第 1 章

Windows 环境下开发环境的搭建与编码规范

Python 支持多种操作系统，本书以 Windows10（64 位）为平台，编程工具可以使用纯文本编辑环境（如 Windows 记事本、TextPad 等），或集成开发环境（如 IDLE、PyCharm、Spyder、Eclipse 等）。

1.1 开发环境搭建与使用

1.1.1 Python 及 Anaconda 的下载与安装

Python 安装很简单，打开 Python 官方网站主页 https://www.python.org/，如图 1-1 所示。

图 1-1　Python 官方网站主页

单击 Downloads 下的 Windows，进入下载页面，如图 1-2 所示，选择适合自己的版本下载安装。

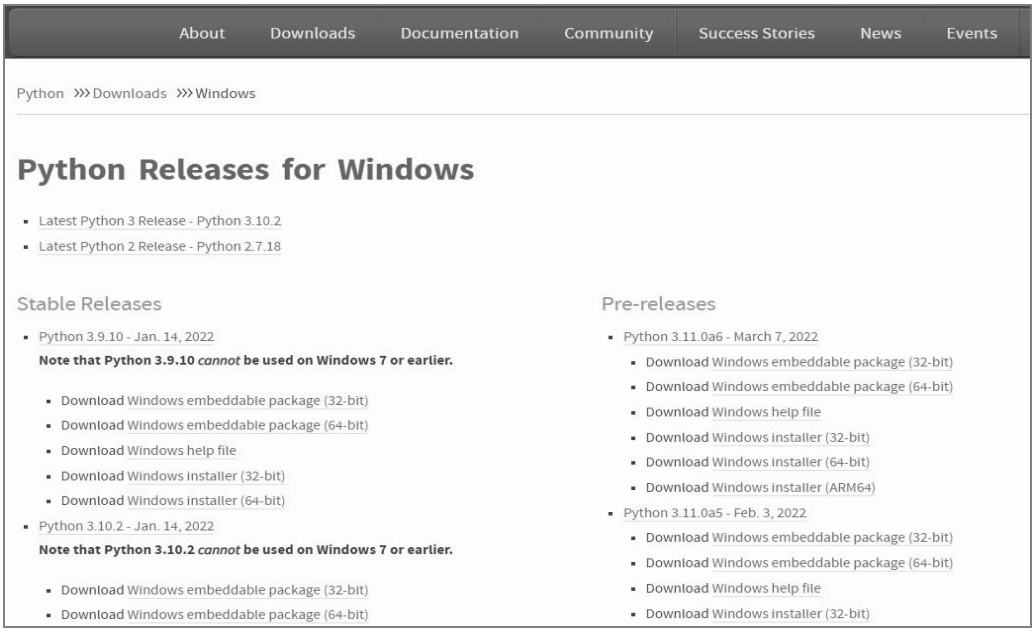

图 1-2　Python 官网下载页面

例如选择 Python 3.10.2 下载完成后，会得到一个名为 python-3.10.2-amd64.exe 的文件，然后运行该文件安装即可。

也可以根据个人爱好，选择下载安装 Anaconda 安装包，该安装包集成了大量常用的 Python 扩展库，大幅度节约了用户配置 Python 开发环境的时间。首先，登录 Anaconda 官网 https://www.anaconda.com/products/individual，如图 1-3 所示。

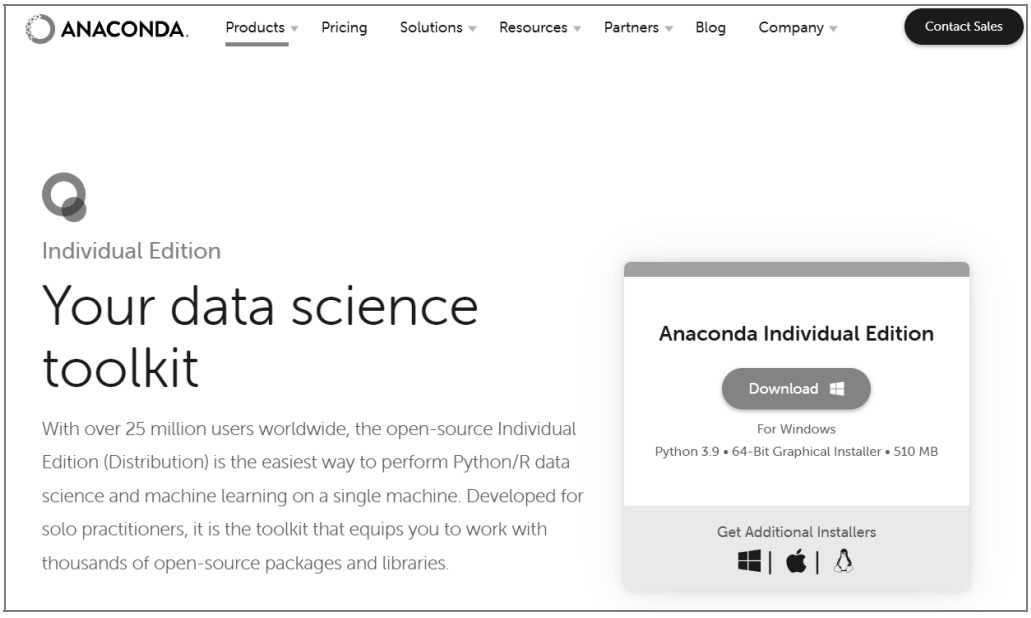

图 1-3　Anaconda 官网下载页面

其次，单击 Download 下载，得到一个名为 Anaconda3-2021.11-Windows-x86_64.exe 的文件，运行安装即可。安装运行完之后，Windows 开始菜单中会增加如图 1-4 所示的菜单。

