安徽省高水平教材建设项目
计算机软件与新技术人才培养系列教材

Python 程序设计实践教程

王鹤琴 张进军 沈 洋／主编

中国科学技术出版社
CHINA SCIENCE AND TECHNOLOGY PRESS

计算机软件与新技术人才培养系列教材

Python 程序设计实践教程

王鹤琴 张进军 沈 洋／主编

中国科学技术出版社

计算机软件与新技术人才培养系列教材

软件技术
数据结构与算法
C语言程序设计
C#程序设计
Python程序设计实践教程
Java程序设计
Java Web应用开发
Java框架开发技术
Web前端开发
Web应用系统开发
HTML5与JavaScript程序设计
HTML5+CSS3网页设计技术
Android应用开发
iOS应用开发

大数据技术
Linux操作系统基础
Spark技术与应用
Python数据分析
Python数据工程师实战案例教程
大数据基础
数据库高级管理技术
大数据集群搭建维护与数据存储
大数据采集与数据处理
大数据可视化技术应用
数据挖掘应用
智能数据分析与应用

云计算技术
云计算基础
OpenStack系统架构与部署
云存储规划与部署
网络存储规划与实施
云安全技术与应用
服务器虚拟化技术项目式教程
虚拟化技术与应用
云数据中心架构与运维
公有云运维与应用实践
云平台建设与维护实战

人工智能技术
人工智能基础
机器学习
深度学习
人工智能算法设计
人工神经网络技术
智能视觉技术
智能算法容器化部署
自然语言处理技术
智能语音应用开发

读科学 得真知
科学普及出版社漫漫读微信号

ISBN 978-7-5236-1246-0
9 787523 612460 >
定价：52.00元

安徽省高水平教材建设项目

计算机软件与新技术人才培养系列教材

Python
程序设计实践教程

王鹤琴　张进军　沈　洋／主编

中国科学技术出版社

·北　京·

图书在版编目（CIP）数据

Python 程序设计实践教程 / 王鹤琴，张进军，沈洋主编．-- 北京 ：中国科学技术出版社，2024. 12.
（计算机软件与新技术人才培养系列教材）．-- ISBN 978-7-5236-1246-0

Ⅰ．TP312.8

中国国家版本馆 CIP 数据核字第 202562BF94 号

策划编辑	王晓义
责任编辑	付晓鑫
装帧设计	唐韵设计
责任校对	焦　宁
责任印制	徐　飞

出　　版	中国科学技术出版社
发　　行	中国科学技术出版社有限公司
地　　址	北京市海淀区中关村南大街 16 号
邮　　编	100081
发行电话	010-62173865
传　　真	010-62173081
网　　址	http://www.cspbooks.com.cn

开　　本	889mm × 1194mm　1/16
字　　数	470 千字
印　　张	16
版　　次	2024 年 12 月第 1 版
印　　次	2024 年 12 月第 1 次印刷
印　　刷	北京荣玉印刷有限公司
书　　号	ISBN 978-7-5236-1246-0 / TP · 509
定　　价	52.00 元

前言

随着计算机技术的不断发展，Python 已经成为一门广泛使用的主流编程语言。它以简洁明了的语法、强大的功能和丰富的库资源，受到了广大程序员的喜爱。越来越多的高校将 Python 作为编程课程的教学语言，越来越多的企业和开发者选择 Python 进行项目开发。学习如何利用 Python 进行编程，是众多相关专业学生需要学习及掌握的基本技能。

本教材响应新时期国家对职业教育的新要求，落实立德树人根本任务，贯彻《高等学校课程思政建设指导纲要》、党的二十大精神，将专业知识与思政教育有机结合，实现价值引领、知识传授和能力培养的紧密结合。在新时代的教育背景下，立德树人被视为教育的根本任务。党的二十大报告强调了教育的重要性，提出要全面贯彻党的教育方针，培养德智体美劳全面发展的社会主义建设者和接班人。在这一指导思想下，本教材旨在在技术传授之外，融合思政元素，通过具体编程实践体现社会主义核心价值观和新时代中国精神。

本教材主要具备以下特色。

1. 融合思政元素。本教材在讲授 Python 程序设计的同时，特意嵌入了课程思政的内容。例如，在案例选择上，注重结合国家发展需求、社会问题解决方案及中华优秀传统文化的传承，引导学生在解决实际编程问题的同时，思考社会责任与文化自信，进而培养正确的价值观念。

2. 立德树人理念。本教材特别强调在知识与技能教育中培养学生的职业道德和社会责任感，将立德树人理念渗透到每个实例和项目中。在实验指导中，鼓励学生遵循法律法规、遵守网络伦理、倡导诚信代码，以实现技术应用与个人道德修养的双重提升。

3. 实践育人策略。本教材以项目实践促进技能培养与价值观塑造，设计了多个贴近学生生活、服务社会发展的项目任务。这些任务旨在让学生在解决具体技术问题的过程中，体验到劳动的价值，激发爱国情怀，践行社会主义核心价值观。

4. 强化能力培养与思想引领。本教材不仅注重能力的培养，还关注对学生的思想引领。结合党的二十大报告中提出的新思想、新观点、新论断，本教材在案例分析、习题设计中融入国家发展战略及时代主题，引导学生理解并践行习近平新时代中国特色社会主义思想。

5. 互动交流平台。本教材借助配套的微课视频、程序源代码、教学课件，鼓励学生展开讨论，分享技术实践与价值体验，使学生在学习 Python 编程的同时，能够在思想交流中加深对立德树人和社会主义核心价值观的认识。

本教材以项目式教学为主线，循序渐进导入 Python 程序设计知识，所选项目案例丰富、贴近生活，注重培养 Python 程序设计的思路、方法、技巧及良好的编程风格。全书内容共分为 15 个项目，并且为了方便读者参考与学习，书中所有变量形式与实际应用程序中的形式一致，均采用正体。其中，前 7 个项目为基础知识，重点学习 Python 程序设计的语法规则；项目 8 至项目 13 为进阶知识，重点学习利用 Python 语法进行编程；项目 14 和项目 15 为拓展知识，介绍 Python 在数据可视化与数据分析中的应用。

本教材适合作为高等院校 Python 程序设计课程的教材，也可作为计算机培训机构与编程自学人员的参考书，还可作为 Python 程序开发人员、程序爱好者及计算机等级考试者的参考书。本书提供微课视频，并另外配套程序源代码、教学课件和习题答案等电子资源，有需要者可发邮件至 2393867076@qq.com 领取。

由于时间仓促，书中难免存在疏漏和不足之处，恳请广大读者批评指正。

编者

2024 年 9 月

目录

项目 4　掌握数据类型

项目 5　运用运算符

项目 6　运用程序控制语句

项目 11 异常处理

项目 12 熟悉常用标准库和第三方库

项目 13 图形用户界面编程

项目 11 异常处理

学习目标

➲知识目标

（1）认识异常的概念，了解异常产生的原因。

（2）掌握异常处理机制在编码中的应用。

➲能力目标

能够自定义异常与清理异常。

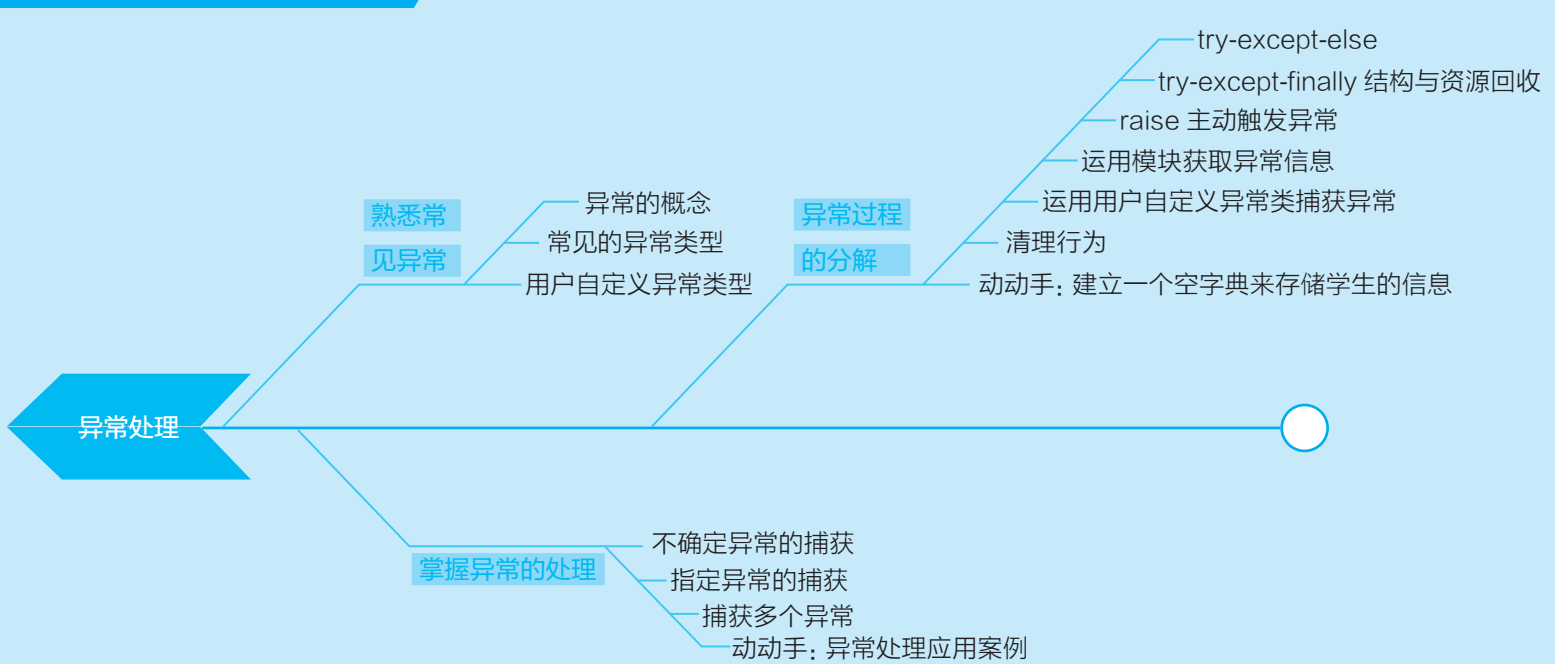
➲素质目标

（1）提高编码过程中的异常应对意识和代码开发过程中的严谨化程度。

（2）进行编码技能的纵向化拓展，培养综合型工匠能力。

（3）陶冶敢于担当、善于钻研的精神，塑造不畏困难的大国工匠情怀。

思维导航



识海启航

异常处理机制是 Python 用于处理程序运行过程中可能出现的错误和异常的一种机制。它通过使用关键字 try、except、else、finally 和 raise 来实现异常的捕获、处理、抛出和清理行为。作为学习计算机编程的青年一代，我们应当秉持大国工匠精神，面对问题和挫折时迎难而上。异常处理机制正是编程中应对问题的有效范式，熟练掌握这种机制不仅能提升程序调试能力，还能培养我们直面挑战、解决问题的职业素养。

