`sorted()` y `.sort()` parecen similares —ambas **ordenan elementos**—, pero en Python tienen **diferencias importantes**.

1.Sorted()

- Es una **función integrada** (no un método).
- **Devuelve una nueva lista** ordenada.
- **No modifica** la lista original.
- Funciona con **cualquier iterable** (listas, tuplas, diccionarios, conjuntos, etc.).
- Se puede usar **fuera de listas**.

Ejemplos:

```
nombres = ["Luis", "Ana", "Carlos"]

ordenados = sorted(nombres)

print(ordenados)  # ['Ana', 'Carlos', 'Luis']

print(nombres)    # ['Luis', 'Ana', 'Carlos']  (no cambia)
```

2. .sort()

- Es un **método específico de las listas**.
- **Ordena en el sitio** (*in place*), o sea **modifica la lista original**.
- **No devuelve nada** (`None`).
- No puede aplicarse a otros tipos (no sirve para tuplas, diccionarios, etc.).

Ejemplo:

```
nombres = ["Luis", "Ana", "Carlos"]

nombres.sort()

print(nombres)  # ['Ana', 'Carlos', 'Luis']
```

3. Parámetros comunes

Ambas aceptan los mismos argumentos opcionales:

Parámetro	Descripción
`key`	función para transformar los valores antes de compararlos (por ejemplo `key=str.lower`)
`reverse`	si es `True`, ordena de mayor a menor

Ejemplo:

```
nombres = ["Luis", "ana", "Carlos"]

print(sorted(nombres, key=str.lower))  # ['ana', 'Carlos', 'Luis']

nombres.sort(key=str.lower, reverse=True)

print(nombres)                  # ['Luis', 'Carlos', 'ana']
```

4. Consejo

- Usa **`sorted()`** cuando:
  → No quieras modificar el original.
  → Estés trabajando con algo que no sea una lista (p. ej., un diccionario o un conjunto).
- Usa **`.sort()`** cuando:
  → Quieras ahorrar memoria y **te baste con reordenar la lista existente**.

5. Función lambda

Una **función `lambda`** es simplemente una **función anónima** (sin nombre) que se usa para operaciones pequeñas y rápidas.

lambda argumentos: expresión

Es como escribir una función normal, pero en una sola línea y sin def.

Ejemplo

```
# Con def:
def cuadrado(x):
    return x ** 2

# Con lambda:
lambda x: x ** 2
```

Ambas hacen lo mismo.
La diferencia es que `lambda` **no tiene nombre** (aunque puedes asignárselo si quieres):

```
f = lambda x: x ** 2
print(f(5))  # 25
```

Puesta en práctica de la función lambda:

```
def listar_alfabetico(listin):
    if not listin:
        print("(El listín está vacío)")
        return
    for nombre, telefono in sorted(listin.items(), key=lambda x: x[0].lower()):
        print(f"{nombre} - {telefono}")
```

```
sorted(listin.items(), key=lambda x: x[0].lower())
listin.items() devuelve una lista de tuplas:
EJ: [("Luis", "6123"), ("ana", "6543"), ("Carlos", "6987")]
```

En cada iteración, x es una de esas tuplas, por ejemplo:

```
x = ("Luis", "6123")
La expresión dentro de la lambda es:
```

```
x[0].lower()
Esto significa:
```

x[0] → toma el primer elemento de la tupla (el nombre).

.lower() → lo pasa a minúsculas.

Así que lambda x: x[0].lower() devuelve el nombre en minúsculas de cada par (nombre, teléfono).