📅DAY 01 — Comprehensions & Lambdas

Goal : Write *clean, expressive, Pythonic code*

This is foundational of ML engineers

1️⃣ Why This Matters (ML Engineer Perspective)

In ML:

- You manipulate **lists of features**
- You clean **datasets**
- You transform **arrays and labels**

Bad Python:

```
new = []
for x in data:
    if x > 0:
        new.append(x * 2)
```

Good Python (what interviewers & ML code expect):

```
new = [x * 2 for x in data if x > 0]
```

Shorter, Faster, Cleaner and Easier to debug

2️⃣ List Comprehensions (Core Skill)

Basic Syntax:

```
[expression for item in iterable if condition]
```

Example:

```
num = [1,2,3,4,5]
```

```
squares = [n**2 for n in num]
```

with condition:

```
even_squares = [n**2 for n in num if n % 2 == 0]
```

⚠️ **Common Beginner Mistake**

❌ **Don’t write complex logic inside comprehensions.**

Bad:

```
[x**2 if x % 2 == 0 else x for x in nums]
```

Good:

```
def transform(x):
    return x**2 if x % 2 == 0 else x

result = [transform(x) for x in num]
```

3️⃣ Dictionary Comprehensions (Very Important)

Used heavily in:

- Feature encoding
- Frequency counting
- Mapping labels

Syntax:

```
{key: value for items in iterabel}
```

Example:

```
names = ["alice", "bob", "charlie"]
lengths = {names: len(names) for name in names}
```

with condition

```
filtered = {n: len(n) for n in names if len(n) > 3}
```

4️⃣ Set Comprehensions

Used to remove duplicates quickly.

```
nums = [1,2,3,4,4]
unique = {n for n in nums}
```

5️⃣ Lambda Functions (Anonymous Functions)

Normal Function:

```
def add(x, y):
    return x + y
```

Lambda Version

```
add = lambda x, y: x + y
```

use lambdas **Only for simple logic**.

6️⃣ map(), filter(), reduce()

map()

Apply function to each item.

```
nums = [1,2,3,4]
squared = list(map(lambda x : x ** 2, nums))
```

(Equivalent to list comprehension)

---

filter()

Keep only items that pass a condition.

```
even = list(filter(lambda x : x % 2 == 0, nums))
```

---

reduce() (from functools)

Reduce list to a single value.

```
from functools import reduce

nums = [1,2,3,4]
product = reduce(lambda x, y: x * y, nums)
```

7️⃣ Pythonic Rule (Important)

Prefer comprehensions over map/filter

Use reduce rarely (mostly in functional pipelines)
