高等院校计算机应用系列教材

Python程序设计

吴仁群　编著

清華大學出版社
北　京

内 容 简 介

本书是一本实用性很强的 Python 语言基础教程，既讲解了 Python 程序设计的基础知识，又提供了大量实用性很强的编程实例。本书共分 8 章，分别介绍了 Python 语言概述、Python 语言基础、函数与模块、常见数据结构、迭代器与生成器、面向对象程序设计、Python 异常处理机制、文件和数据库操作等。

本书内容丰富、结构清晰、案例典型，具有很强的实用性和可操作性，适合作为高等院校 Python 程序设计课程的配套教材，也可作为广大计算机用户的自学参考书。

图书在版编目 (CIP) 数据

Python 程序设计 / 吴仁群编著 . -- 北京 : 清华大学出版社 , 2025. 8. -- (高等院校计算机应用系列教材).

ISBN 978-7-302-70077-7

Ⅰ. TP312.8

中国国家版本馆 CIP 数据核字第 20256UZ420 号

责任编辑：刘金喜

封面设计：高娟妮

版式设计：思创景点

责任校对：成凤进

责任印制：沈　露

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：三河市铭诚印务有限公司

经　　销：全国新华书店

开　　本：185mm×260mm　　**印　　张：**13.5　　**字　　数：**345 千字

版　　次：2025 年 9 月第 1 版　　**印　　次：**2025 年 9 月第 1 次印刷

定　　价：58.00 元

产品编号：106486-01

前言

Python语言是一种解释型、面向对象且具有动态语义的高级编程语言。根据TIOBE官网发布的2025年5月编程语言排行榜，Python、C++和C位居前三，Java排名第四。Python分别在2007年、2010年、2018年、2020年、2021年、2024年6次获得TIOBE“年度编程语言”称号，成为史上获得该称号次数最多的编程语言。Python的成功得益于其简洁易用的语法、强大的数据处理能力、活跃的社区支持、跨平台兼容性以及出色的并行处理能力。此外，Python在Web开发、数据分析、人工智能、机器学习、游戏开发等多个领域展现了广泛的应用场景。

作为一本实践性很强的Python语言基础教材，本书具有以下特点。

(1) 涵盖Python程序设计语言的基础知识，讲解内容由浅入深，符合学生学习计算机语言的规律。

(2) 遵循理论知识和实践知识并重的原则，尽量采用图例的方式阐述理论知识，并辅以大量实例，帮助学生理解、巩固和运用所学知识。

(3) 大部分章节提供综合性实例，这些综合性实例具备知识综合性、紧密联系实际和较强的启发性，可以帮助学生灵活运用各种知识，举一反三地解决实际问题。

本书共分8章。第1章介绍了Python语言的发展历程、特点、开发平台和开发过程，以及如何进行程序调试；第2章介绍了Python语言编程的基础语法，包括变量和数据类型、表达式、控制语句和循环语句等；第3章介绍了Python函数和模块的定义及使用；第4章介绍了常用数据结构，如字符串、列表、元组、集合、字典、栈和队列；第5章介绍了Python语言迭代器与生成器的概念及其用法；第6章介绍了Python的面向对象程序设计基础；第7章介绍了Python的异常处理机制；第8章介绍了Python的输入和输出及数据库操作。

本书由吴仁群编著。在编写过程中，得到了清华大学出版社的大力支持。同时，编者参考了本书“参考文献”中所列举的图书，特此向这些书籍的作者及清华大学出版社表示诚挚的感谢。此外，本书的出版得到了北京印刷学院学科专项(21090124017)的资助。

本书附赠教学课件、案例源代码、教学大纲、教案和教学日历，读者可通过扫描下方二维码进行下载。

教学资源

由于时间仓促，书中难免存在一些不足之处，欢迎广大读者批评指正。

服务邮箱：476371891@qq.com。

编　者

2025年3月

目录

第1章 Python语言概述 1

1.1 Python语言发展历程及特点 1

1.1.1 Python语言的发展历程 1

1.1.2 Python语言的特点 2

1.1.3 Python语言的应用 3

1.2 Python开发环境配置 3

1.2.1 Python开发环境 3

1.2.2 Python安装 4

1.2.3 环境变量设置 7

1.2.4 用户模块文件管理 10

1.3 Python的使用 11

1.3.1 命令行方式 11

1.3.2 IDLE方式 12

1.3.3 Spyder方式 14

1.4 本章小结 14

1.5 思考和练习 14

第2章 Python语言基础 15

2.1 Python基础语法 15

2.1.1 Python程序基本框架 15

2.1.2 Python编码 16

2.1.3 Python注释 17

2.1.4 行与缩进 18

2.1.5 常用的函数和语句 19

2.1.6 Python关键字 21

2.1.7 Python标识符 22

2.2 变量与数据类型 23

2.2.1 变量 23

2.2.2 数据类型概述 25

2.2.3 可变类型和不可变类型的内存分配区别 31

2.2.4 数据类型转换 33

2.3 运算符和表达式 34

2.3.1 算术运算符与算术表达式 34

2.3.2 关系运算符与关系表达式 35

2.3.3 逻辑运算符与逻辑表达式 35

2.3.4 赋值运算符与赋值表达式 36

2.3.5 位运算符 37

2.3.6 成员运算符 38

2.3.7 身份运算符 38

2.3.8 运算符优先级 39

2.4 条件控制与循环语句 40

2.4.1 条件控制语句 40

2.4.2 循环语句 43

2.4.3 跳转语句 47

2.5 综合应用 49

2.6 本章小结 50

2.7 思考和练习 50

第3章 函数与模块 51

3.1 函数 51

3.1.1 函数定义和调用 51

3.1.2 函数参数说明 54

3.1.3 变量作用域 62

3.1.4 三个典型函数 65

3.1.5 函数递归 69

3.1.6 常用函数 70

3.2 模块 73

3.2.1 Python模块概述 73

3.2.2 自定义模块 75

3.2.3 Python常用模块 76

3.3 本章小结 80

3.4 思考和练习 80

第4章 常见数据结构 81

4.1 字符串 81

4.1.1 字符串概述 81

4.1.2 字符串常见函数及方法 87

4.1.3 字符串应用举例 89

4.2 元组 94

4.2.1 元组概述 94

4.2.2 元组常用函数和方法 95

4.2.3 元组应用举例 96

4.3 列表 98

4.3.1 列表概述 98

4.3.2 列表常用函数和方法 99

4.3.3 列表应用举例 …… 100
4.4 集合 …… 106
4.4.1 集合概述 …… 106
4.4.2 集合常用函数和方法 …… 108
4.4.3 集合应用举例 …… 109
4.5 字典 …… 113
4.5.1 字典概述 …… 113
4.5.2 字典常用函数和方法 …… 114
4.5.3 字典应用举例 …… 115
4.6 栈和队列 …… 116
4.6.1 栈和队列概述 …… 116
4.6.2 deque常用函数 …… 118
4.6.3 栈和队列应用举例 …… 118
4.7 本章小结 …… 120
4.8 思考和练习 …… 121

第5章 迭代器与生成器 123
5.1 迭代器 …… 123
5.1.1 迭代器概述 …… 123
5.1.2 迭代器应用 …… 127
5.2 生成器 …… 128
5.2.1 生成器概述 …… 128
5.2.2 生成器的函数或方法 …… 133
5.2.3 生成器应用举例 …… 134
5.3 本章小结 …… 136
5.4 思考和练习 …… 136

第6章 面向对象程序设计 137
6.1 类和对象 …… 137
6.1.1 类和对象的概述 …… 137
6.1.2 成员变量 …… 140
6.1.3 成员方法 …… 143
6.1.4 成员增加与删除 …… 149
6.2 继承 …… 152
6.2.1 继承的含义 …… 152
6.2.2 方法的覆盖 …… 154
6.2.3 super关键字 …… 155
6.2.4 抽象类 …… 158
6.3 综合应用 …… 160
6.4 本章小结 …… 161
6.5 思考和练习 …… 161

