

## PYTHON NOTES-3

### ADVANCED PYTHON FUNCTIONAL PROGRAMMING

#### FUNCTIONAL PROGRAMMING

- Functional programming also separates data and functions.
- It is a declarative type of programming style. Its main focus is on “**what to solve**” in contrast to an imperative style where the main focus is “**how to solve**”. It uses expressions instead of statements. An expression is evaluated to produce a value whereas a statement is executed to assign variables.
- In python, we keep data and functions totally separated.

```
def multiply_by2(any_list):  
    new_list = []  
    for item in any_list:  
        new_list.append(item*2)  
    return new_list
```

-----> function

```
print(multiply_by2([1,2,3]))
```

-----> data

Any Functional programming language is expected to follow these concepts-

- 1. Pure Function
- 2. Recursion
- 3. Functions are First-Class and can be Higher-Order
- 4. Variables are Immutable.

#### PURE FUNCTIONS

These functions have two main properties-

- It always produces the same output for the same arguments. For example, 3+7 will always be 10 no matter what.
- It does not change or modifies the input variable. The second property is also known as immutability.

# MAP FUNCTION

- map () function returns a map object (which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)

## Syntax

*map (function, iterables)*

| Parameter       | Description  |
|-----------------|--|
| <i>function</i> | Required. The function to execute for each item  |
| <i>iterable</i> | Required. A sequence, collection or an iterator object. You can send as many iterables as you like, just make sure the function has one parameter for each iterable. |

```
#example-1
def myfunc(n):
    return len(n)

x = map(myfunc, ('apple', 'banana', 'cherry'))
print(list(x))

#example-2

def any_func(a,b):
    return (a + b)

y=map(any_func,('kaju' , 'cham', 'gulab'),('katli','cham',
'jamun'))
print(list(y))
```

```
map examples.py > ...
1  def multiply_by2(item):
2      return item*2
3
4  y = map(multiply_by2,[7,6,4])
5
6  print(y)           #object location
7  print(list(y))     #output, we converted the output to list for better readability
8
```

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

```
Mohd.Uzair@UzairPC MINGW64 /g/python_practice
$ env C:\\Users\\Mohd.Uzair\\AppData\\Local\\Programs\\Python\\Python38-32\\python.exe c:\\Users\\Mohd.Uzai
\\pythonFiles\\lib\\python\\debugpy\\no_wheels\\debugpy\\launcher 50593 -- g:\\python_practice\\functionalpr
<map object at 0x03841FD0>
[14, 12, 8]
```

```
main.py saved https://opp.khanuzair.repl.run
1 def squaring(a):
2     return a**2
3 z=map(squaring,[1,2,3,4,5,6,7,8,9,10])
4
5 print(z)
6 print(list(z))
```

```
<map object at 0x7f3e7622a370>
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

## Filter Function

- we can iterate map or filter object only once. But we can iterate list or tuple 'n' number of times.
- The filter () method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.

SYNTAX- Filter (function, sequence)

- **Parameters:**
- function: function that tests if each element of a
- sequence true or not.
- sequence: sequence which needs to be filtered, it can
- be sets, lists, tuples, or containers of any iterators.
- **Returns:**
- returns an iterator that is already filtered.

Example- from list y having numbers and we will filter even numbers using filter function

```
filter func example.py > ...
1 def any_num(item):
2     return item % 2 == 0 #we want to return only even number here
3
4 y = [4,5,8,9,7,122] #give y list contain number
5
6 evens = filter(any_num,y) #filtering even number from list y
7 print(list(evens))       #display list even
8
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Mohd.Uzair@UzairPC MINGW64 /g/python_practice
$ env C:\\Users\\Mohd.Uzair\\AppData\\Local\\Programs\\Python\\Python38-32\\python.exe c:
\\pythonFiles\\lib\\python\\debugpy\\no_wheels\\debugpy\\launcher 55013 -- "g:\\python_pr
[4, 8, 122]
```

Example- from given number list z we have to filter odd numbers

```
def func1(item):  
    return item % 2 != 0  
  
z=(67,88,90,76,22,87)  
  
only_odd = filter(func1,z)  
print(tuple(only_odd))
```

```
(67, 87)
```

Example- given list of letters we have to segregate the vowels using filter here-

filter func example2.py X

filter func example2.py > ...

