

★ 服务热线: 400-615-1233
★ 配套精品教学资料包
★ www.huatengedu.com.cn

Python 在金融中的应用

Python ZAI JINRONGZHONG DE YINGYONG

高等职业教育财经商贸系列创新教材

Python 在金融中的应用

主编 赵阳 邱小津

北京邮电大学出版社



ISBN 978-7-5635-7803-0



9 787563 578030 >

定价: 49.00元

策划编辑: 袁相芬
责任编辑: 张海红
封面设计: 刘文东

高等职业教育财经商贸系列创新教材

Python 在金融中的应用

主编 赵阳 邱小津



北京邮电大学出版社
www.buptpress.com

高等职业教育财经商贸系列创新教材

Python 在金融中的应用

主 编 赵 阳 邱小津
副主编 李庚倩 孙 悦 于 翠



北京邮电大学出版社
www.buptpress.com

内 容 简 介

本书以金融领域的实际应用场景为核心,将 Python 编程技术与金融业务需求紧密结合。每个项目均围绕金融行业的典型任务展开,通过真实案例引导学生从理论到实践,逐步掌握 Python 在金融中的具体应用。全书共有五个项目,分别是 Python 基础、爬虫基础+金融商业案例实战、数据分析基础+商业案例实战、爬虫进阶+金融商业案例实战、金融数据分析进阶+商业案例实战。学生不仅能学习 Python 的基础语法,还能通过实际数据计算投资组合的最优配置,真正实现“学以致用”。

本书融入前沿技术,紧跟行业发展趋势,同时注重职业素养与价值观培养。本书既可作为各院校金融服务与管理专业的教材,也适合计算机专业的学生了解金融业务。

图书在版编目(CIP)数据

策划编辑:袁相芬 责任编辑:张海红 封面设计:刘文东

出版发行:北京邮电大学出版社

社 址:北京市海淀区西土城路 10 号

邮政编码:100876

发 行 部:电话:010-62282185 传真:010-62283578

E-mail: publish@bupt.edu.cn

经 销:各地新华书店

印 刷:三河市骏杰印刷有限公司

开 本:787 mm×1 092 mm 1/16

印 张:14.5 插页 1

字 数:300 千字

版 次:2025 年 月第 1 版

印 次:2025 年 月第 1 次印刷

ISBN 978-7---

定 价:49.00 元

• 如有印装质量问题,请与北京邮电大学出版社发行部联系 •

服务电话:400-615-1233

前言

在数字经济蓬勃发展的今天，金融行业正经历着前所未有的数字化转型。Python 作为当今最受欢迎的编程语言之一，凭借其简洁的语法、丰富的库生态系统和强大的数据处理能力，成为金融科技领域不可或缺的工具。从基础的数据分析到复杂的量化交易策略，从风险建模到智能投顾系统，Python 正在重塑金融业务的每一个环节。本书从最基础的 Python 语法讲起，循序渐进地引导学生探索 Python 在金融数据分析、投资组合优化、金融文本挖掘等核心业务场景中的应用，让技术真正服务于金融实践。

本书的特色具体表现为以下几点。

1. 以金融场景为导向，理论与实践深度融合

本书以金融领域的实际应用场景为核心，立足于金融业务的实际需求，将 Python 编程技术与金融业务需求紧密结合。每个项目均围绕金融行业的典型任务展开，通过真实案例引导学生从理论到实践逐步掌握 Python 在金融中的具体应用，帮助学生掌握金融科技时代必备的编程技能和业务思维。学生不仅能够学习 Python 的基础语法，还能通过实际数据计算投资组合的最优配置，真正实现“学以致用”。

2. 从基础到进阶，循序渐进的学习路径

本书的内容设计遵循“由浅入深”的原则，从 Python 基础语法入手，逐步过渡到金融领域的高级应用。项目一重点讲解 Python 的基础知识，如变量、数据类型、控制结构、函数与模块等，确保零基础学生也能轻松入门。其他项目则深入探讨 Python 在金融数据分析、数据挖掘、爬虫技术等领域的应用，帮助学生构建完整的知识体系，为未来的职业发展奠定坚实基础。

3. 丰富的实战案例，强化动手能力

本书注重实践能力的培养，每个项目均包含多个实战案例，涵盖金融领



域的多个细分方向。通过这些实战案例，学生不仅能够巩固所学知识，还能提升解决实际问题的能力。

4. 融入前沿技术，紧跟行业发展趋势

通过 Python 基础知识、网络数据爬虫技术、数据库基础及实战、大数据分析技巧、大数据风控等多方面金融与 Python 结合的内容，全面讲解了金融数据的获取、处理、分析及结果呈现。本书学时分配建议如下。

内 容	学 时
项目一 Python 基础	8
项目二 爬虫基础 + 金融商业案例实战	8
项目三 数据分析基础 + 商业案例实战	8
项目四 爬虫进阶 + 金融商业案例实战	12
项目五 金融数据分析进阶 + 商业案例实战	12
总计	48

本书由长春职业技术大学的赵阳、邱小津担任主编，李庚倩、孙悦、于翠担任副主编。具体编写分工为：项目一、项目四由赵阳和李庚倩编写；项目二、项目三、项目五由邱小津、于翠和孙悦编写。赵阳负责全书统稿。

尽管我们在教材的特色建设方面做出了很多努力，但难免存在不足之处，恳请各相关高职院校教师和学生在使用本书的过程中予以关注，并将意见或建议及时反馈给我们，以便修订时完善，在此表示由衷的感谢。

编 者



目 录

项目一	Python基础.....	1
模块一	Python初了解与安装	2
模块二	Python基础知识	8
模块三	Python最重要的三大语句.....	12
模块四	Python函数与模块.....	17
模块五	综合实战1-大数据分词与词云图绘制.....	19
模块六	综合实战2-文字识别、人脸识别实战	22
项目二	爬虫基础+金融商业案例实战.....	27
模块一	爬虫基础知识	28
模块二	金融数据挖掘实战-百度新闻深度挖掘	35
模块三	金融数据挖掘实战-正则表达式提取百度新闻	39
模块四	金融数据挖掘实战-搜狐&新浪数据挖掘	42
模块五	舆情评分系统+文本分析技巧	48
项目三	数据分析基础+商业案例实战.....	54
模块一	数据分析的武器NumPy&Pandas库.....	55
模块二	Pandas库数据清洗	70
模块三	Pandas库数据分析	76
模块四	Pandas库办公自动化	91
模块五	案例实战一对黄金ETF联接基金价格进行预测	101



项目四	爬虫进阶+金融商业案例实战	111
模块一	爬虫利器：Selenium库基础&进阶操作	112
模块二	新浪财经股票实时数据挖掘实战	127
模块三	东方财富网数据挖掘	137
模块四	股权穿透Python代码实现	145
模块五	豆瓣类平台短评数据挖掘实战	155
项目五	金融数据分析进阶+商业案例实战	165
模块一	数据可视化初步	166
模块二	案例实战-国债收益率曲线分析	193
模块三	案例实战-舆情评分与股价相关性分析	200
模块四	案例实战-上市公司财务健康度分析	208
模块五	案例实战-上市公司前十大控股股东分析	215
参考文献		276





项目一 Python 基础



知识目标

- 了解 Python 编程语言；
- 了解 Python 最重要的三大语句。



能力目标

- 能够完成 Python 程序的安装；
- 能够完成自己的第一个 Python 程序。



素养目标

- 学习 Python 语言知识，具备计算机逻辑思维；
- 了解 Python 为金融实务提供的帮助。

Python 是一种高级编程语言，由荷兰的吉多·范罗苏姆（Guido van Rossum，以下简称吉多）于 1989 年开始开发，第一个公开发行版发布于 1991 年。Python 具有语法简洁、易于学习、功能强大、可扩展性强、跨平台等诸多特点，因此它成为最受欢迎的程序设计语言之一。



模块一

Python 初了解与安装



情景设置

小明是一名金融专业的学生，最近在学习数据分析时，发现很多金融领域的工具和库都是用 Python 编写的。他的老师告诉他，Python 不仅简单易学，而且在金融数据分析、量化交易、风险管理等领域有广泛的应用。小明决定开始学习 Python，但他对 Python 不太了解。那么，Python 到底有什么优势？如何安装 Python 呢？



