第 1 章　Python 编程环境和核心概念

本章主要介绍 Python 的一些背景信息和 Python 编程的核心概念，了解并掌握这些内容将为后续章节打下坚实的基础。

1.1　Python 简介

通常将 Python 定义为一种面向对象的脚本语言。然而，凭借 Python 对函数、模块、类等程序结构的支持，并能组织和编写大规模的程序，经常作为“脚本”身份出现的 Python 也会转换为“程序”角色。本节将简要介绍 Python 的一些背景信息。

1.1.1　使用 Python 能做什么

Python 是目前发展最快的编程语言之一，这主要归功于其广泛的应用领域，下面列举一些 Python 常见的应用领域。

1. 系统编程

系统编程是使用 Python 编写用于维护操作系统日常管理任务的工具，凭借 Python 自身提供的操作系统服务接口，可以轻松完成此类工作。Python 标准库提供了对操作系统中的环境变量、命令行参数、管道、套接字、进程、多线程、文件名扩展等的编程支持，因此，系统编程是 Python 最擅长的工作之一。

提示：标准库是 Python 提供的一系列编程工具，由大量的模块组成。在 Windows 操作系统中安装 Python 时，会自动安装标准库中的所有模块。

2. 数据库编程

Python 提供了对所有主流关系数据库系统的接口，通过定义用于存取 SQL 数据库系统的可移植 API，为各类底层应用的数据库系统提供了统一的编程方式。不过，有些关系数据库系统需要安装第三方模块之后才能使用 Python 编程控制，例如，需要安装 pymssql 模块才能编程控制微软的 SQL Server 数据库。

3. 网络编程

Python 标准库提供了适合多种网络编程任务的模块，能够完成从服务器端到客户端的不同应用需求，包括打开网页并获取其中的数据、创建和解析 XML 文件、编写和发送电

子邮件、使用套接字进行通信、使用 FTP 传输文件等。此外，很多第三方的模块和工具为使用 Python 进行网络编程和开发网站提供了很多方便。

4. 创建图形用户界面

Python 标准库中的 tkinter 模块专门用于创建图形用户界面，并在不同的计算机平台之间具有良好的可移植性，这些平台包括 Windows、Linux、UNIX 和 macOS 等。还可以使用很多第三方工具在 Python 中开发 GUI（图形用户界面）程序并处理图像，例如，使用 PIL 图像库可以处理几十种图片文件类型。

5. 游戏、数据分析、人工智能

使用第三方工具 Pygame，可以在 Python 中开发游戏。使用 NumPy、Pandas、Matplotlib 等第三方工具，可以在 Python 中对数据执行专业计算和可视化分析。使用第三方工具 PyTouch，可以在 Python 中进行人工智能方面的研究。

1.1.2 Python 的优点

与其他编程语言相比，Python 有很多优点。

1. 易学易用

这可能是想要学习和使用 Python 的用户最关心的问题。与 C、C++ 和 Java 等编程语言相比，Python 比较容易学习和掌握。想要精通任何一门编程语言，都需要长时间的学习和经验积累，不过对于想要快速入门并开始上手编程来说，Python 是适合学习的编程语言。

编写好的 Python 程序无须编译和链接即可直接运行，节省很多烦琐的中间环节。这种交互式的编程体验，为程序开发人员提供了一种可以快速编写、测试和修改代码的便捷环境。

2. 功能丰富

Python 标准库提供了大量的模块和工具，很多第三方工具也为 Python 提供了很多扩展功能，程序开发人员可以使用它们轻松完成各类编程任务。Python 是一种面向对象的编程语言，可以使用对象和类来组织程序中的数据和功能。

3. 快速开发

完成同一个编程任务时，使用 Python 编写的代码量通常只有 C++ 或 Java 代码的三分之一或更少。Python 是动态语言，编写 Python 代码时无须声明变量及其数据类型，变量的类型会根据其所引用的数据的类型自动调整。Python 的这两项优势可以缩短开发周期，节省大量的时间。

4. 软件质量

Python 编程语言在语法方面的特性，使 Python 非常注重代码的可读性、一致性和软件质量，从而使 Python 比其他编程语言具有更好的可重用性和可维护性。

5. 组件集成

Python 可以与其他很多编程语言及其组件集成和协作，这项优势使 Python 成为产品定制和扩展的工具。例如，Python 代码可以调用 C 和 C++ 的库，C 和 C++ 的程序也可以调用 Python 代码，Python 代码可以与 Java 组件集成，还可以与 COM 和 .NET 等框架进行通信。

6. 可移植性

使用 Python 编写的大多数程序无须修改，即可在不同的计算机平台上运行。例如，如需在 Windows 和 Linux 之间移植 Python 程序，只需直接复制 Python 程序文件即可。

1.1.3　Python 代码在计算机内部的运行方式

在 Python 中将编写好的代码存储在扩展名为 .py 的文件中，这是 Python 的标准文件。运行 Python 程序时，在 Python 内部将 .py 文件中的代码编译成 Python 特有的“字节码”，字节码可以加快程序的运行速度且与平台无关。Python 会将转换后的字节码保存在扩展名为 .pyc 的文件中，该文件位于一个新建的文件夹中，该文件夹与正在运行 Python 代码所在的 .py 文件位于同一个文件夹。

如果 Python 在当前文件夹中没有写入权限，则无法创建包含字节码的 .pyc 文件，此时 Python 会在内存中临时创建字节码，以确保 Python 程序能够正常运行，程序运行结束后会从内存中删除字节码。

创建字节码后，Python 会将其发送到 Python 虚拟机（Python Virtual Machine）中并运行字节码指令。Python 虚拟机是在安装 Python 时内置在其中的功能。无论是转换为字节码，还是在 Python 虚拟机中运行字节码，都由 Python 自动完成，无须人工干预。

1.2　编写和运行 Python 代码

Python 是一种解释型语言，在 Python 中编写的代码运行在被称为“解释器”的软件中，解释器是一种可以运行其他程序的程序。使用从 Python 官方网站中下载的 Python 安装程序并在计算机中安装的就是 Python 解释器，以及支持的库和相关组件。本节将介绍用户如何在计算机中编写和运行 Python 代码，但是不会涉及 Python 代码的语法细节，这些内容将在本书后续章节详细介绍。

1.2.1　在计算机中安装 Python

在浏览器中访问 Python 官方网站（https://www.python.org），然后在 Download 类别

中下载 Python 安装程序。根据操作系统的类型（Windows、Linux/UNIX 或 macOS）和架构（32 位和 64 位）及 Python 版本，选择相应的 Python 安装程序进行下载。下载完成后，双击 Python 安装程序，在当前操作系统中安装 Python。

安装 Python 和安装其他软件类似，有默认安装或用户自定义安装两种方式，下面介绍它们的操作方法。

1. 以默认方式安装 Python

双击下载好的 Python 安装程序，打开图 1-1 所示的对话框。选择 Install Now 选项，将以默认方式安装 Python，在该选项的下方显示安装 Python 的完整路径和默认安装的组件，包括 IDLE、pip、Python 文档、快捷方式和文件关联等。

图 1-1　选择安装方式

提示：如果选中 Add python.exe to PATH 复选框，则以后每次在系统命令行窗口中使用 Python 时，可以直接输入“python”而无须输入完整路径。如果未选中该复选框，则以后可以在操作系统中通过手动设置环境变量实现该选项的功能，具体方法请参考 1.2.4 小节。

2. 自定义安装 Python

如果想要自己指定 Python 的安装路径和安装的组件，则需要在图 1-1 中选择 Customize installation 选项，将进入图 1-2 所示的界面，在此处选择想要安装的组件。

- Documentation：安装 Python 官方文档和日志文件。
- pip：安装 pip，以后可以使用该工具自动下载并安装第三方工具，从而扩展 Python 的功能。
- td/tk and IDLE：安装 tkinter 模块和 IDLE。使用 tkinter 模块可以在 Python 中创建图形用户界面。IDLE 是 Python 自带的集成开发环境，使用它可以编写、运行和调试 Python 代码。
- Python test suite：安装 Python 标准库中的测试套件，使用它可以测试 Python 代码。
- py launcher：安装 Python 启动器，以后输入 py 即可启动 Python。在计算机中安装

多个 Python 版本时，使用 Python 启动器可以自动运行 Python 的最新版本。如需运行特定的版本，可以输入 py 和版本号，如“py -3.12”。

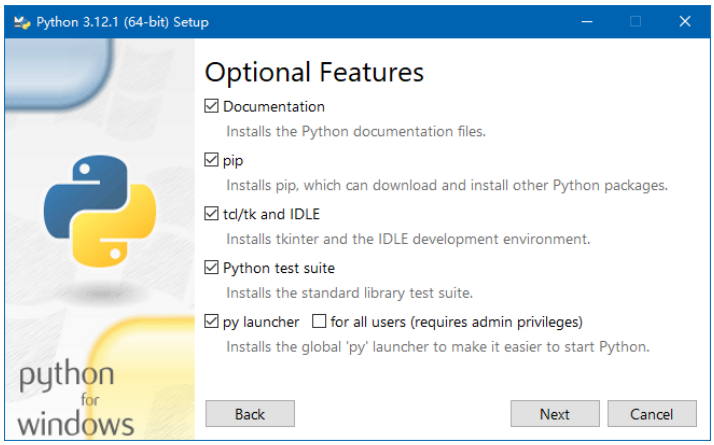

图 1-2　选择安装哪些组件

选择好要安装的组件后，单击 Next 按钮，进入图 1-3 所示的界面，在此处选择安装 Python 时的附加选项，这些选项不会影响 Python 的安装，但是会影响使用 Python 时的体验。图 1-3 中选中的 3 个选项的含义如下：

