Spider Book in Python

华聪

前言

> 总觉得自己不是一个会说话的人，偏偏自己又有很多话要说，那就写下来吧！
>
> ——华聪

0.1 更新说明

由于时间有限，不可能一下就把整本书写完。所以，就放出这么一个栏目，每次把更新的内容和时间都列出来。

表 1: 更新说明

时间	章节	内容
2018-11-25 14:00	1、2	Python、IDE 和 Python 简介
2018-11-29 14:00	3	Python 基本语法

每次更新，可能会对原来的内容进行小幅修改，但是不一一列举了。

0.2 写在前面

写这本书，一方面，是想教会读者如何用 Python 简单地写一个入门级别的爬虫，另一方面，是想对自己学过的知识做个总结，也希望自己能够温故而知新，学到一些新的知识。鉴于笔者的水平，本书中难免会有些错误，希望能通过自己动手实践来发现他们，而不是一味地相信他们。

这本书，面向有一定其它面向对象语言（比如 Java）编程基础，以及有一定前端基础的同学。限于篇幅，书中讲述，以方法为主，不会详细地讲述库怎么用，最多只会介绍有哪些可用的库或者框架。鉴于笔者的水平，本书中难免会有些错误，希望能通过自己动手实践来发现它们，而不是一味地相信它们。

配合百度食用风味更佳哦。

目录

Chapter 1

Python 及其编辑工具的下载安装

> 工欲善其事，必先利其器。
>
> —— 《论语 · 卫灵公》

1.1 Python

Python 的安装有两种选择。

第一，可以去 Python 的官网[1]下载，值得一提的是，Python 有两类主流版本——2.7 和 3.x (x 表示某个数字)。由于 3.x 的版本已经发行好多年了，而且 2.7 版本有好多第三方的库不支持，所以推荐去下载较新的版本。**下载符合自己操作系统的版本。另外需要特别注意，3.x 和 2.7 的语法互不兼容。**

第二，可以去 Anaconda 的官网[2]下载 python 的发行版 Anaconda。Anaconda 是 Python 一个较常用的发行版，里面包括很多科学计算的库。之后要用到的一些库，比如 lxml，Anaconda 已经自带了。再比如一个比较好用的框架 Scrapy，用 Anaconda 的包管理器只需一行命令就可安装，如果用 Python 自带的包管理器 pip，会比较麻烦。**下载符合自己操作系统的版本。**下载完之后，还需要修改一下源[3] ，因为默认的源是国外的，下载包比较慢。另外，Anaconda 还包括一些好用的工具，比如 Spider、jupyter notebook、Ipython 等。这里强烈建议下载 Anaconda。

无论选哪一种，请参考网上的安装教程，完成 Python 的安装。包括环境变量的配置。否则，你将通过绝对路径使用访问 Python，而不是在终端里直接输入 Python 来使用它。

1.2 编辑工具

Python 的编辑工具很多，小到 Windows 的记事本、notepad++ 等，Linux 的 vim、emacs 等，大到“重剑无锋，大巧不工”的 Pycharm，Spyder 之类的集成开发环境 (Integrated Development Environment ,IDE)。可以根据自己的自己的喜好，以及需求，以及收费情况来决定。

[1] Python 官网：https://www.python.org
[2] Anaconda 官网：https://www.anaconda.com
[3] 修改源的教程：https://jingyan.baidu.com/article/1876c8527be1c3890a137645.html

当然，对于新人，还是稍微介绍一下两款 IDE——Pycharm 和 Spyder。

1.2.1 Pycharm

用过 Jetbrain 家 IDE 的家伙们都知道它们家的 IDE 有多好用，补全，重构等。Pycharm 分免费的社区版 (community) 和收费的专业版 (professional)，区别在于后者对众多常用的 Python web 框架提供了良好的支持，以及对 html，js，和 css 等 web 相关文件的支持（写爬虫主要是语法高亮，还有数据库可能比较有用）。

当然 Pycharm 也有它的缺点，打开时，会索引很久（找包的路径，方便补全），尤其是电脑比较差的，此时 cpu 占用也很高。另外，Pycharm 调试没有 Spyder 方便。

