



Python

Experiment No. 7

Aim: To implement Lambda, map and reduce functions

Problem Statements:

1. Write a Python program to sort a list of tuples using Lambda.
2. Write a Python program to square and cube every number in a given list of integers using Lambda with map.
3. Write a Python program to add two given lists and find the difference between them. Use the map() function with function.
4. Write a Python program that creates a list of dictionaries containing student information (name, age, grade) and uses the filter function to extract students with a grade greater than or equal to 95.
5. WAP to calculate the product of the numbers from 1 to 100 using reduce function.

Theory:

Lamba function:

The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name. These functions are throw-away functions, i.e. they are just needed where they have been created. Lambda functions are mainly used in combination with the functions filter(), map() and reduce().

Syntax:

`lambda arguments_list : expression`

The argument list consists of a comma separated list of arguments and the expression is an arithmetic expression using these arguments. You can assign the function to a variable to give it a name.

Example:

```
calc = lambda num: "Even number" if num % 2 == 0 else "Odd number"

print(calc(20))

f = lambda x, y : x + y

print f(1,1)
```

Python lambda properties:

- This function can have any number of arguments but only one expression, which is evaluated and returned.
- One is free to use lambda functions wherever function objects are required.
- lambda functions are syntactically restricted to a single expression.
- It has various uses in particular fields of programming, besides other types of expressions in functions.

Example 1: Program to demonstrate return type of Python lambda keyword

Code:

```
string = 'GeeksforGeeks'
# lambda returns a function object
print(lambda string: string)
```

Output:

```
>>> <function <lambda> at 0x7d4d67a4bac0>
```

Example 2: Invoking lambda return value to perform various operations

Here we have passed various types of arguments into the different lambda functions and printed the result generated from the lambda function calls.

Code:

```
filter_nums = lambda s: ''.join([ch for ch in s if not
ch.isdigit()])

print("filter_nums():", filter_nums("Geeks101"))
do_exclaim = lambda s: s + '!'
print("do_exclaim():", do_exclaim("I am tired"))

find_sum = lambda n: sum([int(x) for x in str(n)])
print("find_sum():", find_sum(101))
```

Output:

```
>>> filter_nums(): Geeks
do_exclaim(): I am tired!
find_sum(): 2
```

Example 3: The lambda function gets more helpful when used inside a function.

We can use lambda function inside map(), filter(), sorted() and many other functions.

map()

map() is a function which takes two arguments:

```
r = map(func, seq)
```

The first argument func is the name of a function and the second a sequence (e.g. a list) seq. map() applies the function func to all the elements of the sequence seq. Before Python3, map() used to return a list, where each element of the result list was the result of the function func applied on the corresponding element of the list or tuple "seq". With Python 3, map() returns an iterator.

The following example illustrates the way of working of map():

Code:

```
def fahrenheit(T):  
    return ((float(9)/5)*T + 32)  
temperatures = (36.5, 37, 37.5, 38, 39)  
F = map(fahrenheit, temperatures)  
temperatures_in_Fahrenheit = list(map(fahrenheit, temperatures))  
print(temperatures_in_Fahrenheit)
```

In the example above we haven't used lambda. By using lambda, we wouldn't have had to define and name the functions fahrenheit() and celsius(). You can see this in the following interactive session:

```
C=[float(39.2),float(36.537),float(38),float(37.8)]  
F=list(map(lambda x:(float(9)/5)*x+32,C))  
print(F)
```

Output:


```
[97.7, 98.60000000000001, 99.5, 100.4, 102.2]  
[102.56, 97.7666, 100.4, 100.03999999999999]
```

map() can be applied to more than one list. The lists don't have the same length. map() will apply its lambda function to the elements of the argument lists, i.e. it first applies to the elements with the 0th index, then to the elements with the 1st index until the n-th index is reached:

Code:

```
a = [1, 2, 3, 4]  
b = [17, 12, 11, 10]  
c = [-1, -4, 5, 9]  
list(map(lambda x, y, z : x+y+z, a, b, c))
```

Output:

 [17, 10, 19, 23]

If one list has fewer elements than the others, map will stop when the shortest list has been consumed.

Filter()

The function `filter(function, sequence)` offers an elegant way to filter out all the elements of a sequence "sequence", for which the function function returns True. i.e. an item will be produced by the iterator result of `filter(function, sequence)` if item is included in the sequence "sequence" and if `function(item)` returns True.

In other words: The function `filter(f,l)` needs a function `f` as its first argument. `f` has to return a Boolean value, i.e. either True or False. This function will be applied to every element of the list `l`. Only if `f` returns True will the element be produced by the iterator, which is the return value of `filter(function, sequence)`.


In the following example, we filter out first the odd and then the even elements of the sequence of the first 11 Fibonacci numbers:

Code:

```
fibonacci = [0,1,1,2,3,5,8,13,21,34,55]
odd_numbers = list(filter(lambda x: x % 2, fibonacci))
print(odd_numbers)

even_numbers = list(filter(lambda x: x % 2 == 0, fibonacci))
print(even_numbers)
```

Output:

 [1, 1, 3, 5, 13, 21, 55]
[0, 2, 8, 34]

Reduce()

The function `reduce(func, seq)` continually applies the function `func()` to the sequence `seq`. It returns a single value.

