高等院校计算机应用系列教材

Python 应用教程：网络数据采集与处理

向光军　主　编

田　林　姚　坤　谢　莹　副主编

清華大學出版社

北　京

内 容 简 介

本书面向高等院校学生及初学者，系统介绍了 Python 在网络数据采集与处理中的应用，内容涵盖数据获取、解析、存储、处理与可视化等关键环节，兼顾理论讲解与实践操作，内容层层递进，贴合教学与项目实战需求。

本书共分 10 章，其中第 1~5 章讲授基础知识，包括 Python 语言基础、网络爬虫基础、HTTP/HTML 基础、requests 与 urllib 网页抓取、正则表达式、XPath 及 BeautifulSoup 等主流解析工具；第 6~8 章深入数据库存储 (MySQL、MongoDB、Redis) 及 Scrapy 框架开发；第 9 和第 10 章则聚焦于数据处理与可视化，介绍 Pandas、jieba、Matplotlib、Seaborn、wordcloud 等常用工具，并配有关键词提取、词云数据可视化、大学排名分析等实战案例。

本书内容完整、案例丰富、语言通俗，注重动手实践与开发规范，并结合反爬虫应对、分布式部署等前沿内容，提升内容的实用性和广度，适合高等院校计算机类专业、新闻传播类专业学生使用，也适合希望掌握数据采集处理能力的编程初学者、数据分析人员及工程开发者作为参考读物。

图书在版编目(CIP)数据

Python应用教程 : 网络数据采集与处理 / 向光军主编.

北京 : 清华大学出版社, 2026. 5. -- (高等院校计算机应用系列教材).

ISBN 978-7-302-70539-0

Ⅰ. TP312.8

中国国家版本馆CIP数据核字第2025L3A958号

责任编辑：刘金喜
封面设计：高娟妮
版式设计：妙思品位
责任校对：成风进
责任印制：刘　菲

出版发行：清华大学出版社

网　　址：https://www.tup.com.cn，https://www.wqxuetang.com

地　　址：北京清华大学学研大厦A座　　邮　编：100084

社 总 机：010-83470000　　邮　购：010-62786544

投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn

质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：三河市天利华印刷装订有限公司

经　　销：全国新华书店

开　　本：185mm×260mm　　**印　张：**17　　**字　数：**372千字

版　　次：2026年6月第1版　　**印　次：**2026年6月第1次印刷

定　　价：68.00元

产品编号：114239-01

前　言

亲爱的读者：

感谢您选择《Python 应用教程：网络数据采集与处理》一书。本书适用于大数据、应用统计学、人工智能、网络安全等相关专业学生学习，旨在帮助您系统学习 Python 网络爬虫技术，并掌握数据存储、数据预处理和数据可视化的基本方法。

随着互联网数据的快速增长，网络爬虫技术已成为数据采集的重要工具。掌握爬虫技术不仅可以帮助您自动化收集大量信息，还能为数据分析、商业决策及人工智能应用提供数据基础。本书的目标是提供一套完整的 Python 网络数据采集与处理指南，内容涵盖从基础入门到高级应用的各个方面。通过循序渐进的讲解和丰富的实践案例，帮助您掌握数据采集、处理与可视化的关键技术。

本书共分为10章，包括 Python 基础知识、网络爬虫、网页解析、数据存储、爬虫框架、数据预处理与可视化技术等内容，使您不仅能掌握网络爬虫的核心概念和应用场景，还能熟练运用 Python 实现数据处理和可视化。

本书从 Python 语言基础讲起，逐步引导您掌握网络爬虫的基本概念和工作原理、HTTP 协议和 HTML 解析等核心知识。随后，书中详细介绍了 requests 库、正则表达式、Xpath、BeautifulSoup 库以及 Scrapy 框架的使用方法，并探讨了数据存储、反爬虫技术及其应对策略。此外，书中还涉及网络爬虫的安全与法律风险问题，让您在学习技术的同时，了解合规开发的重要性。

在数据处理与可视化方面，本书介绍了Pandas、Matplotlib、Seaborn、wordcloud 等数据分析与可视化工具，帮助您实现数据清洗、转换、分析和可视化。通过实例演示如何利用 Python 工具从爬取的数据中提取有价值的信息，并以直观的方式展示数据趋势，使您能够更深入理解数据分析的意义。

本书每一章节均配有详细的代码示例和实现步骤，使您能更直观地理解和应用所学知识。为了便于您学习，本书还提供了习题和实践项目，帮助您在动手实践中巩固所学内容。

为便于教学，本书提供PPT课件、案例代码、教学大纲、授课教案、习题答案等教学资源，可通过扫描下方二维码下载；微课视频可通过扫描书中二维码观看。

PPT 课件　案例代码　教学大纲　授课教案　习题答案

衷心希望本书能成为您学习网络爬虫与数据分析的得力助手，助您在该领域不断精进，取得优异成果。由于编者水平有限，加上科技发展日新月异，书中难免有不足和欠妥之处，恳请您批评、指正。再次表示衷心的感谢！

编　者

2025年11月

目　录

第 1 章

认识 Python

本章旨在帮助初学者掌握 Python 的基本知识和开发环境的配置方法，为后续的数据分析与可视化学习打下基础。通过本章学习，读者可以快速掌握Python编程入门知识，建立起对开发工具与第三方库的初步认识。

主要内容

(1) Python的简介与安装：了解 Python 的特点与应用，掌握其安装及运行方式。
(2) Python 开发工具的使用：介绍 IDLE、PyCharm 和 VS Code 的基本操作。
(3) Python 源配置与库安装：学习配置国内镜像源，使用 pip 安装常用库。

学习目标

(1) 学会安装并运行 Python 程序。
(2) 熟悉主流开发工具的使用。
(3) 掌握如何安装常用第三方库。
(4) 能独立完成简单代码的编写与调试。

本章是学习 Python 的起点，内容虽基础，但至关重要，建议读者认真掌握。

1.1 Python 语言简介

Python 是一种高级、通用、解释型的编程语言，由 Guido van Rossum 于 1991 年设计开发。Python 以其简洁、易读的语法而闻名，被广泛用于 Web 开发、科学计算、人工智能、数据分析等领域。

1.1.1 Python 语言的主要特点

- 简洁易读：Python 采用简单的语法，使得代码易于阅读和理解。它强调代码的可读性和简洁性，使得开发者能够更快地编写出清晰、简洁的代码。
- 动态类型：Python 是一种动态编程语言，不需要显式声明变量的类型，解释器会根据赋给变量的值自动确定其类型。这使得代码编写更加灵活，减少了不必要的类型转换和声明。
- 面向对象：Python 支持面向对象的编程范式，允许开发者使用类、对象、继承、多态等概念来组织和管理代码。这种方式使得代码更易于维护和扩展。
- 可移植性：Python 具有良好的跨平台性，可以在多种操作系统上运行，包括 Windows、Linux、macOS 等。
- 丰富的标准库：Python 拥有丰富而强大的标准库，涵盖了各种常用任务的模块，包括文件操作、网络通信、数据处理、图形界面等，使得开发者能够更方便地完成任务。

1.1.2 Python 语言的应用领域

Python 是一种功能强大且易于学习的编程语言，广泛应用于多个领域，如图 1-1 所示。凭借丰富的第三方库和强大的社区支持，Python 成为从初学者到专业开发者都广泛使用的多领域通用语言。

图 1-1　Python 的应用领域

1. Web 开发

Python 的简洁性和易用性使得它成为了 Web 开发的热门选择，常用的 Django、Flask、FastAPI等框架可快速、高效地构建网站和接口服务。

2. 大数据处理

Python 依靠 Pandas、NumPy、PySpark、Dask 等库高效支持数据清洗、分析和分布式计算，常用于 ETL、日志分析和大规模数据建模。

3. 人工智能 / 机器学习

Python 是人工智能领域的核心语言，拥有 TensorFlow、PyTorch、scikit-learn、OpenCV、NLTK 等强大库，应用于图像识别、语音处理、推荐系统和自然语言处理等场景。

