

智能学生选课推荐系统——项目启动与Python开发环境搭建

教学设计

| | |
|--------|--|
| 课题 | 智能学生选课推荐系统——项目启动与Python开发环境搭建 |
| 课时 | 1课时(40分钟) |
| 教学目标 | <p>知识目标: 了解选课管理的实际需求与痛点,理解Python在数据处理与业务逻辑实现中的优势,认知软件项目的基本开发流程。</p> <p>技能目标: 能够描述智能选课推荐系统的核心功能与技术路径,掌握Python开发环境(PyCharm/VSCode)的安装与配置,能创建项目并运行第一个选课系统脚本。</p> <p>素养目标: 建立"用技术解决管理问题"的工程意识,培养在真实项目情境中分析需求、规划方案的职业素养。</p> |
| 教学重难点 | <p>重点: 选课推荐系统的功能需求分析;Python在项目中的角色定位;Python开发环境搭建与脚本运行。</p> <p>难点: 将抽象的选课业务逻辑与Python编程任务关联;理解项目化开发的阶段性与整体性。</p> |
| 教学资源准备 | 多媒体课件(含选课场景视频);项目功能演示动画;Python及PyCharm/VSCode安装包;项目需求说明书文档;示例选课数据表格。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|--|--|--|
| 1. 项目情境导入 6分钟 | 展示传统选课管理的困境(手工统计、容量超限、规则复杂),引出"智能学生选课推荐系统"项目需求,明确项目目标(自动验证、智能推荐、数据统计)。 | 情境创设 播放教务老师处理选课冲突的场景视频,提问:"如果用Python程序来处理,能解决哪些问题?" 成品演示 展示完整系统运行效果:输入学生信息→自动筛选可选课程→生成推荐列表→统计选课人数。 | 观察思考 观看场景,联系自身选课经历,思考技术改进方案; 直观感知 观察成品演示,初步建立"需求输入-Python处理-结果输出"的认知框架。 | 通过真实业务痛点创设项目情境,让学生明确项目的实用价值;通过成品展示建立学习目标的具象化认知,激发用编程解决管理问题的兴趣。 |
| 2. 项目架构解析 10分钟 | 讲解选课推荐系统的功能模块,分析项目的技术实现路径,明确Python核心语法与数据结构的应用场景。 | 功能拆解 结合系统演示讲解四大模块:学生信息管理、课程数据存储、推荐规则引擎、结果统计输出; 技术映射 说明各模块对应的Python知识:变量与数据类型、列表字典、条件循环、函数封装,绘制技术路线图。 | 聆听记录 理解项目模块划分,在笔记中绘制功能结构图; 讨论交流 小组讨论:"要实现自动推荐功能,程序需要处理哪些数据?需要什么判断逻辑?"并分享观点。 | 将抽象的编程学习转化为具体的项目任务链,帮助学生建立系统化思维;通过讨论强化对Python任务的理解,为后续模块化学习明确方向。 |
| 3. 开发工具认知 8分钟 | 介绍Python集成开发环境的功能特点,讲解IDE在项目开发中的便利性(代码提示、调试、项目管理、版本控制)。 | 工具介绍 展示PyCharm界面,讲解项目创建、代码编辑、运行调试等核心功能; 演示操作 演示创建"CourseRecommendedSystem"项目的完整流程:新建项目→创建main.py文件→编写print("选课系统开发启动!")→运行输出。 | 观看学习 认识IDE的界面布局与主要功能区(项目视图、编辑器、运行窗口、终端); 模仿操作 跟随演示,在电脑上创建同名项目并运行第一个脚本,观察输出结果。 | 通过直观演示降低专业工具的使用门槛,消除对IDE的陌生感;通过成功运行第一个脚本建立成就感,为后续编码实践打下操作基础。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|---|--|---|
| 4. 环境搭建实践 12分钟 | 指导学生完成Python及IDE的确认或安装,创建项目文件夹,编写并运行项目启动脚本,验证环境配置正确。 | 任务发布 发布实践任务:"搭建开发环境,创建项目,输出系统启动信息及项目功能列表"; 巡回指导 巡视学生操作,解答环境配置问题,强调项目命名规范、文件组织结构等良好习惯。 | 动手实践 确认Python及IDE安装状态,创建"CourseRecommendSystem"项目; 编码测试 编写多行print语句输出:系统名称、版本号、四大功能模块名称,运行验证环境。 | 通过实际操作巩固开发环境使用技能,完成项目的"环境准备"里程碑;通过输出项目结构信息强化对系统功能的认知,建立项目归属感。 |
| 5. 项目展望与总结 4分钟 | 总结本课时内容,预告后续课程将逐步实现各功能模块,布置课后任务。 | 知识梳理 回顾项目架构与Python角色,强调本课时是项目的"认知与准备起点"; 任务布置 课后任务:思考并列出3条你认为选课系统应该具备的智能推荐规则(如避免时间冲突、优先推荐高分课程等)。 | 回顾反思 总结收获,明确后续学习将逐步实现数据处理与业务逻辑; 接收任务 记录课后任务,思考如何将业务需求转化为编程逻辑。 | 通过总结强化知识结构,通过展望明确项目的阶段性;课后任务引导学生主动进行需求思考,培养产品经理思维与用户视角。 |

教学成效与反思

| | |
|------|--|
| 教学成效 | 结合项目启动阶段目标评估:85%以上学生能准确说出选课推荐系统的四大功能模块及对应Python技术,全员完成开发环境搭建并成功运行项目启动脚本。通过真实选课场景与系统演示结合,学生对项目实用性认同度高,课堂讨论活跃。项目的"认知与准备"目标达成良好,学生已建立起"模块化开发"的初步意识,为后续功能实现奠定了认知基础。 |
| 教学反思 | 本课时成功地将Python编程概述具象为选课推荐系统的启动课,项目情境贴近学生实际,代入感强。技术架构讲解时,将抽象的编程概念与具体的业务模块绑定,学生理解效果好。不足在于:环境搭建环节因学生电脑性能差异,部分机器IDE启动较慢,占用时间略超预期;对"为什么选Python而非Excel"的对比分析不够深入,部分学生对Python优势感知不强。改进方向:①提供详细的课前环境检查清单和备用轻量级编辑器方案;②增加Python与Excel处理复杂选课规则的对比演示,直观展现编程语言的灵活性与扩展性。整体而言,项目驱动框架让学习目标具象,学生的学习动力与目标感显著提升。 |

智能学生选课推荐系统——学生信息采集模块开发

教学设计

| | |
|--------|--|
| 课题 | 智能学生选课推荐系统——学生信息采集模块开发 |
| 课时 | 1课时(40分钟) |
| 教学目标 | <p>知识目标: 掌握Python基本数据类型(字符串、整数、浮点数、布尔值)的特点与应用场景,理解变量命名规范与赋值机制,了解input()与print()函数的使用方法。</p> <p>技能目标: 能根据选课系统的数据需求设计学生信息结构,能使用合适的数据类型定义变量存储学生姓名、学号、年级、学分等信息,能编写交互式信息采集脚本并进行格式化输出。</p> <p>素养目标: 培养数据建模思维,养成规范命名与代码注释的良好习惯,建立"数据是程序基础"的认知。</p> |
| 教学重难点 | <p>重点: Python基本数据类型的选择与使用;变量的定义与赋值;input()函数的数据类型转换;格式化输出方法。</p> <p>难点: 根据业务需求选择合适的数据类型;input()获取的字符串类型转换为数值类型;理解变量在内存中的存储机制。</p> |
| 教学资源准备 | 多媒体课件(含数据类型对比表);学生信息字段说明文档;代码演示环境;数据类型错误示例与调试视频;项目代码模板文件。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|--|---|---|
| 1. 模块任务导入 5分钟 | 回顾项目架构,聚焦"学生信息管理"模块,明确本课时任务:设计并实现学生信息的数据结构与采集功能。 | 任务聚焦 展示选课系统流程图,强调学生信息是推荐引擎的输入基础,提问:"系统需要记录学生的哪些信息?" 需求分析 引导学生讨论并列出的关键字段:姓名、学号、年级、已修学分、是否重修生等,说明每类数据的作用。 | 回顾联系 回忆上节课项目架构,理解本模块在系统中的定位; 需求思考 结合选课场景,思考并提出需要采集的学生信息字段,参与讨论。 | 通过任务聚焦建立模块与整体的关联,强化项目连贯性;通过需求分析培养数据建模意识,为后续技术学习提供明确的应用场景。 |
| 2. 数据类型探究 10分钟 | 讲解Python基本数据类型的特点,演示不同类型数据的定义与使用,分析各类型在学生信息存储中的适用性。 | 概念讲解 结合学生信息字段,讲解字符串(姓名)、整数(学号、年级)、浮点数(学分)、布尔值(是否重修)的定义方式与适用场景; 代码演示 演示变量定义:name="张三"、student_id=20240101、credits=15.5、is_retake=False,并用type()函数查看类型。 | 观察记录 理解各数据类型的特点,记录定义语法规则; 对比分析 思考:"为什么学号用整数而不是字符串?学分为什么要用浮点数?"与同学交流。 | 通过业务场景与数据类型的紧密绑定,让抽象概念具象化;通过type()函数演示培养类型意识,为类型转换打下认知基础。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|--------------------------|--|--|---|--|
| 3. 信息采集实现 12分钟 | 讲解input()函数的使用与类型转换,演示格式化输出方法,引导学生编写完整的信息采集脚本。 | <p>功能讲解</p> <p>讲解input()函数的工作机制,强调其返回字符串类型,演示int()、float()转换方法;</p> <p>代码演示</p> <p>演示完整采集流程:提示输入→类型转换→格式化输出,代码示例:</p> <pre>name = input("姓名:") grade = int(input("年级:")) credits = float(input("已修学分:")) print(f"学生{name}, 年级:{grade}, 学分:{credits}")``</pre> <p>**任务发布**</p> <p>发布编码任务:"编写student_info.py,采集5项学生信息并格式化输出"。</p> | <p>聆听理解</p> <p>理解input()的字符串特性与转换必要性;</p> <p>模仿实践</p> <p>跟随演示代码,在IDE中逐行输入并测试;</p> <p>独立编码</p> <p>根据需求,完善采集脚本:添加学号、是否重修字段,设计友好的输入提示与输出格式。</p> | 通过师生同步演示降低编码难度;通过类型转换演示突破难点;通过独立编码任务培养实际开发能力,强化知识应用。 |
| 4. 代码调试与优化 9分钟 | 指导学生测试程序,处理常见错误,优化代码的规范性与可读性。 | <p>巡回指导</p> <p>巡视学生编码,针对类型转换错误、变量命名不规范等问题进行个别指导;</p> <p>错误示范</p> <p>展示典型错误(如直接对字符串做数学运算),演示错误信息阅读与调试方法;</p> <p>规范强调</p> <p>强调变量命名使用下划线命名法(student_id)、添加注释说明字段含义等规范。</p> | <p>自测调试</p> <p>运行程序,输入测试数据,观察输出结果;</p> <p>问题解决</p> <p>根据错误提示或教师指导,修正类型转换、语法错误;</p> <p>代码优化</p> <p>按规范修改变量名,为关键代码添加注释。</p> | 通过实际调试培养问题诊断能力;通过错误示范提升错误预判意识;通过规范强调养成职业化编码习惯,为团队协作打下基础。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|-------------------------------------|--|---|--|
| 5. 成果展示与总结 4分钟 | 组织学生展示作品,总结数据类型与信息采集的核心要点,预告下一模块任务。 | 作品展示 邀请2-3名学生演示程序运行,点评代码的规范性与输出格式的友好性; 知识总结 总结:合适的数据类型是数据处理的前提,类型转换是交互程序的关键; 任务预告 下节课将使用条件判断实现选课资格验证,需基于本节采集的学生信息。 | 观摩学习 观看同学作品,对比自己的实现方案; 反思内化 思考本节收获,记录易错点; 展望准备 了解下节任务,思考如何根据年级、学分等条件判断选课资格。 | 通过作品展示增强成就感与学习动力;通过总结强化核心知识点;通过预告保持项目学习的连贯性与期待感。 |

教学成效与反思

| | |
|------|---|
| 教学成效 | 结合模块开发目标评估:90%以上学生能准确选择合适的数据类型存储学生信息,80%以上学生独立完成信息采集脚本开发并实现格式化输出。通过业务需求驱动的数据类型学习,学生对"为什么需要不同类型"理解深刻,类型转换的必要性认知明确。代码调试环节参与度高,学生初步建立了错误诊断意识。模块功能基本实现,为后续条件判断模块提供了有效的数据基础,项目推进顺利。 |
| 教学反思 | 本课时通过"学生信息采集"这一具体模块任务,成功地将抽象的数据类型知识转化为实际的编码实践,学习效果显著。将业务字段与数据类型一一对应的讲解方式,让学生快速理解了类型选择的依据,避免了纯理论讲解的枯燥。不足之处:①对于"布尔值在条件判断中的应用"讲解较浅,部分学生对is_retake字段的作用理解不深;②部分学生在独立编码时,对f-string格式化语法不够熟练,出现语法错误较多。改进方向:①在讲解布尔类型时,增加简单的if判断示例,提前渗透下节课内容;②提供格式化输出的标准模板,减少语法细节对思维的干扰,让学生更专注于业务逻辑。整体上,项目驱动的模块化教学让知识点的学习更有目的性,学生的代码实践能力进步明显。 |

智能学生选课推荐系统——课程筛选逻辑实现

教学设计

| | |
|--------|---|
| 课题 | 智能学生选课推荐系统——课程筛选逻辑实现 |
| 课时 | 1课时(40分钟) |
| 教学目标 | <p>知识目标: 掌握Python条件判断语句(if-elif-else)的语法结构与执行逻辑,理解比较运算符、逻辑运算符的使用规则,了解循环结构(for、while)的基本概念与应用场景。</p> <p>技能目标: 能根据选课规则设计多条件判断逻辑,能使用if语句实现选课资格验证功能,能使用for循环批量处理多个学生的选课申请,能编写完整的课程筛选脚本并输出验证结果。</p> <p>素养目标: 培养逻辑思维与规则建模能力,养成代码缩进规范意识,建立"程序决策自动化"的工程认知。</p> |
| 教学重难点 | <p>重点: if-elif-else条件判断的语法与逻辑;比较运算符与逻辑运算符的组合使用;for循环的基本结构与遍历机制。</p> <p>难点: 多条件组合判断的逻辑设计(如"年级≥ 2且学分≥ 30");条件嵌套的层次理解;循环体内的条件判断综合应用。</p> |
| 教学资源准备 | 多媒体课件(含选课规则说明);决策流程图模板;代码演示环境;多条件判断逻辑示例;学生数据测试集(3-5组)。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|---|---|---|
| 1. 规则场景导入 6分钟 | 展示选课管理中的典型规则(年级限制、学分要求、重修优先),引出本课时任务:使用条件判断实现选课资格自动验证。 | 规则展示 展示选课规则表:高级课程要求年级 ≥ 2 且学分 ≥ 30 ,重修生优先选课,提问:"程序如何自动判断学生是否符合条件?" 人工对比 演示人工逐条检查规则的繁琐过程,强调自动化判断的价值。 | 规则理解 阅读选课规则说明,理解各类限制条件; 思维转换 思考:如何将文字描述的规则转化为程序的判断逻辑?尝试用自然语言描述判断步骤。 | 通过真实规则场景激活学生的逻辑思维;通过人工对比突出自动化判断的效率优势,建立学习动机。 |
| 2. 条件判断语法 11分钟 | 讲解if-elif-else语句的语法结构、执行流程与缩进规范,演示比较运算符与逻辑运算符的使用。 | 语法讲解 结合流程图讲解if语句的执行逻辑,强调条件表达式、冒号、缩进三要素,演示单分支、双分支、多分支结构; 运算符演示 讲解比较运算符(==、!=、>、>=)与逻辑运算符(and、or、not),演示代码: <pre>grade = 2 credits = 35 if grade >= 2 and credits >= 30: print("符合高级课程选课条件") else: print("不符合条件")``</pre> **规则映射** 引导学生分析:上述代码如何对应"年级 ≥ 2 且学分 ≥ 30 "的规则。 | 聆听理解 理解if语句的判断机制,记录语法结构与缩进规范; 跟随实践 在IDE中输入演示代码,修改变量值测试不同分支的执行; 规则转化 尝试将"重修生优先"规则转化为if-else结构,与同学讨论实现方案。 | 通过流程图与代码同步讲解降低理解难度;通过实时测试让学生直观感受条件判断的动态执行;通过规则转化任务培养逻辑建模能力。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|------------------------------------|---|--|---|
| 3. 资格验证实现 12分钟 | 引导学生基于上节课采集的学生信息,编写多条件判断的选课资格验证脚本。 | <p>任务分解 将选课资格验证分解为三个子任务:①检查年级与学分,②判断是否重修生,③综合输出验证结果;</p> <p>代码框架 提供代码框架,演示主体逻辑:</p> <pre># 获取学生信息(上节课内容) name = input("姓名:") grade = int(input("年级:")) credits = float(input("学分:")) is_retake = input("是否重修?(y/n):") == "y" # 资格判断 if is_retake: print(f"{name}是重修生,优先选课") elif grade >= 2 and credits >= 30: print(f"{name}符合高级课程选课条件") else: print(f"{name}暂不符合条件")``` **任务发布** 要求学生完善代码,增加更多规则判断(如一年级只能选基础课)。</pre> | <p>理解框架 阅读代码框架,理解if-elif-else的层次逻辑与重修优先的实现方式;</p> <p>独立编码 在框架基础上,添加新的条件分支,实现更复杂的规则判断;</p> <p>自测验证 输入不同学生数据测试各分支逻辑,确保判断结果符合预期。</p> | 通过任务分解降低复杂度;通过代码框架提供脚手架支持;通过扩展任务培养举一反三能力,强化条件判断的灵活应用。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------|-------------------------------------|---|---|--|
| 4. 批量处理引入 8分钟 | 讲解for循环的基本语法,演示如何使用循环批量验证多个学生的选课资格。 | <p>循环概念 提问:"如果有10个学生申请选课,要运行10次程序吗?"引出循环的概念;</p> <p>语法演示 演示for循环遍历列表的基本语法,代码示例:</p> <pre>students = [{"name": "张三", "grade": 2, "credits": 35}, {"name": "李四", "grade": 1, "credits": 20}] for stu in students: if stu["grade"] >= 2 and stu["credits"] >= 30: print(f"{stu['name']}符合条件") else: print(f"{stu['name']}不符合条件")` ``</pre> <p>**任务体验** 指导学生修改上述代码,添加更多学生数据进行批量验证。</p> | <p>概念理解 理解循环的批量处理优势,认知for语句的遍历机制;</p> <p>代码实践 输入演示代码并运行,观察循环对每个学生的处理过程;</p> <p>数据扩展 在列表中添加3-5组学生数据,测试批量验证功能。</p> | 通过问题引入自然过渡到循环主题;通过简化的字典列表结构降低认知负担;通过批量处理体验让学生感受循环的实用价值,为下节课深入学习列表铺垫。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|-------------------------------------|--|---|---|
| 5. 成果展示与总结 3分钟 | 组织学生展示验证脚本,总结条件判断与循环的核心要点,预告下一模块任务。 | 作品点评 邀请学生演示批量验证程序,点评逻辑严密性与代码规范性; 知识总结 总结:条件判断实现规则自动化,循环实现批量自动化,两者结合是智能系统的基础; 任务预告 下节课将使用列表存储课程信息,实现推荐课程的筛选与排序。 | 观摩学习 观看同学作品,对比判断逻辑的差异; 反思内化 总结条件判断的易错点(如缩进、逻辑运算符); 展望准备 思考:如何用列表存储多门课程的信息?如何从中筛选出符合条件的课程? | 通过作品展示强化学习成就感;通过总结提炼核心知识点;通过预告保持项目学习的连贯性,为列表学习做好心理准备。 |

