第1章

Python 基础

千里之行，始于足下。在利用 Python 开发各种精彩的项目前，我们需要对 Python 的基础知识有一个基本的了解。基础知识看似简单，却是各种复杂代码的基石，只有将基础打扎实，在之后的进阶学习中才能更加游刃有余。

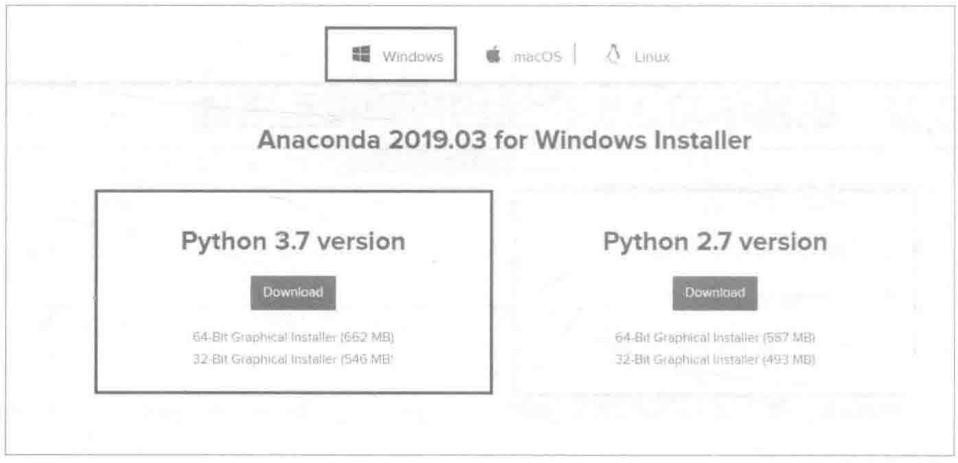
1.1 Python 安装与第一个 Python 程序

本节首先介绍 Python 的安装，然后带领大家编写第一个 Python 程序，以此打开 Python 编程的大门。同时还将介绍 PyCharm 编译器的安装及使用注意事项。

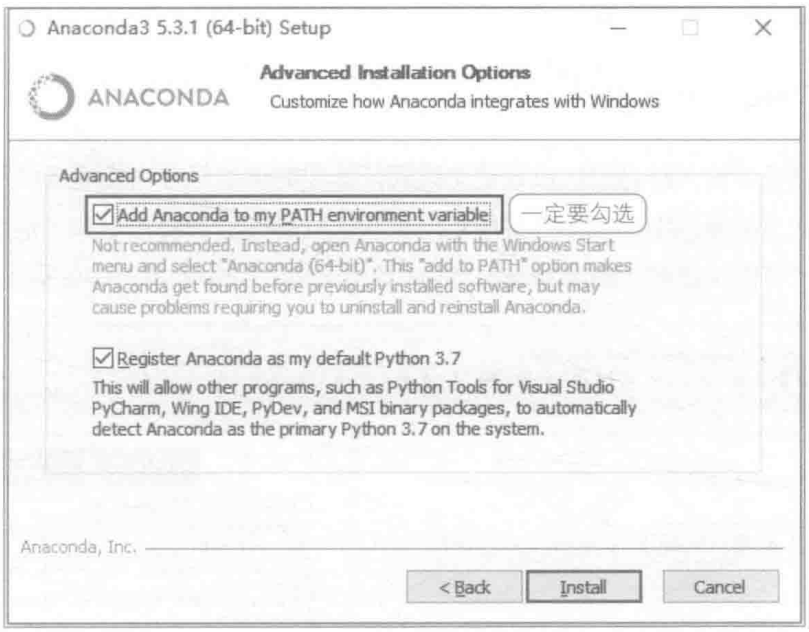
1.1.1 安装 Python

学习 Python 的第一步是什么？自然是安装 Python 了。这里介绍一种非常方便的安装方法——Anaconda 安装。Anaconda 是 Python 的一个发行版本，安装好了 Anaconda 就相当于安装好了 Python，并且里面还集成了很多关于 Python 科学计算的第三方库。

在浏览器中打开 Anaconda 的官网下载地址：https://www.anaconda.com/download/，或者在搜索引擎中搜索 Anaconda，进入官网，根据当前操作系统配置选择相应的版本下载即可。这里选择 Windows 系统的 Python 3.7 版本，如下图所示，默认版本是 64 位的。如果操作系统是 32 位的，那么选择 32 位的版本下载即可。



双击下载好的安装文件，进入安装界面。建议不要改变默认安装路径（防止可能出现的安装问题），然后在弹出的界面中单击多次“Next”按钮。安装到如下图所示的界面时，注意一定要勾选第一个复选框，其作用相当于自动配置好环境变量，对初学者来说比较方便。



如果弹出的界面中询问是否安装 Microsoft VSCode（Install Microsoft VSCode），单击“Skip”按钮跳过即可。最后单击“Finish”按钮，完成安装。

1.1.2　编写第一个 Python 程序

安装完 Python，大家是不是有点跃跃欲试了呢？下面就来编写第一个 Python 程序。

安装 Anaconda 的同时就已经安装了一些不错的 IDE（指集成开发环境，也就是用于编写及运行代码的应用程序），如 Spyder、Jupyter Notebook。后面会介绍另一款 IDE 软件——PyCharm。这里先使用 Spyder 编写程序。

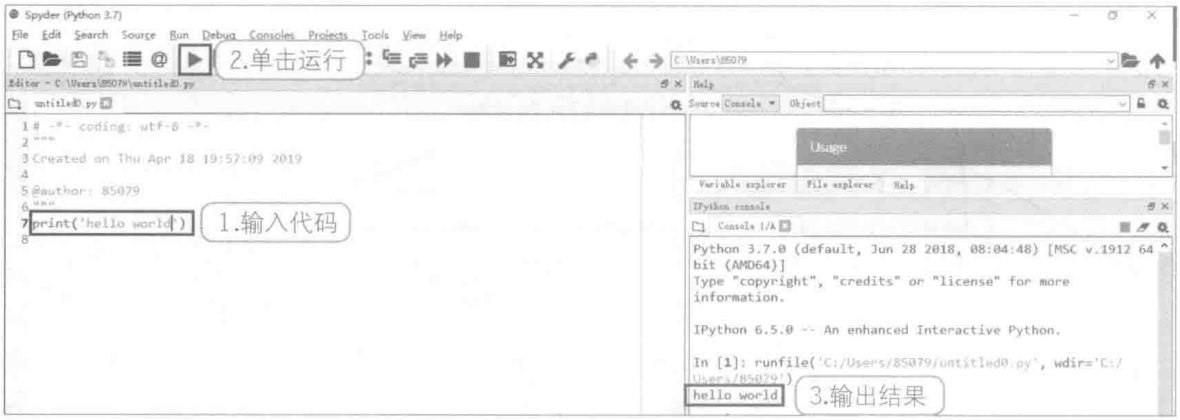
在“开始”菜单中找到并展开 Anaconda 的程序组，单击其中的 Spyder 即可运行该程序。打开 Spyder 后，界面如下图所示。左边框内是输入代码的地方，右边框内是输出代码运行结果的地方，上方的绿色箭头▶是运行代码的按钮，在 Spyder 中，也可以按 F5 键运行代码。

