1 IPython: за пределами обычного Python

Меня часто спрашивают, какой из множества вариантов среды разработки для Python я использую в своей работе. Мой ответ иногда удивляет спрашивающих: моя излюбленная среда представляет собой оболочку IPython (http://ipython.org/) плюс текстовый редактор (в моем случае редактор Emacs или Atom, в зависимости от настроения). IPython (сокращение от *«интерактивный Python»*) был основан в 2001 году Фернандо Пересом в качестве продвинутого интерпретатора Python и с тех пор вырос в проект, призванный обеспечить, по словам Переса, «утилиты для всего жизненного цикла исследовательских расчетов». Если язык Python — механизм решения нашей задачи в области науки о данных, то оболочку IPython можно рассматривать как интерактивную панель управления.

Оболочка IPython является полезным интерактивным интерфейсом для языка Python и имеет несколько удобных синтаксических дополнений к нему. Большинство из них мы рассмотрим. Кроме этого, оболочка IPython тесно связана с проектом Jupiter (http://jupyter.org/), предоставляющим своеобразный блокнот (текстовый редактор) для браузера, удобный для разработки, совместной работы и использования ресурсов, а также для публикации научных результатов. Блокнот оболочки IPython, по сути, частный случай более общей структуры блокнота Jupiter, включающего блокноты для Julia, R и других языков программирования. Не стоит далеко ходить за примером, показывающим удобства формата блокнота, им служит страница, которую вы сейчас читаете: вся рукопись данной книги была составлена из набора блокнотов IPython.

Оболочка IPython позволяет эффективно использовать язык Python для интерактивных научных вычислений, требующих обработки большого количества данных. В этой главе мы рассмотрим возможности оболочки IPython, полезные

при исследовании данных. Сосредоточим свое внимание на тех предоставляемых IPython синтаксических возможностях, которые выходят за пределы стандартных возможностей языка Python. Немного углубимся в «магические» команды, позволяющие ускорить выполнение стандартных задач при создании и использовании предназначенного для исследования данных кода. И наконец, затронем возможности блокнота, полезные для понимания смысла данных и совместного использования результатов.

Командная строка или блокнот?

Существует два основных способа использования оболочки IPython: командная строка IPython и блокнот IPython. Большая часть изложенного в этой главе материала относится к обоим, а в примерах будет применяться более удобный в конкретном случае вариант. Я буду отмечать немногие разделы, относящиеся только к одному из них. Прежде чем начать, вкратце расскажу, как запускать командную оболочку IPython и блокнот IPython.

Запуск командной оболочки IPython

Данная глава, как и большая часть книги, не предназначена для пассивного чтения. Я рекомендую вам экспериментировать с описываемыми инструментами и синтаксисом: формируемая при этом мышечная память принесет намного больше пользы, чем простое чтение. Начнем с запуска интерпретатора оболочки IPython путем ввода команды `IPython` в командной строке. Если же вы установили один из таких дистрибутивов, как Anaconda или EPD, вероятно, у вас есть запускающий модуль для вашей операционной системы (мы обсудим это подробнее в разделе «Справка и документация в оболочке Python» данной главы).

После этого вы должны увидеть приглашение к вводу:

```
IPython 4.0.1 – An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
Help      -> Python's own help system.
Object?   -> Details about 'object', use 'object??' for extra details.
In [1]:
```

Далее вы можете активно следить за происходящим в книге.

Запуск блокнота Jupiter

Блокнот Jupiter — браузерный графический интерфейс для командной оболочки IPython и богатый набор основанных на нем возможностей динамической

визуализации. Помимо выполнения операторов Python/IPython, блокнот позволяет пользователю вставлять форматированный текст, статические и динамические визуализации, математические уравнения, виджеты JavaScript и многое другое. Более того, эти документы можно сохранять так, что другие люди смогут открывать и выполнять их в своих системах.

Хотя просмотр и редактирование блокнота IPython осуществляется через окно браузера, он должен подключаться к запущенному процессу Python для выполнения кода. Для запуска этого процесса (называемого *ядром* (kernel)) выполните следующую команду в командной строке вашей операционной системы:

```
$ jupyter notebook
```

Эта команда запустит локальный веб-сервер, видимый браузеру. Она сразу же начнет журналировать выполняемые действия. Журнал будет выглядеть следующим образом:

```
$ jupyter notebook
[NotebookApp] Serving notebooks from local directory: /Users/jakevdp/…
[NotebookApp] 0 active kernels
[NotebookApp] The IPython Notebook is running at: http://localhost:8888/
[NotebookApp] Use Control-C to stop this server and shut down all kernels…
```

После выполнения этой команды ваш браузер по умолчанию должен автоматически запуститься и перейти по указанному локальному URL; точный адрес зависит от вашей системы. Если браузер не открывается автоматически, можете запустить его вручную и перейти по указанному адресу (в данном примере http://localhost:8888/).

