

# 从 PLC 到 PC 控制系统的转型实战课程

## 第1课：环境设置 + PLC vs Python 流程对比 + Python 基础

讲师：[May] | 课程时长：50 分钟

Environment Setup + PLC vs PC Architecture + Python Basics

Instructor: [May] | Class Duration: 50 Minutes



从传统工业控制到现代软件控制的转型之旅

# 课程目标与概述

## 学习目标

- 理解 PC 控制系统与传统 PLC 的区别
  - **PLC (Programmable Logic Controller) systems vs modern PC-based control systems**
- 掌握基础 Python 编程技能
  - **Python programming skills**
- 能够编写简单的 Python 控制功能
  - **Python scripts for control tasks**

# 开发环境准备 | Dev Environment Setup

## 📍 本课程所需开发环境 Tools Overview

工具类型 Type	工具名称 Tool	用途 Purpose
编辑器 Editor	VS Code (安装 Python 插件)	编写与调试 Python 程式
Python 环境	Python 3.10+	支持 FastAPI
API 测试工具	Postman 官网	模拟 HTTP 请求，测试 GET / POST 接口
模拟器（选用）	ModbusPal / ModRSsim2	模拟 Modbus 工业设备通讯

✓ 强烈建议注册 GitHub 账号，方便未来版本控制与项目协作。

# 控制功能对比：PLC vs Python

任务目标：控制一个灯号在按下按钮后延时2秒开启

控制流程：

- 按钮按下 (I0.0)
- 延时 2 秒
- 灯光输出 (Q0.0)

# 控制功能对比：PLC vs Python

PLC

```
|--[Start_Button]--( )--[TON 2s]--( )--  
[Output_Light]--|
```

任务目标：控制一个灯  
号在按下按钮后延时2秒  
开启

控制流程：

- 按钮按下 (I0.0)
- 延时 2 秒
- 灯光输出 (Q0.0)

对照要点：

- 逻辑控制流程
- 输入/输出方式
- 状态表达方式

Python

```
import time
```

```
def control_light(input):  
    light_status = "OFF"  
  
    if input == "按下 Enter 键启动":  
        time.sleep(2) # 延时2秒  
        print("灯光已开启")  
        light_status = "ON"  
  
    return light_status  
  
# 调用函数  
control_light("按下 Enter 键启动")
```

# 控制功能对比：PLC vs Python

## PLC

```
|--[Start_Button]--( )--[TON 2s]--( )--  
[Output_Light]--|
```

### 控制流程说明

- [Start\_Button] 为一个输入点（如 I0.0），代表按钮被按下
- 触发一个 2 秒延时计时器 TON（例如 T1）
- 延时完成后输出点 [Output\_Light]（如 Q0.0）被置为 ON，控制灯亮

### 控制逻辑特点

- 输入 (Input)：物理按钮输入 (I点)
- 逻辑 (Logic)：使用计时器 TON 控制流程
- 输出 (Output)：驱动灯或继电器的输出点 (Q点)
- 状态：由梯形图中接点开启/断开表达

任务目标：控制一个灯  
号在按下按钮后延时2秒  
开启

控制流程：

- 按钮按下 (I0.0)
- 延时 2 秒
- 灯光输出 (Q0.0)

对照要点：

- 逻辑控制流程
- 输入/输出方式
- 状态表达方式

## Python

```
import time
```

```
def control_light(input):
```

```
    light_status = "OFF"
```

```
    if input == "按下 Enter 键启动":
```

```
        time.sleep(2) # 延时2秒
```

```
        print("灯光已开启")
```

```
        light_status = "ON"
```

```
    return light_status
```

```
# 调用函数
```

```
control_light("按下 Enter 键启动")
```

### 控制流程说明

- input() 等待用户按下键盘，模拟按钮按下
- time.sleep(2) 实现2秒延时
- print() 模拟灯被打开
- return light\_status 代表输出

### 控制逻辑特点

- 输入 (Input)：来自终端键盘（用户动作）
- 逻辑 (Logic)：顺序执行 + 延迟函数
- 输出 (Output)：使用 print() 模拟状态改变
- 状态：由变量和函数控制，输出的是文字信息或设备指令