将输入法切换到英文输入模式，在左边输入代码的地方输入：

```
print('hello world')
```

需要注意的是，输入时必须切换到英文模式，其中的单引号可以换成双引号。

然后单击上方的▶按钮（或按 F5 键），在右边就可以看到输出结果“hello world”，如下图所示。读者可以试试把“hello world”改成其他内容，看看会有怎样的输出结果。



1.1.3 PyCharm 的安装与使用

PyCharm 也是一种 Python 的 IDE，其功能与 Spyder 大致相同，可以用来编写和运行程序。PyCharm 的界面比较美观，而且功能也很强大。下面就来为大家介绍 PyCharm 的安装和配置步骤。

PyCharm 的官网下载地址为：https://www.jetbrains.com/pycharm/download/，选择免费版（Community 版）下载安装即可，如下图所示。

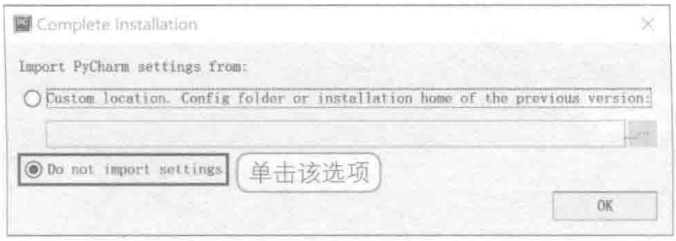
双击下载好的安装文件即可开始安装。安装过程中，在大多数界面单击“Next”按钮和“Install”按钮即可，其中下图所示的界面要勾选两个复选框，以选择 64 位的安装版本及关联后缀名为 .py 的 Python 文件。目前的 PyCharm 版本（2019.1 版）不再支持 32 位操作系统（在

下载页面左侧单击“System requirements”链接可查看系统要求），32 位操作系统可单击下载页面左侧的“Previous versions”链接，下载 2018.3 版 PyCharm，该版本支持 32 位操作系统。

单击“Install”按钮后，等待程序安装，最后单击“Finish”按钮即可完成安装。

初次启用 PyCharm 时需要注意以下事项。

第一步：单击“Do not import settings”单选按钮，如下图所示。



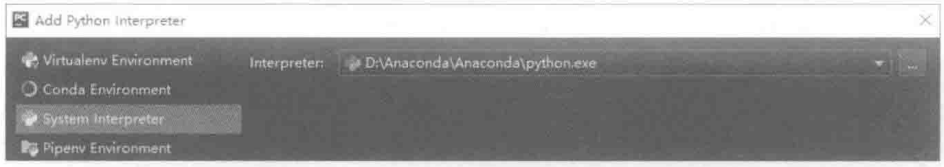
第二步：选择页面风格，建议选择默认的黑色风格。

第三步：选择辅助工具，直接跳过，不做任何设置。

第四步：单击“Create New Project”按钮，创建 Python 项目。

第五步：为项目文件夹命名，这一步要展开“Project Interpreter”选项组，单击“Existing interpreter”单选按钮，如下图所示，这样可以配置之前安装的 Anaconda 运行环境。

如果上图中的“Interpreter”后显示“<No interpreter>”，则单击右侧的□按钮，在弹出的对话框左侧选择“System Interpreter”选项，可以在右侧看到“Interpreter”列表框中显示为“×××\Anaconda\python.exe”，如下图所示，然后单击“OK”按钮。



返回项目创建对话框，单击“Create”按钮即可创建新的 Python 项目。

执行“File>New Project”菜单命令也可以新建项目，然后重复上述步骤。在设置“Project Interpreter”时注意选择“Existing interpreter”选项。

第六步：关闭弹出的官方提示后，等待最下面的 Index 缓冲完毕，如下图所示。缓冲的过程其实是在配置 Python 的运行环境。

第七步：Index 缓冲完毕后即可创建 Python 文件。右击之前创建的项目文件夹，在弹出的快捷菜单中执行“New>Python File”菜单命令，如下图所示。将新的 Python 文件命名为“hello world”。

第八步：创建 Python 文件后，在代码输入框中输入 print('hello world')，其中的单引号可换成双引号。在标题栏上或代码输入框内右击，在弹出的快捷菜单中选择“Run 'hello world'”命令，如下图所示，这样就能运行程序并在下方输出“hello world”。注意，如果 Index 缓冲没有结束，右键菜单中可能看不到“Run 'hello world'”命令，因为运行环境还没配置完毕。

单击界面右上角的▶按钮或按快捷键 Shift+F10 也可以运行程序。

这里再介绍一下 PyCharm 的字体大小等设置。执行“File>Settings”命令，在“Settings”对话框的左侧选择“Editor”选项组下的“Font”选项，在展开的右侧面板中，“Size”用于设置字体的显示大小，“Line spacing”用于设置行距，如下图所示。

