第一章　概述与环境准备

本章将介绍 Python 编程语言的概述和基本特性，以及为什么选择 Python 作为量化交易的工具。通过本章的学习，读者将了解 Python 的历史背景、发展现状以及在量化交易领域的应用优势。

1.1　Python 简介

Python 是一种高级、通用、解释型编程语言，由荷兰计算机科学家 Guido van Rossum 于 1989 年底设计开发。Guido van Rossum 是 Python 的创始人，他于 1956 年出生在荷兰。在 20 世纪 80 年代末和 90 年代初，他在荷兰的 Centrum Wiskunde & Informatica（CWI，荷兰国家数学与计算机科学研究中心）工作。Guido 在期间开始着手设计一种新的编程语言，这个语言后来被命名为 Python。

1.1.1　Python 语言的特点

Python 字面上的意思为蟒蛇，但作为编程语言其并非源自爬行动物，而是与英国喜剧团体 Monty Python 有关。Guido van Rossum 是这个喜剧团体的粉丝，他在设计 Python 语言时希望取一个独特而有趣的名字，因此选择了“Python”。Python 的发展历程如下。

1989 年：Guido van Rossum 开始着手设计 Python 语言，最初的目标是创造一种简单易读、代码清晰且功能强大的编程语言。

1991 年：Python 0.9.0 版本诞生，是 Python 的第一个公开发布版本。

1994 年：Python 1.0 发布，正式成为一个完整的编程语言，并引入了函数式编程特性、模块化和异常处理等功能。

2000 年：Python 2.0 发布，引入了列表推导式、垃圾回收机制等新特性。

2008 年：Python 3.0 发布，这是一个重要的版本升级，修复了一些旧版本的不一致性和设计缺陷，并引入了一些新特性。然而，由于向后不兼容，许多项目和库仍然使用 Python 2.x 版本。

2010 年：Python 2.7 发布，它是 Python 2.x 系列的最后一个版本，引入了一些 Python

3.x 版本中的新特性，同时保持了向后兼容性。

2020 年：Python 2.x 系列在 2020 年 1 月 1 日正式终止支持，意味着不再提供安全更新和错误修复，用户被鼓励升级到 Python 3.x 版本。

2021 年：Python 3.9 发布，最新的稳定版本，继续完善语言功能和性能优化。

经过多年的发展，Python 已经成为全球最流行和广泛应用的编程语言之一，其简洁优雅的语法和强大的功能使其在各个领域都有广泛的应用并受到全球范围内的开发者的青睐。以下是 Python 语言的特点和优势。

(1)简洁优雅的语法。Python 采用简洁、易读的语法风格，注重代码的可读性和简洁性。相对于其他编程语言，Python 的代码量通常较少，可以提高开发效率，降低维护成本。

(2)易学易用。Python 的语法简单直观，学习门槛较低，使得新手程序员能够快速上手。它注重自然语言的表达方式，使得开发者能够更容易地理解和书写代码。

(3)开放性和社区支持。Python 是一种开源编程语言，拥有活跃的全球社区。开发者可以免费获取并使用 Python，而且社区提供丰富的文档、教程、第三方库和工具，为 Python 开发者提供了大量的资源和支持。

(4)多平台支持。Python 可以在多种操作系统上运行，包括 Windows、MacOS 和 Linux 等。这使得 Python 成为一种跨平台的编程语言，方便开发者在不同系统上进行开发和部署。

(5)强大的标准库。Python 拥有丰富的标准库，涵盖了各种功能和模块，如文件操作、网络通信、数据库连接、图形界面等。这些标准库的存在为开发者提供了便利，可以直接调用这些模块，而无须自己从头编写代码。

(6)第三方库和框架。Python 生态系统中有大量的第三方库和框架，支持各种应用领域。例如，对于数据科学，Python 提供了 NumPy、Pandas、Matplotlib 等库；对于 Web 开发，有 Django、Flask 等流行的框架。这些库和框架大大提高了 Python 在不同领域的适用性。

(7)面向对象编程(OOP)。Python 支持面向对象编程，使得开发者可以更好地组织和管理代码，提高代码的复用性和可维护性。

(8)强大的社区支持。Python 社区非常庞大活跃，开发者可以在社区中交流学习、寻求帮助、分享经验。这使得 Python 的问题能够迅速得到解决，也推动了 Python 生态系统的不断发展。

(9)适用于多个领域。由于 Python 的灵活性和丰富的库，它在多个领域有着广泛的应用。包括 Web 开发、数据科学、人工智能、机器学习、自然语言处理、自动化、网络编程等等。

(10)互动式开发环境。Python 支持交互式开发环境，如 IPython 和 Jupyter Notebook，可以逐行执行代码并立即查看结果，方便调试和测试。

1.1.2 Python 解释器

Python 是一种高级编程语言，依据 Python 语言编写的代码具有人类友好性，但计算机

无法直接执行，需要有一个工具将人类可读的代码翻译成机器可以理解的机器码，这个工具就是解释器，高级语言通常都有其解释器，Python 语言的解释工具称为 Python 解释器。

Python 解释器是 Python 编程语言的核心组件，它负责将 Python 代码翻译成机器可执行的指令。Python 是一种解释型语言，与编译型语言不同，Python 代码在执行之前并不需要预先编译成二进制文件，而是通过解释器逐行解释并执行代码。其主要功能有以下四点。

(1)读取 Python 源代码。解释器负责读取 Python 源代码(.py 文件)，并按照代码的逻辑结构进行解析。

(2)词法分析。解释器将源代码划分为一系列词素(tokens)，即代码的最小语法单元。

(3)语法分析。解释器根据词法分析的结果，构建代码的语法树，确定代码的逻辑结构。

(4)执行代码。解释器逐行执行代码，将代码转换为机器可执行的字节码，并存为.pyc 文件，便于后续直接执行。

Python 解释器有多种不同的实现和版本，主要的解释器包括以下四种。

(1)CPython 解释器。官方标准解释器，用 C 语言实现，是 Python 语言的参考实现。

(2)Jython 解释器。运行在 Java 虚拟机上的解释器，可以与 Java 代码交互。

(3)IronPython 解释器。运行在.NET 平台上的解释器，可以与 C# 和其他.NET 语言交互。

(4)PyPy 解释器。采用即时(JIT，Just-In-Time)编译技术的解释器，旨在提高 Python 代码的执行性能。

CPython 是最常用的 Python 解释器，也是官方推荐的解释器，其他解释器通常用于特定的应用场景，例如 Jython 用于与 Java 集成，IronPython 用于与.NET 平台交互，PyPy 用于性能优化。

1.2　量化交易简介

量化交易是金融领域中的一项创新实践，通过利用数据和技术来制订高效的交易策略。通过借助编程工具，结合数据分析和模型构建将制订的交易策略进行算法化实现，量化交易者可以在金融市场中获取更多的机会和优势。

1.2.1　量化交易的概念

量化交易是一种基于数据分析、数学模型和统计方法的交易策略，旨在利用计算机算法进行高频、高效的金融交易。与传统的人工交易相比，量化交易更加依赖数据和技术，通过系统性的方法来寻找市场中的价格差异、趋势和模式，从而制订交易决策。量化交易以其高速度、高频率和严格的风险控制而备受关注。

1.2.2 量化交易的基本流程及相关能力要求

量化交易的基本流程可以概括为以下几个步骤：

(1)问题定义与策略制订：首先，量化交易者需要明确交易的目标，选择适合的交易策略。这可能涉及市场套利、趋势跟踪、均值回归等不同类型的策略。

(2)数据收集与处理：获取市场数据是量化交易的基础。交易者需要收集市场价格、成交量、指标数据等，并对数据进行清洗、处理和整合，以便后续分析。

(3)模型建立与测试：在这一阶段，交易者会设计数学模型、统计算法等，用于分析数据和预测市场走势。然后，通过历史数据对模型进行回测，评估其在过去的表现。

(4)优化与调整：根据回测结果，交易者需要对模型进行优化和调整，以提高其性能和适应性。这可能涉及参数调整、风险控制等方面。

(5)实盘交易：一旦模型经过充分测试和优化，交易者将其应用于实际市场，进行实盘交易。在此阶段，自动化执行是常见的做法，以确保交易的高效和准确。

(6)监控与回顾：交易者需要不断监控交易执行情况，同时定期回顾和分析交易绩效。这有助于发现问题、优化策略，并进行必要的调整。

量化交易不仅需要金融知识，还需要具备编程、数学建模、数据分析和风险管理等多方面的能力。交易者需要深入了解金融市场，同时熟练使用编程语言和工具来实现交易策略。此外，统计学、数学优化、机器学习等知识也是量化交易者所需的重要基础。

1.2.3 量化交易与 Python 的关系

Python 是一种广泛使用的编程语言，因其易学易用、丰富的库和工具支持，成了量化交易领域的热门选择。Python 不仅适用于数据分析和处理，还可以用于建立交易策略、执行交易以及绩效评估等各个环节。

Python 在量化交易中的应用，主要包括五个方面。

(1)数据获取与处理：Python 的库(如 Pandas、NumPy)可以轻松处理和分析市场数据，从而支持交易策略的制订。

(2)模型构建与优化：Python 在机器学习领域的强大功能可以用于构建预测模型、优化策略参数等。

(3)交易执行与回测：Python 的执行速度和丰富的库可以支持交易策略的自动化执行，同时进行历史数据回测，评估策略的绩效。

(4)风险管理：Python 可以用于实现风险管理模型，帮助交易者在交易中进行风险控制和资金管理。

(5)可视化与报告：Python 的可视化库(如 Matplotlib、Seaborn)可以创建交易绩效报告、图表，帮助交易者更好地理解和传达策略的效果。

量化交易是金融领域中的一项创新实践，利用数据和技术来制订高效的交易策略。通

过合理的策略制订、数据分析和模型构建，量化交易者可以在金融市场中获取更多的机会和优势。而 Python 作为一种强大的编程语言，其灵活性和丰富的库使得量化交易变得更加容易实现，使交易者能够更好地应对市场的挑战和机遇。

