

# 从 PLC 到 PC 控制系统的转型实战课程

## 第2课：异步控制与面向对象程序设计 (OOP: Object-Oriented Programming)

物件导向程式设计

讲师: [May] | 课程时长: 50 分钟

Asynchronous & OOP-Based Control Programming

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

- 理解异步控制编程的核心思想与应用场景
  - Understand asynchronous control with `async / await` and event-driven architecture.
- 掌握 Python 的协程与事件循环机制
  - Learn Python's `asyncio` model to handle non-blocking concurrent control flows.
- 熟悉面向对象的编程思想
  - Understand OOP principles: class, object, attributes, methods, encapsulation.

# 同步 vs 异步 vs 并发 vs 并行 (Synchronous vs Asynchronous vs Concurrency vs Parallelism)

## 关键概念总览

中文术语	英文术语	简要定义	应用
同步	Synchronous	一件事做完才能做下一件事 (任务阻塞)	PLC 扫描循环：只有前一步完成，下一步才执行； Python 普通函数执行
异步	Asynchronous	启动后不用等完成，可继续做别的事 (任务非阻塞)	Python 中使用 <code>async/await</code> ，设备延时操作时程序可继续响应其他任务
并发	Concurrency	同一时间段内处理多个任务， <b>任务交替切换执行</b>	单核CPU中使用 <code>asyncio.gather()</code> 同时控制多个灯的开关动作
并行	Parallelism	多个任务真正同时运行， <b>多核 CPU / 多线程或多进程执行</b>	Python 使用 <code>multiprocessing</code> 控制多个物理设备，或工控多通道实时采集处理

# 同步 vs 异步：Synchronous vs Asynchronous

## 📌 针对任务的等待方式

这是从单个任务的“执行流程”角度来定义的，重点是函数/任务是否等待结果返回。

比较维度	描述	关键区别	举例
同步	调用一个任务时必须等它完成，才能执行下一步	等待结果返回，阻塞执行	正常函数调用 <code>time.sleep(2)</code> 会阻塞主线程
异步	启动一个任务后，不必等它完成，可以做其他事	不等待结果返回，非阻塞	使用 <code>async def + await</code> ，延时期间可做别的事情

# 并发 vs 并行：Concurrency vs Parallelism

## 📌 针对任务的调度/运行方式

这是从系统执行多个任务的能力角度来定义的，关注的是多任务是“轮流”执行还是“同时”执行。

比较维度	描述	关键区别	举例
并发	多个任务在同一时间段内交替进行	通常是单核/单线程交替运行	asyncio 在一个线程中切换多个任务
并行	多个任务在同一时刻同时执行	多核CPU多线程真正并行运行	多线程、多进程或GPU并行处理

# 🔍 做饭比喻版解释：同步 vs 异步 & 并发 vs 并行



概念	做饭类比	关键词
同步	烧水 → 等水烧开 → 才开始切菜 → 切完菜 → 才开始炒菜	一件事做完才能做下一件（按部就班）
异步	烧水（放在炉上）→ 开始切菜 → 水开后提示我 → 切菜结束 → 炒菜	发起任务后不等结果，继续干别的事
并发	你一个人来回做：切几刀菜 → 看下水 → 翻炒一下 → 继续切菜...	多任务交替进行（看上去一起做）
并行	你和你朋友一起做：你切菜、他烧水、她炒菜，同时各干各的	多任务真正同时进行（多核/多线程）

# 同步 vs 异步：学习资料

**Concurrency is NOT Parallelism**

The diagram illustrates four scenarios:

- Not Concurrent, Not Parallel:** Shows a single CPU Core 1 executing Task 1 (blue bar) and Task 2 (red bar) sequentially.
- Concurrent, Not Parallel:** Shows two CPU cores, both labeled "CPU Core 1". One core executes Task 1 (blue bar), while the other executes Task 2 (red bar). They are connected by dashed arrows, indicating they share the same physical hardware.
- Not Concurrent, Parallel:** Shows two separate CPU cores, "CPU Core 1" and "CPU Core 2". Each core executes Task 1 (blue bar) and Task 2 (red bar) sequentially.
- Concurrent, Parallel:** Shows two separate CPU cores, "CPU Core 1" and "CPU Core 2". Both cores execute Task 1 (blue bar) and Task 2 (red bar) simultaneously, with each task being split into multiple parallel steps (green, yellow, orange bars).

