Day 18: Lambda Functions and List Comprehensions in Python

Overview

Python provides **lambda functions** and **list comprehensions** to write more concise and efficient code. These features are heavily used in **functional programming**, **data processing**, and **one-liner solutions**.

By the end of this lesson, we will be able to:

✅ Use **lambda functions** to create anonymous, inline functions.

✅ Apply **list comprehensions** for quick list operations.

✅ Combine **lambda functions and list comprehensions** for **efficient data manipulation**.

---

1. Anonymous Functions Using Lambda

🔹 What is a Lambda Function?

A **lambda function** is a small, anonymous function that **doesn't require a name** and can have **any number of arguments but only one expression**.

🔹 Syntax of Lambda Function

lambda arguments: expression

✅ Key Characteristics

✓ Defined using the lambda keyword.

✓ Can take multiple arguments but **only one expression**.

✓ Returns the result **implicitly** (no need for return).

✓ Often used where small functions are required **temporarily**.

🔹 Why Use Lambda Functions?

✓ They provide **a more concise** way to write functions.

✓ They are useful when a function is needed for a **short period**.

✓ They can be used inside **higher-order functions** like map(), filter(), and sorted().

✓ They help in **reducing code complexity** and increasing **readability**.

Syntax of Lambda Functions:

```
lambda arguments: expression
```

- **Example 1: Lambda Function to Add Two Numbers**

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

✅ Equivalent to:

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

- **Example 2: Finding the Square of a Number**

```
square = lambda x: x ** 2
print(square(4))  # Output: 16
```

- **Example 3: Checking Even or Odd**

```
is_even = lambda x: "Even" if x % 2 == 0 else "Odd"
print(is_even(7))  # Output: "Odd"
```

- **Using Lambda with Higher-Order Functions**

🟢 **Example 4: Using lambda with map()**

map() applies a function to each item in an iterable (e.g., list, tuple).

```
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x ** 2, numbers))
print(squared)  # Output: [1, 4, 9, 16, 25]
```

Example 5: Using lambda with filter()

filter() is used to filter elements based on a condition.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4, 6, 8]
```

Example 6: Using lambda with sorted()

Sort a list of tuples based on the second element.

```
students = [("Alice", 85), ("Bob", 72), ("Charlie", 90)]
sorted_students = sorted(students, key=lambda x: x[1])
print(sorted_students)
# Output: [('Bob', 72), ('Alice', 85), ('Charlie', 90)]
```

---

2. List Comprehensions for Concise Operations

What is a List Comprehension?

List comprehensions provide an **efficient and elegant way** to create lists in a **single line of code**, rather than using loops.

Why Use List Comprehensions?

✓ More **readable and concise** than loops.
✓ More **efficient** in terms of execution time.
✓ Makes **list creation and transformation easier**.
✓ Helps in **data filtering and transformation**.

Syntax of List Comprehension

[expression for item in iterable if condition]

✅ **Key Characteristics**

✓ More **readable and efficient** than traditional loops.

✓ Combines loops and conditional logic in a **single line**.

✓ Creates a **new list** based on an existing iterable.

🔹 **Example 1: Creating a List of Squares**

Using a traditional loop:

```
squares = []
for x in range(1, 6):
    squares.append(x ** 2)
print(squares)  # Output: [1, 4, 9, 16, 25]
```

Using List Comprehension:

```
squares = [x ** 2 for x in range(1, 6)]
print(squares)  # Output: [1, 4, 9, 16, 25]
```

✅ **Advantage: Less code, more readability!**

🔹 **Example 2: Generating a List of Even Numbers**

```
evens = [x for x in range(1, 11) if x % 2 == 0]
print(evens)  # Output: [2, 4, 6, 8, 10]
```

🔹 **Example 3: Filtering Words from a List**

```
words = ["hello", "world", "python", "code"]
short_words = [word for word in words if len(word) <= 5]
print(short_words)  # Output: ['hello', 'world', 'code']
```

- **Example 4: Creating a List of Tuples**

Create a list of (number, square) tuples for numbers **1 to 5**.

```
squares = [(x, x ** 2) for x in range(1, 6)]
print(squares)
# Output: [(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
```

- **Example 5: Flattening a Nested List**

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
flattened = [num for row in matrix for num in row]
print(flattened)
# Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

---

3. Combining Lambda Functions & List Comprehensions

✅ **Using lambda inside a list comprehension**

```
numbers = [1, 2, 3, 4, 5]
doubled = [(lambda x: x * 2)(x) for x in numbers]
print(doubled)  # Output: [2, 4, 6, 8, 10]
```

✅ **Filtering even numbers using lambda inside list comprehension**

```
numbers = [1, 2, 3, 4, 5, 6]
evens = [x for x in numbers if (lambda x: x % 2 == 0)(x)]
print(evens)  # Output: [2, 4, 6]
```

Recap of Key Concepts

Feature	Lambda Functions	List Comprehensions
Purpose	Short, **anonymous functions**	Concise **list creation**
Syntax	lambda args: expression	[expression for item in iterable if condition]
Use Case	**Quick inline functions**	**Efficient list operations**
Example	lambda x: x ** 2	[x**2 for x in range(5)]
Applied With	map(), filter(), sorted()	Loops & Conditionals

Conclusion

- ✅ **Lambda Functions** are great for **one-line anonymous functions**.
- ✅ **List Comprehensions** make list operations **more concise and readable**.
- ✅ **Both features improve performance and efficiency** in Python programming.

These concepts are widely used in **data science, web development, and competitive coding**. Mastering them will make your **Python skills much stronger!**