教学成效与反思

| | |
|------|--|
| 教学成效 | 结合模块开发目标评估:85%以上学生能熟练使用if-elif-else实现多条件判断,75%以上学生能编写包含逻辑运算符的复合条件,80%学生完成批量验证脚本的基本功能。通过真实选课规则驱动,学生对条件判断的应用场景理解深刻,逻辑建模能力有显著提升。循环的初步引入让学生认识到批量处理的价值,为下节课深入学习列表做好了铺垫。模块的核心功能(资格验证)已基本实现,项目推进符合预期。 |
| 教学反思 | 本课时通过"选课资格验证"这一具体业务逻辑,成功地将条件判断与循环结构融入项目实践,学习效果良好。将文字规则转化为判断逻辑的过程,有效培养了学生的计算思维。不足之处:①对于"and与or的优先级"讲解不足,部分学生在复杂条件中出现逻辑错误;②循环部分由于时间限制讲解较浅,字典的访问语法对部分学生造成困扰,影响了对循环本身的理解。改进方向:①增加逻辑运算符优先级的对比示例,建议使用括号明确逻辑关系;②将循环演示简化为普通列表(而非字典列表),降低认知负担,或将循环内容扩展为独立课时深入讲解。整体上,通过规则驱动的条件判断教学,学生的逻辑思维得到有效训练,代码实现能力进步明显。 |

智能学生选课推荐系统——课程数据管理与推荐列表生成

教学设计

| | |
|--------|--|
| 课题 | 智能学生选课推荐系统——课程数据管理与推荐列表生成 |
| 课时 | 1课时(40分钟) |
| 教学目标 | <p>知识目标: 掌握Python列表的创建、访问、修改、遍历等基本操作,理解列表的有序性与可变性特点,了解列表常用方法(append、remove、sort等)的功能,认识元组的不可变特性。</p> <p>技能目标: 能使用列表存储多门课程的信息(课程名、学分、容量等),能结合条件判断从课程列表中筛选符合学生条件的推荐课程,能使用列表方法对推荐结果进行排序与格式化输出,能编写完整的课程推荐脚本。</p> <p>素养目标: 培养数据组织与批量管理意识,养成数据结构选择的合理性思维,建立"用合适的容器存储批量数据"的工程认知。</p> |
| 教学重难点 | <p>重点: 列表的创建与访问方法;列表的遍历与条件筛选;列表常用方法的使用;列表与条件判断、循环的综合应用。</p> <p>难点: 从课程列表中按条件筛选并生成新列表的逻辑设计;列表嵌套结构(列表中存储字典)的理解与操作;排序方法的参数使用。</p> |
| 教学资源准备 | 多媒体课件(含列表结构图);课程信息数据表;代码演示环境;列表操作速查表;筛选与排序逻辑流程图。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|--|---|--|
| 1. 数据存储需求 5分钟 | 回顾前三节课内容,引出批量存储课程信息的需求,明确本课时任务:使用列表管理课程数据并实现智能推荐。 | 需求提出 展示选课系统界面,提问:"如果系统有50门课程,用50个变量存储合适吗?"引出批量存储的必要性; 任务聚焦 说明本节课将实现推荐引擎的核心功能:从课程库中筛选出符合学生条件的课程并排序推荐。 | 问题思考 思考批量数据管理的方案,回顾上节课循环中使用的列表结构; 目标明确 理解本课时是推荐系统的核心模块,连接前面的学生信息与条件判断。 | 通过问题引入激活学生对数据结构的需求意识;通过任务聚焦强化模块在整体项目中的关键地位,建立学习动机。 |
| 2. 列表基础操作 10分钟 | 讲解列表的创建、访问、修改、遍历等基础操作,演示列表常用方法的使用。 | 概念讲解 讲解列表的定义方式、索引访问(正向、负向)、切片操作,强调列表的有序性与可变性; 方法演示 演示常用方法,代码示例: <pre>courses = ["Python编程", "数据库", "Web开发"] courses.append("数据分析") # 添加 courses.remove("数据库") # 删除 courses.sort() # 排序 for course in courses: print(course)````</pre> **实践任务** 要求学生创建包含5门课程的列表,练习增删改查操作。 | 聆听理解 理解列表的结构特点,记录基本操作语法; 跟随实践 在IDE中输入演示代码,测试各种操作的效果; 独立练习 创建课程列表,尝试添加新课程、删除指定课程、遍历输出全部课程。 | 通过系统讲解建立列表的完整认知;通过即时实践强化操作记忆;通过独立练习培养基础操作的熟练度。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|--------------------------|---|---|--|---|
| 3. 课程数据建模 11分钟 | 引导学生设计课程信息的数据结构,讲解如何使用字典表示单门课程,用列表存储多门课程。 | <p>结构设计</p> <p>引导讨论:"一门课程需要哪些信息?如何用—个变量表示?"引出字典结构;</p> <p>代码演示</p> <p>演示课程数据建模,代码示例:</p> <pre>courses = [{"name": "Python 编程", "credits": 3, "grade_req": 1, "capacity": 40}, {"name": "数 据分析", "credits": 4, "grade_req": 2, "capacity": 30}, {"name": "Web 开发", "credits": 3, "grade_req": 2, "capacity": 35}] for course in courses: print(f"课 程: {course['name']} , 学分: {course['credits'] '}"))``</pre> <p>**任务发布**</p> <p>要求学生补充课程列表,添加3-5门课程的完整信息。</p> | <p>讨论分析</p> <p>参与讨论,提出课程应包含的字段(课程名、学分、年级要求、容量等);</p> <p>理解结构</p> <p>理解"列表套字典"的嵌套结构,明确列表管理批量数据、字典管理单个对象的分工;</p> <p>数据填充</p> <p>按格式添加更多课程数据,确保字段完整准确。</p> | 通过讨论培养数据建模思维;通过嵌套结构的演示提升数据组织能力;通过数据填充任务巩固结构理解,为后续筛选做准备。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|---|--|--|
| 4. 推荐筛选实现 11分钟 | 引导学生编写推荐逻辑,结合学生信息与课程要求,从课程列表中筛选符合条件的课程并生成推荐列表。 | <p>逻辑分析 讲解筛选逻辑:遍历课程列表,判断学生年级是否满足课程要求,若满足则加入推荐列表;</p> <p>代码演示 演示核心筛选代码:</p> <pre>student_grade = 2 recommend_list = [] for course in courses: if student_grade >= course["grade_req"]: recommend_list.append(course) # 按学分排序推荐 recommend_list.sort(key=lambda x: x["credits"], reverse=True) for course in recommend_list: print(f"推荐: {course['name']}, 学分: {course['credits']}")``</pre> <p>**任务发布** 要求学生完善筛选条件(增加学分要求、容量限制等),优化输出格式。</p> | <p>理解逻辑 理解遍历、条件判断、列表添加的组合流程,明确筛选的实现机制;</p> <p>代码实践 输入演示代码并测试,观察推荐结果;</p> <p>功能扩展 添加更多筛选条件(如学分≥ 3、容量>0),调整排序规则(如按容量从大到小),美化输出格式。</p> | 通过逻辑分析将前几节知识综合应用;通过代码演示突破筛选难点;通过扩展任务培养灵活应用能力,完成推荐引擎核心功能。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|--------------------------|--------------------------------------|--|--|--|
| 5. 系统集成与总结 3分钟 | 组织学生展示推荐脚本,总结列表在项目中的核心作用,预告后续项目扩展方向。 | 作品点评 邀请学生演示完整推荐流程,点评筛选逻辑的合理性与输出的用户友好性; 模块总结 总结:列表是批量数据管理的核心工具,结合条件判断与循环可实现复杂的筛选与推荐逻辑; 项目展望 说明前四节课已完成推荐系统的核心功能,后续可扩展:集合去重、字典统计、函数封装、数据持久化等。 | 观摩学习 观看同学作品,学习优秀的筛选逻辑与输出设计; 反思总结 回顾四节课的项目进展,梳理知识点间的逻辑关系; 展望规划 思考:如何让系统更智能?还可以增加哪些功能? | 通过作品展示检验学习成效;通过模块总结强化列表的核心地位;通过项目展望保持学习热情,为后续深入学习铺垫。 |

教学成效与反思

| | |
|-------------|--|
| 教学成效 | 结合模块开发目标评估:85%以上学生能熟练使用列表存储与管理批量课程数据,80%学生能编写包含条件筛选的推荐逻辑,75%学生能使用sort方法对推荐结果进行排序。通过真实推荐场景驱动,学生对列表作为数据容器的核心作用理解深刻,数据组织能力显著提升。课程推荐引擎的基本功能已实现,前四节课构建的项目框架完整,学生对"从需求到实现"的软件开发流程有了初步体验,项目式学习的价值得到充分体现。 |
| 教学反思 | 本课时通过"课程推荐引擎"这一核心模块,成功地将列表、字典、条件判断、循环等知识点综合应用,形成了完整的功能闭环,学习效果显著。"列表套字典"的数据建模方式,让学生理解了复杂数据的组织方法,为后续面向对象学习做好铺垫。不足之处:①lambda表达式在排序中的使用对部分学生理解困难,建议改用更直观的方法或提前渗透函数概念;②筛选逻辑的扩展任务(如多条件组合)难度较大,部分学生未能在课堂完成,建议简化或提供参考代码。改进方向:①将排序演示简化为默认排序,或使用列表推导式等更符合当前认知水平的方法;②将复杂筛选逻辑设计为选做任务,降低核心任务难度。整体上,项目驱动的四课时教学成功地将抽象的编程知识转化为可见、可用的软件功能,学生的学习动力与成就感显著增强,为后续深入学习奠定了坚实基础。 |

班级图书漂流管理系统——图书信息录入与存储模块

教学设计

| | |
|--------|--|
| 课题 | 班级图书漂流管理系统——图书信息录入与存储模块 |
| 课时 | 1课时(40分钟) |
| 教学目标 | <p>知识目标: 理解列表与元组的概念、特点及应用场景差异，掌握列表的创建、索引、切片、添加、删除等基本操作，理解元组的不可变性及其在数据保护中的作用。</p> <p>技能目标: 能够使用列表存储多本图书的信息记录，运用元组封装单本图书的不可变属性（如ISBN、出版年份），能编写代码实现图书信息的批量录入、遍历显示和基本管理功能。</p> <p>素养目标: 建立"数据结构选择服务于实际需求"的工程意识，培养结构化思维和数据管理的规范意识，理解信息系统中数据存储的重要性。</p> |
| 教学重难点 | <p>重点: 列表的创建与常用操作方法（append、insert、remove等）；元组的定义与不可变特性；列表嵌套结构的理解与应用。</p> <p>难点: 在项目中合理选择列表或元组存储不同性质的数据；列表嵌套元组的混合数据结构设计；多本图书信息的批量录入与循环遍历逻辑。</p> |
| 教学资源准备 | 多媒体课件（含项目演示视频）；班级图书角现场照片或图书登记表样例；Python开发环境（PyCharm/VSCode）；示例代码模板文件。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|--|---|---|
| 1. 项目情境导入 6分钟 | 展示班级图书角的纸质登记表存在的问题（信息分散、统计困难、易丢失），引出"班级图书漂流管理系统"项目需求，明确本课时任务：建立图书信息的数字化存储方案。 | 问题引入 展示混乱的纸质登记表图片，提问："如何用Python程序来管理这些图书信息？需要什么样的数据容器？" 任务发布 明确本课时目标：学习列表和元组，设计并实现图书信息的录入与存储功能。 | 观察思考 结合自身图书借阅经历，思考数字化管理的必要性； 需求分析 讨论一本图书需要记录哪些信息（书名、作者、类型、ISBN等），哪些信息需要修改、哪些不能改。 | 通过真实场景建立项目情境，让学生明确"为什么需要数据结构"；通过需求分析引导学生思考数据的可变性与不可变性，为理解列表与元组的差异做铺垫。 |
| 2. 列表知识探究 10分钟 | 讲解列表的概念、特点（有序、可变、可重复），演示列表的创建、索引访问、添加元素（append/insert）、删除元素（remove/pop）等操作，强调列表适合存储需要动态增删的数据。 | 概念讲解 结合图书信息案例讲解列表定义： <pre>books = ["Python入门", "数据分析", "Web开发"]</pre> ，演示索引、切片操作； 操作演示 演示添加新书： <pre>books.append("机器学习")</pre> ，删除图书： <pre>books.remove("Web开发")</pre> ，强调列表的"可变性"特点。 | 跟随实践 在交互式环境中跟随教师创建图书列表，尝试索引访问、添加和删除操作； 对比理解 思考并回答："为什么图书列表要用列表而不是多个独立变量？" | 通过项目实例讲解抽象概念，增强理解；通过即时操作建立"列表=可增删的容器"认知；通过对比问题引导学生理解数据结构的优势。 |
| 3. 元组知识探究 8分钟 | 讲解元组的概念、特点（有序、不可变、可重复），演示元组的创建与访问，对比列表与元组的应用场景，强调元组适合存储不应被修改的数据（如图书的固定属性）。 | 概念对比 讲解元组定义： <pre>book_info = ("9787115123456", 2020, "技术类")</pre> ，演示访问但无法修改元组元素； 场景分析 提问："图书的ISBN、出版年份为什么要用元组？"引导理解"数据保护"意义。 | 动手尝试 创建元组存储一本书的固定信息，尝试修改元组元素（触发错误），体会不可变性； 讨论交流 小组讨论：图书管理中哪些数据适合用元组、哪些适合用列表？ | 通过对比教学突出元组的"不可变"核心特性；通过错误体验加深记忆；通过讨论培养"根据数据性质选择结构"的工程思维。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------------|---|--|---|--|
| 4. 混合结构设计 7分钟 | 引导学生设计"列表嵌套元组"的混合数据结构，用于存储多本图书的完整信息，演示如何将单本图书的可变信息（书名、借阅状态）与不可变信息（ISBN、出版年份）结合存储。 | 结构设计 演示设计思路：每本书用一个列表存储，其中嵌套元组保存固定属性： <pre>book1 = ["Python入门", "张三", ("9787115001", 2020), "在馆"]</pre> 代码示范 演示创建图书库： <pre>library = [book1, book2, ...]</pre> ，并遍历输出所有图书信息。 | 理解结构 分析混合结构的层次关系（列表→列表→元组），理解为什么这样设计； 模仿编码 根据教师示范，创建包含2-3本图书信息的library列表。 | 通过嵌套结构设计提升学生对复杂数据结构的理解能力；通过编码实践巩固列表与元组的综合应用，为项目功能实现奠定基础。 |
| 5. 项目功能实现 7分钟 | 指导学生编写图书录入与显示功能的完整代码：通过循环批量录入图书信息，并遍历列表输出所有图书的详细信息。 | 任务分解 分解编码任务：①循环输入图书基本信息；②组装为"列表+元组"结构；③添加到图书库；④遍历输出。 巡回指导 指导学生完成代码编写，强调输入验证、格式化输出等细节。 | 编码实践 编写录入循环（如录入3本书），将书名、作者等可变信息与ISBN等不可变信息分别处理； 功能测试 运行程序，录入测试数据，检查输出格式是否清晰。 | 通过完整功能实现将知识点串联成实战技能；通过任务分解降低编码难度；通过测试环节培养调试意识和代码规范习惯。 |
| 6. 总结与展望 2分钟 | 总结列表与元组的核心特性及在项目中的应用，预告下节课将学习字典和集合以实现更高效的图书查询与统计功能。 | 知识梳理 回顾列表（可变、适合动态管理）与元组（不可变、适合数据保护）的特点与选择依据； 任务布置 课后完善录入功能，增加图书类型的选项（如"文学类"、"技术类"）供用户选择。 | 回顾反思 总结本课时实现的功能：搭建了图书信息存储的数据基础； 接收任务 思考：如果有100本书，如何快速找到某本书？为下节课做准备。 | 通过总结强化核心知识，明确项目进度；通过展望和问题引导激发对后续学习内容的期待，保持项目学习的连贯性。 |