Справка и документация в оболочке IPython

Если вы не читали остальных разделов в данной главе, прочитайте хотя бы этот. Обсуждаемые здесь утилиты внесли наибольший (из IPython) вклад в мой ежедневный процесс разработки.

Когда человека с техническим складом ума просят помочь другу, родственнику или коллеге решить проблему с компьютером, чаще всего речь идет об умении быстро найти неизвестное решение. В науке о данных все точно так же: допускающие поиск веб-ресурсы, такие как онлайн-документация, дискуссии в почтовых рассылках и ответы на сайте Stack Overflow, содержат массу информации, даже (особенно?) если речь идет о теме, информацию по которой вы уже искали. Уметь эффективно исследовать данные означает скорее не запоминание утилит или команд, которые нужно использовать в каждой из возможных ситуаций, а знание того, как эффек-

тивно искать неизвестную пока информацию: посредством ли поиска в Интернете или с помощью других средств.

Одна из самых полезных возможностей IPython/Jupiter заключается в сокращении разрыва между пользователями и типом документации и поиска, что должно помочь им эффективнее выполнять свою работу. Хотя поиск в Интернете все еще играет важную роль в ответе на сложные вопросы, большое количество информации можно найти, используя саму оболочку IPython. Вот несколько примеров вопросов, на которые IPython может помочь ответить буквально с помощью нескольких нажатий клавиш.

- Как вызвать эту функцию? Какие аргументы и параметры есть у нее?
- Как выглядит исходный код этого объекта Python?
- Что имеется в импортированном мной пакете? Какие атрибуты или методы есть у этого объекта?

Мы обсудим инструменты IPython для быстрого доступа к этой информации, а именно символ `?` для просмотра документации, символы `??` для просмотра исходного кода и клавишу Tab для автодополнения.

Доступ к документации с помощью символа ?

Язык программирования Python и его экосистема для исследования данных являются клиентоориентированными, и в значительной степени это проявляется в доступе к документации. Каждый объект Python содержит ссылку на строку, именуемую *docstring* (сокращение от documentation string — «строка документации»), которая в большинстве случаев будет содержать краткое описание объекта и способ его использования. В языке Python имеется встроенная функция `help()`, позволяющая обращаться к этой информации и выводить результат. Например, чтобы посмотреть документацию по встроенной функции `len`, можно сделать следующее:

```
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[1].
```

В зависимости от интерпретатора информация будет отображена в виде встраиваемого текста или в отдельном всплывающем окне.

[1] Возвращает количество элементов в последовательности или словаре. — *Здесь и далее примеч. пер.*

Поскольку поиск справочной информации по объекту — очень распространенное и удобное действие, оболочка IPython предоставляет символ ? для быстрого доступа к документации и другой соответствующей информации:

```
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[1].
```

Данная нотация подходит практически для чего угодно, включая методы объектов:

```
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[2]
```

или даже сами объекты с документацией по их типу:

```
In [5]: L?
Type:        list
String form: [1, 2, 3]
Length:      3
Docstring:
list() -> new empty list[3]
list(iterable) -> new list initialized from iterable's items[4]
```

Это будет работать даже для созданных пользователем функций и других объектов! В следующем фрагменте кода мы опишем маленькую функцию с docstring:

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

Обратите внимание, что при создании docstring для нашей функции мы просто вставили в первую строку строковый литерал. Поскольку docstring обычно занимает несколько строк, в соответствии с условными соглашениями мы использовали для многострочных docstring нотацию языка Python с тройными кавычками.

---

[1] Возвращает количество элементов в последовательности или словаре.

[2] Вставляет object перед index.

[3] Новый пустой список.

[4] Новый список, инициализированный элементами списка iterable.

Теперь воспользуемся знаком ? для нахождения этой docstring:

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

Быстрый доступ к документации через элементы docstring — одна из причин, по которым желательно приучить себя добавлять подобную встроенную документацию в создаваемый код!

Доступ к исходному коду с помощью символов ??

Поскольку текст на языке Python читается очень легко, чтение исходного кода интересующего вас объекта может обеспечить более глубокое его понимание. Оболочка IPython предоставляет сокращенную форму обращения к исходному коду — двойной знак вопроса (??):

```
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
```

Для подобных простых функций двойной знак вопроса позволяет быстро проникнуть в особенности внутренней реализации.