必备知识

Python 的设计哲学强调代码的可读性和简洁的语法（尤其是使用空格缩进划分代码块，而非使用大括号或关键词）。Python 支持多种编程范式，包括面向对象、命令式、函数式和过程式编程。

一、Python 简介

Python 的第一个公开版本于 1991 年发布，该版本使用 C 语言实现，并且能够调用 C 语言的库 (library) 文件。Python 的语法中融入了 C 语言的许多特性，同时它也受到了 ABC 语言的影响。自诞生以来，Python 已经具备了 class(类)、function(函数)、exception(异常处理)，以及 list(列表) 和 dict(字典) 等核心数据类型，还有基于模块 (module) 的扩展系统。

最初，Python 是由吉多本人独自研发的，吉多的同事们只是使用 Python 并提出反馈意见，后来他们逐渐体会到 Python 的魅力，纷纷加入 Python 语言的改进工作中。随着 Python 的发布，越来越多的人开始被 Python 吸引，Python 的用户和开发团队也逐步壮大。

2000 年 10 月，Python2.0 发布，Python 从基于 maillist(邮件列表) 的开发方式转变为完全开源的开发方式，这标志着 Python 社区的成熟。随后，Python 2.x 系列陆续发布多个版本，该系列的最后一个版本是 2010 年发布的 Python 2.7。2010 年以后，Python 的维护者们宣布不再推出新的主版本号升级版本，Python2.x 系列慢慢退出历史舞台。2018 年 3 月，吉多宣布将于 2020 年终止对 Python 2.7 的技术支持。

2008 年 12 月 Python3.0 发布，Python 3.0 在语法上和解释器内部都做了很多重大改进。Python 3.0 移除了一些旧的语法和特性，优化了字符串处理和 Unicode 支持等方面的内容。此外，其解释器内部广泛支持面向对象的编码方式，以提高代码的可扩展性和可维护性。这些重大改进促使 Python 3.0 成为一个新的主要版本，为用户提供更好的编程体验。

随后，Python 3.x 系列陆续发布多个版本，包括 Python 3.8、Python 3.9、Python 3.10、Python3.11 等，这些版本有许多新特性和改进，推动了 Python 的进一步发展。截至 2025 年 1 月 1 日，Python 最新稳定版是 3.13.0，于 2024 年 10 月 7 日正式发布。

随着时间的推移，Python 不断演化并发展，成为一种广泛应用于各个领域的高级编程语言。未来，随着人工智能、大数据和云计算等领域的不断发展，Python 将继续发挥重要作用，并成为技术创新和应用的关键驱动力。

二、Python 的安装

要开始使用 Python，首先需要在计算机上安装 Python 环境，安装步骤如下。

(1) 访问 Python 官网。打开浏览器，访问 Python 官方网站 (<https://www.python.org/>)，如图 1-1 所示。

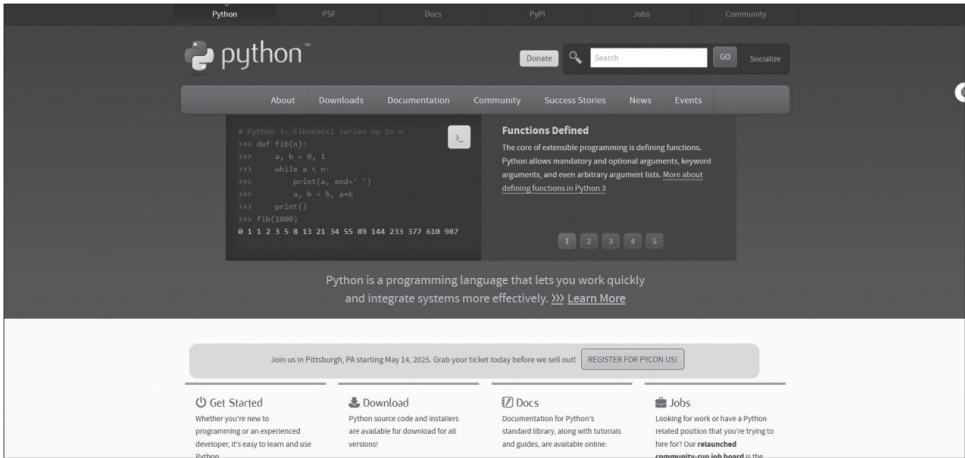


图 1-1 Python 官网

(2) 下载 Python 安装包。在官网首页单击“Downloads”按钮，选择符合操作系统的 Python 版本（Windows、macOS、Linux 等），如图 1-2 所示。

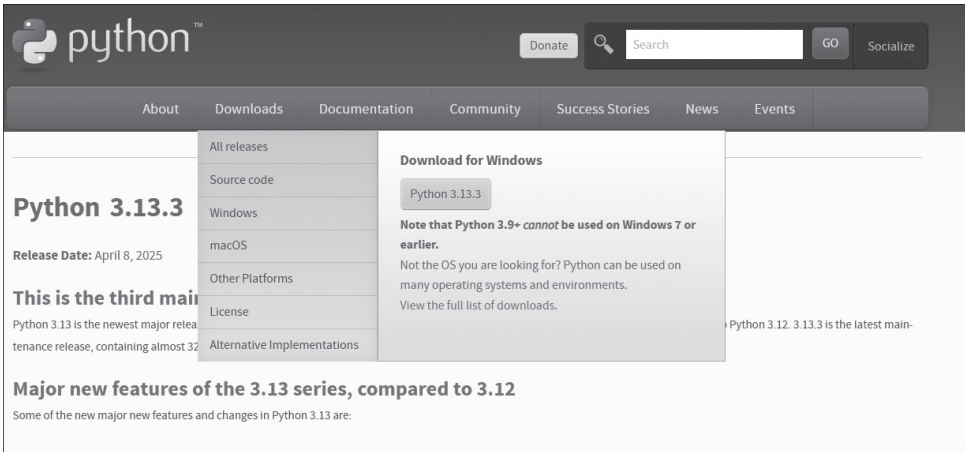


图 1-2 Python 版本下载页面



(3) 运行安装程序。下载完成后，双击安装包，按照提示进行安装。在安装过程中，确保选中“Add Python to PATH”复选框，以便在命令行中直接使用 Python，如图 1-3 所示。

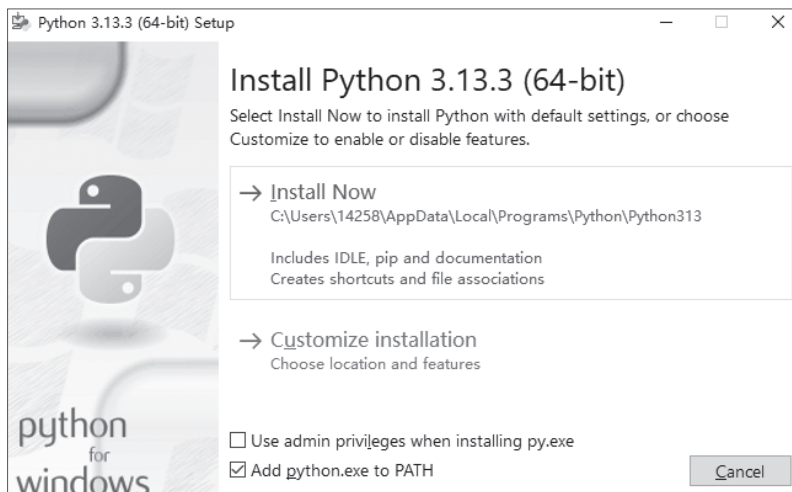


图 1-3 选中“Add Python to PATH”复选框

(4) 验证安装：安装完成后，打开命令行（Windows 上是 CMD 或 PowerShell，macOS 和 Linux 上是 Terminal），输入“python --version”，如果显示 Python 版本号，则说明安装成功。



拓展知识

Python 的语言特点如下。

1. 优点

简单：Python 是一种代表简单主义思想的语言，它让使用者能够专注于解决问题而不是去搞明白语言本身。

易学：Python 极其容易上手，因为 Python 有极其简单的说明文档。

易读、易维护：风格清晰划一、强制缩进。

速度较快：Python 的底层是用 C 语言写的，很多标准库和第三方库也都是用 C 写的。

免费、开源：Python 是 FLOSS（自由 / 开放源码软件）之一。使用者可以自由地发布这个软件的拷贝、阅读它的源代码、对它做改动、把它的一部分用于新的自由软件中。FLOSS 是基于一个团体分享知识的概念。

