📄 Lambda Functions in Python (with Annotations & Decorators)

1. Introduction

- Lambda functions are anonymous, one-line functions created with the `lambda` keyword.
- They are often used for short, throwaway functions (e.g., sorting, mapping, filtering).
- Syntax:

```
lambda arguments: expression
```

---

2. Basic Lambda Function

```python
add = lambda x, y: x + y
print(add(3, 5))  # 8
```

---

3. Lambda vs `def`

```python
def add_def(x: int, y: int) -> int:
    return x + y

add_lambda: Callable[[int, int], int] = lambda x, y: x + y
```

---

4. Using Lambda in Sorting

```python
data: list[tuple[str, int]] = [("Alice", 25), ("Bob", 30), ("Eve", 20)]
sorted_data = sorted(data, key=lambda item: item[1])
print(sorted_data)  # [('Eve', 20), ('Alice', 25), ('Bob', 30)]
```

---

5. Lambda with `map()`

```python
nums: list[int] = [1, 2, 3, 4]
squares = list(map(lambda x: x ** 2, nums))
print(squares)  # [1, 4, 9, 16]
```

---

6. Lambda with `filter()`

```python
nums: list[int] = [1, 2, 3, 4, 5]
evens = list(filter(lambda x: x % 2 == 0, nums))
print(evens)  # [2, 4]
```

---

7. Lambda with `reduce()`

```python
from functools import reduce

nums: list[int] = [1, 2, 3, 4]
product = reduce(lambda a, b: a * b, nums)
print(product)  # 24
```

---

8. Nested Lambda

```python
multiply = lambda x: (lambda y: x * y)
double = multiply(2)
print(double(5))  # 10
```

---

9. Lambda with Conditional Expressions

```python
max_num: Callable[[int, int], int] = lambda a, b: a if a > b else b
print(max_num(10, 20))  # 20
```

---

10. Using Annotations with Lambda

```python
from typing import Callable

adder: Callable[[int, int], int] = lambda x, y: x + y
```

---

11. Lambda in Higher-Order Functions

```python
def apply_func(f: Callable[[int], int], value: int) -> int:
    return f(value)

print(apply_func(lambda x: x ** 2, 5))  # 25
```

---

12. Using Lambda in Decorators

```python
from functools import wraps

def debug(func: Callable[..., Any]) -> Callable[..., Any]:
    @wraps(func)
    def wrapper(*args, **kwargs):
        print(f"Calling {func.__name__} with {args}, {kwargs}")
        return func(*args, **kwargs)
    return wrapper

@debug
def greet(name: str) -> str:
    return (lambda x: f"Hello, {x}")(name)

print(greet("Alice"))
```

---

13. Lambda in Key Functions

```python
words: list[str] = ["apple", "banana", "cherry", "kiwi"]
sorted_words = sorted(words, key=lambda x: len(x))
print(sorted_words)  # ['kiwi', 'apple', 'banana', 'cherry']
```

---

14. Lambda with Default Arguments

```python
increment = lambda x, step=1: x + step
print(increment(5))      # 6
print(increment(5, 3))   # 8
```

---

15. Best Practices

- Use lambda for short, simple functions.
- Use `def` for complex functions (readability).
- Annotate lambdas with `Callable` when used in higher-order functions.
- Use lambdas with map, filter, reduce, sorted for concise logic.

📝 25 Exercises on Lambda Functions in Python

Beginner

1. Write a lambda function to add two numbers.
2. Create a lambda function to square a number.
3. Write a lambda function to check if a number is even.
4. Use a lambda to get the maximum of two numbers.
5. Use a lambda to return the length of a string.

Intermediate

6. Use lambda with `map()` to cube a list of numbers.
7. Use lambda with `filter()` to extract odd numbers.
8. Use lambda with `reduce()` to sum a list.
9. Sort a list of tuples using a lambda as the key.
10. Create a lambda with a default argument.

Higher-Order Functions

11. Pass a lambda as a function argument.
12. Write a function that applies a lambda twice.
13. Write a nested lambda that multiplies two numbers.
14. Use lambda inside `apply_func(f, value)` function.
15. Annotate a lambda with `Callable[[int, int], int]`.

Advanced

16. Use lambda in a dictionary mapping operations.
17. Create a lambda that reverses a string.
18. Use lambda inside a decorator to format a message.
19. Use lambda in a comprehension (`[lambda x: x*2 for ...]`).
20. Use lambda with multiple conditions (e.g., check range).

Challenge

21. Write a lambda that calculates factorial using `reduce()`.
22. Build a `key=lambda` to sort names ignoring case.
23. Use lambda with `ThreadPoolExecutor` to run tasks.
24. Create a lambda that returns another lambda (closure).
25. Combine map + filter + lambda to process a dataset.

🔑 Key Annotations and Decorators for Lambda

Annotation / Decorator	Use Case
`Callable[[Args], ReturnType]`	Type hints for lambda signatures
`-> ReturnType`	Explicit return annotation for lambda equivalents
`@wraps(func)`	Preserve metadata when using lambda in decorators
`lambda x, y: ...`	Inline anonymous functions
`lambda x=default: ...`	Default arguments in lambda