教学成效与反思

| | |
|------|--|
| 教学成效 | 本课时成功将列表与元组的学习置于"图书信息数字化存储"的项目任务中，约85%的学生能够准确说明列表与元组的核心差异并举例说明适用场景。全体学生完成了图书信息的录入与显示功能编码，其中70%以上的学生能独立设计"列表嵌套元组"的混合结构。课堂讨论环节学生参与度高，能够结合项目需求主动思考数据结构选择的合理性。通过真实的图书管理场景，学生对"为什么需要不同数据结构"有了直观认知，为后续复杂功能开发奠定了扎实基础。 |
| 教学反思 | 本课时通过"班级图书角"这一贴近学生生活的场景成功建立了项目情境，学生的代入感和学习动机明显增强。列表与元组的对比教学效果较好，特别是通过"尝试修改元组触发错误"的体验式学习，学生对不可变性的理解深刻。不足之处：①混合结构设计环节部分学生理解困难，出现"列表套列表"或"全用元组"等错误，后续应增加结构图示和更多示例；②录入功能实现时，少数学生对"循环+列表追加"的逻辑不够清晰，导致编码耗时较长，可考虑提供伪代码框架降低难度；③对数据验证（如ISBN格式检查）只做了口头强调，未落实到代码中，下次应增加简单的输入校验演示。改进方向：提前准备更直观的数据结构可视化图示；为不同水平学生提供分层任务（基础版/挑战版）；在巡回指导时重点关注逻辑思维较弱的学生。 |

班级图书漂流管理系统——图书查询与统计模块

教学设计

| | |
|--------|--|
| 课题 | 班级图书漂流管理系统——图书查询与统计模块 |
| 课时 | 1课时(40分钟) |
| 教学目标 | <p>知识目标: 理解字典的键值对结构及其快速查找优势，掌握字典的创建、访问、添加、修改、删除操作；理解集合的无序性、唯一性特点，掌握集合的创建与去重、交并差等操作。</p> <p>技能目标: 能够使用字典重构图书信息存储结构以实现按书名快速查询，运用集合统计图书类型分布和借阅人员信息（避免重复统计），能编写图书查询和统计分析功能的完整代码。</p> <p>素养目标: 建立"数据结构优化提升程序效率"的工程意识，培养数据去重、信息检索的算法思维，理解不同数据结构在实际问题中的取舍逻辑。</p> |
| 教学重难点 | <p>重点: 字典的键值对概念与访问方法（<code>dict[key]</code>、<code>get()</code>）；集合的唯一性与去重操作；字典与集合在项目中的应用场景。</p> <p>难点: 将列表结构转换为字典结构以优化查询效率；理解集合操作（交集、并集）在统计分析中的实际意义；综合运用字典、集合、列表解决复杂查询与统计任务。</p> |
| 教学资源准备 | 多媒体课件（含查询效率对比动画）；上节课完成的图书录入代码；字典与集合操作速查表；项目功能演示视频。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|--|---|--|
| 1. 项目问题引入 5分钟 | 回顾上节课实现的列表存储方案，演示在列表中查找图书需要遍历全部数据的低效问题，引出"如何快速查询"的需求，明确本课时任务：学习字典与集合优化查询和统计功能。 | 场景复现 演示在包含50本书的列表中查找《Python入门》，展示需要逐个对比的过程； 问题提出 提问："图书馆系统如何在几十万本书中秒速找到指定书籍？列表能做到吗？" | 观察对比 观察列表遍历查找的过程，感受效率问题； 问题思考 讨论：能否像查字典一样，通过"书名"直接定位到图书信息？ | 通过效率对比创造认知冲突，让学生意识到列表的局限性；通过"查字典"类比为学习字典数据结构建立生活化认知锚点。 |
| 2. 字典知识探究 10分钟 | 讲解字典的键值对结构、创建方式、访问与修改方法，演示字典的快速查找特性（O(1)时间复杂度），对比列表与字典在查询场景中的效率差异。 | 概念讲解 讲解字典定义： <code>book_dict = {"Python入门": {...}, "数据分析": {...}}</code> ，强调"书名=键、详细信息=值"的映射关系； 操作演示 演示通过书名直接访问： <code>book_dict["Python入门"]</code> ，对比列表遍历的耗时，讲解哈希表原理（简化版）。 | 跟随实践 创建简单字典（如学号与姓名的映射），尝试添加、访问、修改操作； 效率体验 对比列表和字典查找同一数据的代码行数和执行速度。 | 通过直观对比突出字典的核心优势——快速查找；通过动手操作建立"键→值"的映射思维；通过效率体验强化数据结构选择的重要性。 |
| 3. 集合知识探究 8分钟 | 讲解集合的无序性、唯一性特点，演示集合的创建、去重操作，介绍集合的交集、并集、差集在统计分析中的应用（如统计借阅人数、图书类型）。 | 概念讲解 讲解集合定义： <code>book_types = {"文学类", "技术类", "科普类"}</code> ，演示自动去重特性； 场景应用 演示统计案例：从多条借阅记录中提取不重复的借阅人： <code>borrowers = set(["张三", "李四", "张三"])</code> → <code>{"张三", "李四"}</code> 。 | 动手尝试 创建包含重复元素的列表，转换为集合观察去重效果； 讨论交流 小组讨论：统计"班级有多少种类型的图书"为什么要用集合而不是列表？ | 通过去重特性解决实际统计问题；通过应用场景讨论引导学生理解集合的"唯一性"价值，建立"用集合做统计"的思维模式。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------|---|--|---|--|
| 4. 项目结构重构 8分钟 | 指导学生将上节课的列表存储结构改造为字典结构，以书名为键存储图书详细信息，并在图书信息中使用集合存储借阅历史（避免重复记录）。 | 重构方案 演示将 <code>library</code> 列表转换为 <code>library_dict</code> 字典： <pre>library_dict = { "Python入门": {"作者": "张三", "ISBN": "001", "借阅人": set()}, ... }</pre> 代码示范 演示如何遍历旧列表，提取书名作为键，构建新字典。 | 理解结构 分析新旧结构的差异，理解为什么字典更适合查询场景； 协作编码 小组合作完成数据结构转换代码，将至少3本书的信息重构为字典格式。 | 通过结构重构深化对字典应用的理解；通过小组协作培养代码交流与协作能力；通过实际转换巩固字典与集合的综合运用。 |
| 5. 查询功能实现 7分钟 | 指导学生编写图书查询功能：用户输入书名，程序从字典中快速查找并输出图书详细信息；编写统计功能：统计所有图书的类型种类和借阅人总数（利用集合去重）。 | 任务分解 分解编码任务：①输入书名；②使用 <code>get()</code> 方法查询字典；③判断是否存在并输出结果；④遍历字典收集类型到集合。 巡回指导 指导学生处理"书名不存在"的异常情况，强调使用 <code>get()</code> 方法的容错性。 | 编码实践 编写查询函数，实现按书名查找功能，处理查询失败的提示； 功能测试 测试查询已有和不存在的书名，验证集合统计类型数量的准确性。 | 通过完整功能实现将字典与集合的知识转化为实战技能；通过异常处理培养健壮代码意识；通过测试环节强化调试与验证习惯。 |
| 6. 总结与展望 2分钟 | 总结字典与集合的核心特性及在项目中的应用，对比列表、字典、集合三种结构的适用场景，预告下节课将学习函数封装，实现更模块化的系统设计。 | 知识对比 绘制列表、字典、集合的特性对比表，明确查询用字典、去重统计用集合的选择逻辑； 任务布置 课后增加"按类型查询所有图书"功能，思考如何优化实现。 | 回顾反思 总结本课时实现的功能：将查询效率从O(n)优化到O(1)，实现了去重统计； 接收任务 思考：目前代码中查询、统计等操作都在主程序里，如何让代码更清晰？ | 通过对比总结形成数据结构选择的知识体系；通过展望引导学生思考代码组织问题，为下节课函数模块化学习做铺垫。 |

教学成效与反思

| | |
|------|--|
| 教学成效 | 本课时通过"查询效率优化"的实际需求成功引入字典与集合的学习,约80%的学生能够准确说明字典与列表在查询场景中的效率差异,并理解集合去重在统计中的应用。全体学生完成了数据结构从列表到字典的重构,其中75%以上能独立编写图书查询与类型统计功能。课堂讨论中学生能够主动对比不同数据结构的适用场景,体现出较强的工程思维。通过效率对比演示,学生对"算法复杂度"有了初步感知,为后续算法学习奠定基础。项目功能从"能存储"提升到"能高效查询",学生成就感显著提升。 |
| 教学反思 | 本课时通过列表遍历与字典查找的效率对比成功创造了认知冲突,激发了学生对"更优数据结构"的学习需求。字典的"查字典"类比、集合的"去重"特性都通过生活化场景得到了有效理解。不足之处:①数据结构重构环节难度较大,部分学生对"如何遍历列表提取键值对"的逻辑理解不够,导致代码出错率较高,后续应提供详细的重构步骤伪代码;②集合的交并差操作只做了简单介绍,未在项目中深度应用,学生对这些操作的实用价值感知不足,可考虑增加"两个班级借阅情况对比"的拓展案例;③对字典的 <code>get()</code> 方法与直接索引的差异讲解不够充分,少数学生仍使用 <code>dict[key]</code> 导致 <code>KeyError</code> 。改进方向:为重构任务提供可视化的数据流程图;设计更多集合操作的实战小任务;在代码规范中强调容错处理的重要性。 |

班级图书漂流管理系统——功能封装与菜单系统搭建

教学设计

| | |
|--------|---|
| 课题 | 班级图书漂流管理系统——功能封装与菜单系统搭建 |
| 课时 | 1课时(40分钟) |
| 教学目标 | <p>知识目标: 理解函数的概念、作用及模块化编程思想,掌握函数的定义、参数传递、返回值的使用方法,理解全局变量与局部变量的作用域差异。</p> <p>技能目标: 能够将图书录入、查询、统计等功能代码封装为独立函数,设计并实现带循环和选择结构的主菜单系统,通过函数调用整合各模块功能,使系统具备完整的人机交互能力。</p> <p>素养目标: 建立"模块化设计提升代码可维护性"的软件工程意识,培养代码复用、结构化编程的良好习惯,理解团队协作开发中函数封装的重要性。</p> |
| 教学重难点 | <p>重点: 函数的定义与调用语法;参数传递与返回值的使用;将具体功能代码抽象为函数的过程;主菜单的循环与分支控制逻辑。</p> <p>难点: 确定函数的合理粒度（一个函数完成一个明确任务）;处理函数间的数据传递（如共享图书字典）;设计用户友好的交互流程和异常处理。</p> |
| 教学资源准备 | 多媒体课件（含模块化设计原理图）;前两节课完成的项目代码;函数设计流程图;完整系统演示视频。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------|--|---|--|---|
| 1. 项目现状分析 5分钟 | 回顾前两节课实现的录入、查询、统计功能代码,展示代码冗长、逻辑混乱的问题（所有代码堆在主程序中）,引出模块化设计的必要性,明确本课时任务:用函数封装功能,搭建完整系统。 | 代码审查 展示一段包含重复代码的"混乱版"项目代码,提问："如果要修改查询逻辑,需要改几处？如何让代码更清晰？" 需求明确 说明专业开发中"一个功能一个函数"的模块化原则,明确本课时要实现菜单驱动的完整系统。 | 问题识别 阅读混乱代码,找出重复片段和逻辑不清的地方; 目标认同 理解模块化的价值:代码复用、易于维护、便于团队协作。 | 通过"坏代码"案例创造改进需求,让学生体会模块化的必要性;通过对比专业开发标准建立工程化编程意识。 |
| 2. 函数基础知识 9分钟 | 讲解函数的定义语法（def关键字）、参数传递（位置参数、默认参数）、返回值的使用,演示简单函数的创建与调用,强调函数的"黑盒"特性——明确输入输出,内部实现可独立修改。 | 概念讲解 讲解函数定义格式: <pre>def add_book(name, author): # 函数体 return result</pre> 演示参数传递与返回值的使用。 原理阐释 用"工厂流水线"类比函数:输入原料（参数）→ 加工处理（函数体）→ 输出产品（返回值）。 | 跟随实践 定义简单函数（如计算两数之和）,尝试不同参数类型和返回值; 对比理解 对比有参数/无参数、有返回值/无返回值的不同函数类型。 | 通过生活化类比降低抽象概念的理解难度;通过多样化示例建立函数的完整认知;通过动手实践巩固基础语法。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|--------------------------|--|---|---|--|
| 3. 功能函数封装 10分钟 | 指导学生将图书录入、查询、统计等功能代码逐一封装为独立函数,明确每个函数的职责、参数设计和返回值,强调函数命名的规范性（动词+名词）。 | <p>任务分解 列出需封装的函数清单: <code>add_book()</code> , <code>query_book()</code> , <code>show_statistics()</code> , <code>display_menu()</code> 等;</p> <p>示例演示 演示封装查询功能:</p> <pre>def query_book(book_dict, book_name): return book_dict.get(book_name, "未找到")</pre> <p>讲解参数选择（需要图书字典和书名）与返回值设计。</p> | <p>协作封装 小组分工,每组负责封装1-2个功能函数,讨论参数设计合理性;</p> <p>代码审查 小组间交换检查函数代码,验证是否符合"单一职责"原则。</p> | 通过任务分解将复杂项目拆解为可管理的小模块;通过示例演示建立函数封装的标准流程;通过小组协作模拟真实开发中的分工合作。 |
| 4. 菜单系统设计 9分钟 | 指导学生设计主菜单系统:使用while循环实现持续交互,用if-elif分支调用不同功能函数,实现"显示菜单→用户选择→执行功能→返回菜单"的完整流程。 | <p>流程设计 绘制菜单系统流程图:循环显示选项→获取用户输入→分支判断→调用对应函数→判断是否退出;</p> <p>代码框架 提供主程序框架:</p> <pre>while True: display_menu() choice = input("请选择:") if choice == '1': add_book(...) elif ...</pre> <p>演示如何整合各函数。</p> | <p>理解逻辑 分析菜单循环的控制逻辑,理解"持续交互"的实现原理;</p> <p>完善代码 在框架基础上补充完整的选项分支,调用已封装的功能函数,实现系统集成。</p> | 通过流程图可视化菜单逻辑,降低循环嵌套的理解难度;通过提供框架降低编码门槛,让学生聚焦于功能整合;通过系统集成体验模块化开发的优势。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------|--|---|---|--|
| 5. 系统测试优化 5分钟 | 指导学生进行完整系统测试:执行录入→查询→统计→退出的完整流程,发现并修复bug(如输入异常、功能冲突),优化用户体验(添加提示信息、美化输出)。 | 测试指导 发布测试任务清单:①测试所有菜单选项;②测试异常输入(如输入字母而非数字);③检查退出功能; 问题诊断 巡视学生测试,帮助定位和解决常见问题(如未正确传递参数、循环无法退出等)。 | 全流程测试 模拟真实用户操作,按照测试清单逐项验证功能; 问题修复 记录发现的bug,修改代码并重新测试,直到系统稳定运行。 | 通过系统化测试培养质量意识和调试能力;通过问题解决过程提升代码纠错技能;通过用户体验优化建立产品思维。 |
| 6. 项目总结展望 2分钟 | 总结项目三个课时的完整开发流程(数据存储→高效查询→模块化集成),点评模块化设计对大型项目的重要性,预告后续可扩展的方向(如数据持久化、图形界面)。 | 项目回顾 展示项目从"混乱代码"到"结构清晰系统"的演变,强调函数封装带来的可维护性提升; 能力肯定 肯定学生已掌握"数据结构+算法+工程化"的综合能力,鼓励课后继续完善功能。 | 成果展示 小组展示完成的系统运行效果,交流设计思路; 反思总结 填写项目开发日志,记录收获与待改进之处。 | 通过项目回顾强化知识体系的完整性;通过成果展示增强学生的成就感和自信心;通过反思总结培养持续改进的职业素养。 |

教学成效与反思

| | |
|------|---|
| 教学成效 | 本课时成功将分散的功能代码整合为模块化系统,约90%的学生能够准确说明函数封装的作用并完成至少3个功能函数的编写。全体学生完成了带菜单循环的完整系统搭建,其中70%以上能独立设计函数的参数与返回值。小组协作封装环节展现出良好的团队分工能力,代码审查环节学生能够相互指出设计不合理之处。通过三课时的项目开发,学生从"学语法"转变为"做项目",对Python编程的实用价值有了深刻认知,编程思维和工程素养显著提升。系统测试环节学生主动发现并修复bug的能力超出预期,体现出较强的问题解决能力。 |
| 教学反思 | 本课时通过"混乱代码"与"模块化代码"的对比成功建立了函数封装的必要性认知,学生的学习动机强烈。函数基础知识的讲解结合项目实例,避免了抽象概念的枯燥感。不足之处:①部分学生对函数参数传递(特别是可变对象如字典的传递)理解不够深入,出现"修改参数影响原数据"的困惑,后续应增加内存模型的可视化讲解;②菜单系统的异常处理(如用户输入非法字符)在时间压力下讲解不够充分,部分学生的系统在异常输入时会崩溃,应提供异常处理的代码模板;③小组协作时部分组员参与度不均,个别学生只是"观看"而非"编码",需要设计更明确的分工机制和个人任务检查点。改进方向:提前准备函数参数传递的动画演示;在功能函数模板中预留异常处理的位置并提示学生填充;在小组任务中明确每个成员必须完成的最小编码量。整体来看,三课时的项目式教学成功地将数据结构、算法思维、工程化实践有机融合,学生的综合能力得到了全面锻炼,达到了预期的教学目标。 |

