
# 第 1 章

# 搭建Python开发环境

工欲善其事，必先利其器。Python的学习过程少不了代码开发环境，可以帮助开发者加快开发速度，提高效率。本章介绍Python可视化编程的基础，包括软件的安装、搭建代码开发环境以及初步认识Python程序等。本书中使用的环境是基于Python 3.10.9的Anaconda，是截至2023年5月的新版本。

## 1.1 集成开发工具Anaconda

Anaconda是Python的一个集成管理工具或系统，它把Python进行相关数据计算与分析所需要的包都集成在了一起，用户只需要安装Anaconda，即可使用Python语言及其相关的分析包进行可视化分析。

### 1.1.1 什么是 Anaconda

Anaconda是一个用于科学计算的Python发行版，支持Linux、macOS和Windows系统，提供了包管理与环境管理的功能，可以很方便地解决多版本Python并存、切换以及各种第三方包的安装问题。Anaconda利用工具/命令conda来进行包和环境的管理，并且其中已经包含Python语言本身和相关的配套工具。

Anaconda是一个打包的集合，里面包含NumPy、Pandas、Matplotlib等数据科学相关的开源包，在数据分析、数据可视化、大数据和人工智能等多方面都有涉及，包括Scikit-Learn、TensorFlow和PyTorch等。

Anaconda官方网站对其功能的概括如图1-1所示。


图 1-1　主要的机器学习库

Anaconda的优点总结起来就8个字：省时省心，分析利器。

- 省时省心：Anaconda 通过管理工具包、开发环境、Python 版本大大简化了用户的工作流程，不仅可以方便地安装、更新、卸载工具包，而且能自动安装相应的依赖包，同时还能使用不同的虚拟环境隔离不同要求的项目。
- 分析利器：Anaconda 官方网站中是这么宣传自己的，适用于企业级大数据分析的 Python 工具，其包含 720 多个数据科学相关的开源包，在数据可视化、机器学习、深度学习等多方面都有涉及，不仅可以进行数据分析，还可以用在大数据和人工智能领域。

### 1.1.2　安装 Anaconda

Anaconda的安装过程比较简单，首先进入Anaconda的官方网站下载需要的版本，这里选择Windows版本的64-Bit Graphical Installer，如图1-2所示。如果官方网站下载速度较慢，还可以到清华大学开源软件镜像站下载。

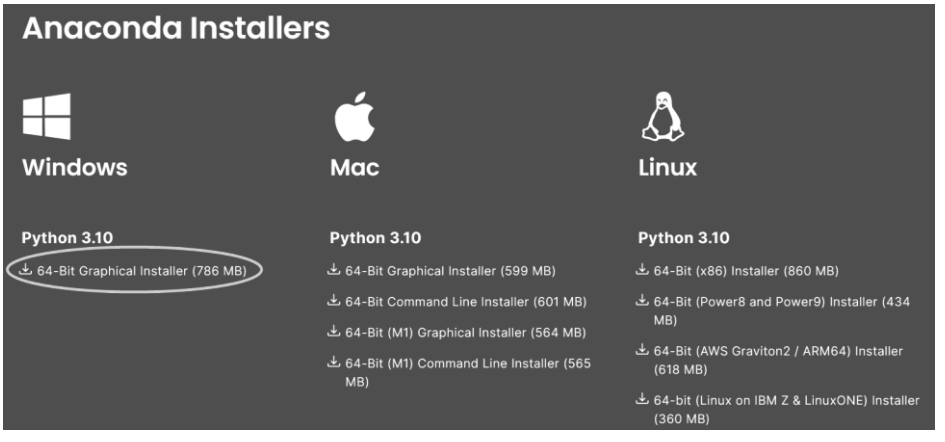



图 1-2　下载 Anaconda

软件下载好后，以管理员身份运行Anaconda3-2023.03-1-Windows-x86_64.exe文件，单击Next按钮，安装过程比较简单，最后单击Finish按钮即可，主要安装过程如图1-3所示。

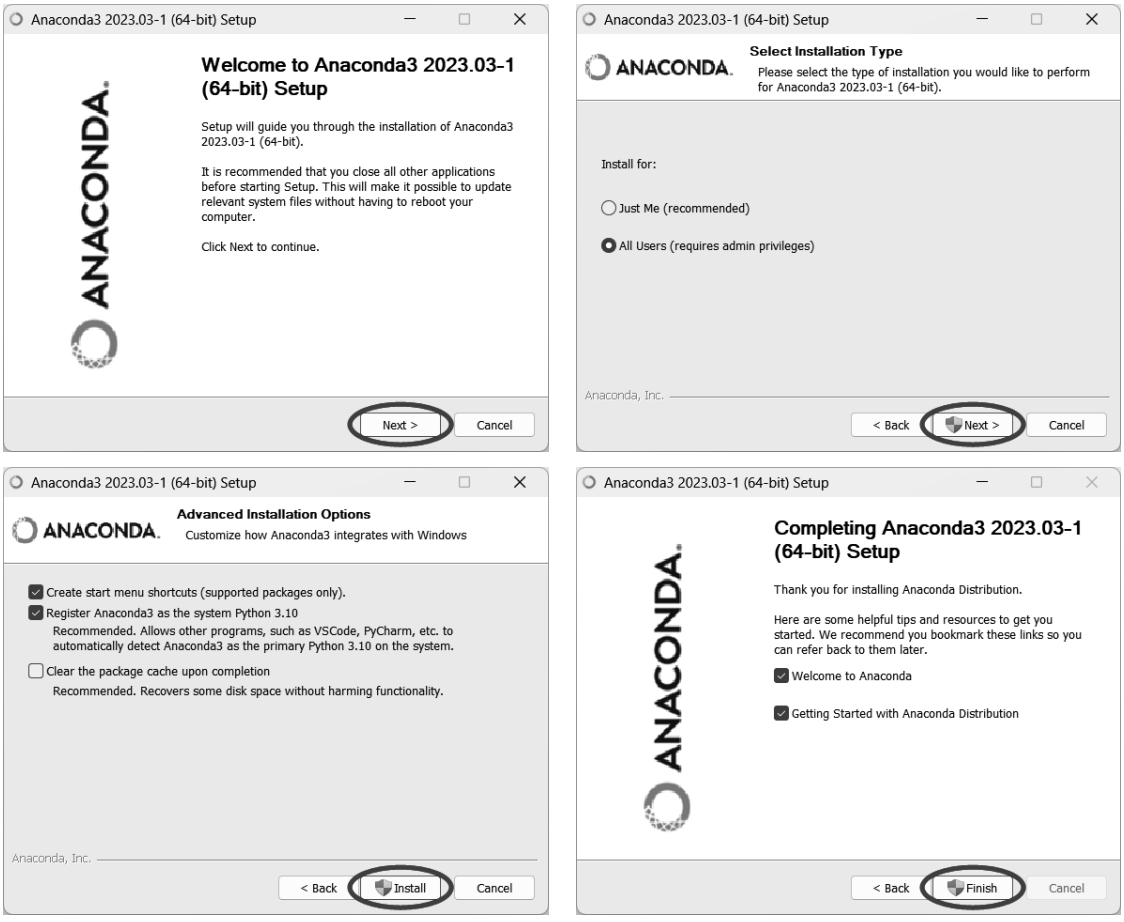



图 1-3　安装 Anaconda

安装结束后，正常情况下会在计算机的“开始”菜单中出现Anaconda3 (64-bit)文件夹，单击其下方的Anaconda Prompt，然后输入python，如果出现Python的版本信息，就说明安装成功，如图1-4所示。

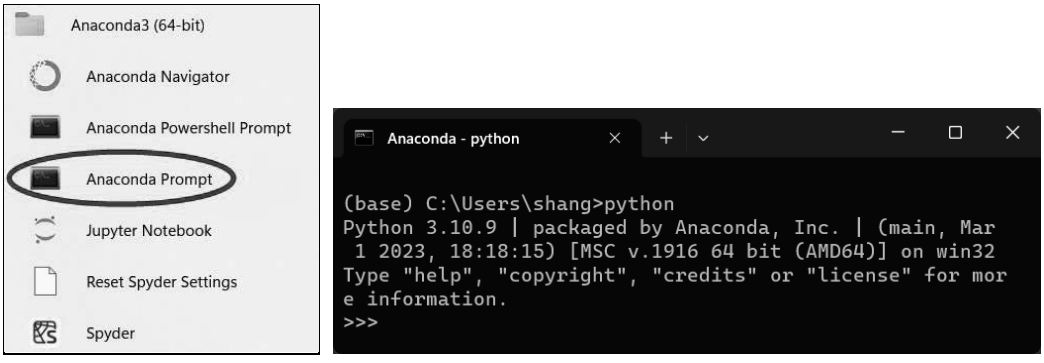



图 1-4　查看 Python 版本

Anaconda是一个基于Python的数据处理和科学计算平台，内置了许多非常有用的第三方库，安装Anaconda后，就相当于把Python和NumPy、Pandas、Scipy、Matplotlib等常用的库自动安

装好了。但是，如果选择非集成环境Python的话，那么还需要使用pip install命令逐个安装各种库，尤其是对于初学者来说，这个过程是非常痛苦的。

Python第三方包众多，但是调用包的时候，有时会遇到问题，比如安装包失败，安装速度很慢，很影响自己的工作进度，可能还会报错，这是由于我们在CMD窗口使用pip安装的时候，默认下载的是国外资源，会由于网速不稳定甚至没有网速而出现问题，解决办法如下：

（1）首先搜索需要安装的包名称，然后去国外的网站进行下载。在本地安装包时，用户可以在窗口中看到系统会自动安装相关包，但是可能也会出现下载失败的情况，出现这种情况时，只需继续去国外网站下载缺失的包，然后在本地安装即可。

（2）第二种方法是一劳永逸的方法，选择国内镜像源，相当于从国内的一些机构下载所需的Python第三方包。那么如何选择国内镜像源，如何配置呢？首先在计算机中显示隐藏的文件，并找到C:\Users\shang\AppData\Roaming，其中shang是个人的计算机的名称，然后在该路径下新建一个文件夹，命名为pip，在pip文件夹中新建一个TXT格式的文本文档，将下面这些代码复制到该文本文档中，关闭并保存。最后将文本文档重新命名为pip.ini，这样就创建了一个配置文件。

```
[global]
timeout = 60000
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
[install]
use-mirrors = true
mirrors = https://pypi.tuna.tsinghua.edu.cn
```

文档中的链接地址还可以更换成如下地址：

- 阿里云：http://mirrors.aliyun.com/pypi/simple/。
- 中国科技大学：https://pypi.mirrors.ustc.edu.cn/simple/。
- 豆瓣（douban）：http://pypi.douban.com/simple/。
- 清华大学：https://pypi.tuna.tsinghua.edu.cn/simple/。
- 中国科学技术大学：http://pypi.mirrors.ustc.edu.cn/simple/。

通过以上操作，后续我们使用pip安装第三方包的时候，就默认选择国内源进行安装，这样安装速度较快。

## 1.2 常用代码开发工具

Python数据分析的常用代码开发工具有Spyder、JupyterLab和PyCharm。由于本书介绍的是数据可视化，经常需要展示一些图表，相对而言，个人认为JupyterLab这个开发工具比较合适。本节我们会逐一介绍上述3个代码开发工具，读者根据自己的喜好选择其一即可。

### 1.2.1　简单易用的 Spyder

安装Anaconda后，默认会安装Spyder工具，因此不需要再单独安装。Spyder是Python的作者为它开发的一个简单的集成开发环境，与其他的开发环境相比，它最大的优点就是模仿MATLAB的“工作空间”的功能，可以方便地观察和修改数组的值。

