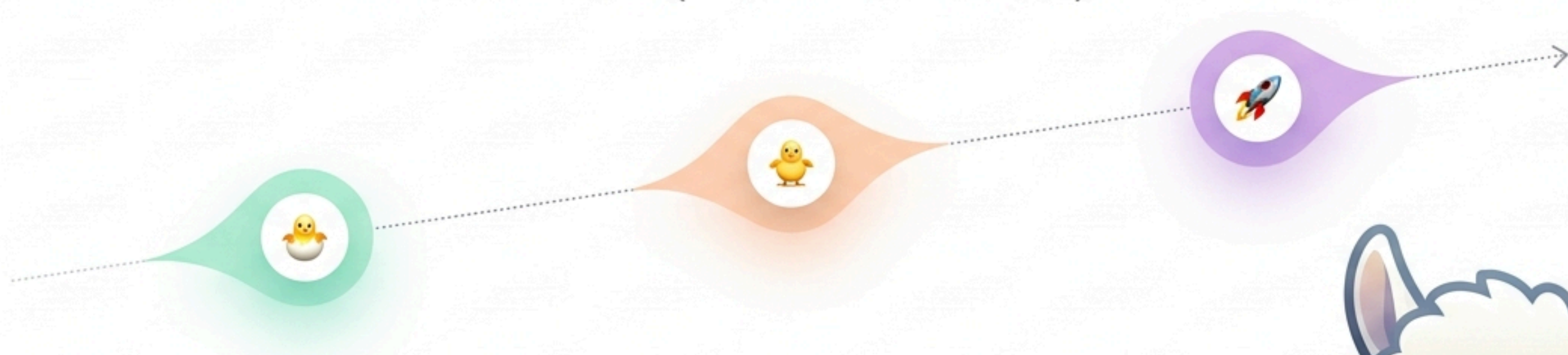


Python: De Principiante a Experto 🦙🐍

Trucos, buenas prácticas y código "Pythonic" que debes conocer (sin teoría aburrida).



Desliza para dejar de escribir código como si estuvieras en 1999.



Nivel 🐤 Principiante

Adiós a los contadores manuales

enumerate()

Usa `enumerate()` para recorrer colecciones y obtener el índice automáticamente sin variables de control manuales.

```
i = 0
for item in lista:
    print(i, item)
    i += 1
```



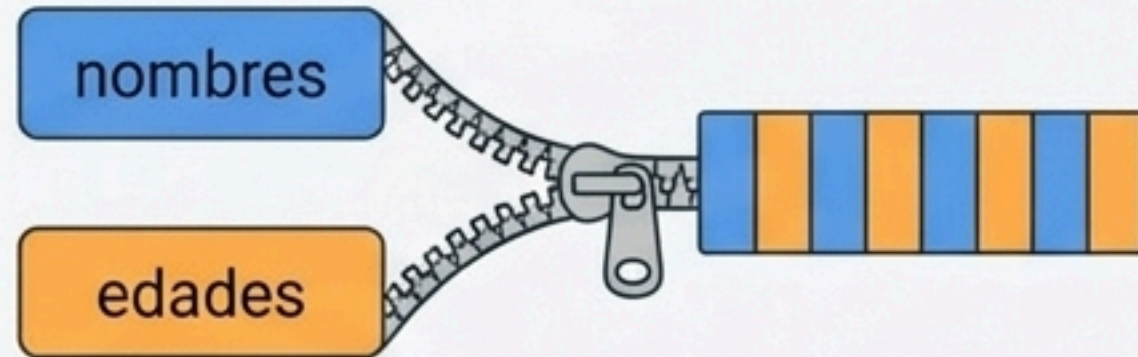
```
for i, item in enumerate(lista):
    print(i, item)
```



¿Aún sumas `i += 1` a mano?
Deja que Python trabaje por ti.

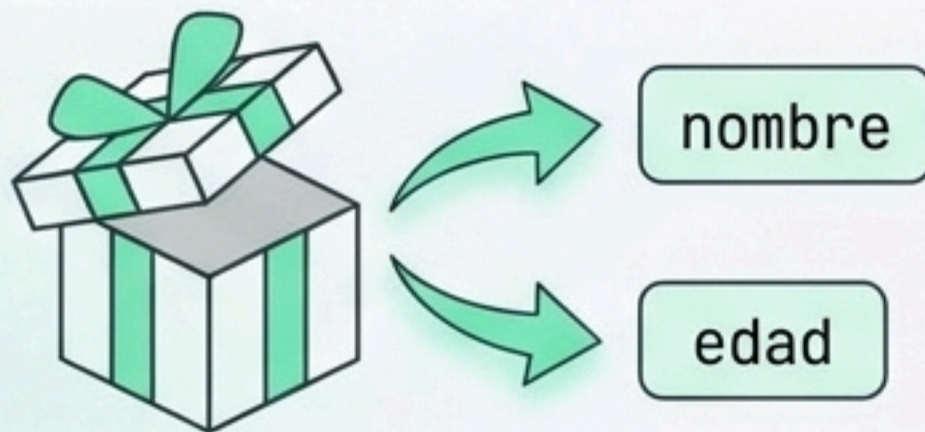
Nivel 🐣 Principiante

Colecciones en paralelo y Extracción directa



Itera múltiples listas a la vez. Añade `strict=True` para detectar asimetrías.

```
for n, e in zip(nombres, edades, strict=True):
```



Extrae estructuras directamente en variables descriptivas.

```
nombre, edad = ["Brais", 35]
```

Dos listas. Un paseo. Cero índices.