第7章 Python异常处理机制 162
7.1 异常的含义及分类 …… 162
7.2 异常处理 …… 163
7.2.1 异常处理的含义及必要性 …… 163
7.2.2 try…except异常处理的基本结构 …… 164
7.2.3 多try…except异常处理 …… 166
7.2.4 raise抛出异常 …… 168
7.2.5 多次raise抛出异常 …… 169
7.2.6 自定义异常 …… 171
7.3 综合应用 …… 172
7.4 内置异常 …… 173
7.5 本章小结 …… 174
7.6 思考和练习 …… 174

第8章 文件和数据库操作 175
8.1 输入和输出 …… 175
8.1.1 概述 …… 175
8.1.2 os模块和shutil模块 …… 176
8.1.3 Python os.path模块 …… 177
8.1.4 文件对象操作 …… 181
8.2 数据库操作 …… 189
8.2.1 概述 …… 189
8.2.2 基本SQL语句 …… 190
8.2.3 SQLite数据库 …… 192
8.2.4 Access数据库 …… 197
8.2.5 MySQL数据库 …… 200
8.3 建立数据源操作 …… 204
8.4 本章小结 …… 207
8.5 思考和练习 …… 207

参考文献 208

第 1 章

Python 语言概述

Python语言是目前使用最为广泛的编程语言之一，它是一种解释型、面向对象，并具有动态语义的高级编程语言。

本章学习目标：

- 了解Python的发展历程
- 理解Python语言的特点
- 了解Python的应用领域
- 掌握Python开发环境的安装与设置

1.1 Python语言发展历程及特点

1.1.1 Python语言的发展历程

Python是一种非常适合初学者学习的高级编程语言，它由荷兰人Guido van Rossum于1989年在荷兰国家数学与计算机科学研究所设计，并于1991年发布了首个公开发行版本。

Python是在多种编程语言的基础上发展而来的，这些语言包括ABC、Modula-3、C、C++、Algol-68、Smalltalk、Unix shell和其他脚本语言等。与Perl语言一样，Python源代码同样遵循GPL(GNU通用公共许可)协议。目前，Python由一个核心开发团队进行维护，但Guido van Rossum 仍在其中发挥着至关重要的指导作用。

Python语言由荷兰人Guido van Rossum于1989年发明。在为ABC语言编写插件的过程中，他产生了创建一种简洁而实用的编程语言的想法，并开始着手实现。由于他对Monty Python喜剧团的喜爱，因此将其命名为Python。

1994年1月，Guido van Rossum发布了Python 1.0版本，包含lambda、map、filter和reduce等功能。2000年10月16日，Python 2.0版本发布，加入了内存回收机制，为现代Python语言框架奠定了基础。随后，在Python 2.0基础上增加了一系列功能，并发布了多个Python 2.x版本。例

如，2004年11月30日，发布了Python 2.4，增加了流行的Web框架Django等。Python 2.7是Python 2.x版本系列的最后一个版本。相较于Python的早期版本，Python 3.0进行了重大升级，但在设计时并未考虑向下兼容。因此，新的Python程序建议使用Python 3.x系列版本的语法，除非运行环境无法安装Python 3.x，或程序本身使用了不支持Python 3.x系列的第三方库。需要注意的是，Python 3.5及以上版本无法在Windows XP及更早的操作系统上运行，而Python 3.9及以上版本无法在Windows 7及更早的操作系统上运行。截至本书编写时，Python最新版本为Python 3.13.0。有关Python版本发展的详细信息，可访问相关网站(https://www.python.org)获取。

如今，Python已成为一种知名度高、影响力大、应用广泛的主流编程语言。在电影制作、搜索引擎开发、游戏开发等领域，Python都扮演着重要的角色。未来很长的一段时间内，Python会继续增强其功能，扩大用户群体，从而巩固其在编程语言中的重要地位。

近年来，Python发展迅速。根据KDnuggets的最新调查，Python生态系统已经超过了R语言，成为数据分析、数据科学与机器学习的第一大语言。

1.1.2 Python语言的特点

Python是一种解释型、面向对象、具有动态语义的高级编程语言。它具有以下几个特点。

1. 简洁易学

Python设计简洁、易于上手，其面向对象的特性使得编程更加清晰。用户可以更专注于解决问题而不必沉迷于细节，从而避免了很多重复工作。Python拥有相对较少的关键字和简单的结构，配合明确的语法，使学习过程更加简单。

2. 运行速度相对较快

Python的底层以及许多标准库是使用C语言编写的，因此其运行速度相对较快。

3. 解释运行，灵活性强

作为一门解释型语言，Python不需要像C语言那样预先编译，这使得它的灵活性更强。解释运行的特性使得使用 Python 更加简单，也更便于将Python程序从一个平台移植到另一个平台。

4. 免费开源，扩展性强

Python是一种免费和开源的编程语言，这一点至关重要，有助于扩大其用户群体。随着用户数量的增加，Python的功能也日益丰富。用户可以自由地发布软件的副本、查看源代码、进行修改，并将其部分内容用于新的自由软件项目。这实际上形成了一种良性循环。

5. 丰富的类库与良好的移植性

Python拥有丰富的库，并且可移植性非常强，可以与C和C++等语言配合使用。这样的特性使其能胜任很多工作，例如数据处理和图形处理等。

6. 面向对象

与其他主要的语言如C++和Java相比，Python以一种既强大又简单的方式实现了面向对象编程。

1.1.3　Python语言的应用

近年来，Python发展迅速，稳居TIOBE官网发布的编程语言排行榜的前三名。Python是目前大学教学中最常用的语言，在统计领域更是排名第一。在许多软件开发领域，包括脚本和进程自动化、网站开发以及通用应用程序等，Python越来越受欢迎。随着人工智能的发展，Python已成为机器学习的首选语言。

Python的主要应用领域如下。

- 云计算：Python是云计算领域最热门的语言，典型应用如OpenStack。
- Web开发：Python拥有许多优秀的Web框架，许多大型网站都是使用Python开发的，例如YouTube、Dropbox和豆瓣网等。
- 人工智能和科学计算：Python在人工智能领域内的机器学习、神经网络、深度学习等方面占据主流地位。Python擅长进行科学计算和数据分析，支持各种数学运算，可以绘制高质量的2D和3D图像。
- 系统操作和维护：Python是系统操作和维护人员的基本编程语言。
- 金融定量交易和金融分析：在金融工程领域，Python是使用广泛的编程语言之一，其重要性逐年上升。
- 图形GUI：Python通过PyQT，WXPython，TkInter等模块提供了丰富的图形功能。

Python在公司和政府机构中的具体应用如下。

- Google：Google的应用程序引擎和相关代码(如Googl.com、Google爬虫、Google广告)广泛使用Python。
- CIA：美国中央情报局(CIA)的网站是用Python开发的。
- NASA：美国国家航空航天局(NASA)广泛使用Python进行数据分析和计算。
- YouTube：世界上最大的视频网站YouTube是用Python开发的。
- Dropbox：美国最大的在线云存储网站完全基于Python实现，每天处理超过10亿次文件上传和下载。
- Instagram：Instagram是美国最大的照片共享社交网站，每天有3000多万张照片被共享，这些功能均使用Python开发。
- Facebook：Facebook许多基本库是通过Python实现的。
- Redhat：Linux发行版中的Yum包管理工具是用Python开发的。
- 豆瓣：豆瓣的业务都是通过Python开发的。
- 知乎：中国最大的问答社区是通过Python开发的。

除此之外，还有搜狐、金山、腾讯、盛大、网易、百度、阿里、淘宝、土豆、新浪、果壳等公司也在大量使用Python来完成各种任务。

1.2　Python开发环境配置

1.2.1　Python开发环境

1. 默认编程环境

在安装Python软件后，系统会自动安装一个默认编程环境——IDLE。初学者可直接利用这

个环境进行学习和程序开发，无须额外安装其他软件。

需要注意的是，由于Python 3.9及以上版本无法在Windows 7及更早版本的操作系统上运行，本书将基于Python 3.8.10介绍相关知识(所有程序均已在该版本下调试通过)。

