LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG OOP

TIẾT 1: OPP - TÍNH ĐÓNG GÓI

MỤC TIÊU

- Kiến thức: Hiểu các khái niệm Lớp, Đối tượng, Thuộc tính, Phương thức và ý nghĩa Tính Đóng Gói.
- Kỹ năng: Phân tích Class đơn giản, viết Class nhỏ, tạo đối tượng và sử dụng thuộc tính, phương thức.
- Tư duy: Hình thành tư duy lập trình có cấu trúc.

ĐẶT VẤN ĐỀ

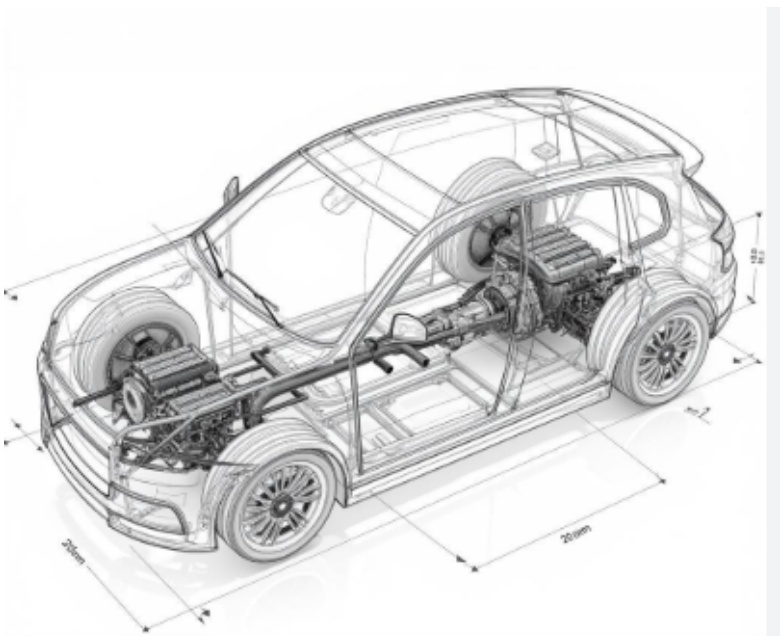
```
{
  "main": {
    "temp": 300.15,
    "feels_like": 302.45,
    "pressure": 1010,
    "humidity": 75
  }
}
```

- Đọc nhiệt độ và độ ẩm ?
- Cách viết này có nhược điểm gì
- Làm thế nào để gọn gàng hơn?

KHÁI NIỆM LẬP TRÌNH OPP

CLASS , OBJECT

Class (Lớp): Xem như bản thiết kế chiếc xe



Object (Đối tượng hay thể hiện): Thể hiện của lớp: xe đua , xe bán tải...

ATRRIBUTE , METHOD

Attribute (Thuộc tính): Đặc điểm của xe

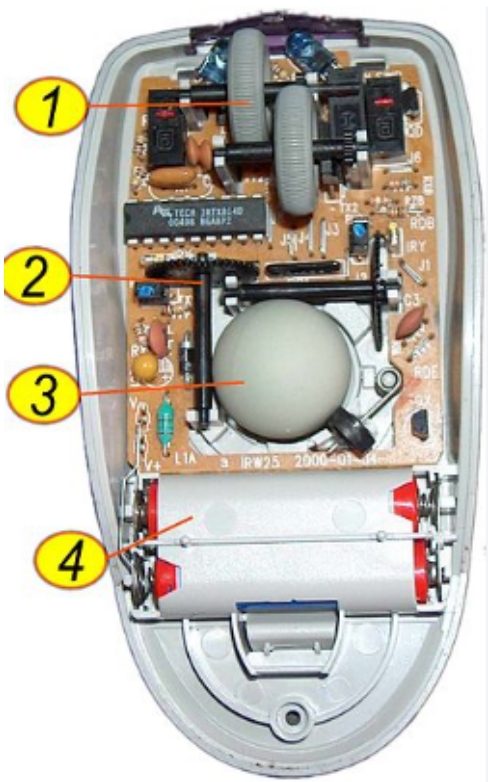
- Giá tiền
- Màu săc
- Động cơ

Method (Phương thức): Phương thức tác động lên chiếc xe

- Tăng tốc (đạp ga)
- Giảm tốc (Thắng xe)
- Phát ra còi
- Điều hướng (thẳng, trái,phải ...)