- 关联 Python 文件，只有安装 Python 启动器才能使用该选项。
- 创建 Python 快捷方式。
- 将 Python 添加到环境变量中。

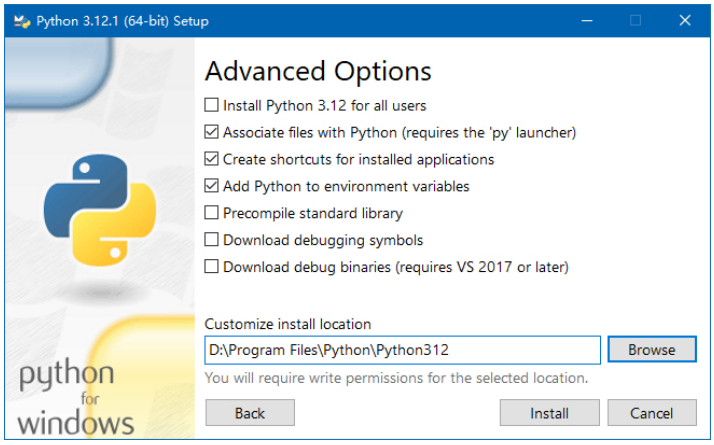

图 1-3　选择安装哪些附加选项

如需更改 Python 的安装位置，可以单击 Browse 按钮，然后选择所需的位置。完成所有设置后，单击 Install 按钮，开始安装 Python。稍后将显示图 1-4 所示的界面，说明已经成功安装 Python，单击 Close 按钮，关闭该对话框。

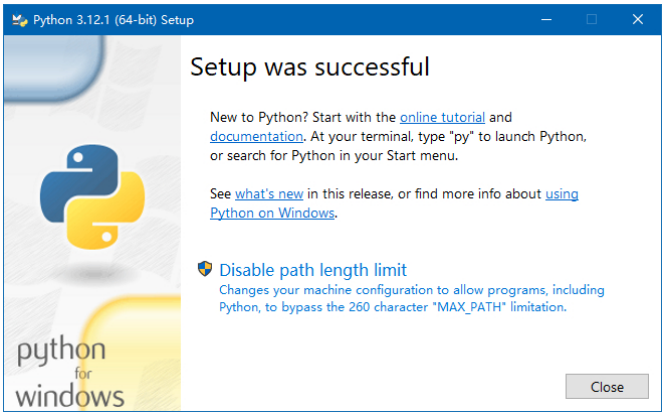

图 1-4　成功安装 Python

如果在安装 Python 后发现某些功能无法使用，说明没有安装相应的功能，此时无须卸载并重新安装 Python，可以直接在现有的安装中添加新的安装选项。此处以 Windows 10 操作系统为例，操作步骤如下：

（1）右击任务栏中的“开始”按钮，在弹出的快捷菜单中选择“应用和功能”命令。

（2）在打开的窗口中选择已安装的 Python，然后单击“修改”按钮，如图 1-5 所示。

图 1-5　选择已安装的 Python 程序并单击“修改”按钮

（3）打开图 1-6 所示的对话框，选择 Modify 选项。

（4）接下来进入的两个界面与图 1-2 和图 1-3 类似，在这两个界面中可以选择要添加的组件。在这两个界面中完成所需的设置后，单击 Install 按钮，将完成组件的添加或删除操作。

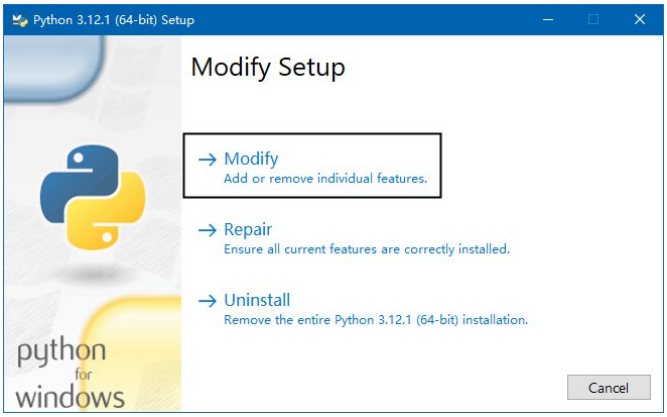

图 1-6　选择 Modify 选项

1.2.2　交互模式和脚本模式

IDLE 是使用 Python 创建的集成开发环境（Integrated Development Environment，IDE）。IDLE 提供了两种窗口类型——命令行窗口和编辑器窗口。在 IDLE 的命令行窗口中输入的代码是在交互模式中工作的。在交互模式中，每次输入一行代码并按 Enter 键，Python 解释器会立即运行该代码并显示结果。如果输入的是多行语句（在 Python 中称为复合语句），则需要在一次性输完这些语句后再执行它们。

输入 Python 代码的另一种方式是使用 IDLE 的编辑器窗口。用户可以创建一个或多个编辑器窗口并在其中编写代码，然后将每个编辑器窗口中的代码保存为相互独立的 .py 文件，便于以后继续编写或运行文件中的代码。使用编辑器窗口输入的代码是在脚本模式中工作的，在该模式中可以输入一行或多行代码，输入的每行代码不会立即运行。运行文件中的代码时，会自动将文件中的所有代码作为一个整体一次性运行。

无论使用交互模式还是脚本模式，都需要在“开始”菜单中展开 Python 文件夹，然后选择以 IDLE 开头的命令，如图 1-7 所示。

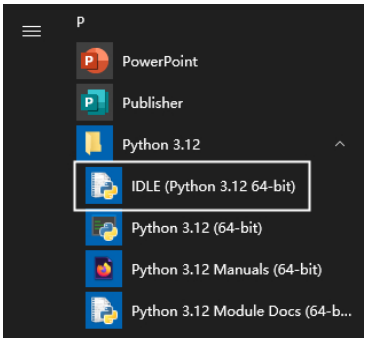

图 1-7　启动 IDLE

启动 IDLE 后，Python 解释器将运行在交互模式的命令行窗口中。在窗口的顶部显示 Python 的版本和版权信息，这些信息的下方显示 3 个大于号（>>>）和一条闪烁的竖线，等待用户输入 Python 代码，如图 1-8 所示。

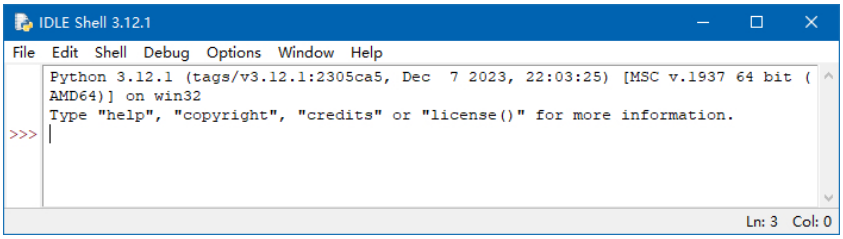

图 1-8　IDLE 的命令行窗口

3 个大于号是 IDLE 命令行窗口中的主提示符，在主提示符的右侧输入一条语句并按 Enter 键，将立即执行该语句并显示结果。图 1-9 所示的“6”是输入“1+5”后的计算结果。

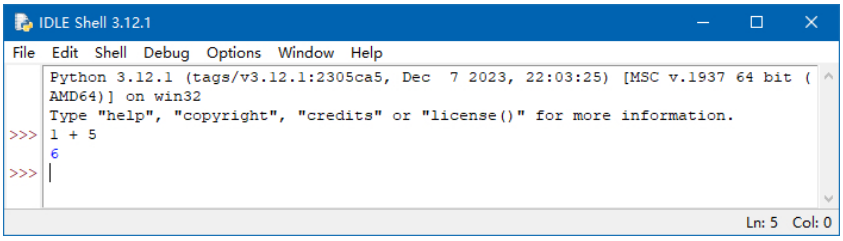

图 1-9　输入代码后立即显示运行结果

输入多行语句时，首行语句的开头显示主提示符，其他行语句的开头显示次要提示符，次要提示符以 3 个圆点（…）表示，如图 1-10 所示。

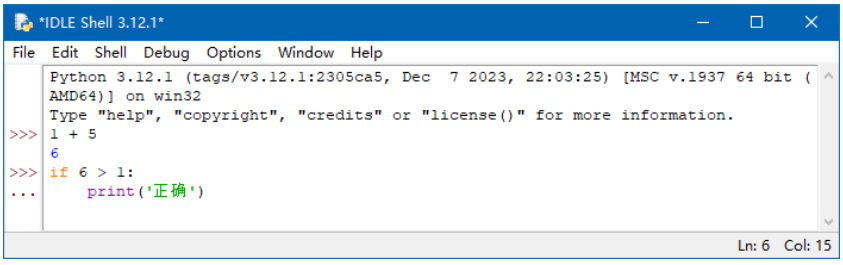

图 1-10　输入复合语句时显示主要提示符和次要提示符

如需在脚本模式中编写 Python 代码，可以在 IDLE 的命令行窗口顶部的菜单栏中选择 File → New File 命令，如图 1-11 所示。将打开一个空白窗口，其外观与 Windows 记事本程序类似，可以在该窗口中输入一行或多行代码，并将其保存为以 .py 为扩展名的文件，以后可以反复查看、编辑或运行该文件中的代码。

