Python Lambda Functions

Last Updated : 23 Jan, 2026

Lambda functions are small anonymous functions, meaning they do not have a defined name. In Python, lambda functions are created using the lambda keyword for short, simple operations.

- No name unless you assign it to a variable
- Only one expression
- Implicit return of that expression

Lambdas are small, short-lived functions created and used immediately, mainly to pass simple logic as an argument to another function.

Example: In the example, we defined a lambda function to convert a string to its upper case using upper().

```
a = 'GeeksforGeeks'
upper = lambda x: x.upper()
print(upper(a))
```

Output

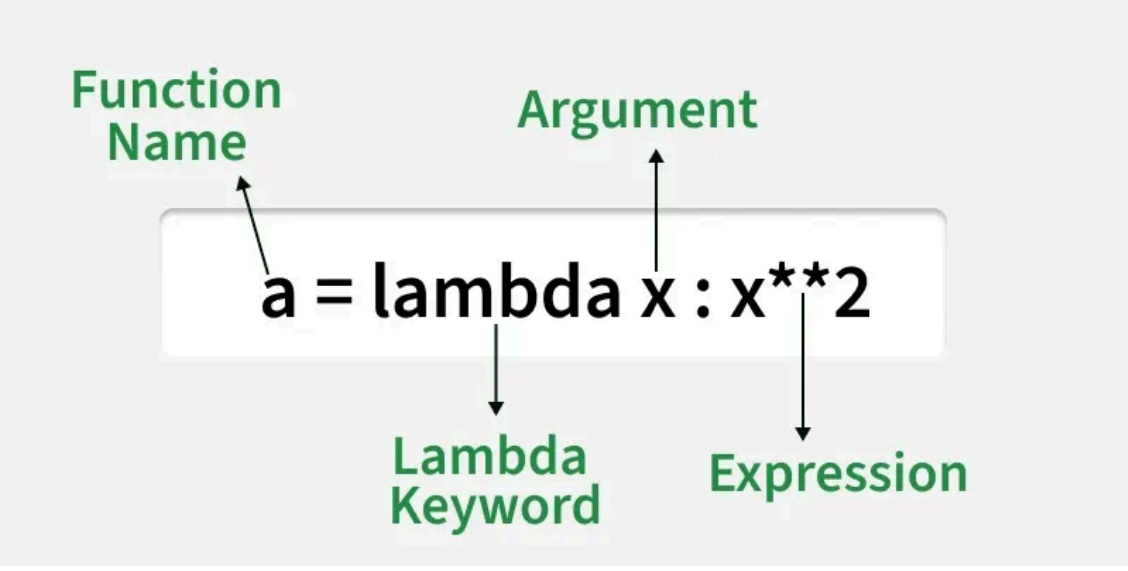
```
GEEKSFORGEEKS
```

Explanation:

- a stores the string 'GeeksforGeeks'.
- upper is a lambda function that takes an argument x and returns x.upper().
- upper(a) applies the lambda to a, converting it to uppercase.

Syntax of Lambda Expression

Python lambda expression has the following syntax:



Python Lambda Expression

- **Function name (a):** Stores the lambda function so it can be reused later.
- **Lambda keyword (lambda):** Defines an anonymous (inline) function in Python.
- **Argument (x):** The input value passed to the lambda function.
- **Expression (x**2):** The operation performed on the argument and returned as the result.

Use Cases of Lambda Functions

Let's see some of the practical uses of the Python lambda function.

1. Using with Condition Checking

A lambda function can use conditional expressions (if-else) to return different results based on a condition.

```
check = lambda x: "Positive" if x > 0 else "Negative" if x < 0 else "Zero"
print(check(5))
print(check(-3))
print(check(0))
```

Output

```
Positive
Negative
Zero
```

Explanation:

- The lambda function takes x as input.
- It uses nested if-else statements to return "Positive," "Negative," or "Zero."

Below example checks divisibility by 2 and returns "Even" or "Odd" accordingly.

```
check = lambda x: "Even" if x % 2 == 0 else "Odd"
print(check(4))
print(check(7))
```

Output

```
Even
Odd
```

Explanation:

- The lambda checks if a number is divisible by 2 (x % 2 == 0).
- Returns "Even" for true and "Odd" otherwise.

2. Using with List Comprehension

Lambda functions can be combined with list comprehensions to apply the same operation to multiple values in a compact way.

```
func = [lambda arg=x: arg * 10 for x in range(1, 5)]
for i in func:
    print(i())
```

Output

```
10
20
30
40
```

Explanation:

- lambda function multiplies each element by 10.
- list comprehension iterates through li and applies the lambda to each element.

3. Using for Returning Multiple Results

Although a lambda can contain only one expression, it can still return multiple results by combining them into a tuple.

```
calc = lambda x, y: (x + y, x * y)
res = calc(3, 4)
print(res)
```

Output

```
(7, 12)
```

Explanation: lambda function performs both addition and multiplication and returns a tuple with both results.

4. Using with filter()

filter() function uses a lambda expression to select elements from a list that satisfy a given condition, such as keeping only even numbers.

```
c = [1, 2, 3, 4, 5, 6]
even = filter(lambda x: x % 2 == 0, c)
print(list(even))
```

Output

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

Explanation:

- The lambda function checks if a number is even (x % 2 == 0).
- filter() applies this condition to each element in nums.

5. Using with map()

map() function applies a lambda expression to each element of a list and returns a new list with the transformed values.

```
a = [1, 2, 3, 4]
double = map(lambda x: x * 2, a)
```

```
print(list(double))
```

Output

```
[2, 4, 6, 8]
```

Explanation:

- The lambda function doubles each number.
- map() iterates through a and applies the transformation.

6. Using with reduce()

reduce() function repeatedly applies a lambda expression to elements of a list to combine them into a single result.

```
from functools import reduce
a = [1, 2, 3, 4]
mul = reduce(lambda x, y: x * y, a)
print(mul)
```

Output

```
24
```

Explanation:

- The lambda multiplies two numbers at a time.
- reduce() applies this operation across the list.

lambda vs def Keyword

In Python, both lambda and def can be used to define functions, but they serve slightly different purposes. While def is used for creating standard reusable functions, lambda is mainly used for short, anonymous functions that are needed only temporarily.

```
# Using lambda
square = lambda x: x ** 2
print(square(3))

# Using def
def sqdef(x):
    return x ** 2
print(sqdef(3))
```

Output

```
9
9
```

Explanation:

- lambda function **square** takes a number and returns its square.
- regular function **sqdef** does the same but is defined using the **def** keyword.
- Both approaches give the same result but **lambda** is more concise.

Now, let's see a comparison between these two in tabular form:

Feature	lambda Function	Regular Function (def)
Definition	Single expression with lambda.	Multiple lines of code.
Name	Anonymous or named if assigned.	Must have a name.
Statements	Single expression only.	Can include multiple statements.
Documentation	Cannot have a docstring.	Can include docstrings.
Reusability	Best for short, temporary functions.	Better for reusable and complex logic.

Article Tags: Python Python-Functions python-lambda python