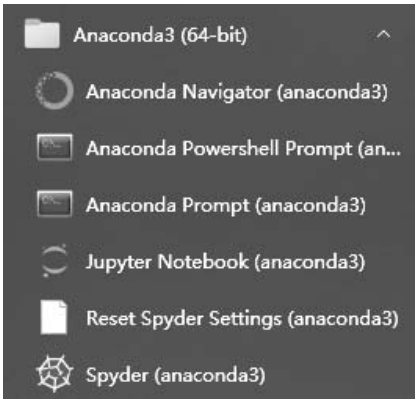

图 1-4　Anaconda 菜单

1.1.2　PyCharm 的安装与使用

PyCharm 是由 JetBrains 公司开发的 Python 集成开发环境，其带有一整套可以帮助用户在使用 Python 语言开发时提高效率的工具，具有调试、语法高亮、项目管理、代码跳转、智能提示、自动完成、单元测试等功能。此外，该集成开发环境还提供了一些高级功能，用于支持 Django 框架下的专业 Web 开发，目前已成为 Python 专业开发人员和初学者最常用的工具。

1. PyCharm 下载

（1）登录网址 http://www.jetbrains.com，打开 JetBrains 的官网，选择 Developer Tools 进入 PyCharm 选项页面，如图 1-5 所示。

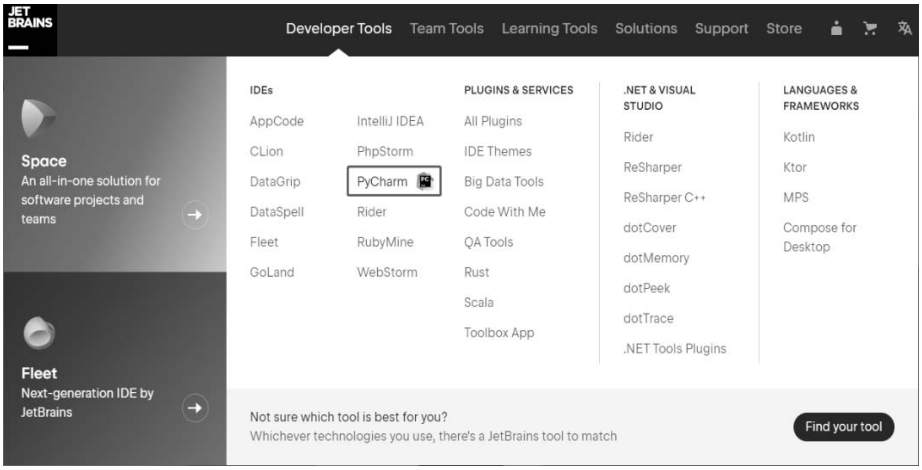

图 1-5　PyCharm 选项页面

（2）单击 PyCharm 项进入 PyCharm 下载页面，如图 1-6 所示。

图 1-6　PyCharm 下载页面

（3）单击 Download，进入平台与版本选择界面，如图 1-7 所示。

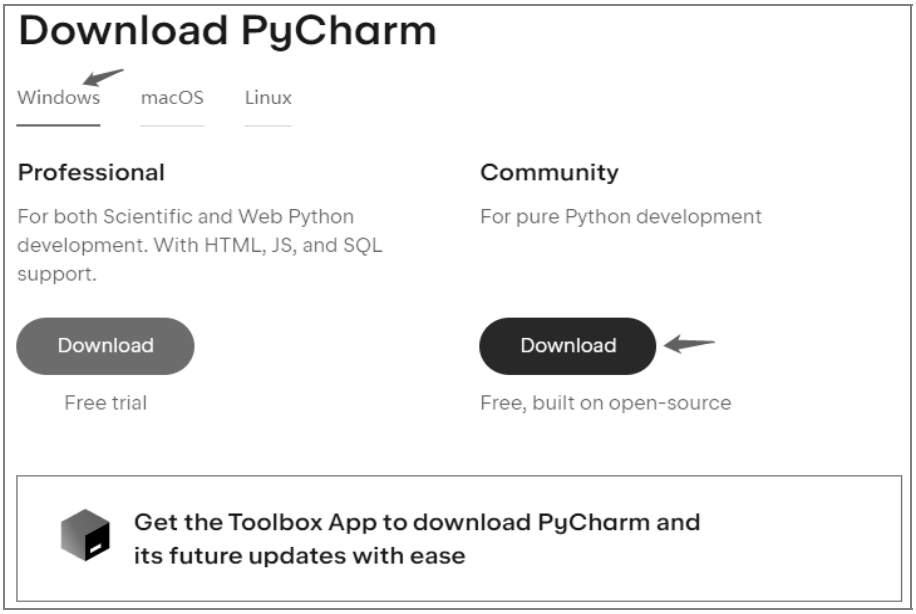

图 1-7　平台与版本选择界面

这里有两个版本可供选择，其中专业版（Professional）是功能最丰富的版本，与社区版（Community）相比，其增加了 Web 开发、Python Web 框架、Python 分析器、远程开发、支持数据库与 SQL 等高级功能。但专业版需要付费，而社区版则可以完全免费使用。