2. 其他常用开发环境

除了IDLE，以下是一些常用的开发环境：

- Eclipse+PyDev
- PyCharm
- Wing IDE
- Eric
- PythonWin
- Anaconda3(内含Jupyter和Spyder)
- zwPython

其中，应用比较广泛的是Anaconda3。Anaconda3是一个开源的Python发行版本，其包含了conda、Python等180多个科学计算包及其依赖项，基本能满足用户的开发需求，无须在开发中考虑是否需要安装相应的模块。感兴趣的用户可以通过官方网站(https://www.anaconda.com/download)下载相关软件。

1.2.2 Python安装

Python 3.8.10的安装过程如下。

(1) 从网站(www.python.org)下载Python 3.8.10安装程序(可以选择python-3.8.10-amd64.exe或python-3.8.10.exe)，然后双击安装该程序。此时，系统将打开一个对话框，如图1-1所示。

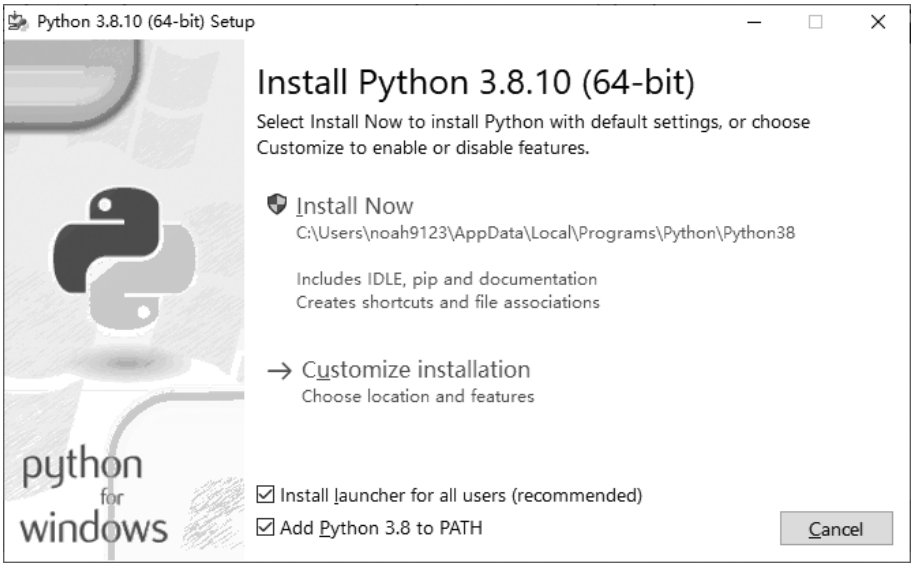

图 1-1　Python 3.8.10(64-bit)Setup 对话框

(2) 在图1-1所示对话框中选中所有复选框后，单击【Customize installation】选项，打开图1-2所示的对话框。

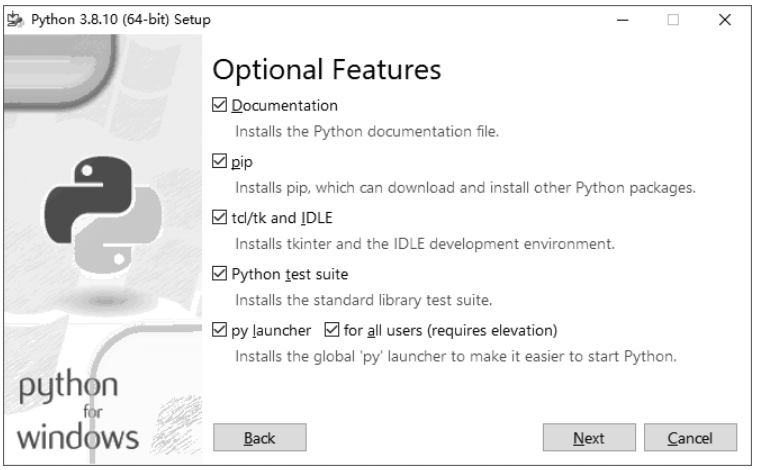

图 1-2　Optional Features 对话框

(3) 在图1-2所示对话框中选中所有复选框后，单击【next】按钮，打开如图1-3所示的对话框。

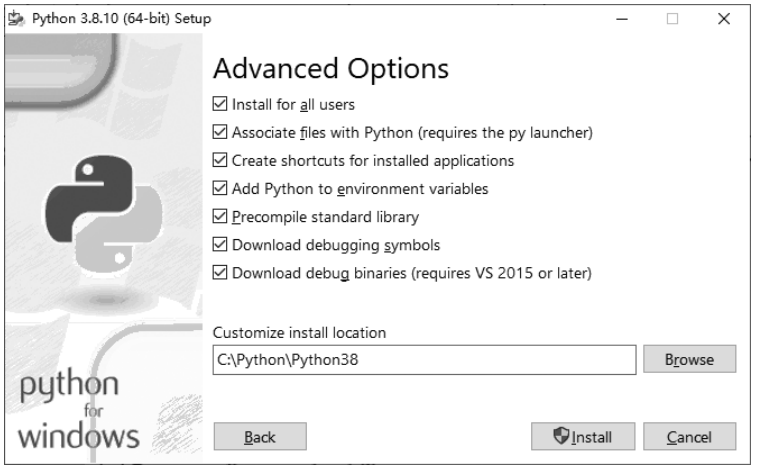

图 1-3　Advanced Options 对话框

(4) 在图1-4所示对话框中选中所有复选框后，单击【Browse】按钮。在打开的对话框中将安装路径设置为C:\ Python \Python38，然后单击【Install】按钮开始安装Python，如图1-4所示。

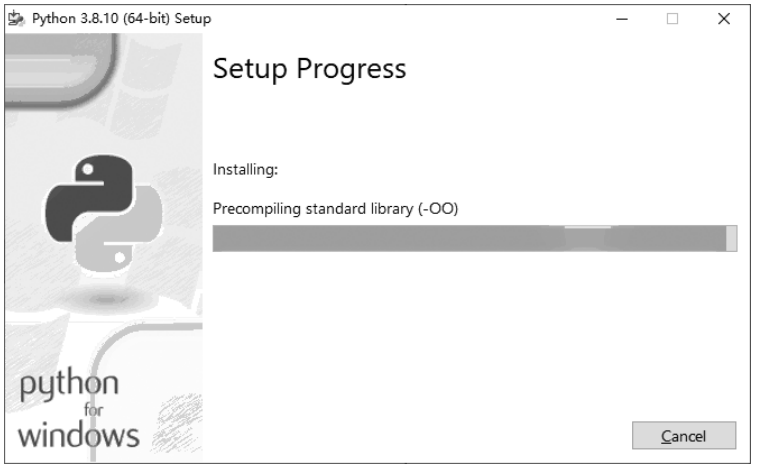

图 1-4　Setup Progress 对话框

(5) Python安装完毕后，将打开图1-5所示的对话框，单击【Close】按钮。

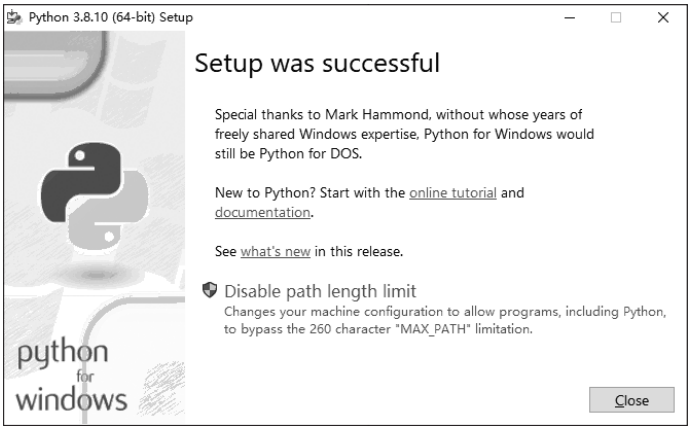

图 1-5　Setup was successful 对话框

完成以上操作后，Python系统目录的结构将如图1-6所示。

名称	修改日期	类型	大小
DLLs	2023/9/10 8:59	文件夹	
Doc	2023/9/10 8:59	文件夹	
include	2023/9/10 8:58	文件夹	
Lib	2023/9/10 8:59	文件夹	
libs	2023/9/10 8:59	文件夹	
Scripts	2023/12/14 9:27	文件夹	
tcl	2023/9/10 8:59	文件夹	
Tools	2023/9/10 8:59	文件夹	
LICENSE.txt	2021/5/3 11:54	文本文档	32 KB
NEWS.txt	2021/5/3 11:55	文本文档	934 KB
python.exe	2021/5/3 11:54	应用程序	100 KB
python3.dll	2021/5/3 11:54	应用程序扩展	59 KB
python38.dll	2021/5/3 11:54	应用程序扩展	4,113 KB
pythonw.exe	2021/5/3 11:54	应用程序	98 KB
vcruntime140.dll	2021/5/3 11:54	应用程序扩展	94 KB
vcruntime140_1.dll	2021/5/3 11:54	应用程序扩展	36 KB