任务 1　熟悉常见异常

1. 异常的概念

程序异常是指在程序运行过程中，由于某种原因导致程序无法正常执行的情况。这些异常可能是程序设计错误、输入数据不合法、系统资源不足等原因所引起的。当程序发生异常时，通常会中断程序的执行流程，并抛出一个异常对象。这个异常对象包含了关于异常的详细信息，例如异常类型、异常值等。程序员可以通过捕获和处理这些异常来修复程序中的错误，或者向用户提供有用的错误信息。

微课 11-1

程序在运行时，如果 Python 解释器遇到一个错误，会停止程序的执行，并且提示一些错误信息，这就是异常。异常分为显性异常和隐性异常。显性异常在代码运行时直接可以被看到，通过开发工具（如 IDE）能够提示，而隐性异常在代码静态时无法体现，只有在代码运行时才会显示。

在程序开发过程中，错误是常态化现象。程序的错误大致可分为两类：语法错误和运行时错误。在 Python 中，这种运行时产生的错误被称为异常（exceptions）。

异常涉及的语句和处理方法包括如下内容。

（1）try-except 语句，用于捕获并处理异常。在 try 块中编写可能引发异常的代码。如果发生异常，则执行 except 块中的代码来处理异常。

（2）else 子句，可选。如果 try 块中的代码没有引发任何异常，则执行 else 块中的代码。

（3）finally 子句，可选。无论是否发生异常，都会执行 finally 块中的代码。通常用于资源的释放等清理操作，确保程序的稳定性。

（4）raise 语句，用于抛出自定义的异常。可以指定异常类型和异常信息。

（5）自定义异常。可以通过继承内置的 Exception 类或其子类来创建自定义异常。自定义异常可以包含额外的属性和方法，以便更好地描述和处理异常情况。

（6）异常类的继承关系。异常类之间存在继承关系，子类可以继承父类的属性和方法，也可以覆盖父类的方法。这样可以方便地处理不同类型的异常情况。

（7）预定义清理行为。在 finally 子句中，可以使用预定义的清理行为，例如关闭文件、释放资源等。这些清理行为会在程序结束时自动执行，确保资源得到正确释放。

2. 常见的异常类型

在编写 Python 代码的过程中，难免会出现一些错误的情况，比如语法错误、变量名错误等，这时候就需要通过异常处理来避免程序意外停止，从而简化程序调试的过程，提高编码效率。异常类型的识别对于异常的处理与避免意义重大。常见的异常类型主要包括以下内容。

（1）SyntaxError 语法错误。

SyntaxError 是 Python 中的一种常见错误，它表示代码的语法有误。当 Python 解释器遇到无法理解的代码时，就会引发 SyntaxError。具体如下：

```
print "Hello,World!"
```

Python 3 已不再支持上面这种写法，所以在运行时，解释器会报语法错误，当 Python 解释器发现程序中有语法错误时，会抛出 SyntaxError 异常。例如：

```
pirnt 'hello world'
```

在 Python3.X 版本中，print 应该加括号，正确的写法是：

```
print('hello world')
```

如果写成 print 'hello world'，运行程序时就会抛出 SyntaxError 异常。

（2）NameError 变量名错误。

当 Python 遇到未定义的变量时，会抛出 NameError 异常。例如：

```
a = 1
print(b)
```

因为变量 b 未被定义，所以会抛出 NameError 异常。

（3）TypeError 类型错误。

当字符串和整数不能直接相加时，便会抛出 TypeError 异常。当尝试使用不支持的操作类型时，也会抛出 TypeError 异常。例如：

```
a = 'hello'
b = 5
print(a + b)
```

（4）ZeroDivisionError 除数为零错误。

当尝试除以零时，会抛出 ZeroDivisionError 异常。例如：

```
a = 5/0
```

由于除以零是非法的操作，所以会抛出 ZeroDivisionError 异常。

（5）indexError 索引错误。

当尝试访问列表或元组中不存在的元素时，会抛出 indexError 异常。例如：

```
a = [1,2,3,]
print(a[3])
```

由于 a 中只有三个元素，访问索引 3 将会抛出 indexError 异常。

（6）KeyError 字典键错误。

当尝试访问字典中不存在的键时，会抛出 KeyError 异常。例如：

```
a = {'name':'Tom','age':20}
print(a["gender])
```

因为 a 中不存在键 'gender'，所以会抛出 KeyError 异常。

（7）ValueError 值错误。

当函数参数类型正确但参数值错误时，会抛出 ValueError 异常。例如：

```
a = int('abc')
```

因为 'abc' 不能被转换为整数类型，所以会抛出 ValueError 异常。

（8）运行时错误。

程序在语法上都是正确的，但在运行时发生了错误。如：

```
a = 1/0
```

上面这句代码的意思是用 1 除以 0，并赋值给 a。因为 0 作除数是没有意义的，所以运行后会产生运行时错误。

常见的几种运行时错误异常情况如下。

①缺少冒号。在定义函数、循环或条件语句时，要确保在语句后面加上冒号。例如：

```
if x > 0  # 错误，缺少冒号
    print("x is positive")
```

修改为：

```
if x > 0:  # 正确
    print("x is positive")
```

②缩进错误。Python 使用缩进来表示代码块，如果缩进不正确，会导致 SyntaxError。要确保在代码块中使用一致的缩进。例如：

```
for i in range(5):
    print(i)  # 错误，缺少缩进
```

修改为：

```
for i in range(5):
    print(i)  # 正确
```

3. 用户自定义异常类型

Python 支持使用类来表示异常。这些类可以是 Python 的内置异常类，也可以是用户自定义的异常类。使用类来输出异常是一种比较好的方式，因为它提供了更高的灵活性和可扩展性。

用户自定义的异常类与内置异常类并无差别，只是内置异常类定义在 exceptions 模块中。当 Python 解释器启动时，exceptions 模块就会事先加载。

Python 允许用户定义自己的异常类，并且用户自定义的异常类必须从任何一个 Python 的内置异常类派生而来。

下面的示例使用 Python 内置的 Exception 异常类作为基类，创建一个用户自定义的异常类 URLError。

【例 11.1】(example11_01.py）创建和使用异常类 URLError。

```
class URLError(Exception):
    pass
```

```
try:
    raise URLError(" 这是 URL 异常 ")
except URLError as inst:
    print(inst.args[0])
```

代码运行结果：

```
这是 URL 异常
```

inst 变量是用户自定义异常类 URLError 的实例变量，inst.args 就是该用户定义异常类的 args 属性值，还可以将所创建的用户自定义异常类再当作其他用户自定义异常类的基类。

自定义的 URLError 异常类作为基类，创建一个用户自定义的异常类 HostError。

【例 11.2】(example11_02.py) 以 URLError 为基类创建异常类 HostError。

```
class HostError(URLError):
    def printString(self):
        print (self.args)
try:
    raise HostError("Host Error")
except HostError as inst:
    inst.printString()
```

运行结果：

```
('Host Error',)
```

借助重写类的 __str__( ) 方法可以改变输出字符串。

【例 11.3】(example11_03.py) 输出异常信息。

```
class MyError(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)
try:
    raise MyError(100)
except MyError as e:
    print(' 异常发生的数值为 :', e.value)
```

代码运行结果：

```
异常发生的数值为 : 100
```

注意：一般异常类在创建的时候都以 Error 结尾，与标准异常类命名一样。

任务 2　掌握异常的处理

微课 11-2

异常处理是 Python 编程中一个重要的环节。它允许捕获并处理可能出现的异常情况，从而确保程序的稳定性和可靠性。在 Python 中，异常处理主要通过使用

try–except 语句块来实现。

具体来说，可以将可能引发异常的代码放在 try 块中，然后在 except 块中定义对异常的处理逻辑。当 try 块中的代码引发异常时，程序将跳转到匹配的 except 块进行处理。此外，可以使用不同的 except 块来处理不同类型的异常，也可以使用一个 except 块来处理多个异常类型。例如，可以通过调用 traceback 模块中的 print_exc( ) 方法来打印异常的详细信息，包括异常的类型、原因以及异常出现的位置。这对于在开发和调试阶段找出问题的原因非常有帮助。

Python 提供了许多标准异常类，包括但不限于 AttributeError, KeyError, FloatingPointError 等。每种异常都有其特定的触发条件和处理方法。理解和掌握这些异常的特性和处理方式，对于有效地进行异常处理至关重要。

当发生异常时，需要对异常进行捕获，然后进行相应的处理。Python 的异常捕获通常使用 try... except... 结构实现。把可能发生错误的语句放在 try 块里，用 except 块来处理异常。每一个 try 块都必须至少对应一个 except 块。

异常处理的基本语法结构如下：

```
try:
    代码块
except
    异常的处理
else:
    执行成功
finally:
    异常收尾
```

在使用异常处理时，需要注意以下几点。

（1）避免过多依赖于异常处理机制，以提高程序的容错性和健壮性。

（2）按照 try–except–else–finally 的顺序编写代码，确保异常被正确捕获和处理。

（3）使用 raise 语句抛出自定义的异常，以便在程序中进行更详细的错误处理和提示。

（4）只匹配一个 except 子句，避免多个 except 子句同时捕获相同的异常。

（5）先写子类异常再写父类异常，确保子类异常能够优先捕获。

（6）如果 except 捕获的错误与触发的错误不一致，程序会捕获不到，需要检查代码逻辑。

1. 不确定异常的捕获

由于异常的种类众多，不是所有发生异常的情况下都能确定具体的异常类型。为此，在不确定异常类型时，可用万能异常 Exception 予以捕获。

【例 11.4】(example11_04.py）捕获异常 Exception。

```
try:
    a=1
    b=0
    print(a/b)
except Exception as e: # 不适用 except 异常捕获时
    print(e)  # division by zero
    print(e.args)  # ('division by zero',)
```

运行结果：

```
division by zero
```

```
('division by zero',)
```

使用 try 块来执行可能引发异常的代码，如果发生异常，则使用 except 块来捕获它，并将异常对象存储在变量 e 中。然后，可以使用 print( ) 函数输出异常信息。需要注意的是，使用 Exception 类可以捕获所有类型的异常。如果只想捕获特定类型的异常，可以将 Exception 替换为该异常类的名称。例如，如果只想捕获 ValueError 异常，可以将“except Exception as e:”替换为“except ValueError as e:”。

2. 指定异常的捕获

当确定异常类型时，可以指定异常类型来捕获特定的异常。例如，当打开文件时找不到文件，就会报出 FileNotFoundError 异常，表示文件不存在。对于指定类型的异常，可以使用 try–except 语句来捕获，具体操作步骤如下。