YouTube 🔗

**Concurrency Vs Parallelism!**

Get a Free System Design PDF with 158 pages by subscribing to our weekly newsletter: <https://bit.ly/bytebytegoytTopic> Animation tools: Adobe Illustrator and...

04:13

# await vs asyncio (See Notebook)

# 面向对象程序设计 (OOP)

## 1. OOP的四大核心概念

### 1.1 类 (Class)

- 类是“模具”或“蓝图”，定义对象的属性和行为
- 例如：定义一个“狗”的类，包含属性（颜色、年龄）和行为（叫、跑）

### 1.2 对象 (Object)

- 对象是类的实例，是具体的实体
- 例如：“小白”是一只具体的狗，是“狗”类的一个对象

### 1.3 封装 (Encapsulation)

- 把属性和方法包裹在类内部，隐藏实现细节
- 对外提供接口，保护数据安全

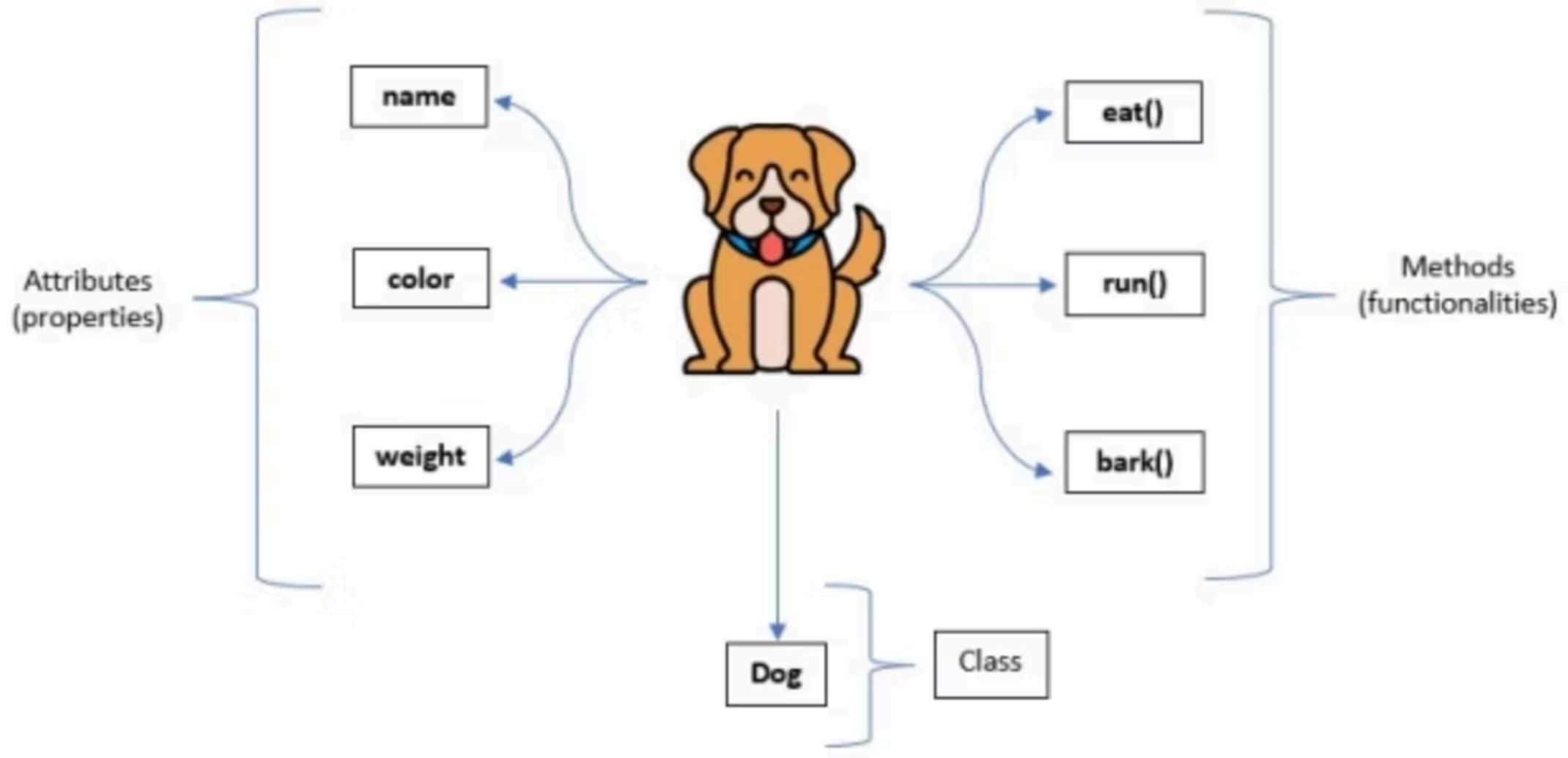
### 1.4 继承 (Inheritance)