图 1-6　Python 系统目录结构

以下是关于Python系统目录中各个文件和子目录的说明。

(1) python.exe：python.exe文件是一个可执行文件，在命令行(cmd)下执行python.exe会打开一个Windows命令行窗口。

(2) DLLs目录：该目录包含Python的*.pyd(Python动态模块)文件与一些Windows的*.dll(动态链接库)文件。

(3) Doc目录：该目录存放文档。在Windows平台上，只有一个python3810.chm文件，其中集成了Python的所有文档，双击即可打开阅读。

(4) include目录：该目录包含Python的C语言接口头文件，当在C程序中集成Python时，会用到这个目录下的头文件。

(5) lib目录：该目录包含Python的标准库、包、测试套件等。

(6) libs目录：该目录存放Python的C语言接口库文件。

(7) Scripts目录：该目录中包含pip可执行文件，通过pip可以安装各种Python扩展包(这也是该目录需要被添加到PATH环境变量中的原因)。

(8) tcl目录：该目录中包含桌面编程所需的工具包。

(9) Tools目录：该目录包含一系列工具，目录中的README.txt文件详细说明了各工具的用途。

1.2.3　环境变量设置

在Python语言的应用中，有两个非常重要的环境变量：PATH和PYTHONPATH。

PATH用于指定系统安装位置和系统内置模块的位置。如果在安装系统时选中了“Add Python 3.8.10 to Path”复选框，则该环境变量会自动设置。否则就需要使用手动方式来设置PATH，主要是将系统安装位置和该位置下的Scripts子目录添加到系统的PATH环境变量中。

PYTHONPATH用于指定用户自定义模块或第三方模块的位置，以便让Python解释器能够找到这些模块。如果没有将所需模块的路径添加到PYTHONPATH，在导入该模块时可能会出现“找不到该模块”的错误。因此，必须把所需模块的路径添加到 PYTHONPATH。

环境变量的设置过程基本相似，以下将以Windows 10为例说明如何设置环境变量PYTHONPATH，而环境变量PATH的设置可参照此过程。

设置环境变量PYTHONPATH的步骤如下。

(1) 右击Windows 10系统桌面上的【计算机】图标，弹出图1-7所示的快捷菜单。

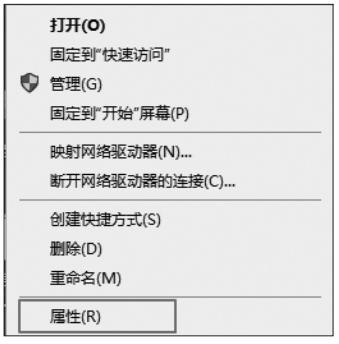

图 1-7　快捷菜单

(2) 在图1-7所示的快捷菜单中选择【属性】命令，打开图1-8所示的窗口。

图 1-8　【控制面板 \ 系统和安全 \ 系统】窗口

(3) 在图1-8所示的窗口中选择【高级系统设置】选项，打开如图1-9所示的【系统属性】对话框。

图 1-9　【系统属性】对话框

(4) 在图1-9所示的【系统属性】对话框中单击【环境变量】按钮，打开如图1-10所示的【环境变量】对话框。

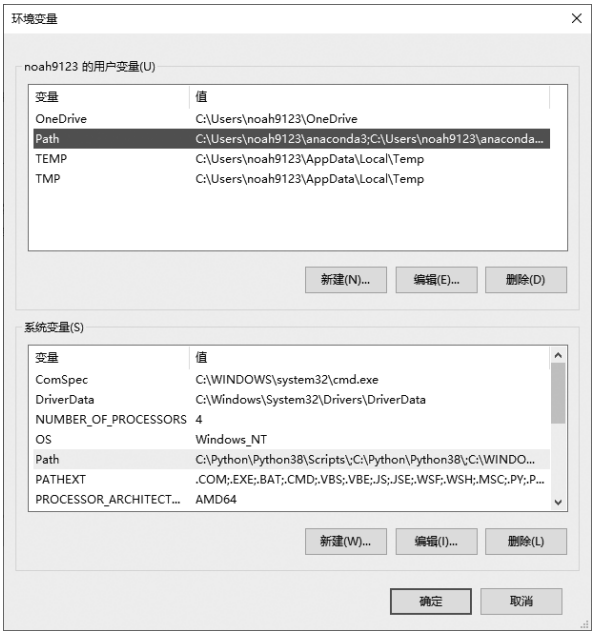

图 1-10　【环境变量】对话框

(5) 在图1-10所示的【环境变量】对话框的【用户变量】区域中单击【新建】按钮，打开【新建系统变量】对话框添加PYTHONPATH环境变量，如图1-11所示。

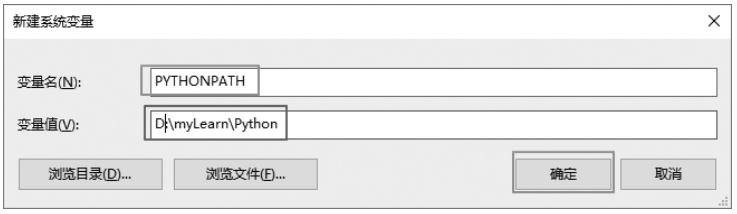

图 1-11　新建系统变量

(6) 在图1-11所示的【新建系统变量】对话框【变量名】文本框中输入PYTHONPATH，在【变量值】文本框中输入“D:\mylearn\Python”，然后单击【确定】按钮。此时，将打开图1-12所示的【环境变量】对话框。

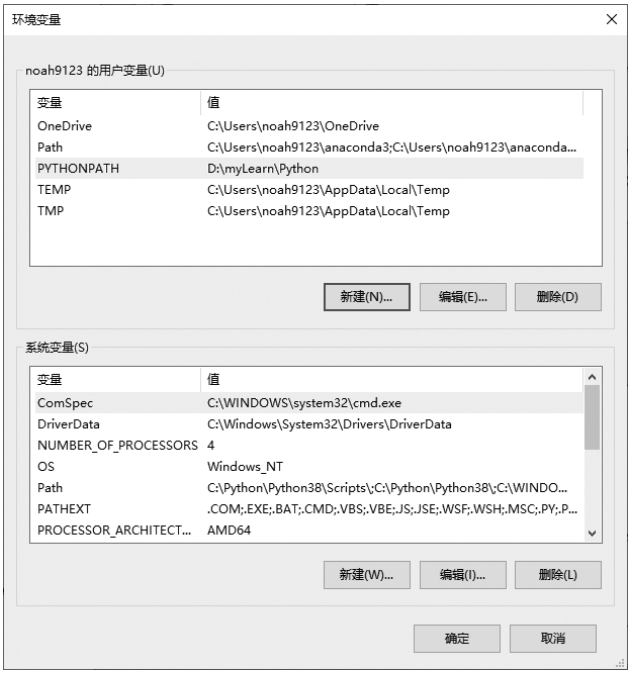

图 1-12　【环境变量】对话框

从图1-12可以看出，已成功添加一个新的环境变量PYTHONPATH。需要指出的是，如果发现某个环境变量的值需要修改，可以单击【编辑】按钮进行修改；如果不再需要某个环境变量，可以单击【删除】按钮将其移除。本文开发模块均在D:\mylearn\Python目录下，因此这里设置PYTHONPATH的值为D:\mylearn\Python(用户可以根据自己的实际情况设置该值)。