知识点

PyCharm 使用常见问题

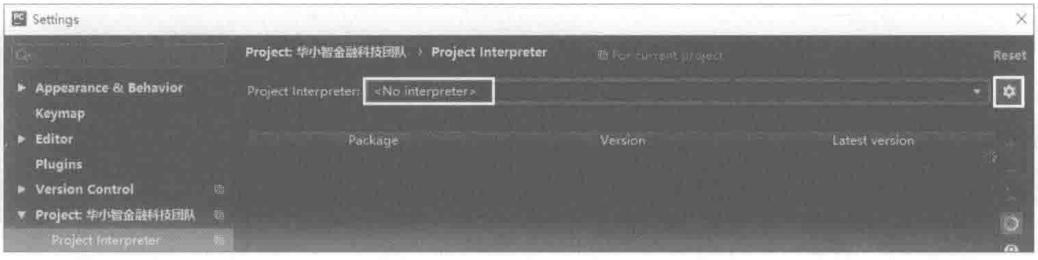
Q1：为什么我第一次打开要等很久，才能进行下一步操作？

A1：第一次打开的时候都有一段等待缓冲的时间，特别是第一次安装的时候，等待最下面的 Index 缓冲完，再进行之后的操作就没有问题啦。

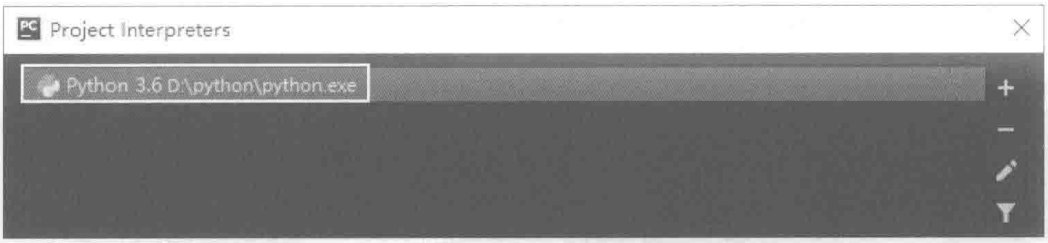
Q2：为什么重新打开 PyCharm 的时候显示没有配置 interpreter（运行环境）？如下图所示。

A2：这是因为每次重新打开 PyCharm 时，它都默认建立了一个新的 project（项目），Python 文件是属于这个 project 的，如果这个 project 没有运行环境，Python 文件也没有办法运行，那么这个时候需要配置运行环境。

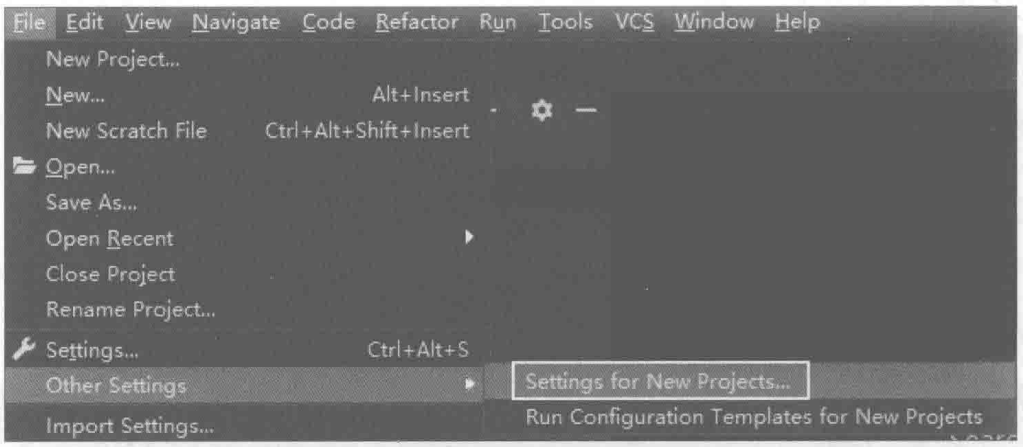
解决方法：单击上图右侧的“Configure Python interpreter”（配置 Python 编译器，即配置运行环境），或者执行“File>Settings”菜单命令，进入设置 Project Interpreter 的界面，如下图所示。在图中可以看到“Project Interpreter”列表框中显示的是“<No interpreter>”，这就是为什么每次重新打开 PyCharm 之后，PyCharm 总是显示没有配置运行环境，因为 PyCharm 默认的运行环境为空。



首先讲治标的方法：进入上图中的界面后，单击右侧的按钮，在弹出的菜单中选择“Show All”命令，然后选择下图所示的运行环境。



再讲治本的方法：修改 PyCharm 的默认运行环境设置。执行“File>Other Settings”菜单命令，选择其中的“Settings for New Projects”命令（有的老版本为“Default Settings”命令），如下图所示。



然后选择“Project Interpreter”，选择已有的 interpreter，再单击右下角的“Apply”按钮，然后单击“OK”按钮，这样默认的 interpreter 就被关联上了。这里配置的是默认设置，而不是单个项目的设置，所以以后打开 PyCharm 时就再也不用配置运行环境了。

1.2 Python 基础知识

安装完 Python 之后，下面来学习 Python 的基础知识。这些基础知识都是后面具体项目实战的基石，大家需好好掌握。

1.2.1 变量、行、缩进与注释

代码文件：1.2.1 变量、行、缩进与注释.py

本小节主要讲解变量、行、缩进与注释的一些基础内容。注意在 Python 中输入代码时一定要切换到英文模式。

1. 变量

变量相当于一个代号。初中数学学过的一次函数 y=x+1，其中，x 就是变量（称为自变量），y 也是变量（称为因变量）。变量的命名必须以字母或下画线开头，后面可以跟任意数量的字母、数字、下画线的组合。本书建议用英文字母开头，如 a、b、c、a_1、b_1 等。注意两点：第一，不要用 Python 的保留字或内置函数来命名变量，例如，不要用 print 来命名变量，因为它与内置函数 print() 重名；第二，变量的命名对英文字母区分大小写，如 D 和 d 是两个不同的变量。

“=”符号可以给变量赋值，演示代码如下：

```
x = 10
print(x)
y = x + 15
print(y)
```

第 1 行代码表示将 10 赋值给变量 x，第 2 行代码表示输出变量 x 的值，第 3 行代码表示将 x 的值加上 15 后赋值给变量 y，第 4 行代码表示输出变量 y 的值。

执行该程序，输出结果如下：

```
10
25
```

知识点

print() 函数

print() 函数用于将结果打印输出，以后会经常用这个函数来输出结果。

2. 行

在 Python 中，代码都是一行一行输入的，输入完一行后按 Enter 键即可换行。

3. 缩进

缩进是 Python 中非常重要的一个知识点，它类似于 Word 的首行缩进。缩进的快捷键是 Tab 键，在 if、for、while 等语句中都会用到缩进。先来看下面的代码：

```
x = 10
if x > 0:
    print('正数')
else:
    print('负数')
```

第 2 ～ 5 行代码是之后会讲到的 if 判断语句。其实判断语句很简单，if 表示“如果”，将上面的代码翻译成中文就是：

```
让x等于10
如果x大于0：
    打印输出'正数'
否则：
    打印输出'负数'
```

在输入第 3 行和第 5 行代码之前必须按 Tab 键来缩进，否则运行程序时会报错。

如果要减小缩进量，可以按快捷键 Shift+Tab。如果要同时对多行代码调整缩进量，可以选择多行代码，按 Tab 键统一增加缩进量，再按快捷键 Shift+Tab 统一减小缩进量。