（1）在 except 关键字后面跟上异常类型，如 ValueError、TypeError 等。

（2）在 except 代码块中编写处理异常的代码。

（3）如果需要捕获多个异常类型，可以使用元组将它们括起来，如 (ValueError, TypeError)。

（4）如果需要捕获所有异常，可以使用 Exception 类，但通常不推荐这样做，因为这样会隐藏程序中的其他错误。

【例 11.5】(example11_05.py）捕获异常 FileNotFoundError。

```
try:
    f=open("example.txt")
    f.read()
except FileNotFoundError as e:
    print(e)  #No such file or directory: 'example.txt'
    print(e.args)  # (2, 'No such file or directory')
    print(e.strerror)  # No such file or directory
    print(e.filename)  # example.txt
    print(e.errno)
```

代码运行结果：

```
[Errno 2] No such file or directory: 'example.txt'
(2, 'No such file or directory')
No such file or directory
example.txt
2
```

注意：通常这类异常的设定要同代码所涉及的业务保持一致，即处理文件打开业务就对可能存在的 FileNotFoundError 异常进行捕获，其他类型的业务也应同样处理。

3. 捕获多个异常

当可能存在多个异常时，既可以定义多个 except 来捕获多个异常，也可以在一个 except 中声明多个异常。如果想要捕获多个异常，可以使用元组将它们放在一起。可以通过以下步骤完成。

（1）需要定义一个函数，该函数可能会引发多种类型的异常。例如，可以定义一个函数，该函数接受一个整数作为参数，并尝试将其除以零，这将引发 ZeroDivisionError 异常。

（2）可以使用 try–except 语句来捕获函数可能引发的异常。为了捕获多个异常，可以使用元组将它们放在一起。在这个例子中，将捕获 ZeroDivisionError 和 TypeError 异常。

（3）当运行这段代码时，由于试图将一个整数除以零，会引发 ZeroDivisionError 异常。然后，由于没有处理 TypeError 异常，程序将终止并显示错误消息。

（4）想要在捕获异常后执行一些操作，可以在 except 子句中添加相应的代码。例如，可以在捕获到异常后打印一条消息，并将结果设置为 None。

【例 11.6】(example11_06.py）分开捕获多个异常。

```
try:
    a=[1,2]
    print(a[3])
except IOError as e:  # 由于内容不属于 IOError，所以会跳过这个异常
    print(" 读写异常 ")
    print(e)
except IndexError as e:
    print(f" 数组越界异常 :{e}")  # 数组越界异常 :list index out of range#
```

代码运行结果：

```
数组越界异常 :list index out of range
```

【例 11.7】(example11_07.py）同时捕获多个异常。

```
try:
    a=[1,2]
    print(a.index(3))
except(IOError,ValueError) as e:
    print(f" 读写异常或结果异常 :{e}")  # 读写异常或结果异常 :3 is not in list
```

代码运行结果：

```
读写异常或结果异常 :3 is not in list
```

4. 动动手：异常处理应用案例

【实践案例 01】(case11_01.py)raise 触发异常——猜数字游戏。

猜数字游戏：想一个 1 到 100 之间的数字，然后猜测这个数字是多少。每次猜测后，程序会告诉是否猜对了或者给出提示：若猜的数大于实际数，提示“猜大了”；若猜的数小于实际数，则提示“猜小了”。在没有猜对的情况下会反复提示“请输入一个 1~100 的整数：”，直到猜对时会提示“恭喜你！猜中了！”，然后程序结束。这是在正常输入数字的情况下，当输入的为非数字或者非整数时便会提示异常信息。

```
import random
data=random.randint(1,100)
while 1==1:
   try:
      num = input(" 请输入一个 1~100 的整数 :")
      if num.isdigit():
         if int(num)>data:
```

```
            print(" 猜大了！ ")
        elif int(num)<data:
            print(" 猜小了！ ")
        else:
            print(" 恭喜你！猜中了！ ")
            break
    else:
        raise ValueError('num 非数字，请输入数字 ')
except ValueError as e:
    print(" 结果异常，请检查输入项 ")
```

代码运行结果（正常输入数字的情况下）：

```
请输入一个 1~100 的整数 :4
猜小了！
请输入一个 1~100 的整数 :5
猜小了！
请输入一个 1~100 的整数 :66
猜小了！
请输入一个 1~100 的整数 :77
猜大了！
请输入一个 1~100 的整数 :70
猜小了！
请输入一个 1~100 的整数 :71
恭喜你！猜中了！
输入异常的情况下：
请输入一个 1~100 的整数 :e
结果异常，请检查输入项
请输入一个 1~100 的整数 :45.6
结果异常，请检查输入项
请输入一个 1~100 的整数 :
```

【实践案例 02】(case11_02.py) 开发一个计算器程序。

要开发一个计算器程序，首先需要了解基本的算术运算，如加、减、乘、除。接下来，将使用 Python 的 input( ) 函数获取用户输入的两个数字和一个运算符，然后根据运算符执行相应的操作，并输出结果。

```
while 1==1:
    try:
        num1=float(input(" 运算数 1："))# 运算数 1
        op=input(" 运算符：") # 运算符
        num2=float(input(" 运算数 2："))# 运算数 1
        if op=='+':
            print(num1,"+",num2,"=",num1+num2)
        elif op=='-':
            print(num1,"-",num2,"=",num1-num2)
        elif op=='*':
            print(num1,"*",num2,"=",num1*num2)
        elif op=='/':
            print(num1,"/",num2,"=",num1/num2)
```

```
        else:
            print("Error!")
    except ValueError as e:
        print(" 数据类型异常，请检查输入项 ")
    except ZeroDivisionError as e:
        print(" 除数异常，请检查输入项 ")
```

代码运行结果：

```
运算数 1：4
运算符：+
运算数 2：5
4.0 + 5.0 = 9.0
运算数 1：5
运算符：/
运算数 2：0
除数异常，请检查输入项
运算数 1：4
运算符：*
运算数 2：r
数据类型异常，请检查输入项
运算数 1：
```


任务 3　异常过程的分解

1. try-except-else

在原本的 try-except 结构的基础上，Python 异常处理机制还提供了一个 else 块，也就是原有 try-except 语句的基础上再添加一个 else 块，即 try-except-else 结构。else 包裹的代码，只有当 try 块没有捕获到任何异常时，才会得到执行；反之，如果 try 块捕获到异常，即调用对应的 except 来处理异常，else 块中的代码也不会得到执行。

微课 11–3

【例 11.8】(example11_08.py) 用 try-except-else 进行异常捕获。

```
try:
    age = 18+"a"
    print(age)
except TypeError as e:
    print(e)
else:
    print(" 无异常 ")
```

代码运行结果：

```
unsupported operand type(s) for +: 'int' and 'str'
```

注意：只有当 try 中定义的语句无异常时，才会走到 else 分支 print(“无异常”)，打印结果“无异常”。

2. try-except-finally 结构与资源回收

Python 的异常处理机制还提供了一个 try-except-finally 结构，其中 finally 块通常用于进行扫尾清理工作，如资源回收。finally 块的功能是：无论 try 块是否发生异常，最终都会进入 finally 块，并执行其中的代码。

【例 11.9】(example11_09.py）利用 finally 语句进行资源回收。

```
try:
    f = open('D://Code//Python//Chapter09//file//example.txt')
except FileNotFoundError as e:
    print(e)  # [Errno 2] No such file or directory
else:
    print(" 文件读写正常 ")
finally: # 无论程序是否有异常，都会走到 finally 分支
    print(" 程序运行完成 ") # over
```

代码运行结果如下。

文件存在：

```
文件读写正常
程序运行完成
```

文件不存在：

```
[Errno 2] No such file or directory: 'D://Code//Python//Chapter09//file//example000.txt'
程序运行完成
```

注意：不论是否会有异常发生，最终都会执行 finally 块中的部分。finally 块非常适合用于资源回收，如关闭文件、释放网络连接等。

3. raise 主动触发异常

当程序出现错误，Python 会自动引发异常。此外，也可以通过 raise 语句引发异常。一旦执行了 raise 语句，raise 后面的语句将不会执行。raise 语句的基本语法格式为：

```
raise [ 异常类型 [(reason)]]
```

raise 语句有三种常用的用法。

（1）单独一个 raise，该语句用于引发当前上下文中捕获的异常（通常在 except 块中）。如果不在 except 块中使用，将默认引发 RuntimeError 异常。

（2）raise 后面跟一个异常类名称，表示引发执行类型的异常。

（3）raise 异常类名称 (描述信息)：表示引发指定类型的异常，并附带异常的描述信息。

【例 11.10】(example11_10.py）raise 主动触发异常。

```
try:
    num = input(" 请输入一个数字 :")
    if not num.isdigit():
        raise ValueError('num 非数字，请输入数字 ')
except ValueError as e:
```

```
    print(" 结果异常，请检查输入项 ")
```

代码运行结果：

```
请输入一个数字 :e
结果异常，请检查输入项
```

注意：raise 对异常的处理具有主动性，当异常产生会直接将异常按照类型予以举出。

4. 运用模块获取异常信息

在实际调试程序的过程中，仅获得异常的类型通常是不够的。为了更有效地解决问题，还需要借助更详细的异常信息。捕获异常时，通常使用 sys 模块和 traceback 模块中的相关函数，有两种方式可获得更多的异常信息。

1）借助 sys 模块中的 exc_info( ) 方法

【例 11.11】(example11_11.py）sys 模块中的 exc_info( ) 方法。

```
import sys
try:
    f=open('read1.txt')
    f.read()
except Exception as e:
    print(e)  # [Errno 2] No such file or directory
    print(sys.exc_info()) # 打印异常信息
```

代码运行结果：

```
[Errno 2] No such file or directory: 'read1.txt'
(<class 'FileNotFoundError'>, FileNotFoundError(2, 'No such file or directory'), <traceback object at 0x000001AAF8751680>)
```

2）使用 traceback 模块中的相关函数

使用 traceback 模块查看异常传播轨迹，首先需要将 traceback 模块引入，该模块提供了如下两个常用方法。

（1）traceback.print_exc( )：将异常传播轨迹信息输出到控制台或指定文件中。

（2）traceback.format_exc( )：返回一个字符串而不是打印到文件。

【例 11.12】(example11_12.py）traceback 模块中的 print_exc( ) 方法。

```
import traceback
try:
    f = [0, 1, 2, 3]
    f.index(5)
except ValueError as e:
    print(traceback.print_exc())
```

代码运行结果：

```
Traceback (most recent call last):
```

```
  File "D:\Code\Python\Chapter09\example11_12.py", line 4, in <module>
    f.index(5)
ValueError: 5 is not in list
None
```

注意：将异常信息输出到控制台或指定文件中，此时控制台打印的为捕获的异常信息，非报错。

5. 运用用户自定义异常类捕获异常

在 Python 中，可以使用 try–except 语句来捕获并处理异常。如果想要创建自定义的异常类，可以通过继承 Exception 类或其子类来创建新的异常类。

下面的示例代码展示了如何使用用户自定义异常类来捕获异常。

语法结构：

```
class 自定义异常名 (Exception):pass
```

class 是定义类的关键字，Exception 表示继承自 Exception 类，也可以继承自其他 Exception 的子类。在大部分情况下，创建自定义异常类都可采用与上面相似的代码来完成，同时，自定义异常的命名应尽量准确地描述该异常的性质。大部分自定义异常类不需要类体定义，因此使用 pass 语句作为占位符即可。

