La Clase `Token` - Empaquetando la Información

Objetivo: Comprender el concepto de encapsulación de datos en Python. Analizar cómo los dataclasses ofrecen una solución moderna y concisa para crear objetos de datos como los tokens.

Necesidad de un Contenedor Organizado

- **Repaso de lo visto previamente:** Se sabe que `'AVANZA'` es un `'INSTRUCCION'` y `100` es un `'NUMERO'`. Pero esta información está dispersa.
- **El Problema:** Se necesita agrupar en una sola "caja": el tipo, el valor literal y la línea donde se encuentra.
- **Estructuración de Datos.** Se pasa de datos primitivos (strings, números) a necesitar una estructura de datos compuesta.

`@dataclass` - El Poder de la Decoración en Python

- **Alternativas en Python y sus Contras:**
    - `tuple`: `('INSTRUCCION', 'AVANZA', 3)`. Es eficiente pero ilegible. ¿Qué significa el elemento en el índice 2?
    - `dict`: `{'type': 'INSTRUCCION', 'value': 'AVANZA', 'line': 3}`. Es legible pero verboso y propenso a errores de tipeo en las claves.
    - Clase tradicional (`class Token: ... __init__ ... __repr__ ...`): Funcional, pero requiere escribir mucho código repetitivo (boilerplate).
- **Introducción al Decorador `@dataclass`:**
    - ¿Qué es un decorador en Python?: Una función que modifica o mejora otra función o clase. `@dataclass` "decora" la clase `Token` y le añade potencia.
    - **¿Qué le brinda a la clase Token?** Automáticamente genera:
        - `__init__(self, type, value, line)`: El constructor.

        - `__repr__(self)`: La representación en string, que es lo que permite hacer `print(token)` y obtener una salida tan clara y útil para depuración.

        - `__eq__(self, other)`: La comparación, para poder hacer `token1 == token2`.

- **Análisis del Código `Token`:**

```
@dataclass
class Token:
    type: TokenType
    value: Any
    linea: int
```

    - **Tipado Dinámico (`value: Any`):** Python es flexible. El campo `value` puede contener un string, un float o `None` sin problema. Esto simplifica el código del lexer, pero pasa la responsabilidad de la verificación de tipos al consumidor (el parser). Las "anotaciones de tipo" (`: str`, `: any`) son opcionales pero son una excelente práctica para la legibilidad.

    - **Inmutabilidad (Opcional):** Mencionar que se podría hacer el dataclass inmutable con `@dataclass(frozen=True)`, una buena práctica para este caso de uso.

Actividad Práctica - Visualizando la Estructura

- Dada la línea `GROSOR_LAPIZ 5.5`, escribir las instancias de `Token` que se generarían.

    - `Token(type='INSTRUCCION', value='GROSOR_LAPIZ', linea=10)`

    - `Token(type='NUMERO', value=5.5, linea=10)`

- Lo interesante es darse cuenta cómo el tipo de `value` cambia (`str` en el primer token, `float` en el segundo), lo cual es manejado naturalmente por el tipado dinámico de Python.

Resumen y Avance

- **Resumen:** Un `Token` es el paquete de información fundamental. En Python, un `dataclass` es la herramienta idiomática, moderna y concisa para crear esta

estructura, brindando legibilidad y funcionalidades automáticas con mínimo esfuerzo.

- **Avance a la Siguiente Sesión:** “Teniendo listas las categorías y el contenedor. Ahora ahora se debe construir la fábrica: el `Lexer`”