❖ 提示：

在Windows 2000和Windows XP系统中设置环境变量的步骤如下。

(1) 右击系统桌面上的【我的电脑】图标。

(2) 在弹出的快捷菜单中选择【属性】命令。

(3) 在打开的窗口中选择【高级】选项卡。

(4) 在【高级】选项卡中单击【环境变量】按钮。

说明：

在Windows系统中，存在两种环境变量：用户变量和系统变量。这两种环境变量可以存在同名的变量。用户变量只对当前用户有效，而系统变量则对所有用户有效。

Windows系统执行用户命令时，若用户未提供文件的绝对路径，系统会首先在当前目录下查找相应的可执行文件或批处理文件。若找不到，系统会依次在系统变量的PATH中指定的路径中寻找相应的可执行程序文件(查找顺序是从左往右，最前面的路径优先级最高，一旦找到命令，后续路径将不会再继续查找)。如果还找不到，系统会在用户变量的PATH路径中查找。如果系统变量和用户变量的PATH中都包含了某个命令，则优先执行系统变量PATH中的命令。

在Windows系统中，用户变量和系统变量(如PATH)的名称不区分大小写，因此设置Path和PATH是没有区别的。

1.2.4 用户模块文件管理

Python语言是一种解释型语言，为了让解释器python.exe能够正确解析用户模块文件，必须对这些模块文件进行适当的管理，以便解释器能够识别，并执行它们。管理用户模块文件的方式通常有以下三种。

1. 将用户的模块文件(如abc.py)放置在\lib\site-packages目录下

\lib\site-packages子目录位于Python的安装位置中。当PATH环境变量包含Python的安装位置时，位于\lib\site-packages子目录下的模块文件可以被Python解释器Python.exe导入并执行。

然而，如果把模块文件都放在此目录下，可能会导致模块文件混乱，甚至可能损坏某些模块文件。因此，一般不建议采取这种方式。

2. 使用.pth文件

在 site-packages目录中创建.pth文件，将模块的路径逐行写入，每行指定一个路径。以下是一个示例，其中.pth文件也可以使用注释：

```
# .pth file for the my project(这行是注释)
D:\myLearn\python
D:\myLearn\python\mysite
D:\myLearn\python\mysite\polls
```

这种方法是一个不错的选择，但存在管理上的问题，并且无法在不同的Python版本之间共享。

3. 使用PYTHONPATH环境变量

PYTHONPATH是Python中一个重要的环境变量，用于指定导入模块时的搜索路径。可以通过以下方式访问。

(1) 暂时设置模块的搜索路径(修改sys.path)。

导入模块时，Python会在指定的路径下搜索相应的.py文件，这些搜索路径存放在sys模块的sys.path变量中。可以通过以下方式查看sys.path变量的内容。

```
>>>import sys
>>>sys.path
PATH=C:\Python\Python38\Scripts\; C:\Python\Python38\ ;C:\WINDOWS\system32; C:\
WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\
WINDOWS\System32\OpenSSH\ ;C:\Users\dellpc\AppData\Local\Microsoft\WindowsApps;
```

可以通过append函数在sys.path后添加搜索路径。例如，若要导入的第三方模块位于D:\myLearn\python，可以在Python解释器中添加sys.path.append('D:\\myLearn\\python')。

需要注意的是，这种方法只是暂时性的，下次再进入交互模式时需要重新设置。

(2) 永久设置模块的搜索路径。

设置PYTHONPATH环境变量。关于如何设置PYTHONPATH的方法在前面已经介绍，在此不再赘述。

1.3　Python的使用

1.3.1　命令行方式

在这种方式下，首先使用编辑器(如EditPlus)编辑生成模块文件，然后在DOS提示符下使用Python的python.exe命令及有关参数来运行模块文件。下面将介绍如何在命令行方式下使用Python。

(1) 按下Windows+R键(即Win+R)，打开【运行】对话框，如图1-13所示。

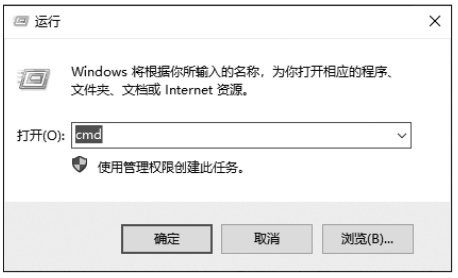

图 1-13　【运行】对话框

(2) 在图1-13所示的【运行】对话框中输入cmd后，单击【确定】按钮，打开图1-14所示的命令窗口(通常称为DOS窗口)。

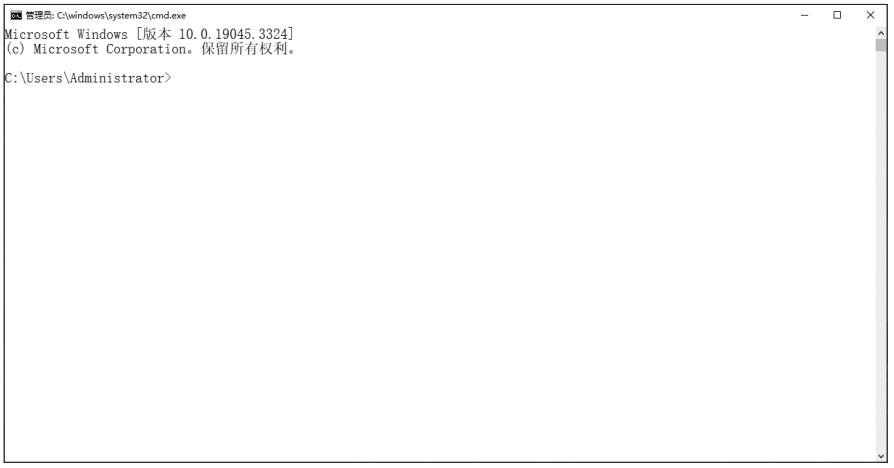

图 1-14　命令窗口

(3) 在图1-14所示命令窗口的提示符后可以输入各种DOS命令(如切换盘符、进入工作目录

等)。例如，将盘符切换到E盘并进入工作目录D:\myLearn\python后，命令窗口将如图1-15所示。

图 1-15　操作后的命令窗口

此时，在图1-15所示窗口中，用户可以在DOS提示符“D:\myLearn\python”后输入Python的python.exe命令及相关参数来运行模块文件。例如：

```
D:\myLearn\python>python PBT01.py
```

其中，PBT01.py为用户建立的模块文件。该模块文件可使用EditPlus进行编辑。本文所有的模块文件均在EditPlus中编辑，并通过命令行使用python.exe解释器运行。

建议初学者使用EditPlus编辑模块文件(无论包含多少命令)，然后在命令行下使用python.exe解释器进行解释和运行。

PBT01.py的内容为：

```
print('Hello Python!')
```

Python的命令行参数详见表1-1。

表1-1　Python命令行参数

选项	描述
-d	在解析时显示调试信息
-O	生成优化代码 (.pyo文件)
-S	启动时不引入查找Python路径的位置
-V	输出Python版本号
-c cmd	执行 Python 脚本，并将运行结果作为 cmd 字符串
file	执行指定的Python文件