【例 11.13】(example11_13.py) 用户自定义异常处理。

```
class MyCustomError(Exception):
    def __init__(self, message):
        self.message = message
        # 重写 __str__() 方法，返回错误信息
    def __str__(self):
        return f"My custom error occurred: {self.message}"

    def divide_numbers(a, b):
        try:
            result = a / b
            print(" 结果为 :", result)
        except ZeroDivisionError as e:
            raise MyCustomError(" 除数不能为零 ") from None

divide_numbers(10, 2)  # 输出：结果为 : 5.0
divide_numbers(10, 0)
```

代码运行结果：

```
结果为 : 5.0
Traceback (most recent call last):
  File "D:\Code\Python\Chapter09\example11_13.py", line 17, in <module>
    divide_numbers(10, 0)
  File "D:\Code\Python\Chapter09\example11_13.py", line 14, in divide_numbers
    raise MyCustomError(" 除数不能为零 ") from None
MyCustomError: My custom error occurred: 除数不能为零
```

6. 清理行为

在 Python 中，“预定义清理”指的是使用 with 语句结合上下文管理器（Context Manager）的一种技术。这种技术用于确保代码块执行结束后，自动进行资源的清理与释放，从而确保资源的正常关闭，即便有异常出现也能够对资源加以释放。

通常，一个类只要实现了 __enter__ 和 __exit__ 方法，就可以成为一个上下文管理器。__enter__ 方法在进入代码块之前被调用，而 __exit__ 方法则在代码块执行后被调用。无论是否发生异常，__exit__ 方法接受三个参数，分别是异常类型、异常值和异常回溯信息。使用预清理的主要方式是通过 with 语句，with 语句在进入代码块之前调用上下文管理器的 __enter__ 方法，然后在代码块执行结束后通过执行 __exit__ 方法来释放资源。有些对象定义了标准的清理行为，无论对象操作是否成功，不再需要该对象的时候就会起作用。

【例 11.14】(example11_14.py）清理行为。

```
class MyFile:
def __init__(self,filename,mode):
    self.filename=filename
    self.mode=mode
    self.file=None
    def __enter__(self):
        self.file=open(self.filename,self.mode)
        return self.file
    def __exit__(self,exc_type,exc_value,traceback):
        if self.file:
            self.file.close()
file_path="example.txt"
with MyFile(file_path,"w") as f:
    f.write("Hello World!")
```

代码运行结果：在 example11_14.py 文件同级目录创建 example.txt 文件并写入 "Hello World!"。

在上面的代码中，MyFile 类是一个上下文管理器，当使用 with 语句打开时，会自动在代码块执行结束后关闭文件，即便有异常出现也同样能保证文件的正确关闭。

7. 动动手：建立一个空字典来存储学生的信息

【实践案例 03】(case11_03.py）学生信息的数字化管理。

实现一个简单的学生成绩管理系统，要求用户输入学生的姓名和成绩，然后将学生的信息存储在一个字典中。用户可以查询学生的成绩，也可以修改学生的成绩。在查询和修改成绩的过程中，需要捕获可能出现的异常，如 KeyError（学生姓名不存在）和 ValueError（输入的成绩不是数字）。

```
students = {}
while True:
    try:
        name = input(" 请输入学生姓名（输入 q 退出）: ")
        if name == "q":
            break
        score = float(input(" 请输入学生成绩："))
        students[name] = score
    except ValueError:
```

```
            print(" 输入的成绩不是数字，请重新输入。")
        except KeyError:
            print(" 学生姓名不存在，请重新输入。")
    while True:
        try:
            name = input(" 请输入要查询的学生姓名（输入 q 退出）：")
            if name == "q":
                break
            print(" 学生的成绩是：", students[name])
        except KeyError:
            print(" 学生姓名不存在，请重新输入。")
        except ValueError:
            print(" 输入的成绩不是数字，请重新输入。")
```

代码输出结果：

```
请输入学生姓名（输入 q 退出）：张某
请输入学生成绩：87
请输入学生姓名（输入 q 退出）：鲁某
请输入学生成绩：90
请输入学生姓名（输入 q 退出）：q
请输入要查询的学生姓名（输入 q 退出）：张某
学生的成绩是：87.0
请输入要查询的学生姓名（输入 q 退出）：e
学生姓名不存在，请重新输入。
请输入要查询的学生姓名（输入 q 退出）：
```

✪课后加油站

异常处理应用拓展

异常处理在程序中非常重要，它可以帮助捕获和处理程序运行过程中可能出现的错误。以下是一些常见的异常处理应用拓展。

（1）文件操作。

在读取或写入文件时，可能会遇到文件不存在、权限不足等问题。通过使用异常处理，可以确保程序在遇到这些问题时不会崩溃，而是给出相应的提示信息。程序代码（tuozhan11_01.py）如下：

```
try:
    with open("file.txt", "r") as file:
        content = file.read()
except FileNotFoundError:
    print(" 文件不存在 ")
except PermissionError:
    print(" 权限不足 ")
```

（2）网络请求。

在进行网络请求时，可能会遇到网络连接失败、超时等问题。通过使用异常处理，可以确

保程序在遇到这些问题时不会崩溃，而是给出相应的提示信息。程序代码（tuozhan11_02.py）如下：

```
import requests
url = "https://www.example.com"
try:
    response = requests.get(url, timeout=5)
    response.raise_for_status()
except requests.exceptions.RequestException as e:
    print(" 网络请求出错：", e)
```

（3）数据库操作。

在进行数据库操作时，可能会遇到连接失败、查询错误等问题。通过使用异常处理，可以确保程序在遇到这些问题时不会崩溃，而是给出相应的提示信息。程序代码（tuozhan11_03.py）如下：

```
import sqlite3
db_name = "test.db"
try:
    conn = sqlite3.connect(db_name)
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM table_name")
    results = cursor.fetchall()
except sqlite3.Error as e:
    print(" 数据库操作出错：", e)
finally:
    cursor.close()
    conn.close()
```

（4）多线程 / 多进程。

在进行多线程或多进程编程时，可能会遇到线程同步问题、资源竞争等问题。通过使用异常处理，可以确保程序在遇到这些问题时不会崩溃，而是给出相应的提示信息。程序代码（tuozhan11_04.py）如下：

```
from threading import Thread
def worker():
    try:
        # 执行任务
    except Exception as e:
        print(" 线程出错：", e)
threads = []
for i in range(5):
    t = Thread(target=worker)
```

```
        threads.append(t)
        t.start()
    for t in threads:
        t.join()
```

总之，通过合理地使用异常处理，可以使程序更加健壮，能够更好地应对各种可能出现的问题。

巩固与练习

一、选择题

1. 以下哪个选项不是 Python 中的异常类型？（　　）

A. ValueError　　B. TypeError　　C. IndexError　　D. KeyError

2. 在 Python 中，哪个关键字用于定义自定义异常？（　　）

A. class　　B. def　　C. try　　D. except

3. 以下哪个函数用于抛出一个指定的异常？（　　）

A. raise ValueError("Invalid value")

B. raise TypeError("Invalid type")

C. raise IndexError("Invalid index")

D. raise KeyError("Invalid key")

4. 在 Python 中，哪个关键字用于捕获异常？（　　）

A. class　　B. def　　C. try　　D. except

5. 以下哪个函数用于获取当前异常的详细信息？（　　）

A. traceback.format_exc( )　　B. sys.exc_info( )

C. logging.exception( )　　D. print_exc( )

二、编程题

1. 编写程序，通过 raise 引发一个 TypeError 异常，捕获后输出“捕获到 TypeError”。

2. 编写程序，按照用户输入的直角三角形的面积，若边长为负值则抛出异常（直角三角形的面积公式：S=h*f/2 ）。

3. 编写程序实现对学生的 Python 课程期末成绩进行等级评定，大于等于 90 分的为“优秀”，80 分到 90 分的（包括 80 分）的为“良好”，60 分到 80 分的（包括 60 分）为“合格”，60 分以下为“不合格”，最后把学生成绩打印出来。

☆学·做·思

1. 做一份知识归纳图表，描述 Python 异常处理的作用，并对比有异常处理和没有异常处理的区别。

思维导图：

思考总结：

2. 调研有关 Python 异常处理的应用案例，完成一篇 300 字的总结报告。你认为案例应用场景最多的是哪一种？请给出你的理由。

总结报告：

思考拓展：

项目12 熟悉常用标准库和第三方库

学习目标

➲知识目标

（1）学习 Python 中标准库和第三方库的含义。

（2）了解常用标准库和第三方库的安装和引用方法。

（3）体会常用标准库和第三方库在实际场景中的应用。

➲能力目标

（1）能够熟练掌握标准库 turtle、random 和 time 的使用方法。

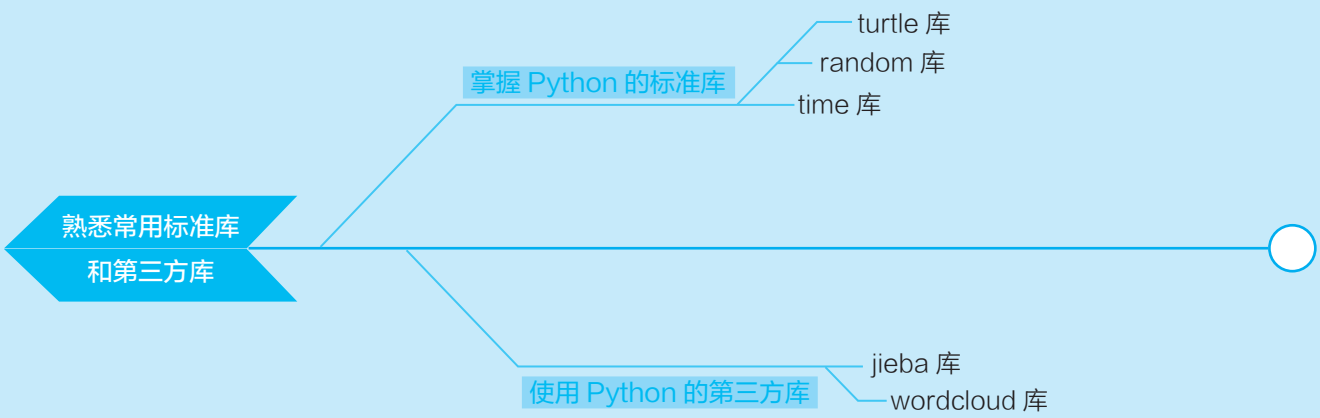
（2）能够熟练掌握第三方库 jieba 和 wordcloud 的使用方法。

➲素质目标

（1）综合利用 Python 库的知识和方法，举一反三，在实际运用中掌握更多的标准库和第三方库的用法，能够根据实际情况运用 Python 技术解决本专业的具体问题，提高编程能力。