1.3　Python 开发环境的搭建

工欲善其事，必先利其器。在使用 Python 开发程序之前，在计算机上搭建 Python 开发环境是必不可少的环节，其中比较头疼的就是包管理和 Python 不同版本的问题，特别是当你使用 Windows 的时候。本教材编写时 Python 最新稳定版本是 3.11，且支持到 2027 年，如图 1.3.1 所示。

Python version	Maintenance status	First released	End of support	Release schedule
3.12	prerelease	2023-10-02 (planned)	2028-10	PEP 693
3.11	bugfix	2022-10-24	2027-10	PEP 664
3.10	security	2021-10-04	2026-10	PEP 619
3.9	security	2020-10-05	2025-10	PEP 596
3.8	security	2019-10-14	2024-10	PEP 569

图 1.3.1　Python 不同的版本

为了解决这些问题，有不少发行版的 Python，比如 WinPython、Anaconda 等将 python 和许多常用的 package 打包，方便 pythoners 直接使用，此外，还有 virtualenv、pyenv 等工具管理虚拟环境。

本教材选择了 Anaconda，是因为其强大而方便的包管理与环境管理的功能。对初学者来说，是十分友好的。

本节将带你搭建 Python 最新版 3.11 开发环境。

1.3.1　Anaconda 安装与 Jupyter Notebook 的使用

1. 下载 Anaconda

Anaconda 是一个用于科学计算的 Python 发行版，支持 Linux、Mac、Windows 系统，提供了包管理与环境管理的功能，可以很方便地解决多版本 Python 并存、切换以及各种第三方包安装问题。Anaconda 利用工具/命令 conda 来进行 package 和 environment 的管理，并且已经包含了 Python 和相关的配套工具。

如果计算机上已经安装了 Python，安装不会有任何影响。实际上，脚本和程序使用的默认 Python 是 Anaconda 附带的 Python，所以安装完 Anaconda 已经自带安装好了 Python，无须另外安装。

下面介绍如何下载 Anaconda，具体步骤如下（本小节主要以 Windows 系统为例）：

（1）查看计算机操作系统的位数，以决定下载哪个版本。

（2）下载 Anaconda。进入官网（https://www.anaconda.com），单击右上角的“免费下载”按钮“Free Download”，如图 1.3.2 所示。

图 1.3.2 Anaconda 官网界面

(3)单击 Download,根据计算机操作系统选择相应的操作系统(Windows/macOS/Linux),我们选择“Windows”,同时选择 Python 版本,由于 2020 年之后官方不再支持 Python 2,所以建议大家使用 Python 3 及以上版本。如图 1.3.3 所示。

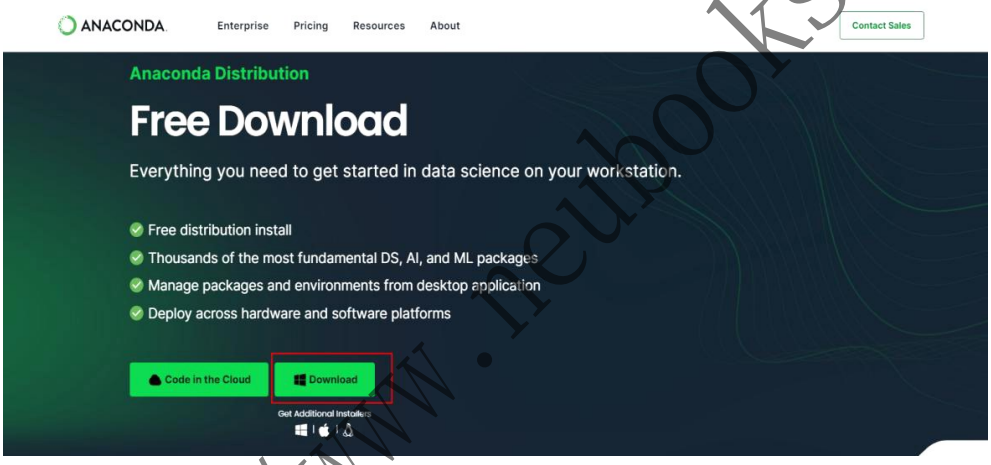

图 1.3.3 Anaconda 官网下载界面

当然,我们还可以将下载的当前网页向下滑动,在最底部的介绍中,也可以单击下载。

注意:选择下载的版本一定要与本机操作系统相同的位数,如图 1.3.4 所示。

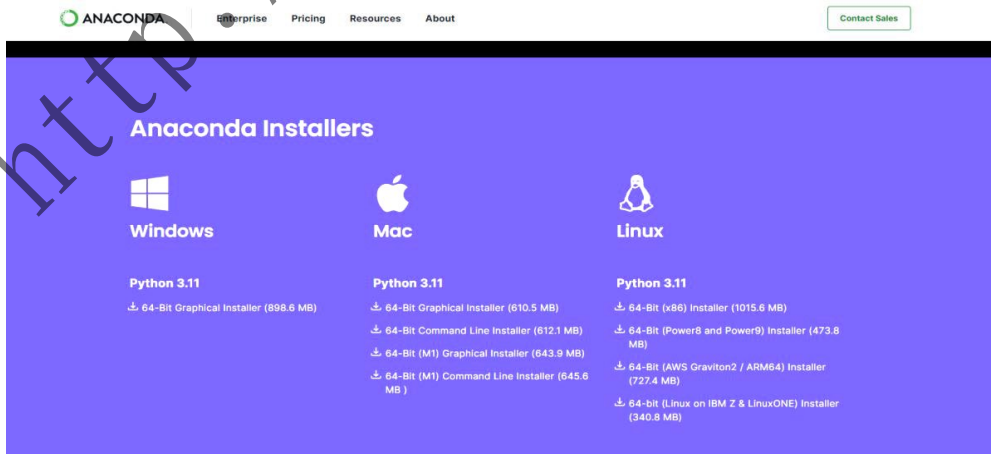

图 1.3.4 Anaconda 官网下载界面(网页最底端)

(4)等待下载。下载后在 C 盘的“下载”目录处可以找到.exe 文件,其文件图标如图 1.3.5 所示。

2. 安装 Anaconda

下载完成后,开始安装 Anaconda,具体步骤如下:

(1)在安装 Anaconda 软件的时候,鼠标右键单击安装软件→选择以管理员的身份运行,如图 1.3.6 所示。

图 1.3.5　Anaconda 下载后的图标

图 1.3.6　Anaconda 以管理员身份运行

(2)单击“下一步”按钮 Next。如图 1.3.7 所示。

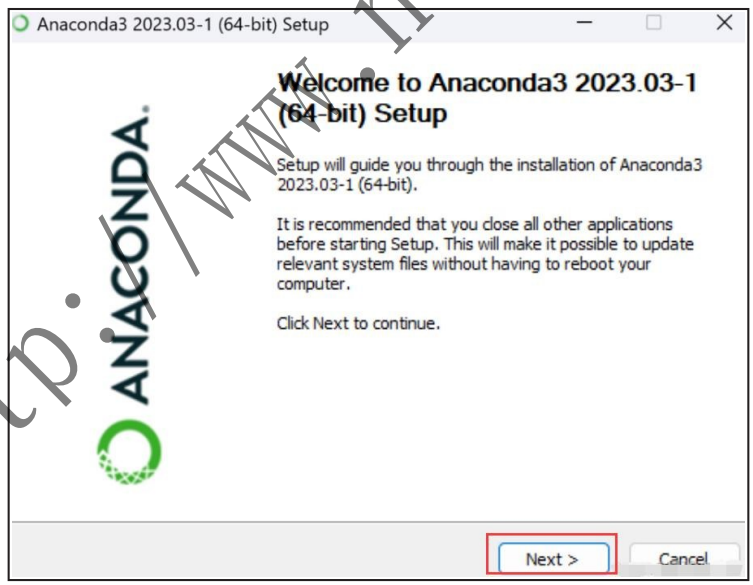

图 1.3.7　单击 Next 按钮

(3)单击 I Agree 按钮接受协议,如图 1.3.8 所示。然后选择安装类型,可选 Just me 或者 All Users,这里选择的是前者,最后单击 Next 按钮,如图 1.3.9 所示。