天气数据自动采集与推送系统——HTTP通信与天气API数据获取 教学设计

| | |
|-----------|---|
| 课题 | 天气数据自动采集与推送系统——HTTP通信与天气API数据获取 |
| 课时 | 1课时(40分钟) |
| 教学目标 | 知识目标: 技能目标: 素养目标: |
| 教学 重难点 | 重点: 难点: |
| 教学资源准备 | 多媒体课件（含HTTP通信原理动画）；和风天气API密钥（或其他免费天气API）；示例API响应JSON数据；requests库安装包；项目代码模板文件。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------|---|--|--|---|
| 1. 项目任务导入 7分钟 | 展示项目最终目标（自动采集多城市天气并推送），明确本课时任务是实现项目的核心模块——数据采集功能，引出"如何让Python程序从互联网获取天气数据"这一技术问题。 | 场景创设 演示手机天气APP的实时更新效果，提问："这些数据从哪来？我们的Python程序如何获取？" 任务发布 展示项目架构图，标注本课时目标：编写weather_fetcher.py模块，实现"输入城市名→返回温度、天气"功能。 | 观察思考 观看演示，联系日常使用经验，思考数据来源； 明确目标 理解本课时在项目中的定位：数据采集是后续所有功能的基础。 | 通过真实应用场景建立技术需求认知；通过项目架构图让学生明确本课时任务在整体中的关键作用，强化目标导向。 |
| 2. 核心概念讲解 8分钟 | 讲解HTTP协议的客户端-服务器模型，介绍REST API的概念、特点与调用方式，展示天气API文档的关键内容（接口地址、参数、响应格式）。 | 原理阐述 用动画演示HTTP请求过程：Python程序（客户端）→发送请求→服务器→返回数据；强调"URL+参数"的请求构造方式； 文档解读 打开和风天气API文档，逐项讲解：接口URL、必需参数（key、location）、响应JSON结构，演示在浏览器中直接访问API。 | 聆听理解 理解HTTP的"请求-响应"机制，记录API调用的三要素：地址、参数、响应格式； 实例观察 观看浏览器访问API的演示，直观感受JSON数据的返回，尝试识别数据中的城市、温度字段。 | 将抽象的网络通信概念具象为"发送网址+参数，获得数据"的简单模型；通过浏览器演示消除技术神秘感，建立"API就是特殊网址"的认知。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|----------------------------|---|--|--|--|
| 3. 工具库与代码演示 10分钟 | 介绍requests库的作用与安装，演示使用requests.get()调用天气API的完整代码，讲解参数传递、响应处理、JSON解析的具体方法。 | <p>库的引入 说明requests是Python的HTTP客户端库，演示pip install requests安装过程；</p> <p>代码演示 边讲解边编写代码：</p> <pre>import requests url = "https://api.qweather.com/v7/weather/now" params = {"key": "你的密钥", "location": "101010100"} response = requests.get(url , params=params) data = response.json() print(f"温度： {data['now'] ['temp']}℃")</pre> <p>逐行解释：URL定义、参数字典、发送请求、解析JSON、字段访问；</p> <p>运行展示 运行代码，展示获取到的北京天气数据。</p> | <p>安装实践 跟随教师完成requests库的安装确认；</p> <p>代码跟随 边听边在自己的IDE中输入代码，理解每行的作用；</p> <p>结果验证 运行代码，观察控制台输出的天气数据，对照API文档验证字段。</p> | 通过完整代码演示建立"库调用-参数传递-数据解析"的操作流程；通过逐行讲解降低理解难度；通过即时运行验证建立"代码有效"的信心。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|----------------------------|--|--|---|--|
| 4. 功能模块编写 12分钟 | 指导学生编写完整的天气数据获取函数，封装为项目模块，要求实现：输入城市名、返回温度和天气状况、添加异常处理、输出格式化结果。 | 任务细化 发布编码任务：创建weather_fetcher.py，定义get_weather(city)函数，实现上述功能；提供城市代码对照表； 技术提示 提示关键点：使用字典存储多城市代码、用try-except捕获网络异常、格式化输出使用f-string； 巡回指导 巡视学生编码，解答JSON嵌套访问、异常处理等问题，强调代码注释与函数文档字符串的编写。 | 独立编码 创建py文件，定义函数框架，填充请求逻辑； 调试测试 测试不同城市（如"北京""上海"），验证数据准确性；尝试故意输入错误密钥，观察异常处理效果； 代码优化 根据测试结果调整代码，添加注释说明各部分功能。 | 通过独立编码巩固API调用技能，形成可复用的项目模块；通过测试培养"编码-调试-优化"的开发习惯；通过异常处理强化代码健壮性意识。 |
| 5. 成果展示与项目展望 3分钟 | 选取学生作品展示运行效果，总结HTTP通信与API调用的核心要点，预告后续课程将在此基础上实现多城市并发采集与数据存储。 | 作品点评 选2-3名学生演示代码，点评优点（如良好的异常处理）与改进点； 知识串联 总结：本课时完成了项目的"数据源"模块，后续将学习用多线程同时采集多城市、用数据库存储历史数据、用定时任务实现自动化； 任务布置 课后任务：为函数添加"获取未来3天天气"的功能（需查阅API文档）。 | 观摩学习 观看同学作品，学习不同实现思路； 总结反思 回顾requests使用流程，明确本模块在项目中的"基石"作用； 接收任务 记录课后任务，理解这是对API调用能力的深化练习。 | 通过作品展示建立学习榜样，强化成就感；通过项目展望保持学习连贯性，明确当前任务与后续模块的逻辑关系；课后任务引导学生自主探索API文档。 |

教学成效与反思

| | |
|------|---|
| 教学成效 | 结合本课时项目任务评估：85%以上学生成功调用API获取天气数据并正确解析，75%学生能够独立编写带异常处理的完整函数。通过"浏览器直接访问API"的演示，有效降低了学生对网络通信的畏难情绪。学生对requests库的"三行代码实现网络数据获取"表现出浓厚兴趣，课堂参与度高。项目数据采集模块的基础功能已基本实现，为后续多线程、数据存储等课时奠定了坚实基础。部分学生在调试JSON嵌套字段访问时遇到困难，但在教师指导下均得以解决。 |
| 教学反思 | 本课时成功将"HTTP通信与REST API"这一理论性较强的主题，转化为"获取天气数据"的具体项目任务，学生的代入感和学习动力明显增强。API文档解读环节设计合理，通过浏览器演示让抽象的"接口调用"变得直观可感。代码演示采用"边讲边写"方式，学生跟随效果好于预期。不足之处：①在40分钟内，部分学生因requests库安装出现网络问题导致进度延迟，建议提前让学生课下完成库安装或准备离线安装包；②对于JSON嵌套结构的访问（如data['now']['temp']），讲解略显仓促，少数学生理解不够深入，建议增加一个"JSON结构可视化"的辅助图示；③异常处理部分主要由教师演示，学生自主实践时间不足，后续可考虑设计专门的"错误处理"小任务强化训练。整体上，项目驱动的教学框架使技术学习目标清晰，学生编写的weather_fetcher.py模块已具备项目集成的条件，教学目标达成度较高。 |

天气数据自动采集与推送系统——多线程并发采集多城市数据 教学设计

| | |
|-----------|--|
| 课题 | 天气数据自动采集与推送系统——多线程并发采集多城市数据 |
| 课时 | 1课时(40分钟) |
| 教学目标 | 知识目标: 技能目标: 素养目标: |
| 教学 重难点 | 重点: 难点: |
| 教学资源准备 | 多媒体课件（含进程与线程对比图、并发执行动画）；上节课完成的weather_fetcher.py模块；多城市代码列表文件；性能测试脚本模板；线程池使用示例代码。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|--------------------------|---|---|--|---|
| 1. 性能瓶颈引入 6分钟 | 演示串行采集10个城市天气数据的耗时问题，提出"如何加速数据采集"的技术需求，引入多线程并发编程的概念。 | 问题演示 运行上节课的weather_fetcher模块，用循环串行采集10个城市，用time模块记录总耗时（约10-15秒），提问："如果要采集100个城市呢？" 概念引入 展示并发执行示意图，说明"同时发送多个请求"的思想，介绍线程是实现并发的一种方式。 | 观察分析 观看串行执行过程，感受耗时问题，理解在网络请求中"等待响应"的时间占主导； 讨论思考 小组讨论："能否让程序同时请求多个城市的数据？"初步建立并发需求认知。 | 通过实际运行建立性能痛点，让学生体会"串行等待"的低效；通过对比引出并发方案，明确本课时要解决的核心问题。 |
| 2. 多线程原理讲解 8分钟 | 讲解进程与线程的基本概念、区别与应用场景，介绍Python的threading模块，演示线程的创建、启动与join()方法的使用。 | 概念对比 用类比说明：进程是独立的程序实例（如打开的软件），线程是程序内部的执行路径（如软件的多个功能同时运行）；强调线程共享内存、切换快的特点； 基础演示 编写简单示例： <pre>import threading def task(name): print(f" {name}线程执行 中") t1 = threading.Thread (target=task, args=("A",)) t1.start() t1.join()</pre> 讲解Thread对象创建、target参数、start()启动、join()等待。 | 聆听理解 理解"并发≠并行"的概念，记录线程的三要素：目标函数、参数、启动方法； 代码跟随 在IDE中输入演示代码，运行观察线程执行效果，尝试创建多个线程并观察输出顺序的随机性。 | 通过类比降低抽象概念的理解难度；通过简单示例建立"创建线程-启动-等待"的基本操作流程，为后续复杂应用打基础。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|---|---|---|
| 3. 并发采集实现 12分钟 | 指导学生改造天气采集代码，使用线程实现多城市并发请求，演示ThreadPoolExecutor的使用方法，对比串行与并发的性能差异。 | <p>方案讲解 说明改造思路：为每个城市创建独立线程调用get_weather()，将结果存入列表；</p> <p>代码演示 演示手动创建线程版本，然后引入线程池简化代码：</p> <pre>from concurrent.futures import ThreadPoolExecutor cities = ["北京", "上海", "广州", ...] with ThreadPoolExecutor(max_workers=5) as executor: results = executor.map(get_weather, cities)</pre> <p>讲解线程池的优势：自动管理线程数量、简化代码；</p> <p>性能对比 分别运行串行版本和并发版本，展示耗时对比（如从15秒降至3秒）。</p> | <p>理解方案 理解"每个城市一个线程"的并发模型，记录线程池的使用模板；</p> <p>实践编码 创建multi_thread_fetcher.py，使用ThreadPoolExecutor改造采集代码；</p> <p>测试验证 运行代码，记录并对比串行与并发的耗时数据，体会性能提升。</p> | 通过代码改造巩固多线程应用能力；通过线程池简化实现，降低并发编程门槛；通过性能对比建立直观的"并发价值"认知。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|--------------------------|---|--|--|--|
| 4. 线程安全探讨 10分钟 | 讲解多线程中的数据竞争问题，演示不安全的共享数据访问，介绍线程锁(Lock)的使用方法，指导学生在项目中安全收集结果。 | <p>问题演示 演示多线程同时修改同一列表可能出现的问题（如数据丢失），说明这是"数据竞争"；</p> <p>解决方案 介绍threading.Lock()的使用：</p> <pre>lock = threading.Lock() with lock: shared_list.append(data)</pre> <p>强调在项目中应使用线程安全的数据结构或加锁保护；</p> <p>最佳实践 推荐使用线程池的map()或submit()方法返回结果，避免手动管理共享数据。</p> | <p>观察问题 观看演示，理解多线程同时访问数据的风险；</p> <p>学习方案 理解Lock的"互斥访问"机制，记录加锁的代码模板；</p> <p>优化代码 检查自己的并发采集代码，确保结果收集方式是线程安全的，如使用executor.map()的返回值而非手动append。</p> | 通过问题演示建立线程安全意识；通过解决方案教学培养规范编程习惯；通过代码优化强化"安全第一"的工程理念。 |
| 5. 项目集成与展望 4分钟 | 总结多线程并发编程的核心要点，将本课时的并发采集模块集成到项目中，预告后续课程将实现定时自动采集与数据存储。 | <p>知识总结 回顾threading模块使用流程、ThreadPoolExecutor优势、线程安全注意事项；</p> <p>模块集成 说明multi_thread_fetcher可作为项目的"高效采集引擎"，后续定时任务将调用此模块；</p> <p>任务布置 课后任务：尝试调整线程池的max_workers参数（如5、10、20），测试不同线程数对性能的影响，思考最优值。</p> | <p>回顾反思 总结并发编程的关键点，明确本模块在项目中的"性能优化"作用；</p> <p>接收任务 记录课后任务，理解这是对并发性能调优的探索性实验。</p> | 通过总结强化知识结构；通过模块集成保持项目连贯性；课后任务引导学生探索并发参数优化，培养实验精神。 |

教学成效与反思

| | |
|------|--|
| 教学成效 | 结合本课时项目任务评估：90%学生成功使用ThreadPoolExecutor实现多城市并发采集，80%学生能够通过性能测试数据说明并发的优势。通过串行与并发的直观对比（耗时从15秒降至3秒），学生对多线程价值有了深刻认知，学习积极性高。线程池的引入有效降低了并发编程复杂度，学生普遍反馈"比想象中简单"。项目的并发采集模块已具备实用性，为后续定时任务提供了高效的数据获取能力。线程安全环节虽有难度，但通过问题演示和最佳实践指导，学生建立了基本的安全意识。 |
| 教学反思 | 本课时成功将"多线程与并发编程"这一传统难点，转化为"提升天气采集速度"的具体优化任务，学生的学习动机强且目标明确。性能对比实验设计合理，数据直观有说服力。线程池的引入时机恰当，避免了过早陷入底层线程管理的复杂性。不足之处：①在40分钟内，线程安全部分讲解略显紧凑，部分学生对Lock的理解停留在"照搬代码"层面，建议后续增加一个专门的线程安全小案例演示；②对"I/O密集型 vs CPU密集型"的区分提及较少，少数学生可能误以为多线程适用所有场景，建议增加一句说明"多线程适合网络请求等待多的任务"；③课后任务的参数调优实验虽有探索性，但缺少对"过多线程可能导致性能下降"的提示，建议补充说明。整体上，项目驱动使并发编程学习变得具体可感，学生编写的并发模块已达到生产级代码的雏形，教学目标达成度高。 |

天气数据自动采集与推送系统——定时任务与自动化采集 教学设计

| | |
|-----------|--|
| 课题 | 天气数据自动采集与推送系统——定时任务与自动化采集 |
| 课时 | 1课时(40分钟) |
| 教学目标 | 知识目标: 技能目标: 素养目标: |
| 教学 重难点 | 重点: 难点: |
| 教学资源准备 | 多媒体课件（含定时任务场景案例、系统架构图）；前两节课完成的weather_fetcher.py和multi_thread_fetcher.py模块；schedule库安装包；日志配置示例代码；完整项目运行演示视频。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|--|---|--|
| 1. 自动化场景导入 6分钟 | 展示生活中的自动化案例（如定时提醒、自动备份），引出"让天气采集程序自动运行"的需求，明确本课时要实现的项目最终目标。 | 场景列举 列举自动化应用：闹钟、定时发送报表、服务器定时备份等，提问："我们的天气系统如何做到每小时自动采集？" 目标演示 播放完整项目运行视频：程序启动后自动每30分钟采集一次数据，保存到文件并输出日志，无需人工干预。 | 联系经验 回忆日常使用的自动化功能，理解"定时执行"的价值； 明确目标 观看演示，建立"让程序自己跑起来"的认知，理解本课时是项目的"收官之作"。 | 通过生活化案例建立自动化认知；通过完整演示展示项目最终形态，激发学生的成就欲望和完成项目的动力。 |
| 2. 定时任务原理 7分钟 | 讲解定时任务的实现机制，介绍Python的schedule库，演示基本的任务调度代码，说明"调度循环"的概念。 | 原理讲解 说明定时任务的本质：程序持续运行，定期检查是否到达执行时间；介绍schedule库是对时间检查逻辑的封装； 基础演示 编写简单示例： <pre>import schedule import time def job(): print("任务执行！") schedule.every(10).seconds.do(job) while True: schedule.run_pending() time.sleep(1)</pre> 逐行讲解：定义任务函数、设置调度规则、启动循环检查。 | 聆听理解 理解"持续检查+触发执行"的调度机制，记录schedule的核心API； 代码跟随 输入演示代码并运行，观察每10秒自动执行一次的效果，尝试修改间隔时间（如30秒、1分钟）。 | 通过原理讲解揭开定时任务的"神秘面纱"；通过简单示例建立"设置规则-启动循环"的操作模型，为项目集成打基础。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|---|---|---|
| 3. 项目模块集成 13分钟 | 指导学生创建主程序文件，整合前两节课的采集模块，实现"定时自动采集多城市天气并保存"的完整功能，添加时间戳与日志输出。 | <p>架构讲解 说明项目文件结构：weather_fetcher.py（单城市）、multi_thread_fetcher.py（并发）、main_scheduler.py（定时主程序）；</p> <p>集成演示 演示 main_scheduler.py 编写过程：</p> <pre>import schedule from multi_thread_fetcher import fetch_all_cities def auto_fetch(): data = fetch_all_cities() # 保存到文件，添加时间戳 print(f"[{time.strftime('%Y-%m-%d %H:%M:%S')}]] 采集完成") schedule.every(30).minutes.do(auto_fetch)</pre> <p>讲解数据保存方案（CSV或JSON）、时间戳格式化；</p> <p>运行测试 启动程序，观察首次执行和后续自动触发的效果。</p> | <p>理解架构 明确各模块的职责分工，理解 main_scheduler 是"指挥中心"；</p> <p>动手编码 创建 main_scheduler.py，导入并发采集模块，设置定时规则（如每30分钟），添加数据保存代码；</p> <p>测试验证 运行程序，确认定时执行正常，检查保存的数据文件格式与内容。</p> | 通过模块集成培养系统化编程思维；通过完整实现建立"小模块组成大系统"的认知；通过实际运行验证项目的实用性。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------|--|--|--|---|
| 4. 稳定性优化 10分钟 | 讲解长时间运行程序的常见问题，指导学生添加异常处理、日志记录、资源管理等代码，确保程序稳定可靠。 | <p>问题分析 说明长期运行可能遇到的问题：网络故障、API限流、内存泄漏等；强调健壮性的重要性；</p> <p>优化方案 演示添加try-except包裹任务函数、配置logging模块记录日志：</p> <pre>import logging logging.basicConfig(filename='weather.log', level=logging.INFO) def auto_fetch(): try: # 采集逻辑 logging.info("采集成功") except Exception as e: logging.error(f"采集失败: {e}")</pre> <p>讲解日志级别（INFO、ERROR）、日志文件的查看方法；</p> <p>最佳实践 建议添加程序启动提示、优雅退出机制（Ctrl+C处理）。</p> | <p>认识风险 理解"程序不能假设一切正常"的思想，记录异常处理的必要性；</p> <p>优化代码 为auto_fetch()添加try-except，配置logging输出到文件，测试故意制造错误（如API密钥错误）观察日志记录；</p> <p>完善程序 添加程序启动提示信息，如"天气采集系统已启动，每30分钟执行一次"。</p> | 通过问题分析培养风险意识；通过优化方案教学提升代码质量；通过日志实践建立生产级程序的开发习惯。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|--|--|--|
| 5. 项目总结与展望 4分钟 | 回顾整个项目的三个核心模块，总结从HTTP请求到多线程再到自动化的完整技术链条，展望后续可扩展的功能方向。 | 项目回顾 梳理项目实现路径：第1课时建立数据源→第2课时提升采集效率→第3课时实现自动化运行； 成果展示 请学生展示运行中的程序和保存的数据文件，分享项目完成的成就感； 扩展方向 提出可选的增强功能：数据可视化（matplotlib）、极端天气告警推送（邮件/短信）、Web界面展示等； 任务布置 课后任务：让程序在电脑上持续运行24小时，观察日志并统计采集成功率。 | 总结反思 回顾三次课的学习历程，理解"技术服务于需求"的思想； 展示成果 演示自己的运行程序，分享实现过程中的收获与困难； 展望未来 思考项目的实用场景和可改进方向，接收课后实验任务。 | 通过系统回顾强化知识体系；通过成果展示建立成就感与自信；通过扩展方向保持学习兴趣和探索欲望。 |

