项目 1

Python 语言概述及安装、配置

本项目首先初步介绍 Python 语言，包括它的起源、发展前景和优缺点，帮助读者对 Python 有一个初步的认识；然后介绍在不同系统下如何下载和安装 Python，并使用不同的方法进行 Python 的编程练习；最后对如何规范代码和如何使用帮助和文档进行说明。

1.1 任务 1 认识 Python 语言

Python 是一门近年来流行的编程语言，读者可能会有诸多疑问：Python 有什么与众不同的地方？学习它之后我们能干什么？等等。本节致力于回答这些问题。当然，完整地学习本书之后，读者一定会对答案有更深刻的认识。

1.1.1 Python 的起源和发展前景

Python 是由荷兰人吉多 • 罗萨姆于 1989 年发布的。作为 Monty Python 飞行马戏团的狂热爱好者，吉多 • 罗萨姆选择了 Python 作为这门语言的名字。Python 的第一个公开发行版发行于 1991 年。

Python 的官方定义（https://www.python.org）：Python 是一种解释型的、面向对象的、带有动态语义的高级程序设计语言。通俗来讲，Python 是一种少有的、既简单又功能强大的编程语言，它注重的是如何解决问题而不是编程语言的语法和结构。

根据 TIOBE 排行榜（https://www.tiobe.com/tiobe-index），Python 的使用率从 2001 年后呈线性增长。2018 年 9 月，Python 在 TIOBE 排行榜的排名是第三，如图 1-1 所示。TIOBE 排行榜每月更新一次，依据的指数由世界范围内的资深软件工程师和第三方供应商提供，其结果作为当前业内程序开发语言的流行使用程度的有效指标。现在全世界差不多有 600 多种编程语言，但流行的编程语言也就二十多种。在 TIOBE 排行榜能跻身前三，这在一定程度上说明该编程语言的实用性与强大性，更关键的是该语言与当今 IT 发展的契合度。

Sep 2018	Sep 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.436%	+4.75%
2	2		C	15.447%	+8.06%
3	5	↑	Python	7.653%	+4.67%
4	3	↓	C++	7.394%	+1.83%
5	8	↑	Visual Basic .NET	5.308%	+3.33%
6	4	↓	C#	3.295%	-1.48%
7	6	↓	PHP	2.775%	+0.57%
8	7	↓	JavaScript	2.131%	+0.11%
9	-	⇈	SQL	2.062%	+2.06%
10	18	⇈	Objective-C	1.509%	+0.00%
11	12	↑	Delphi/Object Pascal	1.292%	-0.49%
12	10	↓	Ruby	1.291%	-0.64%
13	16	↑	MATLAB	1.276%	-0.35%
14	15	↑	Assembly language	1.232%	-0.41%
15	13	↓	Swift	1.223%	-0.54%

图 1-1　TIOBE 排行榜

（https://www.tiobe.com/tiobe-index）

Python 的应用非常广泛，很多公司都在使用 Python。例如：Google 用它实现网络爬虫（一种按照一定的规则，自动地抓取万维网信息的程序或者脚本）和搜索引擎中的很多组件；Intel、Cisco 、IBM 等知名企业使用 Python 进行硬件测试；许多大型网站就是用 Python 开发的，如国外知名的 YouTube、Instagram，以及国内的豆瓣、知乎等；微信公众号也支持 Python 语言；经济市场预测领域、高科技含量领域等都有 Python 语言的身影及 NASA（美国航空航天局）也大量地使用 Python，不仅用于主程序开发，也用做脚本语言。

Python 在科学计算领域也有着重要地位，Python 在科学计算库方面有着近乎完美的生态系统，集成了用 C 与 Fortran 写的经过高度优化的代码而显示出的极佳性能等优势，这些优势使其在科学计算中有优秀的表现。在日渐火热的人工智能领域，Python 也是崭露头角。例如，对于深度学习，在 Python 下有许多知名的第三方库：TensorFlow、Theano、Keras、PyTorch 等。你只要有相关的背景知识，加上 Python 基本语法，就能做深度学习的相关应用。

学习 Python 以后的发展也是多元化的，你可以持续发展成为 Python 开发工程师、自动化开发工程师、前端开发工程师、运维工程师、大数据分析和数据挖掘工程师、数据研发工程师等。可以说，Python 带来无限可能。

1.1.2　Python 的优缺点

首先需要指出的是 Python 也有我们熟悉的东西。类似于其他通用编程语言，Python 同样有语句、表达式、操作符、函数、模块、方法和类。另外，Python 还有许多优势，这些优势让 Python 大放异彩。

1. Python 的优点

Python 的优点可总结为以下几点。

（1）语言简洁

Python 是一种代表简单主义思想的语言。吉多 • 罗萨姆对 Python 的定位是“优雅，明确，简单”。Python 拒绝了“花俏”的语法，而选择明确的没有或者很少有歧义的语法，着重解决问题。Python 开发者的哲学是“用一种方法，最好是只有一种方法来做一件事”，这样的开发