（2）培养不怕困难、细致严谨、精益求精、勇于探索的工匠精神和良好习惯。

思维导航



识海启航

在 Python 强大而灵活的编程语言中，熟悉并熟练运用标准库与第三方库是每一位开发者成长道路上不可或缺的一步。标准库是 Python 安装时自带的一组功能强大的模块。它涵盖了从数据处理、文件操作、网络编程到系统管理等多个方面的基础功能，为开发者提供了解决常见问题的“工具箱”。而第三方库则是由 Python 社区成员开发的，用于扩展 Python 功能的专业工具包。它涵盖了机器学习、数据分析、Web 开发、科学计算等众多专业领域，让 Python 在各个领域的应用都如虎添翼。

掌握这些库不仅能够大幅提升开发效率，还能让代码更加简洁、易读、可维护。通过深入学习并实践标准库与第三方库的使用，你可以轻松构建出功能丰富、性能卓越的应用程序。无论你是初学者想要快速上手 Python 项目开发，还是资深开发者希望进一步优化代码质量与性能，熟悉并灵活运用这些库都是不可或缺的能力。因此，让我们一起踏上这段探索之旅，深入挖掘 Python 标准库与第三方库的无限潜力吧！

任务 1　掌握 Python 的标准库

微课 12–1

Python 有一套很有用的标准库（Standard Library），它是 Python 的重要组成部分。Python 语言的核心只包含数字、字符串、列表、字典、文件等常见类型和函数，而由 Python 标准库提供了系统管理、网络通信、文本处理、数据库接口、图形系统、XML 处理等额外的功能。当安装 Python 解释器时，标准库会随其被安装。Python 标准库命名接口清晰、文档良好，很容易学习和使用，同时可以让编程事半功倍，是程序开发的利器。

本任务主要学习 Python 常用的标准库 turtle 库、random 库和 time 库。从对这三种标准库的学习中掌握 Python 标准库的概念和使用方法，以便后期对其他标准库进行自主学习。

1. turtle 库

turtle 库也称海龟绘图库，对应的文件名为 turtle.py，是 Python 的一个直观有趣的图形绘制标准库。它提供了绘制线、圆以及其他形状的函数，使用该标准库可以创建图形窗口，在图形窗口中通过简单重复动作直观地绘制界面与图形。

在 Python 中，标准库需要使用 import 语句导入。如果程序中需要使用 turtle.py 文件中写好的各种函数，则需要首先导入该库文件，语句如下：

```
import turtle
```

1）turtle 空间坐标体系

turtle 库是 Python 语言中一个很流行的绘图库，它通过模拟一个小乌龟在平面上移动的方式来进行绘图。想象一个小乌龟从一个平面坐标系的原点 (0,0) 位置开始，它根据一组函数指令的控制，在这个平面坐标系中移动，从而在它爬行的路径上绘制出图形。turtle 库绘图坐标体系如图 12–1 所示。

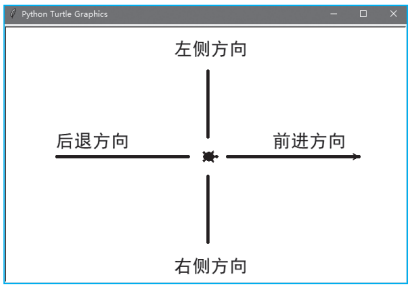


图 12-1　turtle 库绘图坐标体系

turtle 库中常用的函数及功能如下。

turtle.goto(x,y)：表示移动到窗口上的某一点（带轨迹）。turtle 库的 goto( ) 函数是以绘图窗口中心为坐标原点，把窗口划分为四个象限的坐标系。如果移动过程中不想带轨迹，则可以配合 turtle.penup( ) 和 turtle.pendown( ) 这两个函数使用。

turtle.penup( )：提起画笔（海龟起飞），库别名为 turtle.pu( )。

turtle.pendown( )：落下画笔（海龟降落），库别名为 turtle.pd( )。

turtle.forward(d)：前进（d 表示距离），库别名为 turtle.fd(d)。

turtle.backward(d)：后退（d 表示距离），库别名为 turtle.bk(d)。

turtle.circle(r,angle)：画一个半径为 r，角度为 angle 的圆。若半径 r ⩾ 0，则代表圆心在海龟左侧；反之，在右侧。

2）turtle 角度坐标体系

turtle 角度坐标体系函数及功能如下。

turtle.seth(angle)：改变海龟的行动方向，使海龟朝向 angle，其中 angle 表示绝对角度。

turtle.left(angle)：使海龟往左转 angle 的角度，其中 angle 表示相对于海龟当前自身的角度。

turtle.right(angle)：使海龟往右转 angle 的角度，其中 angle 表示相对于海龟当前自身的角度。

【例 12.1】(example12_01.py）使用 turtle 库画一个简单的图形。

```
import turtle
turtle.penup()
turtle.goto(-350,-150)
turtle.pendown()
for x in range(4):
    turtle.forward(300)
    turtle.left(90)
turtle.penup()
turtle.goto(200,-150)
turtle.pendown()
turtle.circle(150)
turtle.penup()
turtle.done()
```

代码运行结果如图 12-2 所示。

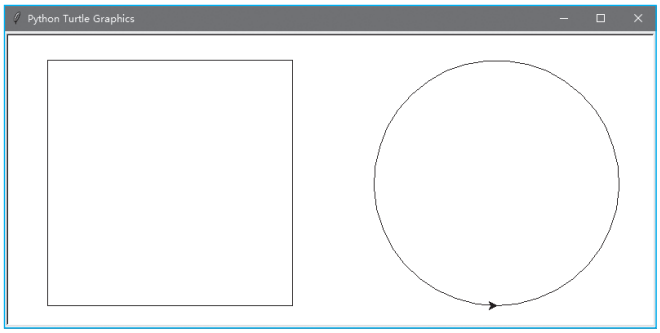


图 12-2　使用 turtle 库画一个简单的图形

3）相关画笔函数

turtle.pensize(width)：设置画笔宽度，当无参数输入时返回当前画笔的宽度。

turtle.pencolor(r,g,b)：设置画笔颜色，(r,g,b)：颜色对应的红、绿、蓝的数值，当无参数输入时返回当前画笔颜色，也可以直接输入对应颜色的字符串，例如，“purple”“red”等。

turtle.speed( )：设置画笔的速度，从 1~10，数字越大则速度越快。

turtle.begin_fill( ) 和 turtle.end_fill( ) 函数用来设置填充域色彩，这两个函数结对使用。

【例 12.2】(example12_02.py) 使用 turtle 库画一颗闪闪的红星。

```
import turtle
turtle.pensize(5)
turtle.begin_fill()
turtle.pencolor("yellow")
turtle.fillcolor("red")
for i in range(5):
    turtle.fd(100)
    turtle.left(72)
    turtle.fd(100)
    turtle.right(144)
turtle.end_fill()
turtle.done()
```

代码运行结果如图 12-3 所示。

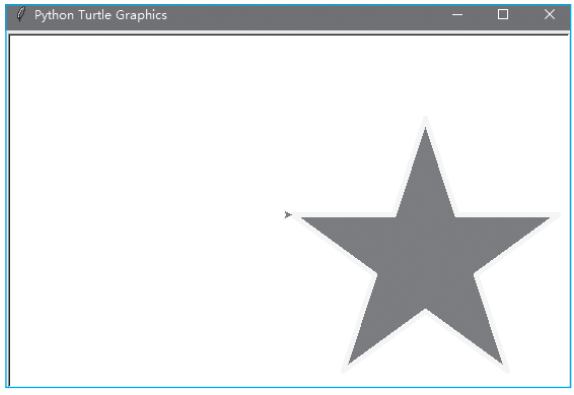


图 12-3　使用 turtle 库画一颗闪闪的红星

以上代码使用函数 turtle.pencolor("yellow")、turtle.fillcolor("red") 设置画笔的颜色为黄色、背景颜色为红色；使用函数 turtle.begin_fill( ) 和 turtle.end_fill( ) 实现背景填充；使用 for 循环函数对小海龟的行走路线进行规划，经过 5 次循环后回到坐标原点，完成五角星的绘画；turtle.done( ) 的作用

是暂停程序，停止画笔绘制，但绘图窗口不关闭，目的是给用户时间来查看图形，否则图形化窗口会在程序完成时立即关闭。

2. random 库

随机数在计算机应用中十分常见。Python 语言提供了 random 库用于生成各种分布的伪随机数序列，主要目的是生成随机数。这个库提供了不同的随机数函数，因此使用时只需要查阅该库的随机数生成函数，然后放到符合使用场景的函数使用即可。random 库是采用梅森旋转算法生成伪随机数序列，可用于除随机性要求更高的加解密算法外的大多数工程应用。

1）基本随机数函数

（1）random( ) 函数。random( ) 函数是 random 库中最基本的函数，主要功能是生成一个 [0.0,1.0) 之间的随机小数。所有其他随机数都是基于这个函数而来的。

【例 12.3】(example12_03.py）使用 random( ) 函数

```
import random
print(random.random())
```

代码运行结果：

```
0.5147738520232679
```

（2）seed( ) 函数。Python 中的随机数需要使用随机数种子来生成；随机数种子确定后，生成的随机序列（每一个数，每个数之间的关系）也会随之确定。随机数种子函数为 random.seed(a)，默认参数 a 等于当前系统时间。如果参数 a 值固定，则随之生成的随机数也会是同一个值（简而言之就是给 random 库里面产生“随机数”的函数定了一个标准参数）。如果参数 a 值不固定，生成的随机数会不一样。

【例 12.4】(example12_04.py）使用 seed( ) 函数理解伪随机数。

```
import random
random.seed()
print(" 种子为系统当前时间，随机数 1 为：",random.random())
random.seed()
print(" 种子为系统当前时间，随机数 2 为：",random.random())
random.seed(10)
print(" 种子为 10，随机数 3 为：",random.random())
random.seed(10)
print(" 种子为 10，随机数 4 为：",random.random())
```

代码运行结果：

```
种子为系统当前时间，随机数 1 为：0.5268311900759516
种子为系统当前时间，随机数 2 为：0.353473194861734
种子为 10，随机数 3 为：0.5714025946899135
种子为 10，随机数 4 为：0.5714025946899135
```

随机数 1 和随机数 2 都是使用 random.seed( ) 生成当前系统时间的种子。两行代码在执行时，对应的系统时间是变化了的，种子不一样，对应的随机值也就不一样。随机数 3 和随机数 4 都是使用 random.seed(10) 生成随机数种子，种子一样，所以随机值也一样。

实际上在 random 库调用时不给种子也可以产生随机数，如果不给种子，则默认的种子是当前第一次调用 random 函数所对应的系统时间。设置随机数种子的好处是可以准确复现随机数序列，用于重复程序的运行轨迹。

