🚀 What is a Lambda Function?

At the deepest level:

A lambda function is an anonymous (nameless), one-line function in Python used for short, simple operations.

Key words:

- **Anonymous** → it has no name
- **One-line** → only a single expression
- **Lightweight** → used where normal `def` feels too heavy

🔷 LEVEL 1 — Basic Syntax

Normal function:

```
def square(x):
    return x * x
```

Same thing using lambda:

```
square = lambda x: x * x
```

Calling it:

```
print(square(5))   # 25
```

General syntax of lambda

```
lambda arguments : expression
```

Example with two arguments:

```
add = lambda a, b: a + b
print(add(10, 20))   # 30
```

🔷 LEVEL 2 — Why is it called “anonymous”?

Because you **don’t need a function name**.

You can even use it directly:

```
print((lambda x: x + 5)(10))
```

Output:

```
15
```

Here:

- No function name was stored
- It was created and used instantly

🔷 LEVEL 3 — How is lambda different from def ?

Feature	lambda	def
Name	Optional	Required
Lines	Only one	Multiple
Return	Automatic	Must use `return`
Complexity	Simple logic	Complex logic
Use case	Short functions	Big logic

Example comparison:

Using def

```
def multiply(a, b):
    return a * b
```

Using lambda

```
multiply = lambda a, b: a * b
```

Both do the same thing.

🔷 LEVEL 4 — What does Python really do internally?

When you write:

```
square = lambda x: x * x
```

Python creates a **function object**, just like `def`, but without a name.

You can check its type:

```
print(type(square))
```

Output:

```
<class 'function'>
```

So **lambda is still a real function**, just written differently.

🔷 LEVEL 5 — Where lambda REALLY shines (Most important part)

Lambdas are mostly used with:

1️⃣ `map()`

Applies a function to every item in a list.

```
nums = [1,2,3,4]

squares = list(map(lambda x: x**2, nums))
print(squares)
```

Output:

```
[1, 4, 9, 16]
```

Equivalent to:

```
def sq(x):
    return x**2

squares = list(map(sq, nums))
```

2️⃣ `filter()`

Filters elements based on condition.

```
nums = [1,2,3,4,5,6]

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

Output:

```
[2, 4, 6]
```

3️⃣ `sorted()` with custom key

Sort by length of string:

```
words = ["apple", "banana", "kiwi"]

words.sort(key=lambda w: len(w))
print(words)
```

Output:

```
['kiwi', 'apple', ‘banana']
```

Here lambda tells Python **how to compare elements**.

🔷 LEVEL 6 — Lambda with conditions

You can use `if-else` inside lambda:

```
check = lambda x: "Even" if x % 2 == 0 else "Odd"
print(check(5))   # Odd
```

But you **cannot use loops inside lambda**.

❌ Not allowed:

```
lambda x: for i in range(x)   # invalid
```

🔷 LEVEL 7 — Why use lambda? (Core reasons)

You use lambda when:

- Function is very small
- You need it **only once**
- You don't want to clutter your code with `def`

Example:

Instead of this:

```
def is_positive(x):
    return x > 0

nums = [-2, -1, 0, 1, 2]
positives = list(filter(is_positive, nums))
```

You can simply write:

```
positives = list(filter(lambda x: x > 0, nums))
```

Cleaner and shorter.

🔷 LEVEL 8 — Limitations of lambda

You should **NOT use lambda when**:

- Logic is complex
- Multiple lines are needed
- Readability is important

Bad lambda example:

```
calc = lambda x: (x**2 + x - 3) / (x - 1) if x != 1 else 0
```

Better to write:

```
def calc(x):
    if x == 1:
        return 0
    return (x**2 + x - 3) / (x - 1)
```

🔷 LEVEL 9 — Lambda vs Comprehension

Sometimes list comprehension is better:

Lambda + map

```
squares = list(map(lambda x: x**2, range(5)))
```

Better Pythonic way

```
squares = [x**2 for x in range(5)]
```

Both work, second is more readable.

🔥 Final Cheat Sheet

Aspect	Lambda
Type	Function
Name	Optional
Lines	Only one
Return	Automatic
Best with	map, filter, sorted
Not for	Complex logic

One-line core meaning

“A lambda function is a small, nameless, one-line function used for simple operations, mainly with map, filter, and sorting.”