揭阳职业技术学院

JIEYANG POLYTECHNIC

物联网 Python 应用开发

教案

主讲：钱德明

物联网教研室编写

二〇二六年二月十日

目录

第 1 章 课程介绍

1.1 教学内容

课程设置：介绍课程的性质、学时、先修课程、适用专业和教材。
课程的性质与任务：阐述课程在专业中的定位及其教学任务。
课程考核：说明考核方式、时间和各部分所占比例。
课程思政：强调物联网技术在国家发展中的重要性，以及 Python 开发的开源精神。
参考资料：提供教材和相关参考资料的详细信息。

1.2 教学目标

知识目标：让学生了解课程的性质、任务和考核方式。
能力目标：培养学生对课程的整体认识能力，为后续学习做好准备。
素质目标：通过课程思政，增强学生的责任感和使命感，树立正确的价值观。

1.3 教学重难点

重点：课程的性质与任务、考核方式以及课程思政内容。
难点：对课程的整体理解和认识。

1.4 课程思政设计

引入国家发展战略：介绍物联网技术在智慧城市建设、工业互联网等领域的应用，激发学生的民族自豪感和责任感。

强调团队合作：通过课程介绍，让学生认识到团队协作在物联网项目开发中的重要性，培养团队合作意识。

1.5 详细内容讲解

1.5.1 课程设置

课程性质：专业任选课
学时：64 学时
先修课程：C 程序设计、电子技术、STM32 原理及应用、无线网络应用及 Python 基础
适用专业：电子信息工程、物联网应用技术、自动化等专业
教材：《物联网 Python 开发实战》，安翔著，电子工业出版社，2018.3

1.5.2 课程的性质与任务

课程性质：本课程是自动化、测控、电子信息、机械电子、机器人工程、物联网应用技术等专业的专业任选课。

课程任务：通过本课程的学习，使学生熟悉 Python 语言开发物联网终端设备、网关设备、Web 后台程序的具体方法及优势，掌握利用 Python 进行编程，并利用 MicroPython 实现物联网从传感器到无线网络再到云端应用的全栈开发，从而提高学生的实验研究能力、系统设计能力、分析计算能力、总结归纳能力及自学新知识的能力。

1.5.3 课程考核

考核方式：课堂作业 40% + 期末考核作业 40% + 20%考勤
考核时间：课堂作业和期末考核作业在课程进行过程中完成，考勤贯穿整个学期

1.5.4 课程思政

物联网技术的重要性：介绍物联网技术在国家发展中的重要地位，如智慧城市建设、工业互联网等，强调学习物联网技术对于推动国家科技进步和产业升级的重要意义。

Python 语言的开源精神：介绍 Python 语言作为全球广泛使用的编程语言，其开源、免费的特点体现了国际合作与共享的精神，引导学生树立正确的价值观和国际视野。

1.5.5 参考资料

《物联网 Python 项目实战》，安欣，电子工业出版社，2018.3

HaaSEDUK1.https://haas.iot.aliyun.com/haasapi/index.html?spm=a2cpu.b17074837.0.0.11ab60b1Lh6jzb#/Python/docs/zh-CN/startup/HaaS_EDU_K1_startup

1.6 课堂实训项目

无

1.7 课后作业

查阅相关资料，了解物联网技术的最新发展趋势和应用场景，撰写一篇不少于 500 字的心得体会。

第 2 章 物联网与 Python

2.1 教学内容

介绍物联网的组成、架构、发展现状。

当前市面上典型的物联网应用方案。

阐述 Python 这门编程语言的特性和优点。

讲述用 Python 语言开发物联网终端设备、网关设备、Web 后台程序的具体方法及优势。

2.2 教学目标

了解物联网的基本概念、架构和发展现状。

熟悉 Python 语言在物联网开发中的应用优势。

掌握搭建 Python3 开发环境的方法。

2.3 教学重难点

重点：Python 语言开发物联网设备的具体方法及优势。

难点：Python3 开发环境的搭建和配置。

2.4 课程思政设计

增强民族自豪感：通过介绍我国在物联网领域的快速发展和取得的成就，增强学生的民族自豪感和自信心。

培养创新意识：对比国内外物联网应用方案，引导学生思考如何在借鉴国外先进经验的基础上，结合我国实际情况进行创新，培养学生的创新意识和国际竞争力意识。

2.5 详细内容讲解

2.5.1 物联网的组成与架构

感知层：负责数据采集，包括传感器、RFID 读写器、摄像头等设备。

网络层：负责数据传输，包括有线网络（如以太网）和无线网络（如 WiFi、蓝牙、ZigBee、LoRa、NB-IoT 等）。

平台层：负责数据管理和处理，包括云计算平台、大数据平台等。

应用层：提供具体的应用服务，如智能家居、智慧城市、工业物联网等。

2.5.2 物联网的发展现状

全球物联网市场：介绍全球物联网市场的规模和发展趋势，包括设备数量、数据量、应用场景等。

国内物联网市场：介绍国内物联网市场的规模和发展趋势，包括政策支持、技术创新、应用场景等。

2.5.3 Python 编程语言的特性和优点

简洁易读：Python 语言的语法简洁明了，易于学习和阅读。

功能强大：Python 语言拥有丰富的标准库和第三方库，可以满足各种开发需求。

跨平台：Python 语言可以在多种操作系统（如 Windows、Linux、macOS 等）上运行。

开源免费：Python 语言是开源的，可以免费使用和修改。

2.5.4 Python 在物联网开发中的应用

物联网终端设备开发：使用 Python 开发物联网终端设备，如传感器数据采集、设备控制等。

物联网网关设备开发：使用 Python 开发物联网网关设备，如数据传输、协议转换等。

物联网 Web 后台程序开发：使用 Python 开发物联网 Web 后台程序，如数据存储、用户管理、数据分析等。

2.5.5 Python3 环境搭建

Windows 平台：

打开浏览器，访问[Python 官方网站](https://www.python.org/downloads/windows/)。

下载最新版本的 Python 安装包（如 Python 3.9）。

运行安装程序，选择“Add Python 3.9 to PATH”选项，然后点击“Install Now”。

安装完成后，打开命令提示符，输入`python --version`和`pip --version`，验证 Python 和 pip 是否安装成功。

Linux 平台：

打开终端，运行以下命令下载 Python 源码：

```bash
wget https://www.python.org/ftp/python/3.9.x/Python-3.9.x.tgz
```

解压源码包：

```bash
tar -zxvf Python-3.9.x.tgz
cd Python-3.9.x
```

配置并编译安装：

```bash
./configure
```

```
make
sudo make install
```

验证安装：

```bash
python3 --version
pip3 --version
```

macOS 平台：

使用 Homebrew 安装 Python：

```bash
brew install python3
```

验证安装：

```bash
python3 --version
pip3 --version
```

2.6 课堂实训项目：搭建 Python3 开发环境

实施步骤

1. Windows 平台：

打开浏览器，访问[Python 官方网站](https://www.python.org/downloads/windows/)。

下载最新版本的 Python 安装包（如 Python 3.9）。

运行安装程序，选择“Add Python 3.9 to PATH”选项，然后点击“Install Now”。

安装完成后，打开命令提示符，输入`python --version`和`pip --version`，验证 Python 和 pip 是否安装成功。

2. Linux 平台：

打开终端，运行以下命令下载 Python 源码：

```bash
wget https://www.python.org/ftp/python/3.9.x/Python-3.9.x.tgz
```

解压源码包：

```bash
tar -zxvf Python-3.9.x.tgz
cd Python-3.9.x
```

配置并编译安装：

```bash
./configure
make
sudo make install
```

验证安装：

```bash
python3 --version
pip3 --version
```

3. macOS 平台：

使用 Homebrew 安装 Python：

```bash
brew install python3
```

验证安装：

```bash
python3 --version
pip3 --version
```

预期结果

学生能够成功安装 Python3，并验证版本号。

学生能够熟练使用命令行工具进行 Python 环境的安装和验证。

2.7 课后作业

1. 完成 Python3 开发环境的搭建，并撰写一篇关于 Python3 开发环境搭建过程的报告，包括遇到的问题和解决方法。

2. 查阅相关资料，了解 Python 在物联网领域的其他应用案例，并进行分析和总结。

第 3 章 Python 编程基础

3.1 教学内容

搭建 Python 编程环境。

Python 语言基础知识，包括数据类型、运算符、控制结构等。

掌握基本的 Python 编程技巧。

介绍 HaaS Python 及 MicroPython 标准库。

3.2 教学目标

掌握 Python 编程环境的搭建和配置。

熟悉 Python 语言的基本语法和编程技巧。

了解 HaaS Python 及 MicroPython 标准库的使用。

3.3 教学重难点

重点：Python 语言的基本语法和编程技巧。

难点：HaaS Python 及 MicroPython 标准库的理解和应用。

3.4 课程思政设计

培养团队合作精神：在搭建 Python 编程环境的过程中，强调团队协作的重要性，让学

生明白在编程开发中，团队成员之间的相互配合和沟通是项目成功的关键。

鼓励开源精神：介绍 Python 语言的开源社区文化，鼓励学生积极参与开源项目，为开源社区贡献自己的力量，培养学生的开源精神和共享意识。

3.5 详细内容讲解

3.5.1 Python 编程环境搭建

安装 Python 解释器：在不同操作系统（Windows、Linux、macOS）下安装 Python 解释器。

配置环境变量：将 Python 解释器的路径添加到系统环境变量中，方便在命令行中直接调用。

安装代码编辑器：推荐使用 VS Code、PyCharm 等代码编辑器，提高编程效率。

3.5.2 Python 语言基础知识

数据类型：

整数（int）：表示整数，如 1、2、3 等。

浮点数（float）：表示小数，如 1.1、2.2、3.3 等。

字符串（str）：表示文本，如"hello"、"world"等。

布尔值（bool）：表示真（True）或假（False）。

列表（list）：表示有序的集合，如[1, 2, 3]等。

元组（tuple）：表示有序的不可变集合，如(1, 2, 3)等。

字典（dict）：表示键值对的集合，如{"name": "Alice", "age": 25}等。

集合（set）：表示无序的集合，如{1, 2, 3}等。

运算符：

算术运算符：+、-、*、/、%、、//等。

比较运算符：==、!=、>、<、>=、<=等。

逻辑运算符：and、or、not 等。

赋值运算符：=、+=、-=、*=、/=等。

控制结构：

条件语句：if、elif、else 等。

循环语句：for、while 等。

跳转语句：break、continue、pass 等。

3.5.3 基本的 Python 编程技巧

函数定义：使用`def`关键字定义函数，如`def my_function():`。

模块导入：使用`import`关键字导入模块，如`import math`。

异常处理：使用`try`、`except`、`finally`等关键字处理异常，如`try: ... except Exception as e: ...`。

文件操作：使用`open`函数进行文件的读写操作，如`with open('file.txt', 'r') as f: ...`。

3.5.4 HaaS Python 及 MicroPython 标准库

HaaS Python：介绍 HaaS Python 的基本功能和应用场景，如设备控制、数据传输等。

MicroPython：介绍 MicroPython 的基本功能和应用场景，如嵌入式开发、物联网设备开发等。

标准库：介绍常用的 Python 标准库，如`sys`、`os`、`time`、`math`等。

3.6 课堂实训项目：Python 基础编程练习

实施步骤

1. 数据类型转换：

   编写一个 Python 程序，实现以下功能：

   将字符串`"123"`转换为整数。

   将整数`123`转换为字符串。

   将字符串`"3.14"`转换为浮点数。

   示例代码：

   ```python
   s = "123"
   i = int(s)
   print(i)
   f = float("3.14")
   print(f)
   ```