思维使得编程语言既简洁又强大。例如，完成同一个任务，C 语言要写 1000 行代码，Java 只需要写 100 行，而 Python 可能只要 20 行。

Python 语言的简洁对大多数学习者、开发者来说都是喜闻乐见的。对学习者来说，Python 的代码是很“直白”的，非常容易懂，也非常容易学习。对开发者来说，比较直观的语言有两个好处：一是加速了开发速度，用比较少的语言就能够完成想要的结果；二是强化了可读性，相应地提高了代码的可重用性和可维护性。实际上，优秀的程序员知道，代码是为下一个会阅读它而进行维护或重用的人写的。如果那个人无法理解代码，在现实的开发场景中，则代码毫无用处。

（2）丰富的基础代码库

基础代码库称作 Python 的“内置电池”。Python 具有丰富和强大的库来被调用。用 Python 开发，许多功能不必从零编写，直接使用现成的即可。当用一种语言开始进行真正的软件开发时，除了编写代码外，开发者还需要很多基本的已经写好的现成的东西，以帮助其加快开发进度。Python 3.0 有 70 多种内置功能函数 BIF 和大量预加载内置模块，覆盖了网络、文件、图形用户界面（Graphical User Interface，GUI）、数据库、文本等大量内容，这些已经能够帮助快速处理常见的基本需求了。

例如，要编写一个电子邮件客户端，如果先从最底层开始编写网络协议相关的代码，则估计需要很长时间。高级编程语言通常都会提供一个比较完善的基础代码库，让开发者直接调用，如针对电子邮件协议的 SMTP 库、针对桌面环境的 GUI 库，在这些已有的代码库的基础上开发，一个电子邮件客户端几天就能开发出来。

Java 和 C 等也有不错的库以供使用，然而对这些程序设计语言来说，最大的问题是即使完成简单的操作也要编写大量的代码。为了完成一个简单的工作，我们必须花费大量时间编写很多无用、冗长的代码。Python 则不同，Python 调用它的库是非常简单的，这一点归功于 Python 的第一个特点——语言简洁。

除此之外，Python 还有一个强大的后援——PyPI（https://pypi.python.org/pypi）。PyPI 是第三方 Python 模块集中存储库，可以把它当成一个社区或论坛，其界面如图 1-2 所示。当你的需求在内置模块中找不到时，你可以求助于 PyPI。全世界的 Python 用户都可以上传他们的模块以供分享，可以想象它的强大。当然你也可以上传你的模块供全世界使用，这确实是一件很有成就感的事情。

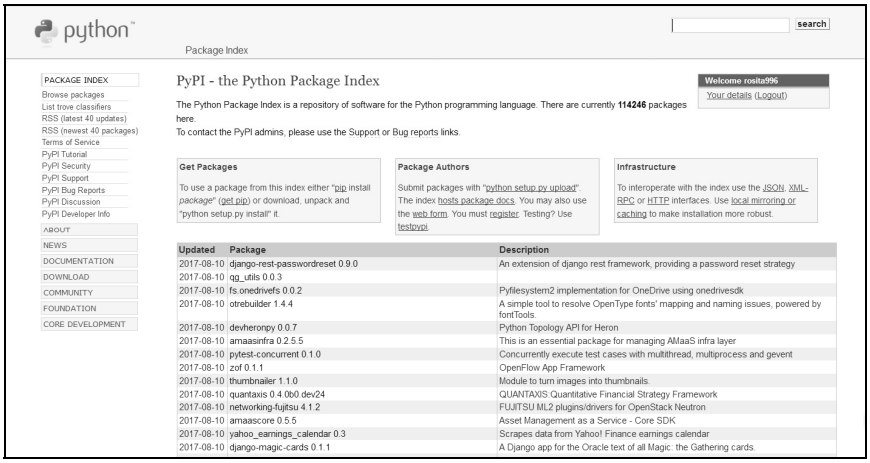

图 1-2　PyPI 社区

（来源：https://pypi.python.org/pypi?:action=login）

（3）可扩展性

这个特性常常为 Python 爱好者津津乐道，Python 经常用于将不同语言（尤其是 C/C++）编写的程序“粘”在一起，即 Python 的很多模块或者组件都是用其他语言写的，而 Python 的一个功能就是把这些模块很轻松地联结在一起。所以，人们也常常称 Python 为“胶水语言”。

常见的一种应用情形是，使用 Python 快速生成程序的原型（有时甚至是程序的最终界面），然后对其中有特别要求的部分，用更合适的语言改写。例如，3D 游戏中的图形渲染模块，性能要求特别高，就可以用 C/C++重写，而后封装为 Python 可以调用的扩展类库。Python 本身被设计为可扩充的。Python 提供了丰富的 API 和工具，可以使用户避免被“过分”的语法羁绊而将精力主要集中到所要实现的程序任务上。例如，在现实开发需求中，我们想要做系统调用或者组件集成的时候，可能第一时间就想到 Python。我们也可为现成的模块加上 Python 接口，使其发展成为 Python 可调用的。在这一点上，我们常用的 C、C++、Java 等均不能达到同样的效果。