Anaconda安装成功后，默认会将Spyder的启动程序添加到环境变量中，可以通过单击计算机的“开始”按钮，再单击其快捷方式启动Spyder，也可以在命令提示符中输入spyder命令启动Spyder，如图1-5所示。

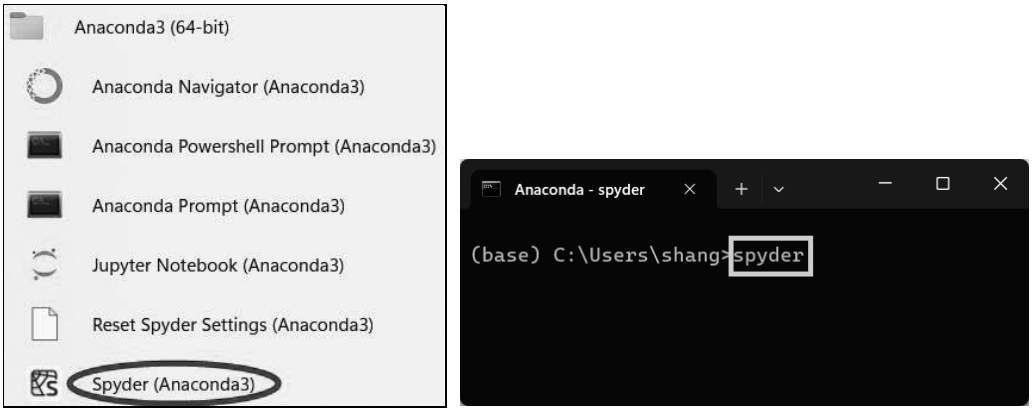



图 1-5　启动 Spyder

Spyder界面由多个窗格构成，包括Editor、Console、Variable explorer、File explorer、Help等，用户可以根据自己的喜好调整它们的位置和大小，如图1-6所示。

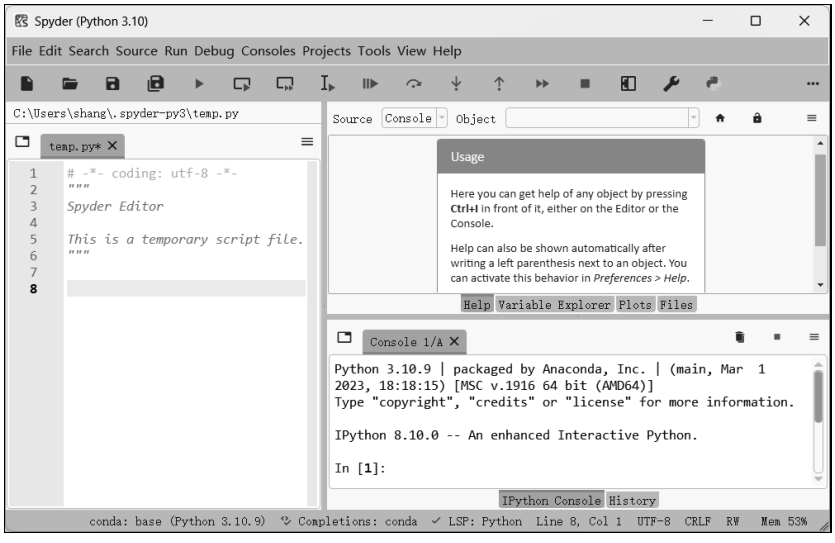



图 1-6　Spyder 界面

高效使用Spyder窗格可以方便我们进行Python代码的开发。表1-1中列出了Spyder的主要窗格及其作用。

表 1-1　Spyder 的主要窗格及其作用

| 窗格名称 | 作　　用 |
| --- | --- |
| Editor | 编辑程序，可用标签页的形式编辑多个程序文件 |
| Console | 在别的进程中运行的Python控制台 |
| Variable Explorer | 显示Python控制台中的变量列表 |
| File Explorer | 文件浏览器，用于打开程序文件或者切换当前路径 |
| Help | 查看对象的说明文档 |

在使用Spyder进行代码开发时，需要在Editor窗格的空白区域编写代码，例如print("Hello Python!")，编写完毕后，可以通过工具栏上的运行按钮运行程序，也可以按快捷键F5，我们可以在Spyder界面右下方的Console窗格中看到结果或报错信息等，如图1-7所示。

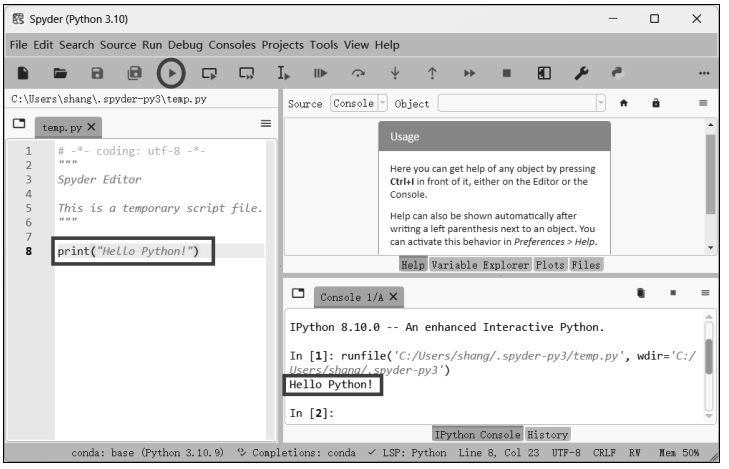



图 1-7　运行示例程序

快捷键可以方便我们进行代码的开发和测试，Spyder的常用快捷键如表1-2所示。此外，可以通过Tools→Preferences→Keyboard Shortcut查看所有快捷键。

表 1-2　Spyder 的主要快捷键

| 快捷键 | 中文名称 |
| --- | --- |
| Ctrl+R | 替换文本 |
| Ctrl+1 | 单行注释，单次注释，双次取消注释 |
| Ctrl+4 | 块注释，单次注释，双次取消注释 |
| F5 | 运行程序 |
| Ctrl+P | 文件切换 |
| Ctrl+L | 清除Shell |
| Ctrl+I | 查看某个函数的帮助文档 |
| Ctrl+Shift+V | 调出变量窗口 |
| Ctrl+up | 回到文档开头 |
| Ctrl+down | 回到文档末尾 |

### 1.2.2　功能强大的 JupyterLab

JupyterLab是Jupyter Notebook的新一代产品，它集成了更多功能，是使用Python（R、Julia、Node等其他语言的内核）进行代码演示、数据分析、数据可视化等很好的工具，对Python的愈加流行和在AI领域的领导地位有很大的推动作用，它是本书默认使用的代码开发工具。

安装Anaconda后，默认安装JupyterLab工具，启动JupyterLab的方法比较简单，只需要在命令提示符中输入jupyter lab命令即可。JupyterLab程序启动后，浏览器会自动打开编程窗口，默认地址是http://localhost:8888。

可以看出，JupyterLab页面左边是存放笔记本的工作路径，右边就是我们需要创建的笔记本类型，包括Notebook和Console，还可以创建Text File、Markdown File、Show Contextual Help等其他类型的文件，如图1-8所示。

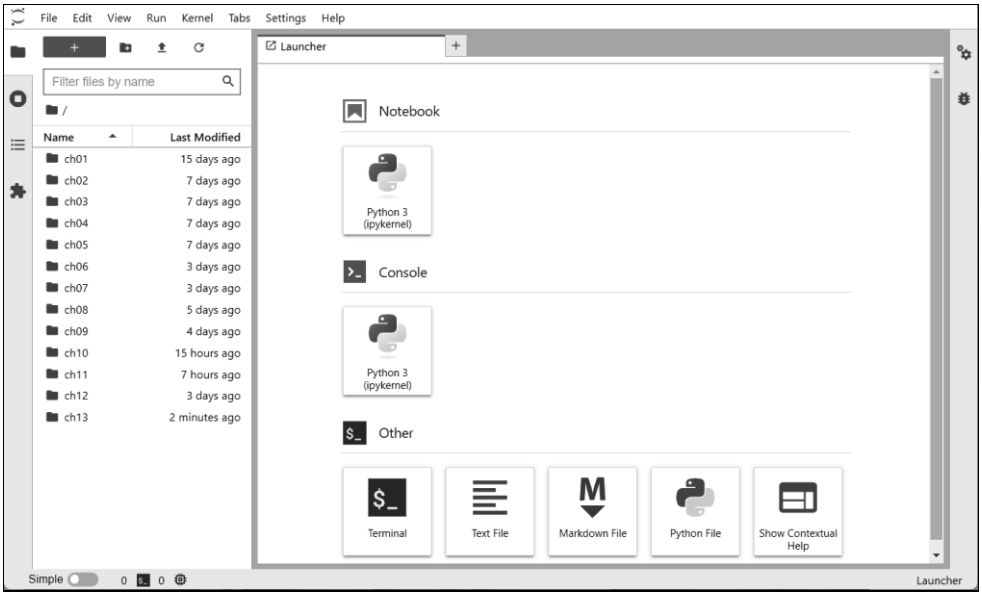



图 1-8　JupyterLab 界面

我们可以对JupyterLab的参数进行修改，如对远程访问、工作路径等进行设置，配置文件位于C盘系统用户名下的.jupyter文件夹中，文件名称为jupyter_notebook_config.py。

如果配置文件不存在，就需要自行创建，单击图1-8中的Other选项下的Terminal，使用jupyter notebook --generate-config命令即可生成配置文件，并且会显示出文件的存储路径及名称，如图1-9所示。

```
Windows PowerShell
版权所有 (C)  Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！ https://aka.ms/PSWindows

PS C:\Users\shang> jupyter notebook --generate-config
Writing default config to: C:\Users\shang\.jupyter\jupyter_notebook_config.py
PS C:\Users\shang>
```

图 1-9　配置 JupyterLab

JupyterLab提供了一个命令来设置密码：jupyter notebook password，生成的密码存储在jupyter_notebook_config.json文件中，下方将会显示文件的路径及名称，如图1-10所示。

```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！ https://aka.ms/PSWindows

PS C:\Users\shang> jupyter notebook --generate-config
Writing default config to: C:\Users\shang\.jupyter\jupyter_notebook_config.py
PS C:\Users\shang> jupyter notebook password
Enter password:
Verify password:
[NotebookPasswordApp] Wrote hashed password to C:\Users\shang\.jupyter\jupyter_notebook_config.json
PS C:\Users\shang>
```

图 1-10　配置 JupyterLab 密码

如果需要允许远程登录，还需要在jupyter_notebook_config.py中找到下面几行代码，取消注释并根据项目的实际情况进行修改，修改后的配置如下：

```
c.NotebookApp.ip = '*'
c.NotebookApp.open_browser = False
c.NotebookApp.port = 8888
```

如果需要修改JupyterLab的默认工作路径，需要找到下面的代码，取消注释并根据项目的实际情况进行修改，本书修改后的配置如下：

```
c.NotebookApp.notebook_dir = u'D:\\Python 数据可视化之 Matplotlib 与 Pyecharts 实战'
```

