



Lambda Functions



CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents



- ▶ Defining a Lambda Function
- ▶ Uses of the Lambda Functions
- ▶ Lambda within Built-in (map()) Functions-1
- ▶ Lambda within Built-in (filter()) Functions-2
- ▶ Lambda within User-Defined Functions



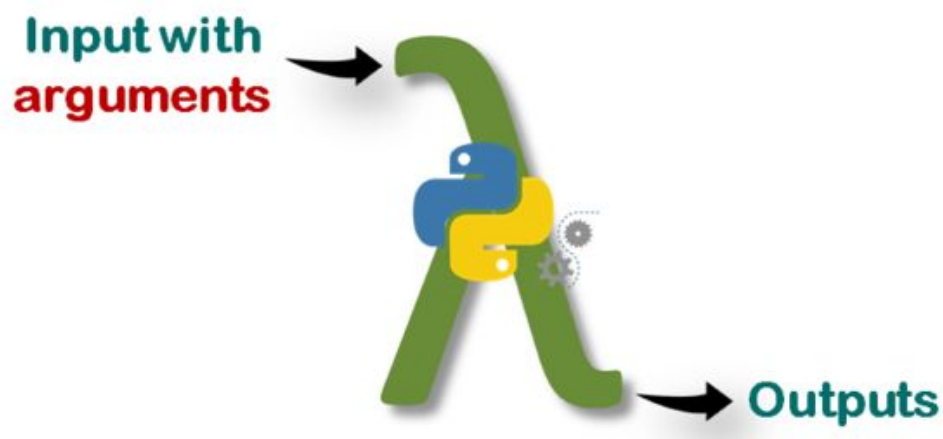
1

Defining a Lambda Function

Defining a `lambda` Function(review)



The formula syntax is : `lambda parameters : expression`



Why we need `lambda` functions?



- ▶ Compare the two types of functions :

```
1 def square(x)
2     return x**2
```

- ▶ And now we'll define `lambda` function to do the same.

```
1 lambda x: x**2
```

Why we need `lambda` functions?



- ▶ Compare the two types of functions :

```
1 def square(x)
2     return x**2
```

⚠ Avoid:

- Note that you do not need to use `return` statement in `lambda` functions.

Defining a Lambda Function(review)

- Multiple parameters/arguments :

```
1 lambda x, y: (x+y)/2 # takes two numbers, returns the result
```

Defining a Lambda Function(review)

- The formula syntax of conditional lambda statement is :

```
lambda parameters : first_result if conditional statement else second_result
```

```
lambda x: 'odd' if x%2 != 0 else 'even'
```

evaluated first *else*

⚠ Avoid:

- Note that you can't use the usual conditional statement with lambda definition.

▶ Defining a `lambda` Function(review)

- ▶ Conditional statements in a lambda function :

```
1 lambda x: 'odd' if x % 2 != 0 else 'even'
```



2 ▶ Uses of the `lambda` Functions

Uses of the `lambda` Functions(review)

- ▶ Usage alternatives of Lambda functions:
 - ▷ ... its own syntax using parentheses,
 - ▷ ... assigning it to a variable,
 - ▷ ... inside several built-in functions,
 - ▷ ... inside user-defined functions (def),

Uses of the `lambda` Functions(review)

- ▶ **By enclosing the function in parentheses**

First use

The formula syntax is :

`(lambda parameters: expression)(arguments)`

```
1 print((lambda x: x**2)(2))
```

What is the output? Try to figure out in your mind...

Uses of the `lambda` Functions(review) »

- By enclosing the function in parentheses :

The formula syntax is :

`(lambda parameters : expression)(arguments)`

```
1 print((lambda x: x**2)(2))
```

```
1 4
```

Uses of the `lambda` Functions(review) »

- Multiple arguments :

- Option - I

```
1 print((lambda x, y: (x+y)/2)(3, 5)) # takes two int,  
    returns mean of them
```

- Option - II

```
1 average = (lambda x, y: (x+y)/2)(3, 5)  
2 print(average)
```

Uses of the Lambda Functions(review) »

- Or you can use multiple arguments using the same syntax :

```
1 print((lambda x, y: (x+y)/2)(3, 5)) # takes two int,  
    returns mean of them
```

```
1 4.0
```