高层语言：用 Python 语言编写程序的时候无须考虑如何管理程序使用的内存等底层细节。

可移植性：由于它的开源本质，Python 已经被移植在许多平台上（经过改动使它能够工作在不同平台上）。这些平台包括 Linux、Windows、FreeBSD、Macintosh、Solaris、



OS/2、Amiga、AROS、AS/400、BeOS、OS/390、z/OS、Palm OS、QNX、VMS、Psion、Acom RISC OS、VxWorks、PlayStation、Sharp Zaurus、Windows CE、PocketPC、Symbian 以及 Google 基于 linux 开发的 Android 平台。

解释性：一个用编译性语言（如 C 或 C++）写的程序可以从源文件转换到一个计算机使用的语言（二进制代码，即 0 和 1），+ 这个过程通过编译器和不同的标记、选项完成。

运行程序的时候，连接 / 转载器软件把程序从硬盘复制到内存中并且运行。而 Python 语言写的程序不需要编译成二进制代码。使用者可以直接从源代码运行程序。

在计算机内部，Python 解释器把源代码转换成称为字节码的中间形式，然后再把它翻译成计算机使用的机器语言并运行。这使得使用 Python 更加简单。也使得 Python 程序更加易于移植。

面向对象：Python 既支持面向过程的编程也支持面向对象的编程。在“面向过程”的语言中，程序是由过程或仅仅是可重用代码的函数构建起来的。在“面向对象”的语言中，程序是由数据和功能组合而成的对象构建起来的。

Python 是完全面向对象的语言。函数、模块、数字、字符串都是对象。并且完全支持继承、重载、派生、多继承，有益于增强源代码的复用性。Python 支持重载运算符和动态类型。相对于 Lisp 这种传统的函数式编程语言，Python 对函数式设计只提供了有限的支持。有两个标准库（functools，itertools）提供了 Haskell 和 Standard ML 中久经考验的函数式程序设计工具。

可扩展性、可扩充性：如果需要一段关键代码运行得更快或者希望某些算法不公开，可以部分程序用 C 或 C++ 编写，然后在 Python 程序中使用它们。

Python 本身被设计为可扩充的。并非所有的特性和功能都集成到语言核心。Python 提供了丰富的应用程序接口（API）和工具，以便程序员能够轻松地使用 C 语言、C++、Cython 来编写扩充模块。Python 编译器本身也可以被集成到其他需要脚本语言的程序内。因此，很多人还把 Python 作为一种“胶水语言”使用。使用 Python 将其他语言编写的程序进行集成和封装。在 Google 内部的很多项目，如 Google Engine，使用 C++ 编写性能要求极高的部分，然后用 Python 或 Java/Go 调用相应的模块。

可嵌入性：可以把 Python 嵌入 C/C++ 程序，从而向程序用户提供脚本功能。

丰富的库：除了标准库以外，还有许多其他高质量的库，如 wxPython、Twisted 和 Python 图像库等等。Python 标准库确实很庞大，它可以帮助处理各种工作，包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、公共网关接口（CGI）、文件传送协议（FTP）、电子邮件、XML、XML-RPC、HTML、WAV 文件、密码系统、



图形用户界面 (GUI)、Tk 和其他与系统有关的操作。

规范的代码：Python 采用强制缩进的方式使得代码具有较好可读性而 Python 语言写的程序不需要编译成二进制代码。Python 的作者设计限制性很强的语法，使得不好的编程习惯（例如 if 语句的下一行不向右缩进）都不能通过审查。其中很重要的一项就是 Python 的缩进规则。Python 和其他大多数语言如 C 的区别就是，一个模块的界限完全是由每行的首字符在这一行的位置来决定，而 C 语言是用一对大括号来明确的定出模块的边界，与字符的位置毫无关系。通过强制程序员们缩进（包括 if, for 和函数定义等所有需要使用模块的地方），Python 程序确实更加清晰和美观。

高级动态编程：虽然 Python 可能被粗略地分类为“脚本语言”（script language），但实际上一些大规模软件开发计划，如 Zope、Mnet 及 BitTorrent，Google 也广泛地使用它。Python 的支持者喜欢称它是一种高级动态编程语言，原因是“脚本语言”泛指仅作简单程序设计任务的语言，如 shellscript、VBScript 等只能处理简单任务的编程语言，并不能与 Python 相提并论。

做科学计算优点多：说起科学计算，首先会被提到的可能是 MATLAB。除了 MATLAB 的一些专业性很强的工具箱还无法被替代之外，MATLAB 的大部分常用功能都可以在 Python 世界中找到相应的扩展库。和 MATLAB 相比，用 Python 做科学计算有以下优点。

首先，MATLAB 是一款商用软件，并且价格不菲。而 Python 完全免费，众多开源的科学计算库都提供了 Python 的调用接口。用户可以在任何计算机上免费安装 Python 及其绝大多数扩展库。

其次，与 MATLAB 相比，Python 是一门更易学、更严谨的程序设计语言，它能让用户编写出更易读、易维护的代码。

最后，MATLAB 主要专注于工程和科学计算。然而即使在计算领域，也经常会遇到文件管理、界面设计、网络通信等各种需求。而 Python 有着丰富的扩展库，可以轻松完成各种高级任务，开发者可以用 Python 实现完整应用程序所需的各种功能。

2. 缺点

单行语句和命令行输出问题：很多时候不能将程序连写成一行，如 `import sys; for i in sys.path: print i`。而 perl 和 awk 就无此限制，可以较为方便地在 shell 下完成简单程序，不需要如 Python 一样，必须将程序写入一个 .py 文件。

给初学者带来困惑：独特的语法，这也许不应该被称为局限，但是它用缩进来区分语句关系的方式还是给很多初学者带来了困惑。即便是很有经验的 Python 程序员，也可能陷入陷阱当中。



运行速度慢：这里指与 C 和 C++ 相比。Python 开发人员尽量避开不成熟或不重要的优化。一些针对非重要部位的加快运行速度的补丁通常不会被合并到 Python 内。所以很多人认为 Python 很慢。不过，根据二八定律，大多数程序对速度要求不高。在某些对运行速度要求很高的情况，Python 设计师倾向于使用 JIT 技术，或者使用 C/C++ 语言改写这部分程序。可用的 JIT 技术是 PyPy。

3. 和其他语言的区别

对于一个特定的问题，只要有一种最好的方法来解决。这在由 Tim Peters 写的 Python 格言（称为 The Zen of Python）里面表述为：There should be one-and preferably only one-obvious way to do it。这正好和 Perl 语言（另一种功能类似的高级动态语言）的中心思想 TMTOWTDI（There's more than one way to do it）完全相反。

Python 的设计哲学是“优雅”“明确”“简单”。Python 开发者的哲学是“用一种方法，最好是只有一种方法来做一件事”。在设计 Python 语言时，如果面临多种选择，Python 开发者一般会拒绝花哨的语法，而选择明确的、没有或者很少有歧义的语法。由于这种设计观念的差异，Python 源代码通常被认为比 Perl 具备更好的可读性，并且能够支撑大规模的软件开发。在 Python 解释器内运行 `import this` 可以获得完整的列表。

更高级的虚拟机。Python 在执行时，首先会将 .py 文件中的源代码编译成 Python 的字节码（byte code），然后再由 Python 虚拟机来执行这些编译好的字节码。这种机制的基本思想跟 Java，.NET 是一致的。然而，Python 虚拟机与 Java 或 .NET 的虚拟机不同的是，Python 虚拟机是一种更高级的虚拟机。这里的高级并不是通常意义上的高级，不是说 Python 虚拟机比 Java 或 .NET 的功能更强大，而是说和 Java 或 .NET 相比，Python 虚拟机距离真实机器的距离更远。或者可以这么说，Python 虚拟机是一种抽象层次更高的虚拟机。基于 C 的 Python 编译出的字节码文件，通常是 .pyc 格式。除此之外，Python 还可以以交互模式运行，比如主流操作系统 Unix/Linux、Mac、Windows 都可以直接在命令模式下直接运行 Python 交互环境。直接下达操作指令即可实现交互操作。



模块自测

一、单选题

1. 在 Windows 系统中，通常可以从以下哪个网站下载 Python 官方安装程序？（ ）
A. <https://www.python.org/>
B. <https://python.com/>
C. <https://www.anaconda.com/>