（4）单击 Community下的 Download，下载完成后，可得到一个名为 pycharm-community- 2022.1.1.exe 的文件。

2. PyCharm 安装

PyCharm 安装过程如下所示。

（1）双击运行安装文件，进入安装界面，如图 1-8 所示。

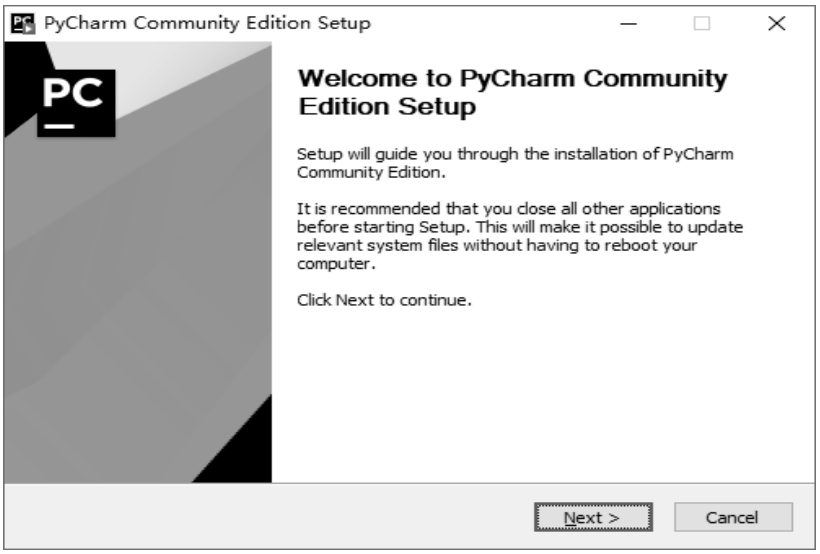

图 1-8　PyCharm 安装界面

（2）单击 Next，进入设置安装路径界面，设置完路径之后，单击 Next 按钮，进入设置快捷方式界面，如图 1-9 所示。

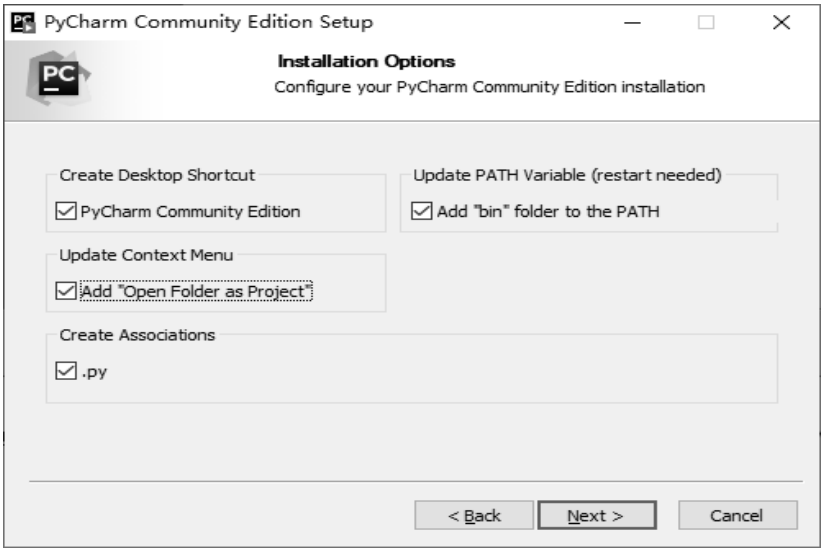

图 1-9　设置快捷方式界面

（3）单击 Next 按钮，进入开始菜单文件夹界面，如图 1-10 所示，该界面不用设置，采用默认值即可。单击 Install 按钮进行安装。

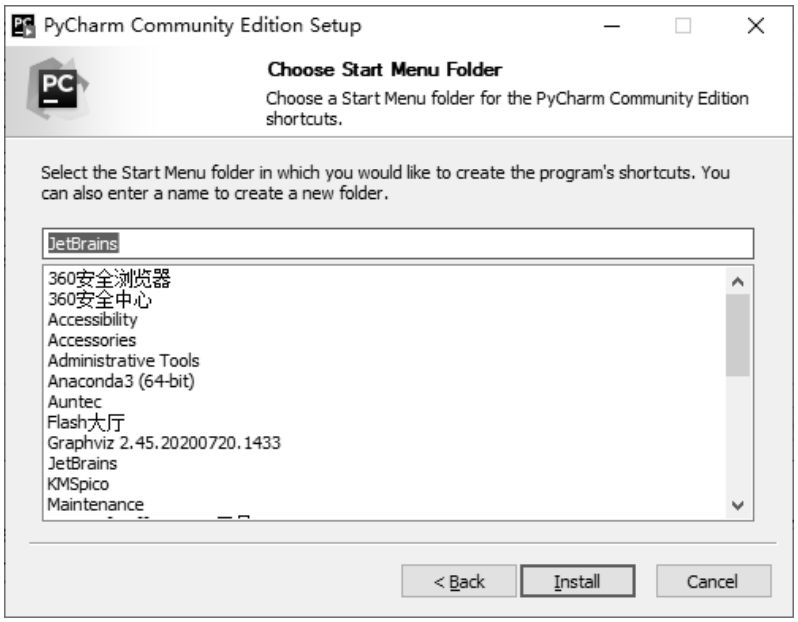

图 1-10　选择开始菜单文件夹界面

3. PyCharm 的使用

1）初次运行 PyCharm

通过菜单或桌面快捷方式，启动 PyCharm 应用程序，进入阅读协议界面，如图 1-11 所示。

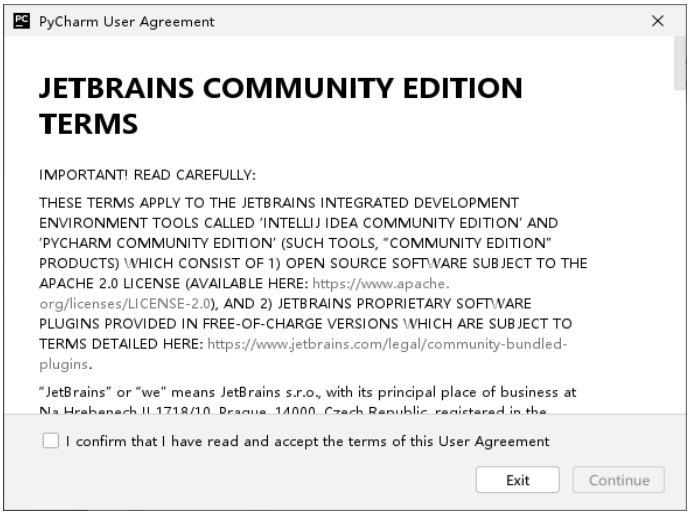

图 1-11　PyCharm 阅读协议界面

拖拽协议文本框的滚动条到文本框的最下面，表示已经阅读完协议，此时 Continue 按钮为灰色不可用，当选择复选框后，才能显示为可用。单击 Continue 按钮，进入用户 PyCharm 欢迎界面，此处可进行插件扩展安装等，如图 1-12 所示。