待需要配置的参数都修改完毕后，需要重新启动JupyterLab才能生效，启动后首先需要我们输入刚刚配置的密码，如图1-11所示。

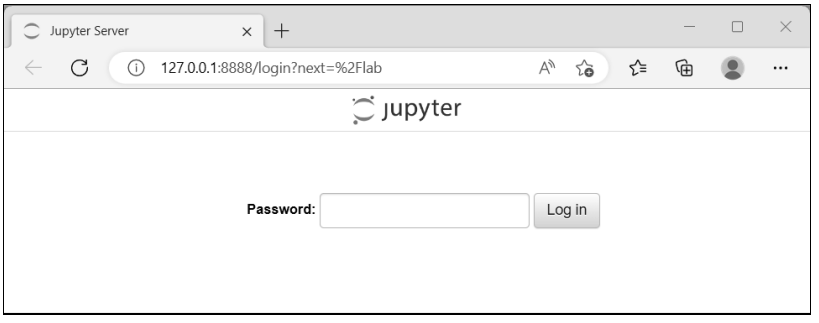



图 1-11　输入密码

输入密码后，单击Log in按钮，在新的编程窗口中，左边的工作路径会发生变化，现在呈现的就是D盘的“Python数据可视化之Matplotlib与Pyecharts实战”文件夹。

### 1.2.3　高效流行的 PyCharm

PyCharm是一个比较常见的Python代码开发工具，可以帮助用户在使用Python语言开发时提高效率，比如调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制等。

在开始安装PyCharm之前，需要确保计算机上已经安装了Java 1.8以上的版本，并且已配置好环境变量。安装PyCharm后，还需要配置其代码开发环境，首次启动PyCharm，会弹出配置窗口，如图1-12所示。

如果之前使用过PyCharm并有相关的配置文件，则在此处选中Config or installation folder单选按钮；如果没有使用过PyCharm，保持默认设置，即选中Do not import settings单选按钮，然后单击OK按钮。在同意用户使用协议页面，勾选确认同意选项，并单击Continue按钮，如图1-13所示。

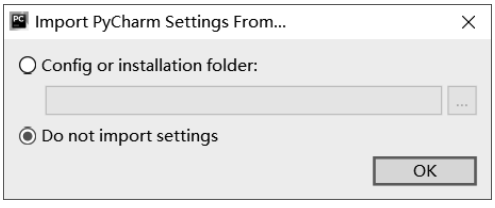



图 1-12　软件配置窗口


图 1-13　用户使用协议

确定是否需要进行数据共享，可以直接单击Don’t send按钮，如图1-14所示。选择主题，左边为黑色主题，右边为白色主题，根据需要选择即可，这里我们选择Light类型，并单击Next:Featured plugins按钮继续后面的插件配置，如图1-15所示。

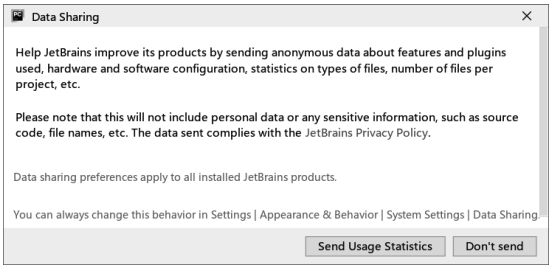



图 1-14　数据共享设置

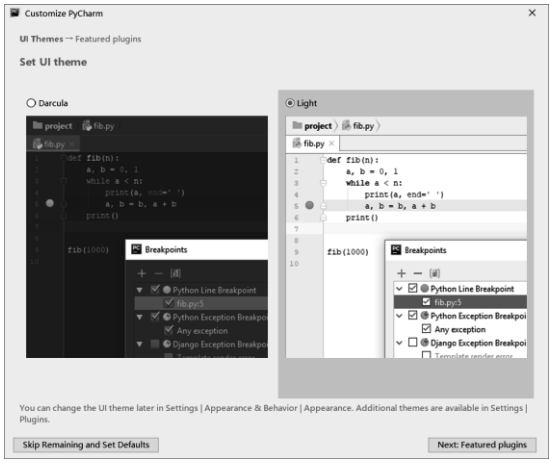



图 1-15　选择软件主题

PyCharm设置完成后，单击Create New Project选项，就可以开始创建一个新的Python项目。在New Project页面，在Location中设置项目路径并选择解释器。注意，这里默认使用Python的虚拟环境，即第一个New environment using选项，再单击Create按钮，如图1-16所示。

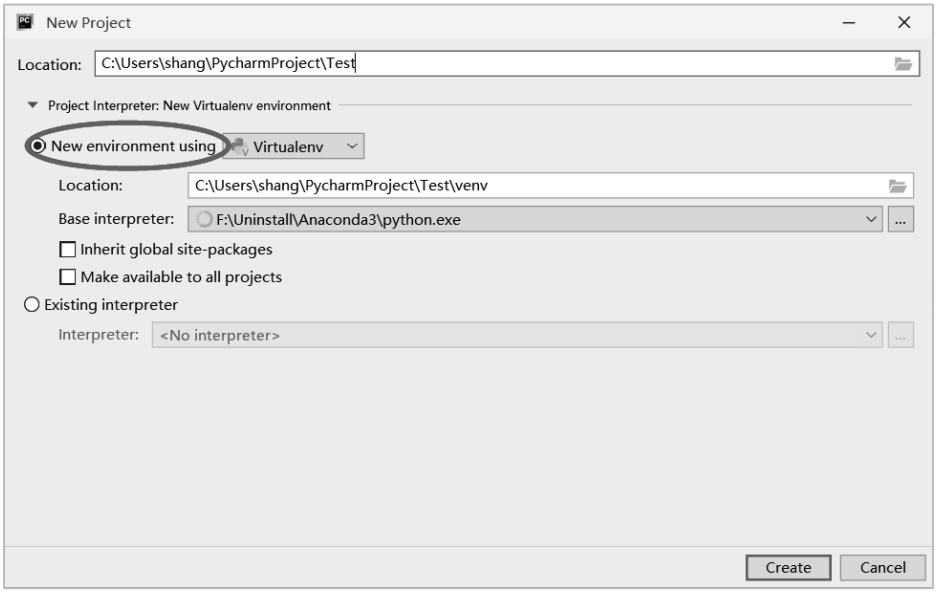



图 1-16　配置新项目

如果不使用虚拟环境，一定要修改，则需要选择第二个Existing interpreter选项，然后选择需要添加的解释器，再单击Create按钮，如图1-17所示。在弹出的PyCharm欢迎页面，取消勾选Show tips on startup复选框，不用每次都打开欢迎界面，单击Close按钮，退出使用指导过程。

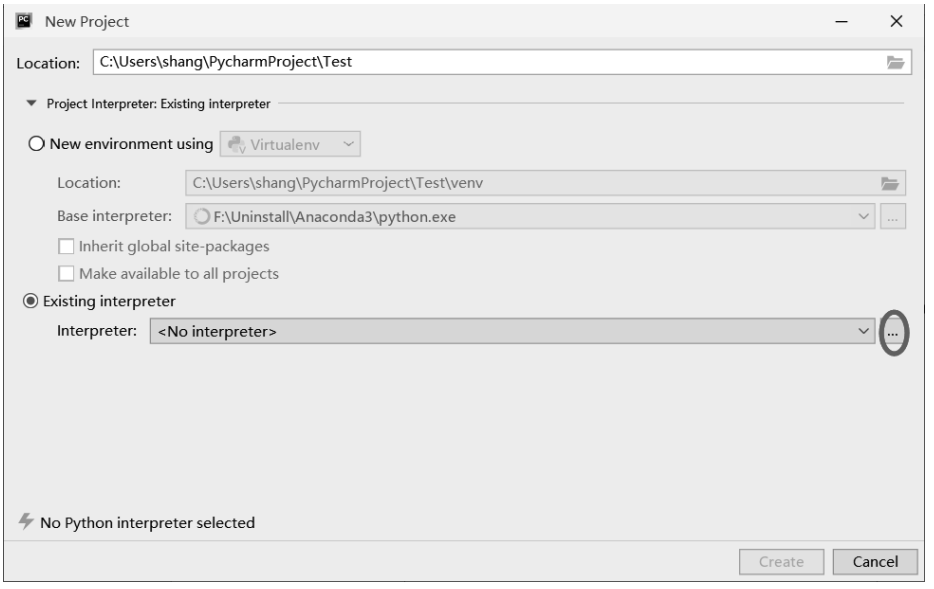



图 1-17　配置解释器

创建Python文件，在项目名称的位置右击，依次选择New→Python File，输入文件名称Hello，并按Enter键即可，如图1-18所示。

在文件中输入代码：print("Hello Python!");，然后在文件中的任意空白位置右击，选择Run 'Hello'选项，在界面的下方显示Python代码的运行结果，如图1-19所示。

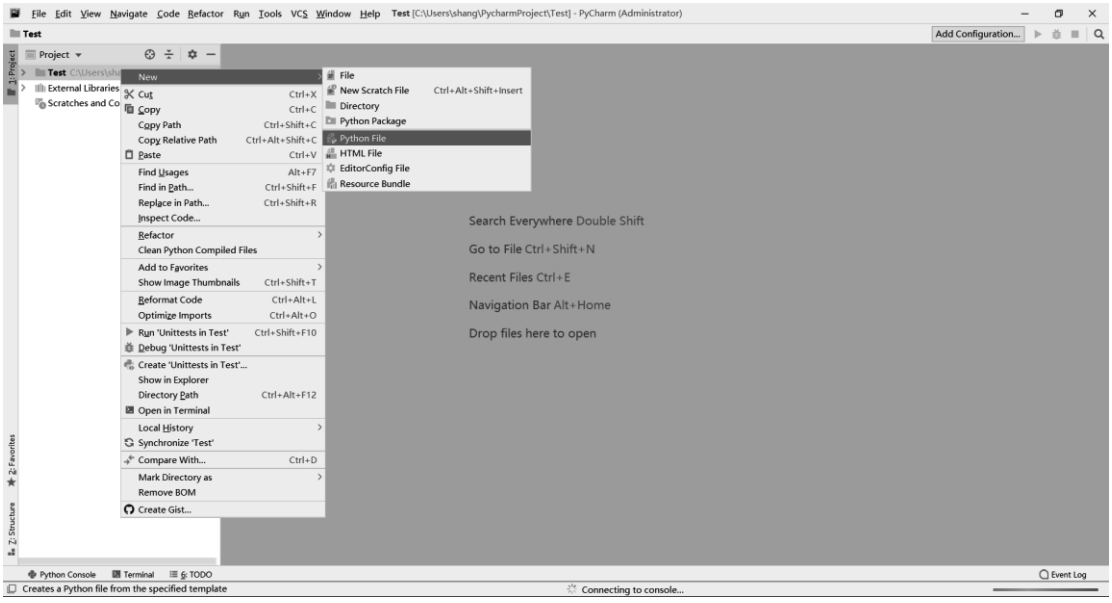



图 1-18　新建 Python 文件

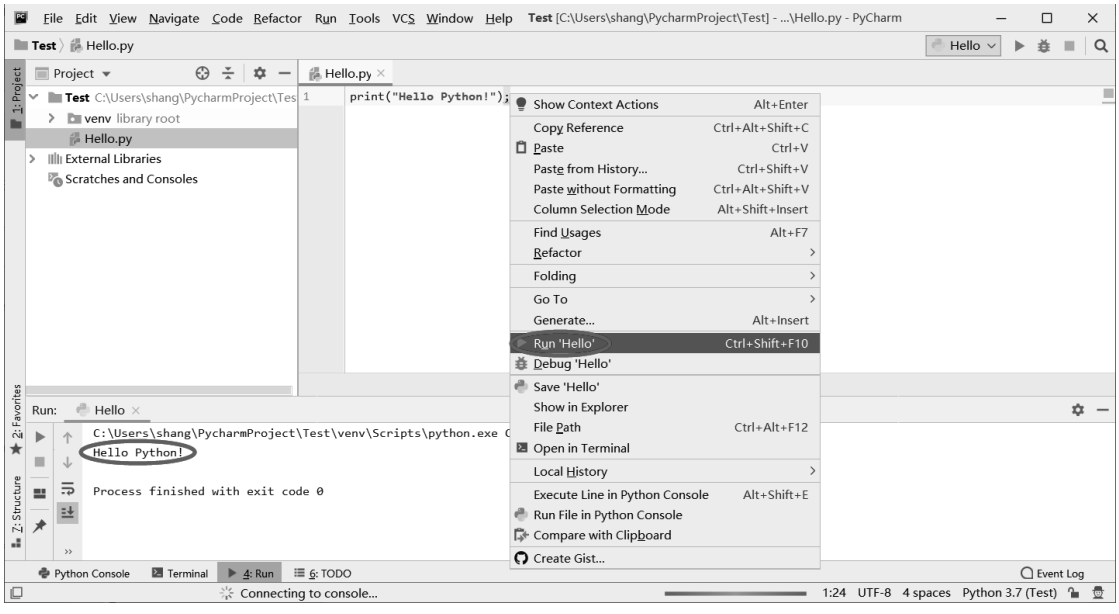



图 1-19　运行 Python 代码

# 1.3　认识Python程序

使用Python进行数据可视化必须掌握Python编程相关知识，本节先来认识一个Python程序的特点及其构成。

### 1.3.1 一个简单的 Python 程序

首先用一个简单的Python程序计算圆的面积，示例代码如下：

```
import math
r = float(input("请输入圆的半径："))
area = math.pi * math.pow(r, 2)
print("圆的面积为：", area)
```