1.3.2　IDLE方式

在该方式中，首先需要启动IDLE(Python 3.8.10)。启动IDLE后将打开图1-16所示的Shell窗口。

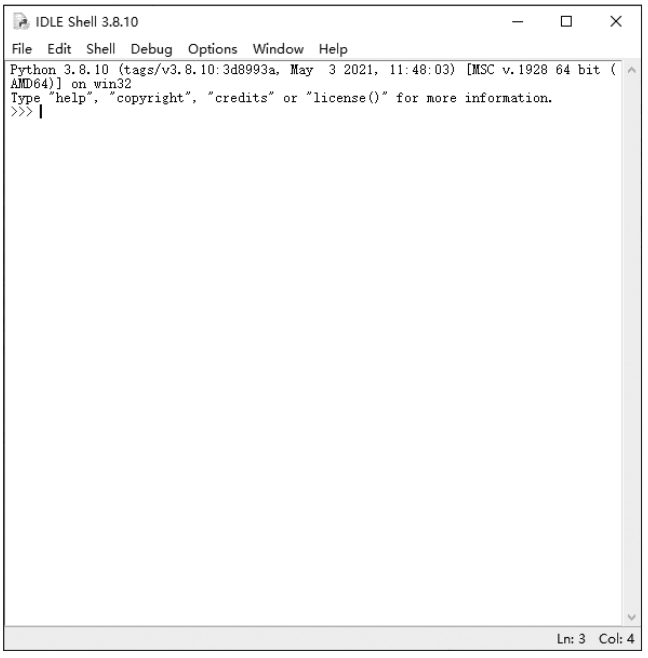

图 1-16　Shell 窗口

在Shell窗口的“>>>”提示符下可以输入各种命令，并且在输入过程中可以随时进行保存。

除了在“>>>”下输入各种命令外，用户也可将命令编辑成一个模块文件(.py文件)。可以通过File菜单中的New File命令来创建新的文件，或使用Open命令打开已编辑的文件。当使用New File命令时，将会打开图1-17所示窗口。

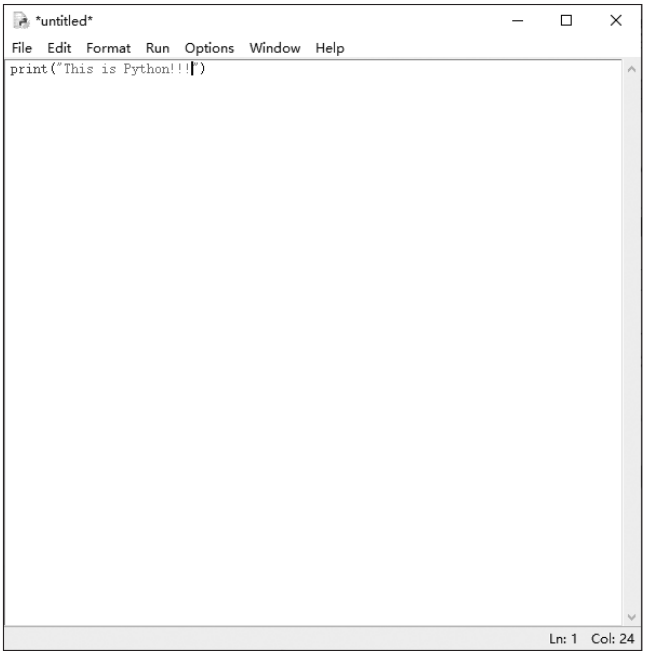

图 1-17　模块文件编辑窗口

在图1-17所示窗口中，用户可以编辑模块文件。当然，模块文件也可以使用任何其他编辑工具(包括EditPlus)进行编辑。

模块文件编辑完成后，用户可以通过Run菜单中的Run Module命令来运行该模块文件。

1.3.3 Spyder方式

使用Anaconda3内置的Spyder进行程序开发。启动Spyder后，将打开如图1-18所示窗口。

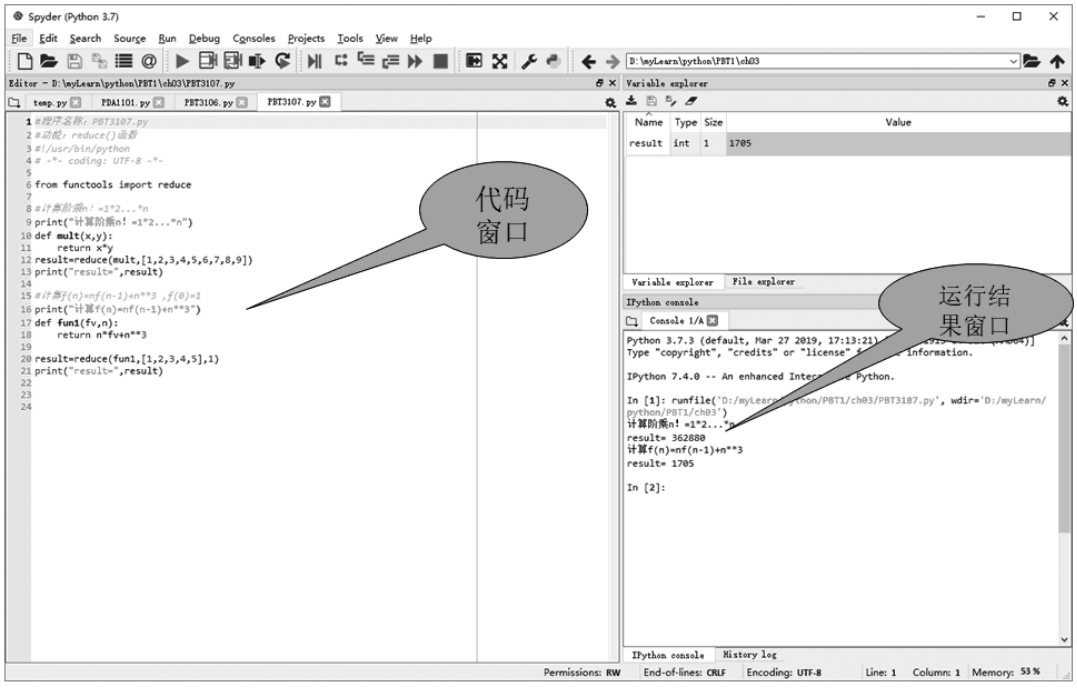

图 1-18　Spyder 窗口

图1-18所示窗口的左侧是代码窗格，右下方是运行结果窗格。总体而言，使用Spyder进行开发相对方便，但由于该软件体积较大，在计算机系统配置较低的情况下，启动速度可能较慢。

1.4　本章小结

本章主要介绍了Python语言的发展历程、核心特点、开发环境的安装与配置、用户模块文件的管理，以及Python使用的三种常见方式等内容。

1.5　思考和练习

1. 简述Python语言的发展历程。
2. 简述Python语言的特点。
3. 简述如何设置PYTHONPATH环境变量。
4. 安装Python3.8.10软件。编写一个简单程序，并利用解释器运行。

第 2 章

Python 语言基础

Python语言是在其他编程语言(如C语言)的基础上发展而来的，因此与其他语言有许多相似之处(例如循环结构和判断结构等)。然而，作为一门语言，Python语言也有其自身的特点。在学习Python语言时，用户可以通过对比其他语言，更加深入地理解Python的独特之处。

本章学习目标：

- 了解Python程序的基本语法
- 掌握Python语言的变量与数据类型
- 掌握Python语言的常见运算符
- 掌握Python语言的条件控制与循环语句

2.1 Python基础语法

2.1.1 Python程序基本框架

Python语言的程序由一系列函数、类和语句组成，其基本框架如图2-1所示。

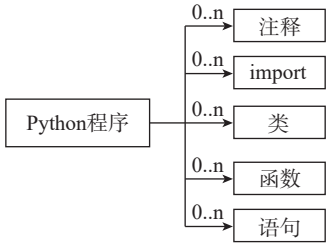

图 2-1 Python 程序框架 (一般性)

Python程序是顺序执行的。在Python中，首先执行最先出现的非函数定义和非类定义的未缩进代码。C语言是从main( )函数开始执行的，因此，为了保持与C语言的习惯相同，建议在Python程序中增加一个main( )函数，并将对main( )函数的调用作为最先出现的非函数定义和非类