（4）开源

Python 是自由/开源软件，使用者可以自由地发布这个软件的副本，阅读它的源代码，对它做改动，把它的一部分用于新的自由软件中。

基于团体分享知识的理念，Python 在一群人的不断创造和改进中持续变得更加优秀。

（5）跨平台性

在 20 世纪 90 年代，操作系统比较单一，跨平台性并没有那么重要，但是随着如 Linux、Mac OS 等操作系统的出现，跨平台几乎成为各大企业的主要需求之一。而一个软件若想要在每个平台上发布，需要在每个平台上都做开发，这无疑太难了，并且投入的成本巨大。所以，可跨平台的编程语言是市场的需求，也是时代的需要。

Python 的跨平台性可总结为“一次编写，到处运行”。所有 Python 程序无须修改就可以在下述任何平台上面运行：Linux、Windows、FreeBSD、Macintosh、Solaris、OS/2、Amiga、AROS、AS/400、BeOS、OS/390、z/OS、Palm OS、QNX、VMS、Psion、Acom RISC OS、VxWorks、PlayStation、Sharp Zaurus、Windows CE、PocketPC 等。Python 是跨平台的，和 Java 相似，主要是源码跨平台，编译之后不一定能跨平台。Java 要装虚拟机，Python 要装编译运行环境。

由于 Python 的开源本质，任何人任何企业都可以自由使用 Python，Python 已经被移植在许多平台上。反过来，Python 本身的优势和对多平台的支持也使得 Python 的应用越来越广泛。

（6）面向对象

Python 既支持面向过程的编程，也支持面向对象的编程。

（7）可嵌入型

Python 可以嵌入到 C/C++程序，从而向程序用户提供脚本功能。

（8）解释型语言

Python 是解释型语言，这使得 Python 语言简单且易于移植（只需要把 Python 程序复制到另一台计算机上，它就可以进行了）。

2. Python 的缺点

当然，作为众多编程语言中的一种，Python 也有不可避免的缺点，Python 的主要缺点有以下两点：

（1）运行速度相对较慢（较 C、C++而言）

Python 是解释型语言，其代码在执行时会一行一行地翻译成计算机能理解的机器码，这个翻译过程非常耗时，所以运行速度很慢。相比较而言，C 程序（编译型语言）是在运行前直接编译成计算机能执行的机器码，所以运行速度非常快。

但是，Python 节省下来的大量的编程时间和维护时间足以弥补这一缺点，快速开发意味着更容易抓住商机，而且在多数情况下，程序员的人工成本比硬件设备要大。

目前大量的应用都是 I/O 密集型负载，而并非 CPU 密集型负载。虽然 Python 的执行效率比较低，但它毕竟是在 CPU 上执行的。我们知道 CPU 比 I/O 设备快许多个数量级，因此对于 I/O 密集型负载，系统更多的时候是在等待 I/O 操作而并非程序本身的执行过程，在这种情况下，使用 C 或 Python 并没有太显著的区别。

Python 也有很多手段可以提高运行速度。例如，Python 采用 PyPy[一种使用实时（Just-In-Time，JIT）技术的 Python 编译器]和调用 C 扩展能够在很大程度上提高速度。Python 用户常常提的一点是：Python 并不慢，因为在实际的应用中，如果你认为程序跑得还不够快，可以“粘”一段 C 程序在关键处理上，如对内存的读取、排序等，这样能够同时兼顾开发和速度。因为 Python 运行时调用了大量 C 库，而 C 是很快的。

反过来想想，这正反映了其“胶水语言”的事实。至今还没有一种高级语言，开发速度比 Python 快，运行速度比 C 快，我们在实际应用中可以同时利用 C 和 Python 的优点。也就是说，在实际开发中，不同工具通过合适的结合和使用，是可以相互提升和弥补缺陷的。

（2）代码不能加密

一般来讲，发布 Python 程序，实际上就是发布源代码。这一点跟 C 语言不同，C 语言不用发布源代码，只需要把编译后的机器码（可以直接运行的程序，如 Windows 上常见的 EXE 文件）发布出去。要从机器码反推出 C 代码是不可能的，所以，编译型语言都没有这个问题，而对于解释型的语言，则必须把源码发布出去。当然，靠卖软件授权的商业模式已经一去不返了，现在更多的是靠网站和移动应用卖服务的模式，后一种模式不需要把源码给别人。

总地来说，Python 是一种想让你在编程实现自己想法时不那么“碍手碍脚”的程序设计语言，你可以花较少的代价实现想要的功能，并且编写的程序清晰易懂。Python 的优势不在于运行效率，而在于开发效率和高可维护性，而不管在任何领域，质量和效率往往都是人们比较关注的问题，尤其是对企业来说，敏捷开发正是降低成本较有效手段之一，这也是人们选择 Python 的主要原因之一。在实际使用 Python 时，要注意有的放矢，扬长避短。例如，当我们对运行速度要求比较高时，慎用 Python，而考虑相对有优势的 C 或者 Java 等，而当要求高开发速度，或者需要多调用其他语言模块的时候，Python 可能就是比较好的选择了。所以，针对特定的问题选择特定的工具也是一项技术能力。