1.2.2 Spyder

优点，打开速度比 Pycharm 快。调试方便，类似于 MATLAB，可以直接查看变量的数值和类型，运行完后也存在。缺点就是，补全与 Pycharm 无法媲美。

本章小结

本章简单介绍了 Python 和其编辑工具，由于工具很多，没有一一介绍，仅介绍了最好用的两款，对安装还有不懂的同学可以百度一下哦

Chapter 2

Python 概述

> 千里之行, 始于足下。
>
> ——《老子》

2.1 Python 的自白

大家好，我叫 Python，我是一个面向对象的语言，和大家特别好相处，大家都亲切地叫我 PY。图 2.1是我的自拍。

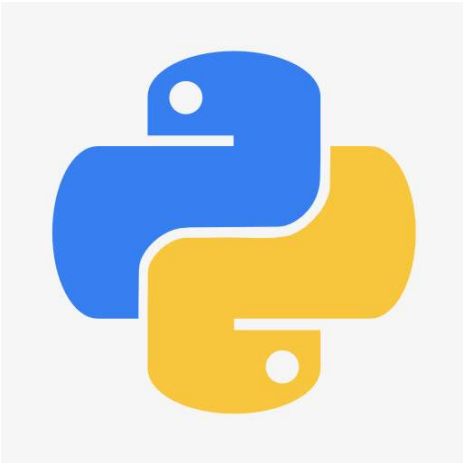

图 2.1: Python

帅吧？（读者：dei!）hhhhhhhhhh 我也是这么认为的，嘻嘻。

emmm... 要凉，一紧（Gao）张（Xing）我都忘记要说什么了。且容我看一下台词！唉，台词呢？找到啦！铛铛铛铛！图 2.2闪亮登场！

有点难懂对吧？我来“解释”一下吧。

```
hc@hc-Lenovo-B50-45:~/Documents/spider book$ python
Python 3.7.0 (default, Jun 28 2018, 13:15:42)
[GCC 7.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

图 2.2: Zen Of Python

友情提醒

关于 The zen of Python，网上有很多解释，仅供参考。一千个 Pythoner 就有一千零一个对 The zen of Python 解释，（还有一个是译文/原文），每个人，相同的人不同的时间，使用不同的 Python 版本，对 The zen of Python 可能有不同的理解，https://blog.csdn.net/gzlaiyonghao/article/details/2151918 这个博客里有译文。下面仅介绍其中几点个人理解。

我一个很明显的特点就是代码美观，不像 Java、C、C++ 等语言，通过丑陋的花括号来规定作用域，我采用了**缩进（通常是 4 个空格）**来规定作用域的范围，因此看上去十分美观。

我的代码十分简洁明了，没有多余的分号。一般能用一句话实现功能的，就不会用一个循环去实现（这样循环就多了，代码的层次、逻辑，就不再那么清晰）。比如，我会用列表（集合、字典）表达式来代替循环。再比如，我有垃圾回收机制，不用的东西，不用自行 free[1]。

```
# list comprehension l = [x**2 for x in range(1,10)]

# print print(l)

# for loop l=[] for x in range(1,10): l.append(x**2)
```

[1]这方面博客很多，但是初学者不需要钻太深，只要知道有垃圾回收即可

```
# print print(l)
```

代码清单 2-1.py

友情提醒

这章里暂时不用完全理解代码的意思! 但是如果你有精力，理解代码对你自己日后写 Python 代码也有好处

可以从图 2.3看出代码的功能是一样的，把所有的 1-9 的整数的平方的列表输出到控制台。另外，还可以看出，Python 虽然是一门面向对象的语言（如 Java，c++），但是也支持面向过程（如 C）。因此，在编程的时候是很自由的。你不用编写主类，再编写一个主方法，你甚至不用写一个主函数，只要你编写一段能被 Python 解释器（就是图 2.3中敲的 python，类似于 C 里面的 gcc 和 Java 里的 javac）“解释”的代码，就能得到结果——或者正确，或者抛出异常。

```
hc@hc-Lenovo-B50-45:~/Documents/spider book/code$ python 2-1.py
[1, 4, 9, 16, 25, 36, 49, 64, 81]
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

图 2.3: 2-1.py 运行结果

友情提醒

在代码量大或者需要团队协作的时候，还是建议写上主函数，主函数怎么写优雅，会在下章讲述

这里还要“解释”一下刚才为什么要反复特别强调“解释”这个词。和 C、Java 等编译型语言不一样，我——Python 不需要编译，而是由 Python 解释器，一步一步解释运行。这样子，以牺牲少许性能为代价，节约了漫长的编译时间（如果你的电脑不是很好，你又编译过大型的项目或者软件，你就知道编译有多么痛苦，Linux 有些发行版比如 Gentoo 等就是编译安装的，特别慢，编译一个 gcc，要好几个小时。而且编译语言，常常编译了好久没出错，到运行的时候再报错，然后又要 debug，又要再 compile 一次，很花时间），所以总得来说，我——Python，还是挺节约时间的。另外，由于解释器的交互作用（输入一行 Python 代码，返回一个结果），我经常被写这本书的家伙当作计算器用，无论是进制转换，还是快速幂，还是矩阵乘法，抑或是简单的加减乘除，都有极其简单的调用方法，windows 自带的 calc 里有的，我都有，calc 里没有的，我也基本都有，哪怕真的没有，我也可以造轮子般造出来。

说完了我的主要优点，再说一下我的其它特点。

首先，我很慢，这主要归功于我是一门解释型的语言。也许有人不明白，为什么“解释”就慢[2]。因为，一个智能化的编译器可以预测并针对重复和不需要的操作进行优化。这也会提升程序执行的速度。

[2] 关于 Python 慢的原因，感兴趣的同学可以参考这篇博客《为什么说 python 很慢》https://www.jianshu.com/p/3fa56d9f58cb。本书不会一一解释，仅挑选其中和语言特性有关的进行简单的讲解

第二，我很慢，因为我是动态性语言不是静态性语言，我在声明 l 是个 list 的时候，不像 java 一样 List<String> l = new ArrayList<String>()。而是 l=[]。也就是说，解释器在解释的时候，不知道 l 是什么类型，只知道 l 有没有定义过，只有在运行的时候判断这个变量的类型，以及是否有对应的方法。具体见图 2.4。这个动态性的特点，让程序员在编程的时候可以少写很多代码，但是如果程序员犯迷糊，就会帮倒忙（动态一时爽，调试火葬场），比如字符串和整数（甚至小数）后面都可以接% 运算符，但是前者表示用数据来进行格式化，代替%d，%s 等占位符，后者是表示取模运算。[3] 可以利用 type 类进行判断。

```
hc@hc-Lenovo-B50-45:~/Documents/spider book/code$ python
Python 3.7.0 (default, Jun 28 2018, 13:15:42)
[GCC 7.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> l.append(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'l' is not defined
>>> l=[]
>>> l
[]
>>> l.append(1)
>>> type(l)
<class 'list'>
>>> l=''
>>> l.append(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'str' object has no attribute 'append'
>>> type(l) is str
True
>>>
```