- 新类继承已有类，复用已有代码
- 例如：“导盲犬”继承“狗”，自动拥有“狗”的属性和行为，还可以新增功能

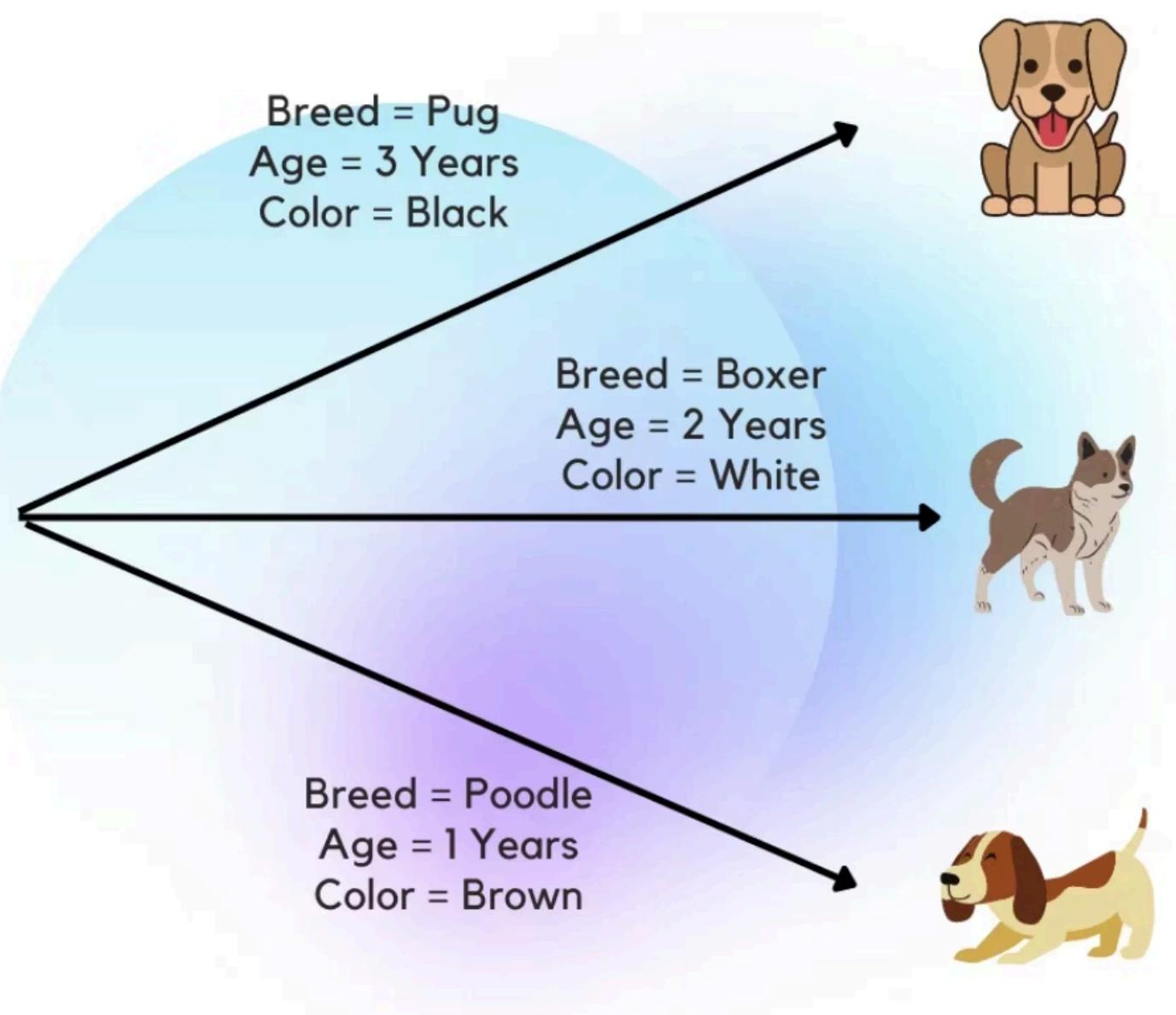
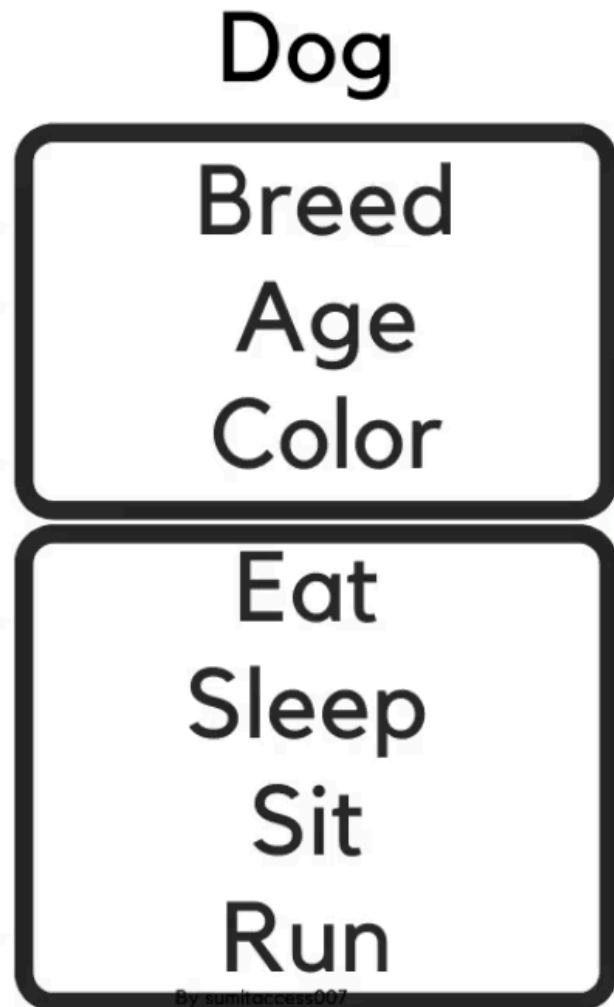
### 1.5 多态 (Polymorphism)