2. 控制结构：

   编写一个程序，输入一个数字，判断它是正数、负数还是零。

   示例代码：

   ```python
   num = float(input("Enter a number: "))
   if num > 0:
       print("Positive number")
   elif num == 0:
       print("Zero")
   else:
       print("Negative number")
   ```

3. 循环结构：

   编写一个程序，打印 1 到 10 的平方。

   示例代码：

   ```python
   for i in range(1, 11):
       print(f"{i} squared is {i2}")
   ```

预期结果

学生能够熟练使用 Python 的基本数据类型和控制结构。

理解输入输出、条件判断和循环的基本用法。

3.7 课后作业

1. 编写一个 Python 程序，实现一个简单的计算器功能，能够进行加、减、乘、除等基本运算操作。

2. 查阅相关资料，了解 HaaS Python 及 MicroPython 标准库的其他功能和应用场景，并进行分析和总结。

第 4 章 Python 数据结构

4.1 教学内容

掌握 Python 字符串、列表等数据结构。

掌握 Python 元组、字典、集合等特有的数据结构。

掌握 Python 高级特性，如生成器、迭代器、装饰器等。

熟练掌握和运用 Python 的各种数据结构是编写高质量 Python 程序的基础。

4.2 教学目标

熟练掌握 Python 的各种数据结构及其操作方法。

理解 Python 高级特性的原理和应用场景。

能够运用 Python 数据结构解决实际问题。

4.3 教学重难点

重点：Python 的各种数据结构及其操作方法。

难点：Python 高级特性的理解和应用。

4.4 课程思政设计

树立严谨的科学态度：通过讲解 Python 数据结构在实际项目中的应用，引导学生树立严谨的科学态度和数据意识，认识到数据的准确性和完整性对于物联网系统的重要性。

强调法律法规意识：强调在数据处理过程中要遵守法律法规，保护用户隐私，培养学生的法律意识和职业道德。

4.5 详细内容讲解

4.5.1 Python 字符串

字符串的创建：使用引号（' ' 或 " "）创建字符串，如` s = "hello" `。

字符串的访问：通过索引访问字符串中的字符，如` s[0] `。

字符串的切片：通过切片操作访问字符串的子串，如` s[1:3] `。

字符串的拼接：使用` + `运算符拼接字符串，如` s1 + s2 `。

字符串的格式化：使用` format `方法或 f-string 格式化字符串，如` f"Hello, {name}"`。

字符串的方法：介绍常用的字符串方法，如` upper() `、` lower() `、` find() `、` replace() `等。

4.5.2 Python 列表

列表的创建：使用方括号（[ ]）创建列表，如` lst = [1, 2, 3] `。

列表的访问：通过索引访问列表中的元素，如` lst[0] `。

列表的增删改查：使用` append() `、` insert() `、` remove() `、` pop() `等方法进行增删改查操作。

列表的遍历：使用` for `循环遍历列表，如` for item in lst: ... `。

列表的常用方法：介绍常用的列表方法，如` len() `、` max() `、` min() `、` sum() `等。

4.5.3 Python 元组

元组的创建：使用圆括号（( )）创建元组，如` tpl = (1, 2, 3) `。

元组的访问：通过索引访问元组中的元素，如` tpl[0] `。

元组的特点：元组是不可变的，不能进行增删改操作。

元组的常用方法：介绍常用的元组方法，如` len() `、` max() `、` min() `等。

4.5.4 Python 字典

字典的创建：使用花括号（{ }）创建字典，如` dct = {"name": "Alice", "age": 25} `。

字典的访问：通过键访问字典中的值，如` dct["name"] `。

字典的增删改查：使用` update() `、` pop() `、` del `等方法进行增删改查操作。

字典的遍历：使用` for `循环遍历字典，如` for key, value in dct.items(): ... `。

字典的常用方法：介绍常用的字典方法，如` keys() `、` values() `、` items() `、` get() `等。

4.5.5 Python 集合

集合的创建：使用花括号（{ }）或` set() `函数创建集合，如` s = {1, 2, 3} `或` s = set([1, 2, 3]) `。

集合的访问：通过迭代访问集合中的元素，如` for item in s: ... `。

集合的增删改查：使用` add() `、` remove() `、` discard() `、` pop() `等方法进行增删改查操作。

集合的运算：介绍集合的并集、交集、差集等运算，如` s1.union(s2) `、` s1.intersection(s2) `、` s1.difference(s2) `等。

4.5.6 Python 高级特性

生成器：使用生成器表达式或` yield `关键字创建生成器，如` gen = (x for x in range(10)) `。

迭代器：使用` iter() `函数创建迭代器，如` it = iter([1, 2, 3]) `。

装饰器：使用` @ `符号定义装饰器，如` @decorator def my_function(): `。

匿名函数：使用` lambda `关键字定义匿名函数，如` lambda x: x2 `。

4.6 课堂实训项目：实现购物车功能

实施步骤

1. 定义购物车类：

   创建一个购物车类` ShoppingCart `，包含以下功能：

   添加商品到购物车。

   删除购物车中的商品。

   显示购物车中的所有商品。

   计算购物车中商品的总价。

   示例代码：

   ```python
   class ShoppingCart:
       def __init__(self):
           self.items = []
       def add_item(self, item, price):
           self.items.append({"item": item, "price": price})
           print(f"Added {item} to cart.")
```

```
        def remove_item(self, item):
            for i, product in enumerate(self.items):
                if product["item"] == item:
                    del self.items[i]
                    print(f"Removed {item} from cart.")
                    break
        def show_cart(self):
            if not self.items:
                print("Your cart is empty.")
            else:
                for item in self.items:
                    print(f"{item['item']} ${item['price']}")
        def total_price(self):
            total = sum(item["price"] for item in self.items)
            print(f"Total price: ${total}")
```

2.测试购物车功能：

创建一个购物车实例，添加和删除商品，显示购物车内容和总价。

示例代码：

```python
cart = ShoppingCart()
cart.add_item("Apple", 1.5)
cart.add_item("Banana", 0.5)
cart.show_cart()
cart.total_price()
cart.remove_item("Apple")
cart.show_cart()
```

预期结果

学生能够熟练使用 Python 的列表和字典数据结构。

理解类和对象的基本概念，能够实现简单的功能模块。

4.7 课后作业

1.编写一个 Python 程序，使用列表实现一个简单的购物车功能，能够添加、删除、修改商品信息，并计算总价。

2.使用字典和集合，完成一个简单的数据统计和分析程序，能够对数据进行分类、汇总和排序等操作。

第 5 章 物联网核心组件及关键技术点

5.1 教学内容

介绍 WiFi、移动网络、ZigBee 通信、BLE、LoRa、NB-IoT 等网络通信方案。

介绍 HTTP、WebSocket、XMPP、CoAP、MQTT 等网络通信协议。

介绍常用的硬件设备种类，包括处理器、传感器、通信模块等。

介绍几种市面上常用的物联网云平台，如中国移动的 OneNet 平台、AWSIoT 平台、IBM 的 WatsonIoT 平台。

5.2 教学目标

了解物联网核心组件和关键技术点。

熟悉常用的网络通信方案和协议。

了解物联网云平台的基本功能和应用场景。

5.3 教学重难点

重点：常用的网络通信方案和协议。

难点：物联网云平台的使用和配置。

5.4 课程思政设计

激发爱国情怀：介绍我国在物联网核心组件和关键技术领域的自主研发成果，如国产芯片、传感器等，激发学生的爱国情怀和民族自豪感。

培养创新精神：引导学生思考如何在物联网技术发展中坚持自主创新，突破国外技术封锁，培养学生的创新精神和自主创新能力。

5.5 详细内容讲解

5.5.1 网络通信方案

WiFi：

特点：高速、短距离无线通信技术。

应用场景：智能家居、智能办公等。

配置方法：介绍如何配置 WiFi 模块，包括设置 SSID、密码、信道等。

移动网络：

特点：广域覆盖、高速通信。

应用场景：智能交通、工业物联网等。

配置方法：介绍如何配置移动网络模块，包括设置 APN、SIM 卡等。

ZigBee 通信：

特点：低功耗、短距离无线通信技术。

应用场景：智能家居、智能农业等。

配置方法：介绍如何配置 ZigBee 模块，包括设置 PAN ID、信道等。

BLE：

特点：低功耗、短距离无线通信技术。

应用场景：智能穿戴、智能家居等。

配置方法：介绍如何配置 BLE 模块，包括设置 UUID、服务等。

LoRa：

特点：低功耗、长距离无线通信技术。

应用场景：智能农业、智能城市等。

配置方法：介绍如何配置 LoRa 模块，包括设置频率、扩频因子等。

NB-IoT：

特点：低功耗、广域覆盖无线通信技术。

应用场景：智能城市、智能交通等。

配置方法：介绍如何配置 NB-IoT 模块，包括设置 APN、SIM 卡等。

5.5.2 网络通信协议

HTTP：

基本原理：介绍 HTTP 协议的基本原理，包括请求方法（GET、POST 等）、响应状态码等。

应用场景：介绍 HTTP 协议在物联网中的应用场景，如设备与服务器之间的数据传输。

使用方法：介绍如何使用 Python 的`requests`库进行 HTTP 请求和响应处理。

WebSocket：

基本原理：介绍 WebSocket 协议的基本原理，包括握手过程、数据帧格式等。

应用场景：介绍 WebSocket 协议在物联网中的应用场景，如实时数据传输。

使用方法：介绍如何使用 Python 的`websockets`库进行 WebSocket 通信。

XMPP：

基本原理：介绍 XMPP 协议的基本原理，包括 XML 流、消息传递等。

应用场景：介绍 XMPP 协议在物联网中的应用场景，如设备之间的消息传递。

使用方法：介绍如何使用 Python 的`sleekxmpp`库进行 XMPP 通信。

CoAP：

基本原理：介绍 CoAP 协议的基本原理，包括请求方法（GET、POST 等）、响应状态码等。

应用场景：介绍 CoAP 协议在物联网中的应用场景，如资源受限设备之间的数据传输。

使用方法：介绍如何使用 Python 的`aiocoap`库进行 CoAP 通信。

MQTT：

基本原理：介绍 MQTT 协议的基本原理，包括发布/订阅模式、主题等。

应用场景：介绍 MQTT 协议在物联网中的应用场景，如设备与服务器之间的数据传输。

使用方法：介绍如何使用 Python 的`paho-mqtt`库进行 MQTT 通信。

5.5.3 硬件设备

处理器：

类型：介绍常见的处理器类型，如 ARM、x86 等。

应用场景：介绍处理器在物联网中的应用场景，如设备控制、数据处理等。

传感器：

类型：介绍常见的传感器类型，如温度传感器、湿度传感器、光照传感器等。

应用场景：介绍传感器在物联网中的应用场景，如环境监测、设备控制等。

通信模块：

类型：介绍常见的通信模块类型，如 WiFi 模块、蓝牙模块、LoRa 模块等。

应用场景：介绍通信模块在物联网中的应用场景，如数据传输、设备互联等。

5.5.4 物联网云平台

OneNet 平台：

基本功能：介绍 OneNet 平台的基本功能，如设备管理、数据存储、数据分析等。

应用场景：介绍 OneNet 平台在物联网中的应用场景，如智能家居、智能城市等。

使用方法：介绍如何在 OneNet 平台上创建设备、配置参数、实现数据上报和云端控制。

AWSIoT 平台：

基本功能：介绍 AWSIoT 平台的基本功能，如设备管理、数据存储、数据分析等。

应用场景：介绍 AWSIoT 平台在物联网中的应用场景，如智能家居、智能城市等。

使用方法：介绍如何在 AWSIoT 平台上创建设备、配置参数、实现数据上报和云端控制。

WatsonIoT 平台：

基本功能：介绍 WatsonIoT 平台的基本功能，如设备管理、数据存储、数据分析等。

应用场景：介绍 WatsonIoT 平台在物联网中的应用场景，如智能家居、智能城市等。

使用方法：介绍如何在 WatsonIoT 平台上创建设备、配置参数、实现数据上报和云端控制。

5.6 课堂实训项目：基于 HTTP 协议的客户端和服务器程序

实施步骤

1. 服务器端程序：

使用 Python 的`http.server`模块创建一个简单的 HTTP 服务器。

示例代码：

```python
from http.server import BaseHTTPRequestHandler, HTTPServer

class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "text/plain")
        self.end_headers()
        self.wfile.write(b"Hello, world!")

def run_server():
    server_address = ("", 8000)
    httpd = HTTPServer(server_address, SimpleHTTPRequestHandler)
    print("Server running on port 8000...")
    httpd.serve_forever()

if __name__ == "__main__":
    run_server()
```