- You can also assign the lambda statement in parentheses to a variable :

```
1 average = (lambda x, y: (x+y)/2)(3, 5)  
2 print(average)
```

Uses of the Lambda Functions (review) »

- Or you can use multiple arguments using the same syntax :

```
1 print((lambda x, y: (x+y)/2)(3, 5)) # takes two int,  
    returns mean of them
```

```
1 4.0
```

- You can also assign the lambda statement in parentheses to a variable :

```
1 average = (lambda x, y: (x+y)/2)(3, 5)  
2 print(average)
```

```
1 4.0
```


Uses of the `lambda` Functions



► Task :

- Define a `lambda` function to reverse the elements of any iterables.
- Use **parentheses** for arguments and print the result.

Uses of the `lambda` Functions



► The code can be as :

```
1 iterable = "clarusway"
2
3 reverser = (lambda x : x[::-1])(iterable)
4
5 print(reverser)
6
```

Output

```
yawsuralc
```



Uses of the `lambda` Functions

► Task :

- Write a Python program that types 'even' or 'odd' in accordance with the numbers in a `list`.
- Use `lambda` function and loop.
- Your code must contain no more than 2 lines.
- The sample `list` and desired output are as follows :

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

Output

```
1 : odd
2 : even
3 : odd
4 : even
```

19



Uses of the `lambda` Functions

► The code can be as :

```
1 for x in [6, 12, -5, 11]:
2     print(x, ":", (lambda x: "odd" if x%2 != 0 else "even")(x))
3
```

Output

```
6 : even
12 : even
-5 : odd
11 : odd
```

20

Uses of the lambda Functions (review)



By assigning a function object to a variable :

Second use

- ▶ Assigning a variable :

```
1 average = lambda x, y: (x+y)/2
2 print(average(3, 5)) # we call
```

What is the output? Try to figure out in your mind...



Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

21

Uses of the lambda Functions (review)



- ▶ Using variable :

```
1 average = lambda x, y: (x+y)/2
2 print(average(3, 5)) # we call
```

```
1 4.0
```

Uses of the `lambda` Functions (review)



► Task :

- ▷ Define a `lambda` function to reverse the elements of any iterables.
- ▷ Use **variable** for arguments and print the result.

Uses of the `lambda` Functions (review)



► The code can be as :

```
1 iterable = "clarusway"
2
3 reverser = lambda x : x[::-1]
4
5 print(reverser(iterable))
6
```

Output

```
yawsuralc
```



Third use

3

Lambda within Built-in (`map()`) Functions-1

CLARUSWAY©
WAY TO REINVENT YOURSELF

Lambda within Built-in (`map()`) Functions-1

► Lambda within `map()` function :

- `map()` returns a list of the outputs after applying the given function to *each element* of a given *iterable object* such as `list`, `tuple`, etc.

The basic formula syntax is : `map(function, iterable)`

Lambda within Built-in (map()) Functions-1



- Let's square all the numbers in the list using `map()` and `lambda`. Consider this *pre-class* example :

```
1 iterable = [1, 2, 3, 4, 5]
2 map(lambda x:x**2, iterable)
3 result = map(lambda x:x**2, iterable)
4 print(type(result)) # it's a map type.
5
6 print(list(result)) # we've converted it to list type to print
7
8 print(list(map(lambda x:x**2, iterable))) # you can print directly
```

What is the output? Try to figure out in your mind...



Pear Deck Interactive Slide
Do not remove this bar

27

Lambda within Built-in (map()) Functions-1



- The output of this *pre-class* example :

```
1 iterable = [1, 2, 3, 4, 5]
2 map(lambda x:x**2, iterable)
3 result = map(lambda x:x**2, iterable)
4 print(type(result)) # it's a map type.
5
6 print(list(result)) # we've converted it to list type to print
7
8 print(list(map(lambda x:x**2, iterable))) # you can print directly
```

```
1 <class 'map'>
2 [1, 4, 9, 16, 25]
3 [1, 4, 9, 16, 25]
```

Lambda within Built-in (map()) Functions-1



► Task :

- ▷ Do the same thing using user-defined function (`def`).
- ▷ Use the `def` in `map()` function.