Nivel 🐣 Principiante

Slicing & Diccionarios Seguros



Manipula secuencias con pura elegancia matemática: [inicio:fin:paso]

```
reverso = lista[::-1]  
pares = lista[0:10:2]
```



Evita el temido KeyError. Extrae valores con un salvavidas por defecto.

```
rol = user.get("rol", "invitado")
```

Si no existe la llave en el diccionario, no rompas el programa. Solo dame un 'invitado'.



Nivel 🐥 Intermedio

Filtros y Transformaciones Elegantes

Comprensiones de listas

Condensa bucles y filtros simples en una sola línea altamente optimizada.



Before



```
pares = []  
for n in numeros:  
    if n % 2 == 0:  
        pares.append(n)
```



After



```
pares = [n for n in numeros  
          if n % 2 == 0]
```

Una línea es elegante. Pero no intentes meter TODO Python ahí. Si necesitas 3 saltos de línea para entenderla, vuelve al for clásico.



Nivel 🐥 Intermedio

Evaluaciones Semánticas y Rápidas

any(), all() & Expresiones Ternarias

Obtén respuestas directas a condiciones universales sin escribir loops manuales.

¿Hay alguno?

```
if any(nota < 5  
    for nota in notas):
```

¿Todos cumplen?

```
if all(n > 0  
    for n in numeros):
```

Asignación directa

```
estado = "Activo"  
if user.is_logged else "Inactivo"
```

Yo no reviso uno por uno.
Mis poderes de any y all lo
ven todo al instante.