(4)安装路径选择默认路径即可,当然,也可以单击 Browse 选择安装路径。接着单击“Next”,如图 1.3.10 所示。

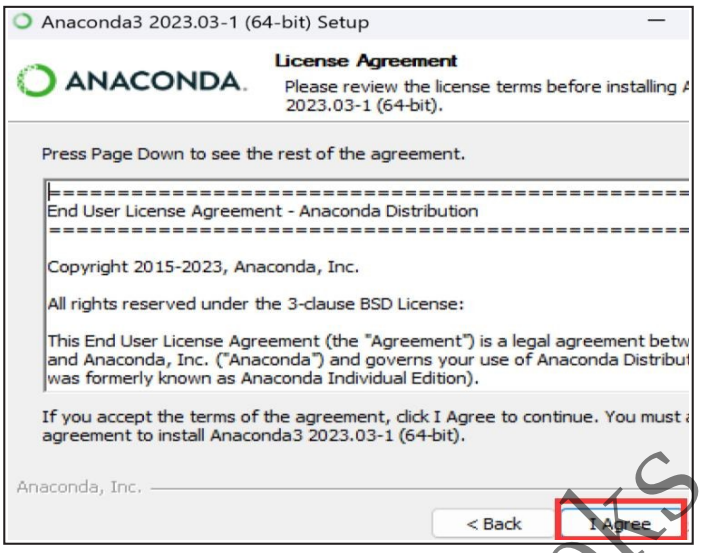

图 1.3.8　单击 I Agree 按钮

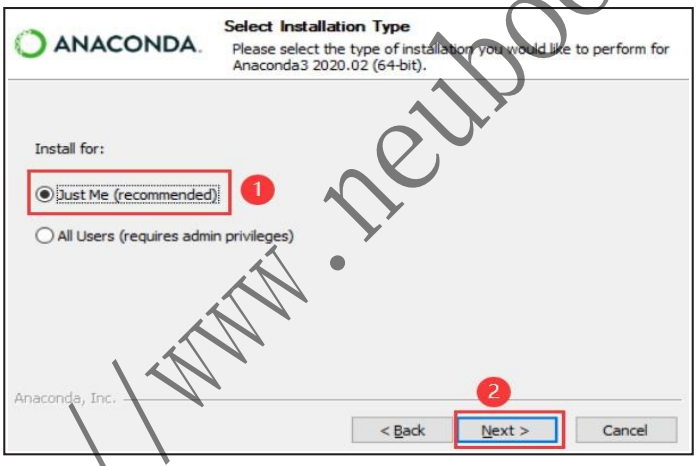

图 1.3.9　选择安装类型

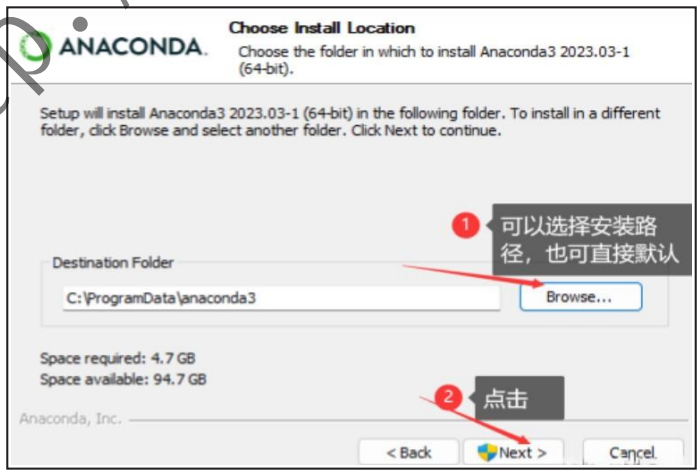

图 1.3.10　选择安装路径

(5)单击“Install”按钮,开始安装 Anaconda。注意勾选相关复选框,如图 1.3.11 所示的选项。

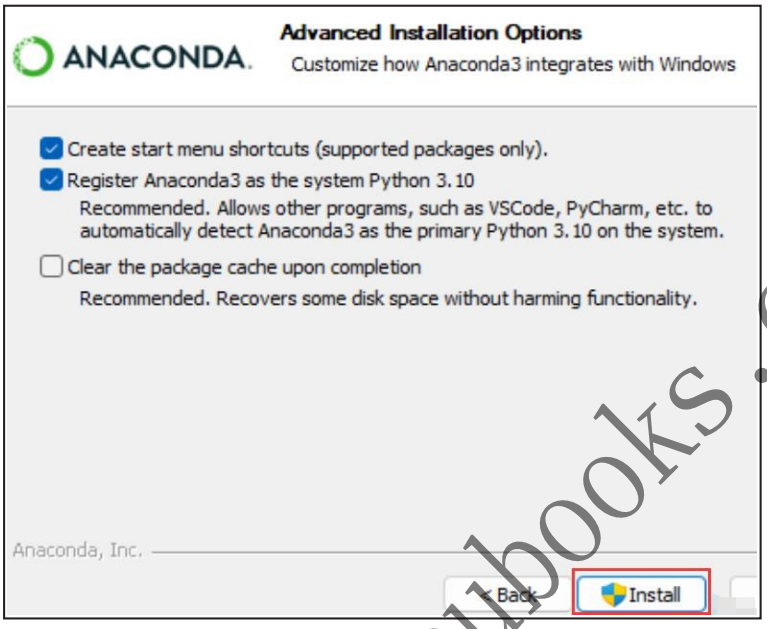

图 1.3.11　单击 Install

(6)继续跟着对话框的提示,单击“Next”按钮(这里不再图片展示),最后单击“Finish”即可。如图 1.3.12 所示。

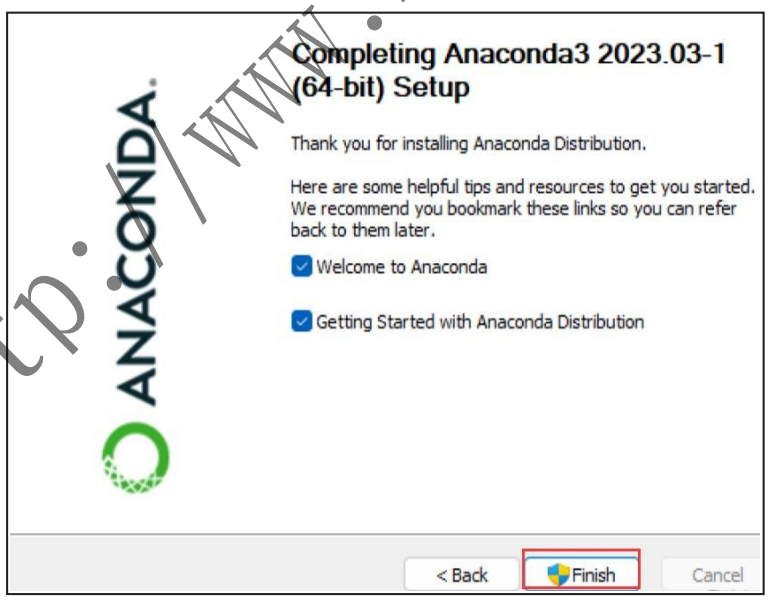

图 1.3.12　单击 Finish

(7)配置环境变量。右键此电脑→属性,单击“高级系统设置”。如图 1.3.13 所示。

图 1.3.13 单击高级系统设置

(8)单击环境变量,如图 1.3.14 所示,然后找到系统变量里的 Path,单击“编辑”,如图 1.3.15 所示。

图 1.3.14 单击环境变量

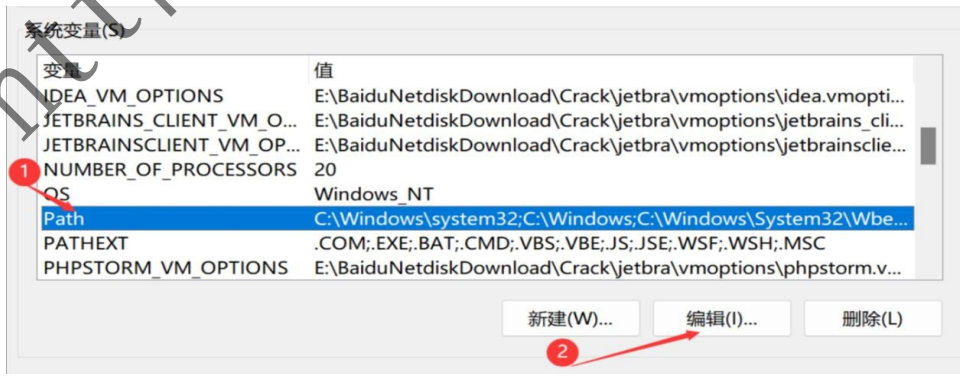

图 1.3.15 找到 Path,单击编辑

(9)单击“新建”,在后面添加 Anaconda 安装路径(注意:之前安装选择的是默认就填默认路径,如果是自己选择的路径,就填选的路径)及 Scripts,本案例的 Anaconda 安装路径为 E:\Anaconda,所以需要添加 E:\Anaconda 和 E:\Anaconda\Scripts,最后单击“确定”即可,如图 1.3.16 所示。