教学成效与反思

| | |
|------|---|
| 教学成效 | 结合本课时项目任务评估：95%学生成功实现定时自动采集功能，85%学生能够添加完整的异常处理与日志记录。通过三节课的连贯学习，学生完整经历了"单功能模块→性能优化→系统集成"的项目开发过程，对软件工程的模块化思想有了深刻体会。定时任务的实现让学生看到了"程序自己工作"的魅力，课堂兴奋度和成就感达到高峰。项目最终成果具有实用性，部分学生表示愿意持续运行并收集数据做后续分析，显示出强烈的学习内驱力。日志记录的引入提升了学生对"生产级代码"的认知。 |
| 教学反思 | 本课时成功将"定时任务与自动化"概念转化为"让天气系统自动运行"的具体目标，学生的学习动机和参与度极高。三节课形成的完整项目链条设计合理，每节课的模块都能无缝集成到最终系统中，验证了项目式教学的有效性。schedule库的简洁API降低了定时任务的学习门槛，学生普遍反馈"比想象中容易"。不足之处：①在40分钟内，对"程序后台运行"的操作系统层面知识（如nohup、系统服务等）未能涉及，部分学生询问"如何让程序关闭终端后仍运行"，建议作为课后拓展资料补充；②日志配置部分主要由教师演示，学生自主设计日志格式的时间不足，建议提供更灵活的日志模板让学生定制；③对"定时任务的精度限制"（如schedule是软实时）未做说明，可能导致学生对时间准确性有过高期待。整体上，项目式教学使三个看似独立的技术主题（HTTP、多线程、定时任务）自然融合为一个有机整体，学生在完成项目的过程中建立了系统化思维和工程实践能力，教学目标超预期达成。建议后续可将此项目作为综合实训案例，引导学生继续扩展功能（如添加Web界面、数据分析模块等）。 |

数字日记本系统——日记文本文件存储模块

教学设计

| | |
|--------|--|
| 课题 | 数字日记本系统——日记文本文件存储模块 |
| 课时 | 1课时(40分钟) |
| 教学目标 | <p>知识目标: 理解文件操作的基本概念和数据持久化的意义，掌握文件的打开模式（读、写、追加）及其区别，理解文件路径、编码格式等基础知识。</p> <p>技能目标: 能够使用open()函数和with语句进行安全的文件操作，实现日记内容的写入、追加和读取功能，能编写代码处理文件异常（如文件不存在）和中文编码问题。</p> <p>素养目标: 建立"数据持久化是程序实用性的关键"意识，培养安全文件操作习惯（使用with自动关闭），理解数字化记录对生活管理的价值。</p> |
| 教学重难点 | <p>重点: 文件的打开与关闭（with语句）；文件的读取方法（read、readline、readlines）；文件的写入模式（'w'覆盖、'a'追加）；中文编码处理（encoding='utf-8'）。</p> <p>难点: 理解不同打开模式的适用场景；处理文件路径问题（相对路径vs绝对路径）；设计合理的日记存储格式（如时间戳+内容）；异常处理（文件不存在、权限问题）。</p> |
| 教学资源准备 | 多媒体课件（含文件操作原理动画）；项目演示视频（展示日记写入和读取效果）；示例日记文本文件；Python开发环境。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|--|--|--|
| 1. 项目情境导入 5分钟 | 展示传统纸质日记的局限性（不便搜索、易丢失、无法备份），引出数字日记本系统的开发需求，明确本课时任务：实现日记内容的文件存储与读取。 | 场景对比 展示纸质日记本和数字日记的对比图，提问："如何让Python程序记住我们写的日记，即使关闭程序后也能找回？" 需求分析 明确项目第一步：将日记保存到电脑硬盘的文本文件中，实现数据持久化。 | 问题思考 回顾之前学过的数据存储方式（变量、列表、字典），思考它们的局限性（程序关闭即消失）； 目标认同 理解文件存储的必要性：数据永久保存、可随时读取。 | 通过生活化场景建立项目情境，让学生理解"数据持久化"的实际意义；通过对比引发认知冲突，激发对文件操作的学习需求。 |
| 2. 文件操作基础 10分钟 | 讲解文件操作的基本流程（打开→读写→关闭），演示open()函数的使用和with语句的安全性，对比不同打开模式（'r'只读、'w'覆盖写、'a'追加写）的效果。 | 概念讲解 讲解文件操作三步骤：打开文件（指定路径和模式）→操作文件（读或写）→关闭文件（释放资源）； 语法演示 演示基本语法： <div><pre>with open('diary.txt' , 'w', encoding='utf- 8') as f: f.write('今 天学习了Python文 件操作')</pre></div> 强调with语句的自动关闭优势。 | 跟随实践 在交互环境中尝试创建并写入一个简单文本文件，观察文件在磁盘上生成； 模式对比 分别测试'w'和'a'模式，观察多次写入时的覆盖与追加差异。 | 通过直观演示建立文件操作的基本认知；通过with语句强调编程规范；通过模式对比实验让学生体会不同场景的技术选择。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------|--|--|--|---|
| 3. 日记写入功能 8分钟 | 指导学生设计日记条目的存储格式（包含日期、时间、内容），编写日记写入函数，实现用户输入日记内容后追加到日记文件中，并自动添加时间戳。 | <p>格式设计 讲解日记条目格式设计：</p> <pre>===== ===== 日期：2026-01-07 15:30 内容：今天学习了 文件操作... ===== =====</pre> <p>代码示范 演示获取当前时间并格式化输出：</p> <pre>from datetime import datetime now = datetime.now().s trftime('%Y-%m- %d %H:%M')</pre> <p>演示完整的写入流程。</p> | <p>功能实现 编写write_diary()函数，实现以下功能：①获取用户输入的日记内容；②自动添加时间戳；③追加写入到diary.txt文件；</p> <p>功能测试 多次运行程序写入不同日记，打开文件验证内容是否正确追加。</p> | 通过格式设计培养结构化数据思维；通过datetime库的引入扩展学生的技术视野；通过完整功能实现将知识转化为实用技能。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|--|--|---|
| 4. 日记读取功能 10分钟 | 指导学生编写日记读取函数，实现打开日记文件并显示所有历史日记内容，处理文件不存在的异常情况，优化显示格式使其易读。 | <p>读取方法 对比三种读取方法： read()读取全部、 readline()逐行读取、 readlines()返回行列表，演示各自适用场景；</p> <p>异常处理 演示try-except处理文件不存在错误：</p> <pre>try: with open('diary.txt' , 'r', encoding='utf- 8') as f: content = f.read() except FileNotFoundError: print("还没有 日记记录哦！")</pre> <p>代码优化 引导美化输出格式，如添加分隔线、颜色等。</p> | <p>编码实践 编写read_diary()函数，读取并显示所有日记内容，处理文件不存在情况；</p> <p>问题解决 调试过程中可能遇到的编码问题（中文乱码），学习使用encoding参数解决。</p> | 通过多种读取方法的对比培养选择最优方案的能力；通过异常处理培养健壮代码意识；通过实际调试提升问题解决能力。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------------|--|---|---|---|
| 5. 功能整合测试 5分钟 | 整合写入和读取功能，实现简单的菜单系统（1.写日记 2.查看日记 3.退出），进行完整流程测试，发现并解决问题。 | 集成指导 指导搭建简单菜单： <pre>while True: print("1.写日记 2.查看日记 3.退出") choice = input("请选择：") if choice == '1': write_diary() ...</pre> 测试引导 引导完整测试：写入多条日记→查看→退出→重新运行→验证数据是否持久保存。 | 系统搭建 整合前面编写的函数，搭建带菜单的完整系统； 全流程验证 测试数据持久化效果，验证程序关闭后再次打开能否读取之前的日记。 | 通过功能整合体验模块化开发的优势；通过持久化验证让学生直观感受文件操作的核心价值；通过真实使用增强成就感。 |
| 6. 总结与展望 2分钟 | 总结文件操作的核心知识点和在项目中的应用，分析当前方案的不足（查询不便、格式单一），预告下节课将学习数据库实现更强大的管理功能。 | 知识梳理 回顾文件操作核心：打开模式、读写方法、异常处理、编码问题； 问题引导 提问："如果有100条日记，如何快速找到某个日期或包含特定关键词的日记？"引出数据库的必要性。 | 反思总结 总结收获：实现了数据永久存储，系统可以真实使用了； 思考改进 思考文本文件存储的局限性，期待学习更高效的存储方案。 | 通过总结强化知识体系；通过问题引导为下节课数据库学习埋下伏笔，保持项目学习的连贯性和期待感。 |

教学成效与反思

| | |
|------|--|
| 教学成效 | 本课时成功将文件操作知识置于"数字日记本"的实用项目中，约85%的学生能够准确说明文件打开模式的区别并在项目中正确应用。全体学生完成了日记写入和读取功能的编码，其中80%以上能够正确处理中文编码和文件异常。通过"写日记→关闭程序→重新打开→查看历史"的完整流程测试，学生对数据持久化的概念有了深刻的直观认知。课堂气氛活跃，多数学生表示课后愿意继续使用和完善这个日记系统，学习动机强烈。项目的实用性成功激发了学生对编程解决生活问题的兴趣。 |
| 教学反思 | 本课时通过"数字日记本"这一贴近学生生活的场景成功建立了学习情境，文件操作这一相对枯燥的知识点变得有趣且实用。with语句的安全性和encoding参数的必要性通过实际问题（资源泄漏、中文乱码）得到了有效强调。不足之处：①文件路径概念讲解不够充分，部分学生不理解相对路径，导致在不同目录运行程序时找不到文件，应增加路径可视化讲解和工作目录的概念；②日记格式设计环节时间略紧，部分学生的格式不够规范，可考虑提供标准格式模板；③对read()、readline()、readlines()三种方法的应用场景区分不够清晰，部分学生在处理大文件时选择了不当方法，后续应增加性能对比演示。改进方向：提前准备路径概念的图示材料；提供日记格式的参考模板；设计一个"读取大文件"的对比实验让学生体会不同方法的性能差异。整体来看，项目驱动的教学方式让抽象的文件操作变得具体可感，学生的实践能力和问题解决能力得到了有效锻炼。 |

数字日记本系统——日记数据库管理模块

教学设计

| | |
|--------|---|
| 课题 | 数字日记本系统——日记数据库管理模块 |
| 课时 | 1课时(40分钟) |
| 教学目标 | <p>知识目标: 理解数据库的基本概念及其相对于文本文件的优势，掌握SQLite数据库的特点和应用场景，理解表、记录、字段等数据库术语，掌握SQL语句的基本语法（CREATE、INSERT、SELECT）。</p> <p>技能目标: 能够使用Python的sqlite3模块连接数据库、创建数据表、执行增删改查操作，实现日记数据从文件存储到数据库存储的迁移，能编写按日期、关键词查询日记的功能代码。</p> <p>素养目标: 建立"结构化数据管理提升系统能力"的工程意识，培养数据建模思维（设计合理的表结构），理解数据库在现代应用中的核心地位。</p> |
| 教学重难点 | <p>重点: SQLite数据库的连接与关闭；数据表的创建（CREATE TABLE）；数据的插入（INSERT）和查询（SELECT）；Python与数据库的交互流程（连接→游标→执行→提交→关闭）。</p> <p>难点: 理解数据库相对于文本文件的本质优势；设计合理的日记表结构（字段类型选择）；SQL语句的编写和参数化查询（防止SQL注入）；查询结果的处理（fetchall、fetchone）。</p> |
| 教学资源准备 | 多媒体课件（含数据库原理动画）；SQLite数据库可视化工具（如DB Browser）；上节课的日记文本文件；项目演示视频。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|---|--|---|
| 1. 问题情境导入 6分钟 | 回顾文本文件存储方案，演示在大量日记中查找特定内容的困难（需要逐行读取和字符串匹配），引出数据库的必要性，明确本课任务：将日记迁移到数据库实现高效管理。 | 效率对比 演示在包含100条日记的文本文件中查找"2026-01-05"的日记，展示代码复杂度和耗时； 需求分析 提问："如何像图书馆检索系统一样，通过条件快速找到想要的日记？"引出数据库的索引和查询优势。 | 问题体验 尝试在上节课的日记文件中手动查找特定日期的日记，感受文本查找的低效； 概念对比 讨论：文本文件vs数据库，在数据量大、查询频繁时哪个更合适？ | 通过效率问题创造认知需求，让学生理解"为什么需要数据库"；通过类比图书馆检索系统建立数据库查询的直观认知。 |
| 2. 数据库基础知识 9分钟 | 讲解数据库的基本概念（数据库、表、记录、字段），介绍SQLite的特点（轻量级、无需服务器、单文件），演示数据库可视化工具查看表结构，对比文本存储与数据库存储的差异。 | 概念讲解 用Excel表格类比讲解：数据库=工作簿、表=工作表、记录=行、字段=列； SQLite介绍 讲解SQLite优势：Python内置支持、数据存储在每个.db文件中、适合小型应用； 可视化演示 使用DB Browser打开示例数据库，展示表结构 and 数据查看方式。 | 类比理解 结合Excel使用经验理解数据库结构概念； 工具体验 打开示例数据库文件，观察表结构、字段类型、记录内容，建立直观认知。 | 通过Excel类比降低抽象概念的理解难度；通过可视化工具让学生直观看到数据库的组织方式；通过对比建立数据库的核心优势认知。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------------|--|--|--|---|
| 3. 数据表创建 8分钟 | 指导学生设计日记表的结构（字段：id主键、日期、时间、内容、标签），编写Python代码连接数据库并创建表，讲解SQL的CREATE TABLE语法和数据类型。 | <p>表结构设计</p> <p>引导设计日记表字段：</p> <pre>CREATE TABLE diary (id INTEGER PRIMARY KEY, date TEXT, time TEXT, content TEXT, tags TEXT)</pre> <p>讲解主键的作用和TEXT、INTEGER等类型；</p> <p>代码演示</p> <p>演示Python连接数据库并创建表：</p> <pre>import sqlite3 conn = sqlite3.connect('diary.db') cursor = conn.cursor() cursor.execute(' CREATE TABLE ...') conn.commit() conn.close()</pre> <p>强调连接→游标→执行→提交→关闭的标准流程。</p> | <p>设计讨论</p> <p>小组讨论：日记表还需要哪些字段？心情、天气、地点等如何设计？</p> <p>编码实践</p> <p>编写代码创建日记数据表，运行后用可视化工具验证表是否创建成功。</p> | 通过表结构设计培养数据建模思维；通过标准操作流程建立规范的数据库编程习惯；通过可视化验证增强成功反馈。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|---------------------------|---|--|---|--|
| 4. 数据插入与查询 10分钟 | 指导学生编写日记插入功能（INSERT语句），实现从用户输入到数据库存储的完整流程，编写查询功能（SELECT语句），实现按日期范围和关键词查询日记。 | <p>插入操作 演示INSERT语法和参数化查询（防SQL注入）：</p> <pre>cursor.execute('INSERT INTO diary (date, time, content) VALUES (?, ?, ?) ', (date, time, content))</pre> <p>强调使用占位符?而非字符串拼接；</p> <p>查询操作 演示SELECT查询：</p> <pre># 查询所有 cursor.execute(' SELECT * FROM diary') results = cursor.fetchall() # 按日期查询 cursor.execute(' SELECT * FROM diary WHERE date=?', (target_date,))</pre> <p>讲解WHERE子句和LIKE模糊查询。</p> | <p>插入功能 编写 add_diary_to_db()函数，获取用户输入并插入数据库，测试添加多条日记；</p> <p>查询功能 编写query_by_date()和 query_by_keyword()函数，实现按日期精确查询和按内容模糊查询，验证查询结果。</p> | 通过参数化查询强调安全编程意识；通过完整功能实现将SQL语句转化为实用技能；通过多样化查询体现数据库的强大查询能力。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|---|---|--|
| 5. 功能整合与迁移 5分钟 | 指导学生将上节课文本文件中的日记数据批量导入数据库，整合菜单系统（添加数据库查询选项），对比文件和数据库两种方案的效果差异。 | <p>数据迁移</p> <p>演示读取文本文件并批量插入数据库的代码逻辑：</p> <pre>with open('diary.txt' , 'r') as f: # 解析每条日记 # 插入数据库</pre> <p>系统升级</p> <p>指导在菜单中添加"按日期查询"、"按关键词查询"等新选项。</p> | <p>数据导入</p> <p>编写迁移脚本，将文本日记批量导入数据库；</p> <p>效果对比</p> <p>测试查询功能，对比文本文件遍历和数据库查询的速度和便利性。</p> | 通过数据迁移实现新旧系统的平滑过渡；通过效果对比让学生直观感受数据库的优势；通过系统升级体验技术进步带来的能力提升。 |
| 6. 总结与展望 2分钟 | 总结数据库的核心优势和SQL基本语句，对比文本文件与数据库的适用场景，预告下节课将学习JSON格式实现跨平台数据备份和迁移。 | <p>知识梳理</p> <p>回顾数据库操作流程和SQL四大语句（CRUD）；</p> <p>场景引导</p> <p>提问："如果想把日记备份到手机或分享给朋友，数据库文件能直接用吗？"引出数据交换格式的需求。</p> | <p>反思总结</p> <p>总结数据库带来的能力提升：快速查询、结构化管理、数据完整性；</p> <p>思考问题</p> <p>思考：如何实现数据的跨平台、跨系统共享？</p> | 通过总结强化数据库核心知识；通过问题引导为下节课JSON数据交换学习做铺垫，保持项目逻辑的连贯性。 |