D. <https://www.jetbrains.com/>

2. 安装 Python 时，在安装向导中选择“Add Python to PATH”选项的作用是（ ）。

A. 将 Python 安装到系统路径下

B. 允许在命令行中直接使用 Python 命令

C. 安装 Python 的扩展库

D. 提高 Python 的运行速度

3. 以下（ ）不是 Python 的安装方式。

A. 使用官方安装程序

B. 使用包管理工具（如 apt、yum 等）

C. 从源代码编译安装

D. 使用 Word 文档安装

二、简答题

简述 Python 的安装过程。

模块二

Python 基础知识



情景设置

小明在学习金融数据分析时，发现数据通常以数字、文本等形式存储。他需要了解如何在 Python 中存储和处理这些数据。例如，如何存储股票的价格数据？如何计算股票的平均价格？



必备知识

变量用于在 Python 程序中保存数据。在 Python 中，所有的数据都是对象，变量保存的是对对象的引用。

根据数据存储形式的不同，Python 中常用的数据类型可以分为两类，分别是数字类型和组合数据类型，其中组合数据类型相对复杂。

一、变量与数据类型

1. 变量

变量一词来源于数学，在计算机语言中是能储存计算结果或能表示值的抽象概念。简



单来说,变量可以看作是用来表示值的盒子,不同的变量代表了储存在计算机内存中不同的信息。在 Python 中,变量是用来存储数据的容器。

声明变量就是给变量赋值的过程,即给变量起个名字,然后给它一个数据,Python 中变量是通过“=”来赋值的。

2. 数据类型

Python 采用动态类型系统,在编译时不会检查对象是否拥有被调用的方法或属性,而是直至运行时才做出检查,所以操作对象时可能会抛出异常。不过,虽然 Python 采用动态类型系统,它同时也是强类型的。Python 禁止没有明确定义的操作,如数字加字符串。

与其他面向对象语言一样,Python 允许程序员定义类型。构造一个对象只需要像函数一样调用类型即可。类型本身也是特殊类型 `type` 的对象(`type` 类型本身也是 `type` 对象),这种特殊的设计允许对类型进行反射编程。

Python 中的数据类型主要包括以下几种。

(1) 整数(int): 整数类型简称整型,英文使用 `int` 表示,它表示整数。整型在 Python 中的取值范围没有限制,可以表示任意大的整数。它有四种进制,分别是十进制、二进制、八进制和十六进制。

(2) 浮点数(float): 浮点数类型简称浮点型,英文使用 `float` 表示,主要用来表示实数。Python 中的浮点型必须带有小数部分,小数部分可以为 0。例如,456 是整数类型,而 456.0 是浮点数类型。

(3) 布尔值(bool): 表示真(True)或假(False)的值,用于逻辑运算,例如,`is_true=True`。

(4) 字符串(str): 计算机程序需要处理的文本信息使用字符串表示,在 Python 中,字符串分为单行字符串和多行字符串两种。前者使用一对单引号(')或者一对双引号(")作为定界符,而后者使用一对三个单引号('')或者三个双引号("")作为定界符,不同的定界符之间还可以互相嵌套。文本数据可以使用单引号或双引号表示,例如,`text="Hello, World!"`。

(5) 列表(list): 序列是 Python 中最基本的数据结构之一,而列表是其典型代表之一。列表是包含 0 个或多个元素的有序序列,列表类型使用中括号表示,没有长度限制,列表中的元素的类型可以各不相同。例如,`colorlist=[99, 88, 'green']`。

(6) 元组(tuple): Python 中的元组与列表类似,不同之处在于元组的元素不能修改。元组使用小括号,而列表使用方括号。元组的创建非常简单,只需要在小括号里添加元素,用逗号分隔即可。当元组中只包含一个元素时,需要在元素后面添加逗号,否则括号会被当作运算符使用。例如,`my_tuple=(1, 2, 'red')`。

(7) 集合(set): 计算机中的集合与数学中的集合概念相同,是指一个无序不重复的元

素序列，使用一对大括号 {} 表示。例如，`my_set = {1, 2, 3}`。

(8) 字典 (dict)：Python 中的字典是一种序列类型，它和列表、元组的不同之处在于，列表、元组是通过索引来定位元素，而字典是由键值对组成，它通过键来查找并定位元素。

字典能够存储任意类型对象，字典的每个键值对用冒号进行分割，每个键值对之间用效号进行分割，整个字典包括在花括号 {} 中，例如，`my_dict = {'name': 'Bob', 'age': 22}`。

(9) 复数 (complex)：包含实部和虚部的数值，例如，`z = 2 + 3j`。

(10) 字节串 (bytes)：以字节为单位的不可变序列，例如，`b = b'python'`。

(11) 字节数组 (bytearray)：以字节为单位的可变序列，例如，`ba = bytearray(b'python')`。

其中，简单结构的数据类型包括整数、浮点数、布尔值、字符串，多维结构的数据类型包括列表、元组、集合、字典。

二、运算符

Python 支持多种运算符，包括算数运算符、比较运算符、逻辑运算符等，如表 1-1 ~ 表 1-3 所示。

表 1-1 算术运算符及其相关说明

算数运算符	说 明	运算符号	示 例	结 果
幂	返回 x 的 y 次幂	**	<code>3**2</code>	9
乘	两数相乘	*	<code>4*6</code>	24
除	x 除以 y	/	<code>6.0/3.0</code> <code>6.0/3</code> <code>6/3</code>	2.0 2.0 2.0
整除	返回商的整数部分	//	<code>5//2</code>	2
加	两数相加	+	<code>3+2</code>	5
减	两数相减	-	<code>10-7</code>	3
取模	返回除法的余数	%	<code>5%2</code>	1

表 1-2 比较运算符及其相关说明

比较运算符	功 能	例 子	结 果
<	小于	<code>1<2</code> <code>3<2</code>	True False
<=	小于等于	<code>2<=2</code> <code>3<=2</code>	True False
==	等于	<code>2==2</code> <code>3==2</code>	True False

续表

比较运算符	功 能	例 子	结 果
>=	大于等于	3>=2	True
		2>=3	False
>	大于	3>2	True
		2>3	False
!=	不等于	3!=2	True
		2!=2	False

表 1-3 算术运算符及其相关说明

运算符	说 明	运算符号	示 例	结 果
非	取反，真为假，假为真	not	not(8>6)	False
与	同时为真，才为真	and	1>2 and 3>2	False
或	其一为真，即为真	or	3<5 or 8<5	True

三、控制语句

- （1）if 语句：当条件成立时运行语句块。经常与 else、elif（相当于 else if）配合使用，称为 if-elif-else 语句。
- （2）for 语句：遍历列表、字符串、字典、集合等迭代器（容器），依次处理迭代器中的每个元素。有时和 else 连用，称为 for-else 语句。
- （3）while 语句：当条件为真时，循环运行语句块。有时和 else 配合使用，称为 while-else 语句。
- （4）try 语句：必与 except 配合使用处理在程序运行中出现的异常情况，称为 try-except 语句。常与 else、finally 配合使用，称为 try-except-else 语句、try-except-finally 语句、try-except-else-finally 语句
- （5）class 语句：用于定义类型。
- （6）def 语句：用于定义函数和类型的方法。
- （7）pass 语句：表示此行为空，不运行任何操作。
- （8）assert 语句：用于程序调试阶段时测试运行条件是否满足。
- （9）with 语句：Python2.6 以后定义的语法，在一个场景中运行语句块。例如，运行语句块前加密，然后在语句块运行退出后解密。
- （10）yield 语句：在迭代器函数内使用，用于返回一个元素。Python 2.5 版本后，该语句变成一个运算符。
- （11）raise 语句：制造一个错误。



- ## 模块自测

D. bool

模块三 Python 最重要的三大语句



必备知识

12

行的称为顺序结构。分支结构则根据条件执行不同的代码。循环结构重复执行相同的代码。Python 用 if 语句实现分支结构，用 for 和 while 语句实现循环结构。

一、if 语句

1.if 单分支结构

Python 单分支结构主要使用 if 保留字对条件进行判断，其基本格式如下：

```
if <表达式>:  
    <语句块>  
<语句 a>
```

<语句块> 中的语句左对齐，与 if 错开一个“Tab”距离，<语句 a> 与 if 左对齐。如果表达式成立（条件满足），即为真 (True)，则执行 <语句块>，并执行 if 的后续语句 <语句 a>；如果不成立，即为假 (False)，则跳过 <语句块>，执行 if 的后续语句 <语句 a>。其流程图如图 1-4 所示。

