Chapter 7 Lambda Functions

A **Lambda** function is a small, anonymous function defined using `lambda` keyword. It is used for small tasks that simple enough to be written in a single line.

Syntax:

Lambda Function	Regular Function
`lambda parameters: expression`	`def name(parameters):` `  return expression`

Where:

- `parameters`: The input parameters for the lambda function.
- `expression`: The operation to be performed and returned.

1. Characteristics of Lambda Functions

- They are one-liners.
- They are anonymous; no name.
- They return the result of their expression implicitly.
- They can have multiple parameters but can **only have a single expression**.

Example 01: Use a lambda function to sum two numbers.

```
print("result =", (lambda x, y: x + y)(5, 3))
```

Output

```
result = 8
```

In the above example, we define a lambda function `lambda x, y: x + y`, and call it immediately on the same line by passing `5` and `3` to the parameters `x` and `y` respectively.

If you want to call a lambda function later or reuse it multiple times, you should assign it to a variable. This approach also improves readability.

Example 02: Assign a lambda function to a variable.

```
add = lambda x, y: x + y   # define a lambda function

print("result1 =", add(5, 3))   # call the lambda function
print("result2 =", add(2, 8))   # call the lambda function
```

Output

```
result1 = 8
result2 = 10
```

You can create a lambda function with no parameter.

Example 03: Define a lambda function that has no parameter, and call it immediately on the same line.

```
(lambda: print("Hello"))() # define and call a lambda function
```

Output

```
Hello
```

Example 04: Define a lambda function that has no parameter, and assign it to a variable.

```
lambda : print("Hello")   # define a lambda function

greating()   # call the lambda function
```

Output

```
Hello
```

You can create a lambda function with default parameters

Example 05: Use a lambda function with default parameters.

```
add = lambda x = 1, y = 2: x + y

print("result =", add())
```

Output

```
result = 3
```

Example 06: Pass a list to a lambda function

```
numbers = [1, 2, 3, 4, 5]

total = (lambda list1: sum(list1))(numbers)

print("Total is", total)
```

Output

```
Total is 15
```

2. Returning a Lambda Function from a Function

You can return a lambda function from a regular function.

Example 07:

```
# A regular function that returns a lambda function
def multiply(n):
  return lambda x: x * n

double = multiply(2)   # double = lambda x: x * 2
triple = multiply(3)   # triple = lambda x: x * 3

print("4 * 2 is", double(4))   # 4 * 2 = 8
print("5 * 3 is", triple(5))   # 5 * 3 = 15
```

Output

```
4 * 2 is 8
5 * 3 is 15
```

You also can return a lambda function from another lambda function.

Example 08:

```
# A lambda function that returns another lambda function
multiply = lambda n: lambda x: x * n

double = multiply(2)  # double = lambda x: x * 2
triple = multiply(3)  # triple = lambda x: x * 3

print("4 * 2 is", double(4))  # 4 * 2 = 8
print("5 * 3 is", triple(5))  # 5 * 3 = 15
```

Output

```
4 * 2 is 8
5 * 3 is 15
```

3. Using Lambda Functions with Built-in Functions

There are some common built-in functions that are often used with lambda functions in Python such as `map()`, `filter()`, `any()`, and `all()` function.

3.1 The `map()` Function

The `map()` function applies a given function to each item in an iterable (e.g., a list, tuple, etc.) and returns a map object.

Syntax:

```
map(function, iterable)
```

Example 09: Use `map()` function to apply a function to each item in a list.

```
numbers = [1, 2, 3, 4]

squares = list(map(lambda x: x ** 2, numbers))

print(squares)
```

Output

```
[1, 4, 9, 16]
```

Example 10: Get multiple integers from the user input

```
list1 = input("Enter numbers separated by spaces: ").split()

list1 = list(map(lambda x: int(x), list1))

print(list1)
```

Output:

```
Enter numbers separated by spaces: 1 3 5 7 9
[1, 3, 5, 7, 9]
```

You can pass multiple iterables to the `map()` function, and the function will be applied to the corresponding items from each iterable in parallel.

Example 11: Add corresponding items from two lists.

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]

result = list(map(lambda x, y: x + y, list1, list2))

print(result)
```

Output

```
[5, 7, 9]
```

3.2 The `filter()` Function

The `filter()` function filters items from an iterable based on a given boolean function. It returns a filter object containing only the items for which the function returns `True`.

Syntax:

```
filter(function, iterable)
```

Example 12: Filter even numbers from a list.

```
numbers = [1, 2, 3, 4, 5, 6]

even_numbers = list(filter(lambda x: x % 2 == 0, numbers))

print(even_numbers)
```

Output

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

3.3 The `any()` Function

The `any()` function returns `True` if any item of an iterable is true; otherwise returns `False`.

Example 13: Check if at least one item in a list has any positive number.

```
list1 = [-1, -2, 3, 4]
list2 = [-1, -2, -3, -4]

list1_has_positive = any(map(lambda x: x > 0, list1))
list2_has_positive = any(map(lambda x: x > 0, list2))

print(list1_has_positive)
print(list2_has_positive)
```

Output

```
True
False
```

3.4 The `all()` Function

The `all()` function returns `True` if all items of an iterable are true; otherwise returns `False`.

Example 14: Check if all items in a list are positive.

```
list1 = [1, 2, 3, 4]
list2 = [1, 2, -3, 4]

list1_all_positive = all(map(lambda x: x > 0, list1))
list2_all_positive = all(map(lambda x: x > 0, list2))

print(list1_all_positive)
print(list2_all_positive)
```

Output

```
True
False
```

Exercises

Write the following programs using lambda functions:

1. Calculate the area of a rectangle.
2. Calculate the double and square, $(2x)^2$, of all numbers in a list.
   - Input: `[1, 2, 3, 4]`
   - Output: `[4, 16, 36, 64]`
3. Convert all items in a list of strings to lowercase.
4. Concatenate corresponding items of two lists of strings.
   - Input: `list1 = ["Hello", "Good"]` and `list2 = ["World", "Morning"]`
   - Output: `["Hello World", "Good Morning"]`
5. Calculate the sum of each row in a 2D list.
6. Compare corresponding items of two lists and return "list1" if the item from the list1 is larger, otherwise return "list2".
   - Input: `list1 = [3, 8, 15]` and `list2 = [5, 7, 10]`
   - Output: `["list2", "list1", "list1"]`
7. Filter strings from a list that are shorter than 5 characters.
8. Filter numbers that are divisible by 3 and not divisible by 2 from a list.
9. Check if any number in a list is divisible by both 3 and 5.
10. Check if all string in a list are palindromes.