2）拓展随机函数

还有一些随机函数，由 random 拓展而来。函数名称、功能描述及应用举例如表 12-1 所示。

表 12-1　拓展随机函数

函数名称	功能描述	应用举例

randint(a, b)	生成一个 [a,b] 之间的随机整数	>>> import random
		>>> random.randint(19,100)
		28

randrange(m,n, [k])	生成一个 [m, n) 之间以 k 为步长的随机整数	>>>random.randrange(10,100,10) 30
random.getrandbits(k)	生成 k 比特长度的随机整数，其中 k 是二进制位数的长度 k: 一个整数，函数的输出范围是 0~2^k-1	>>> random.getrandbits(3) 2
uniform(a, b)	生成一个 [a, b] 之间的随机小数	>>> random.uniform(10,100) 72.04771982642646
choice(seq)	从序列 seq 中随机选择一个元素	>>>random.choice([1,2,3,4,5,6,7,8,9,10]) 8

shuffle(seq)	将序列 seq 中元素随机排列，返回打乱后的序列	>>> s=[1,2,3,4,5,6,7,8,9,10]
		>>> random.shuffle(s)
		>>> s
		[8, 10, 6, 2, 3, 5, 1, 4, 9, 7]

【例 12.5】(example12_05.py）使用 random 库实现简单的微信红包分配。

微信红包大家已不再陌生，近些年来，每当过年期间，在群里发红包、抢红包成了亲朋好友之间感情交流的重要方式。通过 random 库中常见的随机函数，思考一下如何实现简单的红包分配。

```
import random
def red_packet(total, num):
    for i in range(1, num):
        per = random.uniform(0.01, total/(num-i+1)*2)
        total = total - per
        print(" 第 {} 位红包金额：{:.2f} 元 ".format(i, per))
    else:
        print(" 第 {} 位红包金额：{:.2f} 元 ".format(num, total))
red_packet(10, 5)
```

代码运行结果：

```
第 1 位红包金额：3.08 元
第 2 位红包金额：3.35 元
第 3 位红包金额：0.28 元
第 4 位红包金额：3.09 元
第 5 位红包金额：0.19 元
```

3. time 库

1）基本概念

time 库是 Python 中处理时间的标准库，提供获取系统时间、格式化输出时间以及进行精确计时等功能，多用于程序的性能分析。在 Python 中获取时间的常用方法是，先获取时间戳，再将其转换成想要的时间格式。

时间戳（Timestamp），一个能表示一份数据在某个特定时间之前已经存在的、完整的、可验证的数据，通常是一个浮点数，唯一地标识某一刻的时间。Python 中所谓的时间戳指的是从格林威治时间 1970 年 01 月 01 日 00 分 00 秒（北京时间 1970 年 01 月 01 日 08 时 00 分 00 秒）起至现在的总秒数。

在 Python 中，使用一些特殊的符号来格式化时间日期，类似于字符串格式化中的“%s”“%d”等。具体符号如表 12–2 所示。

表 12–2　时间日期格式化符号

符　　号	含　　义
%y	两位数的年份表示（00~99）
%Y	四位数的年份表示（0000~9999）
%m	月份（01~12）
%d	月内中的一天（01~31）
%H	24 小时制小时数（00~23）
%I	12 小时制小时数（01~12）
%M	分钟数（00~59）
%S	秒（00~59）
%a	本地简化的星期名称（如 Mon）
%A	本地完整的星期名称（如 Monday）
%b	本地简化的月份名称（如 Jan）
%B	本地完整的月份名称（如 January）
%X	本地相应的时间表示
%x	本地相应的日期表示

2）库函数

（1）time( ) 函数。time( ) 函数用于返回当前时间的时间戳，time( ) 函数返回的是浮点数。

【例 12.6】(example12_06.py）使用 time() 函数获取当前时间

```
import time
now = time.time()
print(" 当前的时间戳是：%f" %now)
```

代码运行结果：

```
当前的时间戳是：1704470098.429443
```

（2）localtime( ) 函数。localtime( ) 函数的作用是将时间戳转换为本地时间。localtime( ) 函数有一个可选参数用于接收时间戳。如果调用函数时不提供时间戳，localtime( ) 函数默认会使用当前时间戳。它的返回值是以结构化的 struct_time 时间元组方式表示。

struct_time 元组共有 9 个元素，每个元素的含义如表 12-3 所示。

表 12-3　struct_time 元组各字段含义

字　段　名	含　　义
tm_sec	秒，取值区间为 [0,61]（60 和 61 是闰秒）
tm_min	取值区间为 [0,59]
tm_hour	时，取值区间为 [0,23]
tm_mday	一个月中的日期，取值区间为 [1,31]
tm_mon	月份（从一月开始，0 代表一月），取值区间为 [0,12]
tm_year	年份
tm_wday	星期，取值区间为 [0,6]，其中 0 代表星期一，6 代表星期日
tm_yday	一年中的第几天，取值区间为 [0,366]（包含闰年）
tm_isdst	夏令时标志，0 表示非夏令时，1 表示夏令时，-1 表示未知

【例 12.7】（example12_07.py）使用 localtime( ) 函数获取本地时间。

```
import time
print(" 当前时间：",time.localtime())
print("0 时间戳对应的时间：",time.localtime(0))
```

代码运行结果：

```
当前时间：time.struct_time(tm_year=2024, tm_mon=1, tm_mday=6, tm_hour=0, tm_min=35, tm_sec=35, tm_wday=5, tm_yday=6, tm_isdst=0)
0 时间戳对应的时间：time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=8, tm_min=0, tm_sec=0, tm_wday=3, tm_yday=1, tm_isdst=0)
```

从这个例子的运行结果可以看到，localtime( ) 函数返回了 struct_time 类型，并且可以验证时间戳是从 1970 年 01 月 01 日 08 时 00 分 00 秒开始的（北京时间）。

（3）asctime( ) 函数。asctime( ) 函数能接受时间元组并返回一个可读的形式为“Sat Jan 6 00:59:26 2024”（2024 年 1 月 6 日星期六 00 时 59 分 26 秒）的 24 个字符的字符串。asctime( ) 函数接收的参数可以是一个包含 9 个元素的元组，也可以是一个通过函数 localtime( ) 返回 struct_time 类型的时间值。

【例 12.8】（example12_08.py）使用 asctime( ) 函数输出时间。

```
import time
t = (2024,1,6,0,59,26,5,6,0)
print("time.asctime(t):",time.asctime(t))
print("time.asctime(time.localtime()):",time.asctime(time.localtime()))
```

代码运行结果：

```
time.asctime(t): Sat Jan  6 00:59:26 2024
time.asctime(time.localtime()): Sat Jan  6 01:01:55 2024
```

（4）ctime( ) 函数。ctime( ) 函数能把一个时间戳（按秒计算的浮点数）转换为可读的字符串形式，其格式与 asctime( ) 函数的输出相同。如果参数未给或者值为 None，ctime( ) 函数将会默认 time.time( ) 的返回值作为参数，其效果等同于执行 asctime(time.localtime( ))。

【例 12.9】(example12_09.py）使用 ctime( ) 函数输出时间。

```
import time
print("time.ctime(): %s" % time.ctime())
print("time.ctime(0): %s" % time.ctime(0))
```

代码运行结果：

```
time.ctime(): Sat Jan  6 01:19:30 2024
time.ctime(0): Thu Jan  1 08:00:00 1970
```

（5）sleep( ) 函数。time 模块的 sleep( ) 函数用于推迟调用线程的运行，可通过参数指定推迟执行的秒数，也表示进程挂起的时间。

【例 12.10】(example12_10.py）使用 sleep( ) 函数挂起进程时间。

```
import time
print("Start: %s" % time.ctime())
time.sleep(9)
print("End: %s" % time.ctime())
```

代码运行结果：

```
Start: Sat Jan  6 01:31:29 2024
End: Sat Jan  6 01:31:38 2024
```

从这个例子的执行结果可以看到，开始时间和结束时间正好相差 9 秒，和示例程序中的 "time.sleep(9)" 参数相同。

（6）strftime ( ) 函数。strftime ( ) 函数用于接收时间元组，并返回以可读字符串表示的当地时间，格式由参数 format 决定。

【例 12.11】(example12_11.py）使用 strftime ( ) 函数将日期时间格式化。

```
import time
time1 = time.strftime("%Y-%m-%d %X", time.localtime())
time2 = time.strftime("%a %b %d %X %Y", time.localtime())
print(time1)
print(time2)
```

代码运行结果：

```
2024-01-06 01:59:16
Sat Jan 06 01:59:16 2024
```

任务 2　使用 Python 的第三方库

微课 12-2

如果把 Python 看作一个手机，那么第三方库相当于手机里各种各样的 App。当想将数据进行可视化时，可以选择功能全面的 Matplotlib；当想对中文文本进行分词处理时，可以使用 jieba 库；当想做文本分析，提取文本关键词时，可以选择 wordcloud；当想做一些图片处理工作时，可以使用 PIL 库。这些都是很成熟的第三方库。

Python 语言的第三方库是指那些不在 Python 安装包中的函数库，即非标准库。这类库一般是由全球各领域的专业人士结合专业特点和兴趣爱好开发出来的辅助库。使用第三方库之前，需要先在 Python 中安装相应的库文件，类似于在手机里安装 App。

本任务的主要目标是学习 Python 常用的两个第三方库：jieba 库和 wordcloud 库。通过这两个库的学习，我们将了解如何安装第三方库，掌握不同第三方库的功能和调用方法，从而提高自主学习能力。

1. jieba 库

随着人工智能技术的发展，中文文本分析与处理也越来越受关注。中文是紧凑连接的，词与词之间不用空格来分割，而当进行自然语言处理或者语音识别，或者进行其他的文本操作时，词汇是进行文本处理的前提，因此需要一个简单高效的工具把完整的文本分解为具体的词，这样的操作称为“分词”。jieba 库是一个对新手非常友好，能够进行分词的工具，而且还包含了一些其他的功能。

jieba 库是优秀的中文分词第三方库，提供了多种分词模式。jieba 库的分词原理是：利用一个中文词库，确定汉字之间的关联概率，汉字间概率大的组成词组，形成分词结果。除了分词，用户还可以添加自定义的词组。

1）jieba 库的安装

（1）使用基本命令自动安装。

```
pip install jieba
```

（2）使用清华源的镜像快速安装。

```
pip install jieba –i https://pypi.tuna.tsinghua.edu.cn/simple/
```

jieba 库安装界面如图 12–4 所示。

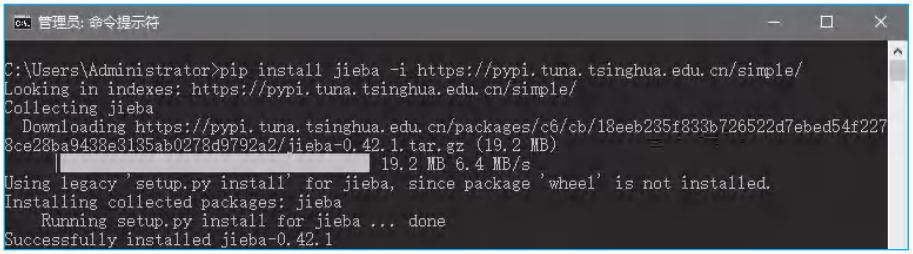


图 12–4　jieba 库安装界面

2）jieba 库的使用

jieba 分词提供了三种主要模式。

（1）精确模式。把一段文本精确地切分成若干个中文单词。若干个中文单词之间经过组合，就精确地还原为之前的文本，其中不存在冗余单词，返回结果是列表类型。

【例 12.12】(example12_12.py) jieba 精确模式分词。

```
import jieba
s=" 全国计算机等级考试 Python 科目 "
print(" 精确模式：",jieba.lcut(s))
```