4. 云计算与分布式系统

Python 在云计算中常作为服务开发与资源调度语言，借助 boto3、google-cloud-python、azure-sdk、Celery 等库实现云端接口调用、分布式任务调度和微服务构建。

5. 自动化运维开发

Python 被广泛用于脚本编写和自动化任务，常使用 Ansible、SaltStack、Fabric、psutil 等工具，实现批量部署、系统监控和日志管理。

6. 网络爬虫

Python 拥有丰富的爬虫工具，如 requests、BeautifulSoup、lxml、Scrapy、Selenium 等库，能够高效采集网页内容，适用于电商数据抓取、舆情监控和科研数据收集。

7. 游戏开发

Python 主要用于小型游戏或脚本支持，借助 Pygame、Panda3D、Cocos2d 等框架，可快速开发 2D/3D 游戏或实现 AI 脚本与自动化测试。

1.2 Python 的安装及运行

1.2.1　在 Windows 中安装 Python


配置 Python 环境变量

要使用 Python 语言进行项目开发，首先需要在电脑上搭建 Python 开发环境。下面是在 Windows 操作系统下安装 Python 工具的步骤。

1. 下载 Python 安装程序

通过 Web 浏览器访问 Python 官方网站https://www.python.org/downloads/windows/，如图 1-2 所示，下载适用 Windows 的安装程序包。可以看到网页中有很多版本，推荐下载并安装稳定版或最新版本，本书选取 Python 3.13.5。32-bit 表示适合 32 位操作系统，64-bit 表示适合 64 位操作系统。

图 1-2　Python 官方网站界面

2. 运行安装程序

双击下载的安装程序，选择 Customize installation 选项，按照提示进行安装。在安装过程中，确保选中了 Add Python to environment variables 复选框，如图 1-3 所示。若未选中，则需要自己配置环境变量。

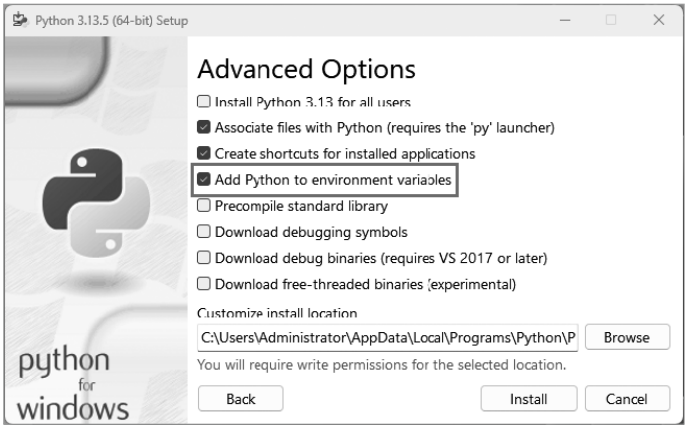

图 1-3　Python 安装界面

3. 验证 Python 安装

通过按 Win+R 快捷键调出运行对话框，输入 cmd 命令进入 DOS 命令界面，在命令行窗口中输入 python 命令，如果可以看到 Python 版本号信息，说明 Python 已经成功安装，如图 1-4 所示。

从图中可以看出，Windows 系统当前使用的 Python 版本是 Python 3.13.5。

图 1-4　Windows 下 Python 的验证

1.2.2　在 Linux 中安装 Python

在 Linux 系统中，Python 通常是默认安装的，但具体情况取决于所使用的发行版本(如 Ubuntu、CentOS、Debian、Fedora 等)和版本号。下面以 Ubuntu 20.04 为例说明如何在 Linux 中安装 Python。

1. 在 Linux 中查看 Python 的版本

在终端输入以下命令查看所安装的 Python 版本。

```
python3 -- version
```

如图 1-5 所示，Linux 中 Python 的版本为 3.8.10，这种情况下需要更新安装 Python 3.13.5。

图 1-5　Linux 下 Python 的验证

2. 在 Linux 中更新安装 Python

在 Linux 中可以使用系统自带的包管理工具更新安装，这是最简单、最稳定的方法。

```
sudo apt update
sudo apt install python3.13 python3.13-pip
```

1.2.3　在 macOS 中安装 Python

在 macOS 系统中，Python 通常也是默认安装的。

1. 在 macOS 中查看 Python 的版本

打开终端界面，使用以下命令查看 Python 的版本。

```
python3 -- version
```

如图 1-6 所示，macOS 中 Python 的版本为 3.13.2，虽然版本不是最新的，但功能上与 Python3.13.5 相同。

图 1-6　MacOS 下 Python 的验证

2. 在 macOS 中更新安装 Python

Homebrew 是 macOS 上最流行的包管理工具，可以方便地下载并安装各种软件，可通过以下命令来安装 Python 的指定版本。

```
brew install python@3.13
```

可以通过以下命令来更新 Python 到最新版本。

```
brew upgrade python
```

1.2.4　Python 的运行

1. 启动交互界面

调出 DOS 命令界面或者终端界面，输入 python 命令进入 Python 交互界面。出现 3 个向右的箭头“>>>”，提示用户输入内容。

```
>>>1+1
2
```

这两行代码表示把“1+1”通过键盘输入到 Python 交互环境中，然后按下Enter键，数字“2”表示Python 交互环境输出的内容。

2. 编写 Python 代码

可以使用文本编辑器(如 Notepad、Sublime Text、Visual Studio Code、PyCharm 等)编写如下 Python 代码。

```
print('Hello World!')
```

在命令行窗口中输入“python 文件名.py”(文件名指编写的 Python 代码文件名)，按Enter键，即可运行 Python 代码。

```
D:\ CrawlerClass\chapter1>python helloWorld.py
Hello World!
```

以上代码表示首先进入到“helloWorld.py”文件所在的文件夹，然后执行 python helloWorld.py命令，第二行为文件输出的内容。

1.3 Python 开发工具介绍

Python 常用的开发工具包括 IDLE、VS Code 和 PyCharm。IDLE 是 Python 自带的简易集成开发环境，适合初学者进行简单编程；VS Code 是一款开源编辑器，支持多种语言和插件，适合各种规模的项目开发，特别是 Web 应用和多语言项目；而 PyCharm 是一个功能强大的专业 IDE，专为 Python 开发设计，适合中大型项目，特别是 Web 开发和数据科学项目，用于高级调试和代码分析。

1.3.1　使用 IDLE

IDLE 是一个简单有效的集成开发环境，在安装 Python 时自动安装，无论是交互式还是文件式，它都有助于快速编写和调试代码，它是小规模 Python 软件项目的主要编写工具。

1. 打开 IDLE

在Windows系统中单击“开始”图标，在“开始”菜单找到 Python 3.13 文件夹，单击文件夹下的 IDLE 图标，即可打开IDLE，如图 1-7 所示。