定义的未缩进代码。这样，Python程序便可从main( )作为执行的起点。建议按照图2-2所示的框架来编写Python程序。

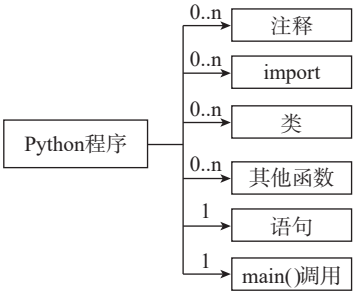

图 2-2　Python 程序框架 (建议性)

以下举例说明。

Python程序	说明
`# 程序名称：ppb2100.py`	注释
`# 程序功能：展示程序框架`	注释
`def sum(x,y):`	函数定义
`    return x+y`	函数内语句
`def main():`	函数定义
`    x=1`	函数内语句
`    y=2`	函数内语句
`    print("sum=",sum(x,y))`	函数内语句(含函数调用)
`main()`	函数调用

本书中绝大部分程序都是按照这种框架进行组织的。需要强调的是，提出这种建议框架的目的是为了帮助初学者养成规范组织代码的习惯。当学习Python到一定程度后，读者会形成更合理、更规范且适合自身的代码组织习惯。

2.1.2　Python编码

1. 编码简介

ASCII编码是由美国信息交换标准委员会(ANSI)于1963年制定的一种字符编码标准，广泛应用于计算机领域。ASCII码只能表示128个字符，包括英文字母、数字、标点符号和一些控制字符。每个字符用7个比特位表示，因此ASCII编码的每个字符只需1个字节来存储。

Unicode是一种字符编码标准，它可以表示世界上几乎所有的字符，包括汉字、日文、韩文等。Unicode为每个字符分配一个唯一的编码值，这些编码值可以用16位或32位表示。

在Python中，字符串以Unicode形式表示，可以表示任意字符。然而，由于计算机硬件只能存储二进制数据，因此在实际应用中，需要将Unicode字符编码为二进制数据，或将二进制数据解码为Unicode字符。

在Python中，常见的中文字符编码方式包括UTF-8、GBK和GB2312等。其中，UTF-8是

一种可变长度的Unicode字符码方式，可以将Unicode字符集中的所有字符编码成一个或多个字节。在UTF-8编码中，ASCII字符使用一个字节表示，而非ASCII字符则使用两个、三个或四个字节表示。UTF-8编码广泛应用于互联网和计算机系统中，被认为是一种通用的字符编码方式。GBK和GB2312是两种常见的中文字符编码方式。在这两种编码方式中，一个中文字符通常使用两个字节表示，但部分冷僻字符需要三个字节或四个字节表示。GBK是GB2312的扩展版本，支持更多中文字符，包括繁体中文和部分外来语。

2. 编码转换

不同编码之间不能直接转换，要借助Unicode编码实现间接转换。例如，将GBK编码转换为UTF-8编码的流程为：首先通过decode()函数将其转换为Unicode编码，然后再使用encode()函数将Unicode编码转换为UFT-8编码。类似地，将UTF-8编码转换为GBK编码格式流程为：首先通过decode()函数转换为Unicode编码，然后使用encode()函数将其转换为GBK编码，如图2-3所示。

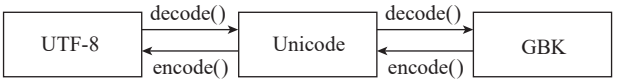

图 2-3　UTF-8 和 GBK 编码之间的转换

Python中提供了两个实用的函数：decode( )和encode( )。

将UTF-8编码转换为GBK编码的过程如下：

```
>>>decode('UTF-8')        # 将UTF-8编码转换成Unicode编码
>>>encode('GBK')          # 将Unicode编码转换成GBK编码
```

将GBK编码转换为UTF-8编码的过程如下：

```
>>>decode('GBK')          # 将GBK编码转换成Unicode编码
>>>encode('UTF-8')        # 将unicode编码转换成UTF-8编码
```

2.1.3　Python注释

Python中的注释分为单行注释和多行注释。

1. 单行注释

单行注释以 # 开头，例如：

```
# 这是一个单行注释
print("Hello, World!")
```

2. 多行注释

多行注释用两个三引号 (单三引号''' 或者双三引号 """) 将注释括起来，例如：

```
# !/usr/bin/python3
'''
这是多行注释，用三个单引号
这是多行注释，用三个单引号
'''
print("Hello, World!")
```

或

```
# !/usr/bin/python3
"""
这是多行注释，用三个双引号
```

```
这是多行注释，用三个双引号
"""
print("Hello, World!")
```

2.1.4 行与缩进

1. 缩进

在Python中，缩进是一个非常重要的概念。与Java和C等语言不同的是，Python使用缩进来指示代码块的层次结构，而不是使用大括号{}。缩进有助于更好地组织和管理代码，控制程序的流程，提高代码的可读性和可维护性。正确使用缩进可以避免语法错误，使代码更加规范和易读。

缩进的空格数是可变的，但同一个代码块的语句必须使用相同数量的缩进空格。一般建议使用Tab键来控制缩进。例如：

```
if True:
    print("This is True")
else:
    print("This is False")
```

下面的例子说明了空格缩进不一致时，会导致运行错误。

【实例2-1】

```
# 程序名称：ppb2101.py
# 程序功能：展示缩进不一致的错误
if True:
    print("This  is  ")  # L1
    print("True")        # L2
else:
    print("This  is  ")  # L3
  print("False")         # L4缩进不一致，会导致运行错误
```

以上程序由于缩进不一致，执行后会出现如下错误：

```
File"ppb2101.py",line 8
    print("False")       # L4
                                        ^
IndentationError: unindent does not match any outer indentation level
```

说明：

在程序ppb2101.py中，L1、L2、L3和L4所对应的行属于同一层次，因此缩进空格数应保持一致，行首字母应纵向对齐，否则在运行时会出错。

2. 多行语句

若语句较长，可以使用反斜杠(\)来实现多行语句，例如：

```
total=item_one+\
        item_two+\
        item_three
```

此外，也可将多行语句放在括号(例如[]、{}或())中，而不使用反斜杠(\)，例如：

```
total=['item_one','item_two','item_three',
        'item_four','item_five']
```

2.1.5　常用的函数和语句

1. input( )函数

input( )是一个内置函数，用于从标准输入读取一行文本。默认情况下，标准输入是键盘。示例代码如下：

```
s=input("请输入：");
print("你输入的内容是：", s)
```

说明：

input( )函数的返回值是字符串，可利用int( )、float( )等函数将数字型字符串转换为对应的数字。例如，int("123")的结果为123，而float("123.12")的结果为123.12。

有关这类转换函数的详细介绍将在后面的章节中讨论。

2. print( )函数

print( )是一个内置函数，用于将特定对象(如Number型数字，字符串等)输出到屏幕上。例如：

```
print("Hello Python! ")
```

输出结果为：

```
Hello Python!
```

输出多项内容时，各项之间用逗号(,)隔开。例如：

```
s="good"
i=100
print("Hello Python! ",s,"i=",i)
```

输出结果为：

```
Hello Python! good  i=100
```

默认情况下，print( )会在输出后换行。若希望不换行，可以在输出内容末尾添加end=字符串。例如：

```
# 不换行输出
print("This-is ", end="-")
print("Python")
```

输出结果为：

```
This-is-Python
```

❖ 特别提示：

灵活运用print语句，可以将输出内容以一种整洁且易于阅读的形式输出到屏幕上。例如：

```
print(" ┌──────┬──────┐ ")
print(" │ 学号 │ 姓名 │ ")
print(" ├──────┼──────┤ ")
print(" │ 1001 │ 张三 │ ")
print(" └──────┴──────┘ ")
```

执行后输出结果为：

```
┌──────┬──────┐
│ 学号 │ 姓名 │
├──────┼──────┤
│ 1001 │ 张三 │
└──────┴──────┘
```

3. pass语句

pass是Python中一个特殊的语句，它是一个空语句，不执行任何操作。通常用于占位，以保持程序结构的完整性。pass语句主要出现在函数、类、控制语句和循环语句中，当相关功能尚未实现时，可以使用pass占位。下面分别举例说明。