代码运行结果：

```
精确模式：[' 全国 ', ' 计算机 ', ' 等级 ', ' 考试 ', 'Python', ' 科目 ']
```

（2）全模式。这种模式会将一段文本中所有可能的词语都扫描出来。在全模式下，jieba 会尝试从不同的角度对文本进行切分，生成所有可能的词语组合。因此，全模式下分词后的信息再组合起来会有冗余且冗余性最大，不再是原来的文本。

【例 12.13】(example12_13.py) jieba 全模式分词。

```
import jieba
s=" 全国计算机等级考试 Python 科目 "
print(" 全模式：",jieba.lcut(s,cut_all=True))
```

代码运行结果：

```
全模式：[' 全国 ', ' 国计 ', ' 计算 ', ' 计算机 ', ' 算机 ', ' 等级 ', ' 考试 ', 'Python', ' 科目 ']
```

（3）搜索引擎模式。搜索引擎模式是在精确模式的基础上，对长词语进一步切分。这种模式特别适合搜索引擎对短词语的索引和搜索，但仍有冗余。

【例 12.14】(example12_14.py) jieba 搜索引擎模式分词。

```
import jieba
s=" 全国计算机等级考试 Python 科目 "
print(" 全模式：",jieba.lcut_for_search(s))
```

代码运行结果：

```
全模式：[' 全国 ', ' 计算 ', ' 算机 ', ' 计算机 ', ' 等级 ', ' 考试 ', 'Python', ' 科目 ']
```

从三种模式对同一字符串语句运行的结果可以发现：精确模式试图将句子最精确地分开，适合文本分析；全模式把句子中所有可能的词语都扫描出来，速度非常快，但是不能解决歧义；搜索引擎模式在精确模式的基础上，对长词语再次进行切分，提高召回率，适合搜索引擎分词。

jieba 库常用的函数及其功能描述如表 12–4 所示。

表 12–4　jieba 库常用的函数及其功能描述

函　　数	描　　述
jieba.cut(s)	精确模式，返回一个可迭代的数据类型
jieba.cut(s, cut_all=True)	全模式，输出文本 s 中所有可能的单词
jieba.cut_for_search(s)	搜索引擎模式，适合搜索引擎建立索引的分词结果
jieba.lcut(s)	精确模式，返回一个列表类型，建议使用
jieba.lcut(s, cut_all=True)	全模式，返回一个列表类型，建议使用
jieba.lcut_for_search(s)	搜索引擎模式，返回一个列表类型，建议使用
jieba.add_word(w)	向分词词典中添加新词 w

3）jieba 库词库的添加与删除

在 jieba 库中，词库是指用于分词的词典，jieba 库提供了内置的词典和用户自定义的词典。用户可以通过添加或删除词语来定制自己的词典。其内置的词典在 jieba 安装默认文件夹 jieba 中的 dict.txt 文件，如图 12-5 所示。

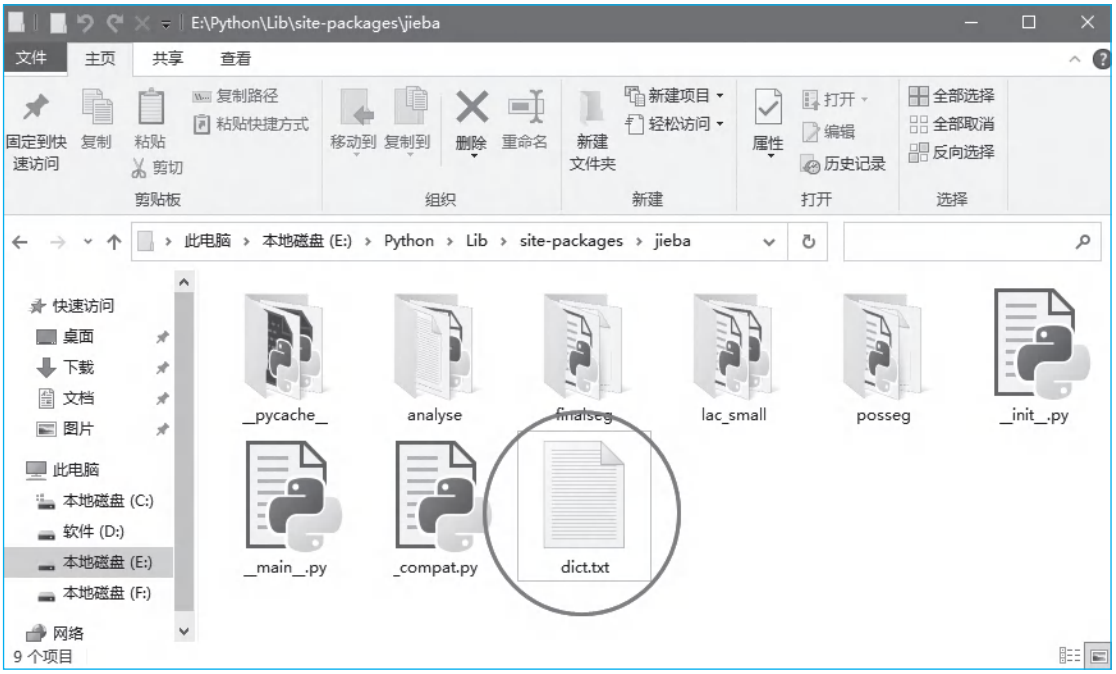


图 12-5　jieba 库词典所在文件

（1）添加单个词语。

在 jieba 库中，可以通过调用 add_word(word, freq=None, tag=None) 方法来向词库中添加单个词语。其中，word 为需要添加的词语，freq 为该词语的词频，tag 为该词语的词性。

【例 12.15】(example12_15.py) jieba 词库添加单个词语。

```
import jieba
s=" 全国计算机等级考试 Python 科目 "
print(" 精确模式：",jieba.lcut(s))
jieba.add_word(" 等级考试 ")
print(" 添加词语后精确模式：",jieba.lcut(s))
```

代码运行结果：

```
精确模式：[' 全国 ', ' 计算机 ', ' 等级 ', ' 考试 ', 'Python', ' 科目 ']
添加词语后精确模式：[' 全国 ', ' 计算机 ', ' 等级考试 ', 'Python', ' 科目 ']
```

从例题中可以看出，在将“等级考试”作为一个词语添加进词库中后，精确模式分词中就将“等级考试”归为一个词了。

（2）添加自定义词典。

当需要添加的词过多时，建议使用添加词典的方式。自定义词典可以包含用户自己添加的词语及其词频和词性等信息。添加自定义词典的方法如下。

创建一个文本文件，例如 userdict.txt，用于存储自定义词典。每行格式为：

```
词语 词频 词性
```

将需要添加的词语、词频和词性等信息写入到 userdict.txt 中，每个词语一行。调用 jieba 的 load_userdict( ) 方法加载自定义词典文件。

例如，添加以下词典：

```
等级考试 10 n
Python 科目 8 n
```

其中，10 和 8 为词语的词频，n 为词语的词性。之后调用词典即可。

【例 12.16】(example12_16.py) jieba 词库添加词典。

```
import jieba
s=" 全国计算机等级考试 Python 科目 "
print(" 精确模式：",jieba.lcut(s))
jieba.load_userdict("F:\\userdict.txt")
print(" 添加词典后精确模式：",jieba.lcut(s))
```

代码运行结果：

```
精确模式：[' 全国 ', ' 计算机 ', ' 等级 ', ' 考试 ', 'Python', ' 科目 ']
添加词典后精确模式：[' 全国 ', ' 计算机 ', ' 等级考试 ', 'Python 科目 ']
```

（3）词库的删除。

在 jieba 中，可以通过调用 del_word(word) 方法来删除词库中的单个词语。其中，word 为需要删除的词语。刚刚在词典中添加了“等级考试”这个词，接下来将其删除。

【例 12.17】(example12_17.py) jieba 词库删除词语。

```
import jieba
s=" 全国计算机等级考试 Python 科目 "
print(" 精确模式：",jieba.lcut(s))
jieba.add_word(" 等级考试 ")
print(" 添加词语后精确模式：",jieba.lcut(s))
jieba.del_word(" 等级考试 ")
print(" 删除词语后精确模式：",jieba.lcut(s))
```

代码运行结果：

```
精确模式：[' 全国 ', ' 计算机 ', ' 等级 ', ' 考试 ', 'Python', ' 科目 ']
添加词语后精确模式：[' 全国 ', ' 计算机 ', ' 等级考试 ', 'Python', ' 科目 ']
删除词语后精确模式：[' 全国 ', ' 计算机 ', ' 等级 ', ' 考试 ', 'Python', ' 科目 ']
```

通过对比可以明显发现在初始的词库中“等级”和“考试”作为两个词语出现，后通过添加词语将其组成“等级考试”一词，最后通过删除词语操作又将其恢复成两个词语分词。

2. wordcloud 库

Python 的 wordcloud 库是非常优秀的词云可视化第三方库。词云是指对文本中出现频率较高的关键词汇通过彩色图形渲染，从而在视觉上予以突出。它不仅是设计与统计的结合，也是艺术与计算机科学的碰撞。词云以词语为基本单元，根据其在文本中出现的频率设计不同大小的字体，从而形成视觉上的不同效果：形成“关键云层”或“关键词渲染”，使得读者可以通过“一瞥”快速领略文本的主旨。这种展示方式已经成为文本展示的样板。词云可视化已在教育和文化等方面得到了

广泛应用。

1）wordcloud 库的安装

（1）使用基本命令自动安装。

```
pip install wordcloud
```

（2）使用清华源的镜像快速安装。

```
pip install wordcloud –i https://pypi.tuna.tsinghua.edu.cn/simple/
```

jieba 库安装界面如图 12–6 所示。

图 12–6　jieba 库安装界面

2）wordcloud 库的使用

wordcloud 库的核心是 WordCloud 类，所有功能都封装在其中。使用时，需要实例化一个 WordCloud 类的对象。使用 wordcloud.WordCloud(参数) 方法可以创建 WordCloud 对象（注意字母大小写）。

WordCloud 在创建时有一系列可选参数，用于配置词云图片。WordCloud 对象创建的常用参数及功能描述如表 12–5 所示。

表 12–5　WordCloud 对象创建的常用参数及功能描述

参　　数	描　　述
font_path	指定字体文件的完整路径，默认 None
width	生成图片宽度，默认为 400 像素
height	生成图片高度，默认为 200 像素
mask	词云形状，默认为 None，即方形图
min_font_size	词云中最小的字体号，默认为 4 号
font_step	字号步进间隔，默认为 1
max_font_size	词云中最大的字体字号，默认 None，根据高度自动调节
max_words	词云图中最大词数，默认为 200
stopwords	被排除词列表，排除词不在词云中显示