2. 客户端程序：

使用`requests`库发送 HTTP 请求到服务器。

示例代码：

```python
import requests
response = requests.get("http://localhost:8000")
print(response.text)
```

预期结果

学生能够实现一个简单的 HTTP 服务器和客户端程序。

理解 HTTP 协议的基本原理和 Python 中的网络编程方法。

5.7 课后作业

1. 查阅相关资料，了解其他物联网网络通信协议的特点和应用场景，并进行分析和总结。

2. 在物联网云平台上创建一个简单的物联网应用项目，并进行功能测试和优化。

第 6 章 构建物联网网关

6.1 教学内容

介绍网关的构成及常用物联网网关。

自主构建网关，包括交叉编译、编译引导程序、内核配置与编译、制作文件系统、分区与下载等步骤。

系统启动流程简介及移植 Python3。

使用树莓派作为网关，包括初次启动树莓派、串口登录树莓派、SSH 访问树莓派、升级 Python 版本等操作。

必备工具安装和板载 WiFi 配置及串口通信。

6.2 教学目标

了解网关的构成和作用。

掌握自主构建物联网网关的方法和步骤。

能够使用树莓派作为网关，并进行相关的配置和操作。

6.3 教学重难点

重点：自主构建物联网网关的方法和步骤。

难点：系统启动流程和 Python3 的移植。

6.4 课程思政设计

强调实践操作：在自主构建物联网网关的过程中，强调实践操作的重要性，鼓励学生勇于尝试、敢于创新，培养学生的实践能力和创新精神。

树立系统思维：介绍网关在物联网系统中的关键作用，引导学生树立系统思维和全局观念，认识到各个组件之间的协同工作对于整个物联网系统的重要性。

6.5 详细内容讲解

6.5.1 网关的构成及常用物联网网关

网关的构成：

硬件部分：包括处理器、存储设备（如 SD 卡、硬盘）、网络接口（如以太网、WiFi）、串口等。

软件部分：包括操作系统（如 Linux）、中间件（如消息队列、数据库）、应用程序（如数据处理、协议转换）。

常用物联网网关：

树莓派：低成本、高性能的单板计算机，适合用于物联网网关开发。

Arduino：适合用于简单的物联网项目，尤其是传感器数据采集和控制。
BeagleBone：功能强大，适合需要高性能处理的物联网项目。
商用网关：如华为、中兴等厂商的工业级网关，适用于复杂工业环境。

6.5.2 自主构建网关

交叉编译：

概念：在一种平台上编译出能在另一种平台上运行的代码。
工具：使用交叉编译工具链（如`arm-linux-gnueabihf-gcc`）。
步骤：

1. 安装交叉编译工具链：
   ```bash
   sudo apt-get install gcc-arm-linux-gnueabihf
   ```
2. 编写 C 代码并使用交叉编译工具编译：
   ```bash
   arm-linux-gnueabihf-gcc -o hello hello.c
   ```

编译引导程序（U-Boot）：

概念：引导程序负责初始化硬件并加载操作系统内核。
步骤：

1. 下载 U-Boot 源码：
   ```bash
   git clone https://github.com/u-boot/u-boot.git
   ```
2. 配置并编译 U-Boot：
   ```bash
   make rpi_4_defconfig
   make
   ```

内核配置与编译：

概念：Linux 内核是操作系统的核心，负责管理硬件资源和提供系统服务。
步骤：

1. 下载 Linux 内核源码：
   ```bash
   wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.10.tar.xz
   tar -xvf linux-5.10.tar.xz
   cd linux-5.10
   ```
2. 配置内核：
   ```bash
   make menuconfig
   ```
3. 编译内核：
   ```bash
   ```

```
make -j$(nproc)
```

制作文件系统：

概念：文件系统用于组织和存储数据。

步骤：

1. 创建文件系统镜像：

```bash
dd if=/dev/zero of=rootfs.img bs=1M count=1024
mkfs.ext4 rootfs.img
```

2. 挂载并安装必要的软件包：

```bash
sudo mount -o loop rootfs.img /mnt
sudo debootstrap --arch=armhf buster /mnt
http://raspbian.raspberrypi.org/raspbian/
sudo umount /mnt
```

分区与下载：

概念：将存储设备划分为多个分区，并将系统镜像写入设备。

步骤：

1. 使用`fdisk`工具分区：

```bash
sudo fdisk /dev/sdX
```

2. 使用`dd`工具写入系统镜像：

```bash
sudo dd if=rootfs.img of=/dev/sdX bs=4M
```

6.5.3 系统启动流程简介及移植 Python3

系统启动流程：

引导程序（U-Boot）：初始化硬件，加载内核。

内核启动：初始化系统核心功能，挂载根文件系统。

用户空间启动：运行`init`进程，启动用户服务。

移植 Python3：

概念：将 Python3 安装到目标系统中。

步骤：

1. 在目标系统上安装 Python3：

```bash
sudo apt-get install python3
```

2. 验证 Python3 安装：

```bash
python3 --version
```

```

6.5.4 使用树莓派作为网关

初次启动树莓派：

准备：插入 SD 卡，连接电源、显示器、键盘和鼠标。

启动：首次启动时，树莓派会自动配置网络并启动桌面环境。

串口登录树莓派：

配置：使用串口工具（如`minicom`、`screen`）连接树莓派。

登录：设置波特率为 115200，连接串口。

SSH 访问树莓派：

启用 SSH：在树莓派上运行以下命令：

```bash
sudo raspi-config
```

选择“Interfacing Options” > “SSH” > “Yes”。

连接：在其他设备上使用 SSH 连接：

```bash
ssh pi@<树莓派 IP 地址>
```

升级 Python 版本：

安装 Python3：

```bash
sudo apt-get update
sudo apt-get install python3
```

验证：

```bash
python3 --version
```

6.5.5 必备工具安装和板载 WiFi 配置及串口通信

必备工具安装：

安装工具：

```bash
sudo apt-get install vim git pip
```

板载 WiFi 配置：

连接 WiFi：

```bash
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

添加 WiFi 配置：

```plaintext
network={
    ssid="your_ssid"
    psk="your_password"
```
```

```
}
```
```

重启网络服务：

```bash
sudo systemctl restart networking.service
```

串口通信：

配置串口：

```bash
sudo nano /boot/config.txt
```

添加以下内容：

```plaintext
enable_uart=1
```

测试串口通信：

```bash
sudo apt-get install minicom
sudo minicom -D /dev/ttyS0
```

6.6 课堂实训项目：使用树莓派搭建物联网网关

实施步骤

1.准备硬件：

准备树莓派（如 Raspberry Pi 4）、电源适配器、SD 卡（至少 8GB）、HDMI 线、键盘和鼠标。

2.安装操作系统：

下载[Raspberry Pi OS](https://www.raspberrypi.org/software/operating-systems/)。

使用[Raspberry Pi Imager](https://www.raspberrypi.org/software/)将操作系统写入 SD 卡。

将 SD 卡插入树莓派，连接电源、显示器、键盘和鼠标，启动树莓派。

3.配置网络和 SSH：

在树莓派启动后，进入桌面环境，点击右上角的网络图标，连接到 WiFi 网络。

第 7 章 网关数据编码与处理

7.1 教学内容

网关读写 CSV 文件。

JSON 文件解析及其数据格式、编解码。

XML 简介，XML 操作与解析，XML 创建与修改，以及二进制数据的读写。

Base64 编解码。

正则表达式简介及`re`模块和贪婪匹配。
```

7.2 教学目标

掌握网关数据编码与处理的基本方法和实例。

熟悉 CSV、JSON、XML 等数据格式的读写和解析操作。

了解 Base64 编解码和正则表达式的应用。

7.3 教学重难点

重点：JSON 文件解析及其数据格式、编解码。

难点：XML 操作与解析，以及正则表达式的应用。

7.4 课程思政设计

树立标准化意识：通过讲解数据编码与处理的方法，引导学生树立标准化意识，认识到数据格式的统一和规范对于物联网数据共享和交换的重要性。

强调数据安全：强调在数据处理过程中要注重数据的安全性和可靠性，培养学生的安全意识和质量意识。

7.5 详细内容讲解

7.5.1 网关读写 CSV 文件

CSV 文件格式：以逗号分隔的文本文件，常用于存储表格数据。

读写操作：

读取 CSV 文件：

```python
import csv
with open('data.csv', mode='r', newline='', encoding='utf-8') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

写入 CSV 文件：

```python
data = [['Name', 'Age'], ['Alice', 24], ['Bob', 27]]
with open('data.csv', mode='w', newline='', encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerows(data)
```

7.5.2 JSON 文件解析及其数据格式、编解码

JSON 文件格式：一种轻量级的数据交换格式，易于人阅读和编写，同时也易于机器解析和生成。

解析操作：

加载 JSON 数据：

```python
import json
with open('data.json', 'r', encoding='utf-8') as file:
    data = json.load(file)
```

```
        print(data)
```

修改 JSON 数据：

```python
data['new_key'] = 'new_value'
```

保存 JSON 数据：

```python
with open('data.json', 'w', encoding='utf-8') as file:
    json.dump(data, file, ensure_ascii=False, indent=4)
```

7.5.3 XML 简介，XML 操作与解析，XML 创建与修改

XML 文件格式：一种标记语言，用于存储和传输数据。

操作与解析：

解析 XML 文件：

```python
import xml.etree.ElementTree as ET
tree = ET.parse('data.xml')
root = tree.getroot()
for child in root:
    print(child.tag, child.attrib)
```

创建 XML 文件：

```python
root = ET.Element('root')
child = ET.SubElement(root, 'child')
child.text = 'This is a child node.'
tree = ET.ElementTree(root)
tree.write('data.xml')
```