1.1.3　Python 与云计算

当今，云计算在我国发展得如火如荼，其应用的领域也是方方面面。Python 作为一门研究云计算的热门语言一直受到广泛关注。实际上，云计算可以采用许多不同的途径与方法来实现，但是我们需要选择最佳的方案。

2010 年 7 月，基于 Python 的开源云平台 OpenStack 问世。在整个 IT 领域，这是一件大事情。从此，云计算就有了允许任何人（不限国籍）均可自由使用，而且品质优秀的软件包，人们不用

再为云计算烦恼了。实际上，这种所谓的“自由云软件”非常复杂，涉及许多外在因素，是一种超大型的“应用软件”，需要具备极高的“灵活性”。选择什么样的编程语言来编写这种“云软件”就成了一个问题，于是 Python 因其是当今世界上最灵活、易用的模块化编程语言而脱颖而出。

随着 Python 在云计算方面的优势展现出来，支持 Python 的云平台也层出不穷，最典型的就是各种 PaaS（Platform as a Service，平台即服务）产品，如 Google App Engine（GAE）、Sina App Engine（SAE）、Baidu App Engine（BAE）、DeployFu、PiCloud、DjangoZoom、Nuage、DotCloud、Pydra 等。

1.2 任务 2 下载和安装 Python

1.2.1 Python 版本差异

在下载和安装 Python 之前，先来看看目前主流的版本，以及它们之间的差异。Python 有两大版本，即 Python 2 和 Python 3，目前业界正处在从 Python 2 向 Python 3 转移的阶段。

在 Python 2 家族中，次版本号为 4，即 Python 2.4，这个版本曾得到广泛应用。Python 3 对 Python 2.4 及更早版本的兼容性并不好，很多用 Python 2.4 写出来的代码，不被 Python 3.0 支持。所以后面又开发出了 Python 2.6 以部分兼容 Python 3.0，也就是说 Python 2.6 比 Python 3.0 推出的时间要晚。

Python 2.6 及以后的版本完全兼容 Python 2.4，并且继承了很多 Python 3.0 的新特性。例如，对于打印输出，可以使用 print 语句，也可以使用 print()函数，后者是 Python 3.0 的特性。

Python 2.7 是 Python 2 家族的最后一个版本，该版本稳定，文档齐全，第三方库和其他工具非常丰富。本书大部分例题使用 Python 2.7 实现，但是我们也会在许多关键的地方告诉读者 Python 3.X 的特性，并对书中的许多例题同时提供 Python 3.X 的代码。此外，本项目还会介绍一个名为 2to3 的脚本，它能自动将 Python 2 代码翻译成合法的 Python 3.X 代码。

1.2.2 Python 虚拟机简介

通常意义上说的 Python 是 CPython，即完全用 C 语言实现的 Python，它支持 C 的扩展。Python 解释器“编译”Python 源代码，生成对应的字节码文件.pyc，字节码随后在 CPython 虚拟机上执行。有时候，我们因为看到.pyc 文件而认为 Python 是编译型的，这也有一些合理性。如果之前运行过的 Python 代码，并生成了.pyc 文件，再次运行时速度明显要快得多，因为这次不需要再次编译生成字节码了。

Python 有很多实现。CPython 是最通用的，被认为是“默认”的实现，同时是其他虚拟机实现的参考解释器。

除了 CPython，较著名的就是 Jython 了，Jython 是完全用 Java 实现的 Python。CPython 生成在 CPython 虚拟机上运行的字节码，而 Jython 生成在 JVM 上运行的 Java 字节码（这同编译 Java 程序生成 Java 字节码的过程是一样的）。Jython 具有许多 Java 特性，如垃圾回收机制。

利用 CPython 很容易为 Python 代码写 C 扩展，因为最终都是由 C 解释器执行的。另外，Jython 和其他 Java 程序共同工作很容易：不需要做其他工作，就可导入任何 Java 类，并在 Jython

程序中使用其他 Java 类。

IronPython 是另一个很流行的 Python 实现，完全用 C#实现，面向.NET 平台。它运行在.NET 虚拟机的平台上，这是 Microsoft 的公共语言运行时（Common Language Runtime，CLR)，同 JVM 相对应。

PyPy 是一个用 Python 写的 Python 的实现，这听起来有点儿像一个奇怪的循环。PyPy 使用 JIT 技术来提高运行速度。我们知道，本地机器码的运行速度比字节码的运行速度快很多，那么，如果能将一些字节码直接编译成本地机器码再去运行它会怎样呢？虽然必须花费一些代价（如时间）来将字节码编译为本地机器码，但如果最终的运行速度更快，那么这个代价就是值得的。这就是 JIT 编译器的动机，即结合了解释器和编译器的优点，简单来讲，JIT 就是想通过编译技术提升脚本解释器系统的速度。

虽然目前 Python 的实现（或者说 Python 虚拟机）很多，但 CPython 仍然有不可替代的优势。我们说过，Python 的应用已经深入到许多不同的领域，有数不清的优秀、便捷的第三方框架可以使用，其中有很多只有基于 CPython 的实现。在本书中，我们只介绍默认的、纯粹的 Python，即 CPython。