图 2.4: python 动态类型

第三，我很慢，因为我支持多线程，但是支持得很不好，通俗地讲，叫鸡肋，[4]不能充分地利用 cpu（开多个进程除外）。当然写爬虫嘛，没有太大影响，因为爬虫主要是 IO 密集型的，计算量很少。所以我在进行 CPU 密集型的任务的时候，会很慢。换句话来说，我不适合进行 CPU 密集型的任务。你可能要反驳我说：“深度学习了解一下?”。那么，我要给你介绍一下我的绰号—— “胶水语言” [5]。我一般来完成业务流程，而需要性能的核心模块（比如 numpy 之类和科学计算有关的库）由我的兄弟 C/C++ 这样的语言来写。如果你快，那很 OK，但是如果你不快还不会抱大腿，那么非常 Sorry，你是比不过我的。

Last but not the least，这回总算和慢不搭边了。我是强类型的语言！我是强类型的语言！我是强类型的语言！重要的话说三遍！以后如果和别人介绍我，你可以说我是动态类型的，但是如果你和别人说我是弱类型的，对不起，咱们不认识！1+ “1” =2，这叫弱类型（某一个变量被定义类型，该变量可以根据环境变化自动进行转换，不需要经过显性强制转换。）；1+ “1” = 报错，这叫强类型（如果不经过强制转换，则它永远就是该数据类型了）。虽然稍微麻烦了一点，但是可以避免很多 Bug（Php：为什么每次躺枪的都是我）。

[3]这点与 c 不同，主要体现在负数上，可以参考博客https://blog.csdn.net/SharonLu1216/article/details/76774829
[4]关于 Python 为什么多线程鸡肋，推荐看https://www.zhihu.com/question/23474039里高赞的回答，虽然有点难
[5]关于胶水语言，可以参考知乎https://www.zhihu.com/question/21626095

本章小结

本章介绍了一些 Python 的性质和特点，虽然内容有点枯燥，但是作者用幽默的语言（不要脸）进行了说明，对以后的学习会有点帮助。下章开始，要介绍 Python 的基本语法了。

Chapter 3

Python 语法基础

> Life is short, you need Python.
>
> ——Bruce Eckel

3.1 Python 代码示例

友情提醒

本章开始，将广泛用到终端，Linux 和 windows 都行，但笔者推荐 Linux，因为后期用到 docker，对一般人来说，Linux 更方便，以下命令，如无特别指明，皆指 Linux 命令

```
import requests req = requests.get('http://www.xicidaili.com/nn/')
print('content',req.content,sep=': ') print(type(req.content))
print('-'*10,'end','-'*10,end='来个换行吧')
```

代码清单 3-1.py

```
hc@hc-Lenovo-B50-45:~/Documents/spider book/code$ vim 3-1.py
hc@hc-Lenovo-B50-45:~/Documents/spider book/code$ python 3-1.py
content: b'<html>\r\n<head><title>503 Service Temporarily Unavailable</title></h
ead>\r\n<body bgcolor="white">\r\n<center><h1>503 Service Temporarily Unavailabl
e</h1></center>\r\n<hr><center>nginx/1.1.19</center>\r\n</body>\r\n</html>\r\n'
<class 'bytes'>
---------- end ----------来个换行吧hc@hc-Lenovo-B50-45:~/Documents/spider book/c
ode$
```

图 3.1: 代码示例

结果如图 3.1所示。

Excuse me? 不是说好了这章讲语法的吗？一上来就扔了个代码和结果，让我们自学?

非也非也！首先，我是要告诉你怎么运行 python 源代码 (推荐命名为 *.py)——python filename。如果你的目的是“系统地”学习语法，那么恭喜你，来错地方了。菜鸟教程[1]和廖雪峰的教程[2] 讲得比我更好，前者适合真菜鸟，后者适合假菜鸟（不会 Python，但是其它语言基础比较好），但是后者质量不错，比前者更好，能学到更多知识。

友情提醒

个人觉得，直接讲语法，会有一些不懂的，或者因为厌倦而放弃，或者看似学会了，但不知道怎么用。给出代码，可以引起好奇心，哇，Python 原来还有这种丧心病狂的功能！可以在给出的代码的基础上进行修改，写少量自己不会写的代码，避免重复劳动，做做实验，加深印象。

接下来，我们打开终端（cmd），确保你已经配置好 Python（包括 pip）的环境变量，以及 requests 库（Linux 用户可以通过 pip list | grep requests 命令来确认是否安装 requests 库，或者输入 python 回车，import requests 回车）。如果没有装 requests 库，在终端输入：pip install requests；如果没有配置好环境变量，那么，请百度相关教程。

之后，输入 Python，按照图 3.2, 输入这些内容。

```
hc@hc-Lenovo-B50-45:~/Documents/spider book/code$ python
Python 3.7.0 (default, Jun 28 2018, 13:15:42)
[GCC 7.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> type(requests),type(requests.get)
(<class 'module'>, <class 'function'>)
>>> resp = requests.get('http://www.xicidaili.com/nn/')
>>> type(resp),type(resp.content),type(print)
(<class 'requests.models.Response'>, <class 'bytes'>, <class 'builtin_function_o
r_method'>)
>>> print(resp.content)
b'<html>\r\n<head><title>503 Service Temporarily Unavailable</title></head>\r\n<
body bgcolor="white">\r\n<center><h1>503 Service Temporarily Unavailable</h1></c
enter>\r\n<hr><center>nginx/1.1.19</center>\r\n</body>\r\n</html>\r\n'
>>>
```