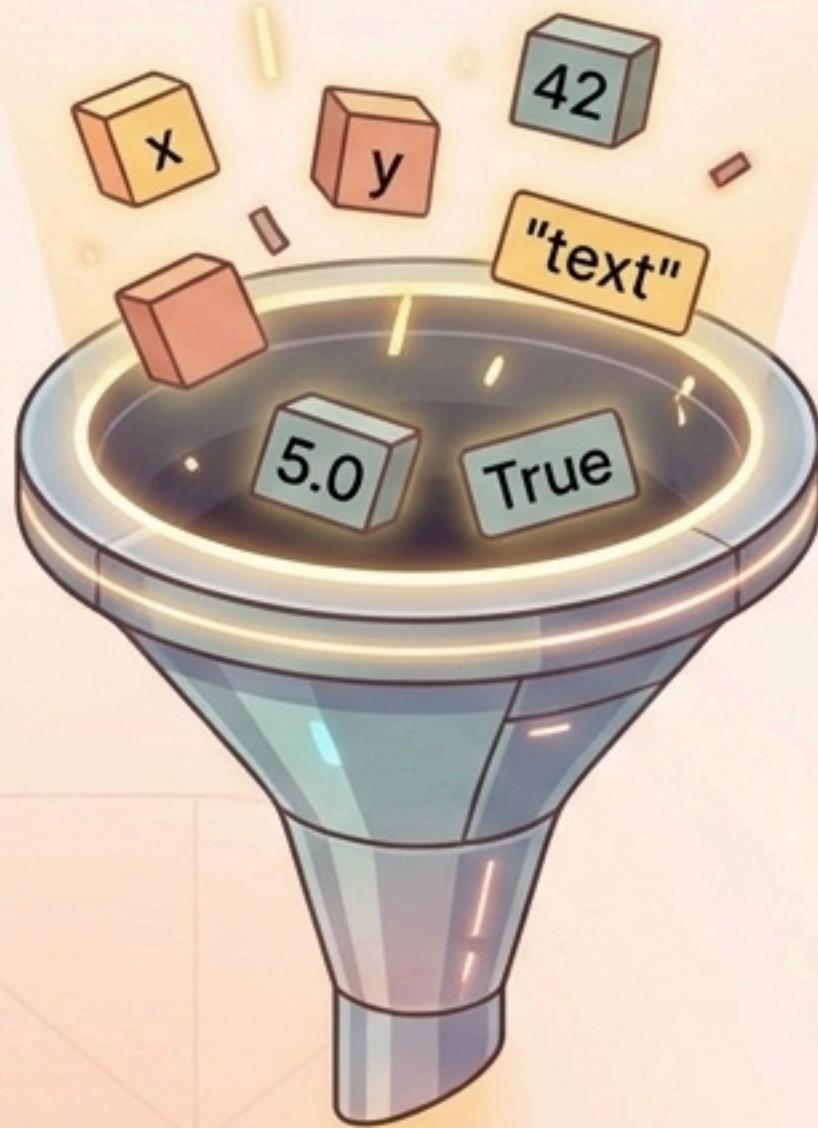


Nivel 🐣 Intermedio

Flexibilidad Total en Argumentos

***args y **kwargs**

Permite que tus funciones acepten un número dinámico de argumentos. Ideal para envoltorios (wrappers) y herencia.



```
def procesar(*args, **kwargs):  
    print(f"Lista posicional: {args}")  
    print(f"Diccionario nombrado: {kwargs}")
```

()



{ }



¿No sabes cuántos datos van a llegar a la función? ¡Tú échalos todos, yo los organizo!



Nivel 🐥 Intermedio

El Flujo Perfecto de Errores y Recursos



Gestor de contexto (Context Manager):

```
with open("archivo.txt") as f:  
    data = f.read()
```

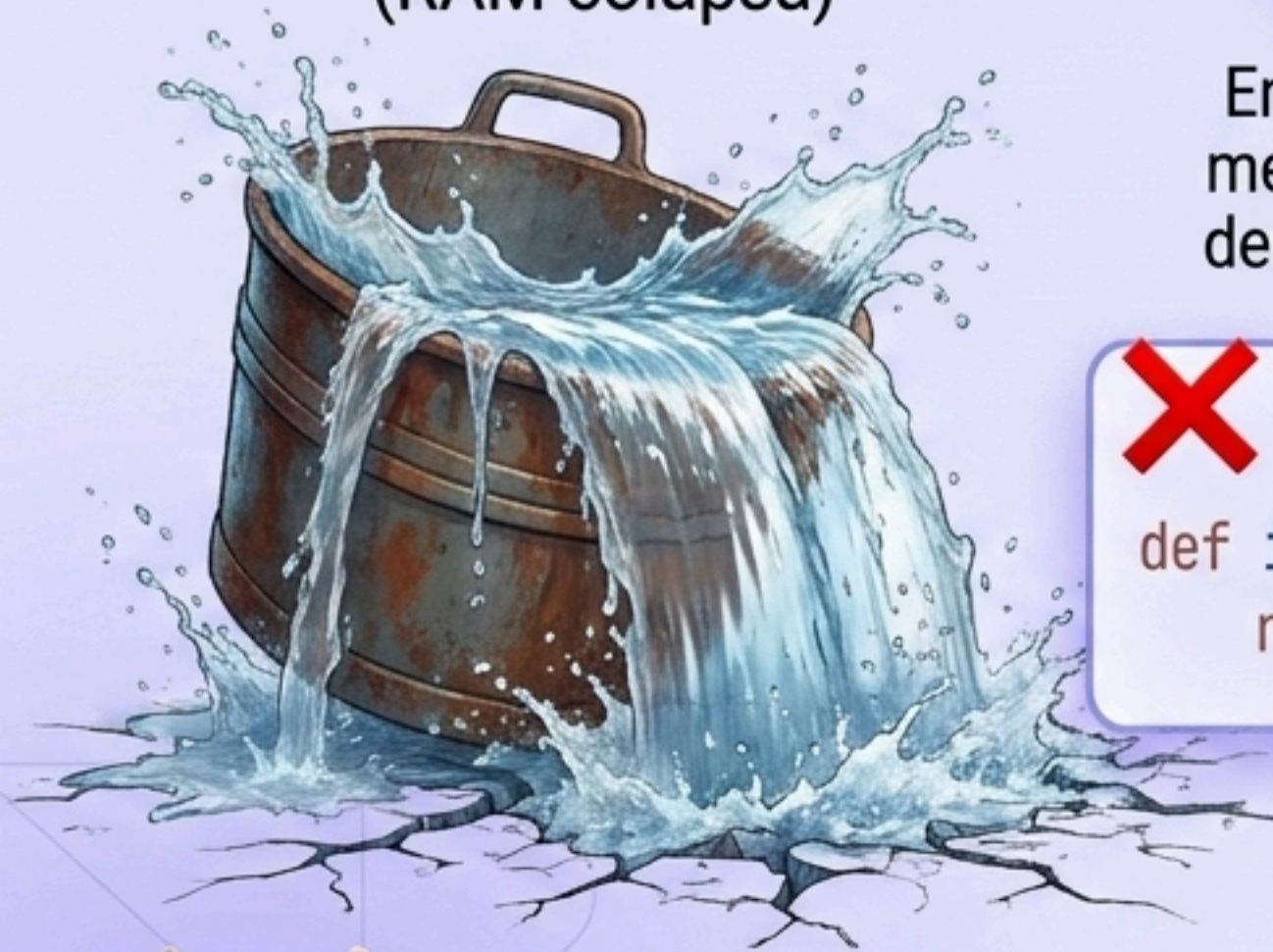


El bloque with es como un buen invitado: siempre cierra la puerta de los recursos al salir, incluso si hay un incendio.