7.5.4 二进制数据的读写

二进制数据：以字节形式存储的数据，常用于处理图像、音频等非文本数据。

读写操作：

读取二进制文件：

```python
with open('image.png', 'rb') as file:
    data = file.read()
    print(data)
```

写入二进制文件：

```python
data = b'\x89PNG\r\n\x1a\n'
with open('image.png', 'wb') as file:
    file.write(data)
```

```

### 7.5.5 Base64 编解码

Base64 编码：一种将二进制数据转换为 ASCII 字符串的编码方式，常用于网络传输。

编解码操作：

编码：

```python
import base64
data = b'Hello, World!'
encoded_data = base64.b64encode(data)
print(encoded_data)
```

解码：

```python
decoded_data = base64.b64decode(encoded_data)
print(decoded_data)
```

7.5.6 正则表达式简介及`re`模块和贪婪匹配

正则表达式：一种用于匹配字符串中字符组合的模式。

操作方法：

匹配：

```python
import re
pattern = r'\d+'
text = 'There are 123 numbers in this sentence.'
match = re.search(pattern, text)
if match:
    print(match.group())
```

查找所有匹配项：

```python
matches = re.findall(pattern, text)
print(matches)
```

贪婪匹配与非贪婪匹配：

```python
greedy_pattern = r'a.*b'
non_greedy_pattern = r'a.*?b'
text = 'aababab'
print(re.findall(greedy_pattern, text))  # ['aababab']
print(re.findall(non_greedy_pattern, text))  # ['aab', 'ab', 'ab']
```

7.5.7 项目任务：实现由迭代器生成字符保存到文本文档

任务要求

利用 Python 编程实现生成 100 以内斐波那契数列，并将结果存入文本文档`foo.txt`，
```

每个数字字符后输出换行符。

实施步骤

1.生成斐波那契数列：

使用生成器实现斐波那契数列的生成。

示例代码：

```python
def fibonacci(n):
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a + b

fib_sequence = list(fibonacci(100))
```

2.将结果写入文件：

将生成的斐波那契数列写入`foo.txt`文件，每个数字占一行。

示例代码：

```python
with open('foo.txt', 'w', encoding='utf-8') as file:
    for number in fib_sequence:
        file.write(f"{number}\n")
```

3.验证结果：

打开`foo.txt`文件，检查内容是否符合要求。

示例代码：

```python
with open('foo.txt', 'r', encoding='utf-8') as file:
    print(file.read())
```

预期结果

在`foo.txt`文件中，按顺序存储了 100 以内的斐波那契数列，每个数字占一行。

7.6 课堂实训项目：数据处理与编码实践

实施步骤

1. CSV 文件处理：

创建一个 CSV 文件`data.csv`，包含以下内容：

```
Name,Age
Alice,24
Bob,27
```

编写 Python 程序读取该文件，并将所有人的年龄加 1 后保存到新的 CSV 文件`updated_data.csv`中。

示例代码：

```python
import csv

with open('data.csv', mode='r', newline='', encoding='utf-8') as file:
    reader = csv.reader(file)
    data = list(reader)

for row in data[1:]:
    row[1] = int(row[1]) + 1

with open('updated_data.csv', mode='w', newline='', encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerows(data)
```

2. JSON 文件处理：
   创建一个 JSON 文件`data.json`，包含以下内容：
   ```json
   {
       "name": "Alice",
       "age": 24
   }
   ```
   编写 Python 程序读取该文件，将年龄加 1 后保存到新的 JSON 文件`updated_data.json`中。
   示例代码：

```
import json

with open('data.json', 'r', encoding='utf-8') as file:
    data = json.load(file)

data['age'] += 1

with open('updated_data.json', 'w', encoding='utf-8') as file:
json.dump(data, file, ensure_ascii=False, indent=4)
```

3.XML 文件处理：

- 创建一个 XML 文件 data.xml，包含以下内容：
- <root>
- <person>
- <name>Alice</name>

- <age>24</age>
- </person>
- <person>
- <name>Bob</name>
- <age>27</age>
- </person>
- </root>
-

编写 Python 程序读取该文件，将所有人的年龄加 1 后保存到新的 XML 文件 updated_data.xml 中。

-
-

示例代码：

-
- import xml.etree.ElementTree as ET
-
- tree = ET.parse('data.xml')
- root = tree.getroot()
-
- for person in root.findall('person'):
- age = person.find('age')
- age.text = str(int(age.text) + 1)
-
- tree.write('updated_data.xml')

4.Base64 编码与解码：

-

编写 Python 程序，将字符串"Hello, World!"进行 Base64 编码，然后解码并打印结果。

-
-

示例代码：

-

```
import base64

data = b'Hello, World!'
encoded_data = base64.b64encode(data)
print(f"Encoded: {encoded_data}")
```

decoded_data = base64.b64decode(encoded_data)
print(f"Decoded: {decoded_data}")

5.正则表达式应用：

编写 Python 程序，使用正则表达式从以下文本中提取所有数字：

7.7 课后作业

任务一：完成课堂实训项目中的所有任务，并提交完整的代码和结果文件。

任务二：查阅相关资料，了解其他常见的数据编码方式（如 Hex 编码、URL 编码等），并编写一个 Python 程序，实现以下功能：

将字符串"Hello, World!"进行 Hex 编码和解码。

将字符串"https://example.com?query=python&lang=en"进行 URL 编码和解码。

提示：可以使用 Python 的 binascii 模块和 urllib.parse 模块。

任务三:编写一个 Python 程序,使用正则表达式从以下文本中提取所有日期格式(如 YYYY-MM-DD）：

plaintext 复制

```
Today is 2024-02-23. Yesterday was 2024-02-22. Tomorrow will be 2024-02-24.
```

提示：可以使用正则表达式 r'\d{4}-\d{2}-\d{2}'。

任务四：思考并总结在数据处理过程中，如何确保数据的安全性和完整性。撰写一篇不少于 300 字的总结报告，阐述你的观点和方法。

第 8 章 网关多进程与多线程

8.1 教学内容

`multiprocessing`模块及其进程同步和进程间通信。

多线程`threading`模块及其线程同步、线程间通信。

多核 CPU 利用率实验、GIL 全局锁、切换的开销，以及多线程与多进程的选择。

异步 IO 协程及其与多线程对比。

Python 中`asyncio`使任务异步运行。

8.2 教学目标

了解网关多进程与多线程的区别。

掌握多进程与多线程在 Python 中的应用方法。

理解异步 IO 协程的原理和应用场景。

8.3 教学重难点

重点：`multiprocessing`模块及其进程同步和进程间通信。

难点：多线程`threading`模块及其线程同步、线程间通信。

8.4 课程思政设计

引入并行计算思想：在讲解多进程与多线程的概念和应用时，引入并行计算和分布式计算的思想，引导学生思考如何通过技术手段提高物联网系统的性能和效率，培养学生的创新思维和优化意识。

强调资源合理分配：强调在多进程与多线程编程中要注重资源的合理分配和利用，避免资源浪费，培养学生的节约意识和环保意识。

8.5 详细内容讲解

8.5.1 `multiprocessing`模块及其进程同步和进程间通信

`multiprocessing`模块简介：`multiprocessing`模块支持子进程的生成、进程间的通信和同步。

进程同步：

使用`Lock`、`Event`、`Condition`等同步机制避免竞态条件和死锁。

示例代码：

```python
from multiprocessing import Process, Lock

def print_numbers(lock):
    for i in range(10):
        with lock:
            print(i)

if __name__ == "__main__":
    lock = Lock()
    p1 = Process(target=print_numbers, args=(lock,))
    p2 = Process(target=print_numbers, args=(lock,))
    p1.start()
    p2.start()
    p1.join()
    p2.join()
```

进程间通信：

使用`Queue`、`Pipe`、`Manager`等通信机制实现数据共享和传递。

示例代码：

```python
from multiprocessing import Process, Queue

def producer(queue):
    for i in range(5):
        queue.put(f"Item {i}")

def consumer(queue):
    while not queue.empty():
```

```
            item = queue.get()
            print(f"Consumed {item}")

    if __name__ == "__main__":
        queue = Queue()
        p1 = Process(target=producer, args=(queue,))
        p2 = Process(target=consumer, args=(queue,))
        p1.start()
        p2.start()
        p1.join()
        p2.join()
    ```
```

#### 8.5.2 多线程`threading`模块及其线程同步、线程间通信

`threading`模块简介：`threading`模块用于创建和管理线程。

线程同步：

使用`Lock`、`Event`、`Condition`等同步机制避免竞态条件和死锁。

示例代码：

```
```python
import threading

def print_numbers(lock):
    for i in range(10):
        with lock:
            print(i)

if __name__ == "__main__":
    lock = threading.Lock()
    t1 = threading.Thread(target=print_numbers, args=(lock,))
    t2 = threading.Thread(target=print_numbers, args=(lock,))
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```
```

线程间通信：

使用`Queue`实现线程间的数据传递。

示例代码：

```
```python
import threading
import queue

def producer(q):
    for i in range(5):
        q.put(f"Item {i}")
```
```
```

```
def consumer(q):
    while not q.empty():
        item = q.get()
        print(f"Consumed {item}")

if __name__ == "__main__":
    q = queue.Queue()
    t1 = threading.Thread(target=producer, args=(q,))
    t2 = threading.Thread(target=consumer, args=(q,))
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```

8.5.3 多核 CPU 利用率实验、GIL 全局锁、切换的开销

多核 CPU 利用率实验：

使用多进程和多线程分别计算 CPU 密集型任务（如计算斐波那契数列）的性能。

示例代码：

```python
import multiprocessing
import threading
import time

def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a

def worker(n):
    print(fibonacci(n))

if __name__ == "__main__":
    start_time = time.time()
    processes = [multiprocessing.Process(target=worker, args=(100000,))
for _ in range(4)]
    for p in processes:
        p.start()
    for p in processes:
        p.join()
    print(f"Multiprocessing time: {time.time() start_time}")

    start_time = time.time()
    threads = [threading.Thread(target=worker, args=(100000,)) for _ in
```

range(4)]
            for t in threads:
                t.start()
            for t in threads:
                t.join()
            print(f"Threading time: {time.time() start_time}")
        ```

    GIL 全局锁：
      Python 的全局解释器锁（GIL）限制了同一时刻只有一个线程执行 Python 字节码。
      解决方法：使用多进程或异步 IO。
    切换的开销：
      进程切换和线程切换的开销较大，合理选择并发模型可以提高性能。

### 8.5.4 异步 IO 协程及其与多线程对比

    异步 IO 协程简介：
      异步 IO 协程用于处理 IO 密集型任务，提高程序的并发能力。
      示例代码：
        ```python
        import asyncio

        async def fetch_data():
            print("Start fetching")
            await asyncio.sleep(2)
            print("Done fetching")
            return {"data": 123}

        async def main():
            result = await fetch_data()
            print(result)

        asyncio.run(main())
        ```
    与多线程对比：
      多线程适合 CPU 密集型任务，但受 GIL 限制。
      异步 IO 协程适合 IO 密集型任务，性能更高。

8.5.5 Python 中`asyncio`使任务异步运行

    `asyncio`模块简介：`asyncio`模块提供了异步 IO 的支持，用于编写单线程并发代码。
    异步任务调度：
      示例代码：
        ```python
        import asyncio

        async def task1():
            print("Task 1 started")