```
1  def classify(item):  
2      vowel = ['a', 'e', 'i', 'o', 'u']  
3      if item in vowel:  
4          return True  
5      else:  
6          return False  
7  
8  letter = ['q','e','e','p','z','g','u']  
9  literal = filter(classify,letter)  
10 print(literal)  
11 print(set(literal))  
12
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Mohd.Uzair@UzairPC MINGW64 /g/python\_practice

```
$ env C:\\Users\\Mohd.Uzair\\AppData\\Local\\Programs\\Python\\  
\\pythonFiles\\lib\\python\\debugpy\\no_wheels\\debugpy\\laur  
<filter object at 0x01541FA0>
```

```
{'u', 'e'}
```

## ZIP FUNCTION

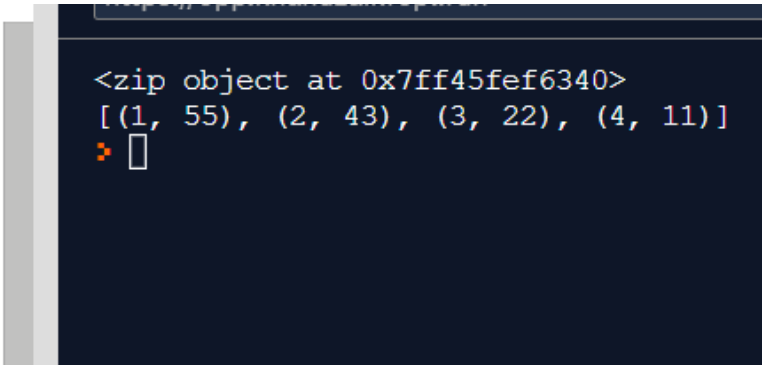
- The `zip ()` function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.
- If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.

**Syntax:** `zip (iterator1, iterator2, iterator3 ...)`

Example- we have one list and one tuple. We are zipping it.

```
a = [1,2,3,4,5]
b = (55,43,22,11)
```

```
y = zip(a,b)
print(y)
print(list(y))
```



```
<zip object at 0x7ff45fef6340>
[(1, 55), (2, 43), (3, 22), (4, 11)]
```

## REDUCE FUNCTION

- The `reduce ()` function is used to reduce the sequence of elements to a single value by processing the elements according to a function supplied. It returns a single value.
- This function is a part of `functools` module so you have to import it before using it.

**Syntax:**

#special note:

```
from functools import reduce #way1- it will only import reduce function
from functools import * #way2- it will import all the function
#we have choice , we can give either of the command here
```

```
'''SYNTAX'''
```

```
reduce(your_function, your_sequence, initialization)
```

Example- we have list and we are reducing the list according to function we define. Here we want to add the elements of list.

```
from functools import reduce
my_list = [1,2,3,4,5]
def accumulator(acc,item):
    return acc + item

result = reduce(accumulator,my_list,0) #we are
initializing acc = 0
print(result)
```



## CODE EXERCISE-

#1 Capitalize all of the pet names and print the list

```
my_pets = ['sisi', 'bibi', 'titi', 'carla']
def capital(item):
    return item.upper()
x = map(capital,my_pets)
print(list(x))
```

#2 Zip the 2 lists into a list of tuples, but sort the numbers from lowest to highest.

```
my_strings = ['a', 'b', 'c', 'd', 'e']
my_numbers = [5,4,3,2,1]
y=zip(my_strings,my_numbers)
print(list(y))
```

#3 Filter the scores that pass over 50%

```
scores = [73, 20, 65, 19, 76, 100, 88]
def passing_marks(marks):
    if marks > 50:
        return True
    else:
        return False
z = filter(passing_marks,scores)
print(list(z))
```

#4 Combine all of the numbers that are in a list on this file using reduce (my\_numbers and scores).What is the total?

```
from functools import reduce
def accumulator(acc,base):
    return acc + base
p = reduce(accumulator,my_numbers+scores,0)
print(p)
```

# LAMBDA EXPRESSIONS

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.