Поэкспериментировав с ним немного, вы можете обратить внимание, что иногда добавление в конце ?? не приводит к отображению никакого исходного кода: обычно это происходит потому, что объект, о котором идет речь, реализован не на языке Python, а на C или каком-либо другом транслируемом языке расширений. В подобном случае добавление ?? приводит к такому же результату, что и добавление ?. Вы столкнетесь с этим в отношении многих встроенных объектов и типов Python, например для упомянутой выше функции len:

```
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[1].
```

[1] Возвращает количество элементов в последовательности или словаре.

Использование ? и/или ?? — простой способ для быстрого поиска информации о работе любой функции или модуля языка Python.

Просмотр содержимого модулей с помощью Tab-автодополнения

Другой удобный интерфейс оболочки IPython — использование клавиши Tab для автодополнения и просмотра содержимого объектов, модулей и пространств имен. В следующих примерах мы будем применять обозначение <TAB> там, где необходимо нажать клавишу Tab.

Автодополнение названий содержимого объектов с помощью клавиши Tab

С каждым объектом Python связано множество различных атрибутов и методов. Аналогично обсуждавшейся выше функции help в языке Python есть встроенная функция dir, возвращающая их список, но на практике использовать интерфейс Tab-автодополнения гораздо удобнее. Чтобы просмотреть список всех доступных атрибутов объекта, необходимо набрать имя объекта с последующим символом точки (.) и нажать клавишу Tab:

```
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
```

Чтобы сократить этот список, можно набрать первый символ или несколько символов нужного имени и нажать клавишу Tab, после чего будут отображены соответствующие атрибуты и методы:

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

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

Если имеется только один вариант, нажатие клавиши Tab приведет к автодополнению строки. Например, следующее будет немедленно заменено на L.count:

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

Хотя в языке Python отсутствует четкое разграничение между открытыми/внешними и закрытыми/внутренними атрибутами, по соглашениям, для обозначения подобных методов принято использовать знак подчеркивания. Для ясности эти закрытые методы, а также специальные методы по умолчанию исключаются из списка, но можно вывести их список, набрав знак подчеркивания:

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

Для краткости я показал только несколько первых строк вывода. Большинство этих методов — методы специального назначения языка Python, отмеченные двойным подчеркиванием в названии (на сленге называемые dunder[1]-методами).

Автодополнение с помощью клавиши Tab во время импорта

Tab-автодополнение удобно также при импорте объектов из пакетов. Воспользуемся им для поиска всех возможных вариантов импорта в пакете `itertools`, начинающихся с `co`:

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

Аналогично можно использовать Tab-автодополнение для просмотра доступных в системе вариантов импорта (полученное вами может отличаться от нижеприведенного листинга в зависимости от того, какие сторонние сценарии и модули являются видимыми данному сеансу Python):

```
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
```

Отмечу, что для краткости я не привожу здесь все 399 пакетов и модулей, доступных в моей системе для импорта.

Помимо автодополнения табуляцией, подбор по джокерному символу

Автодополнение табуляцией удобно в тех случаях, когда вам известны первые несколько символов искомого объекта или атрибута. Однако оно малопригодно, когда необходимо найти соответствие по символам, находящимся в середине или конце

[1] Игра слов: одновременно сокращение от double underscore — «двойное подчеркивание» и dunderhead — «тупица», «болван».

слова. На этот случай оболочка IPython позволяет искать соответствие названий по джокерному символу *.

Например, можно использовать следующий код для вывода списка всех объектов в пространстве имен, заканчивающихся словом `Warning`:

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

Обратите внимание, что символ * соответствует любой строке, включая пустую.

Аналогично предположим, что мы ищем строковый метод, содержащий где-то в названии слово `find`. Отыскать его можно следующим образом:

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

Я обнаружил, что подобный гибкий поиск с помощью джокерных символов очень удобен для поиска нужной команды при знакомстве с новым пакетом или обращении после перерыва к уже знакомому.

Сочетания горячих клавиш в командной оболочке IPython

Вероятно, все, кто проводит время за компьютером, используют в своей работе сочетания горячих клавиш. Наиболее известные — Cmd+C и Cmd+V (или Ctrl+C и Ctrl+V), применяемые для копирования и вставки в различных программах и системах. Опытные пользователи выбирают популярные текстовые редакторы, такие как Emacs, Vim и другие, позволяющие выполнять множество операций посредством замысловатых сочетаний клавиш.

В командной оболочке IPython также имеются сочетания горячих клавиш для быстрой навигации при наборе команд. Эти сочетания горячих клавиш предоставляются не самой оболочкой IPython, а через ее зависимости от библиотеки GNU Readline: таким образом определенные сочетания горячих клавиш могут различаться в зависимости от конфигурации вашей системы. Хотя некоторые из них работают в блокноте для браузера, данный раздел в основном касается сочетаний горячих клавиш именно в командной оболочке IPython.