续表

参　　数	描　　述
background_color	图片背景色，默认黑色

在生成词云时，wordcloud 库默认会以空格或标点分隔符对目标文本进行分词处理。对于英文文本或者内容，因为英文本身就是以空格作为分隔符，无须专门进行分词处理，所以可以直接使用 wordcloud 库生成词云。

【例 12.18】(example12_18.py) 使用英文文本生成词云。

```
import wordcloud
txt = "I love Python,I need a book to help me study Python."
w = wordcloud.WordCloud(background_color="white")
w.generate(txt)
w.to_file(" 英文文本 .png")
```

代码运行结果如图 12–7 所示。

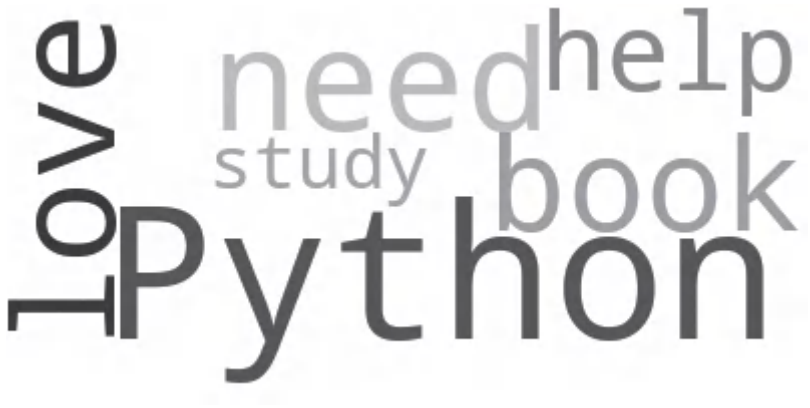


图 12–7　使用英文文本生成词云

这里使用了 WordCloud 类中的 background_color 参数，将词云背景更改成了白色，其他参数保持默认，w.generate(txt) 表示根据 txt 文本生成词云，w.to_file(" 英文文本 .png") 表示将生成的词云图写入到名为“英文文本”的 png 格式的文件中。除了在代码中直接输入文本内容，也可以读取已经存在的文本文件，并将其制作成词云图片。

对于中文格式的文本内容，分词处理还需要由用户手动来完成，因为中文单词之间没有空格分隔。这里就要用到上一节学过的 jieba 库。一般来说，jieba 库和 wordcloud 库的使用是紧密结合的。一般步骤是先将文本分词处理，然后以空格拼接，最后再调用 wordcloud 库函数，并且处理中文时还需要指定中文字体。例如，如果选择了微软雅黑（msyh.ttc，该文件在 C:\Windows\Fonts 文件夹下）作为显示效果，那么需要将该字体文件与代码存放在同一目录下或在字体文件名前加上完整路径。

【例 12.19】(example12_19.py) 使用中文文本生成词云。

本例读取一个名为“爱国故事 .txt”的文本文档，内容为：1910 年春天，12 岁的周恩来进银冈书院读书。一次，校长在课堂上问大家为什么读书时，周恩来慷慨答道：“为了中华之崛起！”这一誓言此后贯彻在他的一生中，他为中华民族解放事业做出了巨大的贡献。

同学们在参照编写代码时可以先行新建好该文本文档。

```
import wordcloud
import jieba
```

```
w = wordcloud.WordCloud(background_color="white",
                        font_path="msyh.ttc",
                        height=300,width=600)
# 读取文本文件内容
txt = open(" 爱国故事 .txt",'r',encoding="utf-8").read()
# 使用 jieba 库进行中文分词，并将分词结果用空格连接起来
words = " ".join(jieba.lcut(txt))
w.generate(words)
w.to_file(" 中文文本 .png")
```

代码运行结果如图 12-8 所示。

图 12-8　使用中文文本生成词云

✪课后加油站

有趣的第三方库——Pywebview

Python 语法简洁、功能强大，可以完成很多任务，原因就是有强大的库支持，有很多很多现成的代码可以用，用户只要负责搭建应用即可。这里介绍一个有趣的第三方库供同学们参考学习——Pywebview。

Pywebview：用于以图形用户界面（GUI）的形式显示 HTML、CSS 和 JavaScript 内容的库，可以将网页或网站转换为桌面应用程序。Pywebview 库是一个强大的 Python 库，它提供了一个简单易用的应用程序编程接口（API），用于在 Python 应用程序中嵌入 Web 浏览器。

1. 功能特性

（1）跨平台支持：Pywebview 支持 Windows、macOS 和 Linux 等主流操作系统，确保代码的可移植性。

（2）简洁的 API：其 API 设计非常简洁，几行代码就可以创建一个基本的 Web 浏览器窗口，降低了学习成本。

（3）与 Python 的紧密集成：在 HTML 页面中可以直接调用 Python 函数，Python 代码中也可以处理 JavaScript 事件，实现前后端的双向通信。

（4）自定义与扩展性：允许自定义浏览器窗口的外观和行为，如设置窗口大小、标题、图标等，同时支持通过 JavaScript API 扩展功能。

2. 使用场景

（1）桌面应用开发：为 Python 桌面应用增加现代化的 Web 界面，使界面更加直观。

（2）快速原型设计：快速验证 Web 应用的概念，无须构建完整的后端。

（3）自动化测试：在自动化测试中模拟用户交互，如自动登录或页面操作。

3. 安装

安装 Pywebview 库非常简单，只需通过 pip 命令进行安装：

```
pip install pywebview
```

4. 示例代码

下面是简单的示例代码，用于创建一个带有特定 URL 的窗口：

```
import webview
w=webview.WebView(width=320,height=240,title="Hello", url="https://www.example.com", resizable=True, debug=False)
w.run()
```

通过 Pywebview 库，开发者可以利用熟悉的 Web 技术（HTML、CSS、JavaScript）来构建用户界面，同时结合 Python 的强大功能实现后端逻辑。这种结合方式不仅能够实现跨平台的桌面应用开发，还能显著提高开发效率和用户体验。

巩固与练习

一、选择题

1. 表示“海龟前进”的方法是（　　）。

A. turtle.penup( )

B. turtle.pendown( )

C. turtle.forword(d)

D. turtle.backward(d)

2. 下列选项中，修改 turtle 画笔颜色的函数是（　　）。

A. pencolor( )　　B. speed( )　　C. pensize( )　　D. seth( )

3. turtle.circle(50, 180) 的执行效果是（　　）。

A. 绘制一个半径为 50 的圆

B. 绘制一个半径为 50 的半圆

C. 绘制一个半径为 50 的圆，分三次画完

D. 绘制一个半径为 50 的圆形

4. random.uniform(a,b) 的作用是（　　）。

A. 生成一个 [a,b] 之间的随机整数

B. 生成一个 [a,b] 之间的随机小数

C. 生成一个均值为 a，方差为 b 的正态分布

D. 生成一个 (a,b) 之间的随机数

5. jieba 库函数 jieba.lcut( ) 返回值的类型是（　　）。

A. 列表

B. 迭代器

C. 字符串

D. 元组

6. 设置下列（　　）属性，可以使 wordcloud 支持中文。

A. font_step

B. font_path

C. mode

D. font

7. WordCloud 类中能根据文本生成词云的方法是（　　）。

A. fit_words( )

B. process_text( )

C. generate_from_frequencies( )

D. generate( )

二、填空题

1. random 库中设置随机数种子的函数是_____。

2. time 库中使用函数是_____推迟调用线程的运行。

3. _____库是优秀的中文分词第三方库，能够将中文文本通过分词获得单个的词语。

4. _____库是非常优秀的词云可视化第三方库，词云是指对文本中出现频率较高的关键词汇通过彩色图形渲染，从而在视觉上予以突出。

5. _____ jieba 库的分词模式分为_____、_____、_____三种模式。

三、编程题

1. 编写程序绘制平行四边形，并填充颜色为黄色，效果如图 12–9 所示。

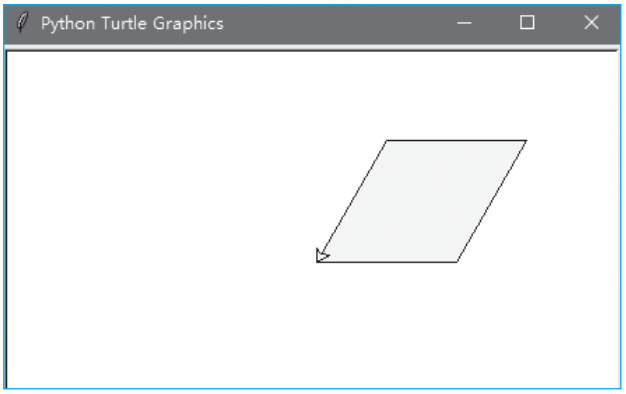


图 12–9　平行四边形绘制效果

2.《中华人民共和国宪法》是中华人民共和国的根本法，具有最高法律效力。请同学们将《宪法》序言中部分内容“本宪法以法律的形式确认了中国各族人民奋斗的成果，规定了国家的根本制度和根本任务，是国家的根本法，具有最高的法律效力。全国各族人民、一切国家机关和武装力量、各政党和各社会团体、各企业事业组织，都必须以宪法为根本的活动准则，并且负有维护宪法尊严、保证宪法实施的职责。”保存成名为《序言 – 部分》的文本文件，再利用 wordcloud 库生成背景颜色为黄色、字体为微软雅黑、图片宽度和高度分别为 600 和 400 的词云，效果如图 12–10 所示。

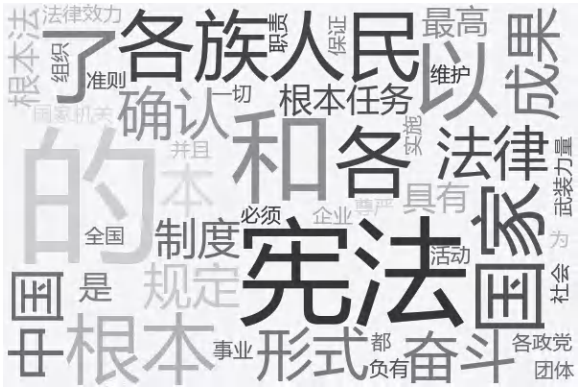


图 12-10　词云效果

☆学·做·思

1. 做一份知识归纳图表，说明 Python 标准库和第三方库的概念和使用方法，并谈谈 Python 常用库的学习和使用会给自己的未来工作带来哪些便利。

思维导图：

思考总结：

2. 通过课本、书籍或者互联网进行自主学习，了解 Python 更多其他常用的标准库或第三方库，挑选一个，进行自学后撰写总结报告，并思考：如何进行 Python 库的学习？

总结报告：

思考拓展：