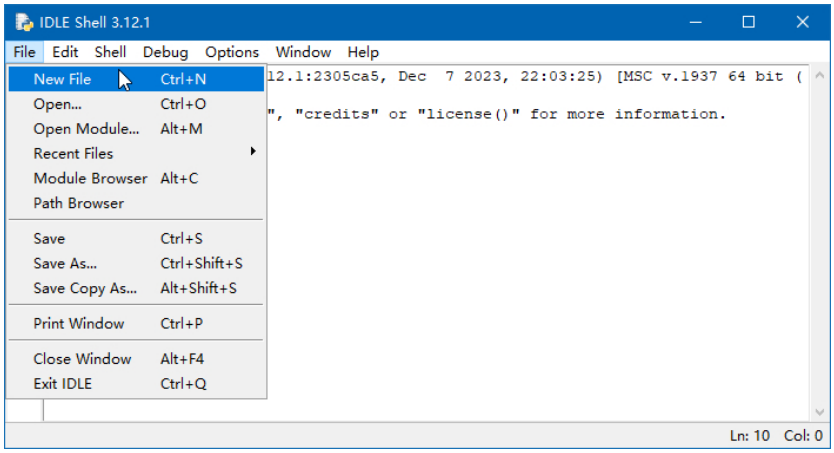

图 1-11　选择 New File 命令

1.2.3　在 IDLE 中编写和运行 Python 代码

可以在 IDLE 的命令行窗口或编辑器窗口中编写和运行 Python 代码。下面编写一个显示简单信息的 Python 程序，运行该程序时，用户需要输入一个名字，然后会显示包含该名字的欢迎信息。

1. 在 IDLE 命令行窗口中编写代码

启动 IDLE，在第一个主提示符的右侧输入以下代码：

```
>>> name = input('请输入你的姓名：')
```

input 是一个 Python 内置函数，该函数接收用户输入的数据，并将数据存储到名为 name 的变量中。

提示：现在不需要为不能理解代码的含义而担心，因为此处只是为了说明如何在 IDLE 中输入代码，代码的语法细节将在后续章节详细介绍。

按 Enter 键，将显示图 1-12 所示的提示信息，它是上一行代码中位于圆括号中的文字。

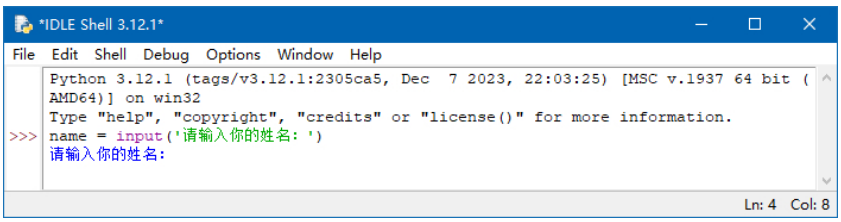

图 1-12　等待用户输入数据

输入一个名字，然后按 Enter 键，不会显示任何结果，只是在下一行新增一个主提示符，如图 1-13 所示。

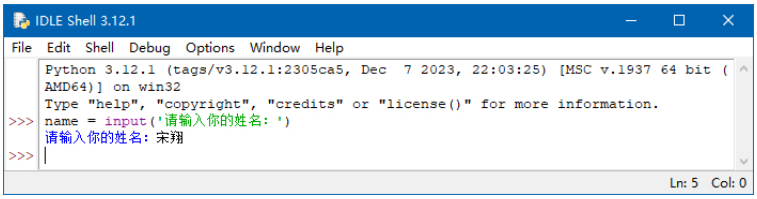

图 1-13　输入数据后自动新增一个主提示符

在第二个主提示符的右侧输入以下代码并按 Enter 键：

```
>>> print('欢迎你，' + name)
```

将显示包含刚才输入的名字的欢迎信息，如图 1-14 所示。

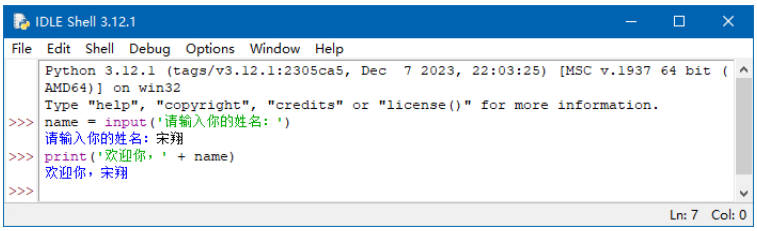

图 1-14　显示带有指定名字的欢迎信息

提示：与 input 类似，print 也是一个 Python 内置函数，用于在命令行窗口中输出指定的信息。由于 IDLE 的交互模式会自动显示代码的运行结果，所以，在该模式中输入代码时，即使不显式使用 print 函数，也会在窗口中输出信息。本例直接输入“'欢迎你，'+ name”也会显示欢迎信息，如图 1-15 所示。细心的读者可能已经注意到，不使用 print 函数时，在代码的运行结果中包含一对单引号，第 3 章会说明导致这种情况的原因。

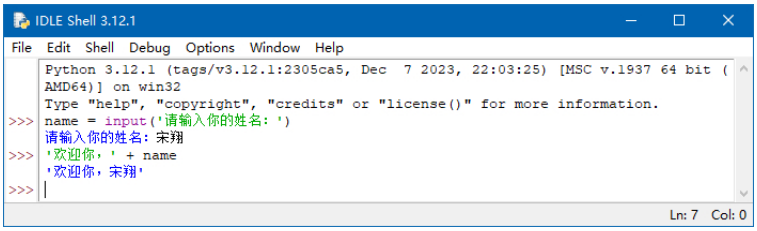

图 1-15　不使用 print 时的运行结果

前面说明了在 IDLE 的交互模式中输入 Python 代码的基本方法。正如前面介绍的，每次输入一行代码并按 Enter 键，就会立即运行该行代码并显示结果。

2. 在 IDLE 编辑器窗口中编写代码

在 IDLE 的编辑器窗口中可以不受影响地输入多行代码，输入好所有代码后再一次性运行这些代码。下面在编辑器窗口中输入前面示例中的代码。首先新建一个编辑器窗口，然后在该窗口中输入前面示例中的两行代码，如下所示：

```
name = input('请输入你的姓名：')
print('欢迎你，' + name)
```

在编辑器窗口的菜单栏中选择 File → Save 命令，在弹出的对话框中设置文件的名称和保存位置，单击“保存”按钮，将编辑器窗口中的代码以文件的形式保存到计算机中。在编辑器窗口顶部的标题栏中将显示该文件的名称和路径，如图 1-16 所示。

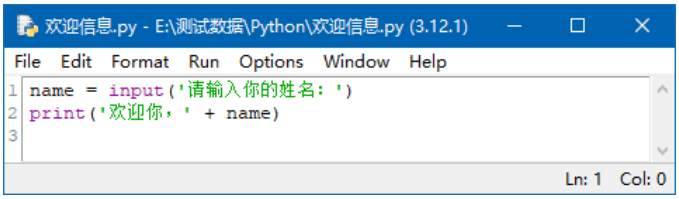

图 1-16　在编辑器窗口中输入代码

保存代码后，只需在编辑器窗口中选择 Run → Run Module 命令，或者按 F5 键，即可运行该窗口中的所有代码，并在 IDLE 的命令行窗口中显示运行结果，如图 1-17 所示。

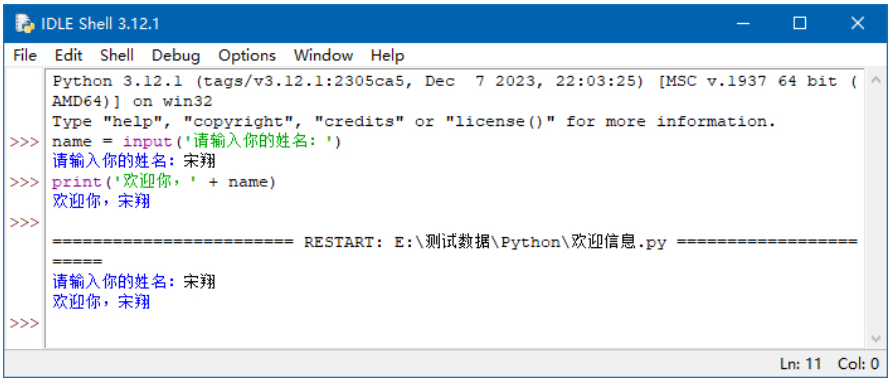

图 1-17　运行编辑器窗口中的代码并在命令行窗口中显示结果

提示：如需修改 .py 文件中的 Python 代码，可以在 IDLE 的命令行窗口或编辑器窗口顶部的菜单栏中选择 File → Open 命令，然后双击包含代码的 .py 文件。

1.2.4　在系统命令行窗口中运行 Python 代码

前面介绍的在 IDLE 的交互模式中输入 Python 代码的方法，也同样适用于系统命令行窗口。系统命令行窗口在 Windows 操作系统中被称为“命令提示符”，在“开始”菜单中可以找到该命令。