4．注释

注释也叫批注，大多起提示作用，运行程序时会直接跳过注释。注释方法主要有两种：

```
# 这之后是注释内容
'''这里面是注释内容'''
```

可以手动输入 # 或 '''，也可以用快捷键来添加注释。在 PyCharm 中，添加注释的快捷键是 Ctrl+/；在 Spyder 中，添加注释的快捷键是 Ctrl+1。选择多行代码后按注释快捷键，可以将选中的代码批量转换为注释。

1.2.2 数据类型：数字与字符串

代码文件：1.2.2 数据类型：数字与字符串.py

Python 中有 6 种数据类型：数字、字符串、列表、字典、元组、集合。其中前 4 种数据类型用得相对较多，本小节先介绍数字和字符串。

数字和字符串的核心知识点是需要知道 1 和 '1' 是两种不同的数据类型。前者是一个数字，可以进行加减乘除的操作；而后者则是一个字符串，也就是常说的文本内容。字符串的最大特点就是在它的两旁有单引号或双引号，演示代码如下所示：

```
a = '我是一个字符串'
```

不同的数据类型是不能相互运算的，例如：

```
a = 1 + '1'
print(a)
```

上述代码在运行时会报错：unsupported operand type(s) for +: 'int' and 'str'（不同类型的数据无法直接运算）。

知识点

如何获取和转换变量的类型

用 type() 函数可以显示变量的类型，演示代码如下：

```
a = 1
print(type(a))
b = '1'
print(type(b))
```

输出结果如下，表明第一个变量是 int 格式（整数格式），第二个变量是 str 格式（字符串格式）。数字类型除了 int 格式外，还有 float 格式（浮点数格式，即小数格式）。

```
<class 'int'>
<class 'str'>
```

通过 str() 函数可以把数字转换成字符串，演示代码如下：

```
a = 1
b = str(a)  # 将数字转换成字符串，并赋值给变量b
c = b + '1'
```

将变量 c 通过 print() 函数输出，结果如下，可以看到实现了字符串拼接的效果。

```
11
```

通过 int() 函数可以把字符串转换成数字，演示代码如下：

```
a = '1'
b = int(a)
c = b + 1
print(c)
```

以上代码的输出结果如下：

```
2
```

1.2.3 数据类型：列表与字典、元组与集合

代码文件：1.2.3 数据类型：列表与字典.py

列表（list）和字典（dictionary）是用来存储内容的容器，在 Python 中经常会用到。

1. 列表

（1）列表入门

列表就像一个容器，可以将不同的数据存储到里面并进行调用。例如，一个班级里有 5 名学生，需要有一个容器把他们的姓名放在一起，可以采用如下所示的列表格式：

```
class1 = ['丁一', '王二', '张三', '李四', '赵五']
```

其中列表的格式为：

```
列表名 = [元素1,元素2,元素3,……]
```

列表里的元素可以是字符串，也可以是数字，甚至可以是另外一个列表。下面的列表就含有三种元素：数字 1、字符串 '123'、列表 [1, 2, 3]。

```
a = [1, '123', [1, 2, 3]]
```

利用 for 循环语句可以遍历列表中的所有元素，演示代码如下：

```
class1 = ['丁一', '王二', '张三', '李四', '赵五']
for i in class1:
    print(i)
```

输出结果如下：

```
丁一
王二
张三
李四
赵五
```

（2）统计列表的元素个数

有时需要统计列表里一共有多少个元素（又叫获取列表的长度），可以使用 len() 函数。len() 函数的一般格式为：len(列表名)。演示代码如下：

```
a = len(class1)
print(a)
```

列表 class1 有 5 个元素，所以程序输出结果如下：

```
5
```

（3）调取列表的单个元素

通过在列表名之后加上“[序号]”调取单个元素，演示代码如下：

```
a = class1[1]
print(a)
```

上述代码输出结果如下：

```
王二
```

有些读者可能会有疑问，这里 class1[1] 调取的为什么不是“丁一”呢？因为在 Python 中序号都是从 0 开始的，所以用 class1[0] 才能调取“丁一”。如果想调取列表中的第 5 个元素“赵五”，那么对应的序号就是 4，相应的代码则是 print(class1[4])。

（4）列表切片

如果想选取列表中的几个元素，如选取上述 class1 中的第 2 ～ 4 个元素，就要用到“列表切片”的方法，一般格式为：列表名 [序号 1: 序号 2]。其中，序号 1 可以取到，而序号 2 则取不到，俗称“左闭右开”，加上前面提到的在 Python 中序号都是从 0 开始的，所以第 2 个元素的序号为 1，第 5 个元素的序号为 4，代码如下：

```
class1 = ['丁一', '王二', '张三', '李四', '赵五']
a = class1[1:4]
print(a)
```

其中，序号 1 的元素“王二”是可以取到的，而序号 4 的元素“赵五”则是取不到的，所以最后的输出结果为：

```
['王二', '张三', '李四']
```

有时不确定序号 1 或序号 2，可以采用只写一个序号的方式，演示代码如下：

```
class1 = ['丁一', '王二', '张三', '李四', '赵五']
a = class1[1:]  # 选取第2个元素到最后一个元素
```

```
b = class1[-3:]  # 选取倒数第3个元素到最后一个元素
c = class1[:-2]  # 选取倒数第2个元素前的所有元素（因为“左闭右开”，所以不包含倒数第2个元素）
```

a、b、c 的打印输出结果如下：

```
['王二', '张三', '李四', '赵五']
['张三', '李四', '赵五']
['丁一', '王二', '张三']
```

（5）添加列表元素

使用 append() 函数可以给列表添加元素，演示代码如下：

```
score = []  # 创建一个空列表
score.append(80)  # 通过append()函数给列表添加一个元素
print(score)
```

上述代码的输出结果如下：

```
[80]
```

此时如果又有一个新的元素需要添加，可以使用 append() 函数继续添加，演示代码如下：

```
score.append(90)  # 给列表再添加一个新元素
```

此时的 score 列表打印输出结果如下所示，这样就在原列表基础上新增了一个元素。

```
[80, 90]
```

这个操作在实战中经常会用到。例如，在之后章节的数据评分中，因为并不清楚有多少个数据，就可以用 append() 函数把这些数据一个个加上去。

（6）列表与字符串之间的转换

列表与字符串之间的转换在文本筛选中有很大的作用，后面会详细介绍，这里先大致了解一下。例如，要把列表 class1 转换成一个字符串 '丁一,王二,张三,李四,赵五'，可以用下面的方式：

```
'连接符'.join(列表名)
```