If `seq = [s1, s2, s3, ... , sn]`, calling `reduce(func, seq)` works like this:

At first the first two elements of `seq` will be applied to `func`, i.e. `func(s1,s2)` The list on which `reduce()` works looks now like this: `[func(s1, s2), s3, ... , sn]`

In the next step `func` will be applied on the previous result and the third element of the list, i.e. `func(func(s1, s2),s3)`

The list looks like this now:

[func(func(s1, s2),s3), ... , sn]

Continues like this until just one element is left and returns this element as the result of reduce()

If n is equal to 4 the previous explanation can be illustrated like this:

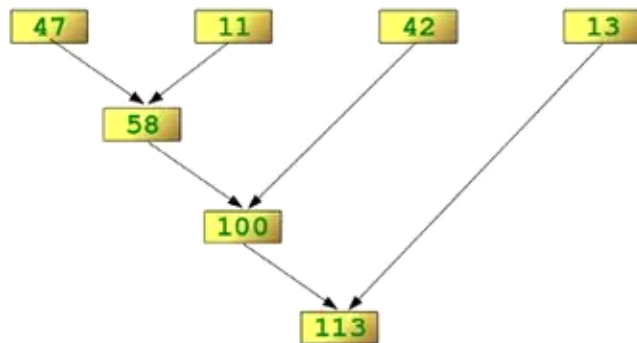
$[s_1, s_2, s_3, s_4]$
 $\text{func}(s_1, s_2)$
 $\text{func}(\text{func}(s_1, s_2), s_3)$
 $\text{func}(\text{func}(\text{func}(s_1, s_2), s_3), s_4)$

We have to import functools to be capable of using reduce:

```
from functools import reduce
```

```
f=lambda x,y: x+y
```

```
Print (reduce(f, [47,11,42,13]))
```



Q.1 Write a Python program to sort a list of tuples using Lambda.

Code:

```
n = int(input("Enter number of Elements : "))
list_tuples = [tuple(int(i) for i in input("Enter elements
in tuple (seperated by comma) : ").split(",")) for j in
range(n)]

print("\nList of Tuples : ")
print(list_tuples)

sorted_tuples = list(map(lambda x:sorted(x), list_tuples))

print("\nSorted List of Tuples : ")
print(sorted_tuples)
```

Output:

```
➡ Enter number of Elements : 2
Enter elements in tuple (seperated by comma) : 10,15,6
Enter elements in tuple (seperated by comma) : 12,50,-2

List of Tuples :
[(10, 15, 6), (12, 50, -2)]

Sorted List of Tuples :
[[6, 10, 15], [-2, 12, 50]]
```

Q.2 Write a Python program to square and cube every number in a given list of integers using Lambda with map.

Code:

```
list_ip = [int(i) for i in input("Enter numbers in list
(seperated by comma) : ").split(",")]

square = list(map(lambda x:x**2, list_ip))
cube = list(map(lambda x:x**3, list_ip))

print("Square of numbers : ", square)
print("Cube of numbers : ", cube)
```

Output:

```
➡ Enter numbers in list (seperated by comma) : 2,3,10,15
Square of numbers : [4, 9, 100, 225]
Cube of numbers : [8, 27, 1000, 3375]
```

Q.3 Write a Python program to add two given lists and find the difference between them. Use the map() function with function.

Code:

```
list1 = [int(i) for i in input("Enter 5 numbers in list 1
(seperated by comma) : ").split(",")]
list2 = [int(i) for i in input("Enter 5 numbers in list 2
(seperated by comma) : ").split(",")]

def add(x,y):
    return x+y

def diff(x,y):
    return x-y

print("\nSum of both lists : ", list(map(add, list1,
list2)))
print("\nDifference of both lists : ", list(map(diff, list1,
list2)))
```

Output:

```
➡ Enter 5 numbers in list 1 (seperated by comma) : 1,2,3,4,5
Enter 5 numbers in list 2 (seperated by comma) : 2,4,6,8,10

Sum of both lists : [3, 6, 9, 12, 15]

Difference of both lists : [-1, -2, -3, -4, -5]
```


Q.4 Write a Python program that creates a list of dictionaries containing student information (name, age, grade) and uses the filter function to extract students with a grade greater than or equal to 95.

Code:

```
student_list = []
student_data = {}
for i in range(int(input("Enter number of students : "))):
    print()
    name = input("Enter name of student : ")
    age = int(input("Enter age of student : "))
    grade = int(input("Enter grade of student : "))
    student_data = {"Name":name, "Age":age, "Grade":grade}
    student_list.append(student_data)

print("\nList of Dictionaries : ")
print(student_list)

print("\nStudents with grade greater than or equal to 95 : ")
print(list(filter(lambda x:x["Grade"]>=95, student_list)))
```

Output:

```
Enter number of students : 3

Enter name of student : ABC
Enter age of student : 19
Enter grade of student : 96

Enter name of student : DEF
Enter age of student : 20
Enter grade of student : 92

Enter name of student : GHI
Enter age of student : 19
Enter grade of student : 95

List of Dictionaries :
[{'Name': 'ABC', 'Age': 19, 'Grade': 96}, {'Name': 'DEF', 'Age': 20, 'Grade': 92}, {'Name': 'GHI', 'Age': 19, 'Grade': 95}]

Students with grade greater than or equal to 95 :
[{'Name': 'ABC', 'Age': 19, 'Grade': 96}, {'Name': 'GHI', 'Age': 19, 'Grade': 95}]
```