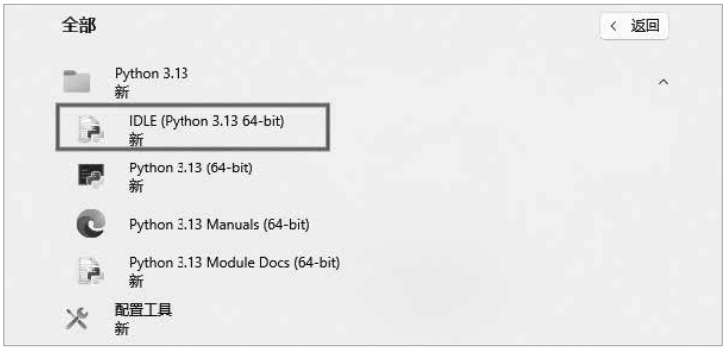

图 1-7　打开 IDLE

2. 运行 Python 程序

可以直接用交互式运行 Python 代码，也可以编写 Python 代码到文件中运行。

(1) 交互式

直接在 IDLE 环境界面的交互环境中输入 Python 代码即可，如图 1-8 所示。

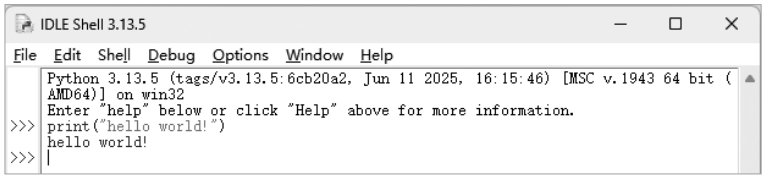

图 1-8　IDLE 的交互式

(2) 文件式

文件式需要先新建文件，写入代码再运行。

① 在菜单栏中选择 File | New File 命令，在打开的空白文件中编写 Python 代码，保存文件，并为文件命名。请确保文件名以“.py”结尾。

② 在菜单栏中选择 Run | Run Module 命令，或者按快捷键 F5，程序将在 IDLE Shell 窗口中运行，并输出结果，如图 1-9 所示。

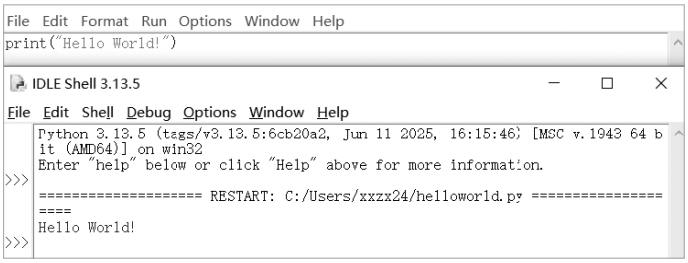

图 1-9　IDLE 的文件式

1.3.2　PyCharm 的安装及使用

PyCharm 是由 JetBrains 公司开发的一款专业的 Python 集成开发环境(IDE)，被广泛应用于 Python 软件开发，特别适合中大型项目和专业开发者使用。

它具有以下主要特点：

- 强大的代码编辑功能：包括智能代码补全、代码检查、重构等，能显著提升开发效率。
- 项目管理与导航：支持复杂项目的结构化管理，便于快速查找文件、类或函数。
- 内置调试与测试工具：提供强大的图形化调试器和单元测试支持。
- 集成工具支持：内置终端、版本控制(如 Git)、数据库管理、任务管理等。
- Web 开发支持(专业版)：支持 Django、Flask 等主流 Web 框架。
- 虚拟环境和包管理：支持 Conda、venv、Pipenv 等，方便项目环境隔离和依赖管理。

1. 下载安装 PyCharm

在 JetBrains 官网(https://www.jetbrains.com/pycharm/download/)下载 PyCharm。网站上方的专业版 Professional 是收费的，下方的社区版 Community 是免费的，如图 1-10 所示，社区版的功能就足够用了。安装 PyCharm，按照软件的提示进行操作。

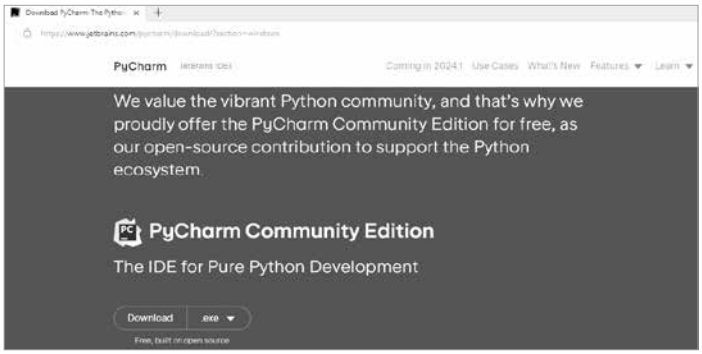

图 1-10　PyCharm 社区版下载页面

2. 创建项目

启动 PyCharm，选择“新建项目”命令。在打开的界面中配置项目名称和存储位置，然后选择“Python 路径”配置 Python 解释器(如果已经安装了 Python，则选择系统中的 Python 解释器)，如图 1-11 所示。

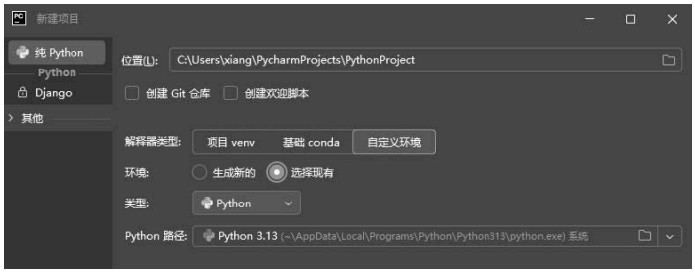

图 1-11　PyCharm 创建项目

3. 编写 Python 代码

在 PyCharm 中选择“新建”|“Python 文件”命令，在文件中编写 Python 代码，如图 1-12 所示。

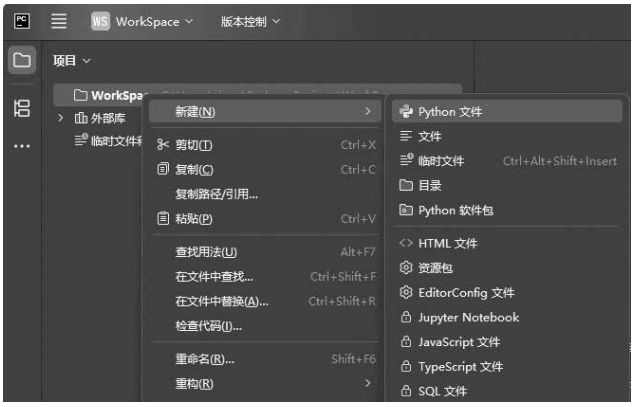

图 1-12　在 PyCharm 中创建 Python 文件

4. 运行程序

单击“运行”按钮(位于菜单栏或工具栏中)，选择要运行的 Python 文件，PyCharm 将在控制台或 Debug 窗口中运行程序并输出结果，如图 1-13 所示。

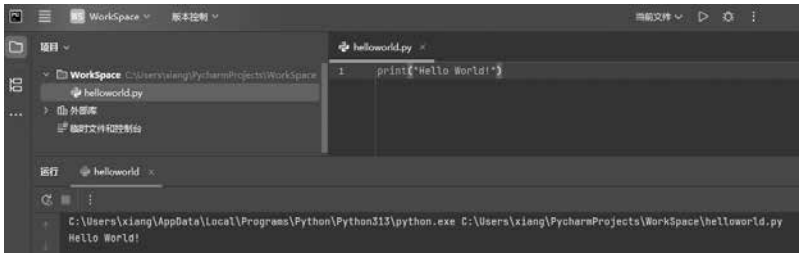

图 1-13　在 PyCharm 中编写代码并运行

1.3.3　Visual Studio Code 的安装及使用


Visual Studio Code 介绍

Visual Studio Code(简称 VS Code)是一款由微软公司开发的免费、开源的轻量级代码编辑器，广泛用于软件开发，它具备以下特点。

- 跨平台支持：可在 Windows、macOS 和 Linux 上运行。
- 强大的编辑功能：支持语法高亮、智能感知(代码补全)、代码片段、语法检查等。
- 内置终端：方便在编辑器中直接运行命令行指令。
- Git 集成：内置 Git 支持，可以直接进行代码版本管理。
- 丰富的扩展插件：通过扩展市场可安装数千种插件，支持多种编程语言(如 Python、Java、C++、JavaScript、Go 等)。
- 调试工具：内置调试器，支持断点、单步调试、变量监视等功能。

1. 下载安装 Visual Studio Code

在官网https://code.visualstudio.com/download中下载 Visual Studio Code适用于 Windows 的安装程序。运行安装程序并按照软件提示进行安装操作。