**Syntax:**            **lambda arguments: expression**

Examples:

```
#lambda expressions
lambda a : a +10
lambda b : b**2
lambda c : c%2
```

Whatever we were doing with map, zip, filter, reduce functions in one way. We can do this using lambda expression too.

lambda expression example.py ●

lambda expression example.py > ...

```
1  my_list = [ 1,2,3]
2  z = map(lambda a:a*2,my_list)    #for multiplyin item by 2
3  print(z)
4  print(tuple(z))
5  y = filter(lambda item: item%2==0,my_list)  #for segregating evens
6  print(list(y))
7  from functools import reduce
8  p=reduce(lambda m,n : m+n,my_list)  #adding items of list
9  print(p)
```

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

```
$ env C:\\Users\\Mohd.Uzair\\AppData\\Local\\Programs\\Python\\Python38-32\\python.exe c:\\
\\pythonFiles\\lib\\python\\debugpy\\no_wheels\\debugpy\\launcher 57466 -- "g:\\python_prac
<map object at 0x00791FA0>
(2, 4, 6)
[2]
6
```

Example- Given a list of tuples, sort the list on the basis of second element of tuple using lambda function. x[1] represent second element .

```
a = [(1,5),(-2,4),(2,-1),(-0.6,-10)]
a.sort(key=lambda x:x[1])  #we gave key = second element
of tuple of every item of list, on the basis of second
elemnt we are sorting the data.if we do not give key , it
will sort on the basis of first element as default.
print(a)
```

```
[(-0.6, -10), (2, -1), (-2, 4), (1, 5)]
```

# COMPREHENSIONS

- Comprehensions in Python provide us with a short and concise way to construct new sequences (such as lists, set, dictionary etc.) using sequences which have been already defined. Python supports the following 4 types of comprehensions:
  - 1.List Comprehensions
  - 2.Dictionary Comprehensions
  - 3.Set Comprehensions
  - 4.Generator Comprehensions

## LIST COMPREHENSION

- List Comprehensions provide an elegant way to create new lists. The following is the basic structure of a list comprehension:  
Syntax:

```
output_list = [output_exp for var in input_list if  
(var satisfies this condition)]
```

```
'''LIST COMPREHENSION'''  
  
a = [item for item in range(1,10)] #creating a list  
having item in range of 1 to 10  
print(a)  
  
print('-----')  
  
b= [item*2 for item in range(1,10)] #list having item*2  
print(b)  
  
print('-----')  
  
c=[char for char in 'MAYUR VIHAR'] #list having all char  
in the word 'MAYR+UR VIHAR'  
print(c)  
  
print('-----')  
  
d=[item**2 for item in range(1,11) if item**2%2==0]  
print(d) #list printing even squares for number in range  
1,11
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]  
-----  
[2, 4, 6, 8, 10, 12, 14, 16, 18]  
-----  
['M', 'A', 'Y', 'U', 'R', ' ', 'V', 'I', 'H', 'A', 'R']  
-----  
[4, 16, 36, 64, 100]  
> []
```

## DICTIONARY COMPREHENSION

- we can also create a dictionary using dictionary comprehensions.

Syntax

```
output_dict = {key:value for (key, value) in iterable if  
(key, value satisfy this condition)}
```



```
1  ✓ a_dict = {
2      'a':2,
3      'b':3,
4      'c':4,
5      'd':5
6  }
7  new_dict = {key:value**2 for key,value in a_dict.items() if value%2==0}
8  print(new_dict)
```

```
print("Output Dictionary using dictionary comprehensions:",  
      dict_using_comp)
```

Output Dictionary using dictionary comprehensions: {'Rajasthan': 'Jaipur',  
'Maharashtra': 'Mumbai',  
'Gujarat': 'Gandhinagar'}