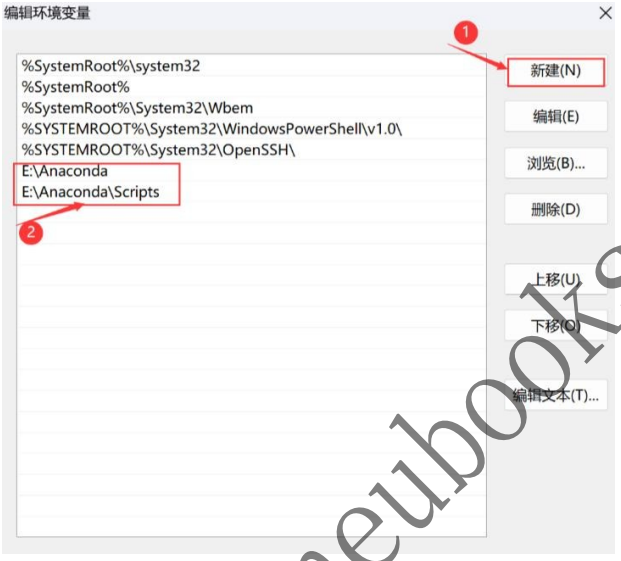

图 1.3.16　添加安装路径及 Scripts

(10)安装完成后,系统开始菜单会显示增加的程序,如图 1.3.17 所示,这就表示 Anaconda 已经安装成功了。

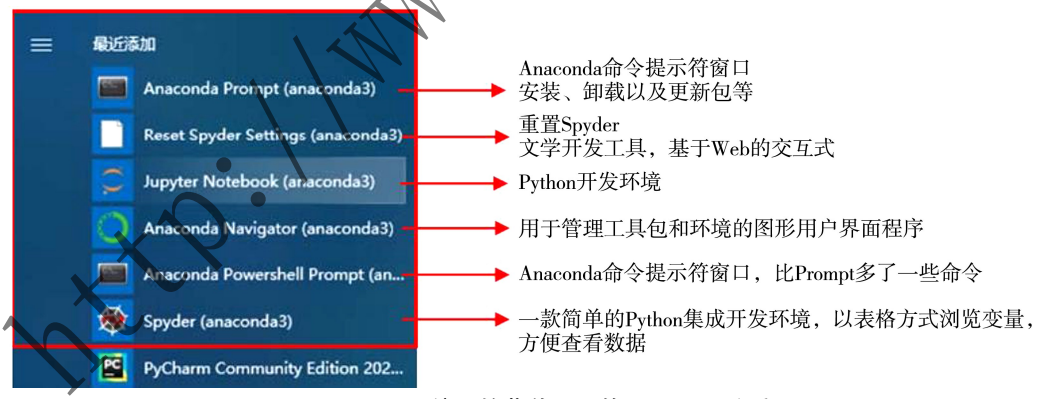

图 1.3.17　系统开始菜单显示的 Anaconda 程序

3. Jupyter Notebook 的使用

当 Anaconda 安装完毕后,我们单击系统开始菜单,找到并单击 Jupyter Notebook 图标。启动后的界面如图 1.3.18 所示。

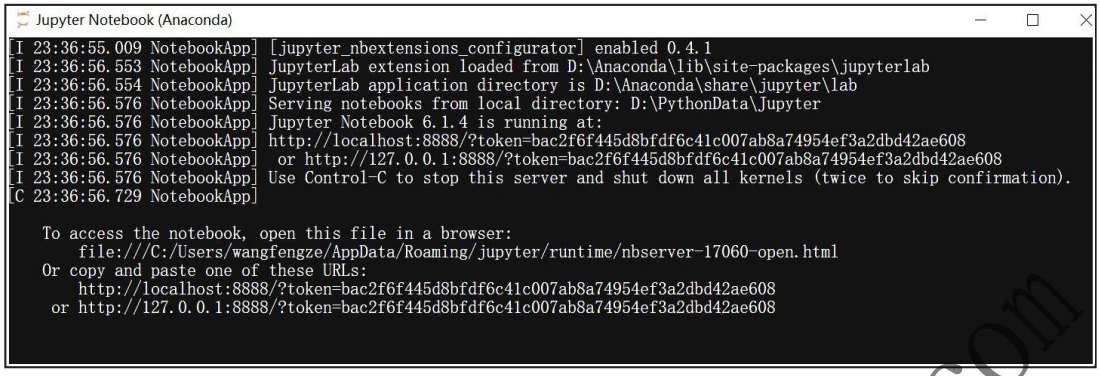

图 1.3.18 启动 Jupyter Notebook

接着，我们会发现，马上就跳转进入网页。这是因为 Jupyter Notebook 是以网页的形式打开，页面中所展示的就是你的电脑 user 用户名路径下面的文件列表(图 1.3.19)，它可以在网页页面中直接编写代码和运行代码，代码的运行结果也会直接在代码块下显示。如在编程过程中需要编写说明文档，可在同一个页面中直接编写，便于作及时的说明和解释。

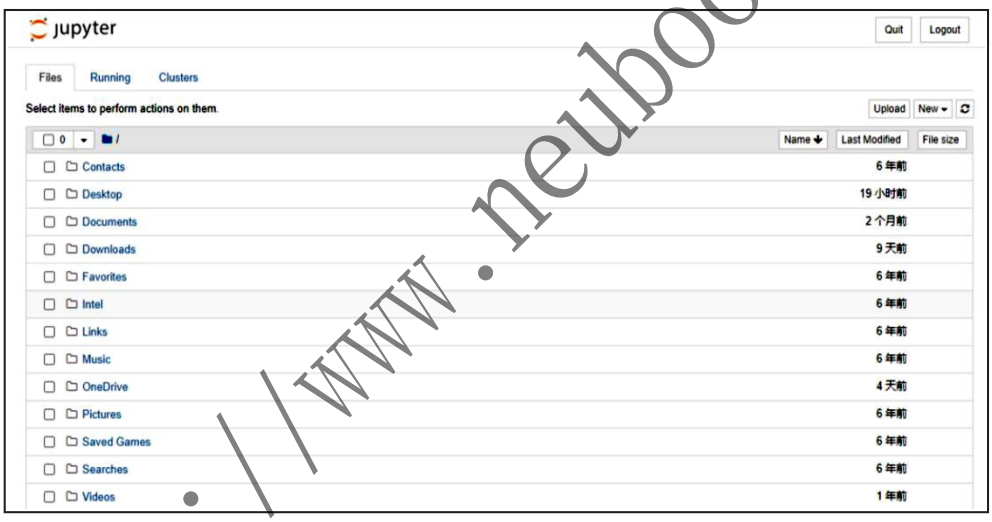

图 1.3.19 Jupyter Notebook 初始界面

(1)新建文件

如图 1.3.20 所示，单击右上角的“New”按钮，选择“Python 3”可新建 Notebook 文件，选择“Folder”可新建文件夹。

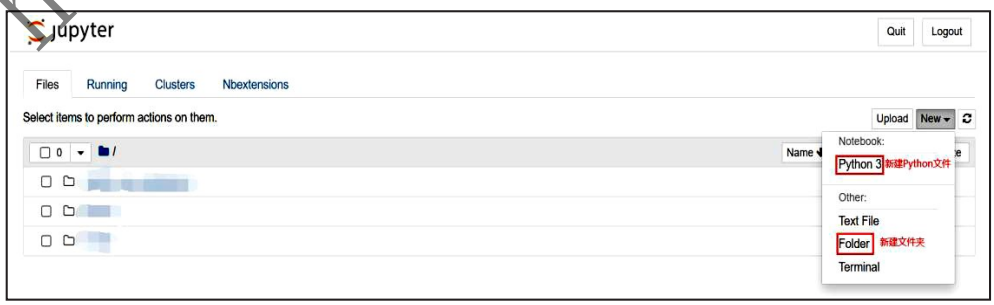

图 1.3.20 Jupyter Notebook 启动界面

选择“Python 3”创建文件后，Jupyter 会在浏览器中打开一个新的页面，可看到如图 1.3.21 所示页面。

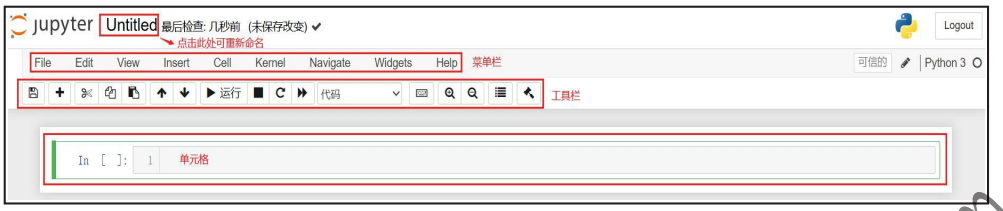

图 1.3.21　Jupyter Notebook 菜单栏和工具栏

- 重命名：单击顶部的【Untitled】可对当前文件重命名。
- 菜单栏：提供保存、打开、新建文件等功能。
- 工具栏：提供运行、剪切、粘贴等与代码操作相关的功能。
- 单元格：在单元格中编写、运行代码。

(2)编写代码

如图 1.3.22 所示，在单元格中即可编写代码，编写完毕后，可单击工具栏的【运行】按钮运行代码，也可使用快捷键(Ctrl+Enter)运行当前单元格，此时，单元格处于“编辑模式”，右上角出现铅笔图标，单元格边框是绿色。

图 1.3.22　Jupyter Notebook 编写代码界面

如图 1.3.23 所示，运行单元格后，铅笔图标消失，单元左侧边框呈蓝色，单元格下方展示代码运行结果，此时，单元格处于“命令模式”。

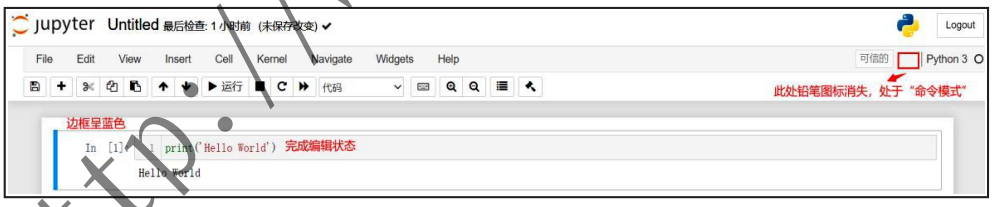

图 1.3.23　Jupyter Notebook 界面命令模式

如图 1.3.24 所示，单元格运行完毕，会有编号(左侧显示 In[1]:)，其代码运行结果将会在文本框下方直接输出。

图 1.3.24　Jupyter Notebook 运行界面

(3)菜单栏介绍

菜单栏如图 1.3.25 所示。

图 1.3.25 Jupyter Notebook 菜单栏

【File】:文件菜单——文件操作。该按钮可以用来打开和存储文件,也可对文件重命名等(图 1.3.26)。

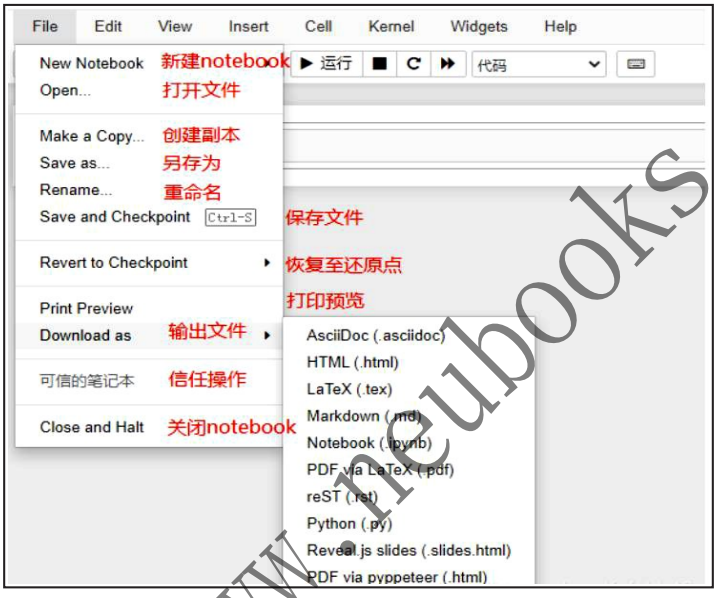

图 1.3.26 Jupyter Notebook 菜单栏 File 界面

具体解释如下:

- New Notebook:新建笔记本(notebook)。二级菜单可以选择新建笔记本的内核(安装语言时会自动安装内核,也可以手动安装内核)。
- Open...:打开笔记本(notebook)。将打开一个新窗口显示仪表盘(Dashboard)以供选择打开的笔记本。
- Make a Copy...:创建当前笔记本的副本。将在当前笔记本所在目录中创建一个名为当前笔记本名称 Copy1 的当前文件的副本,并在新窗口中打开。
- Save as...:另存为。弹出对话框,在其中的输入框中输入当前笔记本的保存路径。路径要求是基于当前笔记本所在目录的相对路径。快捷键为 Ctrl+S。
- Rename...:重命名。弹出对话框,在其中的输入框中输入新标题。
- Save and Checkpoint:保存记事本并作为检查点(还原点)。
- Revert to Checkpoint:恢复笔记本至检查点。弹出一个模态对话框进行确认。注意:恢复操作不可逆。
- Print Preview:打印预览。在新窗口中打开当前页面的打印版,类似网页,不可修改。
- Download as:以其他格式输出记事本。在二级菜单中可以选择输出格式。注意:pdf 等格式需要额外安装插件来提供支持。