2. 安装扩展

启动 Visual Studio Code，在侧边栏中选择 Extensions，搜索 Python 和中文扩展并安装，如图 1-14 所示，安装完成后，重新启动 Visual Studio Code。

图 1-14　搜索 Visual Studio Code 扩展并安装

3. 创建项目

在 Visual Studio Code 中选择“文件”|“新建文件夹”命令，设置文件夹的位置和名称，在文件夹中创建一个新的 Python 文件。

4. 编写 Python 代码并运行

在Python 文件中编写 Python 代码，单击“运行”按钮(位于菜单栏或工具栏中)，选择要运行的 Python 文件，或者直接在终端窗口使用命令 python xxx.py，如图 1-15 所示，Visual Studio Code 将在终端窗口中运行程序并输出结果。

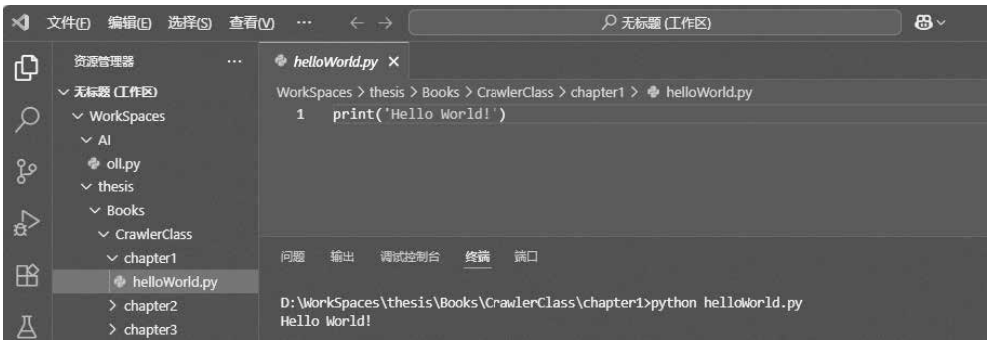

图 1-15　在 Visual Studio Code 中编写代码并运行

1.4 Python 源配置及安装第三方库

1.4.1 Python 源配置

配置 Python 源

学习 Python 的时候，经常需要使用 Python 自带的 pip 工具来下载对应的模块，而用pip下载模块的时候，默认使用 Python 官方提供的下载源。Python 官方下载源地址在国外，国内下载的时候速度不稳定，还容易出错，因此需要修改为国内的下载源。下面给出在 Windows 系统中将 Python 配置为清华源的主要步骤。

(1) 按照pip的官方说明文档，它的配置文件应该放在%APPDATA%/pip/目录下，配置文件名称是pip.ini。先按下 Win+R 快捷键，然后在打开的窗口中输入“%APPDATA%”，再单击“确定”按钮。

(2) 打开的默认路径是 C:\Users\用户名\Appdata\roming 目录，在该目录下创建 pip 目录。在这个目录的空白位置，单击右键，选择“新建”|“文件夹”命名，将新建的文件夹重命名为 pip。

(3) 进入 pip 目录，新建一个文本文件，并命名为 pip.ini。用“记事本”程序打开 pip.ini 文件，如图1-16 所示，在文件中添加以下内容。

```
[ global ]
time-out=60
index-url = https://pypi.tuna.tsinghua.edu.cn/simple/
[ install ]
trusted-host=tsinghua.edu.cn
```

其中，global 区域配置的是下载链接和超时时间，而 install 区域配置的是安装时信任的地址。

图 1-16　pip 源配置

(4) 配置好配置文件后，保存文件并关闭。重新打开一个 cmd 命令行窗口，使用 pip 安装任意一个模块，如图 1-17 所示，可以看到，这里使用的是清华大学的下载地址，下载速度会快很多。

```
C:\Users\xiang>pip install numpy
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple/
Collecting numpy
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/dd/c8/
8c7b/numpy-2.3.1-cp313-cp313-win_amd64.whl (12.7 MB)
Installing collected packages: numpy
Successfully installed numpy-2.3.1
```

图 1-17　清华源安装 NumPy 库

1.4.2　安装 Python 第三方库

Python 语言有标准库和第三方库两类库，标准库随 Python 安装包一起发布，用户可以随时使用，第三方库需要安装后才能使用。Python 第三方库有 3 种安装方式，分别为：pip 工具安装、自定义安装和文件安装。

1. pip 工具安装

最常用且最高效的 Python 第三方库安装方式是采用 pip 工具安装。pip 是 Python 官方提供并维护的在线第三方库安装工具，pip 是 Python 的内置命令，需要通过命令行执行，如图 1-17 所示即为使用 pip 命令安装第三方 NumPy 库。pip 支持安装、下载、卸载、列表、查看、查找等一系列安装和维护子命令。

(1) 安装库的命令格式如下。

```
pip install <拟安装库名>
```

(2) 卸载库的命令格式如下。

```
pip uninstall <拟卸载库名>
```

(3) 可以通过 list 子命令列出当前系统中已经安装的第三方库。

```
pip list
```

(4) pip 的 show 子命令可以列出某个已经安装库的详细信息。

```
pip show <拟查看库名>
```

(5) pip 的 download 子命令可以下载第三方库的安装包，但并不安装。

```
pip download <拟下载库名>
```

(6) pip 的 search 子命令可以联网搜索库名或摘要中的关键字。

```
pip search <拟搜索库名>
```

pip 是 Python 第三方库最主要的安装方式，可以安装 90% 以上的第三方库。在 Windows 操作系统中，有一些第三方库仍然需要用其他方式进行安装。

2. 自定义安装

自定义安装是指按照第三方库提供的步骤和方式安装，第三方库都有主页用于维护库的代码和文档。这种安装方法一般适合用于 pip 中尚无登记或安装失败的第三方库，允许在安装过程中指定特定的选项、路径或配置。下面给出 Python 自定义安装 Django 库的主要步骤。

(1) 下载源代码或二进制文件。

首先，需要下载库的源代码或二进制文件。通常，这些文件可以在库的官方网站或代码存储库中找到，例如需要安装 Django 来开发 Web 应用程序，则可以在官网https://www.djangoproject.com/download/下载源代码。

接下来是解压或安装软件包。如果下载的是源代码，需要先解压缩。如果下载的是二进制文件，双击安装程序即可进行安装。

(2) 进入到软件包目录。

在命令提示符或终端窗口中，使用 cd 命令导航到软件包的目录。例如，如果软件包解压缩到名为“Django-4.1.13”的文件夹中，可以使用以下命令导航到该目录。

```
cd Django-4.1.13
```

(3) 运行安装命令。

通常在命令提示符或终端窗口中使用以下命令以执行自定义安装，这可能需要一些时间，具体取决于软件包的大小和复杂性。如图 1-18 所示，这里省略了具体的安装过程。

```
python setup.py install
```

图 1-18　Django 源代码安装过程

(4) 验证安装。

安装完成后，可以通过导入软件包并运行一些简单的命令或示例代码来验证是否安装成功，如图 1-19 所示。

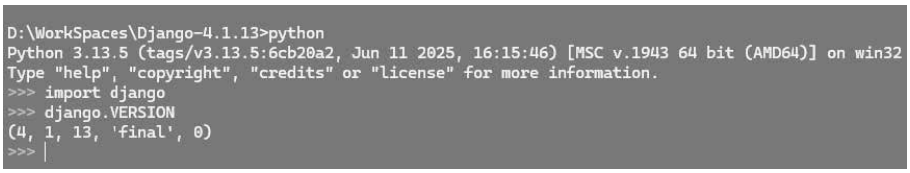

图 1-19　Django 源代码安装验证

3. 文件安装

由于 Python 某些第三方库仅提供源代码，通过 pip 下载文件后无法在 Windows 系统编译安装，会导致第三方库安装失败。在 Python 中，.whl 文件是一种预编译的 Python 包格式，通常用于分发和安装。这种格式的文件包含了 Python 包的所有内容，包括模块、扩展模块(如果有的话)以及元数据。安装.whl 文件非常简单，从网络上下载适用于 Python 版本的解释器和系统的.whl 文件，然后采用 pip 命令安装该文件即可。

```
pip install 目录\文件名.whl
```