图 3.2: type 所有变量

稍微总结一下，通过这些 type, 我们可以看到 Python 做了很多事：导入（import）模块 (requests)。调用模块 (requests) 里面的函数 (get) 向http://www.xicidaili.com/nn/这个网站发送一个 get 请求，以及调用 Python 内置的函数 (print)，将一个 bytes 类型的对象 (响应的正文部分) 输出到控制台。然后还可以知道这些类名也好，函数名也罢，都是某个类的对象，因此，一般的对象的操作（比如赋值）对函数名和类名等，都是可以使用的。也就是说，我们在给自己的对象起名字的时候，**最好不要使用已经有的类的名字，否则会覆盖原有的功能，除非你真的希望这样做**。

有了这些基础的认识，我们先来了解模块 requests。

[1] Python3 教程 http://www.runoob.com/python3/python3-basic-syntax.html
[2] Python 教程 - 廖雪峰的官方网站 https://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000

3.2 模块

小诀窍

学习最好的方法就是模仿，尤其是有点基础的人，找个库的代码看看，可以学到不少知识。

如图 3.3在终端中输入 find 你的 anaconda 安装目录 (我是 /anaconda3) -name requests, 然后，cd 进入其中一个文件夹。再执行 ls 命令。

```
hc@hc-Lenovo-B50-45:~/Documents/spider book$ find ~/anaconda3/ -name requests
/home/hc/anaconda3/pkgs/requests-2.19.1-py37_0/lib/python3.7/site-packages/reque
sts
/home/hc/anaconda3/pkgs/pip-10.0.1-py37_0/lib/python3.7/site-packages/pip/_vendo
r/requests
/home/hc/anaconda3/pkgs/pip-9.0.1-py36_1/lib/python3.6/site-packages/pip/_vendor
/requests
/home/hc/anaconda3/pkgs/pip-18.1-py36_0/lib/python3.6/site-packages/pip/_vendor/
requests
/home/hc/anaconda3/lib/python3.7/site-packages/requests
/home/hc/anaconda3/lib/python3.7/site-packages/pip/_vendor/requests
hc@hc-Lenovo-B50-45:~/Documents/spider book$ cd /home/hc/anaconda3/pkgs/requests
-2.19.1-py37_0/lib/python3.7/site-packages/requests
hc@hc-Lenovo-B50-45:~/anaconda3/pkgs/requests-2.19.1-py37_0/lib/python3.7/site-p
ackages/requests$ ls
adapters.py  compat.py      hooks.py           packages.py      structures.py
api.py       cookies.py     __init__.py        __pycache__      utils.py
auth.py      exceptions.py  _internal_utils.py sessions.py      __version__.py
certs.py     help.py        models.py          status_codes.py
```

图 3.3: 寻找 requests 库的路径

在这些文件中，有一个 ___init___.py，准确地说，每个 Python 的包里面都有一个叫 ___init___.py 的文件。**这是 Python 包和一般文件夹的本质区别，如果 ___init___.py 不存在，这个目录就仅仅是一个目录，而不是一个包，它和里面的模块和嵌套包（比如 numpy.linalg 这个 numpy 包的子包）就不能被导入**。因此，我们如果有自己写一个包的时候，千万不要把 ___init___.py 的文件给删除了，宁可放一个空的 ___init___.py 的文件（IDE 一般会自动为我们创建一个 ___init___.py，不用我们自行创建）。

接下来，我们可以看一下 requests 包的 ___init___.py 文件，看一下怎么编写 ___init___.py 文件，以及涉及的一些 Python 语法。

用编辑器（Linux 下的 gedit，vim/vi，emacs,nano 等，windows 下的 notepad++,notepad 等或者一些跨平台的如 sublime text）打开 ___init___.py 文件，或者用 spider,Pycharm 的 IDE 的代码跳转等功能，或者去 github 或 pypi 上搜索 requests[3]查看 ___init___.py 的代码。

友情提醒

PyPI(Python Package Index) 是 python 官方的第三方库的仓库，所有人都可以下载第三方库或上传自己开发的库到 PyPI。PyPI 推荐使用 pip 包管理器来下载第三方库。

[3]这种方法不一定能找到所有的库。requests github: https://github.com/requests/requests/blob/master/requests/__init__.py pypi: https://pypi.org/project/requests/

100 多行的代码，看下它做了怎样的事。(版本不同，代码可能有一些不同)。

第一行，# -*- coding: utf-8 -*-声明编码方式为 utf-8。

第 3-6 行，# 开头，用于注释，会被解释器读取（如声明编码)，但不会被执行。这里画了一个 Requests。(丑)

后面许多行被 2 对 3 个双引号包括，表示多行字符串, 关于字符串，在小小节 3.3.1节会讲述。讲述了 requests 库的基本用法。

第 43 行到 46 行导入了一些模块。到这里，已经出现了两种导入模块的方法。那么这里就总结一下，顺便剧透一下剩下的用法, 以 requests 库举例。

1) import 模块。调用：该模块. 对象（如果是这里的对象是广义的对象，通过这段时间的学习，相信你已经能够意识到，模块名，函数名，类名，变量，这些其实都是一个对象的引用）举例：import requests 然后 r = requests.get('http://www.baidu.com')

2) from 模块 import 对象。调用：该对象。举例：from requests import get 然后 get('http://www.baidu.com')