- 不同类对象用同一接口调用，表现不同的行为
- 例如：“狗”和“猫”都能“叫”，但叫声不同

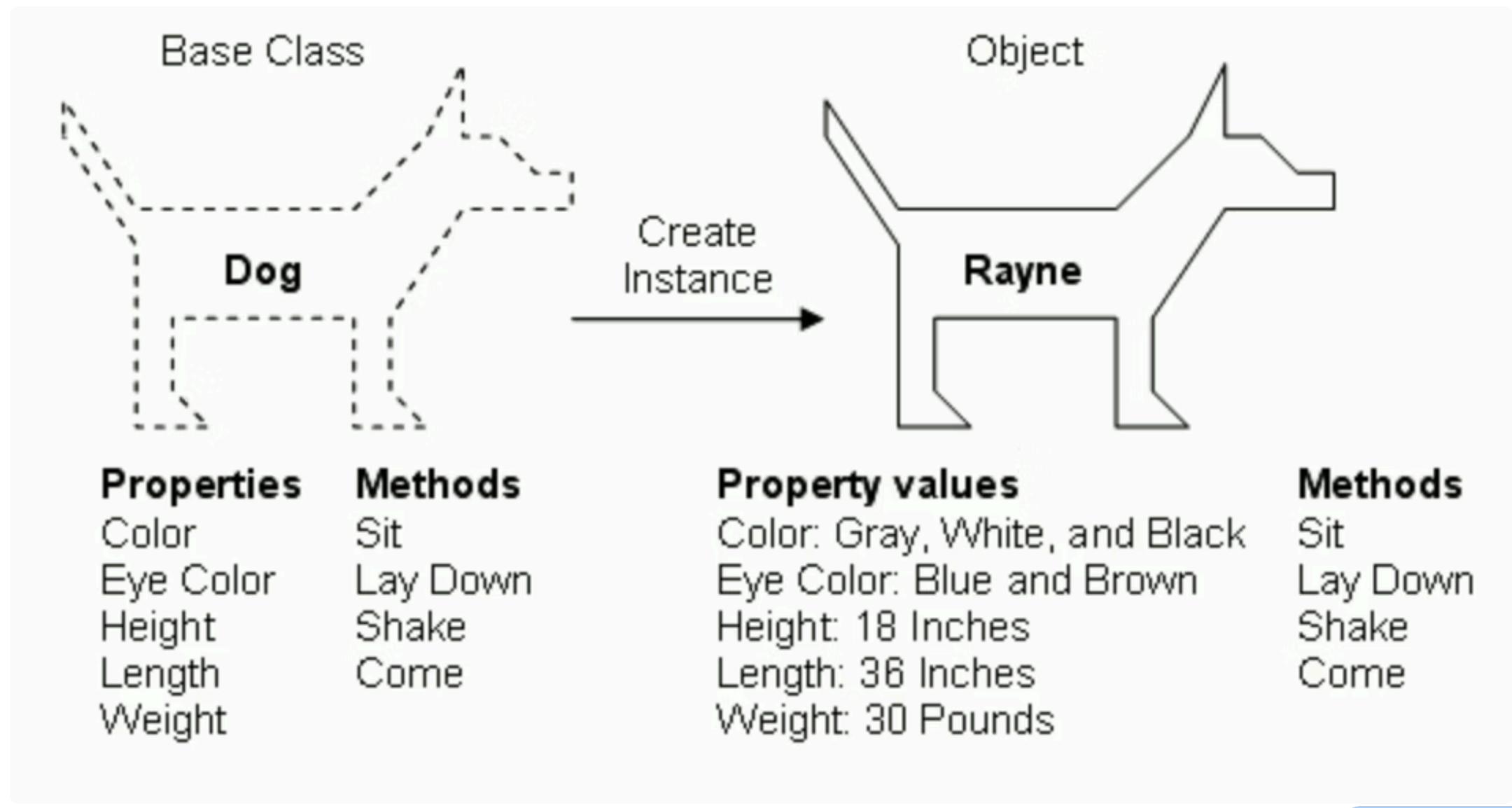
# 面向对象程序设计 (OOP): 🐶 举例



# 面向对象程序设计 (OOP): 🐶 举例



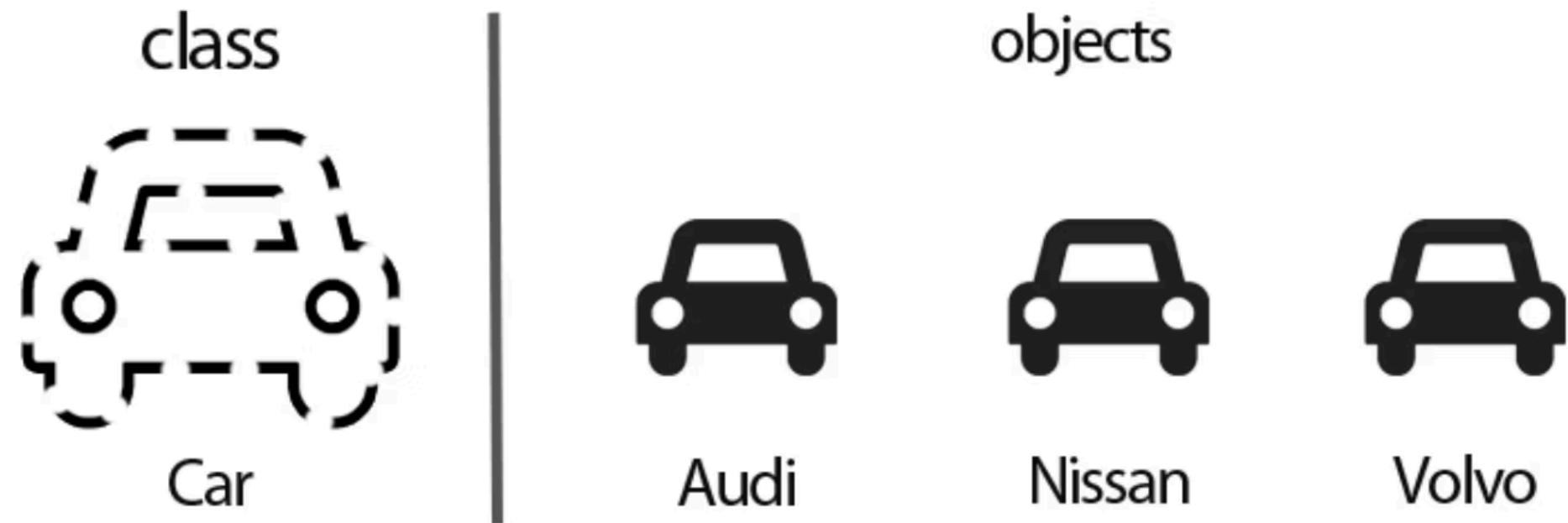
# 面向对象程序设计 (OOP): 🐶 举例



# 面向对象程序设计 (OOP): 🚗 举例

物件導向: [https://codimd.mclmath.ncu.edu.tw/s/W3A\\_thNcA](https://codimd.mclmath.ncu.edu.tw/s/W3A_thNcA)

類別與物件的關係



# 面向对象程序设计 (OOP)

## 2. OOP是什么？

- 编程的一种范式
- 用“类”和“对象”模拟现实世界事物
- 组织代码更清晰，复用性更好

## 3. 为什么要用OOP？

- 代码结构清晰，易维护
- 复用代码，减少重复
- 模拟真实世界，符合直觉
- 方便团队协作和项目扩展

# 面向对象程序设计 (OOP): 🐶 举例

```
class Dog:  
    def __init__(self, name, color):  
        self.name = name # 属性 (Attribute)  
        self.color = color # 属性 (Attribute)  
  
    def bark(self): # 方法 (Method)  
        print(f"{self.name} says: Woof!")  
  
# 创建对象  
my_dog = Dog("小白", "白色")  