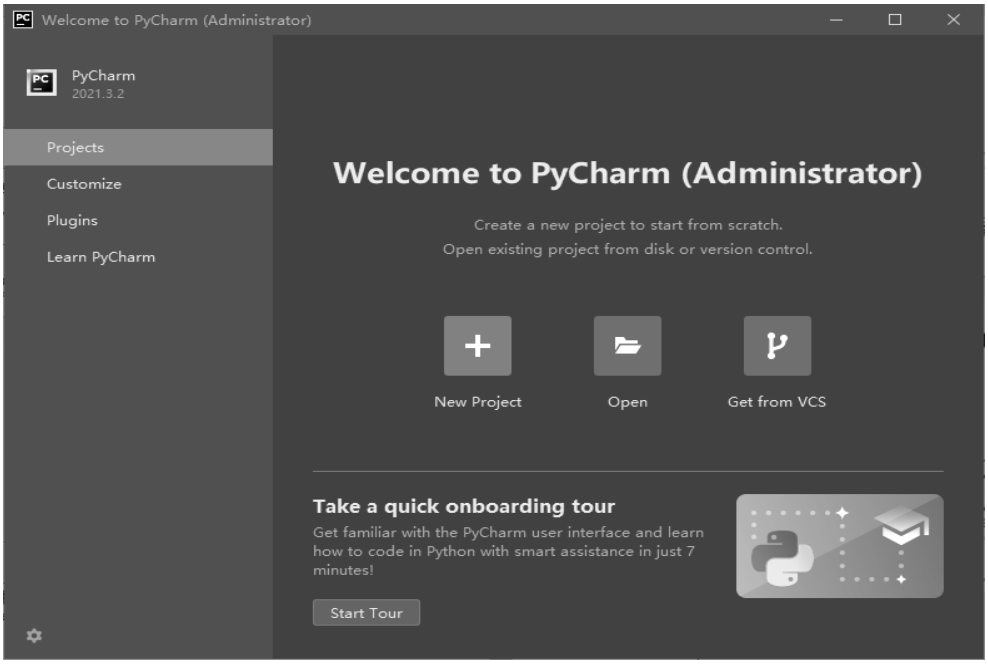

图 1-12　启动 PyCharm 欢迎界面

之后，可不做设置，直接单击 Start Tour 按钮，进入 PyCharm 设置解释器界面，如图 1-13 所示。

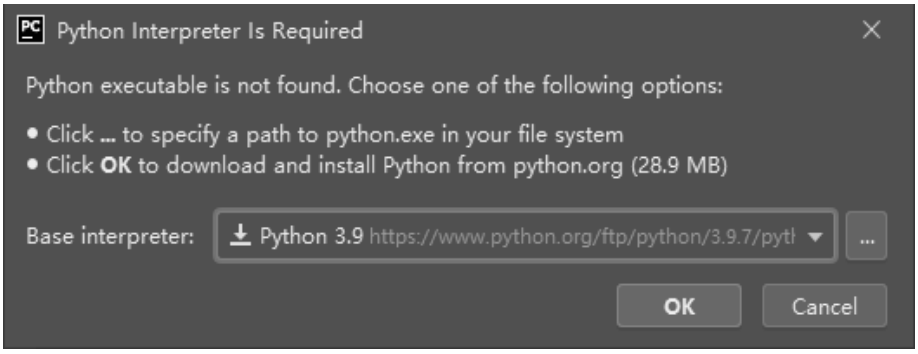

图 1-13　设置解释器界面

接着可为 PyCharm 设置运行程序时的解释器，无论之前安装 Python 还是 Anaconda，一般情况下，找到 Python.exe 文件的位置，选择该文件，单击 OK 按钮进入 PyCharm 开发环境，如图 1-14 所示。

