📆 DAY 02 — Iterators & Generators

Goal : Understand *lazy evaluation* and write memory-efficient Python code

Why it **matters for ML**: datasets can be **millions of rows** — you cannot load everything into memory.

1️⃣ What is Iteration

when you write:

```
for x in data:
    ...
```

Python internally does:

```
iterator = iter(data)
value = next(iterator)
```

iteration is pulling values one by one, not all at once.

2️⃣ Iterator Basics

example:

```
nums = [1,2,3]
it = iter(nums)

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

If you call ''**next()**'' again:

```
StopIteration
```

that's how Python knows the loop is finished.

3️⃣ Custom Iterator (Interview-Level Knowledge)

To create own iterator, implement:

- __iter__()
- __next__()

Example: Custom counter Iterator

```
class Counter:
    def __init__(self, max_value):
        self.current = 0
        self.max_value = max_value

    def __iter__(self):
        return self

    def __next__(self):
        if self.current >= self.max_value:
            raise StopIteration
        self.current += 1
        return self.current
```

Usage:

```
for num in Counter(s):
    print(num)

# output
1 2 3 4 5
```

🧠 ML Insight

Custom iterators are used for:

- Batch loading
- Streaming data
- Reading huge files line-by-line

4️⃣ Why Iterators Matter (Memory)

❌ Bad (loads everything):

```
data = [x * x for x in range(10_000_000)]
```

✅ Good (lazy):

```
data = (x * x for x in range(10_000_000))
```

second one use **almost zero extra memory**.

5️⃣ Generators

Generators are **simpler iterators**.

Generator Function

```
def count_up_to(n):
    for i in range(1, n+1):
        yield 1
```

Usage:

```
gen = count_up_to(5)

print(next(gen)) # 1
print(next(gen)) # 2
```

or:

```
for x in count_up_to(5):
    print(x)
```

Key Rule:

- `return` → stops function
- `yield` → pauses function and remembers state

6️⃣ Generator Expressions (Pythonic & Powerful)

Like list comprehensions, but lazy.

```
squares = (x * x for x in range(10))
```

Instead of :

```
squares = [x * x for x in range(10)]
```

7️⃣ ML-Style Generator Example (Important)

Imagine a dataset too large to load.

```
def data_loader(file_path):
    with open(file_path) as f:
        for line in f:
            yield line.strip()
```

Usage:

```
for record in data_loader("data.txt"):
    process(record)
```

This is **exactly how ML pipelines work internally**.