# Python 控制逻辑基础

## 核心概念

- 变量 (Variable): 表达状态, 如 `led_status = "OFF"`
- 条件判断 (if/else): 控制逻辑分支
- 循环 (loop): 重复执行任务
- 函数 (function): 封装控制逻辑

类比: `input()` ⇌ PLC输入点; `print()` ⇌ PLC输出点

```
# 定义变量表示状态  
led_status = False # 相当于PLC中的状态位  
  
# 定义控制函数  
def toggle_led():  
    global led_status  
    led_status = not led_status  
  
    if led_status:  
        print("LED 已开启")  
    else:  
        print("LED 已关闭")
```

# Python 面向对象编程 (OOP) 关键字及中文解释表

关键字 / 概念	英文关键词	说明 (中文)
类	Class	定义对象的模板或蓝图，描述一类事物的属性和行为
对象	Object / Instance	根据类创建的具体实例，具有类定义的属性和方法
方法	Method	定义在类中的函数，用于描述对象的行为
属性	Attribute	类或对象中存储的数据，用于描述对象的状态
继承	Inheritance	子类继承父类的属性和方法，实现代码复用和层次结构
多态	Polymorphism	不同类的对象对同一消息（方法调用）作出不同的响应
封装	Encapsulation	将数据和方法包装在类内部，隐藏实现细节，保护对象的状态
构造函数	Constructor ( <code>_init_</code> )	类被实例化时自动调用的方法，用于初始化对象属性
继承父类构造函数	super()	用于调用父类的方法，通常用于子类中调用父类的构造函数
私有属性/方法	Private Attribute/Method	以双下划线开头（如 <code>_name</code> ），在类外部不可直接访问
公有属性/方法	Public Attribute/Method	默认访问权限，类内外均可访问
类变量	Class Variable	属于类的变量，所有实例共享
实例变量	Instance Variable	属于对象的变量，每个实例拥有独立副本
抽象类	Abstract Class	不能被实例化，只能被继承，通常包含抽象方法（接口规范）
抽象方法	Abstract Method	只声明不实现的方法，子类必须实现
接口	Interface	定义规范，约束类必须实现某些方法（Python中通过抽象基类实现）

# Python 面向对象编程相关基础关键字及中文解释表

关键字 / 概念	英文关键词	说明（中文）
变量	Variable	用于存储数据的命名空间，值可以改变
数据类型	Data Type	数据的种类，如整数(int)、浮点数(float)、字符串(str)等
条件语句	Conditional	根据条件执行不同代码，如 if、elif、else
循环	Loop	重复执行代码块，如 for、while
函数	Function	封装可复用代码块，用 def 定义
参数	Parameter	传递给函数的变量
返回值	Return Value	函数执行后输出的结果
模块	Module	包含代码的文件或库，用于组织和复用代码
异常处理	Exception Handling	用 try-except 捕获和处理运行时错误
类	Class	面向对象编程的模板，定义属性和方法
对象	Object	类的实例，具体的数据和行为实体
继承	Inheritance	子类继承父类的属性和方法
多态	Polymorphism	不同对象对相同方法调用有不同响应
封装	Encapsulation	隐藏内部实现，保护对象数据
构造函数	Constructor	用 init 初始化对象属性
私有变量/方法	Private Member	以双下划线开头，外部不可访问

# 实作任务：用 Python 实现控制灯逻辑 From 0 to 1

```
import time

led_status = False # False 表示关闭, True 表示开启

# 执行灯的开关切换逻辑
def toggle_led():
    global led_status # 使用全局变量保存灯状态
    led_status = not led_status # 状态切换
    if led_status:
        print("LED 已开启")
    else:
        print("LED 已关闭")

# 模拟按钮输入 (input) + 延时控制 (sleep)
def press_button():
    input("请按下 Enter 键模拟按钮启动...")
    print("正在延时 2 秒...")
    time.sleep(2) # 等待 2 秒
    toggle_led() # 切换 LED 状态

# 加入主程序循环
while True:
    user_input = input("按 Enter 切换 LED, 或输入 'exit' 退出:")
    if user_input == "exit":
        print("程序已退出")
        break
    else:
        press_button()
```