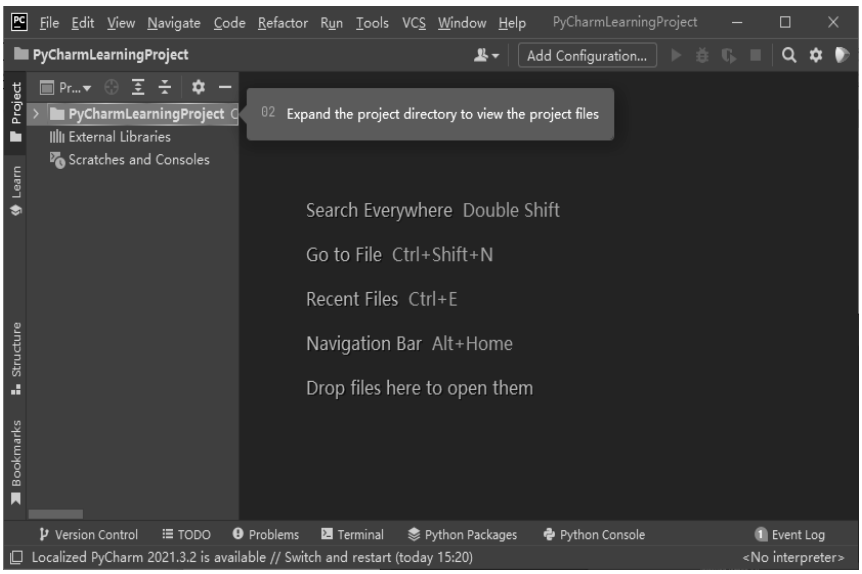

图 1-14　PyCharm 开发环境

安装完 PyCharm 之后，初次运行需要上述过程，之后启动 PyCharm，会自动打开上次关闭时的项目。

2）创建项目

进入 PyCharm 主界面，选择 File→New Project，创建一个新的项目，如图 1-15 所示。

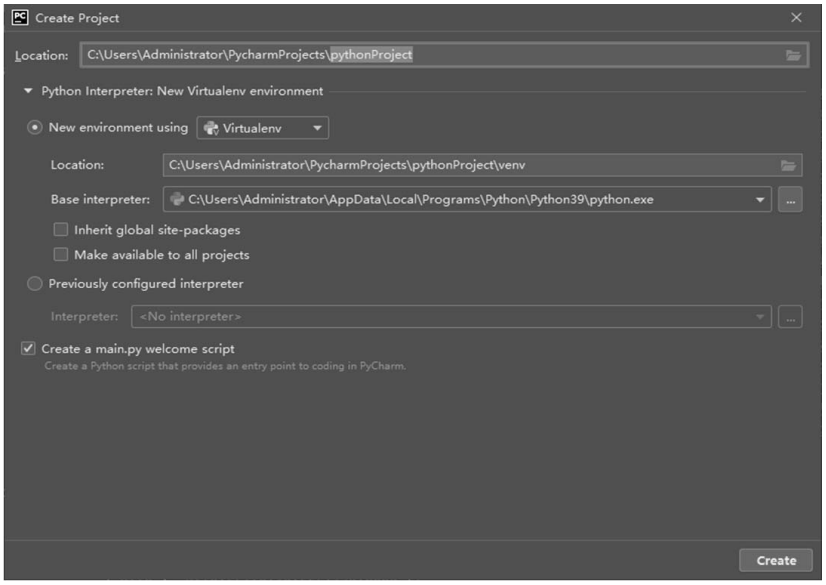

图 1-15　新建项目界面

该窗口中，设置容易管理的存储路径或创建新的路径，之后单击 Create 按钮，便可成功创建新的项目，如图 1-16 所示。

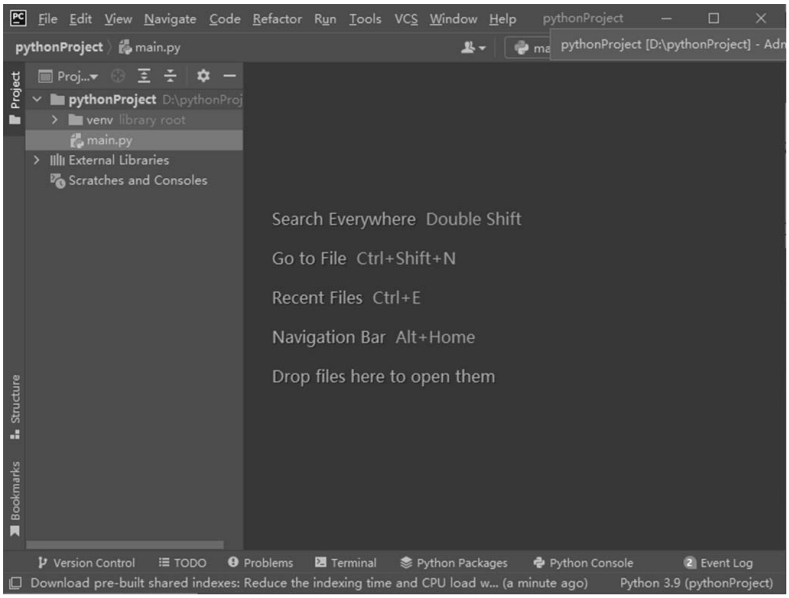

图 1-16　新建 PyCharm 窗口

3）编写“hello word!”程序

右击新建立的项目名称，在弹出的快捷菜单中选择 New→Python File 选项，如图 1-17 所示。

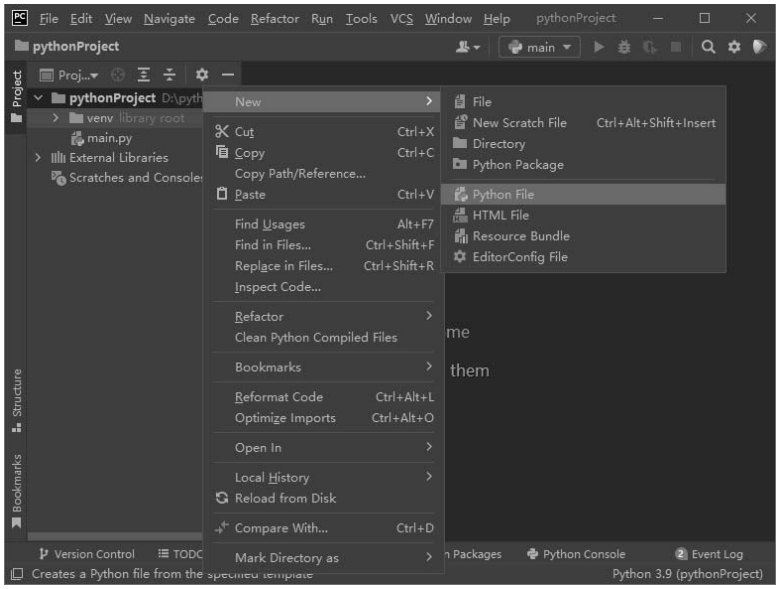

图 1-17　新建 Python 文件

之后在弹出的小窗口中输入文件名，进入编辑区，输入语句：print('hello,world!')，右击编辑区，在弹出的快捷菜单中选择 Run helloworld(此处 helloworld 为生成文件，是输入的文件名)，在窗口下部 Run 窗口内可输出结果：hello, world! 如图 1-18 所示。