Nivel 🚀 Avanzado

Generando bajo demanda (Adiós RAM llena)

Listas
(RAM colapsa)



Generadores con yield

En lugar de cargar millones de datos en la memoria RAM con **return**, devuélvelos bajo demanda según se necesiten usando **yield**.



```
def infinitos():  
    return [1, 2, 3, ...] # Colapsa la memoria
```

Generadores
(Control Total)



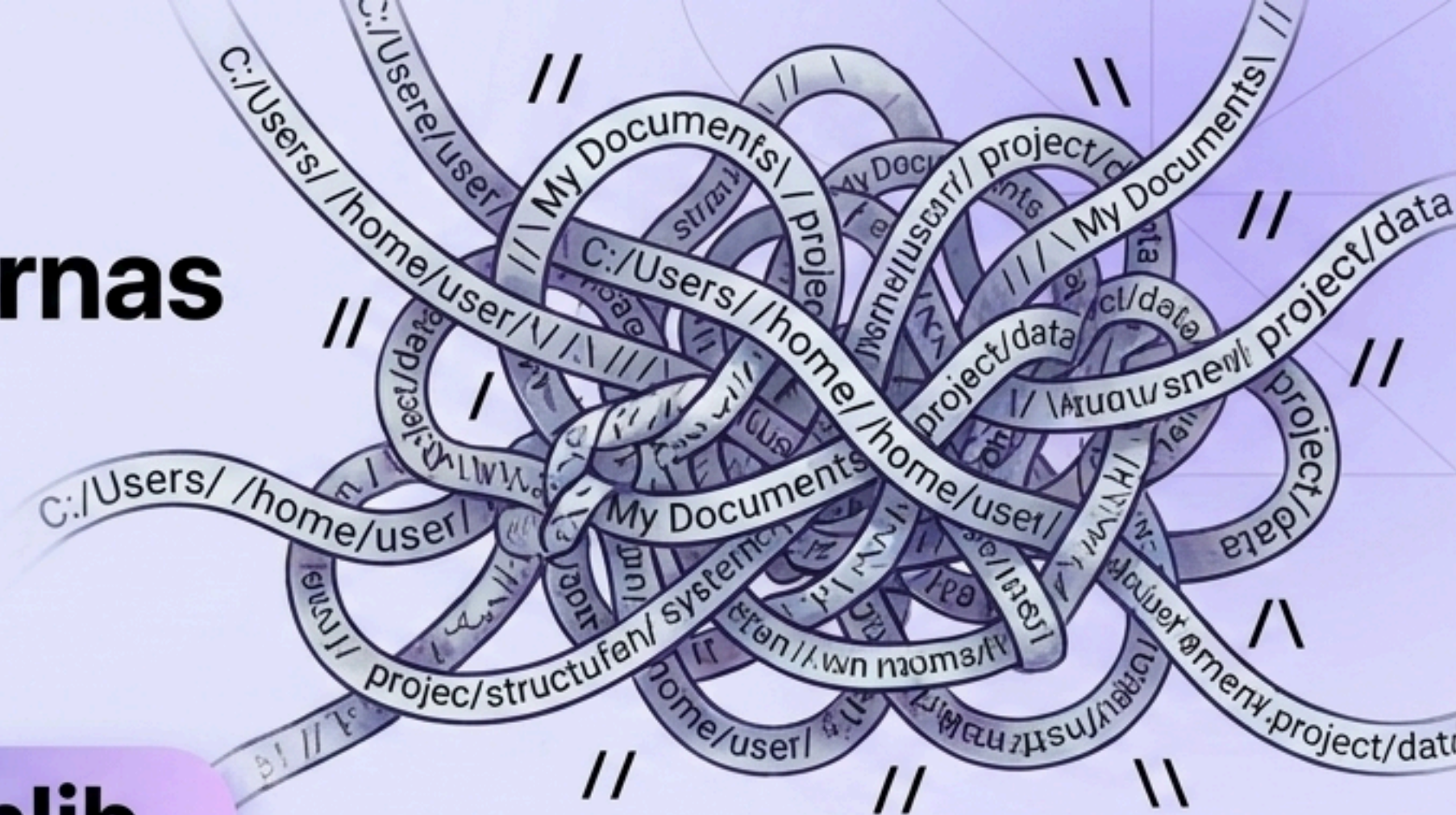
```
def infinitos():  
    yield numero # Entrega uno a uno
```



No intentes comerte el pastel entero de un bocado. Pide porciones con **yield**.