其中，引号（单引号、双引号皆可）中的内容是字符之间的连接符，如“,”“;”等。

所以，若要把 class1 转换成一个用逗号连接的字符串，代码为：','.join(class1)。

```
class1 = ['丁一', '王二', '张三', '李四', '赵五']
a = ','.join(class1)
print(a)
```

上述代码的输出结果如下：

```
丁一,王二,张三,李四,赵五
```

如果把逗号换成空格，那么输出的就是“丁一 王二 张三 李四 赵五”。

字符串转为列表主要用的是 split() 函数，括号里的内容为分割符号，演示代码如下：

```
a = "hi hello world"
print(a.split(' '))
```

输出结果如下，注意这里使用的分割符号为文字之间的空格。

```
['hi', 'hello', 'world']
```

2. 字典

字典是另一种数据存储的方式。例如，class1 里的每个人都有一个数学考试分数，想把他们的姓名和分数一一匹配到一起，那么就需要用字典来存储数据。字典的基本格式如下：

```
字典名 = {键1:值1,键2:值2,键3:值3,……}
```

在字典中，每个元素都有两个部分（而列表中每个元素只有一个部分），前一个部分称为键，后一个部分称为值，中间用冒号相连。

键相当于一把钥匙，值相当于一个箱子，一把钥匙对应一个箱子。那么对于 class1 里的每个人来说，一个人的姓名对应一个分数，相应的字典写法如下：

```
class1 = {'丁一':85, '王二':95, '张三':75, '李四':65, '赵五':55}
```

如果要提取字典中的某一个元素的值，可以通过如下格式实现：

```
字典名['键名']
```

例如，获取王二的分数，可以通过如下代码实现：

```
score = class1['王二']
print(score)
```

上述代码的输出结果如下：

```
95
```

如果想把每个人的姓名和分数都打印出来，代码如下：

```
class1 = {'丁一':85, '王二':95, '张三':75, '李四':65, '赵五':55}
for i in class1:
    print(i + ':' + str(class1[i]))
```

这里的 i 是字典里的键，也就是“丁一”“王二”等内容，class1[i] 输出的就是值，即这些人的分数。因为分数为数字格式，在进行字符串拼接时需要通过 str() 函数进行转换。输出结果如下：

```
丁一:85
王二:95
张三:75
李四:65
赵五:55
```

另外一种遍历字典的方法是通过字典的 items() 函数，代码如下：

```
class1 = {'丁一':85, '王二':95, '张三':75, '李四':65, '赵五':55}
a = class1.items()
print(a)
```

其输出结果如下所示，通过 items() 函数返回的是可遍历的 (键 , 值) 元组数组。

```
dict_items([('丁一', 85), ('王二', 95), ('张三', 75), ('李四', 65),
('赵五', 55)])
```

除了列表和字典外，还有两种存储内容的方式：元组（tuple）和集合（set）。

元组的定义和使用方法与列表非常类似，区别在于列表的符号是中括号 []，而元组的符号是小括号 ()，并且元组中的元素不可修改，元组的演示代码如下：

```
a = ('丁一', '王二', '张三', '李四', '赵五')
print(a[1:3])
```

运行结果如下，可以看到它选取元素的方法和列表是一样的。

```
('王二', '张三')
```

集合是一个无序不重复的序列，和列表也比较类似，用于存储不重复数据。通过大括号 {} 或 set() 函数创建集合，演示代码如下：

```
a = ['丁一', '丁一', '王二', '张三', '李四', '赵五']
print(set(a))
```

运行结果如下，可以看到通过 set() 函数获得了一个集合，删除了重复的内容。

```
{'丁一', '王二', '赵五', '张三', '李四'}
```

相对于列表和字典，元组和集合用得较少。

1.2.4　运算符

代码文件：1.2.4 运算符介绍与实践.py

运算符主要用于将数据（数字和字符串）进行运算及连接，常用的运算符见下表。

运算符	含义	运算符	含义
+	数字相加或字符串拼接	>=	大于等于
-	数字相减	<=	小于等于
*	数字相乘	==	比较两个对象是否相等
/	数字相除	and	逻辑与
>	大于	or	逻辑或
<	小于	not	逻辑非

1. 算术运算符

算术运算符有“+”“-”“*”“/”，这里主要讲一下“+”，因为它除了能进行数字的相加，还能进行字符串的拼接，演示代码如下：

```
a = 'hello'
b = 'world'
c = a + ' ' + b
print(c)
```

上述代码的输出结果如下：

```
hello world
```

2. 比较运算符

比较运算符常用的是“>”“<”“==”。以“<”运算符为例，演示代码如下：

```
score = -10
if score < 0:
    print('该新闻是负面新闻，录入数据库')
```

因为 -10 小于 0，所以输出结果如下：

```
该新闻是负面新闻，录入数据库
```

需要注意的是，不要混淆“==”和“=”。“=”的作用是给变量赋值，如前面讲过的 a=1。而“==”的作用则是比较两个对象（如数字）是否相等，演示代码如下：

```
a = 1
b = 2
if a == b:   # 注意这里是两个等号
    print('a和b相等')
else:
    print('a和b不相等')
```

此处 a 和 b 不相等，所以输出结果为：

```
a和b不相等
```

3. 逻辑运算符

逻辑运算符主要有 not、and、or。例如，当某条新闻的分数是负数，并且它的年份是 2019 年，才把它录入数据库，演示代码如下：

```
score = -10
year = 2019
if (score < 0) and (year == 2019):
    print('录入数据库')
else:
    print('不录入数据库')
```

这里有两个注意点：第一，and 前后的两个判断条件最好加上括号，虽然有的时候不加也没问题，但是这是比较严谨的做法；第二，year==2019 的逻辑判断式中是两个等号。

因为代码中设定的变量值同时满足分数小于 0 且年份为 2019 年的条件，所以输出结果为：

```
录入数据库
```

如果把代码中的 and 换成 or，那么只要满足一个条件，就可以录入数据库。

1.3　Python 语句

本节主要介绍条件语句、循环语句及异常处理语句。其中，条件语句和循环语句是编程语言很重要的知识点，因为涉及编程语言的一些底层逻辑——判断和循环；异常处理语句则可以避免因程序运行异常而导致程序中断。