```
        await asyncio.sleep(1)
        print("Task 1 done")

    async def task2():
        print("Task 2 started")
        await asyncio.sleep(2)
        print("Task 2 done")

    async def main():
        await asyncio.gather(task1(), task2())

    asyncio.run(main())
    ```
```

8.6 课堂实训项目：多进程与多线程编程实践

实施步骤

1.多进程程序

目标：使用`multiprocessing`模块创建多个进程，分别计算1到100的平方和。

实现代码：

```python
import multiprocessing

def calculate_sum(start, end, result, index):
    total = sum(i2 for i in range(start, end + 1))
    result[index] = total

if __name__ == "__main__":
    num_processes = 4
    total_range = 100
    chunk_size = total_range // num_processes

    result = multiprocessing.Array('i', num_processes)
    processes = []

    for i in range(num_processes):
        start = i * chunk_size + 1
        end = (i + 1) * chunk_size if i != num_processes 1 else total_range
        p = multiprocessing.Process(target=calculate_sum, args=(start, end,
result, i))
        processes.append(p)
        p.start()

    for p in processes:
        p.join()
```
```

```
    total_sum = sum(result)
    print(f"Total sum of squares: {total_sum}")
```

预期结果：
程序将 1 到 100 的范围分为 4 个子范围，每个子范围由一个进程计算平方和。
最终输出所有子范围平方和的总和。
2.多线程程序
目标：使用`threading`模块创建多个线程，分别计算 1 到 100 的平方和。

实现代码：
```python
import threading

def calculate_sum(start, end, result, index):
    total = sum(i2 for i in range(start, end + 1))
    result[index] = total

def main():
    num_threads = 4
    total_range = 100
    chunk_size = total_range // num_threads

    result = [0] * num_threads
    threads = []

    for i in range(num_threads):
        start = i * chunk_size + 1
        end = (i + 1) * chunk_size if i != num_threads 1 else total_range
        t = threading.Thread(target=calculate_sum, args=(start, end, result,
i))
        threads.append(t)
        t.start()

    for t in threads:
        t.join()

    total_sum = sum(result)
    print(f"Total sum of squares: {total_sum}")

if __name__ == "__main__":
    main()
```

预期结果：

程序将 1 到 100 的范围分为 4 个子范围，每个子范围由一个线程计算平方和。

最终输出所有子范围平方和的总和。

对比分析

性能对比：

多进程：由于 Python 的全局解释器锁（GIL），多进程可以更好地利用多核 CPU 进行计算密集型任务。

多线程：多线程在处理 IO 密集型任务时表现更好，但在计算密集型任务中受限于 GIL。

资源消耗：

多进程：每个进程都有自己的内存空间，资源消耗较大。

多线程：线程共享同一进程的内存空间，资源消耗较小。

预期结果

学生能够理解多进程和多线程在 Python 中的应用方法。

学生能够通过实验对比多进程和多线程的性能差异。

学生能够掌握如何在 Python 中使用`multiprocessing`和`threading`模块实现并发编程。

第 9 章 网关数据持久化

9.1 教学内容

文件操作即读写文本数据、操作文件和目录、读写压缩文件、内存映射及临时文件与目录。

序列化 Python 对象及 SQLite 数据库命令操作、数据库和表的创建。

SQLite 语句。

SQLite 的 Python 编程。

9.2 教学目标

掌握 Python 文件的操作及 SQLite 的 Python 编程。

理解数据持久化的概念和重要性。

能够运用文件操作和数据库操作实现数据的持久化存储。

9.3 教学重难点

重点：文件操作及 SQLite 数据库命令操作。

难点：SQLite 的 Python 编程和数据库的设计与优化。

9.4 课程思政设计

树立数据备份意识:通过讲解数据持久化的方法,引导学生树立数据备份和恢复的意识,认识到数据持久化对于物联网系统长期稳定运行的重要性。

强调数据安全：强调在数据持久化过程中要注重数据的安全性和完整性，培养学生的安全意识和质量意识。

9.5 详细内容讲解

9.5.1 文件操作

读写文本数据：

读取文本文件：

```python
with open('example.txt', 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
```

写入文本文件：

```python
with open('example.txt', 'w', encoding='utf-8') as file:
    file.write("Hello, World!")
```

操作文件和目录：

创建目录：

```python
import os
os.mkdir('new_directory')
```

删除目录：

```python
os.rmdir('new_directory')
```

列出目录内容：

```python
print(os.listdir('.'))
```

重命名文件：

```python
os.rename('old_name.txt', 'new_name.txt')
```

读写压缩文件：

读取压缩文件：

```python
import gzip
with gzip.open('example.gz', 'rt', encoding='utf-8') as file:
    content = file.read()
    print(content)
```

写入压缩文件：

```python
with gzip.open('example.gz', 'wt', encoding='utf-8') as file:
```

```
        file.write("Hello, World!")
    ```

内存映射及临时文件与目录：
  内存映射文件：
    ```python
    import mmap
    with open('example.txt', 'r+b') as file:
        mm = mmap.mmap(file.fileno(), 0)
        print(mm.readline())  # 读取第一行
        mm.close()
    ```
  创建临时文件：
    ```python
    import tempfile
    with tempfile.NamedTemporaryFile(delete=False) as temp:
        temp.write(b"Hello, World!")
        print(temp.name)  # 打印临时文件名
    ```

9.5.2 序列化 Python 对象及 SQLite 数据库命令操作、数据库和表的创建

序列化 Python 对象：
  使用`pickle`模块：
    ```python
    import pickle

    data = {'key': 'value'}
    with open('data.pkl', 'wb') as file:
        pickle.dump(data, file)

    with open('data.pkl', 'rb') as file:
        loaded_data = pickle.load(file)
        print(loaded_data)
    ```

SQLite 数据库命令操作：
  创建数据库和表：
    ```python
    import sqlite3

    conn = sqlite3.connect('example.db')
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY,
```

```
            name TEXT NOT NULL,
            age INTEGER NOT NULL
        )
    ''')
    conn.commit()
    conn.close()
    ```
```

### 9.5.3 SQLite 语句

数据定义语句：

创建表：

```sql
CREATE TABLE users (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    age INTEGER NOT NULL
);
```

修改表：

```sql
ALTER TABLE users ADD COLUMN email TEXT;
```

删除表：

```sql
DROP TABLE users;
```

数据操纵语句：

插入数据：

```sql
INSERT INTO users (name, age, email) VALUES ('Alice', 25, 'alice@example.com');
```

更新数据：

```sql
UPDATE users SET age = 26 WHERE name = 'Alice';
```

删除数据：

```sql
DELETE FROM users WHERE name = 'Alice';
```

数据查询语句：

查询数据：

```sql
```
```

```sql
SELECT * FROM users;
```

条件查询：

```sql
SELECT * FROM users WHERE age > 25;
```

排序查询：

```sql
SELECT * FROM users ORDER BY age DESC;
```

9.5.4 SQLite 的 Python 编程

连接数据库：

```python
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
```

执行 SQL 语句：

```python
cursor.execute('SELECT * FROM users')
rows = cursor.fetchall()
for row in rows:
    print(row)
```

处理查询结果：

```python
cursor.execute('SELECT * FROM users WHERE age > ?', (25,))
rows = cursor.fetchall()
for row in rows:
    print(row)
```

事务处理：

```python
try:
    cursor.execute('INSERT INTO users (name, age) VALUES (?, ?)', ('Bob',
30))
    conn.commit()
except sqlite3.Error as e:
    print(f"An error occurred: {e}")
    conn.rollback()
finally:
```

```
conn.close()
```

9.6 课堂实训项目：实现数据持久化系统

实施步骤

1. 文件操作：

   创建一个 Python 程序，实现以下功能：

   创建一个文本文件`data.txt`，写入一些文本数据。

   读取`data.txt`文件的内容并打印。

   示例代码：

   ```python
   with open('data.txt', 'w', encoding='utf-8') as file:
       file.write("Hello, World!")

   with open('data.txt', 'r', encoding='utf-8') as file:
       print(file.read())
   ```

2. SQLite 数据库操作：

   创建一个 SQLite 数据库`example.db`，并在其中创建一个表`users`。

   插入一些用户数据，并查询所有用户数据。

   示例代码：

   ```python
   import sqlite3
   conn = sqlite3.connect('example.db')
   cursor = conn.cursor()
   cursor.execute('''
       CREATE TABLE IF NOT EXISTS users (
           id INTEGER PRIMARY KEY,
           name TEXT NOT NULL,
           age INTEGER NOT NULL
       )
   ''')
   cursor.execute('INSERT INTO users (name, age) VALUES (?, ?)', ('Alice',
25))
   cursor.execute('INSERT INTO users (name, age) VALUES (?, ?)', ('Bob', 30))
   conn.commit()

   cursor.execute('SELECT * FROM users')
   rows = cursor.fetchall()
   for row in rows:
       print(row)

   conn.close()
   ```

预期结果

学生能够熟练使用 Python 进行文件操作和 SQLite 数据库操作。

学生能够理解数据持久化的概念和重要性。

学生能够实现一个简单的数据持久化系统，能够对数据进行存储、查询和管理。

9.7 课后作业

1.任务一：完成课堂实训项目中的所有任务，并提交完整的代码和结果文件。

2.任务二：查阅相关资料，了解其他数据库（如 MySQL、PostgreSQL）的基本操作方法，并与 SQLite 进行对比分析。

3.任务三：编写一个 Python 程序，使用 SQLite 数据库实现一个简单的用户管理系统，支持用户注册、登录、查询用户信息等功能。

第 10 章 Python 扩展

10.1 教学内容

Python 扩展的原因。

连接硬件的纽带。

C 语言扩展 Python 及其传递整型参数和传递字符串参数。

10.2 教学目标

了解 Python 扩展的原因和方法。

掌握 C 语言扩展 Python 的基本方法和技巧。

能够运用 Python 扩展实现与硬件的连接和交互。

10.3 教学重难点

重点：C 语言扩展 Python 及其传递整型参数和传递字符串参数。

难点：Python 扩展在硬件连接中的应用和优化。

10.4 课程思政设计

强调灵活应变：在讲解 Python 扩展的原因和方法时，强调扩展在满足物联网系统多样化需求中的重要作用，引导学生树立灵活应变的思维，培养学生的创新能力和解决问题的能力。

激发探索欲望：介绍 Python 扩展在连接硬件、实现物联网功能等方面的应用，激发学生对物联网技术的兴趣和探索欲望。

10.5 详细内容讲解

10.5.1 Python 扩展的原因

性能优化：Python 是一种解释型语言，运行速度相对较慢。对于性能要求较高的计算任务，使用 C 语言编写扩展模块可以显著提高执行效率。

硬件访问：Python 本身无法直接访问底层硬件资源，通过 C 语言扩展可以实现对硬件设备的直接操作。

功能扩展：某些功能（如特定的算法、协议）在 Python 中难以实现或实现效率较低，可以通过 C 语言扩展来实现。

10.5.2 连接硬件的纽带

硬件接口：介绍常见的硬件接口，如 GPIO、I2C、SPI 等。

驱动程序：讲解如何编写硬件驱动程序，实现 Python 与硬件的交互。

示例：使用 Python 扩展模块控制 GPIO 引脚。