1.2.3　下载 Python

Mac OS X 会预安装 Python 2，绝大多数 UNIX/Linux 也会预安装，不同的发行版可能有不同的版本，例如，RHEL（Redhat Enterprise Linux）6/CentOS 5 预安装的是 Python 2.4，而 RHEL 6/CentOS 6 预安装的是 Python 2.6。

Windows 没有内置任何 Python 版本。在 cmd 命令行下输入“python V”可以查询是否安装 Python 及其版本信息，如图 1-3 所示。

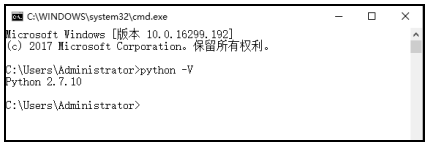

图 1-3　cmd 命令行

访问 Python 的官网 https://www.python.org，在下载页面中单击 downloads 按钮，如图 1-4 所示。根据所用操作系统（Windows、Linux/UNIX、Mac OS X 或者其他）来选择对应的安装程序。有时候该页面会自动获悉用户的操作系统类别，跳转到对应的下载列表。对于 Windows 平台，这里提供了 32 位和 64 位各 3 种安装包：基于 Web 页面安装的可执行安装包、EXE 格式的自解压缩安装包、调用 Microsoft Installer 的 MSI 安装包。对于 UNIX/Linux，推荐的安装方式是下载源代码然后编译安装。

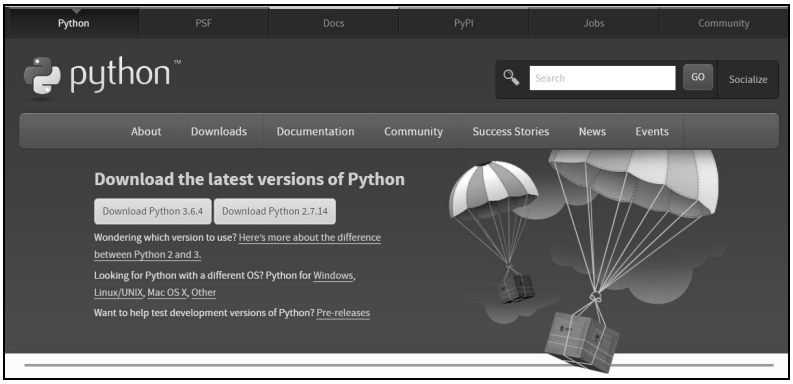

图 1-4　Python 官网上的下载页面

1.2.4 在 Windows 环境下安装 Python

在 Windows 环境下安装 Python 非常简单，我们以 MSI 格式的安装包为例，假设要安装 Python 2.7（主版本号和此版本号之后还有一个修订版本号，所以下载的安装包可能是 2.7.14 或其他数字号）。对于 Windows 32 位系统，选择 Download Windows x86 MSI installer；对于 Windows64 位系统，选择 Download Windows x86-64 MSI installer。我们会得到一个扩展名为 msi 的文件，将它放在计算机的任何位置均可。

下载完毕后，双击运行所下载的文件，开启 Python 安装向导，可以接受默认设置，一路单击“下一步”按钮，直到安装完成。不过，为了省去后面配置环境变量的操作，也可以在安装向导中选择“Add python.exe to Path”选项，如图 1-5 所示。

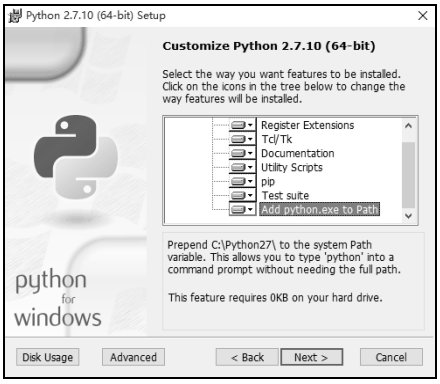

图 1-5　安装向导

若一切正常，在 Windows 的“开始”菜单即可找到 IDLE（Python GUI），即 Python 集成开发环境（Python Integrated Development Environment），它既简单又非常有用，打开后会有如下信息：

```
Python 2.7.13rc1 (v2.7.13rc1:4d6fd49eeb14, Dec  3 2016, 21:56:35) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

1.2.5 在 Windows 下配置 Python 环境

如果在安装 Python 的时候没有启用“Add python.exe to Path”选项，则需要手动将环境变量添加到系统中。这样做的目的是为后面的工作提供便利，以便用户在任何路径下都能用命令行工具（cmd 或 PowerShell）启动 Python 或 Python 上的第三方框架提供的工具。

操作系统其实并不是那么“聪明”，如果不显性地“告诉”它要运行的程序的详细路径，则它就会“犯糊涂”。在没有配置环境变量的情况下，例如，Python 安装在默认位置 C:\Python 2.7\python.exe，当在命令行工具中输入“python”的时候可能得到一个错误提示：“‘python’不是内部或外部命令，也不是可运行的程序或批处理文件。”则这时必须输入完整路径 C:\Python 2.7\python.exe，程序才能如期运行。

配置环境变量能让操作系统稍微“聪明”一点儿，例如，把 C:\Python 2.7\这个文件夹加入

环境变量中，当只输入“python”而不是“C:\Python 2.7\python.exe”时，它就会在指定的位置，即 C:\Python 2.7\下去寻找，这样就能找到 python.exe 了。

当环境变量中有多个路径时，系统会遍历所有的路径，依次寻找。所以，我们要添加 3 个目录：