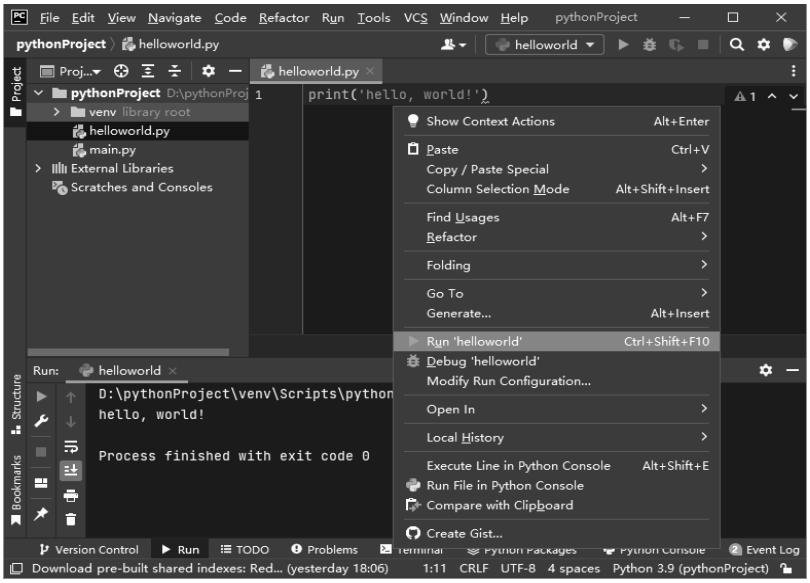

图 1-18　PyCharm 编辑、运行 Python 程序

1.2　扩展库的安装

1.2.1　PyCharm 开发环境里扩展库的安装

选择 File→Setting，在弹出的设置窗口中选择 Project→Python interpreter 选项，此时设置窗口如图 1-19 所示。

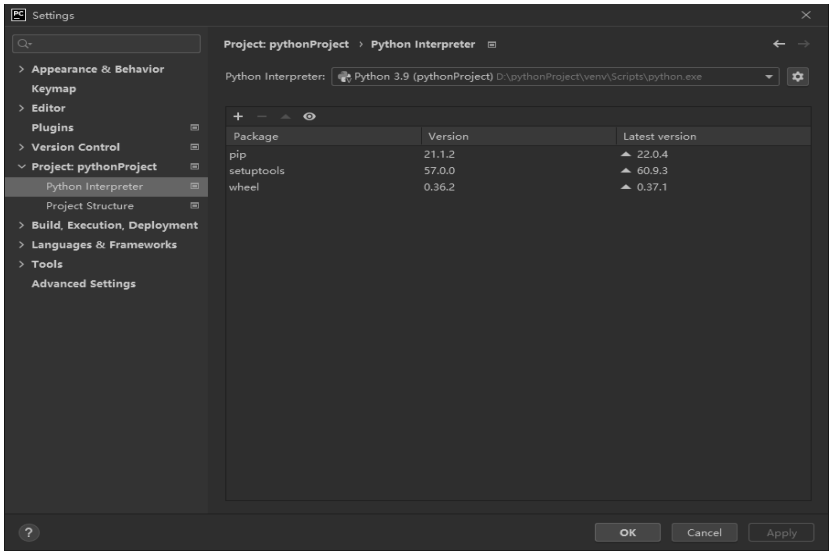

图 1-19　PyCharm 设置窗口

单击“+”号，在弹出的窗口上部，输入要安装的扩展库名，然后单击下面的 Install Package 按钮，安装所需扩展库，如图 1-20 所示（此处以安装 requests 扩展库为例）。

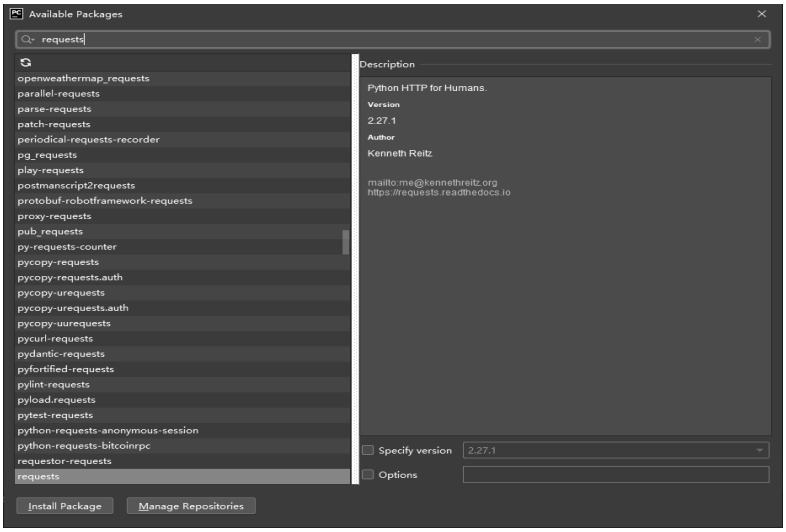

图 1-20　PyCharm 开发环境安装扩展库窗口

1.2.2　PyCharm 开发环境 Terminal 终端下快速安装

选择 File→Setting→Python Interpreter 安装扩展库，这种方法速度相对较慢。可在 PyCharm 中的 Terminal 中运行 pip install <扩展库名称>快速安装。单击窗口底部的 Terminal 按钮，从键盘输入命令 pip install requests（此处以安装 requests 扩展库为例），回车快速安装，如图 1-21 所示。

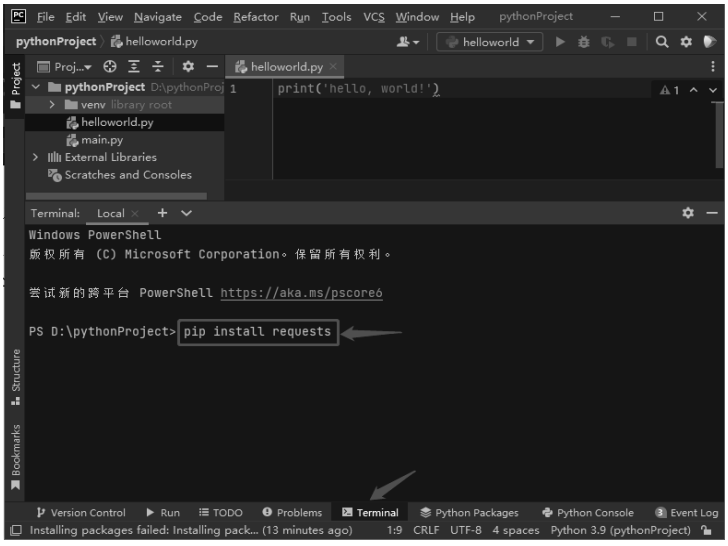

图 1-21　PyCharm 的 Terminal 终端安装扩展库

1.2.3 Anaconda 环境中安装扩展库

选择 Start→Anaconda3→Anaconda Prompt，在打开窗口的命令行中输入 pip install <扩展库名称>即可，如图 1-22 所示（此处以安装 selenium 扩展库为例）。