下面给出使用文件安装 Pandas 库的主要步骤。

(1) 下载.whl 文件。

这些文件可以在官方网站或代码存储库中找到，在网址 https://pypi.org/project/pandas/#files 下载，要下载同操作系统和 Python 版本相匹配的.whl 文件，例如操作系统是 Windows 11 64 位、Python 为 3.13.5，则需要下载如图 1-20 所示的“pandas-2.3.1-cp313-cp313-win_amd64.whl”。

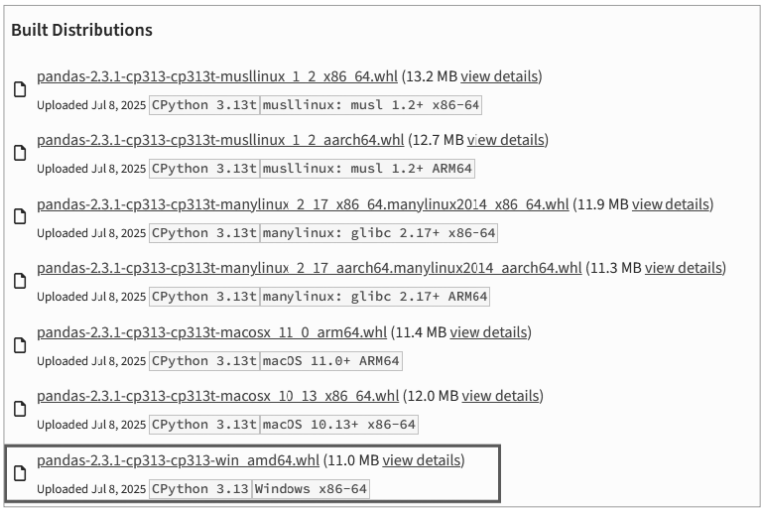

图 1-20　Pandas 库的 .whl 文件下载页面

(2) 进入软件包目录。

在命令提示符或终端窗口中，使用 cd 命令导航到软件包的目录。例如，如果.whl 文件在“WorkSpaces”文件夹中，可以使用以下命令导航到该目录。

```
cd WorkSpaces
```

(3) 运行安装命令。运行安装命令以执行自定义安装，在命令提示符或终端窗口使用以下命令。

```
pip install pandas-2.3.1-cp313-cp313-win_amd64.whl
```

等待安装完成。一旦运行了安装命令，需要等待软件包安装完成。这可能需要一些时间，具体取决于软件包的大小和复杂性，如图 1-21 所示，这里省略了安装的具体过程。

图 1-21　Pandas 库的 .whl 文件安装

(4) 验证安装。

安装完成后，可以通过导入软件包并运行一些简单的命令或示例代码来验证安装是否成功，如图 1-22 所示。

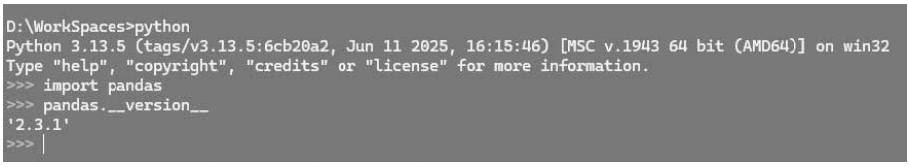

图 1-22　Pandas 库的 .whl 文件安装验证

本章小结

本章主要介绍了 Python 语言的基本特点及其应用场景，帮助读者初步认识Python语言。内容包括在 Windows、 Linux 和 macOS 系统下的安装方法，Python 程序的运行方式(交互式与脚本模式)，常用开发工具(IDLE、PyCharm、VS Code)的使用，以及源配置和第三方库的安装方法。通过学习，读者能够熟练完成 Python 开发环境的搭建，并掌握基础的代码编写与运行方式，为后续的学习和实际编程奠定良好的基础。

本章习题

1. 为什么 Python 被称为高级、通用、解释型编程语言？简要说明其特点。

2. IDLE、PyCharm 和 Visual Studio Code 这三种 Python 开发环境各有什么特点？你会选择哪一种？为什么？

3. 请列举几个常用的 Python 第三方库，并简要说明它们的用途。

4. 除了使用 pip 工具安装第三方库外，你还了解哪些安装第三方库的方法？请简要说明其原理和操作步骤。

5. 如果需要安装一个本地文件中的第三方库，你会选择使用哪种安装方式？为什么？

6. 你认为学习 Python 语言的最大优势是什么？它在你的工作或学习中有何实际应用？

第 2 章

Python 语言基础

本章旨在帮助读者掌握 Python 编程的核心语法规则，包括变量、数据类型、控制结构、函数、面向对象编程和文件操作等内容。掌握这些基础语法，是实现数据处理、爬虫开发和数据可视化等高级应用的前提。

主要内容

(1) 基本语法规则：包括缩进、注释、变量定义以及输入输出方法。
(2) 数据类型：重点介绍数字、字符串、列表、元组、字典、集合等常见类型。
(3) 程序控制结构：通过条件语句和循环语句控制程序流程。
(4) 函数：讲解函数的定义、调用和参数使用，及使用函数的常见注意事项。
(5) 面向对象编程：介绍类的定义与调用，初步理解面向对象思想。
(6) 文件操作：包括对文本文件和 CSV 文件的读写。

学习目标

(1) 掌握 Python 的语法基础和代码书写规范。
(2) 能熟练使用常见的数据类型进行编程操作。
(3) 掌握 if、for、while 等程序控制语句的用法。
(4) 能够编写和调用简单的函数。
(5) 初步理解类与对象的基本概念。
(6) 学会读写文件，为后续数据处理做准备。

本章内容是 Python 编程的核心基础，建议结合实例多练习，边学边做，以巩固理解和提高实操能力。

2.1 语法规则

Python的语法简洁直观，强调可读性。其基本语法规则包括缩进、注释、变量和输入输出等。在编写Python 程序时，要注意语法一致性、变量命名规范及代码结构清晰。

2.1.1 缩进

Python 通过不同层次的缩进表达功能范围的界定。一般在 Python 中以 4 个英文空格为单位实现不同层级的缩进，且不同层级的缩进量必须相同，示例如下。

```
if a>5:                                    # 层级1
     print("a>5")                              # 层级2
else:                                      # 层级1
     print("a<5")                              # 层级2
```

2.1.2 注释

Python 在程序中用注释说明或解释具有特定用途、功能、注意点等的关键信息。

1. 单行注释

单行注释用于简单功能说明，使用 # 实现，示例如下。

```
>>> print('Hello Python')          # 输出字符串——这里是注释信息
```

2. 多行注释

多行注释用于注释功能较多尤其是模块化封装的功能，如函数、类的文档字符串等。多行注释使用三组单引号(' ')或者三组双引号(" ")将注释括起来，示例如下。

```
' ' '
这是多行注释的示例内容
' ' '
print('Hello Python')
```

2.1.3 变量

Python 中的变量需要先定义再使用，以下为常见的命名规则。

(1) 变量名由数字、字母、下画线组成，但不能以数字开头，例如 she_he、_she、she_123 等都是合法的变量名。

(2) 变量名不能使用 Python 关键字，如 if、while、enumerate、true 等，也就是说在命名时要避开 Python 语言中的关键字，Python 语言中的 33 个关键字都不能作为变量名