这个程序可以计算给定半径的圆的面积。程序首先使用import关键字导入Python的math模块，以便在程序中使用数学常量pi和pow函数，pow函数用来计算半径的平方。

在Python中，import语句用来引入其他模块或库的功能，使得我们可以在自己的程序中直接使用这些功能，而不需要重新编写它们。

当我们在程序中使用import关键字时，Python会执行以下操作：

（1）Python会在当前工作目录中寻找指定的模块或库。

（2）如果在当前目录下没有找到指定的模块或库，则会在Python的标准库路径中继续寻找。

（3）如果在Python的标准库路径中也没有找到指定的模块或库，则会尝试查找用户自定义的路径。

（4）一旦Python找到指定的模块或库，就会加载和执行它，并将它的命名空间中的所有对象全部导入当前程序的命名空间中。对于比较大的模块或库，通常只需要引入其中的一部分功能。在这种情况下，可以使用import语句后面跟上from关键字和模块或库中需要引入的具体功能。例如，如果我们只需要使用Python的math库中的pi常数和sqrt函数，那么可以这样写：

```
from math import pi, sqrt
```

这样，我们就只能使用math库中的pi常数和sqrt函数，而不是整个math库的所有功能。这样可以提高程序的运行效率和可读性。

然后，代码会要求用户输入圆的半径。我们使用float()函数将用户输入的字符串转换为浮点数，并将其保存在变量r中。之后，使用公式$\pi r^2$计算圆的面积，并将结果保存在变量area中。

最后，程序会使用print()函数将计算出的圆的面积打印在屏幕上。

要运行这个程序，我们可以将代码保存在一个以.py为扩展名的文件中，例如area.py。打开控制台或终端，并在程序所在的目录下输入以下命令：

```
python area.py
```

程序将会交互式运行，在控制台上提示用户输入半径的值。

这是一个非常简单的示例，但它演示了Python的基本语法和功能。我们看到，一个Python程序包括很多内容，如变量、函数、字符串等，这些概念我们会在后续的内容中详细介绍。

如果你刚开始学习Python，请试着把这个程序打印出来并检查每一行的作用，以便更好地理解Python程序的工作方式。

### 1.3.2　Python 的常量和变量

在Python中，变量和常量是两种不同的概念。

变量是在程序执行过程中可以改变值的标识符。在Python中，变量使用等号“=”进行赋值。例如：

```
x = 10                  #把 10 赋值给变量 x
name = "ChatGPT"        #把字符串"ChatGPT"赋值给变量 name
```

变量名可以包含字母、数字和下画线，但不能以数字开头。Python的变量名区分大小写，因此name和Name是两个不同的变量。

常量是指在程序中永远不会改变值的标识符。在Python中，常量通常使用全大写字母表示。Python并没有约定好如何定义常量，但约定俗成的是，使用全大写字母来表示常量，例如：

```
PI = 3.1415926          #把浮点数 3.1415926 赋值给常量 PI
MAX_COUNT = 100         #把整数 100 赋值给常量 MAX_COUNT
```

虽然Python中没有真正意义上的常量，但程序员可以通过这种方式来告诉读者，这是一个不会改变值的标识符。

### 1.3.3　编写 Python 程序的注意事项

编写Python程序时，请注意以下几个要点。

（1）注意缩进：在Python中，缩进非常重要。程序中的每个语句块必须使用相同数量的空格，否则会出现语法错误。建议在每个缩进层次中使用4个空格。

（2）导入模块：Python提供了许多内置模块和第三方库。在程序中导入所需的模块可以使代码更加简洁和易于维护。通常，import语句应该放在程序的开头。

（3）变量命名规范：Python中的变量名应该清晰、简洁和易于理解。建议使用有意义的名称来描述变量的用途，并使用小写字母、下画线和数字。变量名应该以字母或下画线开头，不能以数字开头。函数名同样按照这个规则。

（4）注释：Python中的注释可以提高代码的可读性。单行注释可以使用井号（#），多行注释可以使用三引号（'''...'''）。

（5）错误处理：编写正确的Python程序需要包括错误处理。try-except块可以捕获和处理程序中的异常。在try块中包含可能引发异常的代码，在except块中包含异常处理代码。

以下是一个简单的Python代码示例，其中包含缩进和注释。

```
#创建一个变量 sum，初始化为 0
sum = 0
#for 循环，变量 i 在 1～100 范围内循环
for i in range(1,101):
    #在循环体内，每一次都将 i 加到 sum 上
```

```
        sum = sum + i
    print(sum)    #输出 sum 的值
```

在此示例中，sum = sum + i语句被缩进了4个空格。这种缩进方式是一种符合PEP 8规范的代码风格。

注释使用井号（#）表示，示例中的注释可使代码更易于理解和维护。

在Python中，通常是一行写完一条语句，如果要写多条语句，就需要使用分号分隔。此外，如果语句很长，还可以使用反斜杠（\）来实现换行，但是在[]、{}或()中的多行语句不需要使用反斜杠，示例代码如下：

```
    order_mon = 91; order_tue = 78; order_wed = 83; order_thu = 85; order_fri = 82;
order_sat = 129; order_sun = 116
    order_total = order_mon + order_tue + order_wed + \
                  order_thu + order_fri + order_sat + order_sun
    order_day = ["order_mon", "order_tue", "order_wed", "order_thu",
                 "order_fri", "order_sat", "order_sun"]
```

# 1.4 包管理工具pip

在实际编程工作中，会用到很多模块和开发工具或第三方包，此时可以使用pip来安装它们，pip是最常用的Python第三方包管理工具。下面介绍如何通过pip进行第三方包的安装、更新、卸载等操作。

安装单个第三方包的命令如下：

```
    pip install packages
```

安装多个库包，需要将包的名字用空格隔开，命令如下：

```
    pip install package_name1 package_name2 package_name3
```

安装指定版本的包，命令如下：

```
    pip install package_name==版本号
```

当要安装一系列包时，如果写成命令可能会比较麻烦，此时可以把要安装的包名及版本号写到一个文本文件中。例如，文本文件的内容与格式如下：

```
    arrow==1.2.2
    astropy==5.1
    astunparse==1.6.3
    atomicwrites==1.4.0
    anaconda-client==1.11.0
    anaconda-navigator==2.3.1
```

然后使用-r参数安装文本文件下的包，命令如下：

```
pip install -r 文本文件名
```

查看可升级的第三方包，命令如下：

```
pip list -o
```

更新第三方包，命令如下：

```
pip install -U package_name
```

使用pip工具可以很方便地卸载第三方包，卸载单个包的命令如下：

```
pip uninstall package_name
```

批量卸载多个包的命令如下：

```
pip uninstall package_name1 package_name2 package_name3
```

卸载一系列包的命令如下：

```
pip uninstall -r 文本文件名
```

此外，在JupyterLab中也可以很方便地使用pip工具，在JupyterLab窗口中单击Console控制台下的Python 3启动按钮，如图1-20所示。

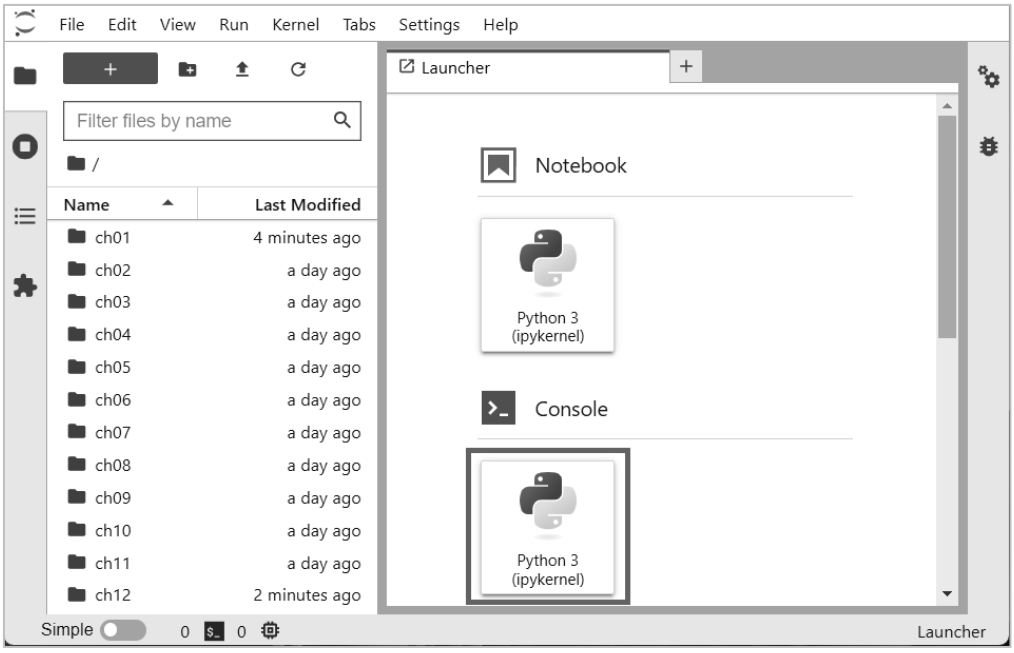



图 1-20　打开 Console

然后，在下方的代码输入区域输入相应的代码即可，如图1-21所示。也可以使用pip安装、更新和卸载第三方包。

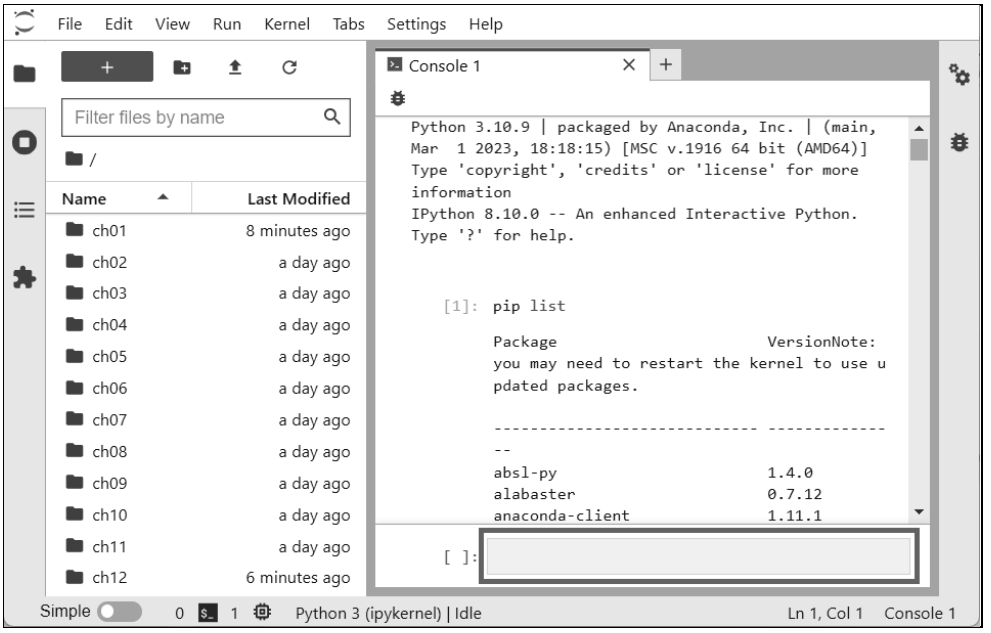



图 1-21　代码输入区域

# 1.5 本章小结

Python是深受广大数据分析师喜爱的数据分析工具，本章通过结合实际操作和范例介绍了Python编程的相关知识，包括软件集成环境Anaconda的安装、常用代码开发工具、Python程序的构成和特点等，对于从未接触过Python编程的初学者来说，如果想用Python进行数据可视化，掌握编程技能可以说是最基本的要求。