Nivel 🚀 Avanzado

Rutas Inteligentes y Modernas



```
ruta = directorio + "/" + "archivo.txt"
```

pathlib

Manipula rutas del sistema como objetos robustos, no como simples cadenas de texto. Funciona a la perfección en cualquier sistema operativo sin romper rutas.



```
from pathlib import Path  
ruta = Path(directorio) / "archivo.txt"
```



Unir rutas correctamente entre distintos sistemas operativos.

Unir rutas concatenando strings es jugar a la ruleta rusa con los sistemas operativos.

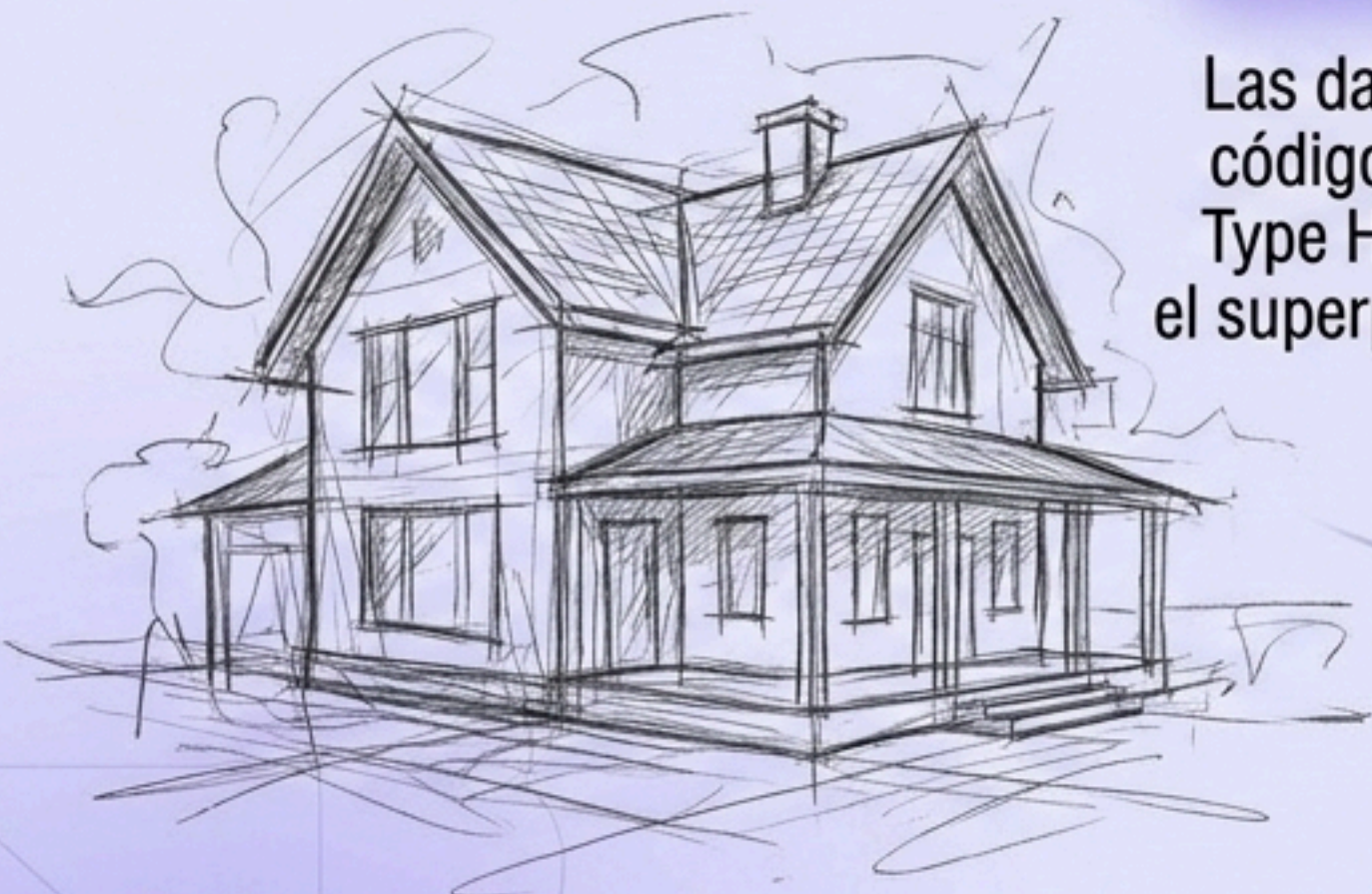
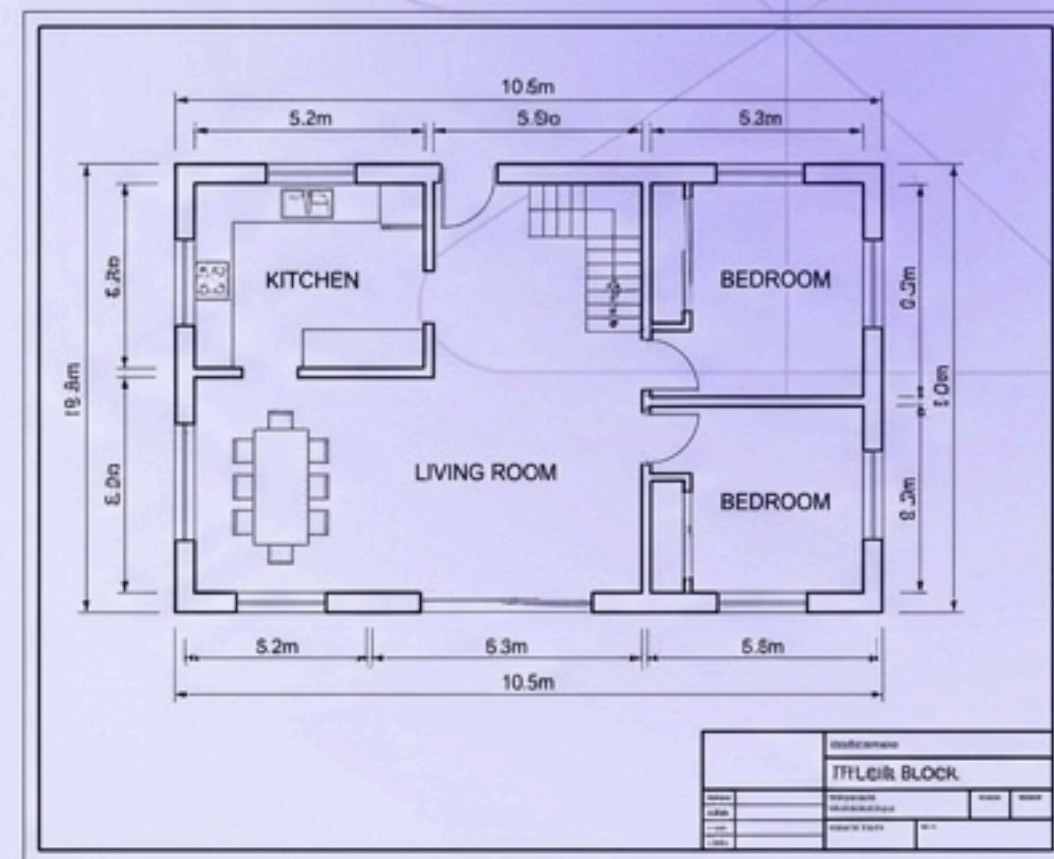


