len(object) -> integer

Return the number of items of a sequence or mapping.
```

> Using `?` and/or `??` gives a powerful and quick interface for finding information about what any Python function or module does.

使用`?`或`??`可以很快的让你查找到Python函数或模块的文档或代码，这是很强大的一个功能。

Exploring Modules with Tab-Completion

使用Tab补全来探索模块

> IPython's other useful interface is the use of the tab key for auto-completion and exploration of the contents of objects, modules, and name-spaces. In the examples that follow, we'll use `<TAB>` to indicate when the Tab key should be pressed.

IPython的另一个有用的功能是使用制表符`tab`来进行自动补全以及对Python对象、模块和命名空间进行探索。在下面的例子中，我们将会使用`<TAB>`来表示需要敲击制表符键的地方。

Tab-completion of object contents

对象内容的Tab补全

> Every Python object has various attributes and methods associated with it. Like with the `help` function discussed before, Python has a built-in `dir` function that returns a list of these, but the tab-completion interface is much easier to use in practice. To see a list of all available attributes of an object, you can type the name of the object followed by a period ("`.`") character and the Tab key:

所有的Python对象都有着自己的属性和方法。就像Python有着內建的`help`函数一样，Python也有一个內建的`dir`函数，会列示出对象的属性和方法的列表，但是在IPython中，使用tab补全会更加简单。查看一个对象所有可用的属性和方法，你需要输入对象的名称和后面的一个点（`.`），然后点击Tab键：

```
In [10]: L.<TAB>
L.append   L.copy     L.extend   L.insert   L.remove   L.sort
L.clear    L.count    L.index    L.pop      L.reverse
```

> To narrow-down the list, you can type the first character or several characters of the name, and the Tab key will find the matching attributes and methods:

如果希望缩小列表范围，你可以输入属性或方法的头几个字母，自动补全会找到能匹配这些字符的属性和方法：

```
In [10]: L.c<TAB>
L.clear  L.copy   L.count

In [10]: L.co<TAB>
L.copy   L.count
```

> If there is only a single option, pressing the Tab key will complete the line for you. For example, the following will instantly be replaced with `L.count`:

如果能匹配的属性或方法只有一个，那么键入Tab的时候，IPython会自动将整个属性或方法的名称补充完整。例如，下面的输入会最终产生`L.count`：

```
In [10]: L.cou<TAB>
```

> Though Python has no strictly-enforced distinction between public/external attributes and private/internal attributes, by convention a preceding underscore is used to denote such methods. For clarity, these private methods and special methods are omitted from the list by default, but it's possible to list them by explicitly typing the underscore:

虽然Python并没有明确强制定义公有的和私有的属性和方法，但是习惯上，如果属性或方法名称是以下划线开头的话，就被认为是私有的。为了是列表清晰，默认的情况下，tab补全列表会忽略下划线开头的属性和方法，但如果你需要显示它们，你可以在点后面明确输入一个下划线来展示：

```
In [10]: L._<TAB>
L.__add__          L.__gt__           L.__reduce__
L.__class__        L.__hash__         L.__reduce_ex__
```

> For brevity, we've only shown the first couple lines of the output. Most of these are Python's special double-underscore methods (often nicknamed "dunder" methods).

为了简洁起见，我们这里只展示了前面两行内容。大部分这种命名都属于Python特殊的双下划线方法。

Tab completion when importing

当载入模块是使用tab补全

> Tab completion is also useful when importing objects from packages. Here we'll use it to find all possible imports in the `itertools` package that start with `co`:

Tab补全也同样适用于载入模块的时候。下面我们试着从`itertools`包中查找所有以`co`开头的内容：

```
In [10]: from itertools import co<TAB>
combinations                  compress
combinations_with_replacement count
```

> Similarly, you can use tab-completion to see which imports are available on your system (this will change depending on which third-party scripts and modules are visible to your Python session):

同样的，你也可以使用tab补全方式来查看系统中所有可以被载入的模块（根据你当前Python进程的上下文不同，第三方包和模块的可见性也会不同，因此列示的内容也会有差别）：

```
In [10]: import <TAB>
Display all 399 possibilities? (y or n)
Crypto              dis                 py_compile
Cython              distutils           pyclbr
...                 ...                 ...
difflib             pwd                 zmq

In [10]: import h<TAB>
hashlib             hmac                http
heapq               html                husl
```

> (Note that for brevity, I did not print here all 399 importable packages and modules on my system.)

（当然为了简洁起见，这里肯定没有列出所有399个可用的包和模块出来）

Beyond tab completion: wildcard matching

Tab补全进阶：通配符匹配

> Tab completion is useful if you know the first few characters of the object or attribute you're looking for, but is little help if you'd like to match characters at the middle or end of the word. For this use-case, IPython provides a means of wildcard matching for names using the `*` character.

Tab补全对于你知道对象或属性的头几个字母的情况下非常有效，但是如果你只记得中间或末尾处的字符时，tab补全就无法发挥了。对于这种情况，IPython提供了一种使用通配符`*`来匹配内容的方法。

> For example, we can use this to list every object in the namespace that ends with `Warning`:

例如，我们可以使用它列出任何末尾为`Warning`的对象：

```
In [10]: *Warning?
BytesWarning                  RuntimeWarning
DeprecationWarning            SyntaxWarning
FutureWarning                 UnicodeWarning
ImportWarning                 UserWarning
PendingDeprecationWarning     Warning
ResourceWarning
```

> Notice that the `*` character matches any string, including the empty string.

这里的`*`号能匹配任何字符串，包括空字符串。

> Similarly, suppose we are looking for a string method that contains the word `find` somewhere in its name. We can search for it this way:

类似的，如果我们希望找到所有名称中含有`find`字符串的对象内容，我们可以这样做：

```
In [10]: str.*find*?
str.find
str.rfind
```

> I find this type of flexible wildcard search can be very useful for finding a particular command when getting to know a new package or reacquainting myself with a familiar one.

作者发现这种通配符的方式对于在一个新的包中找到你想要的内容，或者你忘记了一个熟悉的包中的内容是特别有效。