3) import 模块 as 别名。调用：别名. 对象，不能用该模块名. 对象访问。(模块名比较丑的时候，比较好用）import numpy as np 然后 a=np.mat([[1,2],[1,3]])

4) 模块可以用相对路径引用，. 模块，.. 模块，见 46 行，* 可以匹配所有对象。

小诀窍

如果你感到没有看懂，可以在 Python 解释器中输入 globals() 函数，用各种方式导入一个模块，再输入 globals() 函数，看看有什么区别，你就明白了

接下来是 2 个 def 代码，我们暂时先不要仔细地看下面的内容，只要知道这是 2 个函数，里面用了 if 控制流程即可。后面是捕获异常。后面就没啥熟悉的知识了。

代码的结构已经剖析完了，接下来来聊聊为什么要用模块。

我们写模块的目的是，让代码模块化，更容易复用。在这方面做得比较好的，除了模块，还有俩，一个是函数，一个是类，而且，在模块中，函数和类也是很常见的，或者是写在模块本身，或者是写在其它模块被导入。

3.2.1 函数

函数，在 c 里面见过，但是，如果只是 C 里函数的功能，那么这门语言就不配用 “人生苦短，我用 Python” 这句广告了。Python 中的函数究竟有什么强大之处?

首先，由于 Python 是动态类型的，所以不用声明返回类型，参数也不用类型。

第二，由于 Python 内置的数据结构元组（tuple）(见小小节 3.3.4)，使得 **Python 可以返回 0-多个返回值**。

第三，Python 可以给函数提供默认参数，默认参数的存在，使得 Python 函数的调用，千变万化，可以省略默认参数，可以提供不同功能 (在函数中加上 if)，类似于 Java 中的方法重载，却比它简单得多。不过，无论是函数，还是方法，Python 都不支持重载，因为，函数名或者方法名，本身

就是一个对象的引用，如果你定义了两个相同名字的对象，那么后面一个势必会取代前面一个，把它覆盖，你将无法调用前面一个。另外需要注意，有默认值的参数，必须放在无默认值的参数后面。

第四，Python 在调用函数写实参的时候，可以为每个参数添加函数名来增加可读性，也方便程序员编码，因为由于你指定了名字，Python 解释器就可以机智地区分他们，可以少记很多参数的顺序，如 requests.get(headers={},url='http://www.baidu.com')（我们可以把它叫做关键字参数，调用实参的时候没有函数名的参数叫位置参数）。关键字参数必须放在位置参数的后面。

小诀窍

关于关键字参数必须放在位置参数的后面这一点，可以参考数学中的排列组合问题——排队，有位置限制的，必须先排，没位置限制的，可以放到他们排好以后再排，在 Python 中，位置参数就是位置有要求的参数

友情提醒

Python 中的函数和方法没有访问权限这一概念，如果你不希望你的方法被其它模块调用，请以 _ 开头命名函数

友情提醒

对于简短的函数，可用 lambda 表达式代替，f1=lambda :'hello world',f2=lambda x,y:print(x+2*y)

3.2.2 类

和类联系在一起的一个概念，就是面向对象，由于爬虫涉及的面向对象的知识很少，这里就作一个简单的介绍，如有需要，可以查找网上的教程。

类，用 class 类名（父类名）: 声明。一个类里面存在 0 个或多个方法。方法，可以理解成和类绑定在一起的函数，但是比函数多了一个 self 参数，表示函数所属的对象本身。其中有一些魔法方法，开头和结尾都有 2 个 _，**在特定的时期会调用**。比如 ___init___，在类初始化的时候调用，可以理解成 java 中类的构造方法，还有一些魔法方法，实现它们可以让我们以更简单的方法去调用它们，后面要讲的列表，字典，之所以调用如此方便，就是因为在编写这些类的时候，实现了其中的一些魔法方法。函数或方法，之所以对那么多种类的对象都适用，因为某些函数在编写的时候，并不是直接处理对象的，而是，**调用对象写好的魔法方法**，对于魔法方法，我们不需要过多关心，了解它们存在即可。

创建类，用类名（参数）的方式，比如 a=list((1,2,3))

另外，Python 没有接口的概念，Python 有多重继承，但是不常用，继承其它类，一个父类基本够用，调用父类的方法用得不多，可以用到时百度。

3.3 常用内置数据结构

Python 好用的一个原因，就是内置了许多好用又常用的数据结构，比如字符串，列表，元组，字典等，每种数据结构都有很多丰富的好用的方法。

3.3.1 字符串

字符串（比如'Python',"py"）用一对单引号或一对双引号表示，基本没有区别，在引号内有双引号时，推荐用单引号，在引号内有单引号时，推荐用双引号，否则需要转义。