1.3.1　if 条件语句

代码文件：1.3.1 if条件语句介绍.py

if 条件语句主要用于判断，基本的语法格式如下所示，注意不要遗漏冒号及代码前的缩进。如果条件满足，则执行代码 1；如果条件不满足，则执行代码 2。

```
if 条件:
    代码1
else:
    代码2
```

前面的内容已经多次出现过 if 条件语句，这里再做一个简单的演示，代码如下：

```
score = 85
if score >= 60:
    print('及格')
else:
    print('不及格')
```

因为 85 大于 60，所以输出结果为“及格”。

如果有多个判断条件，可以使用 elif 语句进行处理，演示代码如下：

```
score = 55
if score >= 80:
    print('优秀')
elif (score >= 60) and (score < 80):
    print('及格')
else:
    print('不及格')
```

这里的 elif 是 elseif 的缩写，用得相对较少，了解即可。

1.3.2 for 循环语句

代码文件：1.3.2 for循环语句介绍.py

for 语句的底层逻辑是循环，其常用格式如下所示，注意冒号和缩进。

```
for i in 区域:
    要重复执行的代码
```

演示代码如下：

```
class1 = ['丁一', '王二', '张三']
for i in class1:
    print(i)
```

这里的 i 就是列表里的元素，上述代码的输出结果为：

```
丁一
王二
张三
```

for 后面的 i 只是一个代号，可以换成任何内容，如 j 或一个字符串，只需和要重复执行的代码内容匹配即可。例如，在上述代码中，将 for 后面的 i 换成 j，第 3 行代码就要改成 print(j)。

for 语句还常与 range() 函数合用。range() 函数可创建一个整数列表，它的基本用法如下：

```
a = range(10)  # 从0～9的整数，类似列表切片的“左闭右开”，不包含10
```

其本质是创建一个如下所示的列表：

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

for 语句与 range() 函数结合的演示代码如下：

```
for i in range(3):
    print('hahaha')
```

上述代码的输出结果为：

```
hahaha
hahaha
hahaha
```

需要注意的是，for i in range(3) 的确会循环 3 次，但在 Python 中，第 1 个元素的序号是 0，所以如果输入如下代码：

```
for i in range(3):
    print(i)
```

那么输出的结果是从 0 开始的，也就是 0 ～ 2，如下所示：

```
0
1
2
```

有些读者可能会对 for i in range(3) 中的 i 有些疑惑，刚才的 i 代表列表里的元素，这里的 i 又是什么意思呢？下面给大家做一个总结：

- 对于“for i in 区域”来说，若区域是一个列表，则 i 代表列表的元素；
- 对于“for i in 区域”来说，若区域是一个字典，则 i 代表字典的键名；
- 对于“for i in 区域”来说，若区域是一个 range(n)，则 i 代表 0 ～ n-1 的 n 个整数。

1.3.3　while 循环语句

代码文件：1.3.3 while循环语句介绍.py

while 语句的底层逻辑也是循环，其使用格式如下所示，注意冒号及缩进。

```
while 条件:
    要重复执行的代码
```

下面举个例子：

```
a = 1
while a < 3:
    print(a)
    a = a + 1  # 也可以写成 a += 1
```

a 一开始等于 1，满足小于 3 的条件，打印输出 1，然后 a 在 1 的基础上加上 1 等于 2；此时 a 仍然小于 3，所以仍会执行打印输出的命令，打印输出 2，然后 a 在 2 的基础上加上 1 等于 3；此时 a 已经不满足小于 3 的条件，循环便终止了。最后输出如下：

```
1
2
```

while 经常与 True 搭配使用，写成 while True 进行永久循环，其基本结构如下所示：

```
while True:
    代码块
```

大家可以试试输入如下代码，体验一下永久循环的效果。

```
while True:
    print('hahaha')
```

如果想停止 while True 的循环，单击 PyCharm 界面右上角的■按钮即可。

1.3.4 try/except 异常处理语句

代码文件：1.3.4 异常处理语句介绍.py

通过 try/except 异常处理语句可以避免因为某一步程序出错而导致整个程序终止，使用方法如下：

```
try:
    主代码
except:
    主代码出错时要执行的代码
```

演示代码如下：

```
try:
    print(1 + 'a')
except:
    print('主代码运行失败')
```

根据已经学过的知识，print(1 + 'a') 这行代码是会报错的，因为数字和字符串不可以直接相加。那么使用 try/except 之后，try 这一部分代码出错后就会跳转到 except 部分执行相应的代码，即 print(' 主代码运行失败 ')，最后输出的内容为：

```
主代码运行失败
```

在具体项目实战中，也常常会用到 try/except 异常处理语句。注意不要过度使用 try/except 异常处理语句，因为有时需要利用程序的报错信息来定位出错的地方，从而进行程序调试。

1.4　函数与库

本节将介绍编程中比较重要的两个知识点：函数与库。通过函数与库，可以避免很多重复性及复杂的工作。所谓函数，就是把具有独立功能的代码块组织为一个小模块，在需要的时候可以反复调用。函数分为编译器自带的函数（内建函数）和用户自己创建的函数（用户自定义函数）。内建函数是编译器开发者已经定义好的函数，用户可以直接使用，如 print() 函数；自定义函数是用户按照需求自己编制并定义的函数。当各种函数很多的时候，开发者会把函数分组，分别放到不同的文件里，这样每个文件包含的代码就相对较少而且可以是同一类函数，这个文件就称为库（也叫模块）。

总之，函数和库的作用就是将一些常用的代码封装好，用户需要实现相应功能时就不必重复写代码，而是直接调用即可。

1.4.1　函数的定义与调用

代码文件：1.4.1 函数的定义与调用.py

自定义函数的格式如下，用 def 来定义一个函数，注意冒号及缩进。

```
def 函数名(参数):
    代码
```

用一元一次函数 y=x+1 来演示一下 Python 函数的写法：

```
def y(x):
    print(x+1)
y(1)
```

第 1 行和第 2 行代码定义了一个函数 y(x)，第 3 行代码调用 y(x) 函数，其中将 1 赋值给 y(x) 函数的参数 x。输出结果如下所示：

```
2
```

调用函数很简单，只要输入函数名，如函数名 y，如果函数含有参数，如函数 y(x) 中的 x，那么在函数名后面的括号中输入相关参数即可。上面代码第 3 行的 y(1) 就是表示 y(1)=1+1，如果第 3 行代码换成 y(2)，那么最后输出的结果就是 3。

函数参数只是一个代号，可以换成其他内容，例如，可以把其中的 x 换成 z，代码如下：

```
def y(z):
    print(z+1)
y(1)
```