- 信任笔记：设置笔记本是否可信。
- Close and Halt：关闭窗口，同时关闭当前记事本的内核进程。注意：如果只是关闭了笔记本的窗口是不会关闭基本的进程的。

【Edit】：编辑菜单——单元格操作。该按钮可以用来编辑单元格，如图 1.3.27 所示。

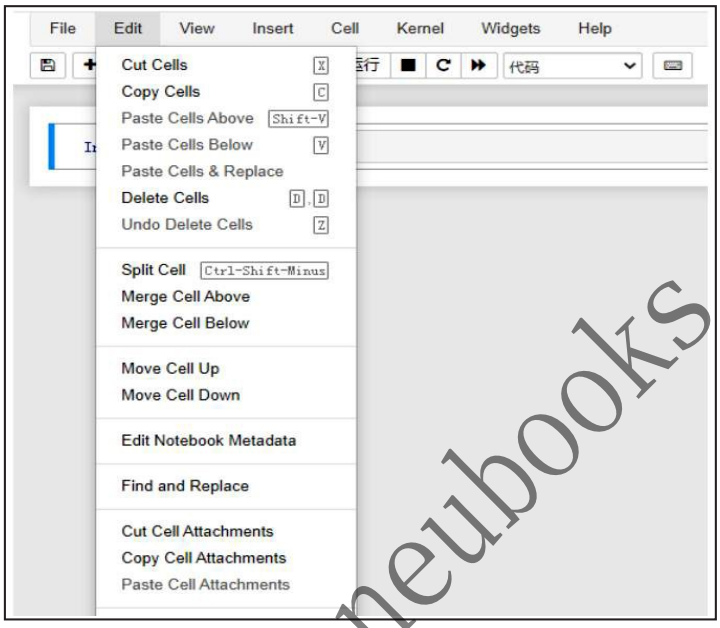

图 1.3.27　Jupyter Notebook 菜单栏 Edit 界面

具体解释如下：

- Cut Cells：剪切单元格。命令模式下，快捷键为 X。
- Copy Cells：复制单元格。命令模式下，快捷键为 C。
- Paste Cells Above：将复制单元格粘贴到当前单元格上方。命令模式下，快捷键为 Shift＋V。
- Paste Cells Below：将复制单元格粘贴到当前单元格下方。命令模式下，快捷键为 V。
- Paste Cells & Replace：粘贴并替换当前单元格。
- Delete Cells：删除当前单元格。命令模式下，快捷键为 D。
- Undo Delete Cells：撤销删除单元格。命令模式下，快捷键为 Z。
- Split Cell：(在光标处)拆分单元格。快捷键为 Ctrl＋Shift＋－。
- Merge Cell Above：合并当前单元格与上方相邻单元格。
- Merge Cell Below：合并当前单元格与下方相邻单元格。
- Move Cell Up：上移当前单元格。
- Move Cell Down：下移当前单元格。
- Edit Notebook Metadata：编辑笔记本元数据。弹出对话框，元数据为 JSON 格式。
- Find and Replace：查找并替换。支持大小写匹配、正则表达式。
- Cut Cell Attachments：剪切单元格附件。
- Copy Cell Attachments：复制单元格附件。

- Paste Cell Attachments：粘贴单元格附件。
- Insert Image：插入图片。弹出对话框可选择插入本地图像文件。仅支持 MarkDown 单元格。

【View】：视图——笔记本外观控制。该按钮可以选择不同方式展示文件，如图 1.3.28 所示。

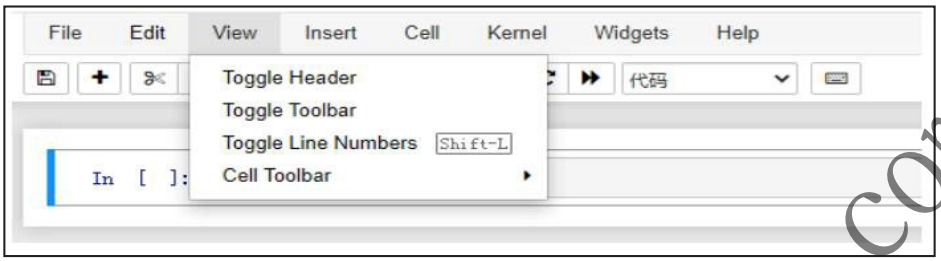

图 1.3.28　Jupyter Notebook 菜单栏 View 界面

具体解释如下：

- Toggle Header：切换显示/隐藏标题栏。
- Toggle Toolbar：切换显示/隐藏工具栏。
- Toggle Line Numbers：切换显示/隐藏单元格行号。
- Cell Toolbar：在单元格中显示单元格工具栏。二级菜单为：

-编辑元数据：在单元格工具栏中显示编辑元数据按钮。单击后可修改单元格的 JSON 元数据。

-原始单元格格式：单元格默认样式。隐藏单元格工具栏。

-幻灯片：在单元格工具栏中显示幻灯片类型下拉框。可设置每个单元格在幻灯片格式下的展示方式。

-附件：在单元格工具栏中显示附件按钮。可编辑当前单元格的附件。

-Tags：在单元格工具栏中显示标签按钮。为单元格提供标签管理。

【Insert】：插入单元格，如图 1.3.29 所示。

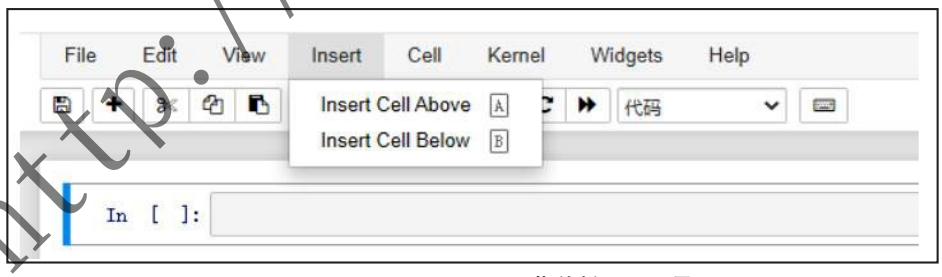

图 1.3.29　Jupyter Notebook 菜单栏 Insert 界面

具体解释如下：

- Insert Cell Above：在当前单元格上方插入空白单元格。命令模式下，快捷键为 A。
- Insert Cell Below：在当前单元格下方插入空白单元格。命令模式下，快捷键为 B。

【Cell】：运行单元格，如图 1.3.30 所示。

具体解释如下：

- Run Cells：运行当前单元格。快捷键为 Ctrl＋Enter。

• Run Cells and Select Below：运行当前单元格，并选中下方相邻单元格。注意，当前单元格为最后一个单元格时会在下方插入空白单元格。快捷键为 Shift＋Enter。

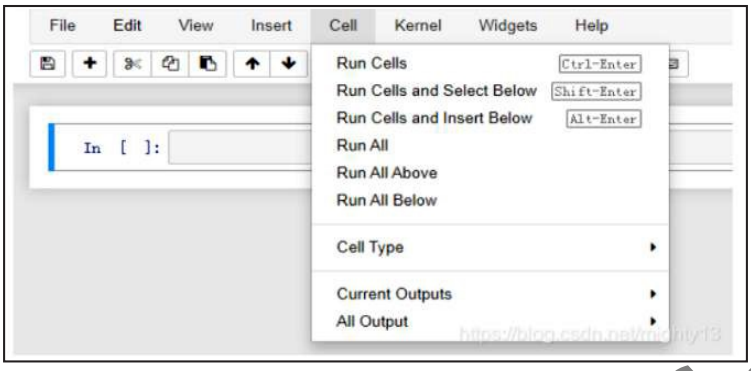

图 1.3.30　Jupyter Notebook 菜单栏 Cell 界面

• Run Cells and Insert Below：运行当前单元格，并在下方插入空白单元格。快捷键为 Alt＋Enter。

• Run All：运行所有单元格。

• Run All Above：运行上方所有单元格（不包括当前单元格）。

• Run All Below：运行当前单元格及下方所有单元格。

• Cell Type：设置单元格类型。二级菜单为：

-Code（代码模式）：命令模式下，快捷键为 Y。

-Markdown（Markdown 模式）：命令模式下，快捷键为 M。

-原生 NBConvert。命令模式下，快捷键为 R。

• Current Outputs：当前单元格输出操作。二级菜单为：

-Toggle：切换显示/隐藏当前单元格输出。命令模式下，快捷键为 O。

-Toggle Scrolling：切换滚动显示当前单元格输出。命令模式下，快捷键为 Shift＋O。

-Clear：清除当前单元格输出。

• All Output：所有输出操作。二级菜单为：

-Toggle：切换显示/隐藏所有单元格输出。

-Toggle Scrolling：切换滚动显示所有单元格输出。

-Clear：清除所有单元格输出。

【Kernel】：内核操作。该按钮可以中断或重启程序，如图 1.3.31 所示。

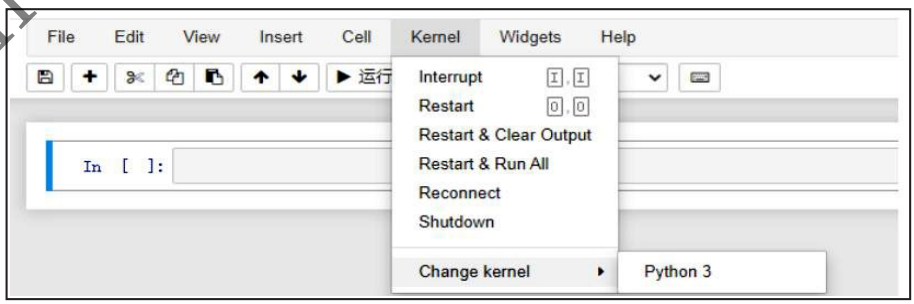

图 1.3.31　Jupyter Notebook 菜单栏 Kernel 界面

具体解释如下：

- Interrupt：中断当前笔记本内核（解释器）与服务器的连接。相当于在控制台执行 Ctrl＋C。命令模式下，快捷键为 I，I。
- Restart：重启当前笔记本内核（解释器）。重启内核将丢失所有变量。命令模式下，快捷键为 O，O。
- Restart & Clear Output：重启当前笔记本内核（解释器）并清除所有单元格输出。
- Restart & Run All：重启当前笔记本内核（解释器）并重新运行所有单元格。
- Reconnect：重新连接服务器。
- Shutdown：关闭内核。关闭连接后需要重启内核进行重连。
- Change kernel：改变笔记本内核（解释器）。二级菜单列出当前已安装的内核。