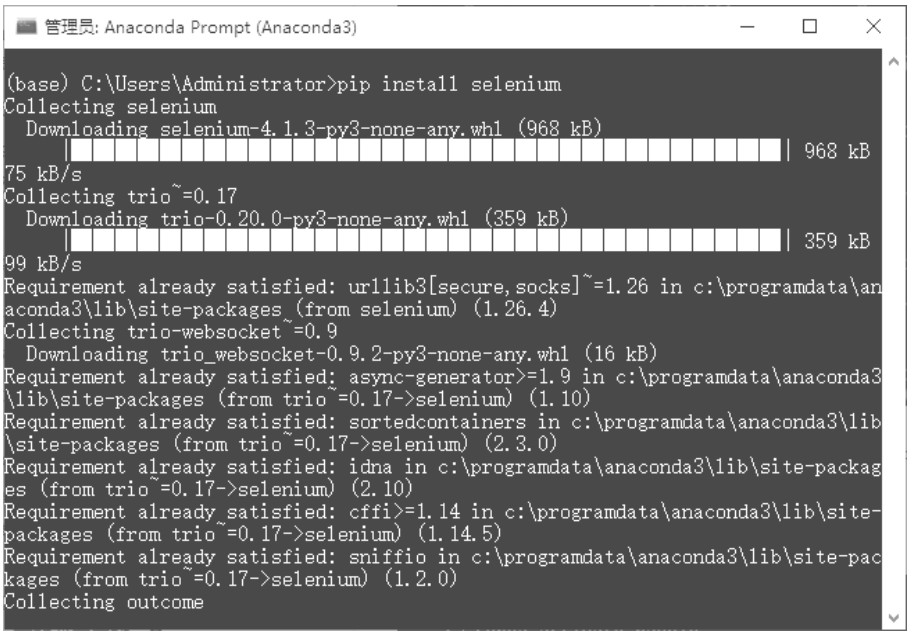

图 1-22　Anaconda Prompt 窗口安装扩展库

1.3　标准库与扩展库中对象的导入

Python 默认安装仅包含基本的核心模块，启动时也仅加载了基本模块，在需要时再显式地导入和加载标准库和第三方扩展库，这样可以减少程序运行的压力，并且具有很强的可扩展性，也有助于提高系统的安全性。

1.3.1 import 模块名 [as 别名] 导入

使用 import 模块名[as 别名]这种方式导入以后，使用时需要在对象之前加上模块名作为前缀，必须以“模块名.对象名”的形式访问。如果模块名很长的话，在导入的时候为模块设置一个简短的别名，然后用“别名.对象名”的形式来使用其中的对象。

例如，需要利用标准库 math 中的 sqrt()求整数的算术平方根，执行如下代码：

```
import math
math.sqrt(20)
```

再如，利用 numpy 扩展库生成一维数组，执行如下代码：

```
import numpy
arr = numpy.array([4, 6, 8, 9])
```

在导入 numpy 的时候，常为其设置一个别名，之后使用的时候可以用别名，代码如下所示：

```
import numpy as np
arr = np.array([4, 6, 8, 9])
```

1.3.2　from 模块名 import 对象名 [as 别名] 导入

使用 from 模块名 import 对象名[as 别名]这种方式仅导入确定的某一个对象，可为该对象指定一个别名，这种导入方式可减少查询次数，提高访问速度，减少打包后文件大小，同时也可以减少代码输入量。

```
from math import sqrt
st = sqrt(20)           # 求 20 的算术平方根
print(st)
from random import randint
rt = randint(1, 10)     # 产生[1, 10]之间任意整数
print(rt)
```

1.3.3　from 模块名 import *

from 模块名 import *中的“*”代表模块中通过__all__变量指定的所有对象。这种方式简单直接，写起来省事，但一般不建议使用这种写法：一方面这样写会降低代码的可读性，有时候很难区分模块中导入的函数和自定义函数；另一方面这种导入方式会导致命名空间的混乱，如果多个模块中有同名的对象，则只有最后一个导入的模块中的对象是有效的，之前导入的模块中的同名对象都会被屏蔽，无法正常使用，无益于代码的编写与维护。

1.4　Python 编程规范及语法特点

任何一种语言都有一些约定俗成的编码规范，Python 也不例外，尤其在标识符命名、代码缩进、编码规则、结构布局、注释等方面。本节主要介绍 Python 社区对代码编写的一些共同的要求、规范和代码优化建议，初学者最好在一开始就遵循这些规则和建议，养成良好的编程习惯。

1.4.1　命名规则

命名规范在编写代码中起到很重要的作用，虽然不遵循命名规范，程序也可以运行，但是使用命名规范可以更加直观地了解代码所表示的含义，增强代码的可读性。有如下命名规则。

（1）模块名尽量短小，并且全部使用小写字母，可以使用下画线“__”分隔多个单词。

（2）类名采用单词首字母大写形式（Pascal 风格）。

（3）函数、类的属性和方法的命名规则同模块类似，也是全部用小写字母，多个单词之间用“__”分隔。

（4）常量命名全部用大写字母，可加下画线。

（5）变量命名必须用字母或下画线开始，但以下画线开始的变量在 Python 中有特殊的含义。变量名对英文字母的大小写是敏感的。

（6）任何命名都不能和 Python 保留字冲突，也不能使用类型名、函数名及已导入的模块名或其成员名作为变量名，这会改变其类型和含义，甚至会导致其他代码无法正常执行。可以通过 dir（__builtins__）命令查看所有的内置对象名。

1.4.2 代码缩进

Python 与其他程序设计语言不同，没有专门分隔代码块的符号，而是采用代码缩进加“:”来区分代码之间的层次。缩进量一般默认为四个空格符，或直接用 Tab 键代替。

在 Python 中，对于类定义、函数定义、流程控制语句，以及异常处理语句、上下文管理语句等，行尾的冒号和下一行的缩进表示一个代码块的开始，而缩进结束，则表示一个代码块的结束。Python 对代码的缩进要求严格，同一个级别的代码块的缩进量必须相同，如果不采用合理的代码缩进，程序将抛出 SyntaxError 异常，如图 1-23 所示。

```
def func_exam(seg):                                          缩进相同：函数体
    keyword = open(r'./data/keyword.txt', 'w+', encoding='utf-8')
    print("去停用词：\n")
    wordlist = []
    stop = open(r'./data/stopwords.txt', 'r+', encoding='utf-8')
    stop_word = stop.read().split("\n")
    for key in seg.split('/'):                               缩进相同：循环体
        if not(key.strip() in stop_word) and (len(key.strip()) > 1) and not(key.strip() in wordlist):
            wordlist.append(key)        缩进相同：if语句块
            print(key)
            keyword.write(key+"\n")
        print('这里循环结束!')
    stop.close()
    keyword.close()
    return ','.join(wordlist)


if __name__ == "__main__":
    print('this is an example!')
```