my_dog.bark() # 输出: 小白 says: Woof!
```



# 控制灯状态的两种方法对比

## 函数式写法（适用于少量设备）

```
red_status = False
blue_status = False

# toggle_red
def toggle_red():
    global red_status
    red_status = not red_status

    print(f"红灯状态：{'开' if red_status else '关'}")

# toggle_blue
def toggle_blue():
    global blue_status
    blue_status = not blue_status

    print(f"蓝灯状态：{'开' if blue_status else '关'}")
```

## OOP 写法（推荐：更清晰、可扩展）

```
class LED:
    def __init__(self, name):
        self.name = name
        self.status = False

    def toggle(self):
        self.status = not self.status

    print(f"{self.name} 状态：{'开' if self.status else '关'}")

# 创建两个对象
red_led = LED("红灯")
blue_led = LED("蓝灯")

# 调用方法
red_led.toggle()
blue_led.toggle()
```



# QA | 下节课预告

## 思维转换与挑战

- 异步操作与事件驱动编程
- 面向对象的编程思维模式
- 簡單理解物件導向程式設計

## 下节课预告

- 面向对象的编程方式
- 巩固 OOP 核心概念：类、对象、属性、方法、封装、组合
  - 学会用类组织控制逻辑，模拟设备之间的交互
  - 能设计出结构清晰、支持扩展的控制系统
  - 实作一个“多按钮控制多灯”的模拟系统

# 课后作业：使用 OOP 实现三色灯控制器

## 作业目标

通过面向对象方式实现一个三色灯（红灯、绿灯、蓝灯）控制系统，帮助你：

- 巩固类与对象的定义与使用
- 理解封装的概念：将状态与行为组合在一个类中
- 练习类之间的组合与交互（按钮控制灯）
- 培养结构清晰、可扩展的编程习惯

## 作业内容

请用 Python + OOP（类与对象）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

#### 系统初始化：

- 创建3个灯对象：红灯（Red）、绿灯（Green）、蓝灯（Blue），初始状态为关闭（False）
- 每个灯是 LED 类的实例，拥有名称属性和 toggle() 方法

#### 主菜单功能（循环执行，直到输入exit）：

用户可以输入以下命令：

用户输入	功能说明
red	切换红灯开/关
green	切换绿灯开/关
blue	切换蓝灯开/关
show	显示当前所有灯的状态（格式见下）
exit	退出程序

#### 示例输出格式：

```
请输入命令 (red/green/blue/show/exit) : red
红灯 状态：开

请输入命令：show
红灯 状态：开
绿灯 状态：关
蓝灯 状态：关
```

#### 加分挑战（选做）

-  将灯放入一个列表中，用循环结构统一处理所有灯对象
-  添加一个命令 all，可以一次切换全部灯的状态

# 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