diferencia entre iterable e iterator en python

Buena pregunta 👏 — esta diferencia entre **iterable** e **iterator** en Python es fundamental para entender cómo funcionan los bucles `for`, las comprensiones de listas, y muchas funciones del lenguaje.

Veamos la explicación clara y con ejemplos:

🧩 1. Iterable

Un **iterable** es **cualquier objeto que puede ser recorrido (iterado)**, es decir, del que puedes obtener un iterador.
👉 En términos técnicos: un objeto es iterable si implementa el método especial `__iter__()`.

Ejemplos de iterables:

- Listas → `["a", "b", "c"]`
- Tuplas → `(1, 2, 3)`
- Cadenas → `"hola"`
- Diccionarios → `{"x": 1, "y": 2}`
- Conjuntos → `{1, 2, 3}`
- Objetos personalizados que implementan `__iter__()`

python Copiar código

```python
lista = [1, 2, 3]
print(hasattr(lista, '__iter__'))  # True
```

🔁 2. Iterator

Un **iterator (iterador)** es **el objeto que realiza la iteración**, es decir, el que "recuerda" en qué posición va y sabe devolver el siguiente elemento al pedirlo.
👉 En términos técnicos: un objeto es un iterador si implementa **dos métodos**:

- `__iter__()` → devuelve el propio iterador.
- `__next__()` → devuelve el siguiente elemento o lanza `StopIteration` cuando no hay más.

Ejemplo:

python Copiar código

```python
lista = [1, 2, 3]
it = iter(lista)  # obtenemos un iterador desde un iterable

print(next(it))  # 1
print(next(it))  # 2
print(next(it))  # 3
# print(next(it))  # StopIteration
```

python Copiar código

```python
print(hasattr(it, '__next__'))  # True
```

🧠 Resumiendo

Concepto	Tiene `__iter__()`	Tiene `__next__()`	Ejemplo
Iterable	✅	❌	`list`, `str`, `tuple`
Iterator	✅	✅	`iter(list)`, `itertools.count()`

💡 Analogía

- **Iterable**: una caja de DVDs.
- **Iterator**: la mano que saca un DVD cada vez que la usas.
  - Puedes crear varias manos (iteradores) a partir de la misma caja (iterable).
  - Cada mano recuerda por dónde iba.

¿Quieres que te muestre cómo crear **tu propio iterador** con una clase personalizada? (así verías cómo funcionan `__iter__` y `__next__` por dentro).