定义函数时也可以传入两个参数，以数学中的二元函数 y(x,z)=x+z+1 为例，演示代码如下：

```
def y(x, z):
    print(x+z+1)
y(1, 2)
```

此时调用函数的时候就得输入两个参数，如上面的 y(1,2)，最后运行结果如下：

```
4
```

定义函数时也可以不要参数，代码如下：

```
def y():
    x = 1
    print(x+1)
y()  # 调用函数
```

前 3 行代码定义了一个函数，第 4 行代码调用了这个函数，在定义这个函数时并没有要求输入参数，所以直接输入 y() 就可以调用函数了，运行结果如下：

```
2
```

1.4.2 函数的返回值与作用域

代码文件：1.4.2 函数的返回值与作用域.py

1. 返回值

函数的返回值用得相对较少，读者可以先初步了解。把 1.4.1 小节开头的代码稍微修改一下，把 print 改成 return，其他内容不变，得到如下代码：

```
def y(x):
    return(x+1)
y(1)
```

再次运行会发现没有任何输出结果，这是因为 return 语句和 print() 函数不同。通俗地说，return 类似看不见的 print() 函数，它把原来该由 print() 输出的值赋值给了 y(x) 函数，更严谨的说法就是该函数的返回值为 x+1。稍稍修改代码就可以把 y(x) 的值显现出来，代码如下：

```
def y(x):
    return(x+1)
```

```
a = y(1)
print(a)
```

y(1) 已经是一个有数值的内容了，只是不能直接看到，将其赋值给变量 a，便可以利用 print() 函数将其打印出来，当然也可以直接写 print(y(1))。

上面这 4 行代码的输出结果如下：

```
2
```

在实际应用中，return 后面一般不加括号，例如，上面的代码一般直接写成 return x+1。

有读者可能会感到疑惑：有了 print() 为什么还要用 return ？这是因为 print() 仅仅是将函数执行结果打印在控制台，之后就无法再使用了，而 return 则是将 return 后面的部分作为函数的输出结果，即函数的返回值，之后可以赋值给别的变量，继续使用该返回值做其他事。

2. 变量的作用域

在函数内使用的变量和函数外的代码是没有关系的，演示代码如下：

```
x = 1
def y(x):
    x = x + 1
    print(x)
y(3)

print(x)
```

单纯看上面几行代码，大家思考一下最后会输出什么内容呢？先来看看输出结果：

```
4
1
```

同样是 print(x)，为什么打印出来的内容不一样呢？这是因为函数 y(x) 里面的 x 和外面的 x 没有关系。之前讲过，可以把 y(x) 换成 y(z)，代码如下：

```
x = 1
def y(z):
    z = z + 1
    print(z)
y(3)

print(x)
```

再看上面的代码，大家应该会更容易明白，这样输出的内容肯定就是 4 和 1 了。这个 y(z) 中的 z 或者说 y(x) 中的 x 只在函数内部生效，它不会影响外部的变量。正如前面所说，函数的形式参数只是个代号，属于函数内的局部变量，因此不会影响函数外部的变量。

1.4.3 常用基本函数介绍

代码文件：1.4.3 常用基本函数介绍.py

下面介绍 Python 中常用的一些函数，它们在之后的项目实战中用得也很多。

1. str() 函数与 int() 函数

str() 函数用于将数字转换成字符串，在进行字符串拼接的时候经常用到，演示代码如下：

```
score = 85
print('A公司今日评分为' + str(score) + '分。')
```

int() 函数用于将字符串转换成数字，演示代码如下：

```
a = '85'
b = int(a) - 10  # 通过int()函数转换后才能进行数值计算
```

2. len() 函数

len() 函数可以用于统计列表元素个数，演示代码如下，运行结果为 3。

```
title = ['标题1', '标题2', '标题3']
print(len(title))
```

len() 函数在实战中常和 range() 函数一起使用，演示代码如下：

```
title = ['标题1', '标题2', '标题3']
for i in range(len(title)):
    print(str(i+1) + '.' + title[i])
```

其中，range(len(title)) 表示 range(3)，所以这里 for 循环语句中的 i 表示的是 0、1、2 这 3 个整数，因此在生成标题序号时要写成 i+1，并通过上面提到的 str() 函数转换成字符串，再进行字符串拼接，最终运行结果如下：

```
1.标题1
2.标题2
3.标题3
```

len() 函数还可以统计字符串的长度，演示代码如下，输出结果为 10，即 10 个字符。

```
a = '123华小智abcd'
print(len(a))
```

3. replace() 函数

replace() 函数主要用于替换指定内容，一般格式为：字符串 .replace(旧内容 , 新内容)。演示代码如下：

```
a ='<em>阿里巴巴</em>电商脱贫成“教材”'
a = a.replace('<em>','')
a = a.replace('</em>','')
print(a)
```

上述代码的运行结果如下：

```
阿里巴巴电商脱贫成“教材”
```

4. strip() 函数

strip() 函数主要的作用是删除空白字符（包括换行符“\n”和空格字符“ ”），一般格式为：字符串 .strip()。演示代码如下：

```
a ='          华能信托2018年上半年行业综合排名位列第5          '
a = a.strip()
print(a)
```

运行上述代码，就可以将字符串中多余的空格删除，结果如下：

```
华能信托2018年上半年行业综合排名位列第5
```

5. split() 函数

split() 函数的主要作用是分割字符串，最后生成的结果为一个列表，一般格式为：字符串 .split(' 分割符 ')。演示代码如下：

```
today = '2019-04-12'
a = today.split('-')
print(a)
```

运行结果如下：

```
['2019', '04', '12 ']
```

如果想调用分割完得到的年份信息或月份信息，可以通过如下代码实现：

```
a = today.split('-')[0]  # 获取年份信息，即分割完得到的第1个元素
a = today.split('-')[1]  # 获取月份信息，即分割完得到的第2个元素
```

1.4.4 库

代码文件：1.4.4 Python库介绍.py

库（也称为模块）是 Python 这些年发展如此迅猛的一个原因，因为很多优秀的 IT 工程师在研发出非常棒的代码之后，愿意把它共享给大家使用，而存储这些共享代码的地方就称为库。有的库是 Python 自带的，有的库则需要用户自己安装。编程时，在代码中使用库之前需要引用库，引用库的两种常见方法如下：

```
import 库名
from 库名 import 库里的一个功能
```