多行字符串用一对 3 引号表示，用于注释或说明。

字符串里，有很多常用的方法。如图 3.4所示。这个内置数据结构将在以后频繁用到。

```
>>> str1='\nPython ,Jython, Cython,PyPy and ironPython\n'
>>> str1.split()
['Python', ',Jython,', 'Cython,PyPy', 'and', 'ironPython']
>>> str1.split('and')
['\nPython ,Jython, Cython,PyPy ', ' ironPython\n']
>>> str1.strip()
'Python ,Jython, Cython,PyPy and ironPython'
>>> str1.rstrip()
'\nPython ,Jython, Cython,PyPy and ironPython'
>>> str1.lstrip('\nPython')
' ,Jython, Cython,PyPy and ironPython\n'
>>> ','.join('Python')
'P,y,t,h,o,n'
>>> ','.join(['3','6','0'])
'3,6,0'
>>> ','.join([3,6,0])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sequence item 0: expected str instance, int found
>>> str1.count('ython')
4
>>> str1.find('ython')
2
>>> str1.find('java')
-1
>>> str1.upper()
'\nPYTHON ,JYTHON, CYTHON,PYPY AND IRONPYTHON\n'
>>> str1.lower()
'\npython ,jython, cython,pypy and ironpython\n'
>>> str1.count('java')
0
>>> str1.index('java')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

图 3.4: Python

strip 默认去除字符串两侧的中英文空格，制表符 \t，换行符 \r, \n 等，添加参数，可以使指定字符去掉。l 表示 left,r 表示 right。

split 方法指定分隔符对字符串进行分割，甚至还可以指定次数，默认全部分割。

string 模块的函数**不支持正则表达式**，如果想调用正则表达式，请用 re 模块。

join 用指定参数将参数们连接到一起（参数必须可迭代的字符序列，比如字符列表，字符元组，字符串等）

replace 方法返回替换后的对象，但不对参数本身进行修改。

find 和 index 方法可以检测字符串中是否包含子字符串 str ，如果指定 beg（开始）和 end（结束）范围，则检查是否包含在指定范围内，如果包含子字符串返回开始的索引值，否则 find 返回-1,index 抛出异常。

count 方法返回字符出现的次数。

字符串中的 \ 表示转义字符，如果在正则表达式或者路径中，需要用到，要用 2 个 \ 来代替。或者用 r'\ is a backslash' 的方式（外面加个 r 表示 raw, 里面的 \ 不转义）。

3.3.2 字节

字节（bytes）既不好用，又不常用，但是在爬虫中又偏偏要用到。

字节用 b" 表示。如 b'<html><body></body></html>'。这里要掌握它与字符串的互相转换。

字符串转字节，用".encode() 方法。字节转字符串用".decode() 方法。可以添加参数，一般为'UTF-8'

3.3.3 列表

这是一类十分常用的，像数组，链表的东西。但是，列表里，可以是任意类型的，可以是任意长度的。

列表用 [] 表示, 类名 list。

append 方法在后面添加指定对象，extend 在后面用参数列表延展原列表。

Python 列表索引从 0 开始，支持负索引，即右往左数。

切片是 Python 又一丧心病狂的语法糖。格式为 l[start:end:step] 支持正向切、反向切、跳着切、切到底、切一半、边切边改、切了当拷贝。包括 start 但不包括 end。

友情提醒

关于切片什么时候会改变原值，什么时候不会，请自行思考（考虑一下引用）并实验

切片的强大，使得一些 list 的方法不常用了，比如 reverse 方法和内置的 reversed 函数。

友情提醒

Python 里有很多像 reverse,reversed 这样命名的方法或函数，比如 sort,sorted，性质有所不同。前者改变自身，后者不改变自身，而是返回一个拷贝。

```
>>> l=[1,2,3,4,5]
>>> l.append(6)
>>> l.extend([7,8,9])
>>> l
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> l[-1]
9
>>> l[0:4]
[1, 2, 3, 4]
>>> l[0:-1]
[1, 2, 3, 4, 5, 6, 7, 8]
>>> l[0:]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> l[0::3]
[1, 4, 7]
>>> l[3:5]=9,8
>>> l
[1, 2, 3, 9, 8, 6, 7, 8, 9]
>>> a=l[3:5]
>>> a[0]=-1
>>> a
[-1, 8]
>>> l
[1, 2, 3, 9, 8, 6, 7, 8, 9]
>>> reversed(l)
<list_reverseiterator object at 0x7f9bc1f9ca20>
>>> list(reversed(l))
[9, 8, 7, 6, 8, 9, 3, 2, 1]
>>> l
[1, 2, 3, 9, 8, 6, 7, 8, 9]
>>> l.reverse()
>>> l
[9, 8, 7, 6, 8, 9, 3, 2, 1]
>>> l.sort()
>>> l
[1, 2, 3, 6, 7, 8, 8, 9, 9]
```

图 3.5: list

小诀窍

时间久了可能会记反，这里给出一个我的记忆方式，英语中，动词 +ed，是过去分词，常常用来当形容词。因此，sorted 函数返回一个形容词版（有序的）的列表，不改变列表本身。而 sort 是方法，表示一个对象的行为或者动作，这个行为或动作的对象是 self, 即列表本身。

友情提醒

reversed 方法返回一个生成器，sorted 方法返回一个列表。生成器可以生成列表，但是不会立即生成，在用 for 语句迭代的时候和列表相似，所以比较节约内存。

3.3.4 元组

元组用 (1,) 或 (1,2,3) 表示，() 可以省略（如果不会发生歧义，比如交换变量，可以省略 ()，但是函数参数那里，就不能省略）

元组和列表十分相似，所以，不介绍它的语法了。唯一的区别是，元组不支持修改。类名为 tuple。

元组可以干啥？函数多值返回，并行赋值（交换两个变量可以用 a,b=b,a），可以当字典的键（这点列表不行）。

3.3.5 字典

字典是 Python 中又一好用的数据结构, 类名 dict, 用 key1:value,key2:value 表示，特点是可以直接根据键名获取键值。一方面，用过其它语言 map 数据结构的人，都知道 map 有多好用，存取有多方便。另一方面，现在很多网页都是通过 json 格式传输数据的，这意味着爬虫有时需要抓取 json 格式的数据。而 json 和字典一样，都是键值对的形式，这意味着，json 和 dict 可以方便地转换（标准库里的 json.dumps,json.loads,json.load，json.dump）。有些 NoSQL（比如 Mongodb）也是键值对类型的，因此，也可以通过 dict 来封装对应的数据。

小诀窍

这里，又出现了令人混淆的函数名，可以这样记忆 dumps,loads 的 s 可以记忆为 string，说明这两个函数是对 string 类型操作，剩下两个是对文件操作。

dict 的基本用法可以见图 3.6。

假设字典的名字叫 d。

获取键值的方式有 3 种，一种是 d[键名]，一种是 d.get(键名)，一种是 d.setdefault(键名，默认值)。区别在于，在不存在键名的时候，用 d[键名] 获取键值会报错，get 方法会返回 None（类似于 c 里的 NULL 各 java 里的 null，以及一些其它语言中的 nil）。d.setdefault 方法会返回默认值并修改字典。

update 允许用新的字典去更新旧的字典。如果旧字典没有对应的键，那么旧字典会增加对应的键名和键值。如果旧字典有对应的键名，那么新字典会更新对应的键值。如果新字典没有对应的键名，那么旧字典对应的键值不会被改变。

d.keys() 方法获取字典的键名的集合（dict_keys）。之所以叫作集合，因为，**字典里不会有相同的键值**。键名必须是不能修改的对象，比如元组，字符串，数字，但不能是可以修改的对象。

d.values() 方法获取字典里的键值列表（dict_values）。

d.items() 方法获取字典里的键值对的列表（dict_items）。

zip 可以但不仅限于将用作键名的列表，和用作键值的列表，打包成键值对元组的列表（可以是任意的列表）。