图 1-23　代码缩进示例

1.4.3 编码规范

Python 采用 PEP8（Python Enhancement Proposal）作为编码规范。其书写格式有着严格的要求，不按照格式书写有可能导致程序不能正确运行。在《Google 开源项目风格指南》中，还列出了常见的书写格式基本规则建议，虽然这些也许并不影响程序的执行，但良好的编程书写风格会显著提升程序的可读性。

（1）长语句行。除非特殊情况下（例如，导入模块过长或字符串中含有较多的内容等），一般不建议写过长的语句，如果确实需要长语句，考虑将语句拆分成多个短一些的

语句，以保证代码具有较好的可读性。如果语句确实太长超过了屏幕的宽度，最好使用续行符“\”将超长部分放到下一行。

（2）空行。顶级定义之间空两行；变量定义、类定义及函数定义之间，可以空两行；类内部的方法定义之间、类定义与第一个方法之间，建议空一行；一段完整功能的代码之后可以空一行。

（3）空格。对于赋值运算符（=）、关系运算符（==、<、>、!=、<>、<=、>=、in、is）、布尔运算符（and、or、not）等，在其两边加上一个空格符，会使代码更清晰；算术运算符等也建议两侧加空格，与其他运算符保持一致；逗号、冒号（函数参数之间、列表、元组元素之间、字典的键与值之间等情况）后也建议加一空格；当等号用于表示函数关键参数或参数默认值时，不建议在两侧加空格。

（4）文档字符串。文档字符串是 Python 语言独特的注释方式。文档字符串是包、模块、类或函数中的注释性文字，可通过对象的__doc__自动提取，书写的时候用三单引号或三双引号表示。例如，'''这是注释性的文档字符串'''或"""这是注释性的文档字符串"""。一个规范的文档字符串首先是一行概述，接着是一个空行，然后是文档字符串剩余部分。

（5）异常处理。异常指程序运行时引发的错误，错误的原因有很多，如除零、下标越界、文件不存在、网络连接失败等。即使水平再高的程序员也无法预见代码运行时可能会遇到的所有情况，更无法避免这些形形色色的错误。合理地使用异常处理结构可以使得程序更加健壮，具有更高的容错性，不会因为用户不小心的错误输入而造成程序崩溃，也可以使用异常处理结构为用户提供更加友好的提示。但不能过多依赖异常处理结构，适当结合显式判断也是很有必要的。

1.4.4　注释规则

注释指在代码中对代码功能进行解释说明的标注性文字，这有助于增强程序的可读性，Python 解释器会忽略注释的内容。Python 中通常包括 3 种注释类型，分别是单行注释、多行注释和中文编码声明注释。

1. 单行注释

在 Python 中，使用“#”作为单行注释的符号。从符号“#”开始直到本行结束为止，“#”后面所有的内容都作为注释的内容，并被 Python 解释器忽略。

单行注释可以放在要注释代码的前一行，也可以放在要注释代码的右侧。下面的两种注释形式都是正确的。

第一种形式：

```
# 输入股票代码，要求 6 位数字
stock_code = input('请输入股票代码:')
```

第二种形式：

```
stock_code = input('请输入股票代码:')# 输入股票代码，要求 6 位数字
```

说明：在添加注释时，一定要有意义，即注释能充分解释代码的功能及用途。注释可以出现在代码的任意位置，但是不能分割关键字和标识符。

2. 多行注释

在 Python 中，并没有一个单独的多行注释标记，而是将包含在一对三引号（'''...'''）或（"""..."""）之间的代码都称为多行注释，解释器将忽略这些内容。由于这样的代码可以分为多行，所以也称作为多行注释。其语法格式如下：

```
'''
注释内容 1
注释内容 2
...
'''
或者
"""
注释内容 1
注释内容 2
...
"""
```

多行注释通常用来为 Python 文件、模块、类或者函数等添加版权、功能说明等信息，例如，下面的代码将使用多行注释为程序添加功能、开发者、版权、开发日期等说明性信息。多行注释也经常用来解释代码中重要的函数参数含义、功能、返回值等信息，以便于后续开发者维护代码。多行注释也可以采用单行注释多行书写的方式实现。例如：

```
"""
程序功能：财税数据提取与分析
开发者：  admin
版权所有：商学院
开发日期：2022 年 3 月
"""
```

3. 中文编码声明注释

在 Python 中编写代码的时候，如果用到指定字符编码类型的中文编码，需要在文件开头加上中文声明注释，以免出现代码错误。因此，中文注释很重要。

Python3.x 提供的中文注释声明语法格式如下：

```
#-*-coding:编码-*-
```

或者为：

```
#coding:编码
```

例如，保存文件编码格式为 UTF-8，在程序的一开始位置，可以使用下面的中文编码声明注释。

```
#-*-coding:utf-8 -*-
```

以上编码声明中，“-*-”没有特殊的作用，只是为了美观才加上的，所以上面的代码也可以使用“coding:utf-8”代替。

在编写程序的过程中，为代码加注释是必须要做的工作。一般情况下，加注释的位置都是重要的事情、关键的步骤或需要特别说明的程序设计思想等，这样，使自己或别人阅读程序更加方便。

思考练习题

一、填空题

1. Python 安装扩展库常用的命令是_____。

2. Python 的标准库里有很多模块，其中数学模块的名称是_____。

3. 导入扩展库的命令名称是_____。

4. Python 对代码的缩进要求严格，同一个级别的代码块的缩进量必须相同，默认为_____个字符。

5. 注释有助于增强程序的可读性，Python 里注释用到的符号是_____。

二、操练练习题

1. 从相关网站下载适合自己电脑系统的Python安装程序或Anaconda安装包并安装。

2. 下载 Pycharm 集成开发环境安装程序并安装。

3. 安装 selenium、requests、jieba 或其他扩展库，并验证是否安装成功。

即测即练

自学自测　扫描此码