Clase 010 — OOP básico, dataclasses, herencia

Parte: 0 — Prerrequisitos · Fuente: Ramalho, Fluent Python 2e — caps. 5 (Data Class Builders) y 14 (Inheritance) · Python Tutorial cap. 9. · Duración estimada: 90 min.

Clase 010 — OOP básico, dataclasses, herencia

> *Parte: 0 — Prerrequisitos · Fuente: Ramalho, Fluent Python 2e — caps. 5 (Data Class Builders) y 14 (Inheritance) · Python Tutorial cap. 9. · Duración estimada: 90 min.*

Objetivo

Que el alumno escriba clases cuando aportan (no por hábito Java), use @dataclass para records sin boilerplate, entienda herencia con criterio (preferir composición), y conozca los métodos dunder más usados (__repr__, __eq__, __lt__, __len__).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Definir clases con __init__, atributos de instancia y métodos.
2. Usar @dataclass para records inmutables/mutables sin escribir __init__/__repr__/__eq__.
3. Heredar y sobreescribir métodos con super().
4. Implementar dunders esenciales: __repr__, __str__, __eq__, __lt__, __len__, __iter__.
5. Decidir entre clase, dataclass o NamedTuple según el caso.

Temas

#	Tema	Por qué importa
1	Clase mínima: __init__ + atributos + métod	El bloque básico.
2	@dataclass(frozen=True)	Records inmutables sin boilerplate.
3	Herencia + super()	Reutilizar implementación de la clase base
4	Composición > herencia	"Has-a" generalmente mejor que "is-a".
5	Métodos dunder	Integran tu clase con len(), ==, repr(), s
6	dataclass vs NamedTuple vs TypedDict	Elegir según necesidad de mutabilidad/comp

Definiciones y características

Clase / instancia

: Una clase es una plantilla (class Punto:); una instancia es un objeto concreto (p = Punto(3, 4)). __init__ se llama al crear la instancia. self es la convención para referirse a la instancia dentro de los métodos.

Método dunder ("magic method")

: Método con doble underscore (__init__, __repr__, __eq__, __lt__, __len__, __iter__, __add__). Python los invoca implícitamente con sintaxis especial (len(obj) → obj.__len__()).

@dataclass

: Decorador que genera __init__, __repr__, __eq__ automáticamente desde las anotaciones de tipo de la clase. Reduce boilerplate. Con frozen=True la hace inmutable y hashable.

Herencia

: class B(A) — B hereda atributos y métodos de A; puede sobreescribirlos. super() invoca al método de la clase padre. Múltiple herencia existe pero se complica (MRO).

Composición

: "B tiene un A" (atributo) en vez de "B es un A" (herencia). Generalmente preferible: menos acoplamiento, sin problemas de herencia múltiple/diamante.

Polimorfismo

: Distintas clases responden al mismo método con comportamiento distinto (Animal.hablar() → 'guau' o 'miau' según la subclase). Permite tratar instancias heterogéneas uniformemente.

NamedTuple vs dataclass vs TypedDict

: NamedTuple: tupla con nombres, inmutable, hashable, sin métodos custom. dataclass: clase con boilerplate auto, mutable por default (mejor con métodos). TypedDict: dict con esquema (estructural, no nominal).

Dataset / recursos

Sintético: lista de objetos Punto y Estudiante. Sin descarga.

Ejercicios

1. Clase Punto. Define Punto(x, y) con __repr__, __eq__, distancia al origen y __add__ para sumar puntos.

2. Dataclass Estudiante. @dataclass con nombre, notas: list[float], método promedio(). Crea 3 instancias, ordena por promedio.

3. Frozen Vector. @dataclass(frozen=True) para un vector 2D inmutable. Intenta modificar un atributo y observa la excepción.

4. Herencia. Animal con hablar() → 'genérico'. Perro(Animal) que sobreescribe a 'guau'. Gato(Animal) a 'miau'.

5. Composición. Coche que tiene un Motor (composición) en vez de heredar de Motor. Justifica por qué.

Homework verificable

Notebook con: (a) Punto con 4 dunders y tests; (b) @dataclass Estudiante con sort por promedio; (c) @dataclass(frozen=True) Vector que demuestra inmutabilidad lanzando excepción; (d) jerarquía Animal → Perro/Gato con polimorfismo (lista mixta llamando hablar()).

Criterio de aceptación: Las 4 clases pasan tests; frozen=True lanza FrozenInstanceError al asignar.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Olvidé self en un método y el error es con	Cualquier método de instancia recibe self
@dataclass con field mutable default rompe	@dataclass class X: items: list = [] lanza
__eq__ definido pero __hash__ rompe	Definir __eq__ sin __hash__ hace la clase
super().__init__(...) olvidado en subclase	Atributos del padre quedan sin inicializar
Modifico atributo y otra instancia también	Asignaste un mutable como class attribute,

Preguntas frecuentes

¿Cuándo necesito OOP en data science?

Menos de lo que crees. Para análisis exploratorio, funciones + dicts/dataclasses bastan. Necesitas clases cuando hay: estado mutable complejo (modelos sklearn), polimorfismo (varios algoritmos misma interfaz), o frameworks que lo exigen (PyTorch nn.Module).

¿@dataclass o NamedTuple o pydantic?

NamedTuple: record inmutable simple, sin validación. dataclass: record con métodos opcionales. pydantic (no en stdlib): cuando además quieres validación de tipos en runtime, parsing desde JSON, etc. (lo verás en MLOps).

¿Composición > herencia siempre?

Como regla. Usa herencia solo cuando is-a sea genuino (PerroLabrador is-a Perro is-a Animal). Para has-a (Coche tiene un Motor), composición. Para reutilizar comportamiento sin jerarquía, considera mixins o protocols.

¿property y getters/setters Java-style?

En Python no escribes getNombre()/setNombre(). Usa atributo público (self.nombre = ...). Si después necesitas lógica, conviertes a @property sin cambiar el caller. Es la magia.

¿Cuándo __slots__?

Optimización: define los atributos permitidos y ahorra memoria (~50%) al no usar __dict__ por instancia. Útil solo en clases con millones de instancias. Costo: pierde herencia múltiple y dinamismo.

Referencias

- Ramalho, Fluent Python 2e — caps. 5, 11, 14.
- dataclasses docs
- Python Tutorial — Classes

Siguiente clase

Clase 011 — pathlib, lectura y escritura de archivos

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
from dataclasses import dataclass, field, FrozenInstanceError
from math import sqrt
from typing import NamedTuple
```

Archivos complementarios

- notebook.ipynb