【Widgets】：部件操作。如图 1.3.32 所示。

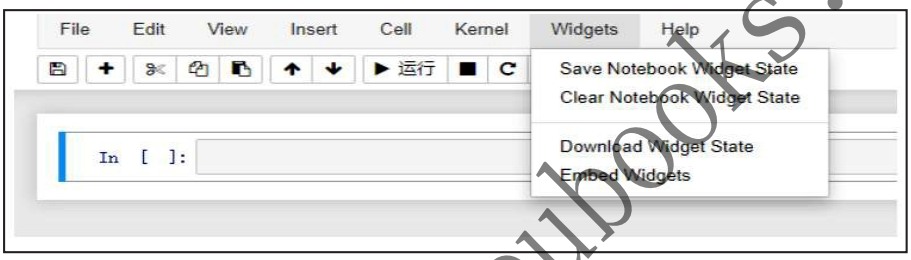

图 1.3.32　Jupyter Notebook 菜单栏 Widgets 界面

具体解释如下：

- Save Notebook Widget State：保存 Notebook 部件状态。
- Clear Notebook Widget State：清除 Notebook 部件状态。
- Download Widget State：下载 Notebook 部件状态。
- Embed Widgets：嵌入 Notebook 部件状态。

【Help】：帮助。该按钮用得最多的功能就属“Keyboard Shortcuts”，用来查看快捷键。如图 1.3.33 所示。

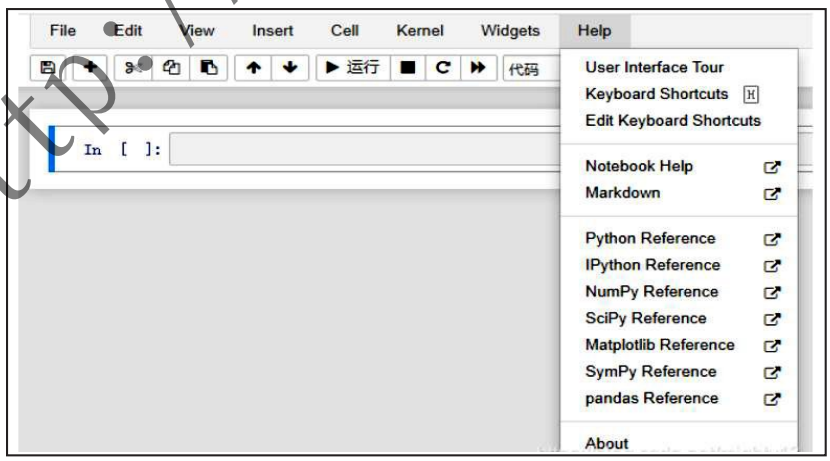

图 1.3.33　Jupyter Notebook 菜单栏 Help 界面

具体解释如下：

- User Interface Tour：操作界面简易说明。
- Keyboard Shortcuts：键盘快捷键。命令模式下，快捷键为 H。
- Edit Keyboard Shortcuts：编辑命令模式快捷键。
- Notebook Help：Notebook 帮助链接。
- Markdown：Github Markdown 帮助链接。
- Python Reference：Python 参考手册链接。
- IPython Reference：IPython 参考手册链接。
- NumPy Reference：NumPy 参考手册链接。
- SciPy Reference：SciPy 参考手册链接。
- Matplotlib Reference：Matplotlib 参考手册链接。
- SymPy Reference：SymPy 参考手册链接。
- pandas Reference：pandas 参考手册链接。
- About：关于。显示 Notebook 的一些版本信息。

(4)工具栏

如图 1.3.34 所示，工具栏功能依次为保存(Notebook 具有自动保存功能，默认 2 分钟后会自动保存)、在下方插入单元格、剪切单元格、复制单元格、粘贴单元格到下方；将选中单元格上移、将选中单元格下移；运行当前单元格、中断系统、重启系统、重启并运行所有代码；单元格功能选择；打开命令配置。

图标	功能
	保存并建立检查点
	在当前单元格下方插入空白代码单元格
	剪切选中的代码块
	复制选中的代码块
	在当前单元格下方插入空白单元格，并将复制的代码块粘贴其中
	上移选中单元格
	下移选中单元格
▶ 运行	中断内核（停止执行代码）
	重启内核
	重启内核并重新运行所有单元格
Markdown	选择单元格类型
	执行notebook命令

图 1.3.34　Jupyter Notebook 工具栏界面

(5)停止运行

当我们在 Jupyter Notebook 中创建了终端或笔记本时,将会弹出新的窗口来运行终端或笔记本。当我们使用完毕想要退出终端或笔记本时,仅仅关闭页面是无法结束程序运行的,因此我们需要通过以下步骤将其完全关闭。

- 方法一,如图 1.3.35 所示。

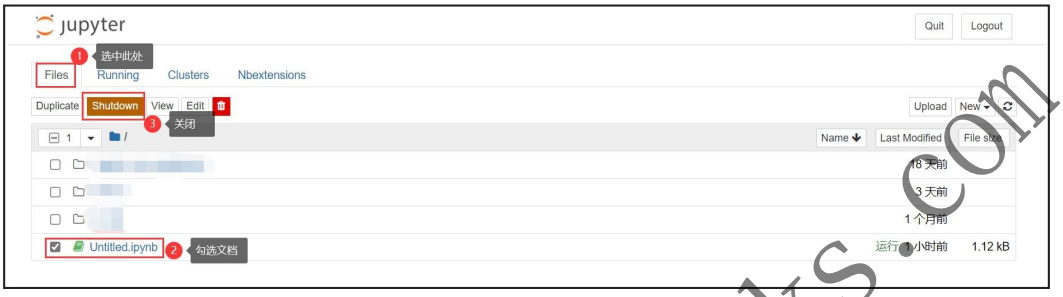

图 1.3.35 Jupyter Notebook 关闭笔记本 1

①进入“Files”页面。

②勾选想要关闭的“ipynb”笔记本。正在运行的笔记本其图标为绿色,且后边标有“Running”的字样;已经关闭的笔记本其图标为灰色。

③单击上方的黄色的“Shutdown”按钮。

④成功关闭笔记本。

注意:此方法只能关闭笔记本,无法关闭终端。

- 方法二,如图 1.3.36 所示。

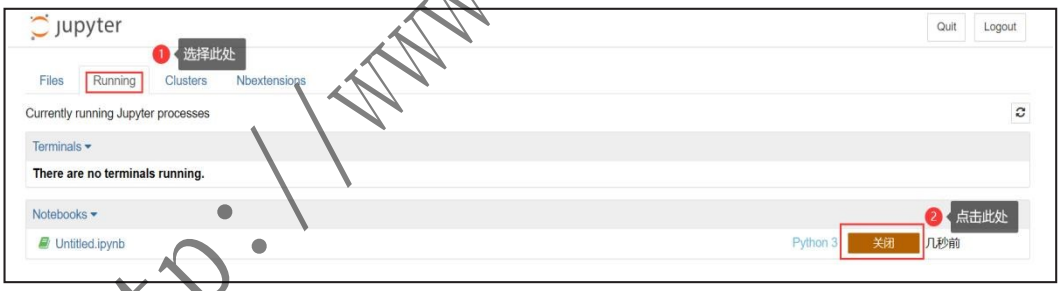

图 1.3.36 Jupyter Notebook 关闭笔记本 2

①进入“Running”页面。

②第一栏是“Terminals”,即所有正在运行的终端均会在此显示;第二栏是“Notebooks”,即所有正在运行的“ipynb”笔记本均会在此显示。

③单击想要关闭的终端或笔记本后黄色“Shutdown”按钮。

④成功关闭终端或笔记本。

注意:此方法可以关闭任何正在运行的终端和笔记本。

(6)快捷键

命令模式:通过键盘命令响应相应操作,通过一个灰色的单元格边界显示,边框为蓝色左边框,其快捷键如表 1.3.1 所示。

表 1.3.1　　命令模式的快捷键

按键	功能	按键	功能
Enter	进入当前单元的编辑模式	X	剪切选中的代码块
Shift＋Enter	运行当前单元并选中下一单元	C	复制选中的代码块
Ctrl＋Enter	运行当前单元	Shift＋V	在当前单元格上方粘贴
Alt＋Enter	运行当前单元并在下方插入新单元	V	在当前单元格下方粘贴
Y	切换到代码状态	Z	撤销删除操作
M	切换到 MarkDown 状态	D,D	删除选中的代码块
R	切换到 Raw NBConvert	Shift＋M	将当前单元格与下一单元格合并
数字键 1 到 6	将当前单元第一行变为 Markdown 的 n 级标题	S/Ctrl＋S	保存并设置检查点
↑/K	选择上一个代码块	L	显示/隐藏当前单元格的代码行号
↓/J	选择下一个代码块	O	显示/隐藏当前单元格的输出内容
A	在当前单元格上方插入新代码块	Shift＋O	显示/隐藏当前单元格的输出内容的滚动条
B	在当前单元格下方插入新代码块	Esc/Q	关闭弹窗
H	展示快捷键帮助	I,I	打断 Kernal 运行
Space	滚动向下	O,O	重启 Kernal
Shift＋Space	滚动向上	Shift＋(↑/↓)	选中多个代码块

编辑模式：允许用户将代码或文本输入到一个单元格中，并通过一个绿色的单元格边框表示，其快捷键如表 1.3.2 所示。

表 1.3.2　　编辑模式的快捷键

按键	功能	按键	功能
Tab	代码补全/缩进	Ctrl＋→	光标右移一个词
Shift＋Tab	工具提示/反缩进	Ctrl＋Backspace	删除前一个词
Ctrl＋[	缩进	Ctrl＋Delete	删除后一个词
Ctrl＋]	反缩进	Ctrl＋M/Esc	进入命令模式
Ctrl＋A	全选	Ctrl＋Shift＋P	打开命令选择板
Ctrl＋Z	撤销	Shift＋Enter	运行当前单元并选中下一单元
Ctrl＋Y/ Ctrl＋Shift＋Z	重复	Ctrl＋Enter	运行当前单元