TÍNH ĐÓNG GÓI

Gom tất cả những thứ liên quan (thuộc tính và phương thức) vào chung một "cái hộp" là Class



- **class Mouse**
- **Thuộc tính**
  - Nút chuột trái
  - Nút chuột phải
- **Phương thức**
  - Nhấn chuột trái
  - Di chuyển chuột

MINH HỌA CÁCH KHỞI TẠO CLASS

```
def __init__(self, open_weather_data):
  main_data = open_weather_data.get('main', {})
  temp_k = main_data.get('temp','N/A')
  self.temperature_c = self.k_to_c(temp_k)
```

```
def k_to_c(self,temp_k):
  if isinstance(temp_k, (int, float)):
    return round(temp_k - 273.15, 1)
  return 'N/A'
```

```
def __str__(self):
  output += f"Nhiệt độ: {self.temperature_c}°C\n"
  return output
```

MỘT VÀI PHƯƠNG THỨC MẶC ĐỊNH CỦA LỚP

- Phương thức `__init___` là hàm đặc biệt để khởi tạo lớp. Sẽ được gọi khi tạo thể hiện của lớp
- Phương thức `__repr__()`: Phương thức này sẽ được gọi khi dùng hàm `print(...)` lên đối đối tượng
- Phương thức `__str__()`: Phương thức này sẽ được gọi khi dùng hảm `str(...)` lên đối tượng
- Ngoài ra còn nhiều phwuong thức đặc biệt khác: A Guide to Python's Magic Methods

LIVE CODE

- Bước 1: Tạo dữ liệu giả lập thời tiết

```
sample_data = {"main": {"temp": 298.15, "humidity": 80}}
```

- Bước 2: Tạo đối tượng thời tiết

```
weather_bmt = My_Weather(sample_data)
```

- Bước 3: Gọi phương thưc lấy thông tin

```
print(f"Nhiệt độ: {weather_bmt.temperature_c}°C")
```

BÀI TẬP THỰC HÀNH

Nhiệm vụ : **Tạo một Class tên là Sensor để mô tả một cảm biến**

- **Tên Class**: Sensor.
- **Khởi tạo với 3 tham số đầu vào**: name, value, unit.
- **Các thuộc tính**: self.name, self.value, self.unit.
- **Phương thức**: display(): In ra thông tin cảm biến theo định dạng, ví dụ: "Cảm biến Nhiệt độ: 35.5 °C".

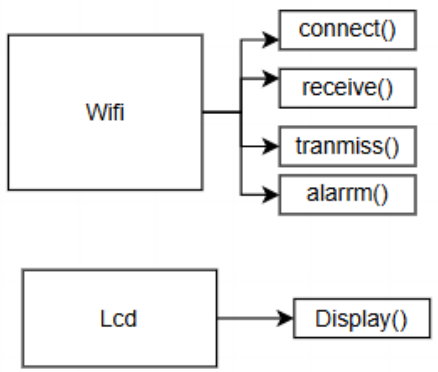
TIẾT 2: OPP - TÍNH KẾ THỪA

MỤC TIÊU

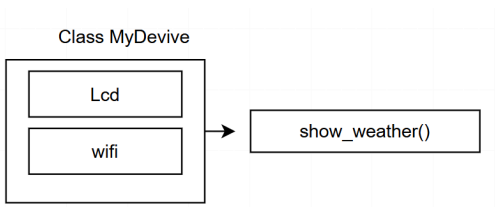
- Kiến thức:
  - Hiểu Kế thừa, Lớp Cha, Lớp Con.
  - Hiểu Overriding và cách dùng super().
- Kỹ năng:
  - Đọc hiểu và phân tích được cấu trúc kế thừa trong code dự án.
- Tư duy:
  - Hình thành tư duy tái sử dụng, tránh lặp lại code

TÌNH HUỐNG 1

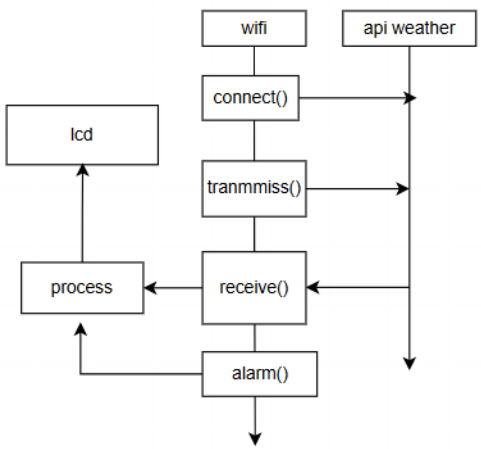
Wifi and LCD



MyDevice



Thiết kế



Làm sao ta có thể tái sử dụng module dựng sẵn?