```
>>> d={'man':'huacong','woman':'syj'}
>>> d['man']
'huacong'
>>> d['is_couple']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'is_couple'
>>> d.get('is_couple')
>>> d.setdefault('is_couple',0) # 单身狗心中永远的痛55555
0
>>> d['is_couple']
0
>>> d
{'man': 'huacong', 'woman': 'syj', 'is_couple': 0}
>>> d.update({'is_couple':1,'have_child':0}) # someday in the future
>>> d
{'man': 'huacong', 'woman': 'syj', 'is_couple': 1, 'have_child': 0}
>>> d.keys()
dict_keys(['man', 'woman', 'is_couple', 'have_child'])
>>> d.values()
dict_values(['huacong', 'syj', 1, 0])
>>> d.items()
dict_items([('man', 'huacong'), ('woman', 'syj'), ('is_couple', 1), ('have_child
', 0)])
>>> d1 = dict([('man', 'huacong'), ('woman', 'syj'), ('is_couple', 1), ('have_ch
ild', 0)])
>>> d1
{'man': 'huacong', 'woman': 'syj', 'is_couple': 1, 'have_child': 0}
>>> d2=zip([1,2,3],[4,5,6])
>>> d3 = dict(d2)
>>> d3
{1: 4, 2: 5, 3: 6}
>>> d
{'man': 'huacong', 'woman': 'syj', 'is_couple': 1, 'have_child': 0}
>>> d4=dict(d)
>>> d4
{'man': 'huacong', 'woman': 'syj', 'is_couple': 1, 'have_child': 0}
```

图 3.6: dict 用法

3.3.6 集合

集合用得不多，但是可以看看图 3.7。

数学里的集合是无序的，不重复的，Python 亦然。对于集合，Python 支持用操作符和方法来完成。现用一张表格来总结，其中

$$A = \{1, 2, 3\}$$
$$B = \{1, 3, 4\}$$

remove 方法和 discard 方法，都可以去掉集合中的元素，区别在于当集合中没有指定元素的时候，前者抛出异常，后者什么都不执行。

clear 方法可以清除所有的元素。

```
>>> a,b = set([1,2,3]) ,set([1,3,4])
>>> a&b,a|b,a-b,a^b
({1, 3}, {1, 2, 3, 4}, {2}, {2, 4})
>>> a.discard(1)
>>> a
{2, 3}
>>> a.remove(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 1
>>> a.add(1)
>>> a.update([1,2,3,4,5,6])
>>> a
{1, 2, 3, 4, 5, 6}
>>> b.issubset(a),b<=a
(True, True)
>>> a.issuperset(b),a>=b
(True, True)
>>> a.pop()
1
>>> a.clear()
>>> a
set()
>>> a&=b
>>> a
set()
>>> a|=b
>>> a
{1, 3, 4}
>>>
```

图 3.7: set 用法

3.4 流程控制

3.4.1 选择

```
if 情况1:# 情况不用加括号
    做一些事
elif 情况2:# 可选 else if的意思
    pass #啥都不做，相当于java和c中的;
elif 情况3:
    做另外的事
else:# 可选
    啥都不做
```

代码清单 if-elif-else

友情提醒

Python 中没有 switch-case 语句，因为 switch-case 和 if-elif-else 代码量差不多，功能也一致，the zen of Python 里有一条 There should be one– and preferably only one –obvious way to do it.

Python 条件，用 and,or,not 来连接，不用 &&,||,!。

表 3.1: Set 用法

操作	结果	操作符	方法
并集	$\{1,2,3,4\}$	a|b	a.union(b)
交集	$\{1,3\}$	a&b	a.intersection(b)
差集	$\{2\}$	a-b	a.difference(b)
相对差集	$\{2,4\}$	a∧b	a.symmetric_difference(b)
A 真包含于 B	False	a<b	a.issuperset(b)
让 set “A” 只保留 set “B” 中有的元素	None	a&=b	a.intersection_update(b)
让 set “A” 添加 set “B” 中有的元素	None	a|=b	a.update(b)
让 set “A” 变成 AB 的相对差集	None	a∧=b	a.symmetric_difference_update(b)
让 set “A” 变成 AB 的差集	None	a-=b	a.difference_update(b)

Python 允许 2<money<=3 这样的写法。

is 判断是否是同一个对象（id(对象) 是否相等），== 判断值是否相等。

c/Java 中的 a?b:c 的条件表达式，在 Python 中写作 b if a else c。返回 b（如果满足条件 a 的话）否则返回 c。

3.4.2 循环

for