（续表）

按键	功能	按键	功能
Ctrl＋Home	移动光标到单元首	Alt＋Enter	运行当前单元并在下方插入新单元
Ctrl＋End	移动光标到单元尾	Ctrl＋Shift＋-	按光标位置分割当前单元
Ctrl＋←	光标左移一个词	Ctrl＋S	保存并设置检查点

1.3.2 标准库、扩展库的导入与使用

Python 中的对象大概可以分为三类：内置对象、标准库对象和扩展库对象。其中内置对象是直接编译进解释器的可以直接使用，没有对应的 Python 源代码；标准库对象是随 Python 安装的，但是需要导入才能使用，相应的 Python 源代码在安装目录中的 Lib 目录中；扩展库需要单独安装之后再导入才能使用，其 Python 源代码在安装目录的 Lib\site-packages 目录中，也有一些扩展库的核心代码编译成为 dll 或 pyd 的动态链接库。

为了加深理解，我们举个例子。

【例 1-3-1】请在单元格中输入 math.sin(0.5)。示例代码如下：

```
# 求 0.5(单位是弧度)的正弦
math.sin(0.5)
```

其运行结果如图 1.3.37 所示。

```
NameError                                 Traceback (most recent call last)
<ipython-input-3-548256046f3c> in <module>
----> 1 math.sin(0.5)

NameError: name 'math' is not defined
```

图 1.3.37 math.sin(0.5)运行结果图

我们发现上述案例，运行结果报错了，究其原因是我们没有导入标准库＜math＞导致的。下面我们导入 math 标准库，再看看运行结果是怎样的。

【例 1-3-2】请导入标准库＜math＞，并输入 math.sin(0.5)。示例代码如下所示：

```
# 导入 math 标准库
import math
math.sin(0.5)   # 输出结果为 0.479425538604203
```

由此可以看出，当我们在编写代码时，一般先导入标准库对象，再导入扩展库对象。建议在程序中只导入确实需要使用的标准库和扩展库对象，确定用不到的没有必要导入，这样可以适当提高代码加载和运行速度，并能减小打包后的可执行文件的体积。本小节将介绍和演示导入对象的三种方式，以及不同方式导入时对象使用形式的不同。

1. import 模块名/包名[as 别名]

使用这种方式导入以后，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”的形式进行访问。如果模块名字很长的话，可以为导入的模块设置一个别名，然后使用“别名.对象名”的方式来使用其中的对象。

【例 1-3-3】请在 Jupyter Notebook 中输入一下代码并运行。

```
import math as mt    # math 库是 Python 提供的内置数学类函数库，可起别名
                       为 mt，后面可以通过 mt 使用 math
print(mt.sin(0.5)) # 输出结果为 0.479425538604203
print(mt.sqrt(16)) # 输出结果为 4.0
```

2. from 模块名/包名 import 对象名/模块名[as 别名]

使用“from 模块名/包名 import 对象名/模块名[as 别名]”的方式仅导入明确指定的对象，使用对象时不需要使用模块名作为前缀，可以减少程序员需要输入的代码量。这种方式可以适当提高代码运行速度，打包时可以减小文件的体积，示例如下。

```
from math import pi as PI     # 从 math 模块中导入圆周率的值
from os.path import getsize # 从指定文件的路径模块中导入 getsize 函数获取文件大小
from random import choice    # 从'随机'模块中导入 choice 函数
r=3
print(round(PI*r*r,2))   # 输出结果为 28.27
print(getsize(r'C:\windows\notepad.exe')) # 输出结果为 201216
print(choice('Python')) # 输出结果为 0
```

3. from 模块名 import *

使用“from 模块名 import *”的方式可以一次导入模块中的所有对象，可以直接使用模块中的所有对象而不需要使用模块名作为前缀，但一般不推荐这样使用，示例代码如下。

```
from itertools import *  # 从迭代器模块中导入其所有模块(对象)


characters = '12345'
for item in combinations(characters,3): # 从 5 个字符串中任选 3 个的组合
    print(item,end= '')   # "end=''"表示输出后不换行
print('\n'+'='*20)         # 换行后输出 20 个等于号
for item in permutations(characters,3): # 从 5 个字符串中任选 3 个的排列
    print(item,end='')
```

其运行结果如图 1.3.38 所示。

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

图 1.3.38　运行结果图

4. 安装扩展库

在实际开发过程中，标准的 Python 安装只包含了内置模块和标准库，我们除了会使用标准库以外，通常还需要使用一些扩展库来帮助我们完成特定的任务，开发人员可以根据实际需要再安装和使用合适的扩展库。

表 1.3.3 展示了常见的 Python 扩展库。

表 1.3.3　　常用的 Python 扩展库

扩展库	简介
NumPy	提供数组支持，以及相应的高效的处理函数
SciPy	提供矩阵支持，以及矩阵相关的数值计算模块
Matplotlib	强大的数据可视化工具、做图库
pandas	强大、灵活的数据分析和探索工具
StatsModels	统计建模和计量经济学，包括描述统计、统计模型估计和推断
Scikit-Learn	支持回归、分类、聚类等的强大的机器学习库
Keras	深度学习库，用于建立神经网络以及深度学习模型
Gensim	用来做文本主题模型的库，文本挖掘可能用到

Python 自带的 pip 工具是管理扩展库的主要方式，支持 Python 扩展库的安装、升级和卸载等操作。常用的 pip 命令的使用方法如表 1.3.4 所示。

表 1.3.4　　常用 pip 命令的使用方法

pip 命令示例	说明
pip freeze	列出已安装模块及其版本号
pip install SomePackage[==version]	在线安装卸载 SomePackage 模块，可以使用方括号内的形式指定扩展库版本
pip install SomePackage. whl	通过 whl 文件离线安装扩展库
pip install --upgrade SomePackage	升级 SomePackage 模块
pip uninstall SomePackage	卸载 SomePackage 模块

下面我们以查看列出已安装模块及其版本号为例，进行演示。

(1)打开 Jupyter 的终端，输入 pip 命令即可，如图 1.3.39 所示。

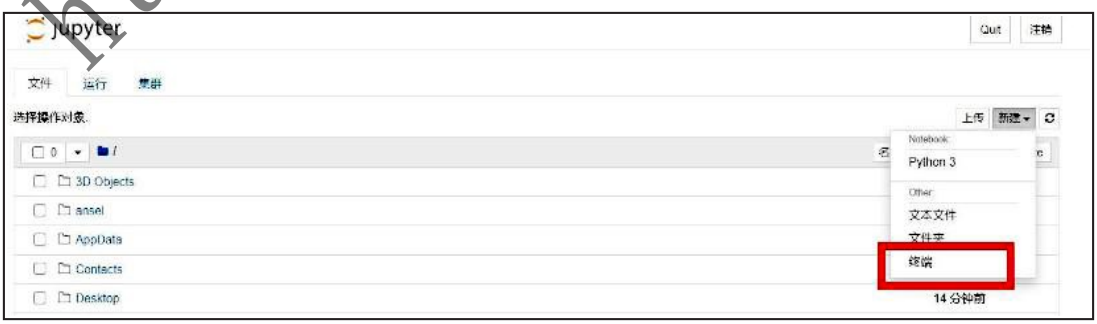

图 1.3.39　打开 Jupyter 的终端

(2)输入 pip freeze 代码,如图 1.3.40 所示。

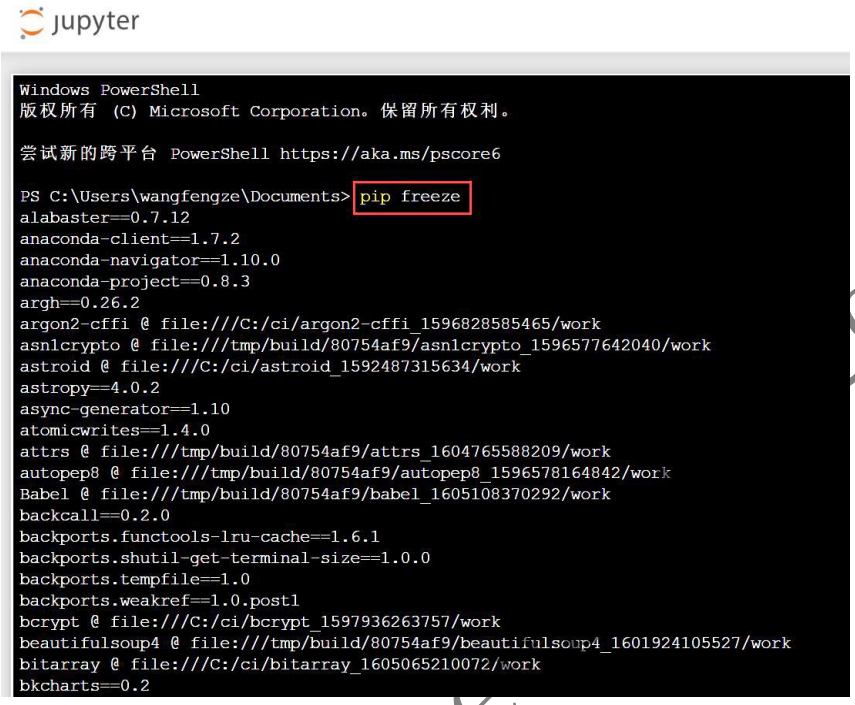

图 1.3.40　查看列出已安装模块及其版本号

在 Windows 平台上,如果在线安装扩展库失败,可以下载扩展库编译好的".whl"文件(一定要选择正确版本且不要修改下载的文件名),然后在命令提示符环境中,使用 pip 命令进行离线安装。例如:

```
pip install pandas-0.24.0-cp37m-win_amd64.whl
```

下面推荐几个 whl 源地址:

- pypi 网站:https://pypi.python.org/pypi/
- whl 集合网:https://www.lfd.uci.edu/~gohlke/pythonlibs/
- 国内镜像网站:http://pypi.douban.com/simple --trusted-host pypi.douban.com