在系统命令行窗口中输入 Python 代码也是交互式的，每输入一行代码并按 Enter 键，将立即运行该代码并显示结果。只不过在系统命令行窗口中输入代码不如 IDLE 方便，因为无法享受 IDLE 提供的很多特性，如语法高亮、智能缩进、自动补全等。

如果已将编写好的 Python 代码保存在一个文件中，则可以在系统命令行窗口中使用以下格式运行该文件中的 Python 代码，格式中的两个部分之间需要保留一个空格。

```
Python 解释器的完整路径 Python 文件的完整路径
```

假设将 Python 安装在以下路径，在该路径中有一个名为 python.exe 的文件，该文件是启动 Python 解释器的可执行文件。

```
D:\Program Files\Python312
```

假设要运行的 Python 代码包含在名为 test.py 的文件中，该文件位于以下路径，其中的代码用于显示“Hello World”。

```
E:\ 测试数据 \Python
```

在 Windows 操作系统的命令提示符中输入以下语句并按 Enter 键，将运行 test.py 文件中的 Python 代码，如图 1-18 所示。

```
"D:\Program Files\Python312\python" E:\ 测试数据 \Python\test.py
```

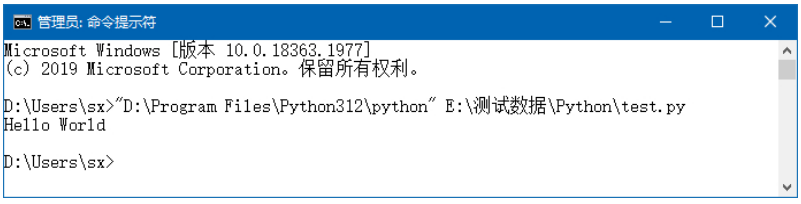

图 1-18　Python 解释器和 Python 文件都需要输入完整路径

注意：由于在第一个路径中包含空格，所以，需要将该路径放到一对英文双引号中，否则将导致程序出错。路径中没有空格时可以省略双引号。

如果在安装 Python 时选中 Add python.exe to PATH 复选框，则会将 Python 的完整路径添加到操作系统的环境变量中，每次在系统命令行窗口中运行 Python 文件时可以输入以下语句，即省略 Python 解释器的路径部分，如图 1-19 所示。

```
python E:\ 测试数据 \Python\test.py
```

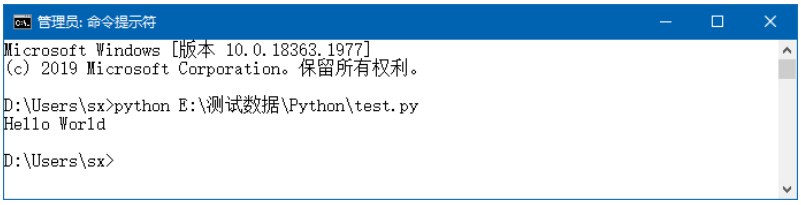

图 1-19　省略 Python 解释器的路径部分

提示：在系统命令行窗口中输入的英文字母的大小写不重要，只要拼写正确即可。

用户可以手动将 Python 解释器的完整路径添加到环境变量中，此处以 Windows 10 为例，操作步骤如下：

（1）右击“开始”按钮，在弹出的快捷菜单中选择“系统”命令。

（2）打开图 1-20 所示的窗口，选择“系统信息”选项。

图 1-20　选择“系统信息”选项

（3）打开“系统”窗口，选择“高级系统设置”选项，如图 1-21 所示。

图 1-21　选择“高级系统设置”选项

（4）打开“系统属性”对话框，单击“环境变量”按钮，如图 1-22 所示。

（5）打开“环境变量”对话框，在上方的列表框中双击名为 PATH 的环境变量，如图 1-23 所示。

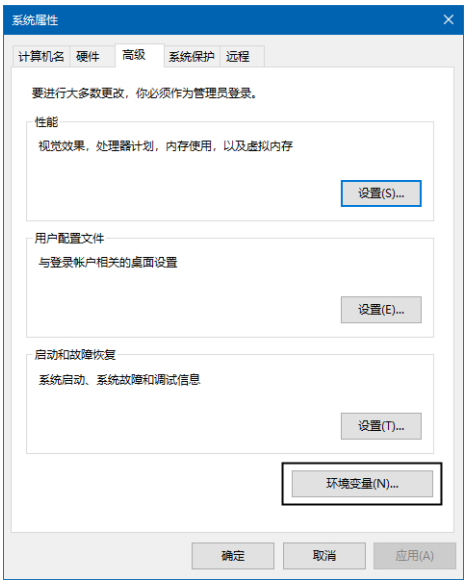

图 1-22　单击“环境变量”按钮

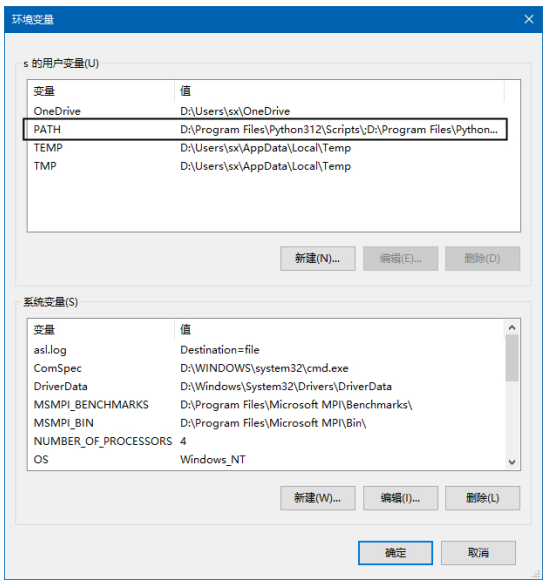

图 1-23　双击名为 PATH 的环境变量

（6）打开“编辑环境变量”对话框，单击“新建”按钮，然后输入 Python 解释器的完整路径并按 Enter 键，将其添加到 PATH 环境变量中，最后单击“确定”按钮，如图 1-24 所示。

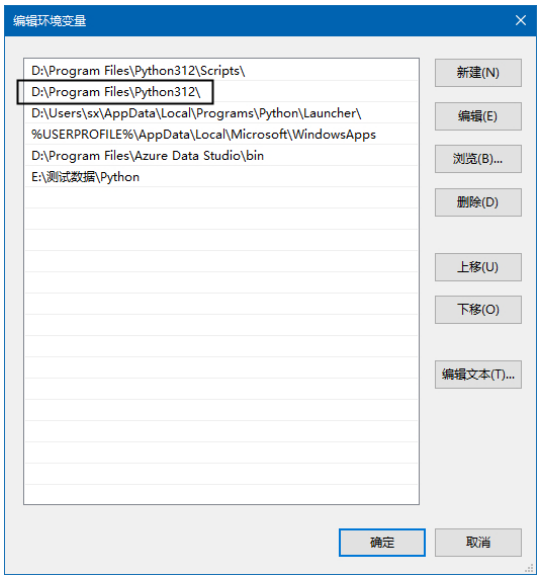

图 1-24　将 Python 解释器的完整路径添加到 PATH 环境变量中

如果在安装 Python 时选中 py launcher 和 Associate files with Python 两个复选框，则会安装 Python 启动器，并将所有扩展名为 .py 的文件关联到 Python 解释器。以后在系统命令行窗口中只需输入 py 即可启动 Python 解释器，还可以在文件夹中直接双击 .py 文件来运行其中的代码，并在系统命令行窗口中显示运行结果。

双击前面示例中的 test.py 文件，可能会发现系统命令行窗口一闪而过，无法看清楚运行结果。解决该问题的方法是在文件底部的空行中添加以下代码，如图 1-25 所示。

```
input()
```

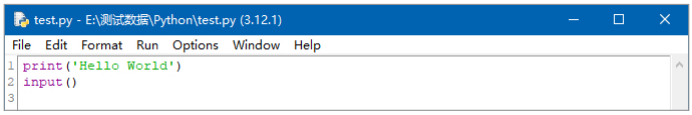

图 1-25　添加一行代码

再次双击 test.py 文件，将在系统命令行窗口中显示该文件的运行结果，只有按 Enter 键，才会关闭该窗口，如图 1-26 所示。

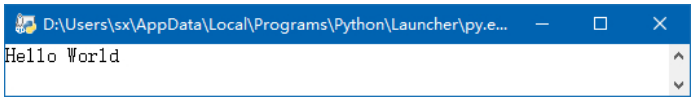

图 1-26　显示运行结果且不自动关闭窗口

技巧：如果希望只输入 Python 文件名，而省略 Python 解释器和文件的路径，就可以在系统命令行窗口中运行文件中的代码，则可以使用类似于本节前面介绍的方法，将 Python 文件的路径添加到 PATH 环境变量中。例如，如果将 test.py 文件所在的路径添加到 PATH 环境变量中，则可以在系统命令行窗口中直接输入 test.py，即可运行该文件中的代码。

1.2.5　使用独立可执行文件运行 Python 代码

存储 Python 代码的文件的扩展名是 .py，运行该类型的文件就会自动执行其中包含的 Python 代码。使用一些第三方工具（如 Windows 中的 py2exe 工具），可以将 .py 文件转换为可独立执行的二进制文件（如 Windows 中的 .exe 文件），在 Python 中将这种文件称为冻结二进制文件。由于在冻结二进制文件中嵌入了 Python，所以，用户无须安装 Python 即可运行冻结二进制文件中的 Python 代码。