```
C:\Python 2.7
C:\Python 2.7\Scripts
C:\Python27\Lib\site-packages
```

第一个路径是 Python 主程序所在的文件夹，其他两个路径是常用的脚本和第三方框架提供的工具。右击“计算机”（Windows 10 中称为“此电脑”）图标，在弹出的快捷菜单中选择“属性”命令，在打开的“系统属性”对话框中选择“高级”选项卡，再单击“环境变量”，然后在“系统变量”列表中找到“path”，双击打开，将前面列举的 3 个目录添加到列表中，用分号隔开。注意不要删除原先的环境变量。

注意：在本书中，有时候使用文件夹（Windows）这一名词，有时候使用目录（UNIX/Linux），这取决于使用的平台，但它们本质上是一样的。

1.2.6　在 Linux/UNIX 下使用 Python 源代码安装 Python

绝大多数 Linux 和 UNIX 会预安装 Python，读者可以在 Shell 中输入“python”进行验证，运行这个命令会启动交互式 Python 解释器，同时会有如下输出：

```
Python 2.7.13rc1 (v2.7.13rc1:4d6fd49eeb14, Dec  3 2016, 21:56:35) [GCC 4.0.1(Apple Computer.Inc.build 5367)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

注意：要退出交互式解释器，可以使用快捷键 Ctrl+D。如果没有安装 Python，则可能会看到类似于下面的错误信息：

```
bash: python: command not found
```

1. 使用包管理器

Linux 有多种包管理系统和安装机制。如果使用的操作系统是包含某种包管理器的 Linux，那么可以很轻松地安装 Python。

注意：在 Linux 中使用包管理器安装 Python 可能需要系统管理员（root）权限。

例如，如果使用的操作系统为 Debian/Ubuntu，则应该可以用下面的命令来安装 Python：

```
apt-get install python
```

如果使用的操作系统 Gentoo Linux，则使用 emerge：

```
emerge python
```

如果使用的操作系统 RHEL/CentOS，则使用 yum：

```
yum install python
```

注意：许多包管理器有自动下载的功能，包括 YUM、Synaptic（Ubuntu Linux 专有的包管

理器）及其他 Debian 样式的管理器。通过这些管理器可以获得最新版本的 Python。

2. 从源文件编译

使用源代码编译方式安装的一个很常见的原因是，Linux/UNIX 版本已经自带了某个特定的 Python 版本，但不符合用户需求。例如，RHEL/CentOS 6.x 自带了 Python 2.6，而用户需要的开发环境是 Python 2.7。不能粗暴地删掉已有的 Python 版本，因为有些 Linux 自带的软件包依赖它，如 YUM。所以，多版本的 Python 必须共存。

通过编译源代码的方式安装 Python 之前，需要先安装一些软件包。

1）readline-devel：如果不安装它，则在 Python 交互式解释器中，用户可能无法正常使用 Backspace（退格）键。

2）zlib、zlib-devel：它们是 Python 的一些有用的工具依赖的包，如 easy_install 和 pip。

以 RHEL/CentOS 为例，安装软件包的方法很简单，使用以下命令即可：

```
yum install readline-devel zlib zlib-devel
```

接下来，设法获取所需要的 Python 版本的源代码文件。可以通过浏览器访问下载网页（参见在 Windows 上安装 Python 的步骤），然后下载合适的 Python 版本的源代码；或者，通过浏览器获取源代码的 URL，然后在 Linux 命令行使用 wget 命令：

```
wget -P ~/python_source_code https://www.python.org/ftp/python/2.7.13/Python-2.7.13.tgz
```

参数 P 表示下载到指定的目录，符号“～”表示用户主目录。如果不指定目录，则默认下载到用户主目录。下载完成后通过 tar 命令来解压缩：

```
tar -xzvf Python-2.7.13.tgz
```

如果使用的 tar 版本不支持-z 选项，则可以先使用 gunzip 进行解压缩，然后使用 tar -xvf 命令。解压缩完成后，进入解压缩的文件夹：

```
cd Python-2.7.13
```

现在可以执行下面的命令，以配置安装路径和其他安装选项：

```
./configure --prefix=<your_install_path>
```

要查看其他可用的配置选项，可执行以下命令：

```
./configure –help
```

可以把软件安装在其他方便的位置，如用户主目录，但我们推荐使用规范的方法，放在/usr/local/目录下，即

```
./configure --prefix=/usr/local/python27
```

执行完这一步之后，就可以使用 make 和 make install 命令了。也可以使用&&符号来一次完成两条命令，例如：

```
make && make install
```

接下来就是编译过程，编译完成后，创建一个软链接（快捷方式）：

```
ln -s /usr/local/python27/bin/python2.7 /usr/bin/python27
```