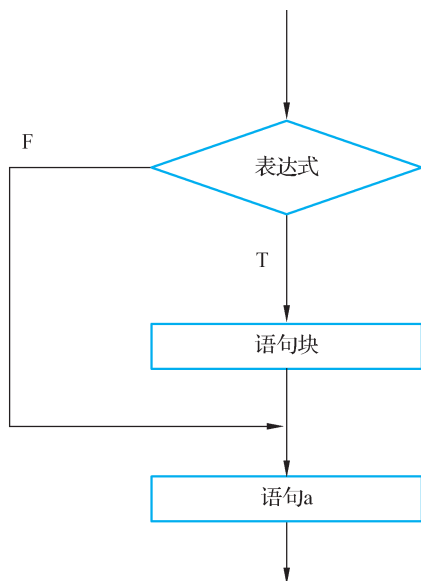


图 1-4 单分支结构流程图

例如，输出 a、b 两个数中的较大值，使用 if 语句进行判断。

如果 a 比 b 小，则将 a、b 的值进行交换，从而保证 a 是两数中的较大值，最后输出 a 的值，代码如下。

```
# 输出 a,b 中的较大值  
a=eval(input('请输入 a 的值:'))  
b=eval(input('请输入 b 的值:'))  
if a<b:
```



```
a,b=b,a  
print(a)
```

2. if-else 二分支结构

Python 二分支结构主要使用 if-else 对条件进行判断,其基本格式如下。

```
if <表达式>:  
    <语句块 1>  
else:  
    <语句块 2>  
<语句 a>
```

if、else 和 <语句 a> 左对齐; <语句块 1> 和 <语句块 2> 左对齐,与 if 和 else 错开一个“Tab”距离。如果表达式成立,即为真 (True),则执行 <语句块 1>,接着执行 <语句 a>;如果条件不满足,即为假 (False),则执行 <语句块 2>,接着执行 <语句 a>。其流程图如图 1-5 所示。

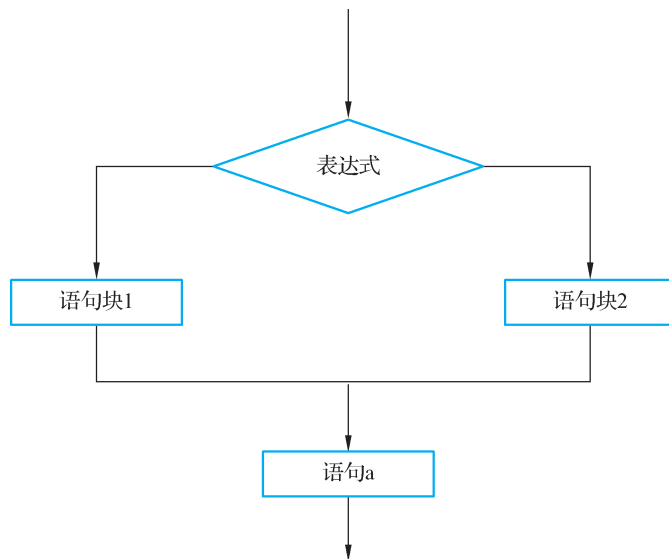


图 1-5 二分支结构流程图

例如,输出 a、b 两个数中的较大值,使用 if-else 语句进行判断。

如果 a 比 b 小,则输出 b; 否则输出 a,代码如下。

```
# 输出 a.b 中的较大值  
a=eval(input('请输入 a 的值:'))  
b=eval(input('请输入 b 的值:'))  
If a<b:  
    print(b)
```

```
Else:  
    print(a)
```

3. if-elif-else 多分支结构

Python 多分支结构主要使用 if-elif-else 对条件进行判断, 并根据不同条件的结果按照顺序选择执行路径, 其基本格式如下。

```
if< 表达式 1>:  
    < 语句块 1>  
elif< 表达式 2>:  
    < 语句块 2>  
.....  
else:  
    < 语句块 n>  
< 语句 a>
```

if、elif、else 和 < 语句 a> 左对齐; < 语句块 1> < 语句块 2> 和 < 语句 n> 左对齐, 与 if、elif、else 错开一个“Tab”距离。elif 子句可以有多个, 但是必须在 if 和 else 子句之间。其流程图如图 1-6 所示。

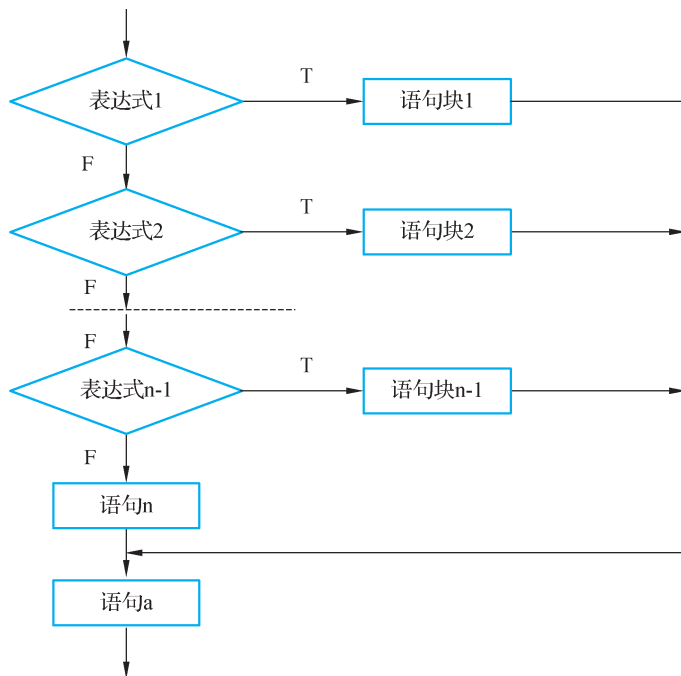


图 1-6 多分支结构流程图

例如, 从键盘上获取用户的输入, 并将其转化为五分制成绩, 使用多分支的格式编程实现。



将输入的百分制成绩转换成五分制，代码如下。

```
score=eval(input("请输入一个百分制成绩："))
if score>=90:
    grade='A'
elif score>=80:
    grade='B'
elif score>=70:
    grade='C'
elif score>=60:
    grade='D'
else:
    grade='E'
print('对应的五分制成绩是：{}'.format(grade))
```

多分支结构的代码顺序编程时要特别注意，程序执行时会寻找并执行第一个符合条件的语句块，然后就跳出 if-elif-else 结构，如果没有条件符合会执行 else 分支，它是可选的分支。例如，在上面程序的序中，如果输入 95，符合第一个分支的条件，因此输出等级 A；如果输入 -12，不符合 if 或 elif 的分支，最终只能匹配到 else 分支，输出等级 E。

二、for 语句

for 语句用于遍历序列（如列表、元组、字符串等）或其他可迭代对象。示例代码如下。

```
python
# 遍历股票价格列表
prices = [150, 160, 155, 170]
for price in prices:
    print("当前股票价格：", price)
```

三、while 语句

while 语句用于在条件为真时重复执行某段代码。示例代码如下。

```
python
# 模拟股票价格波动
current_price = 150
target_price = 170
while current_price < target_price:
```



```
print(" 当前价格 :", current_price)
current_price += 5
print(" 目标价格已达到! ")
```



模块自测

一、判断题

1. if 语句的 else 子句必须与 if 条件关联，不能单独使用。 ()
2. for 循环只能遍历列表、元组等序列类型，不能遍历字典。 ()
3. 在 while 循环中，break 语句用于跳过本次循环的剩余代码，直接进入下一次循环。 ()

二、简答题

举例说明 if-elif-else 多分支结构流程。



模块四 Python 函数与模块



情景设置

小明在分析股票数据时，发现很多计算是重复的，如计算股票的平均价格、最大价格等。他希望能够将这些计算封装成一个函数，方便重复调用。此外，他还听说 Python 有很多强大的模块可以直接使用，如数学计算模块 math。