1.2.6　配置 IDLE

对于 Python 初学者来说，Python 内置的 IDLE 是学习 Python 编程的理想工具，不仅因为其界面简洁、简单易用，更重要的是可以在 IDLE 交互模式中通过输入各种代码来测

试它们的功能，由此可以检验自己对 Python 编程的理解和掌握程度。与其他第三方 IDE 相比，IDLE 省去了大量烦琐的底层细节和选项配置，减少很多麻烦，从而可以将精力放在 Python 语言本身上。简单来说，使用 IDLE 可以马上开始学习 Python 编程，而不是将时间浪费在熟悉 IDE 界面的配置和使用方法上。

IDLE 具有以下特性：

- 在 Windows、UNIX、Linux 和 macOS 等不同的平台上具有相同的工作方式。
- 可以使用 IDLE 窗口顶部的菜单栏中的命令执行各种操作，也可以使用显示在命令右侧的快捷键提高操作速度。
- 提供语法高亮，自动为代码中的语言元素设置不同的字体颜色。
- 智能缩进代码，Python 使用缩进区分不同的代码块，在 Python 中“缩进”是一种语法规则。
- 调用函数时，自动显示函数的参数和简要说明。
- 输入对象和点分隔符后，自动显示对象可用的属性和方法列表，可以从中选择要输入的项目。
- 可以对代码执行复制、粘贴、查找和替换等操作。
- 自动检测匹配的括号，输入匹配的括号时，自动高亮显示相匹配的括号。
- 在 IDLE 的命令行窗口中显示代码的运行结果和错误信息。
- 在 IDLE 的编辑器窗口中可以多次撤销操作。在 IDLE 的命令行窗口中，只要还没有按 Enter 键，就可以对当前正在输入的代码执行多次撤销操作。
- 提供一个支持单步调试、断点调试等功能的调试器。
- 可以自定义 IDLE 的默认选项，使其界面显示和行为方式更符合个人操作习惯。

如需更改 IDLE 的默认选项，可以在 IDLE 命令行窗口顶部的菜单栏中选择 Options → Configure IDLE 命令，打开相应的对话框，在其中的各个选项卡中设置 IDLE 的外观和行为，下面介绍一些常用的选项。

1. 设置代码的字体格式

在 Fonts 选项卡中可以设置代码的字体格式，如图 1-27 所示。在列表框中选择一种字体，然后单击下方的按钮，从打开的列表中选择字体大小。选中 Bold 复选框，可将字体加粗显示。

2. 设置语法高亮

在 Highlights 选项卡中可以设置语法高亮的配色方案，以及创建并选择要使用的颜色主题，如图 1-28 所示。如需更改一种语言元素的颜色，可以单击 Normal Code or Text 按钮，在打开的列表中选择一种元素，或者直接在缩略图中单击元素示例，然后单击 Choose Color for：按钮并选择一种颜色。

单击 Save as New Custom Theme 按钮，可以将当前的配色方案保存为一个新的主题，然后可以选中 a Custom Theme 单选按钮，再单击下方的按钮并选择自定义的主题，如图 1-29 所示。

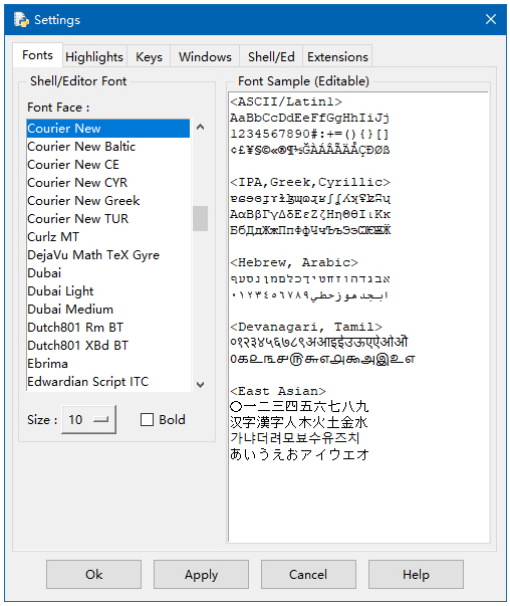

图 1-27　设置代码的字体格式

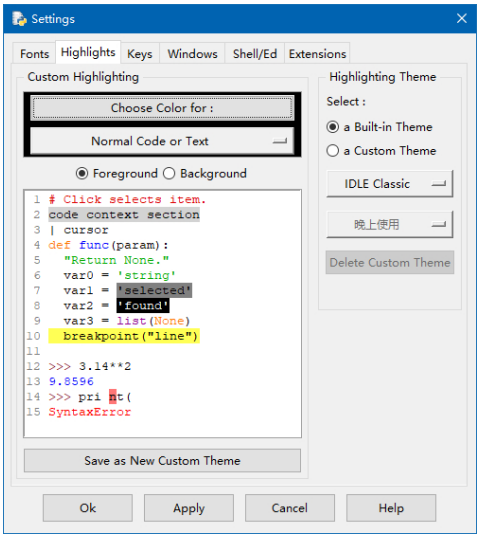

图 1-28　设置语法高亮

图 1-29　选择自定义主题

3. 设置快捷键

在 Keys 选项卡中可以设置很多菜单命令的快捷键，如图 1-30 所示。Python 为不同平台提供了预置的快捷键方案，选中 Use a Built-in Key Set 单选按钮，然后单击右侧的按钮，在打开的列表中可选择带有相应平台的选项。

如需更改 Python 预置的快捷键，可以在列表框中选择一项，然后单击下方的 Get New Keys for Selection 按钮，在弹出的对话框中指定所需的快捷键，如图 1-31 所示。单击

图 1-30 中的 Save a New Custom Key Set 按钮，可以将当前的快捷键组合保存为新的快捷键方案，以便于以后可以在不同的快捷键方案之间快速切换。

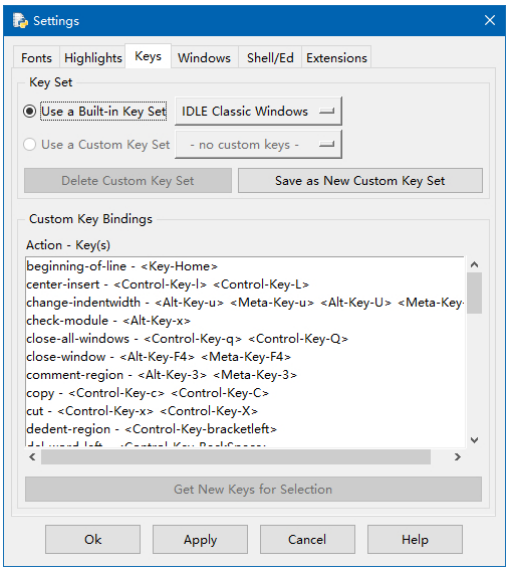

图 1-30　设置快捷键

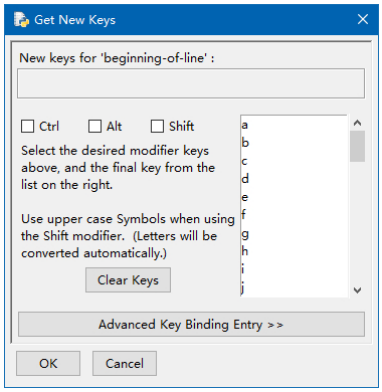

图 1-31　自定义快捷键

4. 设置启动 IDLE 时默认显示的窗口类型

启动 IDLE 时默认显示命令行窗口，如需在启动 IDLE 时默认显示编辑器窗口，可以在 Windows 选项卡中选中 Open Edit Window 单选按钮，如图 1-32 所示。

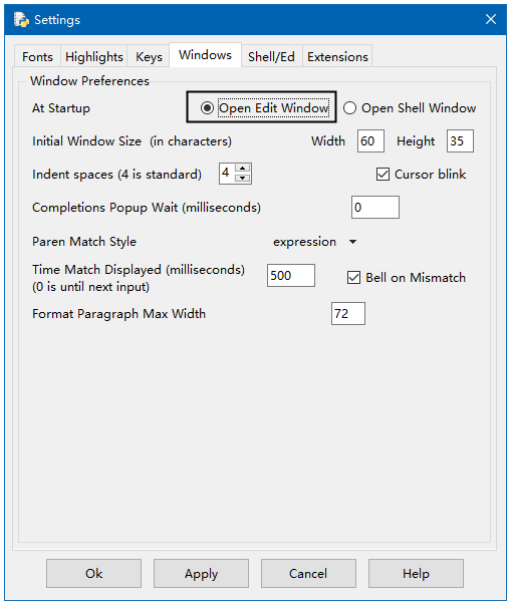

图 1-32　设置窗口显示方式

5. 设置窗口大小

Windows 选项卡中的 Width 和 Height 两个选项用于设置命令行窗口和编辑器窗口的大小，两个选项的值以字符为单位，如图 1-33 所示。

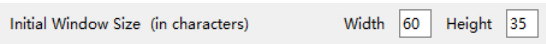

图 1-33　设置窗口大小

6. 设置代码的缩进距离

前面曾经提到过，“缩进”是在 Python 中编写代码时的一种格式规则，当输入复合语句时，必须为不同级别的代码添加缩进，否则，将导致程序出错。Windows 选项卡中的 Indent spaces（4 is standard）选项用于设置缩进距离，如图 1-34 所示。