本章只是一个开始，下一章将带领读者进入Python编程的精彩世界。


# 第 2 章

# Python编程基础

本章详细介绍Python程序的数据类型、运算符和优先级、基础语法、常用函数等，这些都是学习Python编程的必备基础，对于从未接触过编程的初学者来说，本章内容十分重要。

## 2.1 Python数据类型

Python是一种动态类型的编程语言，因此它支持多种数据类型。Python中的基本数据类型包括数字、字符串、布尔、列表、元组、集合和字典等。本节开始介绍这些数据类型的特点与使用。

### 2.1.1 数字

Python中的数字（Number）类型用于存储数值，主要有整型（int）和浮点型（float）两种。

**例1**：我们知道客户是门店最重要的资产，为了更好地了解客户和为客户服务，门店经理让分析师小王统计汇总每日购买商品的客户数，其中今天的客户数为99人，输入代码如下：

```
cust_num1 = 99
```

此时，cust_num1变量的值是99，此为整型数据。

**例2**：门店经理需要统计本月销售数据，经实际计算为2984.5万元，将此数值赋予一个变量，即给门店带来直接利润的客户。会有部分客户进行退货，根据营业流水数据统计，共计8人，因此需要剔除，输入有效客户数的代码如下：

```
cust_num2 = 2984.5
```

此时，变量cust_num2的值即为浮点型。

### 2.1.2 字符串

字符串（String）是Python中常用的数据类型之一。我们可以使用英文输入法下的单引号（''）或双引号（""）来创建字符串，字符串可以是英文、中文或中文和英文的混合形式。例如，输入以下代码：

```
str1 = "Now or never!"
str2 = "学习 Python!"
```

此时，创建了两个字符串str1和str2，执行上述代码后两个字符串如下：

```
str1
'Now or never!'
str2
'学习 Python!'
```

在Python中，可以通过“+”实现字符串与其他字符串的拼接，例如输入以下代码：

```
str3 = str1 + " Work hard to learn Python!"
```

此时str3字符串如下：

```
'Now or never! Work hard to learn Python!'
```

在字符串中，我们可以通过索引获取字符串中的字符，遵循“左闭右开”的原则，注意索引是从0开始的。例如，截取str1的前3个字符，代码如下：

```
str1[:3]
#或者 str1[0:3]
```

输出结果如下：

```
'Now'
```

可以看出，程序输出str1中的前3个字符“Now”，索引分别对应0、1、2。原字符串中的每个字符对应的索引号如表2-1所示。

表 2-1 字符串索引

| 原字符串 | N | o | w | | o | r | | n | e | v | e | r | ! |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 正向索引 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 反向索引 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

此外，还可以使用反向索引实现上述需求，但是索引位置会发生变化，分别对应−13、−12、−11，代码如下：

```
str1[-13:-11]
```

输出结果如下：

```
'Now'
```

同理，我们也可以截取原字符串中的“never”子字符串，索引的位置是7～12，注意包含7，但不包含12，截取字符串的代码如下：

```
str1[7:12]
```

输出结果如下：

```
'never'
```

Python提供了方便灵活的字符串运算，表2-2列出了可以用于字符串运算的运算符。

表 2-2　字符串运算符

| 序　　号 | 操　作　符 | 说　　明 |
| --- | --- | --- |
| 1 | + | 字符串连接 |
| 2 | * | 重复输出字符串 |
| 3 | [] | 通过索引获取字符串中的字符 |
| 4 | [ : ] | 截取字符串中的一部分，遵循“左闭右开”的原则 |
| 5 | in | 成员运算符，如果字符串中包含给定的字符，就返回True |
| 6 | not in | 成员运算符，如果字符串中不包含给定的字符，就返回True |
| 7 | r/R | 原始字符串，所有的字符串都按照字面的意思来输出 |
| 8 | % | 格式字符串 |

下面以成员运算符in为例介绍字符串运算符，例如，我们需要判断“never”是否在字符串str1中，代码如下：

```
'never' in str1
```

输出结果如下：

```
True
```

这里显示的是True，如果不存在结果，就输出False。

### 2.1.3　列表

列表（List）是Python中常用的数据类型之一，使用方括号（[]）表示。列表中的元素用逗号分隔，并且不需要所有元素具有相同的类型。

例如，创建3个门店有效客户的列表，其中list1为客户张三的个人信息，list2为一周有效客户的数量，list3为客户信息表的主要字段，代码如下：

```
list1 = ['张三', '男', 18966666666, '东方路 666 号']
list2 = [91, 78, 83, 85, 82, 129, 116]
list3 = ["客户姓名", "客户性别", "联系电话", "联系地址"]
```

运行上述代码，创建的列表输出如下：

```
list1
['张三', '男', 18966666666, '东方路 666 号']
list2
[91, 78, 83, 85, 82, 129, 116]
list3
['客户姓名', '客户性别', '联系电话', '联系地址']
```

列表的索引与字符串的索引一样，也是从0开始的，也可以进行截取、组合等操作。例如，我们截取list3中索引从1到3但不包含索引为3的字符串，代码如下：

```
list3[1:3]
```

输出结果如下：

```
['客户性别', '联系电话']
```

可以对列表的数据项进行修改或更新，例如将列表list1中索引为1的元素“男”改为“女”。首先查看索引为1的元素，代码如下：

```
list1[1]
```

输出结果如下：

```
男
```

再修改列表的元素，代码如下：

```
list1[1] = '女'
list1
```

输出结果如下：

```
['张三', '女', 18966666666, '东方路 666 号']
```

可以使用del语句来删除列表中的元素，代码如下：

```
del list1[1]
list1
```

输出结果如下：

```
['张三', 18966666666, '东方路 666 号']
```

也可以使用append()方法在列表尾部添加列表项，代码如下：

```
list1.append(29)
list1
```

输出结果如下：

```
['张三', 18966666666, '东方路 666 号', 29]
```

此外，还可以使用insert()方法在中间指定的位置之前添加列表项，代码如下：

```
list1.insert(1,'男')
list1
```

输出结果如下：

```
['张三', '男', 18966666666, '东方路 666 号', 29]
```

### 2.1.4　元组

Python的元组（Tuple）与列表类似，不同之处在于元组的元素不能被修改。注意元组使用小括号表示，而列表使用方括号表示。元组的创建很简单，只需要在括号中添加元素，并使用逗号隔开即可。

例如，创建3个门店有效客户的元组，其中tup1为客户张三的个人信息，tup2为一周有效客户的数量，tup3为客户信息表的主要字段，代码如下：

```
tup1 = ('张三', '男', 18966666666, '东方路 666 号')
tup2 = (91, 78, 83, 85, 82, 129, 116)
tup3 = ("客户姓名", "客户性别", "联系电话", "联系地址")
```

运行上述代码，创建的元组输出如下：

```
tup1
('张三', '男', 18966666666, '东方路 666 号')

tup2
(91, 78, 83, 85, 82, 129, 116)

tup3
("客户姓名", "客户性别", "联系电话", "联系地址")
```

如果元组中只包含一个元素，那么需要在元素后面添加逗号，否则括号会被当作运算符使用，示例如下：

```
tup4 = (29,)
tup4
(29,)

tup5 = (29)
tup5
29
```

元组的索引与字符串的索引一样，也是从0开始，也可以进行截取、组合等操作。例如，我们截取tup3中索引从1到3但不包含索引为3的元素，代码如下：

```
tup3[1:3]
```

输出结果如下：

```
('客户性别', '联系电话')
```

在Python中，也可以通过“+”实现对元组的连接，运算后会生成一个新的元组，代码如下：

```
tup6 = tup1 + tup4
tup6
```

输出结果如下：

```
('张三', '男', 18966666666, '东方路 666 号', 29)
```

注意，元组中的元素是不允许修改和删除的，例如修改元组tup6中第3个元素的值，代码如下：

```
tup6[2] = 18988888888
```

会输出如下错误信息：

```
---------------------------------------------------------------------------
TypeError                 Traceback (most recent call last)
Cell In[45], line 1
----> 1 tup6[2] = 18988888888
TypeError: 'tuple' object does not support item assignment
```

### 2.1.5 集合

集合（Set）是一个无序的不重复元素序列，可以使用大括号或者set()函数创建，注意创建一个空集合必须使用set()，因为{}是用来创建一个空字典的。创建集合的格式如下：

```
parame = {value01, value02, ...}
```

或者

```
set(value)
```

下面以客户购买商品为例介绍集合的去重功能，假设某客户周一分早晚两次在门店购买了6件商品，分别是中性笔、荧光笔、订书机、中性笔、胶水、计算器，这里有重复的商品中性笔，我们可以借助集合删除该重复值，代码如下：

```
order_mon = {'中性笔','荧光笔','订书机','中性笔','胶水','计算器'}
order_mon
```

输出结果如下：

```
{'中性笔', '胶水', '荧光笔', '计算器', '订书机'}
```

运行上述代码，可以看出已经删除了重复值，只保留了5种不同类型的商品名称。

集合的元素是无序的，因此打印输出的时候也是无序的。

例如，上述客户周三在门店购买了4件商品，分别是订书机、胶水、笔筒、会议牌，代码如下：

```
order_wed = {'订书机','胶水','笔筒','会议牌'}
order_wed
```