必备知识

在程序开发中，函数是组织好的、用于实现单一功能或相关联功能的代码段。在前面我们已经接触过一些函数，例如，用于向控制台输出内容的 print() 函数、用于接收用户输入内容的 input() 函数等。我们可以将函数视为一段有名字的代码，这类代码可以在需要的地方以“函数名 ()”的形式调用。

一、函数

函数是组织好的、可重复使用的代码块，用于实现单一或相关联的功能。示例代码如下。



```
python
# 定义计算平均价格的函数
def calculate_average_price(prices):
    return sum(prices) / len(prices)
# 调用函数
prices = [150, 160, 155, 170]
average_price = calculate_average_price(prices)
print("平均价格:", average_price) # 输出: 158.75
```

二、模块

模块是一个包含 Python 定义和语句的文件，其文件名是模块名加上“.py”后缀。通过导入模块，可以使用模块中定义的函数、类等。示例代码如下。

```
python
# 导入 math 模块
import math
# 使用 math 模块中的函数
print(math.sqrt(256)) # 输出: 16.0
```



模块自测

一、单选题

1. 以下关于 Python 函数定义的语法，正确的是（ ）。
A. function my_function(): B. def my_function();
C. def my_function(): D. my_function = def():
2. 若要在函数中返回多个值，最常用的方式是（ ）。
A. 使用多个 return 语句 B. 返回一个列表或元组
C. 使用全局变量传递值 D. 无法返回多个值
3. 导入 math 模块后，若要使用 sqrt 函数计算平方根，正确的调用方式是（ ）。
A. sqrt(4) B. math.sqrt(4)
C. import math.sqrt(4) D. math::sqrt(4)

二、简答题

简述函数与模块的关系。



模块五

综合实战 1- 大数据分词与词云图绘制



情景设置

小明在学习金融新闻分析时，发现很多新闻文本中包含大量的关键词。他希望能够对这些文本进行分词，并绘制词云图，以便更直观地展示关键词的重要性。



必备知识

jieba 库和 wordcloud 库是 Python 中常用的第三方库，第三方库在使用之前需要先安装。

一、jieba 库

jieba 库用于实现中文分词，中文分词即将中文语句或语段拆分成若干汉语词语。例如，用户输入的语句“我是一名学生”经分词系统处理之后，被分成“我”“是”“一名”“学生”这 4 个汉语词语。jieba 支持以下 3 种分词模式。

- (1) 精确模式：试图将句子精准地拆分开。
- (2) 全模式：将句子中所有可以成词的词语都扫描出来，速度非常快。
- (3) 搜索引擎模式：在精确模式的基础上对长词再次拆分，适用于建立搜索引擎的索引。

【例 1-8】通过上述 3 种分词模式对中文语句“我打算到中国教育科学研究院图书馆学习”进行分词，示例代码如下。

```
import jieba

seg_list = jieba.cut("我打算到中国教育科学研究院图书馆学习", cut_all=True)

print ("【全模式】:" + "/ ".join (seg_list))    # 全模式
seg_list = jieba.lcut("我打算到中国教育科学研究院图书馆学习")
print ("【精确模式】:" + "/ ".join (seg_list))  # 精确模式
# 搜索引擎模式

seg_list = jieba.cut_for_search("我打算到中国教育科学研究院图书馆学习")
print ("【搜索引擎模式】:" + ", ".join (seg_list)) 运行程序，结果如下所示：
```

```
[全模式]: 我 / 打算 / 算到 / 中国 / 国教 / 教育 / 教育科 / 科学 / 科学研究 / 研究 /
研究院 / 图书 / 图书馆 / 图书馆学 / 书馆 / 学习
```

「精确模式」：我 / 打算 / 到 / 中国 / 教育 / 科学 / 研究院 / 图书馆 / 学习

[搜索引擎模式]:我,打算,到,中国,教育,科学,研究,研究院,图书,书馆,图书馆,学习

二、wordcloud 库

wordcloud 库用于生成词云图。词云图是一种以词语频率为基础的可视化方式，通过将文本中出现频率较高的词语以更大的字体展示，呈现出词语在文本中的重要性和流行程度。词云图常被用于分析和展示大量文本的关键词，帮助人们直观地了解文本的主题、热点和关键话题等。

wordcloud 库会将文本中的词语出现的频率作为参数来绘制词云，它基于 Pillow 和 Matplotlib 库，支持对词云的形状、颜色和大小等属性进行设置。

根据上述文字生成词云图，结果如图 1-7 所示。



图 1-7 词云图

利用 wordcloud 库生成词云图主要包含 3 个步骤。

利用 WordCloud 类的构造方法 WordCloud() 创建词云对象。

利用 WordCloud 对象的 generate() 方法加载词云图使用的文本。

利用 WordCloud 对象的 to file() 方法生成词云图。

以上步骤用到的 WordCloud() 方法在创建词云对象时可通过参数设置词云的属性。

`Generate()` 方法需要接收一个字符串作为参数，字符串中的内容便是词云图使用的文本。若字符串中的内容全部为汉字，那么在创建词云对象时必须通过 `font_path` 参数指定字体文件的路径，不指定的话将无法显示汉字。

`to_file()` 方法用于以图片形式输出词云图。该方法接收一个表示图片名的字符串作为参数，图片的格式可以为 PNG 或 JPEG 格式。



案例实现

使用 jieba 库进行分词，使用 wordcloud 库绘制词云图。



```
python
import jieba
from wordcloud import WordCloud
import matplotlib.pyplot as plt
# 文章内容
text = ""
```

wordcloud 库用于生成词云图。词云图是一种以词语频率为基础的可视化方式，通过将文本中出现频率较高的词语以更大的字体展示，呈现出词语在文本中的重要性和流行程度。词云图常被用于分析和展示大量文本的关键词，帮助人们直观地了解文本的主题、热点和关键词题等。

wordcloud 库会将文本中的词语出现的频率作为参数来绘制词云，它基于 Pillow 和 Matplotlib 库，支持对词云的形状、颜色和大小等属性进行设置。

```
""
# 分词
words = " ".join(jieba.cut(text))
# 生成词云
wordcloud=WordCloud(width=800,height=400, font_path='simhei.ttf'
).generate(words)
# 显示词云图
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show( )
```



模块自测

一、单选题

1. jieba 库的主要功能是 ()。
A. 数据可视化
B. 中文分词
C. 网络爬虫
D. 机器学习
2. 使用 jieba 进行分词时，默认采用的分词模式是 ()。
A. 精确模式
B. 全模式
C. 搜索引擎模式
D. 混合模式
3. wordcloud 库主要用于生成 ()。
A. 柱状图
B. 词云图



- C. 散点图
- D. 热力图
4. 以下 () 是 jieba 库中用于添加自定义词典的函数。
- A. add_word()
- B. load_userdict()
- C. cut()
- D. suggest_freq()
5. 在 wordcloud 库中, 若要设置词云图的背景颜色为白色, 应使用 ()。
- A. background_color="white"
- B. color="white"
- C. bg_color="white"
- D. back_color="white"

谈谈你对词云图的理解。

模块六 综合实战 2- 文字识别、人脸识别实战



小明在学习金融安全时，发现很多金融机构使用图像识别技术来验证客户身份。他希望能够用 Python 实现一个简单的文字识别和人脸识别程序。



pytesseract 和 opencv 是 Python 中用于图像处理和光学字符识别（OCR）的常用库。

一、pytesseract 库

pytesseract 是 Tesseract OCR 引擎的 Python 封装，用于从图像中提取文本。

1. 主要功能

文本识别：从图像中提取文本。

多语言支持：支持多种语言的文本识别。

图像预处理：结合其他库（如 OpenCV）进行图像预处理以提高识别精度。

2. 基本用法

```
python
import pytesseract
from PIL import Image
# 打开图像
```




```
image = Image.open('example.png')
# 提取文本
text = pytesseract.image_to_string(image)
# 输出识别结果
print(text)
```

3. 常用参数

- (1) lang: 指定语言 (如 eng、chi_sim)。
- (2) config: 自定义配置 (如 --psm 6 用于块文本识别)。

二、opencv 库

opencv 是一个开源的计算机视觉库，广泛用于图像和视频处理。

1. 主要功能

- (1) 图像读取与显示：读取、显示和保存图像。
- (2) 图像处理：包括灰度化、二值化、滤波、边缘检测等。
- (3) 特征检测：如角点检测、轮廓检测等。
- (4) 视频处理：读取、处理和保存视频。