Nivel 🚀 Avanzado

Clases Limpias y Código Tipado

dataclasses & Type Hints

Las dataclasses eliminan automáticamente el código repetitivo (`__init__`, `__repr__`). Los Type Hints documentan tu código y activan el superpoder del autocompletado en tu editor.



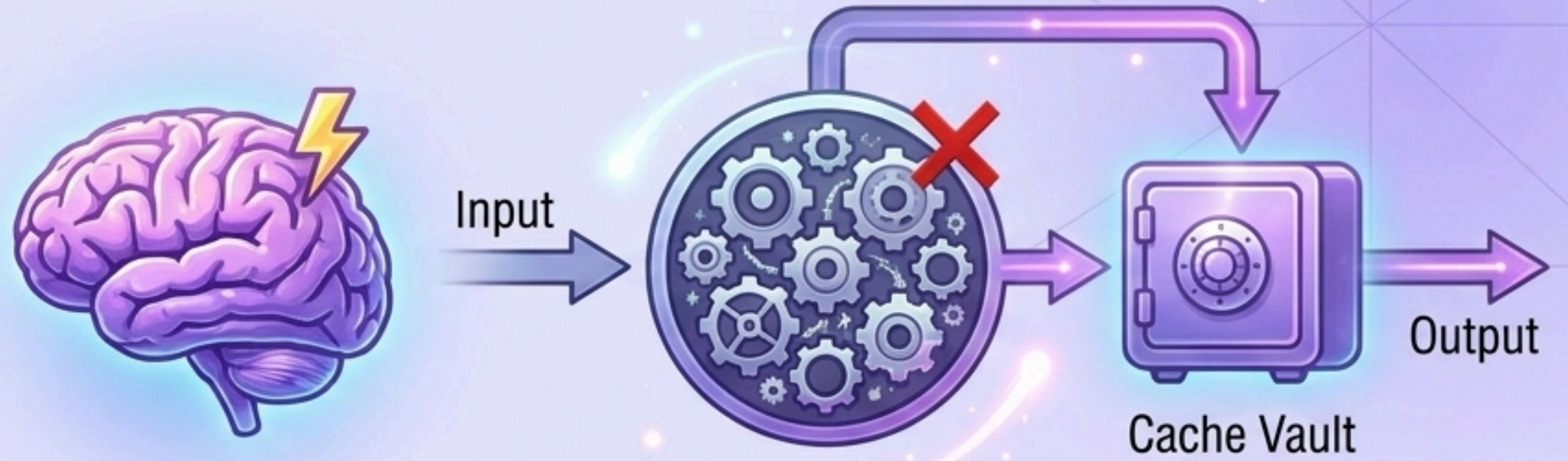
Tipar tu código no es una obligación, es amor propio. Tu 'yo' del futuro te lo agradecerá infinitamente.

```
from dataclasses import dataclass

@dataclass
class User:
    nombre: str
    edad: int = 18
```

