O que é uma Dataclass?

Uma dataclass é uma classe comum que é **decorada** com @dataclass (importada do módulo dataclasses). O decorador inspeciona as **anotações de tipo** das variáveis de classe e, automaticamente, gera métodos especiais (dunder methods) comumente necessários para classes de dados.

O uso de dataclasses reduz significativamente a quantidade de código clichê (boilerplate code) que você teria que escrever manualmente.

Métodos Gerados Automaticamente

O decorador @dataclass gera automaticamente, por padrão, os seguintes métodos:

- **__init__**: Um construtor que aceita argumentos para cada campo anotado e atribui esses valores às instâncias.
- **__repr__**: Um método para obter uma representação "amigável" e informativa da instância (útil para depuração e logs).
- **__eq__**: Um método para comparar duas instâncias da dataclass com base nos valores de seus campos.

Exemplos de Dataclass

Aqui estão alguns exemplos práticos de como usar dataclasses, contrastando com a forma "tradicional" de escrever classes:

Exemplo 1: Ponto (Point)

Imagine uma classe simples para representar um ponto 2D.

Forma Tradicional (sem dataclass)

```
python class Point: def __init__(self, x: int, y: int): self.x = x self.y = y def __repr__(self): return f"Point(x={self.x}, y={self.y})"
```

Usando Dataclass

```
python from dataclasses import dataclass @dataclass class Point: x: int y: int
```

```
p = Point(x=10, y=20)
print(p) # Saída gerada pela dataclass: Point(x=10, y=20)
```

Exemplo 2: Item de Inventário (InventoryItem)

Um exemplo com um valor padrão para um dos campos:

```
from dataclasses import dataclass

@dataclass
class InventoryItem:
    """Classe para rastrear um item no inventário."""
    name: str
    unit_price: float
    quantity_on_hand: int = 0  # Valor padrão de 0

    def total_cost(self) -> float:
        """Método adicional para calcular o custo total."""
        return self.unit_price * self.quantity_on_hand

# Uso
caneta = InventoryItem("Caneta", 1.50)  # Usa o valor padrão de 0 para quantity_
caderno = InventoryItem("Caderno", 5.00, 20)
print(caneta)
print(f"Custo total do caderno: {caderno.total_cost()}")
```

Exemplo 3: Imutabilidade (Frozen Dataclass)

Você pode tornar uma dataclass **imutável** (os valores dos campos não podem ser alterados após a criação) definindo o parâmetro frozen=True no decorador.

```
from dataclasses import dataclass

@dataclass(frozen=True)
class ImmutablePosition:
    name: str
    lon: float
    lat: float

pos = ImmutablePosition('Oslo', 10.8, 59.9)
# pos.name = 'Stockholm' # Isso levantaria um erro (FrozenInstanceError)
```