**Lambda Functions in Python**

A **lambda function** in Python is an anonymous, inline function defined using the lambda keyword. It can have multiple arguments but only a single expression.

**Syntax:**

lambda arguments: expression

* The expression is evaluated and returned.
* It does **not** require an explicit return statement.

**1. Basic Example**

square = lambda x: x \* x

print(square(5)) # Output: 25

Equivalent to:

def square(x):

return x \* x

**2. Multiple Arguments**

add = lambda a, b: a + b

print(add(3, 5)) # Output: 8

maximum = lambda x, y: x if x > y else y

print(maximum(10, 20)) # Output: 20

**3. Using Lambda with Built-in Functions**

**a) map() Function**

map() applies a function to all items in an iterable.

nums = [1, 2, 3, 4]

squared\_nums = list(map(lambda x: x\*\*2, nums))

print(squared\_nums) # Output: [1, 4, 9, 16]

**b) filter() Function**

filter() selects elements based on a condition.

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

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

print(even\_nums) # Output: [2, 4, 6]

**c) sorted() with key Parameter**

students = [("Alice", 25), ("Bob", 20), ("Charlie", 23)]

sorted\_students = sorted(students, key=lambda student: student[1])

print(sorted\_students) # Output: [('Bob', 20), ('Charlie', 23), ('Alice', 25)]

**4. Lambda in List Comprehensions**

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

doubled = [(lambda x: x \* 2)(x) for x in nums]

print(doubled) # Output: [2, 4, 6, 8, 10]

**5. Lambda in reduce()**

reduce() (from functools module) applies a function cumulatively.

from functools import reduce

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

product = reduce(lambda x, y: x \* y, nums)

print(product) # Output: 120

**6. Nested Lambda Functions**

nested\_lambda = lambda x: (lambda y: x + y)

adder = nested\_lambda(10)

print(adder(5)) # Output: 15

**7. Assigning Lambda to a Dictionary**

operations = {

"add": lambda x, y: x + y,

"subtract": lambda x, y: x - y,

"multiply": lambda x, y: x \* y

}

print(operations["add"](10, 5)) # Output: 15

print(operations["multiply"](3, 4)) # Output: 12

**8. When to Use Lambda?**

✅ Use for small, short-lived functions.  
✅ Use in higher-order functions (map, filter, sorted, reduce).  
✅ Use as arguments for built-in functions.

🚫 Avoid when the function is complex (better to use def).  
🚫 Avoid when debugging, as lambdas lack docstrings and are harder to read.

**Conclusion**

Lambda functions make Python more concise and functional-style programming more powerful. However, they should be used wisely to maintain code readability.