```python
C 语言扩展模块示例 (gpio_module.c)
#include <Python.h>
#include <wiringPi.h>

static PyObject* gpio_export(PyObject* self, PyObject* args) {
    int pin;
    if (!PyArg_ParseTuple(args, "i", &pin)) {
        return NULL;
    }
    pinMode(pin, OUTPUT);
    return Py_BuildValue("i", 0);
}

static PyMethodDef GpioMethods[] = {
    {"export", gpio_export, METH_VARARGS, "Export GPIO pin."},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef gpimodule = {
    PyModuleDef_HEAD_INIT,
    "gpio",
    "GPIO control module",
    -1,
    GpioMethods
};

PyMODINIT_FUNC PyInit_gpio(void) {
    PyObject* m;
    m = PyModule_Create(&gpimodule);
    if (m == NULL) {
        return NULL;
    }
    return m;
}
```

10.5.3 C 语言扩展 Python 及其传递整型参数和传递字符串参数

C 语言扩展 Python 的基本方法：

创建扩展模块：使用 Python 的`distutils`或`setuptools`工具创建扩展模块。

定义函数：在 C 语言中定义函数，并使用`PyArg_ParseTuple`解析 Python 传递的参数。

注册函数：将 C 语言函数注册到 Python 模块中。

传递整型参数：

示例代码：

```python
# Python 调用 C 语言扩展模块
import gpio
gpio.export(17)  # 控制 GPIO 17 引脚
```

传递字符串参数：

C 语言扩展模块示例：

```c
static PyObject* gpio_set_name(PyObject* self, PyObject* args) {
    const char* name;
    if (!PyArg_ParseTuple(args, "s", &name)) {
        return NULL;
    }
    printf("GPIO name: %s\n", name);
    return Py_BuildValue("s", "OK");
}

static PyMethodDef GpioMethods[] = {
    {"export", gpio_export, METH_VARARGS, "Export GPIO pin."},
    {"set_name", gpio_set_name, METH_VARARGS, "Set GPIO name."},
    {NULL, NULL, 0, NULL}
};
```

Python 调用：

```python
gpio.set_name("GPIO_17")
```

10.6 课堂实训项目：使用 C 语言扩展实现硬件控制

实施步骤

1.准备环境：

安装 Python 开发头文件和工具链：

```bash
sudo apt-get install python3-dev
sudo apt-get install build-essential
```

2. 编写 C 语言扩展模块：

创建一个 C 文件`gpio_module.c`，实现 GPIO 控制功能。

示例代码：

```c
#include <Python.h>
#include <wiringPi.h>

static PyObject* gpio_export(PyObject* self, PyObject* args) {
    int pin;
    if (!PyArg_ParseTuple(args, "i", &pin)) {
        return NULL;
    }
    pinMode(pin, OUTPUT);
    return Py_BuildValue("i", 0);
}

static PyObject* gpio_set_name(PyObject* self, PyObject* args) {
    const char* name;
    if (!PyArg_ParseTuple(args, "s", &name)) {
        return NULL;
    }
    printf("GPIO name: %s\n", name);
    return Py_BuildValue("s", "OK");
}

static PyMethodDef GpioMethods[] = {
    {"export", gpio_export, METH_VARARGS, "Export GPIO pin."},
    {"set_name", gpio_set_name, METH_VARARGS, "Set GPIO name."},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef gpimodule = {
    PyModuleDef_HEAD_INIT,
    "gpio",
    "GPIO control module",
    -1,
    GpioMethods
};

PyMODINIT_FUNC PyInit_gpio(void) {
    PyObject* m;
    m = PyModule_Create(&gpimodule);
    if (m == NULL) {
        return NULL;
    }
    return m;
```

```
}
```
```

3. 编译扩展模块：
   创建`setup.py`文件：
   ```python
   from setuptools import setup, Extension

   gpio_module = Extension('gpio', sources=['gpio_module.c'])

   setup(
       name='gpio',
       version='1.0',
       description='GPIO control module',
       ext_modules=[gpio_module]
   )
   ```
   编译并安装模块：
   ```bash
   python3 setup.py build
   sudo python3 setup.py install
   ```

4. 测试扩展模块：
   编写 Python 脚本测试扩展模块：
   ```python
   import gpio
   gpio.export(17)
   gpio.set_name("GPIO_17")
   ```

预期结果
学生能够成功编写并编译 C 语言扩展模块。
学生能够通过 Python 调用 C 语言扩展模块实现硬件控制功能。

10.7 课后作业

1. 任务一：完成课堂实训项目中的所有任务，并提交完整的代码和结果文件。
2. 任务二：查阅相关资料，了解其他语言（如 C++）扩展 Python 的方法，并与 C 语言扩展进行对比分析。
3. 任务三：编写一个 Python 程序，使用 C 语言扩展实现一个简单的数学计算功能，如计算平方根或阶乘。
   ```

第 11 章 网关网络编程

11.1 教学内容

网关网络通信方案。

Socket 编程。

`requests`上传文件及下载文件。

`hbmqtt`的安装及命令操作和 API 编程。

LoRa 网络通信：模块初始化及数据监听与接收、数据缓存。

11.2 教学目标

掌握 Socket 编程及其网络通信应用技能。

了解`requests`和`hbmqtt`的使用方法和应用场景。

理解 LoRa 网络通信的原理和实现方法。

11.3 教学重难点

重点：Socket 编程及其网络通信应用技能。

难点：`hbmqtt`的安装及命令操作和 API 编程。

11.4 课程思政设计

树立网络意识：通过讲解网络编程的基本概念和应用，引导学生树立网络意识和安全意识，认识到网络通信在物联网系统中的重要性和复杂性。

强调规范意识:强调在网络编程中要遵守网络协议和规范,保障网络通信的安全和稳定，培养学生的规范意识和责任意识。

11.5 详细内容讲解

11.5.1 网关网络通信方案

网络通信协议：介绍常用的网络通信协议，如 TCP/IP、UDP、MQTT、CoAP 等。

通信方式：讲解网关与终端设备、服务器之间的通信方式，如串口通信、网络通信等。

11.5.2 Socket 编程

Socket 基础：介绍 Socket 的基本概念和工作原理，包括客户端和服务器的通信模型。

Socket 编程步骤：

1. 创建 Socket：

   ```python
   import socket
   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   ```

2. 绑定地址：

   ```python
   s.bind(('localhost', 12345))
   ```

3. 监听连接：
   ```python
   s.listen(5)
   ```
4. 接受连接：
   ```python
   conn, addr = s.accept()
   ```
5. 发送和接收数据：
   ```python
   data = conn.recv(1024)
   conn.sendall(data)
   ```
6. 关闭 Socket：
   ```python
   conn.close()
   s.close()
   ```

11.5.3 `requests`上传文件及下载文件

上传文件：

```python
import requests

files = {'file': open('example.txt', 'rb')}
response = requests.post('http://example.com/upload', files=files)
print(response.text)
```

下载文件：

```python
response = requests.get('http://example.com/download')
with open('downloaded_file.txt', 'wb') as f:
    f.write(response.content)
```

11.5.4 `hbmqtt`的安装及命令操作和 API 编程

安装`hbmqtt`：

```bash
pip install hbmqtt
```

命令操作：

启动 MQTT 代理：

```bash
hbmqtt -c hbmqtt_config.yaml
```

发布消息：

```bash
hbmqtt pub -t "test/topic" -m "Hello, World!"
```

订阅消息：

```bash
hbmqtt sub -t "test/topic"
```

API 编程：

```python
import asyncio
from hbmqtt.client import MQTTClient

async def test_coro():
    C = MQTTClient()
    await C.connect('mqtt://localhost/')
    await C.subscribe([('test/topic', QOS_1)])
    await C.publish('test/topic', b'Hello, World!')
    message = await C.deliver_message()
    print(message.publish_packet.variable_header.topic_name,
message.publish_packet.data)

asyncio.get_event_loop().run_until_complete(test_coro())
```

11.5.5 LoRa 网络通信

模块初始化：

```python
import LoRa
lora = LoRa.LoRa()
lora.init(freq=433, bw=LoRa.BW_125, sf=7, cr=4, power=17)
```

数据监听与接收：

```python
while True:
    data = lora.recv()
    if data:
        print(f"Received: {data}")
```

数据缓存：

```python
lora.set_rxbuf_size(256)
```

11.6 课堂实训项目：实现基于 LoRa 的物联网通信

实施步骤

1. 准备硬件：
   准备 LoRa 模块（如 SX1278）和树莓派。
2. 安装 LoRa 库：
   ```bash
   pip install pyLoRa
   ```
3. 编写 LoRa 通信程序：
   发送端：
   ```python
   import LoRa
   lora = LoRa.LoRa()
   lora.init(freq=433, bw=LoRa.BW_125, sf=7, cr=4, power=17)
   while True:
       lora.send("Hello, World!")
       time.sleep(1)
   ```
   接收端：
   ```python
   import LoRa
   lora = LoRa.LoRa()
   lora.init(freq=433, bw=LoRa.BW_125, sf=7, cr=4, power=17)
   while True:
       data = lora.recv()
       if data:
           print(f"Received: {data}")
   ```

预期结果

学生能够成功实现基于 LoRa 的物联网通信。

学生能够理解 LoRa 网络通信的原理和实现方法。

11.7 课后作业

1. 任务一：完成课堂实训项目中的所有任务，并提交完整的代码和结果文件。

2. 任务二：查阅相关资料，了解其他网络通信协议（如 CoAP、XMPP）的基本操作方法，并与 MQTT 进行对比分析。

3. 任务三：编写一个 Python 程序，使用`requests`库实现一个简单的文件上传和下载功能，并使用`hbmqtt`库实现一个简单的 MQTT 客户端程序，能够订阅和发布消息。

第 12 章 物联网后台 Web 开发

12.1 教学内容

Django 简介。

创建一个网站，Django 安装，创建项目及运行访问。

网站首页，源码文件结构、视图与URL配置。

Django模板系统与模板继承。

Django模型，安装MySQL数据库，创建模型及配置、用户注册与账户登录管理等。

12.2 教学目标

掌握Django框架的基本使用方法和应用场景。

了解Web开发的基本流程和原理。

能够运用Django框架开发一个简单的物联网后台Web应用。

12.3 教学重难点

重点：Django模型，安装MySQL数据库，创建模型及配置、用户注册与账户登录管理等。

难点：Django模板系统与模板继承。

12.4 课程思政设计

强调用户体验：在讲解Django框架和Web开发的过程中，强调用户体验和界面设计的重要性，引导学生树立以用户为中心的开发理念，培养学生的用户意识和服务意识。

激发创新思维：介绍Web开发在物联网后台管理中的应用，激发学生对物联网应用开发的兴趣和创新思维。

12.5 详细内容讲解

12.5.1 Django简介

Django框架简介：Django是一个高级的Python Web框架，它鼓励快速开发和干净、实用的设计。

Django的优势：

内置功能丰富：包括用户认证、表单处理、模板引擎等。

安全性高：提供多种安全机制，如防止SQL注入、跨站脚本攻击等。

可扩展性强：支持多种数据库和中间件。

12.5.2 创建一个网站

安装Django：

```bash
pip install django
```

创建项目：

```bash
django-admin startproject myproject
cd myproject
```

运行访问：

```bash
python manage.py runserver
```

打开浏览器访问 `http://127.0.0.1:8000/`，查看默认的Django欢迎页面。

12.5.3 网站首页

源码文件结构：

`myproject/`：项目的根目录。
`manage.py`：项目的命令行工具。
`myproject/`：项目的 Python 包。
`settings.py`：项目的配置文件。
`urls.py`：项目的 URL 声明。
`wsgi.py`：项目的 WSGI 配置文件。