教学成效与反思

| | |
|------|--|
| 教学成效 | 本课时成功将数据库编程知识融入日记系统的升级改造任务中，约75%的学生能够准确说明数据库相对于文本文件的核心优势并设计合理的表结构。全体学生完成了数据表创建和基本的增删改查操作，其中70%以上能够独立编写按日期和关键词查询的功能代码。通过文本文件与数据库查询的效率对比实验，学生对"为什么需要数据库"有了深刻的直观认知。数据迁移环节让学生体验了技术升级的完整过程，成就感强烈。部分学生主动提出增加"按心情筛选"、"标签分类"等扩展功能，显示出较强的创新意识和学习主动性。 |
| 教学反思 | 本课时通过"查询效率对比"成功建立了从文本文件到数据库的技术升级需求，学生的学习动机明确。Excel类比和可视化工具的使用有效降低了数据库抽象概念的理解难度。SQL语句的讲解结合项目实际需求，避免了语法学习的枯燥感。不足之处：①数据库连接的"连接→游标→执行→提交→关闭"流程对部分学生来说较为陌生，容易遗漏commit()或close()步骤导致数据未保存或资源泄漏，应提供更醒目的流程提示卡；②参数化查询（占位符？）的必要性（防SQL注入）讲解不够充分，部分学生仍使用字符串拼接，安全意识培养不足，后续应增加SQL注入攻击的演示案例；③数据迁移脚本编写时，日期时间的解析和格式转换难度较大，部分学生出错率高，可考虑提供正则表达式或字符串处理的辅助函数。改进方向：制作数据库操作流程的可视化图卡便于学生参考；设计一个"SQL注入危害"的演示实验增强安全意识；为数据迁移任务提供数据解析的工具函数模板。整体来看，项目的技术升级路线清晰，学生在实践中体会到了数据库的强大能力，为后续复杂应用开发奠定了基础。 |

数字日记本系统——日记数据导出与备份模块

教学设计

| | |
|--------|--|
| 课题 | 数字日记本系统——日记数据导出与备份模块 |
| 课时 | 1课时(40分钟) |
| 教学目标 | <p>知识目标: 理解JSON格式的特点及其在数据交换中的广泛应用，掌握Python数据结构与JSON格式的对应关系（字典↔对象、列表↔数组），理解序列化与反序列化的概念和作用。</p> <p>技能目标: 能够使用json模块实现Python数据结构与JSON字符串的相互转换（dumps/loads、dump/load），实现日记数据的JSON格式导出（备份）和导入（恢复/迁移）功能，能处理中文编码和日期格式化问题。</p> <p>素养目标: 建立"标准数据格式促进系统互通"的工程意识，培养数据备份和迁移的安全意识，理解开放数据格式在现代应用中的价值。</p> |
| 教学重难点 | <p>重点: JSON格式的基本语法（键值对、数组）；json.dump()和json.load()的使用（文件操作）；json.dumps()和json.loads()的使用（字符串操作）；ensure_ascii=False处理中文。</p> <p>难点: 理解序列化与反序列化的本质（内存对象↔存储格式）；从数据库查询结果转换为JSON格式（fetchall返回的元组列表需转为字典列表）；处理datetime等特殊类型的序列化（需自定义转换）。</p> |
| 教学资源准备 | 多媒体课件（含JSON格式示例）；上节课的日记数据库文件；JSON在线验证工具演示；跨平台数据交换案例视频。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|--------------------|---|--|---|--|
| 1. 数据交换需求导入 5分钟 | 展示跨平台数据共享场景（电脑日记想导入手机、分享给朋友、更换电脑后恢复数据），演示SQLite数据库文件的局限性（不同系统可能不兼容），引出标准数据交换格式的必要性，明确本课任务：实现JSON格式的数据备份与恢复。 | <p>场景演示</p> <p>展示尝试在手机上打开.db文件的困难，提问："如何让日记数据能在不同设备、不同系统间自由流动？"</p> <p>需求分析</p> <p>讲解数据交换的三大场景：备份（防丢失）、迁移（换设备）、分享（多人协作），引出JSON作为通用格式的价值。</p> | <p>问题体验</p> <p>回顾自己是否遇到过数据迁移的困难（如换手机后丢失数据）；</p> <p>目标认同</p> <p>理解"开放格式"的重要性：不依赖特定软件、人类可读、跨平台通用。</p> | 通过真实场景建立数据交换的实际需求；通过数据库局限性创造认知冲突；通过"开放格式"概念建立JSON学习的价值认知。 |
| 2. JSON格式基础 8分钟 | 讲解JSON的基本语法（对象{}、数组[]、键值对、数据类型），对比JSON与Python数据结构的对应关系，演示在线JSON验证工具，强调JSON的"人类可读"和"机器可解析"双重特性。 | <p>格式讲解</p> <p>展示JSON示例：</p> <pre>{ "date": "2026-01-07", "content": "今天学习了JSON", "tags": ["学习", "编程"] }</pre> <p>讲解语法规则：键必须用双引号、不能有注释、不支持函数等；</p> <p>类型对应</p> <p>绘制Python↔JSON对应表： dict↔对象、list↔数组、str↔字符串、int/float↔数字、True/False↔true/false、None↔null；</p> <p>工具演示</p> <p>使用在线JSON验证工具展示格式检查和美化功能。</p> | <p>格式识别</p> <p>阅读示例JSON，识别对象、数组、键值对等结构元素；</p> <p>对比学习</p> <p>对比Python字典和JSON对象的相似性与差异（如引号规则）；</p> <p>动手尝试</p> <p>在在线工具中编写简单JSON并验证格式正确性。</p> | 通过直观示例建立JSON格式的基本认知；通过类型对应表为后续序列化操作建立知识桥梁；通过工具使用降低格式学习的门槛。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|----------------------------|--|---|--|---|
| 3. 序列化与反序列化 10分钟 | 讲解序列化（Python对象→JSON字符串/文件）和反序列化（JSON→Python对象）的概念，演示json模块的四个核心函数（dumps、loads、dump、load），强调中文处理参数ensure_ascii=False。 | <p>概念讲解 讲解序列化本质：将内存中的对象转换为可存储/传输的格式；</p> <p>函数演示 演示dumps和loads（字符串操作）：</p> <pre>import json data = {"name": "日记", "count": 100} json_str = json.dumps(data, ensure_ascii=False) data_back = json.loads(json_str)</pre> <p>演示dump和load（文件操作）：</p> <pre>with open('data.json', 'w', encoding='utf-8') as f: json.dump(data, f, ensure_ascii=False, indent=2)</pre> <p>强调indent参数用于格式化输出，ensure_ascii用于正确显示中文。</p> | <p>跟随实践 创建包含中文的字典，尝试序列化为JSON字符串，观察ensure_ascii=False的效果差异；</p> <p>文件操作 将数据序列化保存到JSON文件，再读取文件反序列化回Python对象，验证数据一致性。</p> | 通过概念讲解建立序列化的理论基础；通过对比实验（有无ensure_ascii）强化中文处理意识；通过完整操作巩固文件读写与JSON的结合应用。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|--|---|--|
| 4. 日记数据导出 10分钟 | 指导学生编写日记导出功能：从数据库查询所有日记，将查询结果（元组列表）转换为字典列表，序列化为JSON格式并保存到文件，实现完整的数据备份。 | <p>任务分解 分解导出流程：①从数据库SELECT查询全部日记；②将元组结果转为字典格式（需字段名映射）；③调用json.dump()保存到文件；</p> <p>难点突破 演示元组转字典的处理：</p> <pre>cursor.execute('SELECT * FROM diary') rows = cursor.fetchall() diaries = [] for row in rows: diaries.append({ 'id': row[0], 'date': row[1], 'content': row[2] })</pre> <p>代码示范 演示完整的export_to_json()函数实现。</p> | <p>编码实践 编写导出函数，实现数据库→JSON文件的转换，注意添加时间戳到导出文件名（如diary_backup_20260107.json）；</p> <p>功能测试 导出日记数据，用文本编辑器或JSON工具打开查看格式是否正确、中文是否显示正常。</p> | 通过任务分解降低复杂功能的实现难度；通过难点突破解决数据库查询结果的格式转换问题；通过真实文件验证增强成功体验。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|---|---|---|
| 5. 日记数据导入 5分钟 | 指导学生编写日记导入功能：读取JSON备份文件，反序列化为Python对象，批量插入到数据库中，实现数据的恢复和迁移，测试跨设备数据共享。 | <p>流程设计 讲解导入流程：①读取JSON文件； ②json.load()反序列化；③遍历日记列表； ④批量INSERT到数据库；</p> <p>代码示范 演示 import_from_json()函数：</p> <pre>with open('backup.json', 'r', encoding='utf-8') as f: diaries = json.load(f) for diary in diaries: cursor.execute(' INSERT INTO diary ...')</pre> <p>异常处理 引导处理文件不存在、JSON格式错误等异常。</p> | <p>编码实现 编写导入函数，实现JSON文件→数据库的反向操作；</p> <p>恢复测试 删除数据库中的部分数据，使用导入功能从JSON备份恢复，验证数据完整性。</p> | 通过逆向操作巩固序列化/反序列化的双向理解；通过数据恢复测试体现备份功能的实用价值；通过异常处理培养健壮代码意识。 |
| 6. 系统整合与总结 2分钟 | 将导出和导入功能整合到菜单系统，总结三节课完成的完整数字日记本系统（文件→数据库→JSON），对比三种数据持久化方案的特点与适用场景，布置综合任务。 | <p>功能整合 指导在主菜单添加"备份到JSON"和"从JSON恢复"选项，展示完整系统的所有功能；</p> <p>知识总结 绘制三种方案对比表：文本文件（简单、不便查询）、数据库（高效、不便迁移）、JSON（通用、人类可读）；</p> <p>任务布置 课后任务：为JSON导出添加"导出选定日期范围"功能，支持部分数据导出。</p> | <p>系统展示 小组展示完整的数字日记本系统运行效果，演示备份→清空→恢复的完整流程；</p> <p>反思总结 总结三课时的技术进化路线，理解不同技术方案的取舍逻辑。</p> | 通过功能整合展现项目的完整性；通过方案对比建立技术选择的决策思维；通过系统演示增强学生的成就感和自信心。 |

教学成效与反思

| | |
|------|---|
| 教学成效 | <p>本课时成功将JSON数据处理知识置于"数据备份与跨平台迁移"的实用场景中，约80%的学生能够准确说明JSON格式的特点和序列化的作用。全体学生完成了日记数据的JSON导出和导入功能，其中75%以上能够正确处理中文编码和数据格式转换。通过"导出→删除→恢复"的完整测试流程，学生对数据备份的价值有了深刻认知。三课时的项目开发让学生完整体验了从简单文件到数据库再到标准数据格式的技术演进过程，建立了较为完整的数据持久化知识体系。小组展示环节学生能够清晰阐述每种方案的优劣，显示出较强的技术理解和表达能力。多数学生表示愿意继续使用并完善这个日记系统，学习迁移到生活实践的效果显著。</p> |
| 教学反思 | <p>本课时通过"跨平台数据共享"的真实需求成功建立了JSON学习的情境，学生对"为什么需要标准格式"有了清晰认知。Python数据结构与JSON的类型对应讲解清晰，降低了格式转换的理解难度。不足之处：①数据库查询结果（元组列表）转换为字典列表的代码逻辑对部分学生来说较为复杂，出现索引错误和字段映射混乱，后续应提供更详细的数据结构转换示意图；②对datetime等特殊类型的序列化问题只做了简单提及（将日期存为字符串），未深入讲解自定义JSON编码器，部分学生遇到复杂对象序列化时仍会困惑；③JSON格式错误的调试方法讲解不足，部分学生在手动编辑JSON时出现语法错误（如多余逗号、单引号等）但不知如何定位，应强化JSON验证工具的使用指导。改进方向：制作数据结构转换的可视化流程图；增加datetime序列化的拓展案例；提供常见JSON格式错误的排查清单。整体来看，三课时的项目设计逻辑清晰、技术递进合理，学生通过实战掌握了文件操作、数据库和JSON三大核心技术，综合能力得到全面提升，项目式教学的效果得到了充分验证。</p> |

天气数据自动采集与推送系统——HTTP通信与天气API数据获取 教学设计

| | |
|-----------|---|
| 课题 | 天气数据自动采集与推送系统——HTTP通信与天气API数据获取 |
| 课时 | 1课时(40分钟) |
| 教学目标 | 知识目标: 技能目标: 素养目标: |
| 教学 重难点 | 重点: 难点: |
| 教学资源准备 | 多媒体课件（含HTTP通信原理动画）；和风天气API密钥（或其他免费天气API）；示例API响应JSON数据；requests库安装包；项目代码模板文件。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------|---|--|--|---|
| 1. 项目任务导入 7分钟 | 展示项目最终目标（自动采集多城市天气并推送），明确本课时任务是实现项目的核心模块——数据采集功能，引出"如何让Python程序从互联网获取天气数据"这一技术问题。 | 场景创设 演示手机天气APP的实时更新效果，提问："这些数据从哪来？我们的Python程序如何获取？" 任务发布 展示项目架构图，标注本课时目标：编写weather_fetcher.py模块，实现"输入城市名→返回温度、天气"功能。 | 观察思考 观看演示，联系日常使用经验，思考数据来源； 明确目标 理解本课时在项目中的定位：数据采集是后续所有功能的基础。 | 通过真实应用场景建立技术需求认知；通过项目架构图让学生明确本课时任务在整体中的关键作用，强化目标导向。 |
| 2. 核心概念讲解 8分钟 | 讲解HTTP协议的客户端-服务器模型，介绍REST API的概念、特点与调用方式，展示天气API文档的关键内容（接口地址、参数、响应格式）。 | 原理阐述 用动画演示HTTP请求过程：Python程序（客户端）→发送请求→服务器→返回数据；强调"URL+参数"的请求构造方式； 文档解读 打开和风天气API文档，逐项讲解：接口URL、必需参数（key、location）、响应JSON结构，演示在浏览器中直接访问API。 | 聆听理解 理解HTTP的"请求-响应"机制，记录API调用的三要素：地址、参数、响应格式； 实例观察 观看浏览器访问API的演示，直观感受JSON数据的返回，尝试识别数据中的城市、温度字段。 | 将抽象的网络通信概念具象为"发送网址+参数，获得数据"的简单模型；通过浏览器演示消除技术神秘感，建立"API就是特殊网址"的认知。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|----------------------------|---|--|--|--|
| 3. 工具库与代码演示 10分钟 | 介绍requests库的作用与安装，演示使用requests.get()调用天气API的完整代码，讲解参数传递、响应处理、JSON解析的具体方法。 | <p>库的引入 说明requests是Python的HTTP客户端库，演示pip install requests安装过程；</p> <p>代码演示 边讲解边编写代码：</p> <pre>import requests url = "https://api.qweather.com/v7/weather/now" params = {"key": "你的密钥", "location": "101010100"} response = requests.get(url , params=params) data = response.json() print(f"温度： {data['now'] ['temp']}℃")</pre> <p>逐行解释：URL定义、参数字典、发送请求、解析JSON、字段访问；</p> <p>运行展示 运行代码，展示获取到的北京天气数据。</p> | <p>安装实践 跟随教师完成requests库的安装确认；</p> <p>代码跟随 边听边在自己的IDE中输入代码，理解每行的作用；</p> <p>结果验证 运行代码，观察控制台输出的天气数据，对照API文档验证字段。</p> | 通过完整代码演示建立"库调用-参数传递-数据解析"的操作流程；通过逐行讲解降低理解难度；通过即时运行验证建立"代码有效"的信心。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|----------------------------|--|--|---|--|
| 4. 功能模块编写 12分钟 | 指导学生编写完整的天气数据获取函数，封装为项目模块，要求实现：输入城市名、返回温度和天气状况、添加异常处理、输出格式化结果。 | 任务细化 发布编码任务：创建weather_fetcher.py，定义get_weather(city)函数，实现上述功能；提供城市代码对照表； 技术提示 提示关键点：使用字典存储多城市代码、用try-except捕获网络异常、格式化输出使用f-string； 巡回指导 巡视学生编码，解答JSON嵌套访问、异常处理等问题，强调代码注释与函数文档字符串的编写。 | 独立编码 创建py文件，定义函数框架，填充请求逻辑； 调试测试 测试不同城市（如"北京""上海"），验证数据准确性；尝试故意输入错误密钥，观察异常处理效果； 代码优化 根据测试结果调整代码，添加注释说明各部分功能。 | 通过独立编码巩固API调用技能，形成可复用的项目模块；通过测试培养"编码-调试-优化"的开发习惯；通过异常处理强化代码健壮性意识。 |
| 5. 成果展示与项目展望 3分钟 | 选取学生作品展示运行效果，总结HTTP通信与API调用的核心要点，预告后续课程将在此基础上实现多城市并发采集与数据存储。 | 作品点评 选2-3名学生演示代码，点评优点（如良好的异常处理）与改进点； 知识串联 总结：本课时完成了项目的"数据源"模块，后续将学习用多线程同时采集多城市、用数据库存储历史数据、用定时任务实现自动化； 任务布置 课后任务：为函数添加"获取未来3天天气"的功能（需查阅API文档）。 | 观摩学习 观看同学作品，学习不同实现思路； 总结反思 回顾requests使用流程，明确本模块在项目中的"基石"作用； 接收任务 记录课后任务，理解这是对API调用能力的深化练习。 | 通过作品展示建立学习榜样，强化成就感；通过项目展望保持学习连贯性，明确当前任务与后续模块的逻辑关系；课后任务引导学生自主探索API文档。 |