2. 基本用法

```
python
import cv2
# 读取图像
image = cv2.imread('example.jpg')
# 转换为灰度图像
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# 显示图像
cv2.imshow('Gray Image', gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
# 保存图像
cv2.imwrite('gray_example.jpg', gray_image)
```

3. 常用函数

- (1) cv2.imread(): 读取图像。
- (2) cv2.imshow(): 显示图像。
- (3) cv2.imwrite(): 保存图像。



(4) cv2.cvtColor(): 颜色空间转换。

(5) cv2.GaussianBlur(): 高斯模糊。

(6) cv2.Canny(): 边缘检测。

4. 结合使用示例

结合 pytesseract 和 opencv 进行图像预处理和文本识别。

```
python
import cv2
import pytesseract
from PIL import Image
# 读取图像
image = cv2.imread('example.jpg')
# 转换为灰度图像
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# 二值化处理
_, binary_image = cv2.threshold(gray_image, 128, 255, cv2.
THRESH_BINARY)
# 保存预处理后的图像
cv2.imwrite('binary_example.jpg', binary_image)
# 使用 pytesseract 提取文本
text = pytesseract.image_to_string(Image.open('binary_example.
jpg'))
# 输出识别结果
print(text)
```



案例实现

使用 pytesseract 库进行文字识别，使用 opencv 库进行人脸识别。

```
python
import cv2
import pytesseract
# 读取图片
image = cv2.imread("example.jpg")
# 文字识别
text = pytesseract.image_to_string(image)
```



```
print(" 识别到的文字 :", text)
# 人脸识别

face_cascade=cv2.CascadeClassifier(cv2.data.haarcascades+
"haarcascade_frontalface_default.xml")

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5)

# 在图片上标注人脸

for (x, y, w, h) in faces:

    cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)

# 显示图片

cv2.imshow("Detected Faces", image)

cv2.waitKey(0)

cv2.destroyAllWindows( )
```



模块自测

一、单选题

1. pytesseract 库的主要功能是 ()。
A. 图像处理
B. 计算机视觉
C. 光学字符识别
D. 机器学习
2. 在 opencv 中, 读取图像的函数是 ()。
A. cv2.imread()
B. cv2.readimage()
C. cv2.loadimage()
D. cv2.imageRead()
3. 使用 pytesseract 进行 OCR 识别时, 需要安装以下哪个依赖项? ()
A. Tesseract OCR 引擎
B. OpenCV 库
C. NumPy 库
D. Matplotlib 库
4. 在 opencv 中, 图像的默认通道顺序是 ()。
A. RGB
B. BGR
C. HSV
D. GRAY
5. pytesseract.image_to_string() 函数的返回值类型是 ()。
A. 图像对象
B. 字符串
C. 列表
D. 字典

二、简答题

pytesseract 库的核心功能是什么? 请列举其主要应用场景。



项目小结

通过本项目的学习，我们不仅掌握了 Python 的基础知识，通过多个实战案例将理论与实践相结合，还探讨了职业素养、诚信道德等思政内容，在技术学习的同时，树立正确的价值观和职业观。



道德讲堂

某金融机构的一名程序员在编写交易算法时，故意加入了一些不公平的规则，导致客户利益受损。最终，这名程序员不仅失去了工作，还面临法律制裁。这个案例告诉我们，职业道德是职业生涯的底线，任何时候都不能触碰。

在金融领域，诚信是至关重要的职业素养。无论是数据分析还是算法交易，都需要严格遵守职业道德，确保数据的真实性和算法的公平性。作为一名未来的金融从业者，小明需要时刻牢记诚信原则，杜绝任何形式的造假和欺诈行为。



项目自测

一、单选题

1. Python 是一种什么类型的编程语言？（ ）
A. 编译型 B. 解释型 C. 混合型 D. 以上都不对
2. 以下哪个不是 Python 的合法变量名？（ ）
A. _name B. 2name C. name2 D. name_2
3. 以下哪种语句用于条件判断？（ ）
A. for 语句 B. while 语句 C. if 语句 D. try 语句
4. 函数定义的关键字是（ ）。
A. class B. def C. return D. import
5. 要导入一个名为 math 的模块，使用的语句是（ ）。
A. import math B. from math import *
C. include math D. require math

二、填空题

1. Python 安装时，若选中“Add Python to PATH”复选框，则可以在_____中直接运行 Python。
2. 在 Python 中，用_____表示空值。
3. 循环语句 for i in range(5) 中，循环会执行_____次。
4. 函数中 return 语句的作用是_____。
5. 若有模块 mymodule，其中有函数 myfunc 要调用该函数，可使用_____语句。



项目二 爬虫基础 + 金融商业案例实战



知识目标

- 了解爬虫技术；
- 了解金融商业数据特征；
- 掌握金融数据挖掘的方法。



能力目标

- 能够熟练使用爬虫工具库；
- 能够挖掘金融数据；
- 能够熟练运用反爬技术。



素养目标

- 具备合规与伦理意识，避免爬取个人隐私数据；
- 培养数据科学思维，精准挖掘金融商业数据；

金融决策离不开即时、准确的数据支撑。Python 提供了丰富强大的网络爬虫库（如 Requests、Scrapy、BeautifulSoup、Selenium），使得自动化抓取网页数据、应对复杂反爬挑战变得可行且高效。本项目将系统性地阐述利用 Python 进行金融数据爬取的完整流程，涵盖目标网站分析、发送请求、解析与提取数据、应对反爬策略、合规存储数据，并最终将获取的原始金融信息（如股票行情、新闻舆情、公司公告）应用于具体的商业分析案例（如市场监控、风险预警、情感分析），实现从数据源头到商业价值的转化。



模块一 爬虫基础知识



情景设置

小明是一名金融专业的学生，最近在学习网络爬虫知识，网络爬虫框架应该怎么构建呢？如何爬取并存储数据呢？又该如何反爬呢？



必备知识

理解超文本传输协议（HTTP）与网页结构，掌握数据抓取、解析、存储技术，同时需应对反爬机制并遵守法律法规与网站规则。

一、什么是爬虫

爬虫（Spider），也被称为网络爬虫或网络蜘蛛，是一种自动化程序，通过模拟人类浏览网页的行为，从互联网上批量抓取并提取数据。

爬虫可以自动遍历互联网上的各个网页，并根据预设的规则和算法来解析和收集感兴趣的信息。这些信息可以包括网页的文本内容、图片、链接、视频等。爬虫可以用于很多领域，例如搜索引擎的网页索引、数据挖掘、自动化测试、价格比较、舆情监测等。

二、爬虫的工作原理

1. 流程示意图

目标网站→发送 HTTP 请求→获取响应（HTML/JSON 等）→解析数据→存储数据→循环下一页。

2. 关键步骤

- （1）发送请求：通过 HTTP（GET/POST）向目标网站发送请求，获取网页内容。
- （2）解析数据：从 HTML/JSON/XML 中提取目标数据（如股票价格、新闻标题）。
- （3）存储数据：将数据保存到本地文件（CSV、Excel）或数据库（MySQL、MongoDB）。
- （4）处理分页 / 动态内容：自动翻页或处理 JavaScript 动态加载的数据（如 Ajax 请求）。

三、爬虫核心概念

1. HTTP 基础

- （1）请求方法。

GET：获取数据；POST：提交数据。

- (2) 关键字段。
User-Agent：伪装浏览器；Cookie：维持登录状态；Referer：来源页面。
- (3) 状态码。
200：请求成功；403：禁止访问；404：页面不存在。

2. HTML 解析

- (1) HTML 结构：由标签（如 <div>,<table>）和属性（如 class,id）组成。
- (2) 解析工具。
BeautifulSoup：基于 Python 的 HTML/XML 解析库，适合简单页面；
XPath：通过路径表达式定位节点（需配合 lxml 库）；
正则表达式（Regex）：灵活但复杂，适合提取特定模式文本。

3. 动态页面处理

- (1) 问题：部分数据通过 JavaScript 动态加载（如股票实时行情）。
- (2) 解决方案：
Selenium：模拟浏览器操作（如点击、滚动）；
逆向工程：分析网页的 Ajax 请求接口，直接调用 API 获取数据（更高效）。
常用爬虫工具库
常用爬虫工具库及其用途和示例场景如表 2-1 所示。