视图与 URL 配置：

创建视图：

```python
# myproject/views.py
from django.http import HttpResponse

def home(request):
    return HttpResponse("Hello, Django!")
```

配置 URL：

```python
# myproject/urls.py
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),
]
```

12.5.4 Django 模板系统与模板继承

模板系统：

创建模板文件：

```html
<!-myproject/templates/base.html -->
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}My Site{% endblock %}</title>
</head>
<body>
    <header>
        <h1>My Django Site</h1>
    </header>
    <main>
```

```
            {% block content %}{% endblock %}
        </main>
        <footer>
            <p>&copy; 2024 My Company</p>
        </footer>
    </body>
    </html>
    ```
```

继承模板：

```
    ```html
    <!-myproject/templates/home.html -->
    {% extends "base.html" %}
    {% block title %}Home Page{% endblock %}
    {% block content %}
    <h2>Welcome to the Home Page</h2>
    <p>This is the home page of my Django site.</p>
    {% endblock %}
    ```
```

### 12.5.5 Django 模型

安装 MySQL 数据库：

```
    ```bash
    pip install mysqlclient
    ```
```

配置数据库：

```
    ```python
    # myproject/settings.py
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'NAME': 'mydatabase',
            'USER': 'myuser',
            'PASSWORD': 'mypassword',
            'HOST': 'localhost',
            'PORT': '3306',
        }
    }
    ```
```

创建模型：

```
    ```python
    # myproject/models.py
    from django.db import models

    class User(models.Model):
        username = models.CharField(max_length=100)
```

```
    email = models.EmailField()
    password = models.CharField(max_length=100)
```

用户注册与账户登录管理：

注册视图：

```python
# myproject/views.py
from django.shortcuts import render, redirect
from .forms import RegistrationForm

def register(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
    else:
        form = RegistrationForm()
    return render(request, 'register.html', {'form': form})
```

登录视图：

```python
from django.contrib.auth import authenticate, login

def user_login(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username,
password=password)
        if user is not None:
            login(request, user)
            return redirect('home')
    return render(request, 'login.html')
```

12.6 课堂实训项目：开发一个简单的物联网后台 Web 应用

实施步骤

1. 创建 Django 项目：

安装 Django 并创建项目：

```bash
pip install django
django-admin startproject iot_backend
cd iot_backend
```

```
创建一个应用：
```bash
python manage.py startapp iot_app
```
2. 配置数据库：
安装 MySQL 数据库并配置：
```bash
pip install mysqlclient
```
修改`settings.py`文件，配置数据库：
```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'iot_database',
        'USER': 'iot_user',
        'PASSWORD': 'iot_password',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```
3. 创建模型：
在`iot_app/models.py`中定义模型：
```python
from django.db import models

class Device(models.Model):
    name = models.CharField(max_length=100)
    status = models.BooleanField(default=False)

    def __str__(self):
        return self.name
```
4. 创建视图：
在`iot_app/views.py`中创建视图：
```python
from django.shortcuts import render
from .models import Device

def device_list(request):
    devices = Device.objects.all()
    return render(request, 'device_list.html', {'devices': devices})
```
```

```
5. 配置 URL：
   在`iot_app/urls.py`中配置 URL：
   ```python
   from django.urls import path
   from . import views

   urlpatterns = [
       path('', views.device_list, name='device_list'),
   ]
   ```
6. 创建模板：
   在`iot_app/templates/device_list.html`中创建模板：
   ```html
   <!DOCTYPE html>
   <html>
   <head>
       <title>Device List</title>
   </head>
   <body>
       <h1>Device List</h1>
       <ul>
           {% for device in devices %}
           <li>{{ device.name }} {{ device.status }}</li>
           {% endfor %}
       </ul>
   </body>
   </html>
   ```
7.运行项目：
   运行 Django 项目：
   ```bash
   python manage.py runserver
   ```
   打开浏览器访问`http://127.0.0.1:8000/`，查看设备列表。
预期结果
学生能够成功开发一个简单的物联网后台 Web 应用。
学生能够理解 Django 框架的基本使用方法和 Web 开发的基本流程。

12.7 课后作业

1. 任务一：完成课堂实训项目中的所有任务，并提交完整的代码和结果文件。
2. 任务二：查阅相关资料，了解其他 Web 框架（如 Flask）的基本使用方法，并与 Django 进行对比分析，总结它们在物联网后台开发中的优缺点。
   ```

3. 任务三：扩展课堂实训项目，为物联网后台 Web 应用添加用户认证功能（用户注册、登录、登出），并实现设备状态的动态更新功能。具体要求如下：

用户注册时，需要验证用户名和密码的有效性（如密码长度、用户名唯一性等）。

用户登录后，能够看到设备列表，并对设备状态进行更新（如开启/关闭设备）。

设备状态的更新需要通过表单提交，并实时反映在页面上。

12.8 课堂总结

通过本章的学习，我们了解了 Django 框架的基本概念和使用方法，并通过一个简单的物联网后台 Web 应用的开发，掌握了如何使用 Django 进行 Web 开发。我们学习了 Django 的模型、视图、模板系统以及如何配置数据库和用户认证。这些知识将为后续的物联网项目实战提供坚实的基础。

在物联网系统中，后台 Web 应用是用户与设备交互的重要接口。通过 Web 应用，用户可以远程监控和控制物联网设备，查看设备数据，进行设备管理等操作。因此，掌握 Web 开发技术对于物联网开发人员来说是非常重要的。

在后续的学习中，我们将继续深入学习物联网开发的其他关键技术，如前端开发、移动应用开发、数据分析等，以构建更加完善的物联网系统。

第 13 章 物联网 Python 项目实战

13.1 教学内容

项目简介。

终端设备程序开发，采集空气温湿度、获取环境光强度、雨量检测、水位检测、土壤湿度检测、水泵的控制、入侵检测、灯光控制、电量检测等，以及 LoRa 通信和 JSON 消息。

网关程序开发，终端通信、数据库管理、文件备份、服务器通信、2G 模块的使用等。

服务器端程序开发，与网关通信、环境数据可视化、滴灌控制、灯光控制、报警显示与设置、设备管理、备份文件管理等。

13.2 教学目标

掌握物联网系统的整体架构和开发流程。

能够运用所学知识和技能，完成一个完整的物联网项目开发。

培养学生的实践能力和团队合作精神。

13.3 教学重难点

重点：终端设备程序开发、网关程序开发和服务器端程序开发的方法和技巧。

难点：物联网系统的整体架构设计和优化。

13.4 课程思政设计

培养实践能力：通过项目实战，让学生亲身体验物联网系统的开发流程和团队协作的重要性，培养学生的实践能力和团队合作精神。

激发创新意识：引导学生在项目开发中注重技术创新和应用效果，为解决实际问题提供有效的物联网解决方案，培养学生的创新意识和社会责任感。

13.5 详细内容讲解

13.5.1 项目简介

项目背景：介绍项目的背景和应用场景，如智慧农业、智能家居等。

项目目标：明确项目的目标和功能需求，如数据采集、设备控制、数据可视化等。

13.5.2 终端设备程序开发

采集空气温湿度：

使用 DHT11 或 DHT22 传感器采集空气温湿度数据。

示例代码：

```python
import Adafruit_DHT

sensor = Adafruit_DHT.DHT11
pin = 4

humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
print(f"Temperature: {temperature} C, Humidity: {humidity}%")
```

获取环境光强度：

使用 BH1750 传感器获取环境光强度。

示例代码：

```python
import smbus

bus = smbus.SMBus(1)
addr = 0x23

light = bus.read_i2c_block_data(addr, 0x10)
light = light[1] + (light[0] << 8)
print(f"Light Intensity: {light} lx")
```

雨量检测：

使用雨量传感器采集雨量数据。

示例代码：

```python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
pin = 17
GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def rain_callback(channel):
```

```
        print("Rain detected!")

    GPIO.add_event_detect(pin, GPIO.FALLING, callback=rain_callback)
    time.sleep(60)
    GPIO.cleanup()
    ```
水位检测：
  使用水位传感器采集水位数据。
  示例代码：
    ```python
    import RPi.GPIO as GPIO
    import time

    GPIO.setmode(GPIO.BCM)
    pin = 18
    GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

    def water_level_callback(channel):
        if GPIO.input(pin):
            print("Water level is high!")
        else:
            print("Water level is low!")

    GPIO.add_event_detect(pin, GPIO.BOTH, callback=water_level_callback)
    time.sleep(60)
    GPIO.cleanup()
    ```
土壤湿度检测：
  使用土壤湿度传感器采集土壤湿度数据。
  示例代码：
    ```python
    import RPi.GPIO as GPIO
    import time

    GPIO.setmode(GPIO.BCM)
    pin = 23
    GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

    def soil_moisture_callback(channel):
        if GPIO.input(pin):
            print("Soil is dry!")
        else:
            print("Soil is wet!")
```

```
GPIO.add_event_detect(pin, GPIO.BOTH, callback=soil_moisture_callback)
time.sleep(60)
GPIO.cleanup()
```

水泵的控制：

使用继电器模块控制水泵的开关。

示例代码：

```python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
pin = 24
GPIO.setup(pin, GPIO.OUT)

GPIO.output(pin, GPIO.HIGH)  # Turn on the pump
time.sleep(5)
GPIO.output(pin, GPIO.LOW)   # Turn off the pump
GPIO.cleanup()
```

入侵检测：

使用红外传感器或摄像头进行入侵检测。

示例代码：

```python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
pin = 25
GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

def motion_callback(channel):
    if GPIO.input(pin):
        print("Motion detected!")
    else:
        print("No motion detected.")

