

Day21_lambda_functions

- The functions without any specific name (anonymous) and are provided in a single line using the following syntax

Lambda function definition

lambda formalparameters : expression

Lambda function calling : It is called up in two ways

1. Variable access method
2. IIFE (Immediately Invokable Function Execution) method

1. Variable access method

Vaname = lambda formalparameters : expression

Vaname(actual parameters)

#Without Lambda

```
def fun_sq(num):  
    return num*num
```

```
fun_sq(15)
```

```
#225
```

#with Lambda

Ex: Get the square of a number

```
fnsq = lambda num : num*num
```

```
fnsq(20)
```

```
#400
```

2. IIFE (Immediately Invokable Function Execution) method

(lambda formalparameters : expression)(actual parameters)

```
(lambda num: num*num)(10)
```

```
#100
```

In Lambda function we can provide any number of parameters but we can have only a single expression

```
(lambda num1,num2: num1*num2)(10,15)
```

```
#150
```

We can use the shortcut notations in the lambda function for performing validations

#Check the largest of two numbers

```
def fun_large():
```

```
    a,b = 15,18
```

```
    if a>b:
```

```
        print(a)
```

```
    else:
```

```
        print(b)
```

```
(lambda a,b:print(a) if a>b else print(b))(15,18)
```

```
fun_large = lambda a,b:a if a>b else b
```

```
fun_large(10,15)
```

```
#15
```

#Check the largest of three numbers

```
fun_large = lambda a,b,c:a if a>b and a>c else b if b>c else c
```

```
fun_large(10,15,21)
```

#the below function is equivalent to the above

```
def fun_large(a,b,c):
```

```
    if a>b and a>c:
```

```

print(a)
else:
    if b>c:
        print(b)
    else:
        print(c)

```

```
fun_large(10,15,21)
```

Usage of lambda functions

The lambda functions are used in

1. filter function
2. map function
3. reduce function

1. filter function : This functions takes up a sequence of values and return a collection of values depending on a given function criteria

Syntax: filter(function,sequence)

Ex: lst = [12,18,21,33,52,61,64]

Get the list of even numbers only

```
lst = [12,18,21,33,52,61,64]
```

```
def isEven(val):
    if val%2==0:
        return True
    else:
        return False

```

```
isEven(18)
#True

```

```
isEven(21)
#False

```

```
list(filter(isEven,lst))
#[12, 18, 52, 64]

```

#Using lambda functions

```
list(filter(lambda val:val%2==0,lst))
#[12, 18, 52, 64]

```

2. map function: It acts on each and every element and brings up a change depending on the given function criteria

Syntax: map(function, sequence)

#Take a sequence of numbers and get the cube of each element

```
lst = [5,6,8,9,11]
```

```
list(map(lambda val:val**3,lst))
#[125, 216, 512, 729, 1331]

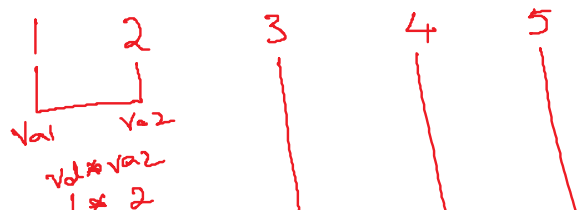
```

3. reduce : It is all about taking a sequence of values and returning a single value. It is available in functools module

Syntax: reduce(function,sequence)

```
import functools as ft
```

```
ft.reduce(lambda va1,va2:va1*va2,range(1,6))
#120
```



Val va2
val * va2
1 * 2
2
Val

3
va2
2 + 3
6
va2.

va2
va2
24
Val
24 * 5 = 120