教学成效与反思

| | |
|------|---|
| 教学成效 | 结合本课时项目任务评估：85%以上学生成功调用API获取天气数据并正确解析，75%学生能够独立编写带异常处理的完整函数。通过"浏览器直接访问API"的演示，有效降低了学生对网络通信的畏难情绪。学生对requests库的"三行代码实现网络数据获取"表现出浓厚兴趣，课堂参与度高。项目数据采集模块的基础功能已基本实现，为后续多线程、数据存储等课时奠定了坚实基础。部分学生在调试JSON嵌套字段访问时遇到困难，但在教师指导下均得以解决。 |
| 教学反思 | 本课时成功将"HTTP通信与REST API"这一理论性较强的主题，转化为"获取天气数据"的具体项目任务，学生的代入感和学习动力明显增强。API文档解读环节设计合理，通过浏览器演示让抽象的"接口调用"变得直观可感。代码演示采用"边讲边写"方式，学生跟随效果好于预期。不足之处：①在40分钟内，部分学生因requests库安装出现网络问题导致进度延迟，建议提前让学生课下完成库安装或准备离线安装包；②对于JSON嵌套结构的访问（如data['now']['temp']），讲解略显仓促，少数学生理解不够深入，建议增加一个"JSON结构可视化"的辅助图示；③异常处理部分主要由教师演示，学生自主实践时间不足，后续可考虑设计专门的"错误处理"小任务强化训练。整体上，项目驱动的教学框架使技术学习目标清晰，学生编写的weather_fetcher.py模块已具备项目集成的条件，教学目标达成度较高。 |

天气数据自动采集与推送系统——多线程并发采集多城市数据 教学设计

| | |
|-----------|--|
| 课题 | 天气数据自动采集与推送系统——多线程并发采集多城市数据 |
| 课时 | 1课时(40分钟) |
| 教学目标 | 知识目标: 技能目标: 素养目标: |
| 教学 重难点 | 重点: 难点: |
| 教学资源准备 | 多媒体课件（含进程与线程对比图、并发执行动画）；上节课完成的weather_fetcher.py模块；多城市代码列表文件；性能测试脚本模板；线程池使用示例代码。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|--------------------------|---|---|--|---|
| 1. 性能瓶颈引入 6分钟 | 演示串行采集10个城市天气数据的耗时问题，提出"如何加速数据采集"的技术需求，引入多线程并发编程的概念。 | 问题演示 运行上节课的weather_fetcher模块，用循环串行采集10个城市，用time模块记录总耗时（约10-15秒），提问："如果要采集100个城市呢？" 概念引入 展示并发执行示意图，说明"同时发送多个请求"的思想，介绍线程是实现并发的一种方式。 | 观察分析 观看串行执行过程，感受耗时问题，理解在网络请求中"等待响应"的时间占主导； 讨论思考 小组讨论："能否让程序同时请求多个城市的数据？"初步建立并发需求认知。 | 通过实际运行建立性能痛点，让学生体会"串行等待"的低效；通过对比引出并发方案，明确本课时要解决的核心问题。 |
| 2. 多线程原理讲解 8分钟 | 讲解进程与线程的基本概念、区别与应用场景，介绍Python的threading模块，演示线程的创建、启动与join()方法的使用。 | 概念对比 用类比说明：进程是独立的程序实例（如打开的软件），线程是程序内部的执行路径（如软件的多个功能同时运行）；强调线程共享内存、切换快的特点； 基础演示 编写简单示例： <pre>import threading def task(name): print(f" {name}线程执行 中") t1 = threading.Thread (target=task, args=("A",)) t1.start() t1.join()</pre> 讲解Thread对象创建、target参数、start()启动、join()等待。 | 聆听理解 理解"并发≠并行"的概念，记录线程的三要素：目标函数、参数、启动方法； 代码跟随 在IDE中输入演示代码，运行观察线程执行效果，尝试创建多个线程并观察输出顺序的随机性。 | 通过类比降低抽象概念的理解难度；通过简单示例建立"创建线程-启动-等待"的基本操作流程，为后续复杂应用打基础。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|---|---|---|
| 3. 并发采集实现 12分钟 | 指导学生改造天气采集代码，使用线程实现多城市并发请求，演示ThreadPoolExecutor的使用方法，对比串行与并发的性能差异。 | <p>方案讲解 说明改造思路：为每个城市创建独立线程调用get_weather()，将结果存入列表；</p> <p>代码演示 演示手动创建线程版本，然后引入线程池简化代码：</p> <pre>from concurrent.futures import ThreadPoolExecutor cities = ["北京", "上海", "广州", ...] with ThreadPoolExecutor(max_workers=5) as executor: results = executor.map(get_weather, cities)</pre> <p>讲解线程池的优势：自动管理线程数量、简化代码；</p> <p>性能对比 分别运行串行版本和并发版本，展示耗时对比（如从15秒降至3秒）。</p> | <p>理解方案 理解"每个城市一个线程"的并发模型，记录线程池的使用模板；</p> <p>实践编码 创建multi_thread_fetcher.py，使用ThreadPoolExecutor改造采集代码；</p> <p>测试验证 运行代码，记录并对比串行与并发的耗时数据，体会性能提升。</p> | 通过代码改造巩固多线程应用能力；通过线程池简化实现，降低并发编程门槛；通过性能对比建立直观的"并发价值"认知。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|--------------------------|---|--|--|--|
| 4. 线程安全探讨 10分钟 | 讲解多线程中的数据竞争问题，演示不安全的共享数据访问，介绍线程锁(Lock)的使用方法，指导学生在项目中安全收集结果。 | <p>问题演示 演示多线程同时修改同一列表可能出现的问题（如数据丢失），说明这是"数据竞争"；</p> <p>解决方案 介绍threading.Lock()的使用：</p> <pre>lock = threading.Lock() with lock: shared_list.append(data)</pre> <p>强调在项目中应使用线程安全的数据结构或加锁保护；</p> <p>最佳实践 推荐使用线程池的map()或submit()方法返回结果，避免手动管理共享数据。</p> | <p>观察问题 观看演示，理解多线程同时访问数据的风险；</p> <p>学习方案 理解Lock的"互斥访问"机制，记录加锁的代码模板；</p> <p>优化代码 检查自己的并发采集代码，确保结果收集方式是线程安全的，如使用executor.map()的返回值而非手动append。</p> | 通过问题演示建立线程安全意识；通过解决方案教学培养规范编程习惯；通过代码优化强化"安全第一"的工程理念。 |
| 5. 项目集成与展望 4分钟 | 总结多线程并发编程的核心要点，将本课时的并发采集模块集成到项目中，预告后续课程将实现定时自动采集与数据存储。 | <p>知识总结 回顾threading模块使用流程、ThreadPoolExecutor优势、线程安全注意事项；</p> <p>模块集成 说明multi_thread_fetcher可作为项目的"高效采集引擎"，后续定时任务将调用此模块；</p> <p>任务布置 课后任务：尝试调整线程池的max_workers参数（如5、10、20），测试不同线程数对性能的影响，思考最优值。</p> | <p>回顾反思 总结并发编程的关键点，明确本模块在项目中的"性能优化"作用；</p> <p>接收任务 记录课后任务，理解这是对并发性能调优的探索性实验。</p> | 通过总结强化知识结构；通过模块集成保持项目连贯性；课后任务引导学生探索并发参数优化，培养实验精神。 |

教学成效与反思

| | |
|------|--|
| 教学成效 | 结合本课时项目任务评估：90%学生成功使用ThreadPoolExecutor实现多城市并发采集，80%学生能够通过性能测试数据说明并发的优势。通过串行与并发的直观对比（耗时从15秒降至3秒），学生对多线程价值有了深刻认知，学习积极性高。线程池的引入有效降低了并发编程复杂度，学生普遍反馈"比想象中简单"。项目的并发采集模块已具备实用性，为后续定时任务提供了高效的数据获取能力。线程安全环节虽有难度，但通过问题演示和最佳实践指导，学生建立了基本的安全意识。 |
| 教学反思 | 本课时成功将"多线程与并发编程"这一传统难点，转化为"提升天气采集速度"的具体优化任务，学生的学习动机强且目标明确。性能对比实验设计合理，数据直观有说服力。线程池的引入时机恰当，避免了过早陷入底层线程管理的复杂性。不足之处：①在40分钟内，线程安全部分讲解略显紧凑，部分学生对Lock的理解停留在"照搬代码"层面，建议后续增加一个专门的线程安全小案例演示；②对"I/O密集型 vs CPU密集型"的区分提及较少，少数学生可能误以为多线程适用所有场景，建议增加一句说明"多线程适合网络请求等待多的任务"；③课后任务的参数调优实验虽有探索性，但缺少对"过多线程可能导致性能下降"的提示，建议补充说明。整体上，项目驱动使并发编程学习变得具体可感，学生编写的并发模块已达到生产级代码的雏形，教学目标达成度高。 |

天气数据自动采集与推送系统——定时任务与自动化采集 教学设计

| | |
|-----------|--|
| 课题 | 天气数据自动采集与推送系统——定时任务与自动化采集 |
| 课时 | 1课时(40分钟) |
| 教学目标 | 知识目标: 技能目标: 素养目标: |
| 教学 重难点 | 重点: 难点: |
| 教学资源准备 | 多媒体课件（含定时任务场景案例、系统架构图）；前两节课完成的weather_fetcher.py和multi_thread_fetcher.py模块；schedule库安装包；日志配置示例代码；完整项目运行演示视频。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|--|---|--|
| 1. 自动化场景导入 6分钟 | 展示生活中的自动化案例（如定时提醒、自动备份），引出"让天气采集程序自动运行"的需求，明确本课时要实现的项目最终目标。 | 场景列举 列举自动化应用：闹钟、定时发送报表、服务器定时备份等，提问："我们的天气系统如何做到每小时自动采集？" 目标演示 播放完整项目运行视频：程序启动后自动每30分钟采集一次数据，保存到文件并输出日志，无需人工干预。 | 联系经验 回忆日常使用的自动化功能，理解"定时执行"的价值； 明确目标 观看演示，建立"让程序自己跑起来"的认知，理解本课时是项目的"收官之作"。 | 通过生活化案例建立自动化认知；通过完整演示展示项目最终形态，激发学生的成就欲望和完成项目的动力。 |
| 2. 定时任务原理 7分钟 | 讲解定时任务的实现机制，介绍Python的schedule库，演示基本的任务调度代码，说明"调度循环"的概念。 | 原理讲解 说明定时任务的本质：程序持续运行，定期检查是否到达执行时间；介绍schedule库是对时间检查逻辑的封装； 基础演示 编写简单示例： <pre>import schedule import time def job(): print("任务执行！") schedule.every(10).seconds.do(job) while True: schedule.run_pending() time.sleep(1)</pre> 逐行讲解：定义任务函数、设置调度规则、启动循环检查。 | 聆听理解 理解"持续检查+触发执行"的调度机制，记录schedule的核心API； 代码跟随 输入演示代码并运行，观察每10秒自动执行一次的效果，尝试修改间隔时间（如30秒、1分钟）。 | 通过原理讲解揭开定时任务的"神秘面纱"；通过简单示例建立"设置规则-启动循环"的操作模型，为项目集成打基础。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|---|---|---|
| 3. 项目模块集成 13分钟 | 指导学生创建主程序文件，整合前两节课的采集模块，实现"定时自动采集多城市天气并保存"的完整功能，添加时间戳与日志输出。 | <p>架构讲解 说明项目文件结构：weather_fetcher.py（单城市）、multi_thread_fetcher.py（并发）、main_scheduler.py（定时主程序）；</p> <p>集成演示 演示 main_scheduler.py 编写过程：</p> <pre>import schedule from multi_thread_fetcher import fetch_all_cities def auto_fetch(): data = fetch_all_cities() # 保存到文件，添加时间戳 print(f"[{time.strftime('%Y-%m-%d %H:%M:%S')}]] 采集完成") schedule.every(30).minutes.do(auto_fetch)</pre> <p>讲解数据保存方案（CSV或JSON）、时间戳格式化；</p> <p>运行测试 启动程序，观察首次执行和后续自动触发的效果。</p> | <p>理解架构 明确各模块的职责分工，理解 main_scheduler 是"指挥中心"；</p> <p>动手编码 创建 main_scheduler.py，导入并发采集模块，设置定时规则（如每30分钟），添加数据保存代码；</p> <p>测试验证 运行程序，确认定时执行正常，检查保存的数据文件格式与内容。</p> | 通过模块集成培养系统化编程思维；通过完整实现建立"小模块组成大系统"的认知；通过实际运行验证项目的实用性。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------|--|---|---|---|
| 4. 稳定性优化 10分钟 | 讲解长时间运行程序的常见问题，指导学生添加异常处理、日志记录、资源管理等代码，确保程序稳定可靠。 | <p>问题分析</p> <p>说明长期运行可能遇到的问题：网络故障、API限流、内存泄漏等；强调健壮性的重要性；</p> <p>优化方案</p> <p>演示添加try-except包裹任务函数、配置logging模块记录日志：</p> <pre>import logging logging.basicConfig(filename='weather.log', level=logging.INFO) def auto_fetch(): try: # 采集逻辑 logging.info("采集成功") except Exception as e: logging.error(f"采集失败: {e}")</pre> <p>讲解日志级别（INFO、ERROR）、日志文件的查看方法；</p> <p>最佳实践</p> <p>建议添加程序启动提示、优雅退出机制（Ctrl+C处理）。</p> | <p>认识风险</p> <p>理解"程序不能假设一切正常"的思想，记录异常处理的必要性；</p> <p>优化代码</p> <p>为auto_fetch()添加try-except，配置logging输出到文件，测试故意制造错误（如API密钥错误）观察日志记录；</p> <p>完善程序</p> <p>添加程序启动提示信息，如"天气采集系统已启动，每30分钟执行一次"。</p> | 通过问题分析培养风险意识；通过优化方案教学提升代码质量；通过日志实践建立生产级程序的开发习惯。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|--|--|--|
| 5. 项目总结与展望 4分钟 | 回顾整个项目的三个核心模块，总结从HTTP请求到多线程再到自动化的完整技术链条，展望后续可扩展的功能方向。 | 项目回顾 梳理项目实现路径：第1课时建立数据源→第2课时提升采集效率→第3课时实现自动化运行； 成果展示 请学生展示运行中的程序和保存的数据文件，分享项目完成的成就感； 扩展方向 提出可选的增强功能：数据可视化（matplotlib）、极端天气告警推送（邮件/短信）、Web界面展示等； 任务布置 课后任务：让程序在电脑上持续运行24小时，观察日志并统计采集成功率。 | 总结反思 回顾三次课的学习历程，理解"技术服务于需求"的思想； 展示成果 演示自己的运行程序，分享实现过程中的收获与困难； 展望未来 思考项目的实用场景和可改进方向，接收课后实验任务。 | 通过系统回顾强化知识体系；通过成果展示建立成就感与自信；通过扩展方向保持学习兴趣和探索欲望。 |