TÌNH HUỐNG 2:

- Giả sử chúng ta viết đơn lẻ như sau

```
class FanData:
    def __init__(self, id, value, timestamp):
        self.id = id
        self.value = value
        self.timestamp = timestamp
```

```
class PumpData:
    def __init__(self, id, value, timestamp):
        self.id = id
        self.value = value
        self.timestamp = timestamp
```

Làm sao ta có thể tránh lặp điểm chung?

KHÁI NIỆM TÍNH KẾ THỪA

TÍNH KẾ THỪA

- Là cơ chế cho phép một Lớp (gọi là Lớp Con - Child Class) **thừa hưởng** lại toàn bộ thuộc tính và phương thức từ một Lớp khác (gọi là lớp Cha - Parent Class).
- Mục đích chính : Tái sử dụng code

GHI ĐÈ PHƯƠNG THỨC (METHOD OVERRIDING)

- Khi Lớp Con định nghĩa lại một phương thức đã có sẵn ở Lớp Cha, hành động đó gọi là ghi đè.
- Mục đích: Giúp Lớp Con có thể tùy biến, chuyên biệt hóa hành vi của mình.
- Ví dụ: Cùng là `__repr__` nhưng FanData sẽ hiển thị chữ "Fan", còn PumpData hiển thị chữ "Pump".

HÀM `supper()`

- Là một hàm đặc biệt dùng để gọi đến các phương thức của Lớp Cha từ bên trong Lớp Con.
- Rất hay được dùng trong `__init__` của Lớp Con để "nhờ" Lớp Cha thực hiện phần khởi tạo các thuộc tính chung.
- `__init___` là hàm đặc biệt để khởi tạo lớp.

MINH HỌA VỀ KẾ THỪA

- Khai báo lớp cha

```
class RelayData:
    def __init__(self, id, value,timestamp):
        self.id = id
        self.value = value
        self.timestamp = timestamp
    def __repr__(self):
        return f"RelayData id={self.id}, value={self.value}, timestamp={self.timestamp}"
```

- Khai báo lớp con

```
class FanData(RelayData):
    def __init__(self, id, value,timestamp):
        super().__init__(id,value,timestamp)
```

BÀI TẬP THỰC HÀNH

- Nhiệm vụ : Xây dựng lớp PumpData với yêu cầu sau
  - Tên Class: PumpData.
  - Kế thừa lớp RelayData
  - Nhận vào 3 thuộc tính: id, value, timestamp
  - Phương thức `__repr__()`: override phương thức `__repr__()` của lớp cha với định dạng

```
f"PumbData id=..., value=...,timestamp=..."
```

TIẾT 3: OPP TÍNH ĐA HÌNH

ĐẶT VẤN ĐỀ

- Bây giờ, hãy tưởng tượng chúng ta có một danh sách tất cả các thiết bị : Bơm , Quạt kế thừa từ lớp Relay
- Chúng ta muốn viết **MỘT chức năng duy nhất** để in trạng thái của từng thiết bị.
- Chẳng lẽ chúng ta phải viết code kiểm tra: if thiết bị là Quạt thì làm thế này, if thiết bị là Bơm thì làm thế kia?
- Để giải quyết ta dùng khái niệm tính đa hình

KHÁI NIỆM TÍNH ĐA HÌNH (POLYMORPHISM)

- "Poly" nghĩa là "nhiều", "Morph" là "hình thái". Đa hình là khả năng một hàm hoặc một đối tượng có thể xử lý nhiều kiểu đối tượng khác nhau như thể chúng là một.
- Nguyên tắc cốt lõi: "Chỉ cần 'nói chuyện' với Lớp Cha, có thể điều khiển được tất cả các Lớp Con."

MINH HỌA TÍNH ĐA HÌNH

```
# Giả sử đã có các class RelayData, FanData, PumpData
fan1 = FanData(1, 'ON', '2025-10-23 19:00:00')
pump1 = PumpData(2, 'OFF', '2025-10-23 19:05:00')
```

```
# Xây dựng hàm báo cáo trạng thái thiết bị
def display_status(device:RelayData):
    print("--- Báo cáo trạng thái thiết bị ---")
    print(device)
```

```
# Quan sát kết quả khi gọi hàm
display_status(fan1)
display_status(pump1)
```

BÀI TẬP THỰC HÀNH

- Tạo thêm class HeaterData kế thừa từ **RelayData**.
- Thêm thuộc tính temperature (nhiệt độ hiện tại của thiết bị).
- Ghi đè phương thức `__repr__()` để hiển thị: "Heater {id}, trạng thái: {status}, nhiệt độ: {temperature}".
- Gọi hàm `display_status()` ở trên để kiểm tra

THANK YOU