输出结果如下：

```
{'会议牌','胶水', '笔筒', '订书机'}
```

我们还可以快速判断某个元素是否在某集合中，例如，判断该客户周一是否购买了“计算器”，代码如下：

```
'计算器' in order_mon
```

输出结果如下：

```
True
```

此外，Python中的集合与数学中的集合概念基本类似，也有交集、并集、差集和补集。

### 1. 集合的交集

例如，统计上述客户周一和周三都购买的商品，代码如下：

```
order_mon & order_wed
```

输出结果如下：

```
{'胶水', '订书机'}
```

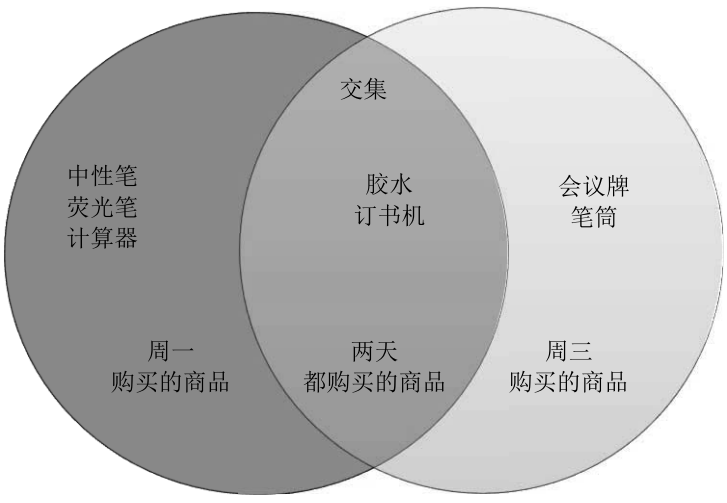



图 2-1　集合间的关系

2. 集合的并集

例如，统计该客户周一和周三购买的商品，代码如下：

```
order_mon | order_wed
```

输出结果如下：

```
{'中性笔', '会议牌', '笔筒', '胶水', '荧光笔', '计算器', '订书机'}
```

3. 集合的差集

例如，统计该客户在周一和周三不同时购买的商品，代码如下：

```
order_mon ^ order_wed
```

输出结果如下：

```
{'中性笔', '会议牌', '笔筒', '荧光笔', '计算器'}
```

4. 集合的补集

例如，统计该客户周一购买，而周三没有购买的商品，代码如下：

```
order_mon - order_wed
```

输出结果如下：

```
{'中性笔', '荧光笔', '计算器'}
```

### 2.1.6 字典

与列表、元组、集合类似，字典（Dictionary）也是一种可变容器，且可存储任意类型的对象。字典的每个键–值对用冒号分隔，每个键–值对之间用逗号分隔，整个字典包括在花括号中，格式如下：

```
dict = {key1:value1, key2:value2}
```

注意，键–值对中的键必须是唯一的，但是值可以不唯一，且数值可以取任意数据类型，但键必须是不可变的，例如创建3个字符串或数字字典，代码如下：

```
dict1 = {'客户数量': 91}
dict2 = {'客户数量': 91, 2023:6}
dict3 = {'客户姓名':'张三','客户性别':'男','联系电话':18966666666,'联系地址':'东方路666号'}
```

运行上述代码，创建的字典输出如下：

```
dict1
{'客户数量': 91}

dict2
```

```
{'客户数量': 91, 2023: 6}

dict3
{'客户姓名': '张三', '客户性别': '男', '联系电话': 18966666666, '联系地址': '东方路 666 号'}
```

在Python中，访问字典中的值需要把相应的键放入方括号中，例如获取客户的联系地址，代码如下：

```
dict3['联系地址']
```

输出结果如下：

```
'东方路 666 号'
```

在Python中，当访问字典中的值时，如果字典中没有该键，那么程序会报错，代码如下：

```
dict3['客户年龄']
```

会输出如下的错误信息：

```
---------------------------------------------------------------------------
KeyError                  Traceback (most recent call last)
Cell In[18], line 1
----> 1 dict3['客户年龄']
KeyError: '客户年龄'
```

在Python中，还可以向已有字典中添加新内容，方法是增加新的键–值对，也可以修改字典已有键–值对中的值，例如，向字典dict2中添加键“客户年龄”，并修改键“2023”中的值为8，代码如下：

```
dict2['客户年龄'] = 29
dict2[2023] = 8
dict2
```

输出结果如下：

```
{'客户数量': 91, 2023: 8, '客户年龄': 29}
```

在Python中，能够删除字典中的单一元素，也能清空和删除字典，例如要删除字典dict2中的键“2023”，然后清空字典，最后删除字典。输入代码如下：

```
del dict2[2023]
dict2
```

输出结果如下：

```
{'客户数量': 91, '客户年龄': 29}
```

可以使用clear()方法清空字典，代码如下：

```
dict2.clear()
dict2
```

输出结果如下：

```
{}
```

使用del()方法删除字典后，再调用该字典，将会显示字典没有被定义的报错信息，代码如下：

```
del dict2
dict2
```

会输出如下错误信息：

```
---------------------------------------------------------------------------
NameError                 Traceback (most recent call last)
Cell In[28], line 1
----> 1 dict2
NameError: name 'dict2' is not defined
```

# 2.2 Python运算符和优先级

与其他程序语言类似，Python编程中也会涉及运算问题，这样就会用到运算符。本节介绍Python常见的运算符，希望读者掌握它们的含义及用法。

## 2.2.1 Python 运算符

Python运算符用于执行各种算术、逻辑和其他类型的计算，以下是Python编程中常用的运算符。

### 1. 算术运算符

常用的算术运算符有+（加）、−（减）、*（乘）、/（除）、%（取模）、**（幂），//（整数除法）。下面就%、**、//运算符举例说明。

- %运算符：%运算符用来计算两个数相除后的余数，例如 x % y 表示 x 除以 y 后的余数。下面是几个示例：

```
7 % 3 = 1
10 % 5 = 0
12 % 7 = 5
```

- **运算符：**运算符是用来计算一个数的指数次幂，例如 x ** b 表示 x 的 y 次幂。下面是几个示例：

```
3 ** 3 = 27
7 ** 2 = 49
```

```
4 ** 3 = 64
```

- //运算符：//运算符用来计算两个数相除后的整数部分，例如 x //y 表示 x 除以 y 后的整数部分。下面是几个示例：

```
7 // 3 = 2
20 // 5 = 4
11 //87 = 1
```

需要注意的是，整数除法得到的结果可能会向下取整，因为它只计算商的整数部分，小数部分会被舍去。

### 2. 比较运算符

常用的比较运算符有>（大于）、<（小于）、==（等于）、!=（不等于）、>=（大于或等于）、<=（小于或等于）。

### 3. 逻辑运算符

常用的逻辑运算符有and（与）、or（或）、not（非）。

- and 运算符：当两个表达式都是 True 时，它返回 True；否则返回 False。下面是几个示例：

```
True and True   # True
True and False  # False
False and False # False
```

- or 运算符：当两个表达式至少有一个是 True 时，它返回 True；否则返回 False。下面是几个示例：

```
True or True    # True
True or False   # True
False or False  # False
```

- not 运算符：将表达式的值进行取反，如果原始值为 True，则返回 False；如果原始值为 False，则返回 True。下面是几个示例：

```
not True   # False
not False  # True
```

这些逻辑运算符在编程中非常有用，可以用于控制语句中的条件判断，比如 if 语句中的条件判断。

### 4. 位运算符

常用的位运算符有&（按位与）、|（按位或）、^（按位异或）、~（按位取反）、<<（左移位）、>>（右移位）。

位运算符是一种在二进制数位上进行操作的运算符。下面是位运算符的解释。

- &运算符：两个位都为 1 时，结果才为 1，否则为 0。

```
x = 5                # 二进制数为 101
y = 3                # 二进制数为 011
  print(x & y)       # 输出：1 （二进制为 001）
```

- |运算符：其中一位为 1 时，结果就为 1，否则为 0。

```
x = 5                # 二进制数为 101
y = 3                # 二进制数为 011
print(x |y)          # 输出：7 （二进制为 111）
```

- ^运算符：如果两个数相应位的值不同，则该位的结果为 1，否则为 0。

```
x = 5                # 二进制数为 101
y = 3                # 二进制数为 011
print(x ^ y)         # 输出：6 （二进制为 110）
```

- ~运算符：结果将二进制数的所有位取反，即 0 变 1，1 变 0。

```
x = 5                # 二进制数为 101
print(~x)            # 输出：-6
```

因为Python的二进制表示使用了补码，所以~x对应的二进制数为010（补码表示），即十进制数为2，然后将其取反为101，再转换成补码表示，即−6。

- <<运算符：操作数的各二进位全部左移若干位，相当于此数乘以 2 的移动次方。

```
x = 5                # 二进制数为 101
print(x << 2)        # 输出：20 （二进制数为 10100）
```

将二进制数101左移两位得到二进制数10100，即十进制数20。

- >>运算符：操作数的各二进位全部右移若干位，相当于此数除以 2 的移动次方。

```
x = 20               # 二进制数为 10100
print(x >> 2)        # 输出：5（二进制数为 101）
```

将二进制数10100右移两位得到二进制数101，即十进制数5。

### 5. 赋值运算符

赋值运算符用于给变量赋值。它们实现了将右侧操作数应用于左侧操作数并将结果分配给左侧操作数的功能。常用的位运算符有=（赋值）、+=（加等于）、−=（减等于）、*=（乘等于）、/=（除等于）、%=（取模等于）、**=（幂等于）、//=（取整除等于）、&=（按位与等于）、|=（按位或等于）、^=（按位异或等于）、>>=（右移等于）、<<=（左移等于）。

- =运算符：将右侧操作数的值赋给左侧操作数。
- +=运算符：等同于 x = x + y。
- −=运算符：等同于 x = x − y。
- *=运算符：等同于 x = x * y。
- /=运算符：等同于 x = x / y。

- %=运算符：等同于 x = x % y。
- **=运算符：等同于 x = x ** y。
- //=运算符：等同于 x = x // y。
- &=运算符：等同于 x = x & y。
- |=运算符：等同于 x = x | y。
- ^=运算符：等同于 x = x ^ y。
- >>=运算符：等同于 x = x >> y。
- <<=运算符：等同于 x = x << y。

### 6. 条件运算符

条件运算符用于根据一个或多个条件对程序执行不同的代码块。Python中常见的条件运算符有if-else和if-elif-else。

1）if-else

该语句根据给定条件来执行代码块。如果条件为True，则执行if子句，并跳过else子句；如果条件为False，则执行else子句。示例如下：

```
x = 10
if x > 5:
    print("x 大于 5")
else:
    print("x 小于或等于 5")
```

以上示例输出x大于5。

2）if-elif-else

该语句包含多个条件，并在每个条件为False（所有条件为False）时提供一个备选行动方案。示例如下：

```
x = 10
if x == 5:
    print("x 等于 5")
elif x >5 and x <= 10:
    print("x 大于 5，小于或等于 10")
else:
    print("x 大于 10")
```

在上面的示例代码中，如果x等于5，那么将会输出“x等于5”。如果x大于5且小于或等于10，那么将输出“x大于5，小于或等于10”。如果x大于10，那么将输出“x大于10”。同时需要注意的是，if-elif-else语句是可以嵌套的。

### 7. 成员运算符

成员运算符in和not in用于检查一个值是否存在于某个序列中，即检查值是不是该序列的元素。

1）in 运算符

若指定的值在序列中找到，则返回True，否则返回False。示例如下：

```
fruits = ["apple", "banana", "cherry"]

if "apple" in fruits:
    print("apple 在水果列表中")
else:
    print("apple 不在水果列表中")
```

2）not in 运算符

若指定的值不存在于序列中，则返回True，否则返回False。示例如下：

```
fruits = ["apple", "banana", "cherry"]

if "orange" not in fruits:
    print("orange 不在水果列表中")
else:
    print("orange 在水果列表中")
```

在上面的示例中，in运算符用于检查列表fruits中是否包含字符串"apple"，结果为True，因此输出"apple在水果列表中"。而not in运算符则用于检查字符串"orange"是否不存在于列表fruits中，结果为True，所以输出"orange不在水果列表中"。

**8. 身份运算符**

身份运算符is和is not用于比较两个对象的内存地址，即判断两个对象是不是同一个对象。

1）is 运算符

若两个对象的内存地址相同，则返回True，否则返回False。示例如下：

```
x = ["apple", "banana", "cherry"]
y = ["apple", "banana", "cherry"]
z = x

if x is z:
    print("x 和 z 是同一个对象")
else:
    print("x 和 z 不是同一个对象")

if x is y:
    print("x 和 y 是同一个对象")
else:
    print("x 和 y 不是同一个对象")
```