表 2-1 常用爬虫工具库及其用途和示例场景

工具库	用 途	示例场景
requests	发送 HTTP 请求，获取网页内容	抓取静态页面（如基金净值表）
Selenium	操控浏览器处理动态加载内容	抓取需登录的股票交易数据
Pandas	数据清洗与分析	整理爬取的财务报表数据

Requests 库的主要函数如表 2-2 所示。

表 2-2 requests 库的主要函数

函 数	说 明
request()	构造一个请求
get()	获取 URL 位置的资源
head	获取 URL 位置的头部信息
post	向 URL 位置提交 POST 请求
put	向 URL 位置提交 PUT 请求
patch	局部更新 URL 位置的资源
delete	删除 URL 位置存储的资源



下面展示的案例使用了 requests 库以 GET 请求的方式爬取百度搜索关键词 python 的网页，代码如下。

```
>>> import requests                    # 导入 requests 库
>>> url="http://www.baidu.com/s"      # 请求的 URL
>>> param=("wd" : "python")           # 请求的参数
>>> headers= ("UserAgent"; "Mozilla/5.0 (WindowsNT10.0; WOW64)
AppleWebKit/535.1 (KHTML, like Gecko) Chrome/51.0.2704.103
Safari/535.1")                        # 请求的报头
>>> response=requests.get ( url, params= param , headers =headers)
# 发送 GET 请求 , 返回响应对象
>>> print(response. text)              # 输出响应的内容
<!DOCTYPE html>
<!--STATUS OK-->
```

五、网络爬虫框架

Python 网络爬虫框架主要包括 Scrapy、Crawley、Portia 等。Scrapy 是使用 Python 开发的一个开源的爬虫框架，方便使用，用途广泛，可用于、网络检测、自动化测试、爬虫开发、数据挖掘等。它提供了 URL 队列、异步多线程访问、定时访问、数据库集成等多项功能，可以构建产品级的爬虫系统。

在 Windows 操作系统中的 cmd 命令行中要先安装该库，具体命令如下：`>pip install Scrapy`。安装完成后，可以输入命令 `Scrapy`，查看命令的使用方法。

Scrapy 框架是一个高度模块化的异步爬虫框架，其核心组件协同工作以实现高效的网络爬取。以下是 Scrapy 的主要组件及其作用。

1. 引擎 (engine)

- (1) 作用：控制所有组件的数据流，触发事件驱动的处理流程。
- (2) 细节：负责调度、信号传递，协调请求 / 响应的流转，是框架的“中枢”。

2. 调度器 (scheduler)

- (1) 作用：管理待爬取的请求队列 (request queue)，决定下一步执行的请求。
- (2) 细节：支持去重 (通过 DUPEFILTER)、优先级排序，可自定义队列存储策略 (如内存、Redis)。

3. 下载器 (downloader)

- (1) 作用：执行实际的 HTTP 请求，获取网页内容并返回响应 (response)。
- (2) 细节：基于 Twisted 异步处理，高效处理并发请求，支持代理、重试等。



4. 爬虫 (Spider)

(1) 作用: 用户自定义的解析逻辑, 负责分析响应、提取数据或生成新请求。

(2) 关键方法: `start_requests()` 生成初始请求; `parse()` 解析响应, 返回 Item 或 Request 对象。

5. 项目管道 (item pipeline)

(1) 作用: 处理爬虫提取的结构化数据, 如清洗、验证、存储。

(2) 常见操作: 数据去重 (如数据库主键检查)、存储到数据库 (如 MongoDB、MySQL)、

文件导出 (JSON、CSV)。

6. 中间件 (middleware)

(1) 下载器中间件 (downloader middleware): 处理请求和响应 (如添加 Headers、代理、异常处理)。

示例: User-Agent 轮换、自动重试。

(2) 爬虫中间件 (spider middleware): 处理爬虫的输入 (响应) 和输出。

示例: 过滤无效响应、修改爬虫输出。

7. 扩展 (extensions)

(1) 作用: 增强框架功能, 如统计、邮件通知、API 集成。

(2) 示例: 统计爬取速度 (log stats)、监控爬虫状态并发送告警。



案例实现

一、使用爬虫框架的基本步骤

使用网络爬虫框架的基础步骤通常如下, 以 Scrapy 为例。

1. 环境准备

(1) 安装框架。

```
bash
pip install scrapy
```

(2) 创建项目。

```
bash
scrapy startproject project_name
```

生成项目目录结构 (包含 spiders、items.py、pipelines.py 等)。

2. 定义数据结构

在 items.py 中定义要爬取的字段 (类似数据库表结构)。



```
python
import scrapy

class ProductItem(scrapy.Item):
    name = scrapy.Field( )
    price = scrapy.Field( )
    url = scrapy.Field( )
```

3. 编写爬虫 (Spider)

在 `spiders` 目录下创建爬虫文件 (如 `product_spider.py`)。

```
python
import scrapy
from project_name.items import ProductItem

class ProductSpider(scrapy.Spider):
    name = "product_spider"    # 爬虫唯一标识
    start_urls = ["https://example.com/products"]    # 初始请求

    def parse(self, response):
        # 解析页面, 提取数据或生成新请求

        for product in response.css("div.product"):
            item = ProductItem( )
            item["name"] = product.css("h2::text").get( )
            item["price"] = product.css(".price::text").get( )
            item["url"] = product.css("a::attr(href)").get( )
            yield item

        # 生成下一页请求
        next_page = response.css("a.next-page::attr(href)").get( )
        if next_page:
            yield response.follow(next_page, callback=self.parse)
```

4. 数据处理 (Pipeline)

在 `pipelines.py` 中定义数据清洗和存储逻辑。

```
python
class ProductPipeline:

    def process_item(self, item, spider):
        # 清洗数据 (如去除价格中的货币符号)
        item["price"] = item["price"].replace("$", "").strip( )
```



```
# 存储到文件或数据库（示例保存为 JSON）
with open("products.json", "a") as f:
    f.write(f"{dict(item)}\n")
return item
```

5. 配置中间件与扩展

（1）启用 Pipeline，在 settings.py 中注册。

```
python
ITEM_PIPELINES = {'project_name.pipelines.ProductPipeline':
300, # 优先级数值（越小越优先）}
```

（2）设置请求头/延迟（反反爬）。

```
python
USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) ..."
DOWNLOAD_DELAY = 2 # 请求间隔秒数
```

6. 运行爬虫

```
bash
scrapy crawl product_spider -o products.json
# -o 参数可自动导出数据到文件（JSON/CSV 等）
```

7. 调试与优化

（1）Shell 调试。

```
bash
scrapy shell "https://example.com/product-page"
# 在交互式环境中测试选择器（response.css( )/xpath( )）
```

（2）调整并发数，在 settings.py 中修改。

```
python
CONCURRENT_REQUESTS = 16 # 并发请求数
```

8. 处理动态内容（可选）

若页面依赖 JavaScript 渲染，可结合 Selenium 或 Splash。

```
python
# 示例：在 Downloader Middleware 中使用 Selenium
from selenium import webdriver
```



```
def get_page(url):  
    driver = webdriver.Chrome()  
    driver.get(url)  
    html = driver.page_source  
    driver.quit()  
    return html
```



拓展知识

关键注意事项如下。

1. 遵守 robots.txt 在 settings.py 中设置。

```
python
```

```
ROBOTSTXT_OBEY=True
```

2. 错误重试，启用自动重试中间件。

```
python
```

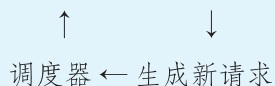
```
RETRY_ENABLED=True
```

```
RETRY_TIMES=3 # 最大重试次数
```

- (1) 反爬策略：使用代理 IP、随机 User-Agent（需中间件支持）。

- (2) 示例流程如下。

启动爬虫→引擎调度→下载页面→Spider 解析→提取数据→管道处理→存储结果



通过以上步骤，可以快速构建一个结构清晰、可扩展的网络爬虫。实际应用中需根据目标网站的特点灵活调整（如动态加载、登录验证等）。

二、数据存储方式

1. 文件存储

- (1) CSV/Excel：适合小规模数据（如单只股票历史价格）。
- (2) JSON：适合嵌套结构数据（如 API 返回的复杂对象）。

2. 数据库存储

- (1) MySQL：关系型数据库，适合结构化数据（如财报指标）。
- (2) MongoDB：非关系型数据库，适合灵活存储（如新闻评论）。