图 1-34　设置代码缩进距离

7. 设置自动完成列表的弹出速度

编写 Python 代码时，在输入对象的名称和一个英文句点之后，将自动弹出一个列表，其中包含该对象的方法，使用方向键选择一个选项，即可将其添加到当前代码中。如需在输入英文句点后立刻显示该列表，可以在 Windows 选项卡中将 Completions Popup Wait（milliseconds）选项设置为“0”，该选项的单位是毫秒，如图 1-35 所示。

图 1-35　设置自动完成列表的弹出速度

1.3　Python 代码的组成结构

虽然现在还没开始编写真正有用的 Python 程序，但是从整体上了解 Python 程序和代码的组成结构，会为以后的学习打下一个好的基础。一个 Python 程序由一个或多个 .py 文件组成，将这些文件称为模块。在每个模块中包含一行或多行代码，每行代码都是一条可执行的语句，每条语句由 Python 关键字和表达式组成，每个表达式由字面值、常量、变量、运算符和函数调用组成。本节将简要介绍组成 Python 代码的各种语言元素，在本书后续章节中介绍 Python 内部提供的各类对象时，会介绍这些语言元素的更多内容。

1.3.1　字面值

字面值、常量和变量是组成 Python 代码的最基本单位，它们也是程序要处理的数

据。Python 中的每类数据都对应于一种特定的数据类型，每种数据类型都有仅适合该类型的特定操作，也有一些操作适用于所有数据类型。

字面值就是用户输入的数据。16、1.6、" 你好 " 等都是字面值。在交互模式中输入一个字面值，然后按 Enter 键，将在下一行显示该字面值。

```
>>> 16
16
```

如果输入的是文本而非数字，则必须将文本输入到一对英文单引号或双引号中，否则将导致程序出错。在 Python 中将文本称为字符串。

```
>>> ' 你好 '
' 你好 '
```

即使输入字符串时使用双引号，按 Enter 键后也会显示为单引号。

```
>>> " 你好 "
' 你好 '
```

1.3.2 常量

Python 内部提供了一些常量，它们是一些具有特定名称的固定不变的值，如 True、False、None 等，可以在 Python 代码中使用这些常量。输入 Python 内置常量时，必须严格遵守其字母大小格式，否则将导致程序出错。

也可以将字面值称为常量，因为输入一个字面值后，其值也是固定不变的。

1.3.3 变量

无论编写实现何种功能的 Python 程序，即使只有一两行 Python 代码，也通常会使用变量。变量是一个由用户指定的名称，使用这个名称可以存储任何数据，可能是输入的一个字面值，也可能是一个简单或复杂的表达式。表达式的概念将在 1.3.6 小节介绍。

在交互模式中输入下面的代码，表示将数字 16 存储到名为 n 的变量中。

```
>>> n = 16
```

接着在交互模式中输入下面的代码，将在下一行显示存储在变量 n 中的值。

```
>>> n
16
```

与其他编程语言不同，在 Python 中使用变量之前，不需要预先声明，实际上，Python 根本没提供用于声明变量的语句。在 Python 中使用一个变量是非常简单的，只需将一个值存储到变量中，接下来就可以在代码中使用该变量代替这个值。

将一个值存储到变量的操作称为“为变量赋值”，使用等号连接变量和值，变量在等号的左侧，值在等号的右侧，正如 1.3.3 小节开头的示例所示。在代码中使用一个变量之前，必须先为该变量赋值，否则将导致程序出错。下面的代码中的最后一行指明错误类型和原因：NameError 表示名称错误，错误的原因是在使用 n 之前没有定义它。所谓“没定义”是指没有预先为 n 赋值，这样会导致 Python 不知道该如何处理 n。

```
>>> n
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    n
NameError: name 'n' is not defined
```

一个好的变量名应该能够通过其名称获悉变量的含义，为了提高代码的可读性，最好为变量设置一个含义清晰的名称。此外，Python 对可在变量名中使用的字符有一些限制：变量名不能以数字开头，只能使用大写或小写的英文字母、数字和下画线，英文字母区分大小写。如果在一个变量名中包含多个英文单词，可以使用下画线分隔各个单词，或者将每个单词的首字母大写。不过，Python 有一个不做强制要求但是约定俗成的规定：只有类名称的首字母才使用大写形式，而变量的首字母使用小写形式。

编写 Python 代码时可能会遇到开头和结尾都带有下画线的变量，它们是 Python 内部定义的变量。使用这种特殊格式是为了避免与用户创建的变量重名。

1.3.4　数据类型

在 Python 中使用的数据都有特定的类型。例如，在 1.3.1 小节中输入的 16 是一个整数，该数字在 Python 中属于整数数据类型。在 1.3.1 小节中输入的“你好”是一个字符串，它在 Python 中属于字符串数据类型。

“数据类型”这一术语在其他编程语言中极其常见。由于 Python 将任何东西都看作对象，所以，在 Python 中将数据类型称为对象类型可能更加贴切，这两种术语在本书中可互换使用。Python 内置的常用对象类型如表 1-1 所示。

表 1-1　Python 内置的常用对象类型

对象类型	在 Python 中的名称
整数	int
浮点数	float
复数	complex
字符串	str
列表	list

续表

对象类型	在 Python 中的名称
元组	tuple
字典	dict
集合	set

除了表 1-1 中列出的对象类型，函数、类、模块等也都是 Python 中的对象类型，代码本身也是一种对象。实际上，Python 中的所有对象类型本身也属于一个共同的对象类型——type。

1.3.5 运算符

字面值、常量和变量都是独立的个体，只有将它们关联在一起才能得到各种不同的结果——这就是运算符所发挥的作用。使用运算符可以将字面值、常量和变量连接在一起，使它们之间执行运算并得出结果。

Python 中的运算符如表 1-2 所示，可以将这些运算符分为算术运算符、比较运算符、布尔运算符、移位运算符等多种类别，本书后续章节将介绍常用运算符的用法。

表 1-2 Python 中的运算符

+	-	*	**	/	//	%
<	>	<=	>=	==	!=	:=
<<	>>	&	|	^	~	@

1.3.6 表达式

表达式由字面值、常量、变量和运算符组成，单个的字面值、常量或变量也是一个表达式。在交互模式中输入一个表达式，会显示表达式的计算结果。下面的代码是一个包含字面值、变量和 + 运算符的表达式，假设已将数字 16 赋值给该变量，则输入该表达式后显示的计算结果是 18。

```
>>> 2 + n
18
```

1.3.7 语句

语句通常由 Python 关键字和表达式组成，不过赋值语句是个例外，因为它不使用关

键字，只需使用一个等号，将右侧的值赋值给左侧的变量。下面的代码将数字 16 赋值给名为 n 的变量。输入这条语句后按 Enter 键，不会显示任何结果。

```
>>> n = 16
```

与表达式只能返回一个计算结果不同，使用语句可以执行特定的操作，并且通常不显示结果，正如上面的示例所示。下面的代码是语句的另一个示例，它使用 Python 关键字 del 删除一个变量，从而手动释放其占用的内存空间。

```
>>> del n
```

上面给出的两个示例都只有一行代码。在 Python 中还有一类语句包含多行代码，将这类语句称为“复合语句”。复合语句的格式和输入方法比较特殊，具体如下：

（1）复合语句的第一行以英文冒号结尾。

```
>>> if 1 > 6:
```

（2）在复合语句的第一行结尾按 Enter 键后，第二行开头的提示符变成以 3 个圆点（…）显示的次要提示符，并自动向右缩进指定的距离。具有相同缩进量的连续多行代码属于同一个单元，具有相同的层次级别。

```
>>> if 3 < 6:
...     print('确实如此')
```

（3）输入好所有复合语句后，按两次 Enter 键，将在两个以次要提示符开头的空行的下一行显示结果。

```
>>> if 3 < 6:
...     print('确实如此')
...
...
确实如此
```

提示：在后续章节中遇到包含复合语句的代码时，不会在书中印出额外的两个空行，以免浪费篇幅。

1.3.8　Python 关键字

本章前面介绍的 Python 内置常量、语句中使用的特定名称，都是 Python 中的关键字。关键字也称为保留字，是 Python 内部使用的名称，用户不能将它们用作变量的名称。

Python 中的关键字如表 1-3 所示，它们在 Python 的不同版本中可能略有差异。输入 Python 关键字时，必须与表 1-3 中英文字母的大小写保持一致。

表 1-3　Python 中的关键字

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

1.3.9　注释

注释是给代码添加的说明性信息，运行代码时会自动忽略注释。在交互模式中输入代码时通常不会添加注释，在脚本模式中编写较长的代码时才会添加注释。

Python 中的注释以 # 符号开头，位于该符号右侧直到同行结尾之间的内容都被 Python 当作注释。下面是为代码添加注释的两种形式，第一种注释单独占据一行，第二种注释位于一行代码的右侧。

```
# 这是一个包含多行代码的程序
n = 16
if n > 100:    # 判断 n 的值是否大于 100
    print('n 的值很大 ')
else:
    print('n 的值很小 ')
```