在 Anaconda 3 开发环境中,除了 pip 之外,也可以使用 conda 命令安装 Python 扩展库,用法与 pip 类似。不过,并不是每个扩展库都有相应的 conda 版本,如果遇到 conda 无法安装的扩展库,进入 Anaconda 3 安装目录的 scripts 文件夹中,使用 pip 安装之后,同样可以在 Anaconda 3 的 Jupyter Notebook 环境中使用。

第二章　Python 基础知识

Python 语言因其简洁、易读等特性以及强大的第三方库支持，成为量化交易中的首选语言。在本章中，我们将探讨 Python 语言的基础知识，包括 Python 语言的基础语法、数据类型、流程控制语句、函数以及文件操作、正则表达式等方面的知识，为后续的量化交易程序设计打下坚实的基础。

2.1　Python 基础语法

本节将介绍 Python 的语法规则、变量和常量的定义和使用，以及如何进行输入输出操作。

2.1.1　Python 的语法规则

本节主要介绍 Python 的语法规则，包括标识符、关键字、注释和代码缩进等重要概念。

1. 标识符

标识符(identifier)是变量、函数、类、模块以及其他对象的名称。

(1)标识符的命名规则

①标识符由大小写字母、数字、下划线“_”组成。

②标识符不能以数字开头、不能包含除下划线以外的其他任何特殊字符，如@、%、& 等。

③以双下划线开头的标识符通常具有特殊意义。

④标识符严格区分大小写。如 name 与 Name 是两个不同的标识符。

⑤关键字不能作为标识符。如 if、while 等。

⑥应避免使用内置函数名作为自定义标识符名。如 print、int 等。

例如，int_1、name2、str1、_name 均是合法的标识符；而 2_student(数字开头)、@room(含特殊字符)、True(关键字)均为不合法的标识符。

(2)标识符的命名规范

①标识符最好具有见名知意的效果。如表示学生姓名时，student_name 比 s_n 更好。

②标识符命名方式。

大驼峰命名法(upper camel case)，也称 PascalCase：多个单词组成且各单词首字母大

写。例如：FirstName、LastName。

小驼峰命名法(lower camel case)，也称 camelCase：多个单词组成且除第一个单词外首字母大写。例如：firstName、lastName。

下划线命名法(under score case)：用下划线“_”连接所有单词。例如：user_name。

2. 关键字

关键字(keyword)，又称为保留字，是 Python 语言中内部使用的、具有特定功能的特殊标识符，它们承载着特定的语义。由于关键字在 Python 中具有特殊的地位和作用，因此它们不能被用作用户自定义的标识符。如果尝试将关键字作为自定义标识符使用，将会在编译时引发错误。这是为了避免与 Python 语言的内部机制和语法规则发生冲突。

在 Jupyter Notebook 中创建代码单元格，在单元格中输入 help("keywords")并单击运行，可查看 Python 关键字，如图 2.1.1 所示。

```
In [1]: help("keywords")

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               break               for                 not
None                class               from                or
True                continue            global              pass
__peg_parser__      def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
```

图 2.1.1　Python 关键字

3. 注释

为了提高代码的可读性，通常对某行或某段代码进行解释和说明，适当地注释可以帮助其他人理解你的代码，同时起到备注的作用，作为自己日后回顾代码时的提示。Python 解释器将忽略所有注释语句，注释语句不会影响程序的运行结果。

Python 的注释有单行注释和多行注释。

(1)单行注释

在 Python 中，单行注释以“#”符号开始，该符号后的内容将被视为注释，直到行末。这些注释内容不会被程序执行。单行注释“#”可以位于任何代码的行末，也可以单独占一行。语法格式为：

```
#注释内容
```

说明：PEP 8 风格指南建议在注释和代码之间至少应有两个空格的间隔，以确保注释与代码的清晰分离。这样可以提高代码的可读性和可维护性。因此，遵循这一规范可以使你的 Python 代码更加整洁和一致。

注意：“#”号只能注释单行，若是多行注释需要在每行开始都加上“#”，或者使用后文介绍的三对单引号或三对双引号。

【例 2-1-1】单行注释示例

```
#用户输入一个三位自然数,输出各位上的数字
x = int(input('请输入一个三位数:'))
a = x // 100  #百位上的数字
b = x // 10 % 10  #十位上的数字
c = x % 10  #个位上的数字
print("百位上的数字是:", a, "十位上的数字是:", b, "个位上的数字是:", c)
```

(2)多行注释

在 Python 中,以三对单引号('''注释内容''')或者三对双引号("""注释内容""")作为多行注释的符号,即位于三对引号之间的任何语句都会被解释器忽略。

多行注释通常放在文件、模块、类或函数的前面,一般用于解释版权、功能等信息。

注释不仅可以提高代码的可读性,还是调试程序的工具。通过将部分代码注释起来,运行注释以后的代码,查看程序是否正常执行,从而缩小排错范围。

4. 代码缩进

缩进(indentation)是指代码块中每行语句前面的空白区域,用于表示程序中的逻辑层次和结构。在 Python 中,代码的缩进是非常重要的,因为它决定了代码块的开始和结束。Python 通过采用代码缩进来区分不同的代码块和逻辑层次,并使用英文冒号“:”来表示条件语句、循环语句、函数定义、类定义等的开始。同一级别的代码块必须保持一致的缩进量。通常情况下,Python 编程规范建议使用四个空格作为缩进级别。正确的缩进可以使代码更加清晰易读,同时也可以避免一些常见的逻辑错误和语法错误。因此,在 Python 编程中,遵循正确的缩进规范是非常重要的。

下面是一个使用缩进的示例代码:

```
x = 1
y = 2
if x > y:
    a = x
else:
    a = y
print(x, "和", y, "中", a, "较大")
```

注意:Python 中如果同一级别的代码块的缩进不一致,将导致编译错误。“IndentationError”即为因缩进产生的错误。PEP 8 建议使用四个空格作为缩进级别,不使用制表符(Tab 键)缩进。

2.1.2 变量和常量

学习变量和常量的概念和使用方法,能够更好地管理和操作数据。本节将学习如何声明和使用变量,以及如何定义和使用常量。

1. 变量

变量(variable)是程序执行过程中用于存储数据的一种方式,顾名思义变量的值是可以改变的。我们可以通过变量名获取和操作变量的值。

(1)变量的声明与赋值

在 Python 中,变量不需要提前声明,变量的赋值操作就是对变量的声明和定义的过程。每个变量在使用前都必须赋值,变量赋值以后该变量才会被创建。为变量赋值的语法格式如下:

```
变量名 = 值
```

说明:

①变量名即为变量的标识符,必须满足标识符的命名规则,同时要做到“见名知意”。

②使用“=”为变量赋值,“=”是赋值运算符,要与比较运算符“==”区分开。

③在 Python 语言中,指定变量名的同时必须强制赋初值,否则编译器会报错。

例如:

```
n  # n 变量未赋初始值,编译器报错:NameError: name 'n' is not defined
```

④Python 在变量声明时不需要指定数据类型,而是直接对变量进行赋值。Python 解释器会根据赋值或运算对象的类型自动推断变量类型。Python 是一种动态类型的语言,即变量的类型随变量值的变化而变化。例如:

```
n = 18
print(type(n))  # 运行结果为<class 'int'>
n = '张三'
print(type(n))  #运行结果为<class 'str'>
```

⑤变量赋值的其他用法

Python 语言允许同时给多个变量赋值。例如:

```
a,b,c = 1,2,3  #执行语句后,变量 a 的值为 1,变量 b 的值为 2,变量 c 的值 为 3
```

也可以同时为多个变量赋相同的值。例如:

```
a = b = c = 1  #变量 a、b、c 的值都为 1
```

(2)变量在内存中的存储

在 Python 语言中,有一个核心理念:一切皆是对象。当 Python 程序执行时,它会在内存中创建并管理各种对象。通过赋值语句,我们可以将对象与变量相关联,这样变量就成了引用该对象的一种方式。实际上,变量更像是一个标签或引用,它并不直接存储对象的值,而是存储了对象在内存中的地址或引用。通过这种方式,我们可以方便地通过变量来访问和操作对象。

如果多个不同的变量被赋值为内存中的同一个数据对象,那么这些变量实际上会指向同一个内存地址。例如,首先创建一个对象 3,在内存中创建一个名为 a 的变量,并将它指向整数 3,然后创建了一个指向同一个对象的变量 b。

```
a = 3
b = a
id(a)   # id() 函数返回对象的内存地址   返回值 2056022485360
id(b)   #返回值 2056022485360
```

当把变量 a 赋值给另一个变量 b,实际上是把变量 b 指向变量 a 所指向的数据。变量 a 和变量 b 的内存地址一致。

```
b = 5
id(a)   #返回值   2056022485360
id(b)   #返回值   2056022485424
```

给 b 赋值 5 后,b 的内存地址改变,而 a 的内存地址仍为原来的地址。

2. 常量

在 Python 中,没有专门定义常量的语法和关键字。也就是 Python 中没有常量,只能将变量当成常量使用。Python 语言通过约定,将变量名全部大写(可以使用下划线增加可读性)的形式表示常量名。例如:

```
PI = 3.14    #浮点类型常量 PI
MAX_LENGTH = 190    #整数类型常量 MAX_LENGTH
CLASS = '一班'   # 字符型常量
```

2.1.3 输入与输出

使用 Python 内置的输入函数 input()和输出函数 print()可以实现程序与用户的交互,并将结果以可读的方式展示出来。这对于编写交互式的量化交易程序和输出结果报告非常重要。

1. 输入函数 input()

input()函数用于向用户显示提示信息,然后获取用户输入的内容,无论用户输入什么内容,程序都将其转换为字符串,因此 input()函数总是返回一个字符串的变量类型,如果想要得到其他类型的数据,需要进行类型转换。

input()函数的语法格式如下:

```
input([prompt])
```

其中,prompt 是提示字符串,可以省略。但通常需要给用户提示信息,提醒用户输入怎样的数据,因此,通常不省略。input()函数的使用方法如下:

```
user_input = input("请输入股票代码 ")
```