教学成效与反思

| | |
|------|---|
| 教学成效 | 结合本课时项目任务评估：95%学生成功实现定时自动采集功能，85%学生能够添加完整的异常处理与日志记录。通过三节课的连贯学习，学生完整经历了"单功能模块→性能优化→系统集成"的项目开发过程，对软件工程的模块化思想有了深刻体会。定时任务的实现让学生看到了"程序自己工作"的魅力，课堂兴奋度和成就感达到高峰。项目最终成果具有实用性，部分学生表示愿意持续运行并收集数据做后续分析，显示出强烈的学习内驱力。日志记录的引入提升了学生对"生产级代码"的认知。 |
| 教学反思 | 本课时成功将"定时任务与自动化"概念转化为"让天气系统自动运行"的具体目标，学生的学习动机和参与度极高。三节课形成的完整项目链条设计合理，每节课的模块都能无缝集成到最终系统中，验证了项目式教学的有效性。schedule库的简洁API降低了定时任务的学习门槛，学生普遍反馈"比想象中容易"。不足之处：①在40分钟内，对"程序后台运行"的操作系统层面知识（如nohup、系统服务等）未能涉及，部分学生询问"如何让程序关闭终端后仍运行"，建议作为课后拓展资料补充；②日志配置部分主要由教师演示，学生自主设计日志格式的时间不足，建议提供更灵活的日志模板让学生定制；③对"定时任务的精度限制"（如schedule是软实时）未做说明，可能导致学生对时间准确性有过高期待。整体上，项目式教学使三个看似独立的技术主题（HTTP、多线程、定时任务）自然融合为一个有机整体，学生在完成项目的过程中建立了系统化思维和工程实践能力，教学目标超预期达成。建议后续可将此项目作为综合实训案例，引导学生继续扩展功能（如添加Web界面、数据分析模块等）。 |

天气数据自动采集与推送系统——TCP推送服务器实现

教学设计

| | |
|--------|---|
| 课题 | 天气数据自动采集与推送系统——TCP推送服务器实现 |
| 课时 | 1课时(40分钟) |
| 教学目标 | 知识目标: 技能目标: 素养目标: |
| 教学重难点 | 重点: 难点: |
| 教学资源准备 | 多媒体课件（含TCP/IP协议栈图、三次握手动画）；前期完成的天气采集模块；socket编程示例代码；网络调试工具（如Telnet或自编客户端）；服务器-客户端通信演示视频。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|----------------------|---|---|--|--|
| 1. 推送需求引入 6分钟 | 展示项目新需求：将采集到的天气数据主动推送给其他设备（如手机、电脑客户端），引出"如何让不同设备通过网络获取数据"的技术问题。 | <p>场景创设</p> <p>演示场景：服务器端运行采集程序，客户端连接后实时接收最新天气数据；提问："HTTP只能客户端主动请求，如何实现服务器主动推送？"</p> <p>协议对比</p> <p>对比HTTP（请求-响应）与TCP（持续连接）的特点，说明TCP适合实时推送场景。</p> | <p>观察思考</p> <p>观看演示，理解"推送"与"请求"的区别，联系即时通讯软件的消息推送原理；</p> <p>需求明确</p> <p>理解本课时要实现的功能：搭建TCP服务器，让客户端连接后自动接收天气更新。</p> | 通过真实需求建立技术动机；通过协议对比帮助学生理解TCP的应用场景，明确本课时的技术选型理由。 |
| 2. TCP/IP原理讲解 8分钟 | 讲解TCP/IP协议的基本概念，介绍socket编程模型，说明IP地址、端口号、三次握手、数据流传输等核心知识。 | <p>协议讲解</p> <p>展示TCP/IP协议栈图（应用层-传输层-网络层），说明TCP在传输层提供可靠连接；用快递类比：IP地址是地址、端口号是门牌号；</p> <p>连接机制</p> <p>用动画演示TCP三次握手过程（SYN→SYN-ACK→ACK），强调"建立连接-传输数据-关闭连接"的流程；</p> <p>Socket概念</p> <p>介绍socket是操作系统提供的网络编程接口，Python通过socket模块封装了底层细节。</p> | <p>聆听理解</p> <p>理解TCP"面向连接、可靠传输"的特点，记录IP地址与端口号的作用；</p> <p>观看动画</p> <p>观察三次握手过程，理解"握手"是建立可靠连接的前提；</p> <p>概念记录</p> <p>记录socket编程的两个角色：服务器（监听等待）、客户端（主动连接）。</p> | 通过协议栈图建立分层网络模型认知；通过类比和动画降低抽象概念难度；通过socket概念引入为后续编程做准备。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|--|---|---|
| 3. 服务器端编写 12分钟 | 演示TCP服务器的完整代码实现，讲解socket创建、绑定、监听、接受连接、数据发送的每个步骤，指导学生编写天气推送服务器。 | <p>代码演示 边讲解边编写服务器代码：</p> <pre>import socket server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) server.bind(('0.0.0.0', 8888)) server.listen(5) print("服务器启动，等待连接...") client, addr = server.accept() print(f"客户端{addr}已连接") client.send("欢迎!".encode('utf-8')) client.close()</pre> | <p>代码跟随 在IDE中输入服务器代码，理解每个API的作用；</p> <p>功能实现 创建weather_server.py，集成之前的采集模块，实现"客户端连接→发送最新天气数据"功能；</p> <p>运行测试 启动服务器程序，观察"等待连接"提示，准备接受客户端连接。</p> | 通过完整代码演示建立服务器编程流程；通过逐行讲解降低socket API的理解难度；通过功能集成强化模块复用意识。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|----------------------------|--|---|--|---|
| 4. 客户端编写与测试 10分钟 | 指导学生编写简单的TCP客户端程序，实现连接服务器、接收数据、显示结果的功能，进行端到端的通信测试。 | <p>客户端演示 演示客户端代码：</p> <pre>import socket client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) client.connect(('127.0.0.1', 8888)) data = client.recv(1024) .decode('utf-8') print(f"收到数据: {data}") client.close()</pre> <p>讲解connect()连接、recv()接收、decode()解码；</p> <p>协同测试 引导学生两人一组：一人运行服务器、一人运行客户端，测试通信效果；</p> <p>异常处理 演示添加try-except处理连接失败、数据接收超时等异常。</p> | <p>编写客户端 创建weather_client.py，实现连接指定IP和端口、接收天气数据、格式化输出；</p> <p>协作测试 与同学配合测试，观察服务器日志与客户端输出，验证数据传输正确性；</p> <p>问题排查 遇到连接失败时，检查IP地址、端口号、防火墙设置等，培养网络调试能力。</p> | 通过客户端编写完善通信闭环；通过协作测试增强互动性与成就感；通过问题排查培养网络编程实战能力。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|--|---|--|---|
| 5. 功能优化与总结 4分钟 | 讨论实际应用中的改进方向（如支持多客户端、定时推送、消息协议设计），总结TCP编程的核心要点，布置课后任务。 | <p>优化讨论</p> <p>提出问题："当前代码只能服务一个客户端，如何支持多个？"引出多线程或异步方案预告；</p> <p>知识总结</p> <p>回顾socket编程的服务器端五步（创建-绑定-监听-接受-收发）、客户端三步（创建-连接-收发）；</p> <p>任务布置</p> <p>课后任务：改进服务器，使其能循环接受多个客户端连接，每个连接发送一次天气数据后关闭。</p> | <p>思考讨论</p> <p>思考多客户端场景的实现方案，联系之前学习的多线程知识；</p> <p>回顾反思</p> <p>总结TCP编程流程，明确socket API的调用顺序；</p> <p>接收任务</p> <p>记录课后任务，理解这是对服务器并发能力的初步探索。</p> | 通过优化讨论引发深度思考，为后续异步编程课程埋下伏笔；通过总结强化知识结构；课后任务培养独立解决问题能力。 |

教学成效与反思

| | |
|------|--|
| 教学成效 | 结合本课时项目任务评估：85%学生成功搭建TCP服务器并实现基本的天气数据推送，80%学生能够编写客户端程序完成端到端通信测试。通过两人一组的协作测试，学生对"网络通信需要两端配合"有了直观认知，课堂互动性强。socket编程的"创建-绑定-监听-接受"流程虽有一定难度，但通过逐行演示和代码跟随，大部分学生能够理解并复现。项目的推送功能模块已初步成型，为后续异步优化提供了基础。部分学生在处理网络异常（如端口占用、连接超时）时遇到困难，但在教师指导下均得以解决，网络调试能力得到锻炼。 |
| 教学反思 | 本课时成功将"TCP/IP网络编程"这一偏底层的主题，转化为"实现天气推送服务器"的具体项目任务，降低了学习门槛。三次握手的动画演示效果好，有效帮助学生理解TCP连接机制。服务器端和客户端代码的分步演示节奏合理，学生跟随效果良好。协作测试环节设计巧妙，让学生在实际通信中体会网络编程的乐趣。不足之处：①在40分钟内，对"字节流"与"消息边界"问题未能深入讲解，部分学生可能在后续处理长消息时遇到粘包问题，建议在演示代码中加入简单的消息长度前缀或分隔符说明；②对"0.0.0.0"与"127.0.0.1"的区别讲解不够清晰，少数学生在配置服务器IP时出现困惑，建议增加网络地址的补充说明；③多客户端支持只是讨论未实现，导致部分学生对"当前代码的局限性"理解不深，建议在课后任务中给出多线程改造的伪代码提示。整体上，项目驱动使抽象的网络编程概念变得具体可感，学生编写的推送服务器已具备基本功能，为项目增加了"分布式"特性，教学目标基本达成。 |

天气数据自动采集与推送系统——异步编程优化与协程调度 教学设计

| | |
|--------|---|
| 课题 | 天气数据自动采集与推送系统——异步编程优化与协程调度 |
| 课时 | 1课时(40分钟) |
| 教学目标 | 知识目标: 技能目标: 素养目标: |
| 教学重难点 | 重点: 难点: |
| 教学资源准备 | 多媒体课件（含同步vs异步对比动画、事件循环机制图）；前期完成的同步版天气采集代码；aiohttp库安装包；异步性能对比测试脚本；asyncio编程示例代码。 |

教学过程

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------|--|--|---|--|
| 1. 性能优化需求 6分钟 | 回顾多线程方案的局限性（GIL限制、线程开销），引出异步编程作为I/O密集型任务的更优解决方案，明确本课时要用异步优化项目性能。 | <p>问题引入</p> <p>回顾第2课时的多线程方案，提问："线程切换有开销，有没有更轻量的并发方式？"说明Python的GIL限制了多线程的CPU利用率；</p> <p>方案对比</p> <p>展示对比图：多线程（抢占式调度、线程开销大）vs 异步（协作式调度、轻量级协程），强调异步适合网络I/O场景。</p> | <p>回顾反思</p> <p>回忆多线程实现，思考其局限性，理解"等待网络响应时线程空闲"的资源浪费；</p> <p>需求明确</p> <p>理解本课时目标：用异步编程替代多线程，实现更高效的并发采集。</p> | 通过问题引入建立技术演进认知；通过方案对比帮助学生理解异步的适用场景，明确优化目标。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|------------------|--|---|--|--|
| 2. 异步编程原理 8分钟 | 讲解协程、事件循环、async/await的基本概念，演示简单的异步函数定义与调用，说明异步编程的执行模型。 | <p>概念讲解</p> <p>用餐厅类比：同步是服务员一桌服务完再服务下一桌，异步是服务员在等待上菜时去服务其他桌；说明协程是"可暂停的函数"，await是"暂停点"；</p> <p>语法演示</p> <p>演示async/await基础：</p> <pre>import asyncio async def hello(): print("开始") await asyncio.sleep(2) print("结束") asyncio.run(hello())</pre> <p>讲解async定义异步函数、await等待异步操作、asyncio.run()启动事件循环；</p> <p>原理说明</p> <p>展示事件循环图：遇到await时切换到其他协程，I/O完成后切回继续执行。</p> | <p>聆听理解</p> <p>理解"协作式调度"的思想，记录async/await是异步编程的核心语法；</p> <p>代码跟随</p> <p>输入演示代码并运行，观察异步函数的执行过程，尝试创建多个协程用asyncio.gather()并发运行；</p> <p>原理认知</p> <p>理解事件循环是异步任务的"调度器"，await是"主动让出控制权"的标记。</p> | 通过类比降低异步概念难度；通过简单示例建立async/await的基本使用模式；通过原理图帮助学生理解异步调度机制。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|---------------------|---|---|--|--|
| 3. 异步HTTP请求 12分钟 | 介绍aiohttp库，演示使用aiohttp实现异步天气数据获取，指导学生将同步的requests代码改造为异步版本。 | <p>库的引入 说明requests是同步库，aiohttp是异步HTTP客户端库，演示安装：pip install aiohttp；</p> <p>代码改造 对比演示同步与异步代码：</p> <pre># 同步版本 response = requests.get(url , params=params) data = response.json() # 异步版本 async with aiohttp.ClientSe ssion() as session: async with session.get(url, params=params) as response: data = await response.json()</pre> <p>讲解ClientSession管理连接池、async with上下文管理、await response.json()等待解析；</p> <p>并发实现 演示用 asyncio.gather()并发请求多城市：</p> <pre>tasks = [fetch_weather(c ity) for city in cities] results = await asyncio.gather(* tasks)</pre> | <p>对比学习 对比同步与异步代码的差异，理解"async with"和"await"的作用；</p> <p>功能改造 创建 async_weather_fetcher.py，定义异步的fetch_weather()函数，使用aiohttp实现；</p> <p>并发实现 编写主函数，用 asyncio.gather()并发请求10个城市，用time模块测量耗时；</p> <p>性能验证 运行代码，记录耗时数据，对比同步、多线程、异步三种方案的性能差异。</p> | 通过对比教学强化同步与异步的区别；通过代码改造巩固异步编程技能；通过性能测试建立"异步更快"的直观认知。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|--------------------------|--|---|--|---|
| | | 性能测试 运行异步版本，对比同步和多线程的耗时（如从15秒→3秒→1秒）。 | | |
| 4. 项目集成优化 10分钟 | 指导学生将异步采集模块集成到项目主程序，讨论异步定时任务的实现方式，处理异步代码中的异常与日志。 | 集成方案 说明如何在定时任务中调用异步函数： <pre>def job(): asyncio.run(async_fetch_all()) schedule.every(30).minutes.do(job)</pre> 或使用asyncio自带的定时功能； 异常处理 演示异步代码的try-except写法与asyncio.gather()的return_exceptions参数； 实践指导 引导学生改造main_scheduler.py，将并发采集改为异步版本，添加异常处理与日志。 | 理解方案 理解如何在同步代码中启动异步任务，记录asyncio.run()的桥梁作用； 代码优化 修改主程序，集成异步采集模块，测试定时任务是否正常触发； 完善细节 添加异步异常处理，确保单个城市请求失败不影响其他城市；测试异常情况（如断网）。 | 通过集成实践强化异步在真实项目中的应用；通过异常处理培养健壮性意识；通过完整测试验证优化效果。 |

| 教学环节 | 教学内容 | 教师活动 | 学生活动 | 设计意图 |
|-------------------|---|---|--|--|
| 5. 技术对比与总结 4分钟 | 总结同步、多线程、异步三种方案的适用场景，回顾整个项目的技术演进路径，展望异步编程的应用前景。 | 技术对比 总结三种方案：同步（简单但慢）、多线程（适合I/O密集但有开销）、异步（最高效但学习曲线陡）；强调选择技术要看场景； 项目回顾 梳理项目的技术升级：HTTP请求→多线程并发→异步优化，体现"迭代优化"的工程思维； 任务布置 课后任务：阅读asyncio官方文档，尝试用异步实现TCP服务器（ <code>asyncio.start_server</code> ）。 | 总结反思 对比三种方案的特点，理解"没有银弹"的技术选型原则； 项目总结 回顾整个学习历程，体会从基础功能到性能优化的完整开发流程； 接收任务 记录课后任务，理解异步编程的应用不限于HTTP请求。 | 通过技术对比培养技术选型能力；通过项目回顾强化系统化学习成果；课后任务拓展异步应用视野。 |

教学成效与反思

| | |
|------|--|
| 教学成效 | 结合本课时项目任务评估：80%学生成功使用 <code>async/await</code> 语法实现异步天气采集，75%学生能够用 <code>asyncio.gather()</code> 实现并发请求。通过性能对比实验（15秒→3秒→1秒），学生对异步编程的价值有了震撼性认知，学习热情高涨。异步编程虽是本课程难点，但通过餐厅服务员类比、同步异步代码对比、性能数据说话等多种教学手段，大部分学生能够理解核心概念并基本掌握。项目的最终版本已实现异步优化，整体性能提升显著。部分学生在理解"协程切换"与"事件循环"机制时有困难，但通过可视化图示和反复演示得以缓解。 |
| 教学反思 | 本课时成功将"异步编程"这一Python高级特性，转化为"极致优化天气采集性能"的具体任务，学习动机强烈。技术演进的叙事线索（同步→多线程→异步）设计合理，让学生理解"为什么需要异步"而非"为了学而学"。性能对比实验的说服力强，数据直观震撼。餐厅类比和事件循环可视化有效降低了抽象概念难度。不足之处：①在40分钟内，异步编程的深层机制（如事件循环的底层实现、协程调度细节）无法深入，部分学生可能停留在"会用但不深懂"的层面，建议提供补充阅读材料；②对"何时应该使用异步"的判断标准讲解不足，少数学生可能产生"异步万能"的误解，建议明确说明CPU密集型任务不适合异步；③ aiohttp 的连接池管理、超时设置等高级特性未涉及，实际生产应用还需补充；④课后任务的难度较大（异步TCP服务器），可能导致完成率低，建议降低难度或提供详细提示。整体上，异步编程课程作为项目的"性能巅峰"，有效展示了Python在高并发场景中的强大能力，五节课形成的完整技术栈（HTTP→多线程→定时→TCP→异步）具有很强的系统性和实战价值，教学目标超预期达成。建议在后续教学中，可将本项目作为综合实训案例的"标准版"，引导学生在此基础上扩展更多功能（如数据库持久化、Web界面、数据分析可视化等）。 |