

# Dataclass

`dataclass` là một **decorator** được thêm vào từ **Python 3.7** trong module `dataclasses`. Nó giúp bạn **tự động tạo ra các phương thức** như:

- `__init__()` - hàm khởi tạo
- `__repr__()` - biểu diễn đối tượng
- `__eq__()` - so sánh bằng
- `__hash__()` - dùng cho tập hợp, từ điển
- `__lt__()`, `__le__()` ... nếu bạn bật `order=True`

## Cách dùng cơ bản:

```
from dataclasses import dataclass
```

```
@dataclass
class Person:
    name: str
    age: int
```

## Python sẽ tự động sinh ra:

```
def __init__(self, name: str, age: int):
    self.name = name
    self.age = age
```

## Ví dụ

```
from dataclasses import dataclass

@dataclass
class Product:
    name: str
    price: float
    quantity: int = 1 # giá trị mặc định

product = Product(name="Laptop", price=1500.0)
print(product) # Product(name='Laptop', price=1500.0, quantity=1)
```

## Thêm cấu hình cho dataclass:

```
@dataclass(order=True, frozen=True)
class Point:
    x: int
    y: int
```

Ý nghĩa:

- `order=True`: tự động tạo các phép so sánh `<`, `<=`, `>`, `>=`
- `frozen=True`: **đóng băng** object - giống `immutable`, không thể thay đổi giá trị

```
p1 = Point(1, 2)
p2 = Point(2, 2)
print(p1 < p2) # True

p1.x = 10 # ❌ Lỗi vì class bị frozen
```

Vì bạn đã khai báo `frozen=True`, tức là "đóng băng" đối tượng. Python **không cho thay đổi bất kỳ thuộc tính nào** sau khi tạo.

Nếu bạn thử gán `p1.x = 10`, bạn sẽ gặp lỗi kiểu:

```
dataclasses.FrozenInstanceError: cannot assign to field 'x'
```

## Dùng `field()` để tùy chỉnh thêm

```
from dataclasses import dataclass, field

@dataclass
class User:
    username: str
    password: str = field(repr=False) # ẩn khi in ra
```

- `field(repr=False)`: không hiển thị trong `__repr__()`
- `field(default=...)`: gán giá trị mặc định
- `field(init=False)`: không cho truyền trong `__init__`

## So sánh với cách viết class bình thường

Cách viết dài dòng:

```
class Book:
    def __init__(self, title, price):
        self.title = title
        self.price = price

    def __repr__(self):
        return f"Book(title={self.title}, price={self.price})"
```

## Viết với dataclass:

```
@dataclass
class Book:
    title: str
    price: float
```

### `__post_init__`: logic sau khi khởi tạo

```
@dataclass
class State:
    name: str
    mood: str

    def __post_init__(self):
        if self.mood not in ["happy", "sad"]:
            raise ValueError("Mood must be 'happy' or 'sad'")
```

## Trường hợp hợp lệ:

```
state = ConversationState(name="Lan", mood="happy")
print(state)
# Output: ConversationState(name='Lan', mood='happy')
```

## Trường hợp không hợp lệ:

```
state = ConversationState(name="Lan", mood="angry")
# Output: ValueError: Mood must be 'happy' or 'sad'
```

- `__post_init__()` là một **phương thức đặc biệt** trong `dataclass`, tự động chạy **sau khi hàm `__init__()` hoàn thành**.
- Đây là nơi thích hợp để **kiểm tra tính hợp lệ** của các giá trị đầu vào hoặc thực hiện các thao tác bổ sung với dữ liệu.

## Chuyển đổi tượng thành từ điển (dict)

### Cách dùng asdict

```
from dataclasses import dataclass, asdict
```

```
@dataclass
```

```
class Person:
```

```
    name: str
```

```
    age: int
```

```
p = Person("Nam", 25)
```

```
# Chuyển thành dict
```

```
data = asdict(p)
```

```
print(data)
```

```
## Kết quả
```

```
{'name': 'Nam', 'age': 25}
```

**Tác giả: Đỗ Ngọc Tú**

**Công Ty Phần Mềm VHTSoft**

---

Phiên bản #3

Được tạo 22 tháng 4 2025 04:26:13 bởi Đỗ Ngọc Tú

Được cập nhật 22 tháng 4 2025 04:50:10 bởi Đỗ Ngọc Tú