Nivel 🚀 Avanzado

Memorización Instantánea



functools.cache / lru_cache

Si una función pura es costosa y repite los mismos cálculos, añade **@cache**. Python guardará los resultados anteriores y los servirá instantáneamente.



```
from functools import cache
```

```
@cache  
def calculo_pesado(n):  
    # Lógica costosa aquí  
    return resultado
```

¿Por qué pensar dos veces lo mismo? Si ya lo calculaste, guárdalo en caché y vuela.

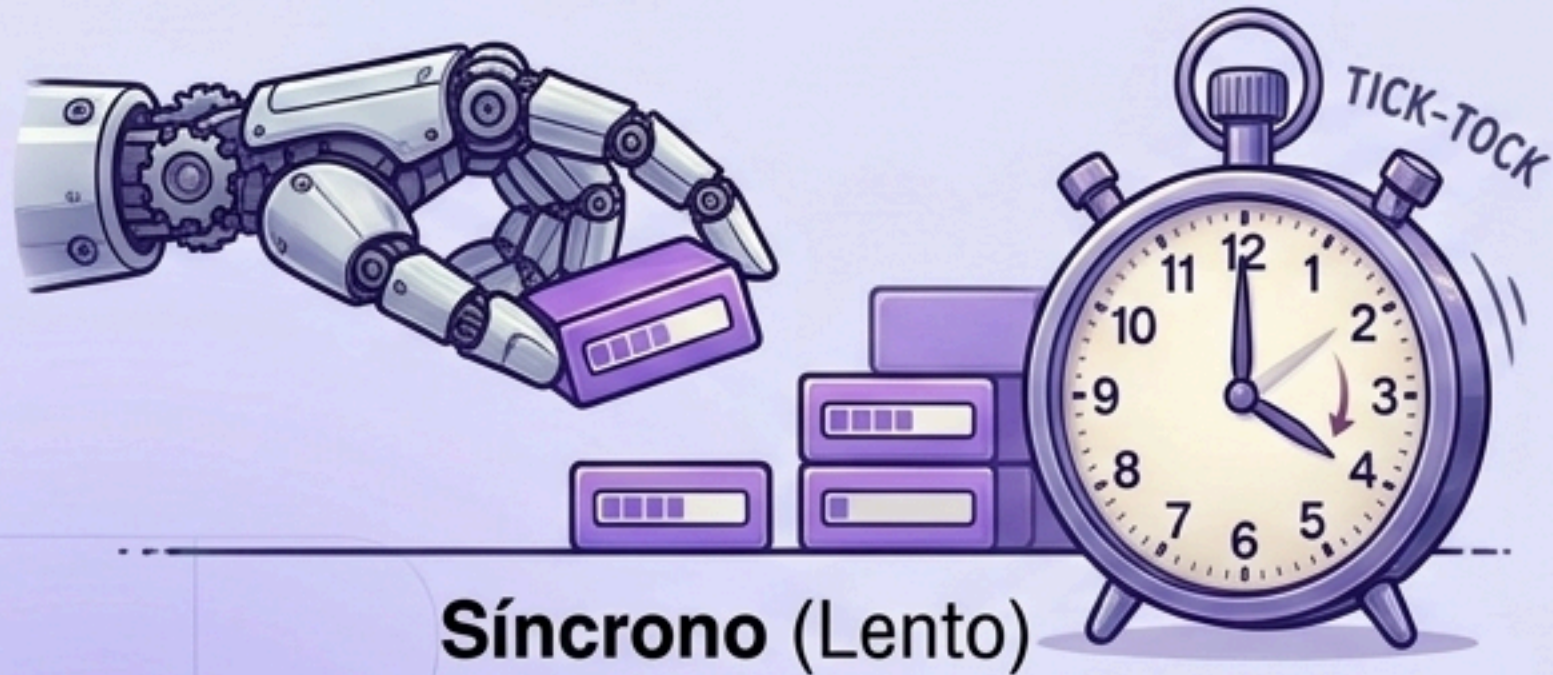


Nivel 🚀 Avanzado

Concurrencia de Alta Velocidad

async / await & asyncio

Optimiza los tiempos de espera masivos en operaciones de I/O (bases de datos, peticiones web). Mientras esperas una respuesta, el programa avanza con otra tarea.