来使用。这 33 个关键字如表 2-1 所示。

表 2-1　Python 关键字

false	none	true	and	as	assert	break
class	continue	def	elif	else	except	del
finally	for	from	global	if	import	in
nonlocal	lambda	is	not	or	pass	raise
return	try	whilt	with	yield		

(3) 变量名区分大小写，例如 a=1 和 A=1 是不同的变量。

(4) 使用可理解、行业通用的变量名，而非无意义的简写，以便于理解变量含义。

2.1.4　输入和输出函数

1. input( ) 函数

在 Python 中，利用 input( ) 对控制台输入的内容进行读取的时候，input( ) 函数会将输入的内容当成字符串类型进行读取和保存，并在程序需要的时候返回这个字符串内容，示例如下。

```
name = input("请输入的名字：")    # 输入：小明
print("您好，" + name)           # 输出为：您好，小明
```

2. print( ) 函数

和输入函数 input( ) 相对应的就是基本输出函数 print( )。在 Python 中，print( ) 函数之后是默认换行的，十分方便，示例如下。

```
print("Hello , World!")          # 输出字符串。输出为：Hello,World!
```

除了输出字符串，print( ) 函数也可以输出其他类型的变量，例如整数、浮点数、列表、字典等，示例如下。

```
age = 25
print("您的年龄是：" , age)       # 输出为：您的年龄是：25
```

2.2　数据类型

Python 中的数据类型丰富，常用的有以下四种：

(1) 数字类型用于数学运算。

(2) 序列类型用于存储有序元素。

(3) 字典类型适合快速查找。

(4) 集合类型用于去重和集合运算。

了解这些基础类型，有助于读者编写功能完善、结构清晰的 Python 程序。

2.2.1 数字类型

1. 整数

Python 数字类型中的整数是指没有小数部分的数值。在 Python 中，整数类型为 int。整数可以是正数、负数或零，示例如下。

```
>>>print(42)
42
```

Python 中整数类型的加、减和乘法分别用符号“+”“-”和“*”表示，一般会通过括号来改变运算的优先级，示例如下。

```
>>> print(2 + 3)
5
>>> print(2 + 3 * 4)
14
>>> print((2 + 3) * 4)
20
```

2. 浮点数

Python 数字类型中的浮点数是指带有小数部分的数值。在 Python 中，浮点数类型为 float。浮点数可以是正数、负数或零，示例如下。

```
>>> print(3.14)
3.14
```

Python 中浮点数的计算与整数的计算类似，同样支持加、减、乘、除和取余数等运算，示例如下。

```
>>> print(2.5 + 3.7)
6.2
>>> print(8.0 - 5.6)
2.4
>>> print(4.2 * 5.0)
21.0
>>> print(10.5 / 2)
5.25
>>> print(10.5 % 3)
1.5
```

需要注意的是，由于浮点数受精度限制，浮点数计算可能存在精度误差问题，示例如下。

```
f = 0.1 + 0.2
print(f)  # 输出结果为0.30000000000000004
```

在这个示例中，本来期望的结果是 0.3，但由于浮点数精度，实际计算结果是 0.30000000000000004。

如果需要进行精确的数值计算，建议使用 Python 中的 decimal 模块或 fractions 模块。

3. 布尔数

Python 数字类型中的布尔数(Boolean)是指只有 True 和 False 两个取值的数据类型。在 Python 中，布尔类型为 bool，示例如下。

```
a = True
b = False
```

在这个示例中，a 和 b 分别被赋值为布尔值 True 和 False。

布尔数通常用于判断条件语句中，例如 if 语句和 while 语句等，用来决定代码的执行路径，示例如下。

```
x = 5
y = 7
if x < y:
    print("x小于y")
else :
    print("x大于等于y")          # 输出结果为  x小于y
```

在这个示例中，使用了比较运算符“<”来比较 x 和 y 的大小关系，结果是 True，因此 if 语句中的代码会执行，输出“x 小于 y”。

Python 中的布尔运算符包括 and、or 和 not 等。

(1) and 运算符。

用于连接两个布尔表达式，只有在两个表达式都为 True 时，结果才为 True，否则为 False，示例如下。

```
a = True and False
print(a)  # 输出结果为 False
```

(2) or 运算符。

用于连接两个布尔表达式，只要有一个表达式为 True，结果就为 True，只有两个表达式都为 False 时，结果才为 False，示例如下。

```
b = True or False
print(b)          # 输出结果为True
```

(3) not 运算符。

用于对单个布尔表达式取反。如果表达式为 True，则返回 False；如果表达式为 False，则返回 True，示例如下。

```
c = not True
print(c)  # 输出结果为 False
```

需要注意的是，Python 中的布尔数实际上也是整数类型的子类型，其中 True 等价于整数 1，False 等价于整数 0。因此，可以将布尔数和整数进行运算，示例如下。

```
x = True + 2
print(x)  # 输出结果为 3
```

在这个示例中，True 被转换为整数 1，与 2 相加后得到整数 3。

2.2.2 序列类型

Python 序列类型一般指列表、元组和字符串。所有序列都可以进行特定的操作，这些操作包括：索引、分片、加、乘以及检查某个元素是否属于序列的成员。除此之外，Python 还有计算序列长度，找出最大元素和最小元素的内建函数。需要注意的是 Python 序列类型的下标都是从 0 开始的。

1. 字符串

在 Python 中，字符串是由数字、字母、下画线组成的一串字符。字符串是用一对单引号(')或双引号(")括起来的，示例如下。

```
message1 = 'Hello, World!'
message2 = "Hello, World!"
```

在这个示例中，变量 message1 和 message2 输出是一样的，都是Hello, World!。在 Python 中，字符串是不可变的，这意味着一旦字符串被创建，就不能在原地修改它。但可以使用字符串方法进行操作和修改。

需要注意的是，字符串形式的数字和普通数字是不一样的，它们不相等，示例如下。

```
message1 = '12'
message2 = 123
```

在这个示例中，message1 代表的是字符串，而 message2 代表的是整数。

在 Python 中，字符串是一种常见的数据类型，它包含一系列字符，可以使用各种操作对其进行处理和操作，以下是 Python 字符串的一些常见操作。

(1) 连接字符串。

使用加号(+)或字符串拼接符号(*)可以将多个字符串连接成一个字符串，示例如下。

```
str1 = "Hello"
str2 = "World"
new_str = str1 + " " + str2            # 通过加号连接字符串
print(new_str)                         # 输出 " Hello World"
repeated_str = str1 * 3                # 通过乘号重复字符串
print(repeated_str)                    # 输出 " HelloHelloHello "
```

(2) 切片。

使用下标(索引)来获取字符串中的一个子字符串，示例如下。

```
my_str = "Hello , World!"
sub_str = my_str [0:5]          # 获取从第一个字符到第五个字符的子字符串
print(sub_str)                  # 输出 " Hello "
sub_str2 = my_str [7:]          # 获取从第七个字符到字符串末尾的子字符串
print(sub_str2)                 # 输出 "World !"
```

(3) 查找和替换。

使用字符串的内置方法来查找和替换特定的字符或子字符串，示例如下。

```
my_str = "Hello, World!"
index = my_str.index("o")                      # 获取 o 字符的索引
print(index)                                   # 输出 4

new_str = my_str.replace ("World" , "Python")  # 将 World 替换为 Python
print(new_str)                                 # 输出 "Hello, Python !"
```

(4) 大小写转换。

使用字符串的内置方法将字符串转换为大写或小写格式，示例如下。

```
my_str = "Hello, World!"
new_str = my_str.upper( )              # 将字符串转换为大写格式
print(new_str)                         # 输出"HELLO, WORLD!"

new_str2 = my_str.lower( )             # 将字符串转换为小写格式
print(new_str2)                        # 输出" hello, world!"
```

2. 列表

在 Python 中，列表(list)是一种有序的数据类型，用于存储一组元素。列表使用方括号[ ]表示，其中的元素用逗号分隔，以下是一个简单的列表示例。

```
my_list = [1, 2, 3, 4, 5]
```