在上面的示例中，我们首先定义了两个列表x和y，它们包含相同的元素。然后我们将x赋值给变量z。因为z和x引用的是同一个列表对象，所以x is z的结果为True。但是x和y引用的是不同的列表对象，所以x is y的结果为False。

2）is not 运算符

若两个对象的内存地址不同，则返回True，否则返回False。示例如下：

```
x = ["apple", "banana", "cherry"]
y = ["apple", "banana", "cherry"]
z = x

if x is not z:
   print("x 和 z 不是同一个对象")
else:
   print("x 和 z 是同一个对象")

if x is not y:
   print("x 和 y 不是同一个对象")
else:
   print("x 和 y 是同一个对象")
```

在上面的示例中，is not和is的作用是相反的。因为z和x引用的是同一个列表对象，所以x is not z的结果为False。但是x和y引用的是不同的列表对象，所以x is not y 的结果为True。

### 2.2.2　运算符的优先级

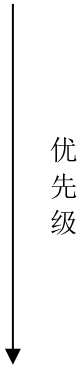
Python运算符是一些用于执行各种算术和逻辑操作的特殊符号。通常，Python中的运算符及其优先级有以下特点：

- 一元操作符优先级最高，例如正号和负号。
- 先乘除，后加减，例如乘号和除号的优先级高于加号和减号。
- 同级运算符从左到右计算。

当然，代码中可以使用圆括号“( )”来调整计算的优先级，在具体计算时可以改变运算符的优先级。

Python中的运算符优先级从高到低依次为：

（1）圆括号：()。
（2）幂运算：**。
（3）一元运算符：~、+、−。
（4）乘、除、模、整除：*、/、%、//。
（5）加、减：+、−。
（6）移位运算：>>、 <<。
（7）位运算符：&、|、^。
（8）比较运算符：>、>=、<、<=、==、!、=。
（9）逻辑运算符：not、and、or。



以下举例说明。

（1）加号和减号：

```
x = 5
y = 3

print(x + y)                # 输出：8
print(x - y)                # 输出：2
```

（2）乘号和除号：

```
x = 6
y = 3

print(x * y)                # 输出：18
print(x / y)                # 输出：2.0
```

（3）取模运算：

```
x = 7
y = 3

print(x % y)                # 输出：1
```

（4）乘方运算：

```
x = 2
y = 3

print(x ** y)               # 输出：8
```

（5）比较运算符：

```
x = 5
y = 3

print(x > y)                # 输出：True
print(x < y)                # 输出：False
print(x == y)               # 输出：False
```

（6）逻辑运算符：

```
x = 5
y = 3

print(x > 3 and y < 6)      # 输出：True
print(x > 3 or y > 6)       # 输出：True
print(not(x > 3))           # 输出：False
```

# 2.3 Python语法基础

前面我们介绍的程序代码都是按顺序执行的，也就是先执行第1条语句，然后是第2条语句、第3条语句，以此类推，一直到最后一条语句，这称为顺序结构。但是很多情况下，仅有顺序结构的代码是远远不够的，Python程序还会涉及条件判断和循环执行等较为复杂的情况，这些也是Python程序经常遇到的语法结构。本节介绍条件语句、循环语句和格式化语句及其使用。

## 2.3.1 条件语句：if 及 if 嵌套

在Python中，可以使用if else语句对条件进行判断，然后根据不同的结果执行不同的代码，此结构称为选择结构或者分支结构。

比如一个程序限制了只能成年人使用，儿童没有权限使用。这时程序就需要做出判断，看用户是不是成年人（年龄满18周岁），并给出提示。

Python中的if else语句可以细分为3种形式，分别是if语句、if else语句和if嵌套语句，它们的执行流程如图2-2～图2-4所示。

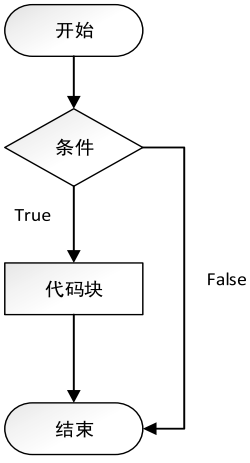



图 2-2　if 语句的流程图

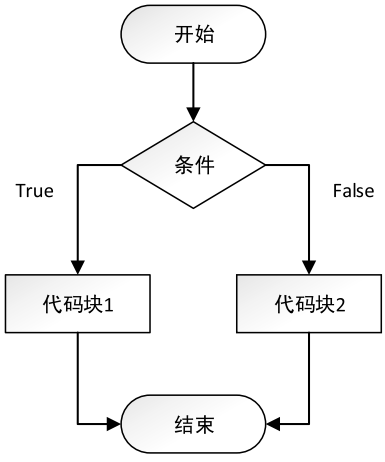



图 2-3　if else 语句的流程图

例如，在门店销售业绩考核中，通常会对业绩分等级，这种情况就可以使用if嵌套语句实现，代码如下：

```
order_sale = 84

if order_sale < 60:
    print("业绩不达标")
else:
    if order_sale <= 75:
        print("业绩一般")
```

```
    else:
        if order_sale <= 85:
            print("业绩良好")
        else:
            print("业绩优秀")
```

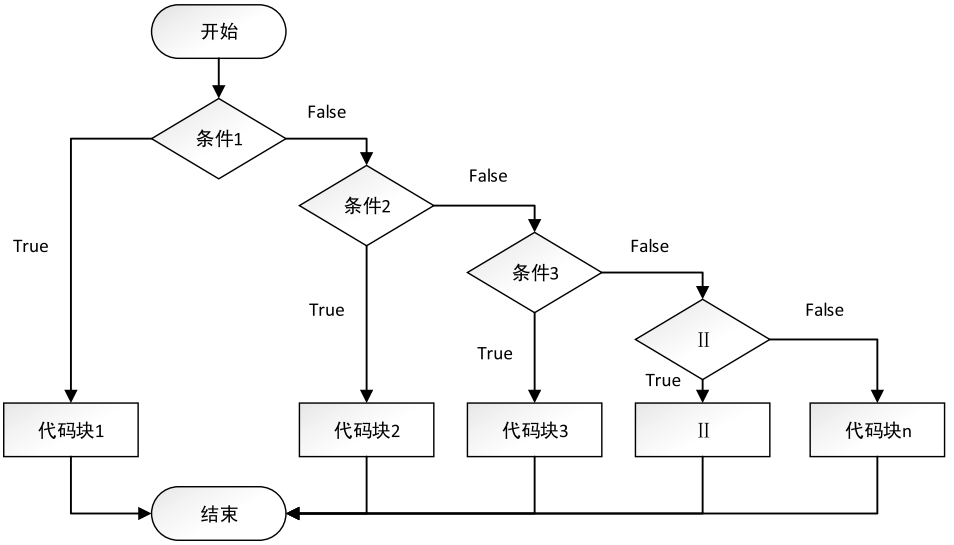



图 2-4　if 嵌套语句的流程图

输出结果如下：

```
业绩良好
```

当然，这个需求还有很多实现方法，这里就不再逐一列出了。

### 2.3.2　循环语句：while 与 for

在Python中，while循环和if条件分支语句类似，即在条件（表达式）为真的情况下，会执行相应的代码块。不同之处在于，只要条件为真，while就会一直重复执行代码块。

while语句的语法格式如下：

```
while 条件表达式：
    代码块
```

这里的代码块指的是缩进格式相同的多行代码，不过在循环结构中，它又称为循环体。while语句执行的具体流程为：首先判断条件表达式的值，其值为真（True）时，则执行代码块中的语句，当执行完毕后，再回过头来重新判断条件表达式的值是否为真，若仍为真，则继续重新执行代码块，如此循环，直到条件表达式的值为假（False），才终止循环。while循环结构的流程图如图2-5所示。

在Python中，for循环是使用比较频繁的，常用于遍历字符串、列表、元组、字典、集合等序列类型，用于逐个获取序列中的元素。

for循环的语法格式如下：

```
for 迭代变量 in 变量:
    代码块
```

其中，迭代变量用于存放从序列类型变量中读取出来的元素，所以一般不会在循环中对迭代变量手动赋值，“代码块”指的是具有相同缩进格式的多行代码（和while一样），由于和循环结构联用，因此又称为循环体。for循环结构的流程图如图2-6所示。

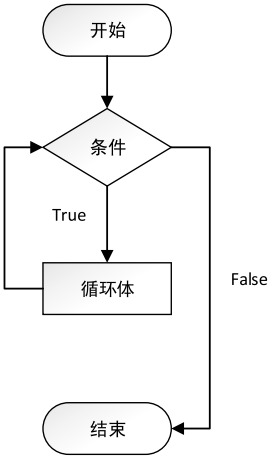



图 2-5　while 语句的流程图

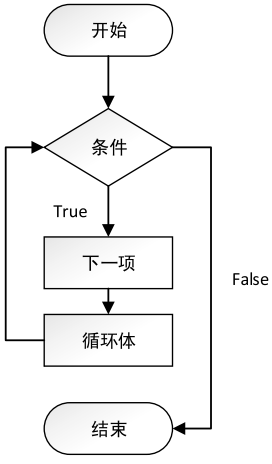



图 2-6　for 语句的流程图

下面介绍如何使用while循环输出九九乘法表，代码如下所示。

```
i = 1
while i<=9:
    j = 1
    while j <= i:
        print('%d*%d=%2d\t'%(i,j,i*j),end='')
        j+=1
    print()
    i +=1
```

运行上述代码，输出结果如下：

```
1*1= 1
2*1= 2  2*2= 4
3*1= 3  3*2= 6  3*3= 9
4*1= 4  4*2= 8  4*3=12  4*4=16
5*1= 5  5*2=10  5*3=15  5*4=20  5*5=25
6*1= 6  6*2=12  6*3=18  6*4=24  6*5=30  6*6=36
7*1= 7  7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49
8*1= 8  8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64
9*1= 9  9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

也可以使用for循环输出九九乘法表，代码如下：

```
for i in range(1, 10):
    for j in range(1, i + 1):
        print(j, '*', i, '=', i * j, end="\t")
    print()
```

当然九九乘法表还有很多其他实现方法，这里就不再详细阐述了。

### 2.3.3 格式化：format 与%

在Python中，可以使用format函数和%对字符串进行格式化，以输出特定格式的字符串。下面重点讲解format()函数及其使用方法。

#### 1．利用 f+strings 进行格式化

在Python 3.6中加入了一个新特性：f+strings，可以直接在字符串的前面加上f来格式化字符串，例如输出“2023年5月南京路店的销售额是965.08万元。”的代码如下：

```
store = '南京路店'
sales = 965.08
s = f'2023 年 5 月{store}的销售额是{sales}万元。'
print(s)
```

输出结果如下：

```
2023 年 5 月南京路店的销售额是 965.08 万元。
```

#### 2．利用位置进行格式化

可以通过索引来直接使用*号将列表打散，通过索引来取值，例如输出“2023年5月南京路店的销售额是965.08万元，利润额是39.01万元。”的代码如下：

```
sales = ['南京路店',965.08,39.01]
s = '2023 年 5 月{0}的销售额是{1}万元，利润额是{2}万元。'.format(*sales)
print(s)
```

输出结果如下：

```
2023 年 5 月南京路店的销售额是 965.08 万元，利润额是 39.01 万元。
```

#### 3．利用关键字进行格式化

也可以通过**号将字典打散，通过键（key）来取值，例如输出“2023年5月南京路店的销售额是965.08万元，利润额是39.01万元。”的代码如下：