Lambda within Built-in (map()) Functions-1



- Disadvantages of the `def` :

```
1 def square(n):    # at least two additional lines of code
2     return n**2
3
4 iterable = [1, 2, 3, 4, 5]
5 result = map(square, iterable)
6 print(list(result))
```

Lambda within Built-in (map()) Functions-1



- Multiple arguments in `lambda` function using `map()` :

```
1 letter1 = ['o', 's', 't', 't']
2 letter2 = ['n', 'i', 'e', 'w']
3 letter3 = ['e', 'x', 'n', 'o']
4 numbers = map(lambda x, y, z: x+y+z, letter1, letter2, letter3)
5
6 print(list(numbers))
```

What is the output? Try to figure out in your mind...

- In the above example, we have combined three strings using `+` operator in `lambda` definition.



Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

31

Lambda within Built-in (map()) Functions-1



- The output :

```
1 letter1 = ['o', 's', 't', 't']
2 letter2 = ['n', 'i', 'e', 'w']
3 letter3 = ['e', 'x', 'n', 'o']
4 numbers = map(lambda x, y, z: x+y+z, letter1, letter2, letter3)
5
6 print(list(numbers))
```

```
1 ['one', 'six', 'ten', 'two']
```

- In the above example, we have combined three strings using `+` operator in `lambda` definition.

Tips :

- Note that `map()` takes each element from iterable objects one by one and in order.

Lambda within Built-in (map()) Functions-1

Task :

- Using `lambda` in `map()` function, Write a program that calculates the arithmetic means of two element pairs in the following two `lists` in accordance with **their order** and collects them into a `list`.

```
1 nums1 = [9,6,7,4]
2 nums2 = [3,6,5,8]
```

Output

```
[6.0, 6.0, 6.0, 6.0]
```

Lambda within Built-in (map()) Functions-1

The code can be as follows :

```
1 nums1 = [9,6,7,4]
2 nums2 = [3,6,5,8]
3
4 numbers = map(lambda x, y: (x+y)/2, nums1, nums2)
5
6 print(list(numbers))
7
```

Lambda within Built-in (map()) Functions-1



► Task :

- Using `lambda` in `map()` function, write a program that sets three meaningful sentences derived from the elements in the following three `lists` in accordance with **their order**.
- Print these sentences on separate lines.

```
1 words1 = ["you", "much", "hard"]
2 words2 = ["i", "you", "he"]
3 words3 = ["love", "ate", "works"]
```

Lambda within Built-in (map()) Functions-1



► The code can be as follows :

```
1 words1 = ["you", "much", "hard"]
2 words2 = ["i", "you", "he"]
3 words3 = ["love", "ate", "works"]
4
5 sentences = map(lambda x, y, z: x + " " + y + " " + z, words2, words3, words1)
6
7 for i in sentences: # attention here! The "sentences" is an iterable
8     print(i)
9
```

Output

```
i love you
you ate much
he works hard
```



Third use

4

Lambda within Built-in (`filter()`) Functions-2

CLARUSWAY©
WAY TO REINVENT YOURSELF

Lambda within Built-in (`filter()`) Functions-2 ➡

► Lambda within `filter()` function :

- `filter()` filters the given sequence (iterable objects) with the help of a function (`lambda`) that tests each element in the sequence to be `True` or not.

The basic formula syntax is : `filter(function, sequence)`

Lambda within Built-in (filter()) Functions-2

- Filtering the even numbers :

```
1 first_ten = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2
3 even = filter(lambda x:x%2==0, first_ten)
4 print(type(even)) # it's 'filter' type,
5                  # in order to print the result,
6                  # we'd better convert it into the list type
7
8 print('Even numbers are :', list(even))
```

Lambda within Built-in (filter()) Functions-2

- The output :

```
1 first_ten = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2
3 even = filter(lambda x:x%2==0, first_ten)
4 print(type(even)) # it's 'filter' type,
5                  # in order to print the result,
6                  # we'd better convert it into the list type
7
8 print('Even numbers are :', list(even))
```

```
1 <class 'filter'>
2 Even numbers are : [0, 2, 4, 6, 8]
```

Lambda within Built-in (filter()) Functions-2 ➤

► Task :