1.3.10　Python 代码编写规范

Python 非常注重代码的简洁性和一致性，因此，在编写 Python 代码时有一些格式规范，也可将其看作一种惯例。

- 一行代码最多不超过 79 个字符。
- 为了避免代码中的字符过于密集而影响可读性，可以在运算符前后、逗号后添加空格。
- 当一行代码较长时，可以使用反斜线将一行代码拆分为多行，拆分后的多行在逻辑上仍然属于同一行。
- 使用空格或制表符作为代码的缩进，为了避免混乱，应该始终只使用其中的一种，最好使用空格。
- 使用空行分隔函数和类，以及函数内较长的代码块。

- 将注释放在单独的一行，不要放在代码行的右侧。
- 使用文档字符串。
- 用户创建的函数的首字母使用小写。如果函数名由多个英文单词组成，则使用下画线分隔它们。
- 用户创建的类的首字母使用大写。如果类名由多个英文单词组成，则每个单词的首字母都应该大写。

1.4　Python 编程的核心概念

在 Python 中有一些与其他编程语言显著不同的概念，了解这些概念对于学习 Python 编程将会很有帮助。本节将介绍 Python 中的一些重要概念，这些概念贯穿整个 Python，在本书后续章节中会详细讲解这些概念。

1.4.1　动态类型

1.3.3 小节曾介绍过，在 Python 中使用变量之前不需要预先声明，为变量赋值时会自动创建该变量，并可在后面的代码中使用它。

无论最初为变量赋的值是哪种数据类型，以后都可以将任何类型的数据赋值给该变量，这意味着 Python 中的变量没有数据类型，而赋给变量的值才有数据类型。在 Python 内部，为变量赋值的操作将在变量和值之间建立关系，这种关系实际上是变量指向特定值的一个引用。正因为如此，可以将任何类型的值赋给同一个变量，为变量赋新值后，该变量指向的原有值会自动从内存中删除，这种机制称为“垃圾收集”。

在 Python 中输入一个字面值或表达式时，都会创建一个代表该值或表达式计算结果的对象。在 Python 中任何事物都是对象，变量只是指向对象的一个引用。

下面的代码将数字 16 赋值给变量 m，然后将变量 m 赋值给变量 n，再显示 m 和 n 的值，它们都是 16。

```
>>> m = 16
>>> n = m
>>> m, n
(16, 16)
```

接下来将数字 18 赋值给变量 m，然后显示 m 和 n 的值，此时 m 的值是 18，而 n 的值还是 16。这意味着将变量 m 赋值给变量 n 时，变量 n 指向的不是变量 m，而是变量 m 指向的数字 16，所以将新的数字赋给变量 m 时，变量 n 仍然指向原来的数字。

```
>>> m = 18
>>> m, n
```

```
(18, 16)
```

上述介绍的就是 Python 中动态类型的概念，它为在程序中使用变量和数据提供了更多的灵活性，这也是使 Python 代码更简洁的原因之一。

1.4.2 可变和不可变对象、序列和映射

如果可以改变对象的值，则该对象是可变对象；如果不能改变对象的值，则该对象是不可变对象。Python 内置了多种对象类型，列表和字典是可变对象，数字、字符串和元组是不可变对象。有些对象可以包含其他对象，将这类对象称为容器，列表、元组和字典都是容器。

在 Python 中有些对象可以使用非负整数作为索引来表示对象中的各个元素，这种对象是序列，字符串、列表和元组都是序列。除了可以使用索引引用序列对象的元素，序列对象还支持切片操作，即通过给定两个索引号来引用位于这两个索引号之间的所有元素，得到的是一个新序列。无论是原序列还是通过切片得到的新序列，它们的起始索引号都是 0。

与序列使用整数作为索引号顺序访问其内部的每个元素不同，在 Python 中还有一类以“键－值”对的形式组成对象中的各个元素，这种对象是映射，字典就是映射。映射中的每一项由键和值组成，通过查找键，可以得到与其对应的值。

1.4.3 可迭代对象

“迭代”在 Python 中是非常常见的术语和操作，是指逐一遍历一个对象中的各个元素。迭代操作通常作用于序列对象，如字符串、列表和元组，以及某些非序列对象，如字典，这类对象就是可迭代对象。以字符串为例，对一个字符串进行迭代是指对该字符串中的每个字符执行特定的操作。

第2章 数　　字

Python 为数字提供了多种类型，包括整数、浮点数、复数、小数、分数、布尔值等。本章将介绍几种常用数字类型的特点、输入方法、不同的运算方式等内容。

2.1 数字的类型

在 Python 中所有可以执行数学运算的对象都是数字。“数字”这一术语是对 Python 中所有数字的统称，Python 中的数字可以是整数、浮点数、复数、布尔值等多种类型，它们在 Python 中都是一种特定的数据类型。

2.1.1 整数

没有小数点的数字是整数，在 Python 中整数类型的名称是 int。在交互模式中输入一个整数，将在下一行显示该数字。

```
>>> 16
16
```

在交互模式中输入以逗号分隔的两个整数，将在一对圆括号中显示它们，这种形式是 Python 中的元组，它也是 Python 中的一种数据类型，本书后续章节会详细介绍元组这种数据类型。

```
>>> 1, 2
(1, 2)
```

使用 Python 内置的 int 函数可以创建一个整数，只需将要输入的整数放到 int 函数右侧的圆括号中。

```
>>> int(16)
16
```

提示：在 Python 中将用于创建数据的函数称为“构造函数”，此处的 int 及本章后面介绍的 float 和 complex 都是构造函数。

通常不会使用这种方式输入整数。不过，如需将一个小数转换为整数，则可以使用

int 函数。在交互模式中输入下面的代码，将小数 1.6 转换为整数 1，直接截去小数部分。

```
>>> int(1.6)
1
```

如需将一个小数四舍五入为一个整体，可以使用 Python 内置的 round 函数。

```
>>> round(1.6)
2
```

上面介绍的数字都是十进制数，这是 Python 默认使用的数制。还可以使用特定的前缀创建二进制数、八进制数和十六进制数：

- 以 0B 或 0b 开头的数字是二进制数，除了前缀，其他部分由 0 和 1 组成。
- 以 0O 或 0o 开头的数字是八进制数，前缀是 0，后面是大写或小写的字母 O。除了前缀，其他部分由 0 ～ 7 组成。
- 以 0X 或 0x 开头的数字是十六进制数，除了前缀，其他部分由 0 ～ 9 或 A ～ F 组成，字母 A ～ F 大小写均可。

在交互模式中输入一个二进制数 110，输入后将显示对应的十进制数 6，即 $1\times2^2+1\times2^1+0\times2^0$。

```
>>> 0b110
6
```

下面的代码输入的是一个八进制数 110，输入后将显示对应的十进制数 72，即 $1\times8^2+1\times8^1+0\times8^0$。

```
>>> 0o110
72
```

下面的代码输入的是一个十六进制数 110，输入后将显示对应的十进制数 272，即 $1\times16^2+1\times16^1+0\times16^0$。

```
>>> 0x110
272
```

2.1.2 浮点数

带有一个小数点的数字是浮点数，在 Python 中浮点数类型的名称是 float。在交互模式中输入一个浮点数，将在下一行显示该数字。

```
>>> 1.6
1.6
```

使用 Python 内置的 float 函数可以创建一个浮点数，只需将要输入的浮点数放到 float 函数右侧的圆括号中即可。

```
>>> float(1.6)
1.6
```

使用 float 函数还可以将一个整数转换为浮点数，转换的结果只是简单地在整数的末尾添加一个小数点和一个 0。

```
>>> float(16)
16.0
```

2.1.3　复数

复数由“实部”和“虚部”两个部分组成，虚部使用不区分大小写的字母 j 或 J 作为后缀，两个部分之间使用 + 连接。在 Python 中浮点数类型的名称是 complex。

在 Python 中输入复数有以下几种方法。

1. 同时输入实部和虚部

在交互模式中输入下面的代码并显示输入的复数。

```
>>> 3 + 6j
(3+6j)
```

2. 只输入虚部

如果复数没有实部，则可以只输入以字母 j 或 J 作为后缀的虚部。

```
>>> 6j
6j
```

3. 使用 complex 函数

使用 Python 内置的 complex 函数创建复数，该函数的第一个参数表示实部，第二个参数表示虚部。

```
>>> complex(3, 6)
(3+6j)
```

在 Python 中复数的两个部分都以浮点数表示。在交互模式中输入下面的代码，使用 complex 对象的 real 和 imag 两个属性获得复数的实部和虚部，将返回带有小数点的数字，说明它们都是浮点数。

```
>>> complex(3, 6).real
3.0
>>> complex(3, 6).imag
6.0
```

提示：在本书后续章节中介绍类时会对属性进行详细介绍。

2.1.4 布尔值

在 Python 中布尔值只有 True 和 False 两个，它们都是布尔类型，该类型是 Python 中的一种数据类型，也是整数类型的子类型。在 Python 中布尔类型的名称是 bool。

注意：True 和 False 是 Python 中的关键字，由于 Python 严格区分大小写，所以 True 和 False 必须首字母大写，其他字母小写。

可以直接在代码中输入 True 和 False 两个布尔值，也可以使用 Python 内置的 bool 函数将任何数据转换为布尔值。在交互模式中输入下面的代码，将整数 16 转换为布尔值 True。

```
>>> bool(16)
True
```

bool 函数会将数字 0 转换为 False，而将所有非 0 数字转换为 True。

```
>>> bool(0)
False
```

如果要转换的不是数字而是字符串，则会将空字符串转换为 False，将非空字符串转换为 True。列表、元组、字典等数据类型也都遵循与字符串相同的规则，bool 函数会将空列表、空元组、空字典转换为 False。