(1) 在函数中使用pass。例如：

```
def  fun():
    pass  # 占位作用
```

在这个示例中，函数fun()的功能暂时未实现，因此使用pass占位。如果去掉pass，将导致语法错误。

(2) 在类中使用pass。例如：

```
class myClass:
    pass  # 占位作用
```

在这个示例中，类myClass的功能暂时未实现，因此使用pass占位。如果去掉pass，将导致语法错误。

(3) 在控制语句中使用pass。例如：

```
if a>1:
    pass  # 控制作用
else
    print("a<=1")
```

在这个示例中，if后的语句块暂时未确定，因此可以用pass占位。如果去掉pass，将导致语法错误。

(4) 在循环语句中使用pass。例如：

```
for i in range(10):
    pass
```

在这个示例中，循环体中的语句块暂时未确定，因此可以用pass占位。如去掉pass，将导致语法错误。

4. del语句

在Python中，一切皆为对象。通过使用del语句，可以删除任何对象。例如：

```
del name       # 删除某个变量
del Classname  # 删除某个类
```

❖ 提示：

由于内存空间是有限的，使用del命令删除一些不再需要的对象(无论是永久性还是临时的)是非常必要的。

5. id( )函数

在Python中，变量实际上存储的对象是内存地址。通过id( )函数，可以查看变量所指向的内存地址。例如：

```
x=10
print(id(x))  # 140703774499760
```

在这个示例中，变量x存储的是地址140703774499760，而该地址单元中存储的值为10。

6. type( )函数

在Python中存在多种类型的对象，如int型对象(即整数型对象)、str型对象(即字符串对象)等。可以使type( )函数来查看对象的类型。例如：

```
x=100
print(type(x))    # <class'int'>
s="I am String"
print(type(s))    # <class'str'>
```

2.1.6　Python关键字

Python关键字是具有特定含义和用途的字符序列，不能用于其他用途。表2-1所示为Python关键字的详细说明。

表 2-1　Python 关键字

关键字	功能描述
False	布尔类型的值，表示假，与True对应
class	定义类的关键字
finally	异常处理使用的关键字，用于指定始终执行的代码
is	用于判断两个变量的指向是否完全一致，内容和地址完全一致，返回True，否则返回False
return	Python函数返回值，函数中一定要有return返回值才是完整的函数。如果没有定义return，则会返回None
None	特殊常量，表示没有值。与False不同，None不是0，也不是空字符串。None和其他任何数据类型比较时，永远返回False。None属于NoneType数据类型。可以将None复制给任何变量，但是不能创建其他NoneType对象
continue	用于告诉Python跳过当前循环块中的剩余语句，继续进行下一轮循环
for	用于遍历任何序列的项目，例如列表或字符串
lambda	用于定义匿名函数
try	程序员可以使用try…except语句来处理异常。把正常的语句块放在try块中，将错误处理语句放在except块中
True	布尔类型的值，表示真，与False相反
def	用于定义函数
from	用于通过import或from…import导入相应的模块
nonlocal	用于在函数或其他作用域中使用外层(非全局)变量
while	用于重复执行一段代码。while是循环语句的一种，while语句可以有一个可选的else从句
and	逻辑判断语句，当and左右两边都为真时，判断结果为真，否则为假
del	删除列表中不需要的对象，删除定义过的对象
global	用于定义全局标量
not	逻辑判断，表示取反
with	控制流语句，用于简化try…finally语句，主要用于实现具有_enter_()和_exit_()方法的类
as	用于给对象取新的名字，例如 with open('abc.txt') as fp、except Exception as e、import numpy as np等
elif	和if配合使用，表示“否则如果”

(续表)

关键字	功能描述
if	用于检验一个条件，如果条件为真，执行一段代码(称为if块)；否则，执行另一段代码(称为else块)。else从句是可选的
or	逻辑判断，or两边任何一个条件为真，则判断结果为真
yield	用法类似return，yield告诉程序，要求函数返回一个生成器
assert	声明某个表达式必须为真，编程中若该表达式为假，将抛出AssertionError
else	与if配合使用，表示条件不满足时执行的代码块
import	通过import或者from…import语句导入相应的模块
pass	表示什么都不做，主要用于弥补语法和空定义上的冲突
break	用于终止循环，无论循环条件是否为False或者序列还没有被完全递归，都会停止循环语句
except	与try配合使用，用于处理异常
in	判断对象是否在序列(如列表、元组等)中
raise	用于抛出异常
async	用于声明一个函数为异步函数，异步函数可以在执行过程中挂起，以便执行其他异步操作
await	用于声明程序挂起。当异步程序执行到某一步时，如果需要等待较长的时间，可以使用await挂起当前操作，转而执行其他异步程序

说明：

- True、False和None的首字母T、F和N必须大写，其余字母均为小写。
- 可以查看所有关键字。例如：

```
import keyword
print(keyword.kwlist)   # 输出关键字列表
```

2.1.7 Python标识符

标识符用于标识包名、类名、变量名、模块名以及文件名等有效字符序列。Python语言规定标识符由字母、下画线和数字组成，并且第一个字符不能是数字。例如，在字符序列中：3max、class、room#、userName和User_name中，3max、room#、class不能作为标识符，因为3max以数字开头，room#包含非法字符“#”。此外，class是一个保留关键字，不能用作标识符。需要注意的是，标识符中的字母是区分大小写的，例如Radius和radius表示不同的标识符。

标识符应遵循以下命名规则。

(1) 标识符尽量采用有意义的字符序列，以便于从中识别出所代表的基本含义。

(2) 包名采用全小写的名词形式。

(3) 类名的首字母大写，通常由多个单词组合而成，每个单词的首字母也应大写，例如class HelloWorldApp。

(4) 接口名的命名规则与类名相同，例如interface Collection。

(5) 方法和函数名通常由多个单词组成，第一个单词通常为小写动词，后续每个单词的首字母应大写，例如balanceAccount和isButtonPressed。

(6) 变量名采用全小写形式，一般为名词，例如使用area表示面积变量，length表示长度变量等。

(7) 不建议使用系统内置的模块名、类型名或函数名，以及已导入的模块名及其成员名作为变量名，因为这可能会改变其类型和含义。可以通过dir(__builtins__)查看所有内置模块、类型和函数。

2.2　变量与数据类型

2.2.1　变量

1. 变量概述

与Java和C等语言不同，Python不需要事先声明变量名及其类型。Python中的变量是通过赋值来创建的。换言之，每个变量在使用前都必须先赋值，只有在赋值后，该变量才会被创建。

```
score=95          # 赋值整型变量
area=86.76        # 赋值浮点型变量
name="吴二"       # 赋值字符串变量
```

在以上示例中，95、86.76和"吴二"分别被赋值给变量score、area和name。

需要注意的是，Python中的变量并不直接存储值，而是存储了值的内存地址或引用。例如，赋值语句score=95的执行过程是：首先会在内存中分配空间以存放值95，然后创建变量score指向这个内存地址，如图2-4所示。

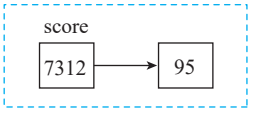

图 2-4　赋值语句

说明：

(1) 如果赋值语句右边是一个表达式，首先计算该表达式的值，然后在内存中分配空间以存放该值。

(2) Python允许同时为多个变量赋值。例如：

```
score1=score2=score3=95
```

在以上示例中，三个变量score1、score2和score3都指向同一块内存空间(即存储95的空间)，如图2-5所示。

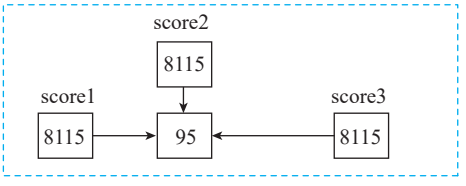

图 2-5　多变量赋值

【实例2-2】

```
# 程序名称：ppb2102.py
# 程序功能：展示多变量赋值
def main():
    score1=score2=score3=85
    print("score1的地址=",id(score1))
```