- Using `lambda` in `filter()` function, write a program that filters out words (elements of the given `list`) with less than 5 chars.
- Print these words which has less than 5 chars on separate lines.

```
1 words = ["apple", "swim", "clock", "me", "kiwi", "banana"]  
2
```

Lambda within Built-in (filter()) Functions-2 ➤

► The code can be as follows :

```
1 words = ["apple", "swim", "clock", "me", "kiwi", "banana"]  
2  
3 for i in filter(lambda x: len(x)<5, words):  
4     print(i)  
5
```

Output

```
swim  
me  
kiwi
```

Lambda within Built-in (`filter()`) Functions-2

► Task :

- This time, let's filter the vowels from the given letters in a **list**.
- Print these letters in a **list**.

```
1  
2 first_ten = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']  
3
```

Lambda within Built-in (`filter()`) Functions-2

- The code should look like :

```
1 vowel_list = ['a', 'e', 'i', 'o', 'u']  
2 first_ten = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']  
3  
4 vowels = filter(lambda x: True if x in vowel_list else False, first_ten)  
5  
6 print('Vowels are :', list(vowels))
```

```
1 Vowels are : ['a', 'e', 'i']
```

- We draw your attention to this issue that *lambda definition* we use in this example gives only **True** or **False** as a result.



Last use

4

Lambda within User-Defined Functions

CLARUSWAY©
WAY TO REINVENT YOURSELF

Lambda within User-Defined Functions



► Lambda within def :

```
1 def modular_function(n):  
2     return lambda x: x ** n  
3  
4 power_of_2 = modular_function(2) # first sub-function derived from def  
5 power_of_3 = modular_function(3) # second sub-function derived from def  
6 power_of_4 = modular_function(4) # third sub-function derived from def  
7  
8 print(power_of_2(2)) # 2 to the power of 2  
9 print(power_of_3(2)) # 2 to the power of 3  
10 print(power_of_4(2)) # 2 to the power of 4
```

What is the output?
Try to figure out in your mind...



Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

46

Lambda within User-Defined Functions

► Lambda within def :

```
1 def modular_function(n):  
2     return lambda x: x ** n  
3  
4 power_of_2 = modular_function(2) # first sub-function derived from def  
5 power_of_3 = modular_function(3) # second sub-function derived from def  
6 power_of_4 = modular_function(4) # third sub-function derived from def  
7  
8 print(power_of_2(2)) # 2 to the power of 2  
9 print(power_of_3(2)) # 2 to the power of 3  
10 print(power_of_4(2)) # 2 to the power of 4
```

```
1 4  
2 8  
3 16
```

Lambda within User-Defined Functions

► Task : (pre-class content)

- We can define a function with the same logic as the previous example that repeats the string passed into it.
- Define a function (**def**) named **repeater** using **lambda** to print the string **n** times.

Lambda within User-Defined Functions

- ▶ The sample code and the output :

```
1 def repeater(n):
2     return lambda x: x * n
3
4 repeat_2_times = repeater(2) # repeats 2 times
5 repeat_3_times = repeater(3) # repeats 3 times
6 repeat_4_times = repeater(4) # repeats 4 times
7
8 print(repeat_2_times('alex '))
9 print(repeat_3_times('lara '))
10 print(repeat_4_times('linda '))
```

```
1 alex alex
2 lara lara lara
3 linda linda linda linda
```

Lambda within User-Defined Functions

▶ Task :

- ▶ Define a *simple* function (`def`) named `functioner` using `lambda` to create your own `print` function with **emoji faces**. Such as :

```
1 # these functions were derived from the "functioner" function
2 myPrint_smile("hello")
3 myPrint_sad("hello")
4 myPrint_neutral("hello")
```

Output

```
hello :)
hello :(
hello :|
```

Lambda within User-Defined Functions

- ▶ The sample code and the output :

```
1 def functioner(emoji=None):  
2     return lambda message : print(message, emoji)  
3  
4 myPrint_smile = functioner(":)")  
5 myPrint_sad = functioner(":(")  
6 myPrint_neutral = functioner(":|")  
7
```

THANKS!

End of the Lesson (Lambda Functions)

next Lesson

Loading Modules

click above

