Clase 013 — Type hints y mypy

Parte: 0 — Prerrequisitos · Fuente: Fluent Python 2e cap. 8 (Type Hints in Functions) · typing docs · mypy docs. · Duración estimada: 75 min.

Clase 013 — Type hints y mypy

> *Parte: 0 — Prerrequisitos · Fuente: Fluent Python 2e cap. 8 (Type Hints in Functions) · typing docs · mypy docs. · Duración estimada: 75 min.*

Objetivo

Que el alumno anote tipos en sus funciones y dataclasses — no por dogma, sino porque permiten que el IDE autocomplete bien, que mypy detecte bugs antes de runtime, y que el lector entienda la intención. Tipos como documentación verificable.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Anotar funciones con tipos en parámetros y retorno (def f(x: int) -> str).
2. Usar tipos compuestos: list[int], dict[str, float], tuple[int, str], Optional[X], X | None.
3. Definir tipos personalizados con TypeAlias y Protocol (structural typing).
4. Ejecutar mypy sobre código y interpretar sus errores.
5. Reconocer cuándo type hints aportan (APIs públicas, data classes) y cuándo no (notebooks exploratorios).

Temas

#	Tema	Por qué importa	
1	Sintaxis básica: x: int, -> bool	Solo anotaciones — no afectan ru	
2	Tipos compuestos modernos (3.9	Sin from typing import List.	
3	Optional[X] y `X	None` (3.10+)	Cuando algo puede ser None.
4	Literal, TypedDict, Protocol	Tipos avanzados útiles.	
5	mypy: instalar y correr	Static type checker.	
6	reveal_type(x) y # type: ignore	Diagnóstico y escape hatch.	
7	Cuándo SÍ y cuándo NO	API pública sí; notebook explorato	

Definiciones y características

Type hint (annotation)

: Anotación de tipo en signature o variable: def f(x: int) -> str. Python NO verifica en runtime — son metadata leída por IDE/linter/mypy. Disponibles en f.__annotations__.

Optional[X] / X | None

: Indica que el valor puede ser X o None. Optional[X] ≡ Union[X, None] ≡ X | None (PEP 604, 3.10+). Usa la sintaxis con | en código nuevo.

TypedDict

: Esquema para diccionarios: class PersonaDict(TypedDict): nombre: str; edad: int. Permite tipear dicts que

vienen de JSON/API sin convertirlos a dataclass.

Literal

: Restringe a un conjunto de valores: Literal['asc', 'desc'] solo acepta esas 2 strings. Útil para flags, modos, enums simples.

Protocol (structural typing)

: Define interfaz por estructura (duck typing tipado): class TienePromedio(Protocol): def promedio(self) -> float: .... Cualquier clase con .promedio() la satisface, sin heredarla.

mypy

: Static type checker oficial. Lee tu código, sigue las anotaciones, reporta inconsistencias antes de ejecutar. Modo --strict lo hace exigente; recomendado para librerías.

Dataset / recursos

Funciones de ejemplo en el notebook. Sin descarga.

Ejercicios

1. Anota una función. Toma una función de los ejercicios de clase 008 (sin tipos) y anótala completa.

2. Optional vs default. Distingue def f(x: int = 0) (default 0) de def f(x: int | None = None) (puede no haber valor).

3. TypedDict. Define class PersonaDict(TypedDict) con nombre: str, edad: int. Úsala como tipo de un parámetro.

4. Corre mypy. Instala mypy, créate un archivo con un bug de tipo intencional (def f(x: int) -> str: return x + 1) y corre mypy archivo.py. Lee y explica el error.

5. Protocol. Define class TienePromedio(Protocol) con método promedio() -> float. Acepta cualquier clase que lo implemente (duck typing tipado).

Homework verificable

Repo con un módulo analytics.py (5+ funciones completamente anotadas), pyproject.toml que incluye mypy en [tool.mypy] con strict = true, y screenshot/log de mypy analytics.py sin errores.

Criterio de aceptación: mypy --strict corre sin errores ni warnings. Tipos consistentes y precisos.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar	
from typing import List da deprecation war	Python 3.9+ usa lowercase: list[int] en ve	
Optional[int] = 0 confunde	Optional[int] admite None. Si tu default e	None = None` (admite None).
mypy se queja "Cannot find module 'libreri	Lib sin type stubs publicados. Fix: pip in	
Anoté pero mypy no encuentra errores	mypy no se llamó. Fix: mypy archivo.py. No	
reveal_type(x) no existe en runtime	Es exclusivo de mypy: lo lees como output	

Preguntas frecuentes

¿Tipo hints son obligatorios?

No — Python no los verifica. Pero son fuertemente recomendados en: APIs públicas (funciones que importan otros), librerías, código compartido. En notebooks exploratorios, casi nunca aportan.

¿Slow my code los type hints?

No — son metadata, no impactan runtime. from __future__ import annotations además las hace lazy (strings, evaluadas solo si introspeccionas).

¿mypy en CI: estricto o permisivo?

Empieza permisivo (sin --strict), arregla lo obvio, luego activa --strict gradualmente. Si arrancas estricto en un repo viejo, te ahogas en errores y termina ignorado.

¿Y si no sé qué tipo poner?

Any (de typing) es válido — desactiva el check para ese caso. Mejor que mentir con un tipo incorrecto. Convención: comentario # TODO: tipo correcto para revisión futura.

¿Pydantic vs dataclass + type hints?

dataclass + hints: tipos solo a nivel mypy, no en runtime. Pydantic: valida en runtime (parsing de JSON con coerción y errores legibles). Para input externo (API, config) → Pydantic. Para internal record → dataclass.

Referencias

- Ramalho, Fluent Python 2e — cap. 8.
- typing docs
- mypy docs
- PEP 484 — Type Hints
- PEP 604 — X | Y syntax

Siguiente clase

Clase 014 — NumPy: tipos, creación, atributos

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
from typing import Optional, Literal, TypedDict, Protocol, TypeAlias
from dataclasses import dataclass
```

Archivos complementarios

- notebook.ipynb