Lambda Functions in Python

MUKESH KUMAR

Agenda

- ✓Introduction to Lambda Functions
- ✓Syntax of Lambda Functions
- ✓Lambda vs Def Functions
- ✓Using Lambda with map(), filter(), and reduce()
- ✓When to Use Lambda Functions
- ✓Limitations of Lambda Functions
- ✓Summary and Discussion

Introduction to Lambda Functions

- A lambda function is a small anonymous function in Python.
- It can take multiple arguments but only contains a single expression.
- Useful for short, simple operations where defining a full function is unnecessary.

Key characteristics of lambda functions:

- **Anonymous**: Lambda functions don't have a name.
- **Single Expression:** They can only contain one expression, which is evaluated and returned.
- **Arguments**: Lambda functions can accept any number of arguments.
- **Syntax**: The syntax is lambda arguments: expression.
- **Usage**: They are often used as anonymous functions inside other functions or when a simple function is needed for a short period.

USAGE

- Syntax:
  - **lambda arguments: expression**

- Example

```
square = lambda x: x * x
print(square(5))  # Output: 25
```

Lambda Examples

- Add 10 to a number:

```
x = lambda a: a + 10
print(x(5))  # Output: 15
```

- Multiply two arguments:

```
x = lambda a, b: a * b
print(x(5, 6))  # Output: 30
```

Lambda Examples

- Summarize three arguments:

```
x = lambda a, b, c: a + b + c
print(x(5, 6, 2))  # Output: 13
```

- Double a number:

```
dbl = lambda n: 2 * n
print(dbl(7))  # Output: 14
```

Lamda Vs User-Defined

Feature	Lambda Function	Def Function
Definition	Single-line, anonymous function	Multi-line, named function
Return Type	Implicitly returns value	Requires return statement
Readability	Good for short operations	Preferred for complex logic
Use Case	Quick, one-time use	Reusable and structured functions

When to Use Lambda Functions

- When a function is simple and used only once.
- When passing functions as arguments to higher-order functions.
- For quick data transformations in map(), filter(), etc.

Limitations of Lambda Functions

- Can only contain a single expression.
- Reduced readability for complex logic.
- Difficult to debug compared to named functions.

Lambda Practice Problems

- Refer Jupyter Notebook

Lambda with map() Function

- **map()** applies a function to each item in an iterable.

- Example

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

Lambda with filter() Function

- **filter()** selects items from an iterable based on a condition.

- Example:

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

Lambda with reduce() Function

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

- Example

```
from functools import reduce
nums = [1, 2, 3, 4]
product = reduce(lambda x, y: x * y, nums)
print(product)  # Output: 24
```

When to Use Lambda Functions

- When a function is simple and used only once.
- When passing functions as arguments to higher-order functions.
- For quick data transformations in map(), filter(), etc.

Limitations of Lambda Functions

- Can only contain a single expression.
- Reduced readability for complex logic.
- Difficult to debug compared to named functions.

Summary

- Lambda functions provide a quick way to define short functions.
- They are commonly used with map(), filter(), and reduce().
- Best used for simple, one-time-use operations.
- Avoid for complex logic requiring better readability.