创建空列表可以使用 [ ] 或者 list( ) 函数。以下是创建空列表的示例。

```
empty_list = [ ]
empty_list = list( )
```

列表可以包含不同类型的元素，例如字符串、整数、浮点数，甚至是其他列表。以下是一个包含不同类型元素的列表示例。

```
mixed_list = ["hello", 123, 3.14, [1, 2, 3]]
```

列表是可变的，这意味着可以通过索引访问或修改它们的元素。例如，要访问列表中的第一个元素，可以使用以下语法。

```
first_element = my_list [0]      #first_element 为 1
```

要将列表中的元素更改为不同的值，可以使用以下语法。

```
my_list [0] = "hello"            #更改值之后 my_list 变为['hello', 2, 3, 4, 5]
```

列表还支持许多操作和方法，例如添加、删除和排序元素，以下是一些常用的列表操作和方法。

(1) 追加元素。

使用 append( ) 方法将一个元素添加到列表的末尾，示例如下。

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)           # 输出[1 ,2 ,3 ,4]
```

(2) 插入元素。

使用 insert( ) 方法将一个元素插入到列表的指定位置，示例如下。

```
my_list = [1, 2, 3] my_list.insert(1, 4)
print(my_list)           # 输出 [1, 4, 2, 3]
```

(3) 删除元素。

使用 remove( )方法删除指定的元素，或使用 del 语句删除指定位置的元素，示例如下。

```
my_list = [1, 2, 3, 4]
my_list.remove(3)
print(my_list)           # 输出 [1, 2, 4]

del my_list [1]
print(my_list)           # 输出 [1, 4]
```

(4) 切片。

使用索引获取列表的一个子列表。注意列表切片是前闭后开结构，示例如下。

```
my_list = [1, 2, 3, 4, 5]
sub_list = my_list [1:3]
print(sub_list)   # 输出 [2,3]
```

切片的结果包括“开始位置的下标”所对应的元素，但是不包括“结束位置的下标”

所对应的元素。

(5) 迭代。

使用 for 循环遍历列表的所有元素，示例如下所示。

```
my_list = [1, 2, 3, 4]
for element in my_list :
    print(element)
# 输出:
1
2
3
4
```

(6) 连接列表。

使用 + 运算符将两个列表连接成一个，示例如下。

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
new_list = list1 + list2
print(new_list)          # 输出 [1 ,2 ,3 ,4 ,5 ,6]
```

3. 元组

在 Python 中，元组(tuple)是一种有序的、不可变的数据类型，用于存储一组元素。元组使用圆括号( )表示，其中的元素用逗号分隔，以下是一个简单的元组示例。

```
my_tuple = (1, 2, 3, 4, 5)
```

创建空元组可以使用 ( ) 或者 tuple( ) 函数。以下是创建空元组的示例：

```
empty_tuple = ( )
empty_tuple = tuple( )
```

与列表不同，元组是不可变的，这意味着一旦创建就无法修改元素。但是，可以通过索引访问元组中的元素。例如，要访问元组中的第一个元素，可以使用以下语法。

```
first_element = my_tuple [0]            #first_element 为 1
```

元组可以包含不同类型的元素，例如字符串、整数、浮点数，甚至其他元组。以下是一个包含不同类型元素的元组示例。

```
mixed_tuple = ("hello", 123, 3.14, (1, 2, 3))
```

元组支持许多操作和方法，例如查找元素和计算元素出现的次数。以下是一些常用的元组操作和方法。

```
index = my_tuple.index (3)      # 查找元素3的索引，结果为2
count = my_tuple.count(3)       # 计算元素3出现的次数，结果为1
```

4. 元组和列表的区别

在 Python 中，元组和列表都是可以存储多个元素的数据类型，但它们有以下几个重要的区别。

- 可变性：元组是不可变的，而列表是可变的。这意味着元组一旦创建就不能修改，而列表可以通过添加、删除或替换元素来修改。
- 语法：元组使用圆括号( )表示，而列表使用方括号[ ]表示。
- 性能：由于元组是不可变的，它们比列表更节省内存，并且在某些情况下可以提供更好的性能。

以下是元组和列表区别的示例。

```
my_tuple = (1, 2, 3, 4, 5)
my_list = [1, 2, 3, 4, 5]
```

可以通过索引访问元组和列表中的元素，但是对于列表，可以通过索引来修改或删除元素，而对于元组则不行。

例如，要将列表中的第一个元素更改为另一个值，可以使用以下语法。

```
my_list[0] = "hello"
```

但是，如果尝试对元组进行相同的操作，则会导致 TypeError 错误，如图 2-1 所示。

```
>>> my_list = [1, 2, 3, 4, 5]
>>> my_list[0] = "hello"
>>> print(my_list)
['hello', 2, 3, 4, 5]
>>> my_tuple = (1, 2, 3, 4, 5)
>>> my_tuple[0] = "hello"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

图 2-1　元组和列表的区别

总的来说，元组适用于那些不需要修改元素的场景，可以作为函数返回值或用于保护不希望被修改的数据。而列表则适用于那些需要频繁修改元素的情况。

2.2.3　字典与集合类型

1. 字典

在 Python 中，字典(dictionary)是一种无序的、可变的数据类型，用于存储键-值对。字典使用花括号{}表示，其中的元素以键值对的形式存储，每个键值对由键和值组成，中间用冒号:分隔，以下是一个简单的字典示例。

```
my_dict = {"name": "小明" , "age": 25, "country": "中国"}
```

这个字典包含三个键值对，其中键 “name”的对应值为“小明”，键“age”的对应值为25，键“country”的对应值为“中国”。创建空字典可以使用 {} 或者 dict( ) 函数，以下是创建空字典的示例。

```
empty_dict = {}
```

或

```
empty_dict = dict( )
```

字典的值可以是任何数据类型，例如字符串、整数、浮点数、列表、元组或其他字典。键必须是唯一的，但值可以重复，以下是一个包含不同类型键和值的字典示例。

```
mixed_dict = {"name": "小李" , "age": 20, "country": "中国" , "pets": ["猫" ,
              "狗"] , "score": {"math": 90, "english": 85}}
```

字典中的值可以通过键来访问。例如，要访问“name”对应的值，可以使用以下语法。

```
name = my_dict["name"]   # 结果为：小明
```

要修改字典中的值，可以使用相同的语法，并指定一个新的值。例如，要将 “age”对应的值更改为 30，可以使用以下语法。

```
my_dict["age"] = 30
```

字典还支持许多操作和方法，例如添加、删除和查找键值对，以下是一些常用的字典操作和方法。

```
my_dict["email"] = "cxtc@example.com"
# 添加一个键值对，字典my_dict变为{' name ': ' 小明 ' , ' age ': 25 , ' country ':
                                  ' 中国 ' , ' email ': ' cxtc@example.com '}
del my_dict["country"]
# 删除一个键值对，字典my_dict变为 {'name': '小明', 'age': 25, 'email': 'cxtc@example.com'}
keys = my_dict.keys( )
# 获取所有键的列表， keys 的值为 dict_keys ([ 'name ' ,' age' ,' email '])
values = my_dict.values( )
# 获取所有值的列表，values 的值为 dict_values ([ ' 小明' ,25 ,' cxtc@example.com '])
```

2. 集合

在 Python 中，集合(set)是一种无序的、不可重复的数据类型，用于存储一组唯一的元素。集合使用花括号{}来表示，其中的元素用逗号分隔，以下是一个简单的集合示例，该集合包含五个不同的整数元素。