```
    d = {'store':'南京路店','sales':965.08,'profit':39.01}
    s = '2023 年 5 月{store}的销售额是{sales}万元，利润额是{profit}万元。'.format(**d)
print(s)
```

输出结果如下：

```
2023 年 5 月南京路店的销售额是 965.08 万元，利润额是 39.01 万元。
```

### 4．利用下标进行格式化

还可以利用下标 + 索引的方法进行格式化，例如输出“2023年5月南京路店的销售额是965.08万元，利润额是39.01万元。”的代码如下：

```
sales = ['南京路店',965.08,39.01]
s = '2023年5月{0[0]}的销售额是{0[1]}万元,利润额是{0[2]}万元。'.format(sales) print(s)
```

代码输出结果如下：

```
2023 年 5 月份南京路店的销售额是 965.08 万元,利润额是 39.01 万元。
```

### 5．利用精度与类型进行格式化

精度与类型可以一起使用，格式为{ :.nf} .format(数字)，其中.n表示保留n位小数，对于整数直接保留固定位数的小数位，例如输出3.14和26.00的代码如下：

```
pi = 3.14159265359
print('{:.2f}'.format(pi))

age = 26
print('{:.2f}'.format(age))
```

输出结果如下：

```
3.14
26.00
```

### 6．利用千分位分隔符格式化

"{:,}".format()函数中的冒号加逗号可以将一个数字每三位用逗号进行分隔，例如输出截至2021年底的全世界人口数据7,888,408,686的代码如下：

```
print("{:,}".format(7888408686))
```

输出结果如下：

```
7,888,408,686
```

## 2.4 Python的函数

函数是一段代码块，通常是为了执行特定的任务或计算特定的值。在编程过程中调用函数可以避免编写重复代码，提高编程效率，因此掌握函数的概念和用法很重要。

### 2.4.1 函数的概念及使用

与其他编程语言类似，Python中包括数学函数、随机数函数、三角函数等。表2-3列举了一些常用的数学函数。

表 2-3 常用的数学函数

| 序　　号 | 函 数 名 | 说　　明 |
|---|---|---|
| 1 | ceil(x) | 返回数字的上入整数，如math.ceil(4.1)返回5 |
| 2 | exp(x) | 返回e的x次幂(ex)，如math.exp(1)返回2.718281828459045 |
| 3 | fabs(x) | 返回数字的绝对值，如math.fabs(−10)返回10.0 |
| 4 | floor(x) | 返回数字的下舍整数，如math.floor(4.9)返回4 |
| 5 | log(x) | 如math.log(math.e)返回1.0，math.log(100,10)返回2.0 |
| 6 | log10(x) | 返回以10为基数的x的对数，如math.log10(100)返回 2.0 |
| 7 | modf(x) | 返回x的整数部分与小数部分，数值符号与x相同 |
| 8 | pow(x, y) | 返回x**y运算后的值 |
| 9 | sqrt(x) | 返回数字x的平方根 |

例如，我们要返回数值圆周率近似值3.1415926的整数部分和小数部分。在Python中，数学运算常用的函数在math模块中，因此首先需要导入math模块，然后使用其中的modf()函数提取整数部分和小数部分。

程序代码如下：

```
import math
math.modf(3.1415926)
```

输出结果如下：

```
(0.14159260000000007, 3.0)
```

可以看出，3.1415926的整数部分是3.0，小数部分是0.14159260000000007，注意这里不是0.1415926，这是因为Python默认的是数值计算，而不是符号计算，其中数值计算是近似计算，而符号计算则是绝对精确的计算。

我们可以通过函数来传递参数，也可以利用函数返回结果。

下面是一个简单的Python函数的例子：

```
def add_numbers(x, y):
    result = x + y
    return result
```

这个函数的名称是add_numbers，有两个参数x和y。它计算输入参数的和，并将结果赋给一个叫result的变量。最后，它通过使用return语句返回结果。现在，我们可以使用这个函数来计算任意两个数字的和：

```
sum = add_numbers(3, 5)
print(sum)
```

这将输出8，因为add_numbers函数将3和5相加并返回结果8。

在编程中可以随时调用函数实现想要的结果，比如，以下例子调用了内置函数len()：

```
string = "Hello World"
length = len(string)
print(length)
```

这个函数接收一个参数，返回给定序列或字符串的长度，从而输出11，因为字符串的长度为11。

### 2.4.2　数据分析中的常用函数

在Python中，函数可以根据其功能分为多种类型，最常见的有两种，一种是内置函数，是Python已经定义好并集成到解释器中的函数，例如print()、len()、type()等；另一种是用户定义函数，是由用户在程序中定义和实现的函数。本小节主要介绍几个在数据分析中常用的内置函数。有关自定义函数，读者可以参考相关资料，本书不做介绍。

#### 1. map()函数：数组迭代

map()是一个内置函数，它接收两个参数，一个是函数，另一个是迭代器（Iterable）。map()函数将传入的函数依次作用到序列的每一个元素上，并把结果作为新的迭代器，返回结果。

例如，求一个数值型列表中各个数值的立方，返回的还是列表，就可以使用map()函数实现，代码如下：

```
def f(x):
    return x**3
r = map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])
list(r)
```

输出结果如下：

```
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

map()函数传入的第一个参数是f，即函数对象本身。由于结果r是一个迭代器，迭代器是惰性序列，即表达式和变量绑定，你不主动遍历它，就不会计算其中元素的值。因此，通过list()函数可以让它把整个序列都计算出来并返回一个列表。

其实，这里可以不使用map()函数，使用循环也可以实现，代码如下：

```
def f(x):
    return x**3
S = []
for i in [1, 2, 3, 4, 5, 6, 7, 8, 9]:
   S.append(f(i))
print(S)
```

输出结果如下：

```
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

map()函数把运算规则抽象化，因此，我们不但可以计算简单的f(x)=x**3，还可以计算任意复杂的函数，例如把列表中所有的数字转为字符串，代码如下：

```
list(map(str,[1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

输出结果如下：

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

从输出结果可以看出，列表中所有的数字都转为字符串了。

### 2. reduce()函数：序列累积

reduce()函数接收的参数有3个：函数f、列表list和可选的初始值，初始值的默认值是0。reduce()函数传入的函数f必须接收两个参数，对列表list的每个元素反复调用函数f，并返回最终计算结果。

例如，计算列表[1, 2, 3, 4, 5]中所有数值的和，初始值是100，代码如下：

```
from functools import reduce
list_a = [1,2,3,4,5]
def fn(x, y):
    return x + y
total = reduce(fn,list_a,100)
print(total)
```

输出结果如下：

```
115
```

此外，还可以使用lambda函数进一步简化程序，代码如下：

```
from functools import reduce
list_a = [1,2,3,4,5]
total = reduce(lambda x,y:x+y ,list_a,100)
print(total)
```

输出结果如下：

```
115
```

### 3. filter()函数：数值过滤

filter()函数用于过滤数据序列，与map()函数类似，filter()函数也需要接收一个函数和一个序列。与map()函数不同的是，filter()函数把传入的函数依次作用于每一个元素，然后根据返回值是True还是False决定保留还是丢弃该元素。

例如，利用filter()函数过滤出1～100中平方根是整数的数，代码如下：

```
import math
def is_sqr(x):
    return math.sqrt(x) % 1 == 0
print(list(filter(is_sqr, range(1, 101))))
```

输出结果如下：

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

其中，math.sqrt()函数是求平方根的函数。

此外，filter()函数还可以处理缺失值等，例如，将一个序列中的空字符串和None都删除，代码如下所示。

```
def day(s):
    return s and s.strip()
list(filter(day, ['order_mon','','order_wed',None,'order_fri','  ']))
```

输出结果如下：

```
['order_mon', 'order_wed', 'order_fri']
```

从输出结果可以看出，使用filter()函数关键在于正确实现一个筛选函数。


filter()函数返回的是一个迭代器，也就是一个惰性序列，计算结果需要用list()函数获得所有结果并返回列表。

#### 4. sorted()函数：列表排序

排序是程序中经常用到的算法。无论使用冒泡排序还是快速排序，排序的核心是比较两个元素的大小。如果是数字，我们可以直接比较，但如果是字符串或者两个字典，直接比较数学上的大小是没有意义的，因此比较的过程必须通过函数抽象出来。

sorted()函数可以对list进行排序，代码如下：

```
sorted([12, 2, -2, 8, -16])
```

输出结果如下：

```
[-16, -2, 2, 8, 12]
```

此外，sorted()函数也是一个高阶函数，可以接收一个key函数来实现自定义的排序，例如按绝对值大小排序，代码如下：

```
sorted([12, 2, -2, 8, -16],key=abs)
```

输出结果如下：

```
[2, -2, 8, 12, -16]
```

key指定的函数将作用于列表的每一个元素上，并根据key函数返回的结果进行排序。

我们再看一个字符串排序的例子，代码如下：

```
sorted(['Month', 'year', 'Day', 'hour'])
```

输出结果如下：

```
['Day', 'Month', 'hour', 'year']
```

默认情况下，对字符串排序是按照ASCII码的大小排序的，大写字母会排在小写字母的前面。

现在，我们提出排序忽略大小写，按照字母顺序排序。要实现这个算法，不必对现有代码大加改动，只要我们能用一个key函数把字符串映射为忽略大小写排序即可。忽略大小写来比较两个字符串，实际上就是先把字符串都变成大写（或者都变成小写）再比较。

这样，我们给sorted()函数传入key函数，即可实现忽略大小写的排序，代码如下：

```
sorted(['Month', 'year', 'Day', 'hour'],key=str.lower)
```

输出结果如下：

```
['Day', 'hour', 'Month', 'year']
```

要进行反向排序，不必改动key函数，可以传入第3个参数reverse=True，代码如下：

```
sorted(['Month', 'year', 'Day', 'hour'], key=str.lower, reverse=True)
```

输出结果如下：

```
['year', 'Month', 'hour', 'Day']
```

## 2.5 本章小结

学习Python需要从数据类型、基础语法等开始，然后逐渐深入学习Python的高级语法和特性，例如列表、字典、元组、函数等。

本章详细介绍了Python程序的数据类型、运算符和优先级、基础语法、常用函数等基础知识，这是学习Python编程的必备基础，对于初次接触编程的读者来说，从Python入手学习编程是一个不错的选择。掌握了本章内容，使用Python的各种工具进行数据分析和可视化就会变得得心应手。


# 第 3 章

# Pandas数据整理与清洗

在实际项目中，我们需要从不同的数据源中提取数据，这些数据往往并不规范，需要对其进行准确性检查、转换和合并整理，并载入数据库中，才能供应用程序分析和应用，这个过程就是数据整理和清洗，也称为数据准备，数据只有经过清洗、贴标签、注释和准备后，才能成为宝贵的资源。Pandas是为解决数据分析任务而创建的库，它提供了大量能使我们快速处理数据的函数和方法，本章将详细介绍如何使用Python的Pandas数据分析工具进行数据准备，包括数据的读取、索引、切片、排序、聚合、透视、合并等。

## 3.1 Pandas的概念与数据结构

本节我们先来介绍Pandas的基本概念与数据结构，了解Pandas的数据结构也是用Pandas处理数据的基础。

### 3.1.1 初识 Pandas

#### 1. Pandas 简介

Pandas是基于NumPy构建的重要的数据分析Python包，提供快速、灵活、清晰的数据结构和数据分析工具。Pandas是“面板数据”的缩写，表示它是为面板数据设计的。所谓面板数据，是指含有两个以上的索引的二维数据。

Pandas作为Python重要的数据分析工具，配合NumPy、SciPy等模块可以完成大多数数据分析任务。其主要功能和特点如下：

（1）Pandas内置了简洁高效的数据结构，如Series和DataFrame。