这里也可以取一个其他的名字，但最好能够“见名知意”，如 py。

由于/usr/bin/已经位于环境变量中，用户可以通过输入软链接的名字（即 python27），直接运行 Python 2.7，同时，原先旧版本的 Python 及依赖它的其他软件包不会受到影响。

1.3　任务 3　使用开发工具

可以通过 3 种不同的方法来启动 Python。最简单的方法就是使用交互式解释器，通过每次输入一行 Python 代码来执行。另一种启动 Python 的方法是运行 Python 脚本，这样会调用相关的脚本解释器。最后一种方法是用集成开发环境中的图形用户界面运行 Python。集成开发环境通常整合了其他的工具，如集成的调试器、文本编辑器，而且支持各种像 CVS 这样的源代码版本控制工具。

1.3.1　使用交互式解释器

在命令行上启动解释器，可以马上开始编写 Python 代码。在 UNIX/Linux 的 Shell 中，或 Windows 的命令提示符窗口/PowerShell 工具，或其他任何命令行工具中，都可以这么做。启动交互式解释器的方法是直接输入“python”。

在交互式环境的提示符“>>>”下，直接输入代码，按 Enter 键，就可以立刻得到代码执行结果。现在我们来试试著名的“hello world”程序：

```
>>> print "hello world!"        # 从井号开始，直到本行结束的所有字符，均为注释
hello world!
>>>
```

1.3.2　使用文本编辑器

在 Python 的交互式命令行写程序的好处是可以马上得到结果，坏处是结果没法保存，下次想运行的时候，还得重新写程序。所以，在实际开发的时候，我们总是使用一个文本编辑器来写代码，并将其保存为一个文件，这样程序就可以反复运行了。

目前有许多优秀的文本编辑器，UNIX/Linux 平台上有著名的 VIM 和 Emacs，Windows 平台上有 Sublime Text、Notepad++。不推荐使用 Windows 自带的记事本作为文本编辑器，因为它不提供显示行号、语法高亮、代码补全等功能。

下面我们把“hello world”程序用文本编辑器写出来：

```
print "hello world!"            # 注意要顶格写，也就是左边不要留空格
```

我们把这个文件保存为 hello.py，要注意的是，文件名只能是英文字母、数字和下画线的组合。它的扩展名名必须是 py。保存好之后，在命令行中切换到这个文件所在的文件夹，然后调用 Python 解释器执行它：

```
python hello.py
```

在上面这条命令中，“python”是命令本身，后面的“hello.py”是源代码文件的文件名（如果不在当前目录下，需要加上路径），是命令的参数。如果当前目录下没有 hello.py 这个文件，

则运行“python hello.py”命令就会报错：

```
python hello.py
python: can't open file 'hello.py': [Errno 2] No such file or directory
```

Windows 上不能直接运行.py 文件，而在 UNIX/Linux 中是可以的。首先在.py 文件的第一行加上以下内容：

```
#!/usr/bin/env python
```

其作用是告诉操作系统，该脚本用 Python 执行，并且要在/usr/bin/这个位置去找 Python 的可执行文件。接下来还须给它授权，通过以下命令实现：

```
chmod a+x hello.py
```

其作用是允许 hello.py 文件被直接执行，但注意要加上路径名，至少要以相对路径的形式体现，例如：

```
./hello.py          # 符号./表示当前目录
```

1.3.3 使用集成开发环境

使用集成开发环境（IDE）有很多好处，除了具备代码补全和语法高亮等功能外，常见的 IDE 还支持项目管理、代码跳转、代码分析、断点执行、Debug 等功能。Python 在 Windows 平台上有一个自带的、轻量级的 IDE，叫作 IDLE，安装好 Python 之后就可以使用它了。

作为 IDE，IDLE 的功能有些简单。在 Windows 下，我们推荐 PyCharm。PyCharm 带有一整套可以帮助用户在使用 Python 语言开发时提高其效率的工具，如调试、语法高亮、项目管理、代码跳转、智能提示、自动完成、单元测试、版本控制等。此外，该 IDE 提供了一些高级功能，用于支持 Django 框架下的专业 Web 开发；同时支持 GAE 和 IronPython。

PyCharm 提供收费的专业版和免费的社区版，可以访问它的主页以下载安装包：

```
http://www.jetbrains.com/pycharm/download/
```

当然，也可以使用 Eclipse 或者 Microsoft Visual Studio。

在 Linux 下，推荐使用 VIM 或 Eclipse，如果使用 Eclipse 则还需要安装 PyDev。

1.3.4 使用 Python 增强工具

除了 IDE 之外，Python 还提供了一些增强工具，下面简单介绍一些常用增强工具。

1. easy_install 和 pip

easy_install 和 pip 均用来下载安装 Python 的一个公共资源库 PyPI 的相关资源包。pip 是 easy_install 的改进版，提供更好的提示信息、删除包等功能。使用 easy_install 和 pip 来安装第三方包非常方便，使用如下命令即可：

```
easy_install install <packageName>
# 或者:
pip install <packageName>
```