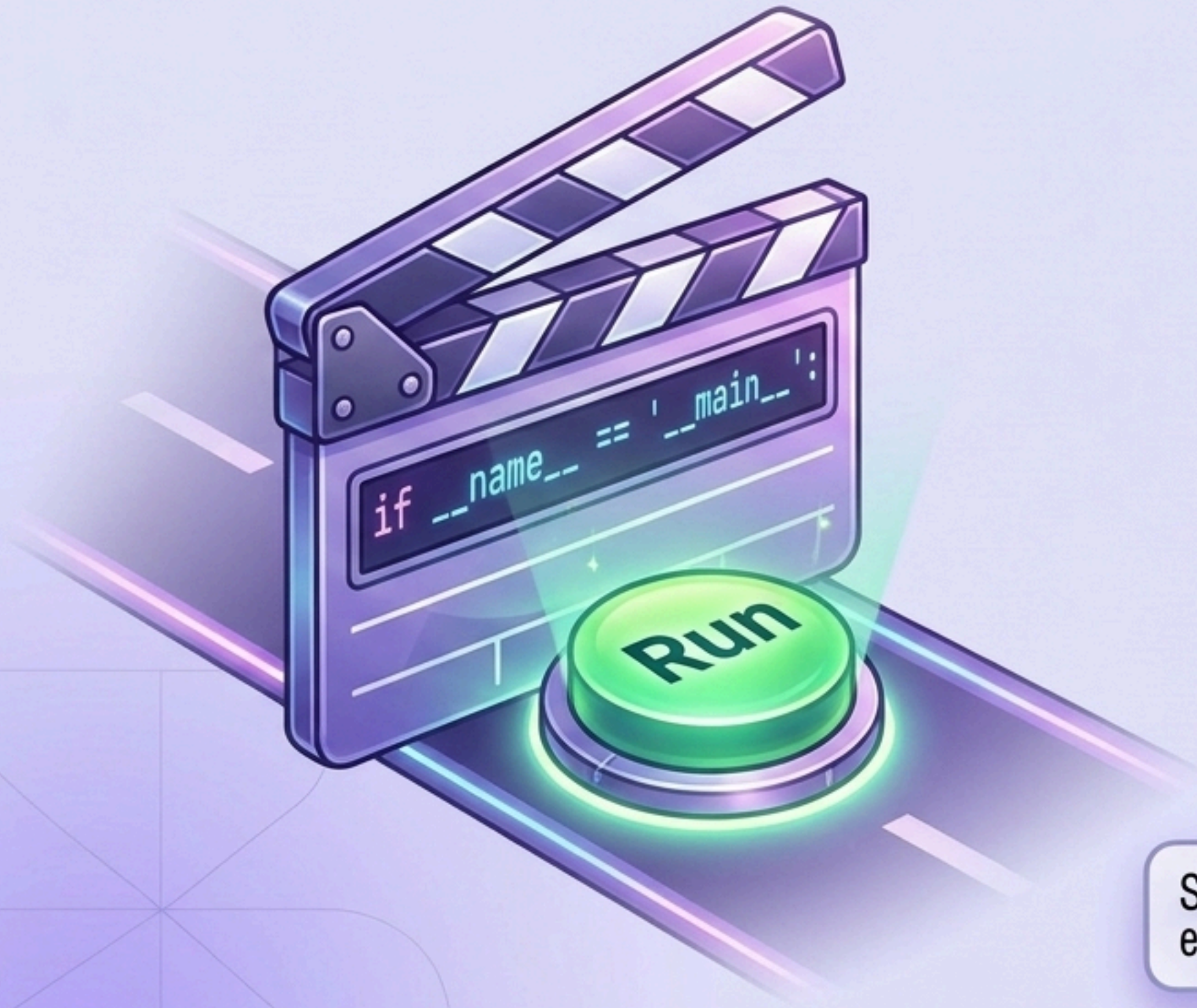
Mientras el servidor web piensa la respuesta, yo sigo cocinando otros datos. ¡No hay tiempo que perder!



```
async def fetch_data():  
    datos = await api_call()  
    # Permite correr otras tareas I/O concurrentemente
```

Nivel 🚀 Avanzado

Control de Módulos (El Director de Orquesta)



```
if __name__ == '__main__':
```

Protege tu código. Evita que los scripts se ejecuten automáticamente por accidente cuando el archivo es importado como un módulo desde otro script. Solo se ejecuta si lo llamas directamente.

```
def main():  
    # Lógica principal  
  
if __name__ == '__main__':  
    main()
```

Si te invitan a otra casa (import), compórtate y no empieces a ejecutar todas tus rutinas sin permiso.



El Verdadero Nivel Experto ✨

1. Escribir código complejo no te hace experto. Hacer lo complejo fácil de leer, sí.

2. Conoce todos los trucos, pero úsalos solo cuando aporten claridad o rendimiento real.

El mejor código es aquel que no necesita ser explicado.

3. Escribe código para humanos; las máquinas ya lo entienden.
(Legibilidad > Magia)

🤍 Like 💬 Comment

🔖 Guarda, comparte y programa feliz.