```
my_set = {1, 2, 3, 4, 5}
```

创建空集合只能通过函数 set( ) 创建，以下是创建空集合的示例。

```
empty_set = set( )
```

集合非常适合用于去重或对元素进行快速的成员检查，以下是一个包含重复元素的集合示例。

```
my_set = {1, 2, 3, 3, 4, 5, 5}
print(my_set)            # 输出 {1, 2, 3, 4, 5}
```

这个集合包含五个不同的整数元素，重复的元素被自动去除。

集合可以包含任何可哈希(hashable)的元素。例如，可以将字符串、整数、浮点数、元组或其他集合添加到集合中，以下是一个包含不同类型元素的集合示例。

```
mixed_set = {"hello" , 123, 3.14 , (1 , 2, 3)}
```

集合支持许多操作和方法，例如添加、删除、求交集和求并集，以下是一些常用的集合操作和方法。

```
my_set.add(6)     # 添加一个元素，集合my_set的值为 {1 ,2 ,3 ,4 ,5 , 6}
my_set.remove(3)  # 删除一个元素，集合my_set的值为 {1 ,2 ,4 ,5 , 6}
intersection = my_set.intersection ({4 , 5, 7}) # 获取两个集合的交集，intersection
的值为 {4 , 5}
union = my_set.union({5, 6, 7})      # 获取两个集合的并集， union 的值为 {1, 2, 4, 5, 6, 7}
```

3. 字典与集合的区别

Python 语言中，字典和集合都是用来存储数据的容器，但它们之间有以下两个关键的区别。

- 数据类型：字典是一种键值对(key-value) 映射的数据结构，其中每个键都是唯一的；而集合是一组唯一且无序的元素的集合。
- 用途：字典通常用于存储和访问具有唯一标识符的数据，例如用户信息、配置文件等。集合通常用于存储和操作无序的、不重复的元素集合，例如去重、交集、并集等操作。

总的来说，如果需要存储键值对，并且需要通过键来访问它们的值，使用字典；如果需要存储无序的、不重复的元素集合，则使用集合。

2.3 程序的控制结构

了解Python 的控制结构是理解程序逻辑的关键，包括选择结构和循环结构，能让程序具有决策和重复能力。

其中选择结构用于根据条件判断执行不同的代码路径，循环结构则实现代码的重复执行。在使用时要注意缩进规范、避免死循环、合理设置条件表达式和避免嵌套过深等，以提高程序的可读性和效率。

2.3.1　选择结构

选择结构通过判断某些特定条件是否满足来决定下一步的执行流程，是非常重要的控制结构。常见的选择结构有单分支选择结构、双分支选择结构、多分支选择结构和嵌套的分支结构。形式比较灵活多变，具体使用哪一种最终还是取决于要实现的业务逻辑。

1. 单分支选择结构

单分支选择结构通过 if 语句实现，如果条件成立，则执行代码块，否则跳过代码块，继续执行后续代码，它的语法表达格式如下。

```
if 表达式：
      执行语句块
```

示例如下。

```
x = 5
if x > 0:
   print("x是正数")       # 输出 "x是正数"
```

需要注意的是，Python 中的 if 语句必须以冒号 (:) 结尾，并且代码块必须缩进，通常使用四个空格进行缩进。

2. 双分支选择结构

双分支是指如果 if 条件成立，则执行满足条件的代码，如果不成立，则跳转到 else，执行 else 下面的代码，它的语法表达格式如下。

```
if 表达式：
      执行语句块 1
else：
      执行语句块 2
```

示例如下。

```
answer = 2
if answer == 2:
     print( '回答正确 ')
else:
     print( '回答错误 ')  #输出 "回答正确"
```

3. 多分支选择结构

多分支是指如果 if 条件成立，则执行满足条件的代码，如果不成立，则跳转到 elif 进行判断，若 elif 条件成立，则执行 elif 条件下的代码。在多分支结构中，可以有多个 elif 条件语句的存在。如果所有的 if 条件语句和 elif 条件语句都不符合，则跳转到 else 下的代码执行，它的语法表达格式如下。

```
if 表达式 a：
      执行语句块 1
elif 表达式 b：
      执行语句块 2
elif 表达式 c：
      执行语句块 3
...
else：
      执行语句块 n
```

示例如下。

```
if info == 'name' :
     num = 1
elif info == 'age ' :
     num = 2
elif info == 'depart ' :
     num = 3
elif info == 'major ' :
     num = 4
else:
     num = 5
```

如果有多个 if，写起来会很烦，使用“if...elif...else...”会让代码显得冗长。如果使用字典改写，代码就会变得非常简洁，示例如下。

```
info_dict = { 'name' : 1, 'age ' : 2, 'depart ' : 3, 'major ' : 4}
num = info_dict.get(age , 5)
```

这段代码表示输出 info_dict 字典中键所对应的数字，否则返回数字 5。

4. 嵌套的分支结构

选择结构嵌套是指在大选择结构下，还在每个分支下包含着小的选择结构，满足条件跳转到大选择结构下的分支后，进入到另一个选择结构中进行更加详细的条件判断和筛选。使用嵌套选择结构时，一定要严格控制好不同级别代码块的缩进量，因为这决定了不同代码块的从属关系和业务逻辑是否能被正确实现，以及代码能否被 Python 正确理解和执行，它的语法表达格式如下。

```
if 表达式a:
      if 表达式 b:
          执行语句块1
    else b:
          执行语句块2
else a:
      执行语句块3
```

示例如下。

```
a = 80
if a>0:
     if a>=60:
       print('及格')
     else:
       print('不及格')
else:
print('分值为负数')       # 输出 '及格 '
```

2.3.2　循环结构

常用的循环结构是 for 循环和 while 循环。

1. for 循环

for 循环用来遍历任何可迭代对象，例如字符串、列表、元组、字典等，基本语法格式如下。

```
for 变量 in 可迭代对象 :
   代码块
```

其中，变量为每一次遍历的对象，可迭代对象则是被遍历的对象，可以是字符串、列表、元组、字典等。

例如，以下代码演示了如何使用 for 循环输出一个列表中的所有元素。

```
fruits = ["apple" , "banana" , "pear"]
for x in fruits :
     print(x)
# 输出:
apple
banana
pear
```

在执行这段代码时，变量 x 依次代表了列表中的每一个元素，代码块则会针对每一个元素执行一次，输出该元素的值。

需要注意的是，在 for 循环中，代码块必须缩进，通常使用四个空格进行缩进。另外，for 循环还支持range( )函数结合使用，用于遍历一系列的数字。例如，以下代码演示了如何使用 for 循环遍历数字 1 到 5。

```
for x in range(1 , 6) :
     print(x)

# 输出:
```

```
1
2
3
4
5
```

这段代码中，range( ) 函数的参数是前闭后开形式，所以 range(1, 6) 表示生成一个从 1 到 5 的数字序列，变量 x 依次代表了这个数字序列中的每一个数字，代码块则会针对每一个数字执行一次，输出该数字的值。

2. while 循环

while 循环用于重复执行一段代码，直到满足特定的条件为止。while 循环会在每次执行循环体之前先检查条件是否成立，只有在条件成立的情况下才会执行循环体，直到条件不成立为止，基本语法格式如下。

```
while 条件 :
    代码块
```

其中，条件为一个表达式，当表达式的值为 True 时，执行循环体；否则跳过循环体，继续执行后续代码。

例如，以下代码演示了如何使用 while 循环输出数字 1 到 5。

```
i = 1
while i <= 5:
    print( i )
    i += 1

# 输出：
1
2
3
4
5
```

在这个例子中，变量 i 的初始值为 1，while 循环不断检查条件 i<=5 是否成立，只要成立，就会执行循环体，输出当前的 i 值，然后将 i 的值加 1，继续下一次循环，直到 i 的值大于 5 时，条件不成立，跳出循环，执行后续代码。

需要注意的是，while 循环中的代码块必须缩进，通常使用四个空格进行缩进。同时，要避免出现死循环的情况，即条件永远满足，导致程序无法跳出循环。

2.4 函数

在 Python 中，函数是一段可重用的代码块，用于完成特定的任务。函数可以接受输