2.1.5 检测数字的类型

前面介绍的用于创建特定数字类型的函数，它们的名称也是这些数字类型的名称。使用 Python 内置的 type 函数可以检测数字或其他数据的类型，并返回类型的名称。

在交互模式中输入下面的代码，将显示数字 168 的类型。因为该数字是整数，所以返回的类型名称是 int。

```
>>> type(168)
<class 'int'>
```

如果是浮点数 1.68，则 type 函数返回的类型名称是 float。如果是复数 16+8j，则 type 函数返回的类型名称是 complex。如果输入的是布尔值，则显示的类型名称是 bool。

```
>>> type(True)
<class 'bool'>
```

2.2 对数字执行运算

将数字和运算符混合在一起就构成了表达式，每个表达式都可以计算出一个值。在

Python 中可以使用算术运算符、比较运算符和布尔运算符对数字执行不同类型的运算，也可以混合使用这些运算符创建复杂的表达式。使用圆括号可以改变运算符默认的运算顺序。

2.2.1　算术运算

使用算术运算符可以对数字执行常规的算术运算，计算结果是一个数字，其类型由参与计算的两个数字的类型决定。Python 中的算术运算符如表 2-1 所示。

表 2-1　算术运算符

运　算　符	名　　称	功　　能	示　　例
+	加	计算两个数字之和	13 + 5 的结果是 18
-	减	计算两个数字之差	13 - 5 的结果是 8
*	乘	计算两个数字之积	13 * 5 的结果是 65
/	除	计算两个数字之商	13 / 5 的结果是 2.6
//	整除	计算两个数字的整数商	13 // 5 的结果是 2
%	求余（模）	计算两个数字的余数	13 % 5 的结果是 3
**	乘方（幂）	计算一个数字的指定次幂	13 ** 5 的结果是 371293

使用 / 运算符执行除法运算时，即使两个数字能够整除，计算结果也是一个浮点数，即在商的末尾添加小数点和一个 0。例如，在交互模式中输入下面的表达式，计算结果是 2.0 而非 2。

```
>>> 10 / 5
2.0
```

浮点数执行算术运算时可能会产生计算误差。在交互模式中输入下面的表达式，计算结果并非 1.8，而是一个非常接近 1.8 的数字。

```
>>> 0.6 + 0.6 + 0.6
1.7999999999999998
```

解决浮点数误差的一种方法是使用 Python 内置的 round 函数，根据参与计算的所有数字中的最大小数位数，来设置 round 函数的第二个参数为计算结果保留的小数位数。下面的代码为计算结果保留一位小数。

```
>>> round(0.6 + 0.6 + 0.6, 1)
1.8
```

下面的代码为计算结果保留 3 位小数。

```
>>> round(0.6 + 0.66 + 0.666, 3)
1.926
```

使用 Python 内置函数可以实现表 3-1 中的一些算术运算。例如，使用 pow 函数可以实现 ** 运算符的功能，使用 divmod 函数可以同时实现 // 和 % 两个运算符的功能。

```
>>> divmod(13, 5)
(2, 3)
```

2.2.2 比较运算

使用比较运算符可以比较数字的大小，计算结果是一个布尔值。Python 中的比较运算符如表 2-2 所示。

表 2-2 比较运算符

运 算 符	名 称	功 能	示 例
==	等于	比较两个数字是否相等	5 == 13 的结果是 False
!=	不等于	比较两个数字是否不相等	5 != 13 的结果是 True
<	小于	比较第一个数字是否小于第二个数字	5 < 13 的结果是 True
<=	小于或等于	比较第一个数字是否小于或等于第二个数字	5 <= 13 的结果是 True
>	大于	比较第一个数字是否大于第二个数字	5 > 13 的结果是 False
>=	大于或等于	比较第一个数字是否大于或等于第二个数字	5 >= 13 的结果是 False

在 Python 中可以在一个表达式中使用多个比较运算符比较多个数字。在交互模式中输入下面的代码，结果是 False。这是因为 1 < 3 < 2 相当于 1 < 3 和 3 < 2，只有两个表达式都成立，结果才是 True，否则结果是 False。

```
>>> 1 < 3 < 2
False
```

2.2.3 布尔运算

使用布尔运算符可以对表达式执行布尔运算，并返回一个布尔值 True 或 False。Python 中的布尔运算符如表 2-3 所示。

表 2-3　算术运算符

运　算　符	名　　称	功　　能	示　　例
and	与	第一个表达式为 False 则返回 False，否则返回第二个表达式的值	1 > 2 and 3 < 6 返回 False
or	或	第一个表达式为 True 则返回 True，否则返回第二个表达式的值	1 > 2 or 3 < 6 返回 True
not	非	如果表达式为 False 则返回 True，否则返回 False	not 1 > 2 返回 True

实际上，Python 中的布尔运算符并不要求表达式的返回值必须是布尔值 True 或 False，也可以对字面量或变量使用布尔运算符，此时将根据布尔运算符的类型和布尔运算符连接的两个值来返回其中一个值。

在交互模式中输入下面的代码，将返回 6，这是因为 and 运算符左侧的值是数字 1，1 相当于布尔值 True，由于 and 运算符两侧的值都是 True 时，才会返回 True，所以，此时需要检测 and 运算符右侧的值并返回该值。

```
>>> 1 and 6
6
```

在交互模式中输入下面的代码，将返回 1，这是因为只要 or 运算符两侧的值有一个是 True 就会返回 True，而本例中的数字 1 相当于 True，所以，无须检测 or 运算符右侧的值，而是直接返回左侧的数字 1。

```
>>> 1 or 6
1
```

下面的代码利用布尔运算符的这种特性，为存储字符串的变量 name 设置了一个默认值，当变量 name 为空时，将返回该默认值。

```
>>> name = ''
>>> name or 'admin'
'admin'
```

2.2.4　使用括号改变运算顺序

当一个表达式包含多种运算符时，表达式中各个部分的运算顺序由各个运算符的优先级决定。前几节介绍的 3 类运算符的优先级由高到低排列如下：

```
算术运算符 > 比较运算符 > 布尔运算符
```

同一类运算符中各个运算符的优先级如下。

- 算术运算符：** 运算符的优先级最高，*、/、// 和 % 4 个运算符的优先级次之，+ 和 − 两个运算符的优先级最低。

- 比较运算符：所有运算符的优先级相同。
- 布尔运算符：所有运算符的优先级相同。

在交互模式中输入下面的代码，表达式的计算结果是 66，先计算优先级较高的乘法，然后计算优先级较低的加法。

```
>>> 16 + 10 * 5
66
```

如需提升运算符的优先级，可以将希望先计算的表达式放在一对圆括号中。下面的代码先计算圆括号中的加法，再计算乘法，计算结果是 130。

```
>>> (16 + 10) * 5
130
```

即使不需要提升运算符的优先级，在一个复杂表达式中使用括号仍然是一个好方法，因为可以使表达式中的各个部分更清晰，提高代码的可读性。

```
>>> (1 < 3) and (6 > 5) and (7 != 8)
```

2.2.5 不同数字类型的混合运算

如果在一个表达式中对不同类型的数字求值，则复杂度较低的数字类型会自动向复杂度较高的数字类型转换。在 Python 中，整数的复杂度最低，浮点数次之，复数的复杂度最高。

下面的代码计算一个整数和一个浮点数之和，由于浮点数具有更高的复杂度，所以，整数会先转换为浮点数，然后计算两个数字之和，计算结果也是一个浮点数。

```
>>> 6 + 1.6
7.6
```

2.3 在不同数制之间转换

使用 Python 内置的几个函数，可以很容易将一个数字在十进制与其他数制之间转换，这些函数是 bin、oct、hex 和 int。

2.3.1 将十进制数字转换为其他进制数字

使用 bin、oct 和 hex 函数可以将一个十进制数字分别转换为二进制数字、八进制数字和十六进制数字，转换结果的数据类型是 str（字符串）。

在交互模式中输入下面的代码，将十进制数 16 转换为二进制数 0b10000。

```
>>> bin(16)
'0b10000'
```

下面的代码将十进制数 16 转换为八进制数 0o20。

```
>>> oct(16)
'0o20'
```

下面的代码将十进制数 16 转换为十六进制数 0x10。

```
>>> hex(16)
'0x10'
```

在上述 3 个示例中，返回的结果都使用单引号包围，说明它们都是字符串。第 4 章将详细介绍字符串的相关内容。

2.3.2　将其他进制数字转换为十进制数字

如需将字符串形式的其他进制数字转换为十进制数，可以使用 Python 内置的 int 函数，这相当于 3.3.1 小节的逆操作。使用 int 函数转换数制时，该函数的第一个参数表示要转换的字符串格式的数字，第二个参数表示该数字的数制。

在交互模式中输入下面的代码，使用 int 函数将二进制数 0b10000 转换为十进制数 16。该函数的第一个参数放在一对单引号中，说明它是一个字符串。第二个参数是 2，说明第一个参数表示的是一个二进制数字。

```
>>> int('0b10000', 2)
16
```

下面的代码将八进制数 0o20 转换为十进制数 16。

```
>>> int('0o20', 8)
16
```

下面的代码将十六进制数 0x10 转换为十进制数 16。

```
>>> int('0x10', 16)
16
```