引用完库之后就可以使用库里面的功能。用如下例子来演示库的使用：如果想让 Python 输出当前的时间，那么引用 time 库即可，这个库是 Python 自带的，不需要安装。

```
import time
print(time.strftime("%Y-%m-%d"))
```

上述代码就可以输出当天的时间，运行结果如下：

```
2018-12-11
```

从 datetime 库里引用 datetime 功能也能实现同样的效果。前一个 datetime 是库名，后一个 datetime 可以理解成功能，然后使用 datetime 功能的 now() 函数获取时间，代码如下：

```
from datetime import datetime
print(datetime.now())
```

输出的时间格式为：

```
2018-12-11 14:55:26.562000
```

也可以使用如下代码，输出结果一样。

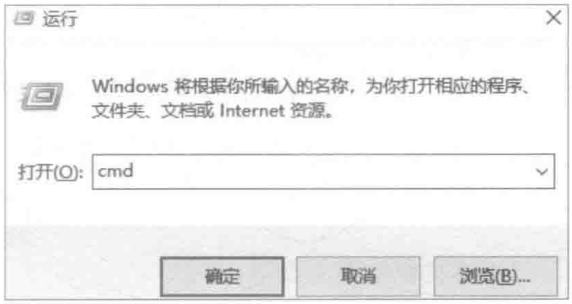
```
import datetime
print(datetime.datetime.now())
```

在项目实战中，如网络数据挖掘领域，经常会用到 Requests 库，它是通过 Python 程序访问网站的基础。下面就以安装 Requests 库为例介绍安装库的两种常用方法。

1．pip 安装法

pip 安装法是通过命令行来安装，命令格式为：pip install 库名。这里以 Windows 系统为例介绍具体操作。

第一步：按快捷键 Win+R（Win 键即键盘左下角的 Windows 标志键，通常在 Ctrl 键和 Alt 键之间），打开“运行”对话框，输入“cmd”后单击“确定”按钮，如右图所示。



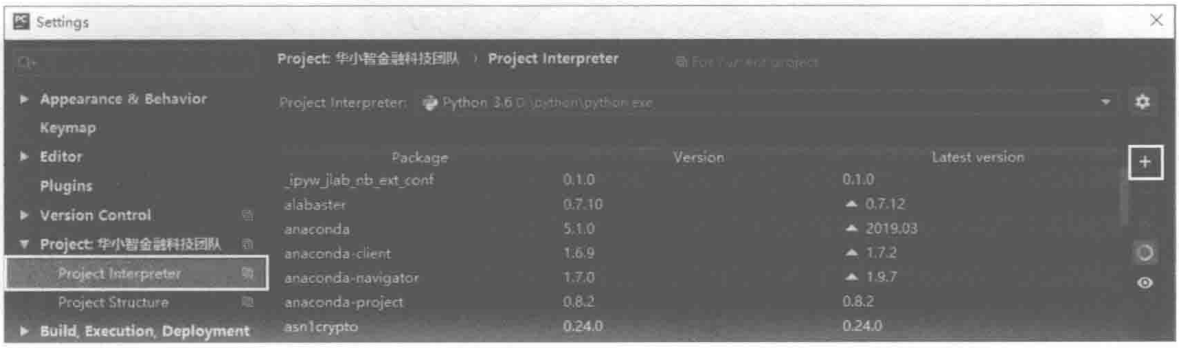
第二步：此时会弹出命令行窗口，输入命令“pip install requests”，如右图所示，然后按 Enter 键，等待安装结束即可。

2．PyCharm 安装法

如果使用的是 PyCharm，可以直接在 PyCharm 中安装库，具体安装步骤如下。

第一步：执行“File>Settings”菜单命令，打开“Settings”对话框。

第二步：展开 Project 选项组，选择“Project Interpreter”选项，单击右侧的加号按钮，如下图所示。



第三步：搜索要安装的库的名字，如 Requests 库，搜索完成后选中要安装的库，单击左下角的“Install Package”按钮，即可进行安装，如下图所示。

相比较而言，PyCharm 安装法更加直观，但如果有些库在 PyCharm 里找不到，用 pip 安装法也很方便。

安装好 Requests 库之后，来小小实战一下吧。输入如下代码：

```
1  import requests
2  url = 'https://www.baidu.com'
3  res = requests.get(url).text
4  print(res)
```

上述代码的含义如下：

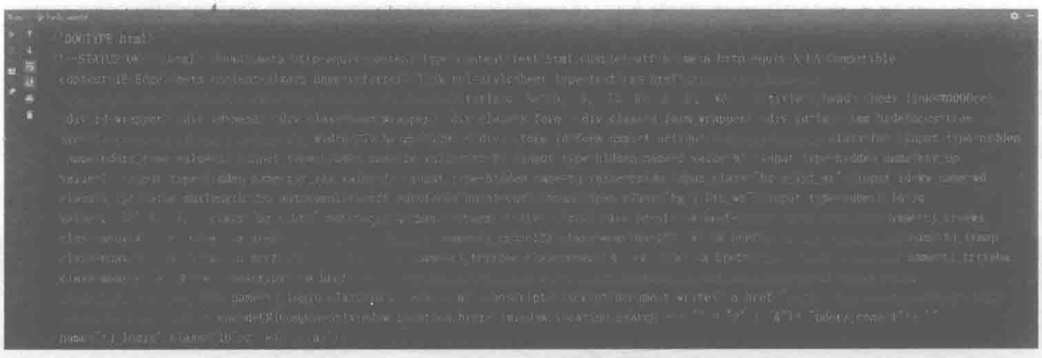
第 1 行：引用 Requests 库；

第 2 行：将一个网址赋值给变量，注意不要只输入“www.baidu.com”，而要输入完整的网址“https://www.baidu.com”；

第 3 行：通过 Requests 库的 get() 函数访问该网址，通过 .text 获取网页源代码的文本内容；

第 4 行：打印输出获取的网页源代码。

上述代码的运行结果如下图所示。



可以看到获取到的内容还是比较粗糙的，有很多需要改进的地方，例如，里面的中文出现了乱码，以及获取到的内容并不完整，但不必着急，之后会详细讲解如何获取完整的内容。此时的重点在于，通过简单的 4 行代码就能获取网页上的信息。如果把网址由“https://www.baidu.com”换成 Python 官网的地址“https://www.python.org”，爬取到的内容则完整得多（因为是英文网站，所以没有乱码现象），感兴趣的读者可以自己试一下。之后爬取百度新闻、搜狗新闻、新浪财经等都是一样的原理，所以大家要树立起数据挖掘并不难学的信心。