## 任务提示：

- 使用全局变量保存 LED 状态
- 每次调用函数时切换状态 (Boolean 取反)
- 通过 while 循环实现持续操作

# Python 控制灯程序关键词与实现对照表

```
import time

led_status = False # False 表示关闭, True 表示开启

# 执行灯的开关切换逻辑
def toggle_led():
    global led_status # 使用全局变量保存灯状态
    led_status = not led_status # 状态切换
    if led_status:
        print("LED 已开启")
    else:
        print("LED 已关闭")

# 模拟按钮输入 (input) + 延时控制
(sleep)
def press_button():
    input("请按下 Enter 键模拟按钮启动...")
    print("正在延时 2 秒...")
    time.sleep(2) # 等待 2 秒
    toggle_led() # 切换 LED 状态

# 加入主程序循环
while True:
    user_input = input("按 Enter 切换 LED, 或输入 'exit' 退出:")
    if user_input == "exit":
        print("程序已退出")
        break
    else:
        press_button()
```

Python 关键词 / 代码片段	中文含义与功能说明	PLC 对应概念 / 功能
import time	导入模块, 提供时间相关函数, 用于延时	TON 定时器 (通电延时器)
led_status = False	布尔变量, 记录灯的状态 (False=关, True=开)	状态位 (如 M0.0)
def toggle_led():	定义函数, 切换灯的状态	过程块或功能块, 封装动作
global led_status	使用全局变量, 函数内修改全局状态变量	全局标志位, 允许多个程序块共享
led_status = not led_status	取反操作, 状态切换	取反位/线圈反转 (翻转继电器状态)
if led_status:	条件判断, 根据状态执行不同操作	接点 (触点) 判断, 条件分支
print("LED 已开启")	模拟灯亮输出提示	输出线圈动作 (点亮灯)
print("LED 已关闭")	模拟灯灭输出提示	输出线圈动作 (关闭灯)
def press_button():	定义函数, 模拟按钮按下操作	按钮触点输入 (I0.0)
input("请按下 Enter 键...")	程序等待用户输入, 模拟按钮动作	等待按钮按下信号
time.sleep(2)	程序延时2秒	TON定时器延时2秒
toggle_led()	调用函数, 切换灯状态	输出继电器线圈切换动作
while True:	无限循环, 持续执行控制逻辑	PLC主程序循环扫描
user_input = input(...)	等待用户输入, 用于控制循环退出	人机界面 (HMI) 输入或手动停止
if user_input == "exit":	判断是否退出程序	条件跳转或程序停止
break	跳出循环, 停止程序	跳出循环或停止程序
else: press_button()	继续执行按钮控制逻辑	按钮启动控制逻辑

# QA | 下节课预告

## 思维转换与挑战

- 从硬件思维到软件思维的转变
- 状态管理与数据持久化
- 异步操作与事件驱动编程

## 下节课预告

- 面向对象的编程方式
- 面向对象的核心概念：**类 (Class)**、**对象 (Object)**、**属性 (Attribute)** 和 **方法 (Method)**
  - 如何使用 `init()` 构造函数初始化对象
  - 如何封装状态与行为，实现更模块化的程序设计
  - 使用实例管理多个设备状态（例如多个LED灯）
  - 与前一节控制灯程序结合，重构为对象导向的设计

# 课后作业：实现红蓝绿三色灯控制器

## 🎯 作业目标

通过实现一个三色灯（红灯、绿灯、蓝灯）控制系统，帮助你：

- 巩固 **函数定义与调用**
- 熟练使用 **变量状态控制 和 布尔值切换**
- 练习 **条件判断 (if / else)**
- 掌握 **输入 + 循环结构 (while / input)** 的基本使用

## 🧠 作业内容

请用 Python 编写一个命令行程序，模拟控制三个灯的开关状态：

### ✓ 功能要求

1. **灯初始化：** 红灯、绿灯、蓝灯初始状态为关闭 (False)
2. **主菜单功能** (循环执行)：
  - 用户可以输入：
    - red: 切换红灯开关状态
    - green: 切换绿灯开关状态
    - blue: 切换蓝灯开关状态
    - show: 显示当前三色灯的状态
    - exit: 退出程序
3. **每次状态切换后，输出灯当前状态 (开 / 关)**

## ✳️ 示例交互

```
text 请输入指令 (red / green / blue / show / exit) : red
--> 红灯状态：开
请输入指令：blue
--> 蓝灯状态：开
请输入指令：show
--> 红灯：开，绿灯：关，蓝灯：开
请输入指令：exit
--> 程序已退出。
```