GPIO.add_event_detect(pin, GPIO.BOTH, callback=motion_callback)
time.sleep(60)
GPIO.cleanup()
```

灯光控制：

使用继电器模块或智能灯泡进行灯光控制。

示例代码：

```python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
pin = 5
GPIO.setup(pin, GPIO.OUT)

GPIO.output(pin, GPIO.HIGH)  # Turn on the light
time.sleep(5)
GPIO.output(pin, GPIO.LOW)   # Turn off the light
GPIO.cleanup()
```

电量检测：

使用电量传感器采集电量数据。

示例代码：

```python
import smbus

bus = smbus.SMBus(1)
addr = 0x40

voltage = bus.read_word_data(addr, 0x02)
voltage = (voltage << 8) & 0xFF00 | (voltage >> 8) & 0x00FF
voltage = voltage * 1.2 / 1000
print(f"Voltage: {voltage} V")
```

LoRa 通信和 JSON 消息：

使用 LoRa 模块进行数据通信，并使用 JSON 格式进行数据传输。

示例代码：

```python
import LoRa
import json

lora = LoRa.LoRa()
lora.init(freq=433, bw=LoRa.BW_125, sf=7, cr=4, power=17)

data = {"temperature": 25, "humidity": 60}
lora.send(json.dumps(data))
```

13.5.3 网关程序开发

终端通信：

使用 Socket 编程或 LoRa 模块与终端设备进行通信。

示例代码：

```python
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('localhost', 12345))
s.listen(5)

while True:
    conn, addr = s.accept()
    data = conn.recv(1024)
    print(f"Received data from {addr}: {data}")
    conn.sendall(b"Data received")
    conn.close()
```

数据库管理：

使用 SQLite 或 MySQL 数据库进行数据存储和管理。

示例代码：

```python
import sqlite3

conn = sqlite3.connect('iot_data.db')
cursor = conn.cursor()
cursor.execute('''
    CREATE TABLE IF NOT EXISTS sensor_data (
        id INTEGER PRIMARY KEY,
        temperature REAL,
        humidity REAL,
        timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
    )
''')
cursor.execute('INSERT INTO sensor_data (temperature, humidity) VALUES (?, ?)', (25.5, 60.0))
conn.commit()
conn.close()
```

文件备份：

使用 Python 进行文件的备份和恢复操作。

示例代码：

```python
import shutil

shutil.copy('data.db', 'backup/data_backup.db')
```

服务器通信：
使用 Socket 编程或 HTTP 请求与服务器进行通信。
示例代码：
```python
import requests

data = {"temperature": 25.5, "humidity": 60.0}
response = requests.post('http://example.com/data', json=data)
print(response.text)
```
2G 模块的使用
模块初始化：
使用 2G 模块（如 SIM800C）进行数据通信。
示例代码：
```python
import serial
初始化串口
ser = serial.Serial('/dev/ttyS0', 9600, timeout=1)
ser.write(b'AT\r\n')  # 发送 AT 命令测试模块
response = ser.read(100)
print(response.decode())
```

发送数据：
通过 2G 网络发送数据到服务器。
示例代码：
```python
发送 HTTP 请求
ser.write(b'AT+HTTPACTION=0,"GET","http://example.com/data"\r\n')
response = ser.read(100)
print(response.decode())
```

13.5.4 服务器端程序开发
与网关通信：
使用 Socket 编程或 HTTP 请求与网关进行通信。
示例代码：
```python
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/data', methods=['POST'])
def receive_data():
```

```
        data = request.json
        print(f"Received data: {data}")
        return jsonify({"status": "success"})

    if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5000)
    ```

环境数据可视化：
  使用 Flask 和 Plotly 实现数据可视化。
  示例代码：
    ```python
    from flask import Flask, render_template
    import plotly.express as px
    import pandas as pd

    app = Flask(__name__)

    @app.route('/')
    def index():
        df = pd.read_csv('sensor_data.csv')
        fig = px.line(df, x='timestamp', y='temperature')
        graph = fig.to_html(full_html=False)
        return render_template('dashboard.html', graph=graph)

    if __name__ == '__main__':
        app.run(debug=True)
    ```

滴灌控制：
  实现基于 Web 的滴灌系统控制。
  示例代码：
    ```python
    from flask import Flask, request, jsonify
    import RPi.GPIO as GPIO

    app = Flask(__name__)
    GPIO.setmode(GPIO.BCM)
    pin = 18
    GPIO.setup(pin, GPIO.OUT)

    @app.route('/irrigation', methods=['POST'])
    def control_irrigation():
        status = request.json.get('status')
```

```
        if status == 'on':
            GPIO.output(pin, GPIO.HIGH)
        elif status == 'off':
            GPIO.output(pin, GPIO.LOW)
        return jsonify({"status": "success"})

    if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5000)
    ```
```

灯光控制：
  实现基于 Web 的灯光控制。
  示例代码：
    ```python
    from flask import Flask, request, jsonify
    import RPi.GPIO as GPIO

    app = Flask(__name__)
    GPIO.setmode(GPIO.BCM)
    pin = 23
    GPIO.setup(pin, GPIO.OUT)

    @app.route('/light', methods=['POST'])
    def control_light():
        status = request.json.get('status')
        if status == 'on':
            GPIO.output(pin, GPIO.HIGH)
        elif status == 'off':
            GPIO.output(pin, GPIO.LOW)
        return jsonify({"status": "success"})

    if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5000)
    ```
报警显示与设置：
  实现基于 Web 的报警系统。
  示例代码：
    ```python
    from flask import Flask, request, jsonify

    app = Flask(__name__)

    @app.route('/alarm', methods=['POST'])
    def set_alarm():
```
```

```
            threshold = request.json.get('threshold')
            # 设置报警阈值
            return jsonify({"status": "success"})
        @app.route('/alarm/status', methods=['GET'])
        def get_alarm_status():
            # 获取报警状态
            return jsonify({"status": "active"})

        if __name__ == '__main__':
            app.run(host='0.0.0.0', port=5000)
        ```
```

设备管理：

实现设备的添加、删除和状态查询。

示例代码：

```python
from flask import Flask, request, jsonify
import sqlite3

app = Flask(__name__)

@app.route('/devices', methods=['POST'])
def add_device():
    device_id = request.json.get('device_id')
    conn = sqlite3.connect('iot_data.db')
    cursor = conn.cursor()
    cursor.execute('INSERT INTO devices (device_id) VALUES (?)',
(device_id,))
    conn.commit()
    conn.close()
    return jsonify({"status": "success"})

@app.route('/devices/<device_id>', methods=['DELETE'])
def delete_device(device_id):
    conn = sqlite3.connect('iot_data.db')
    cursor = conn.cursor()
    cursor.execute('DELETE FROM devices WHERE device_id=?', (device_id,))
    conn.commit()
    conn.close()
    return jsonify({"status": "success"})

@app.route('/devices', methods=['GET'])
def get_devices():
    conn = sqlite3.connect('iot_data.db')
    cursor = conn.cursor()
```
```

```
    cursor.execute('SELECT * FROM devices')
    devices = cursor.fetchall()
    conn.close()
    return jsonify(devices)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```
备份文件管理：
实现文件的备份和恢复。
示例代码：
```python
import shutil
def backup_file(source, destination):
    shutil.copy(source, destination)
def restore_file(source, destination):
    shutil.copy(source, destination)
```

13.6 课堂实训项目：智慧路灯系统设计

任务要求

设计一个智慧路灯系统，能够根据环境亮度自动控制路灯的开关，并将路灯状态上传到云端。

实施步骤

1. 硬件准备：
   准备光敏传感器、继电器模块、树莓派等硬件设备。
2. 环境亮度检测：
   使用光敏传感器检测环境亮度。
   示例代码：
   ```python
   import RPi.GPIO as GPIO
   import time

   GPIO.setmode(GPIO.BCM)
   pin = 26
   GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

   def light_callback(channel):
       if GPIO.input(pin):
           print("It's dark, turning on the light.")
       else:
           print("It's bright, turning off the light.")
```

```
GPIO.add_event_detect(pin, GPIO.BOTH, callback=light_callback)
time.sleep(60)
GPIO.cleanup()
```

3. 路灯控制：
   使用继电器模块控制路灯的开关。
   示例代码：
   ```python
   import RPi.GPIO as GPIO
   import time

   GPIO.setmode(GPIO.BCM)
   light_pin = 26
   relay_pin = 17
   GPIO.setup(light_pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
   GPIO.setup(relay_pin, GPIO.OUT)

   def control_light():
       if GPIO.input(light_pin):
           GPIO.output(relay_pin, GPIO.HIGH)  # Turn on the light
       else:
           GPIO.output(relay_pin, GPIO.LOW)   # Turn off the light

   try:
       while True:
           control_light()
           time.sleep(1)
   except KeyboardInterrupt:
       GPIO.cleanup()
   ```
4. 数据上传到云端：
   使用 HTTP 请求将路灯状态上传到服务器。
   示例代码：
   ```python
   import requests
   import RPi.GPIO as GPIO
   import time

   GPIO.setmode(GPIO.BCM)
   light_pin = 26
   relay_pin = 17
   GPIO.setup(light_pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
   GPIO.setup(relay_pin, GPIO.OUT)
```

```
        def control_light():
            if GPIO.input(light_pin):
                GPIO.output(relay_pin, GPIO.HIGH)  # Turn on the light
                status = "on"
            else:
                GPIO.output(relay_pin, GPIO.LOW)   # Turn off the light
                status = "off"
            return status

        try:
            while True:
                status = control_light()
                response = requests.post('http://example.com/lights',
json={"status": status})
                print(response.text)
                time.sleep(10)
        except KeyboardInterrupt:
            GPIO.cleanup()
```

13.7 课后作业

1. 任务一：完成课堂实训项目中的所有任务，并提交完整的代码和结果文件。
   要求：确保代码能够正常运行，实现智慧路灯系统的自动控制和状态上传功能。
   提交内容：
   完整的 Python 代码。
   运行结果的截图或视频。
   项目报告，包括设计思路、实现过程、遇到的问题及解决方案。

2. 任务二：扩展智慧路灯系统，增加以下功能：
   用户界面：使用 Flask 开发一个简单的 Web 界面，用户可以通过界面手动控制路灯的开关。
   历史数据记录：将路灯的状态变化记录到数据库中，并提供一个页面展示历史数据。
   报警功能：当环境亮度低于某个阈值时，自动发送报警信息（如邮件或短信）。

   示例代码：
   Flask Web 界面：
   ```python
   from flask import Flask, request, render_template
   import RPi.GPIO as GPIO

   app = Flask(__name__)
   relay_pin = 17
   GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(relay_pin, GPIO.OUT)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/light', methods=['POST'])
def control_light():
    status = request.form['status']
    if status == 'on':
        GPIO.output(relay_pin, GPIO.HIGH)
    elif status == 'off':
        GPIO.output(relay_pin, GPIO.LOW)
    return "Light turned " + status

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

历史数据记录：

```python
import sqlite3
from datetime import datetime

def log_light_status(status):
    conn = sqlite3.connect('light_data.db')
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS light_status (
            id INTEGER PRIMARY KEY,
            status TEXT,
            timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
        )
    ''')
    cursor.execute('INSERT INTO light_status (status) VALUES (?)',
(status,))
    conn.commit()
    conn.close()
```

报警功能：

```python
import smtplib
from email.mime.text import MIMEText
```

```
    def send_alert():
        msg = MIMEText("Low light alert!")
        msg['Subject'] = 'Light Alert'
        msg['From'] = 'your_email@example.com'
        msg['To'] = 'recipient_email@example.com'

        with smtplib.SMTP('smtp.example.com') as server:
            server.login('your_email@example.com', 'your_password')
            server.sendmail(msg['From'], [msg['To']], msg.as_string())
```

3. 任务三：撰写一篇项目总结报告，内容包括：
   项目背景和目标。
   系统架构设计。
   关键技术点和实现方法。
   测试结果和性能分析。
   遇到的问题及解决方案。
   项目改进方向和未来展望。
   报告格式：
   标题：智慧路灯系统设计与实现
   作者：[你的名字]
   日期：[具体日期]
   正文：按照上述内容撰写，字数不少于1500字。

13.8 课堂总结

通过本章的学习，我们完成了一个完整的物联网项目——智慧路灯系统的设计与实现。这个项目涵盖了从硬件设备的数据采集、网关的通信与数据处理，到服务器端的数据存储、可视化和用户交互等多个环节。通过这个项目，同学们不仅巩固了物联网开发的各个环节，还提升了团队协作和问题解决的能力。

在物联网开发中，项目实战是非常重要的一环。通过实际操作，同学们可以更好地理解理论知识，并将其应用于解决实际问题。希望同学们在今后的学习和工作中，能够继续探索物联网技术的更多应用场景，为智慧生活和智能城市建设贡献力量。