```
for 变量 in 可迭代的对象: #可以是列表, 元组, 字典, 集合, range对象、生成器等
    做一些事
else: # 可选, 当for循环中没有成功break时
    做别的事
```

代码清单 for-else

while

```
i=0
while 条件成立:
    i+=1 # 没有i++的说法
else:
    做别的事
```

代码清单 while-else

3.4.3 异常处理

```
try:
    raise Excepption('some string')
except KeyError as e:# 捕捉到异常1并垂命名
    pass
except (NameError,TypeError):# 捕捉到2种异常, 按相同方式处理
    pass
except:
```

```
    pass
else:
    pass
finally:# 无论如何都会进行，用于关闭数据库连接等。
    pass
```

代码清单 try-raise-except-finally

3.5 其它特性

嵌套。由于 Python 是动态类型，因此，你可以很容易地构建出多维列表等高级的数据结构。

关键字 in 可以检查对象是否在（列表，元组，字符串，字节，字典的键名，集合中）。

del 语句可以按照索引删除列表元素、按照键名删除字典元素。

assert 用于断言。

* 可用于重复，比如'python'*10。

+ 可用于字符串、元组、列表的拼接。

% 可用于字符串格式化和求模 (c、java、c++ 中是求余)。

Python 支持按位与、按位或、按位非、移位运算。

int，float，str，list，tuple，dict 等都可以用对应的类名进行转化。

yield 类似 return，不同点在于 yield 返回值后还会继续进行，使用了 yield 的函数被称为生成器，用 next 函数调用，得到下一个值。生成器的好处是只有在调用的时候才会生成相应的数据（调用到这个数据的时候才会生成这个数据，没有调用到时就没有这个数据），只记录数据的当前位置。

可以用列表/集合/字典/元组表达式配合条件表达式等完成列表/集合/字典/生成器的初始化。

```
a = [[1 if i==j else 0 for i in range(3)] for j in range(3)]# 三阶对角阵
b = {x**2 for x in range(-5,5)}# {0,1,4,9,16,25}
c = (x for x in range(3))# 生成器
```

代码清单 list comprehension

在一些函数中的定义中，还可以看到，函数的参数还有 *args，**kwargs，分别用于多个位置参数，和多个关键字参数，实际上用到的数据结构为元组和字典。

在调用的时候也可以用 *[],**kwargs，不常用，在参数较多的时候用。

```
def a(*x):
    print(x)

def b(**y):
    print(y['a'],y)

def c(a,b,c,d):
    print(a,b,c,d)

def d(a,b,c=1,d=2,e=5,f=6):
    print(a,b,c,d,e,f)

a([1,2,3,4,5])# 调用的时候只有一个位置参数。
```

```
b(**{'a':1,'b':2,'c':3,'d':4})# 不加星相当于1个位置参数。
c(*[1,2,3,4])# 调用的时候，相当于4个位置参数。
d(*[1,2,3],**{'d':3,'e':4,'f':5})# 调用的时候，3个位置参数和3个关键字参数。
```

代码清单 args-kwargs

到现在 Python 的基本语法已经讲得差不多了，还有文件读写和面向对象没讲，留到需要用到的时候再拓展。可以再看看 requests, 或者找一些标准库的模块，是不是好懂很多（当然有些函数不是用 Python 实现的，所以你可能找不到 Python 代码）。

学完了这么多，稍微放松一下。

```
import sys
寻找女朋友=lambda x:print('%d岁，寻找女朋友ing...'%x)
结婚=lambda x:print('%d岁，结婚啦,xswl'%x)
单身=lambda :print('emmm其实单身也挺好...')
等死=lambda:print('等死,人最终还是要死的，也不知我死的时候是一个人还是两个人，都有可能')

def 生活(结婚年龄):
    for 年龄 in range(5):
        print('%d岁，没印象，应该单身吧？'%年龄)
    for 年龄 in range(5,10):
        print('%d岁~%d岁，有印象，单身!'%(年龄,年龄+1))
    for 年龄 in range(10,20,2):
        print('%d岁~%d岁，依然单身'%(年龄,年龄+2))
    while 40>年龄 >=18:
        try:
            寻找女朋友(年龄)
            还没找到女朋友 = 年龄<结婚年龄-3
            if 还没找到女朋友 and 年龄<30:
                continue
            elif not 还没找到女朋友:
                pass
            else:
                raise Exception('唉，注孤生')
        except Exception as e:
            print(e)
        else:
            结婚(年龄+3)
            return
        finally:
            等死()
            年龄+=1
    单身()

if __name__=='__main__':
    assert len(sys.argv)==2,'输入结婚年龄哦'
    结婚年龄=int(sys.argv[1])
```

本章小结

本章介绍了 Python 的基本语法，内容较多，但较为枯燥，但是这些内容是后面内容的基础。下章开始介绍爬虫。

Chapter 4

爬虫基础