2. anaconda

anaconda 指的是一个开源的 Python 发行版本，其包含了 conda、Python 等多个科学包及其依赖项。anaconda 可以看作 Python 的一个集成安装，anaconda 里面集成了很多关于 Python 科学计算的第三方库，安装它后就默认安装了 Python、IPython、PIP 工具、集成开发环境 Spyder 及众多的包和模块，非常方便。

3. IPython

IPython 是一个 Python 的交互式 Shell，比默认的 Python Shell 好用得多，支持变量自动补全、自动缩进功能，支持 Bash Shell 命令，内置了许多有用的功能和函数。学习 IPython 将会让我们以更高的效率来使用 Python，同时它也是利用 Python 进行科学计算和交互可视化的一个最佳的平台。

4. 2to3 脚本

把 2to3 脚本归为 Python 增强工具其实有点牵强，毕竟它是 Python 自带的，位于 Python 安装目录下的 Tools\Scripts\。2to3 脚本即 2to3.py 文件，它可以自动地将 Python 2 代码翻译成 Python 3 代码。

2to3 脚本的使用方法如下：

首先对 Python 2 源代码进行备份，然后进入 2to3 脚本所在的文件夹，运行此脚本，并以目标文件夹作为参数。例如，假设 Python 2 代码位于 D:\Project1\，那么对应的命令如下：

```
python 2to3.py -w D:\Project1\
```

如果要对单个.py 文件进行翻译，则用这样的命令：

```
python 2to3.py -w D:\Project1\example.py
```

2to3 脚本并不是完全可靠的，主要问题是由于 Python 的动态性，代码有可能翻译之后还需要人工来改。不过更加痛苦的是，由 2to3 翻译的代码只能在 Python 3 运行，造成了这样一个局面：如果用户想要提供一个库，并且 Python 2 和 Python 3 都有，用户就必须保留原始的 Python 2 代码。

1.4　任务 4　获取帮助和查看文档

现在是信息爆炸的时代，一个人不可能牢牢记住所有用到的知识点。即使对单独一门程序设计语言来说，要记住所有的内置对象、函数、模块和类也是不可能的，所以需要掌握获取帮助和查询文档的技巧。

1.4.1　查看特定对象的可用操作

dir()函数是 Python 的一个内建函数（Python 的内建函数可以直接调用，它是 Python 自带的函数，无须通过模块的方式导入，任何时候都可以被使用），函数原型如下：

```
dir([object])
```

其作用是返回一个列表，列举该对象所有的属性和方法。当利用 dir()函数查看一个模块的时候，还能获取该模块中已定义的所有的类、函数和常量。

1.4.2 文档字符串

许多对象都有自己的文档字符串（又称为 DocStrings）。文档字符串用于为模块、类、函数等添加说明性的文字，使程序易读、易懂，更重要的是可以通过 Python 自带的标准方法将这些描述性文字信息输出。文档字符串是对象的属性之一，可以使用 objectName.__doc__来访问它。注意它的名称，前后各有两个下画线。

注意：当不是函数、方法、模块等调用 doc 时，而是具体对象调用 doc 时，会显示此对象从属类型的构造函数的文档字符串。

1.4.3 使用帮助函数

help()函数也是 Python 的一个内置函数。如果不确定一个函数或模块的用途，或者想进一步了解它，就可以用 help()函数来查看帮助文档。操作方法很简单，在 help()括号内填写参数再按 Enter 键即可打开这个模块的帮助文档，例如：

```
help(str)          # 查看关于字符串类型的帮助文档
help(str.join)     # 查看关于字符串对象的 join()方法的帮助文档
```

help()函数用于查看函数或模块用途的详细说明，而 dir()函数用于查看函数或模块内的操作方法，输出的是方法列表。

1.4.4 使用文档

Python 文档可以在很多地方找到。最便捷的方式就是从 Python 官网查看在线文档，也可在 C:\Python2x\Doc\目录下找到一个名为 Python2x.chm 的离线帮助文档。它使用 IE 接口，所以用户实际上是使用网页浏览器来查看文档的。

1.5 小结

本项目介绍了 Python 语言的起源和发展前景、Python 语言的特性和优势，针对不同的系统平台介绍了如何安装 Python 和配置开发环境，还介绍了文本编辑器、IDE 和其他 Python 增强工具，并介绍了 Python 代码的书写规范及查看帮助文档的方法。

- Python 的起源和发展前景。
- Python 的优缺点和擅长的领域。
- Python 的版本差异及 Python 虚拟机的不同实现。
- 下载及安装 Python。
- 配置环境变量。
- 源代码安装 Python。
- 交互式解释器、文本编辑器以及 IDE。
- Python 增强工具。
- 帮助函数。
- 查看文档。

1.6　习题

1．试在自己的计算机上安装 Python（Python 2.7 或 Python 3.X 均可）。

2．为自己安装的 Python 安装 easy_install 和 PIP 工具。

3．为你自己编写身份信息，包括姓名、年龄、身份证号、联系方式等，然后使它们显示在屏幕上。

4．将题 3 的小程序保存为.py 源代码文件，然后执行它。

5．帮助函数 help()能否查询它自身的用法？如果可以，会显示什么信息？