python-refreshment-part-2

March 11, 2024

Python Refreshment Part 2

Written by: Prashant Sundge

Table of Contents

Python Refreshment Part 2: Content

1 Lambda Functions

- Anonymous functions are those which does not contain name (nameless).
- The purpose of lambda funciton is to perform instant operations which will require at that point and no longer needed futher like functions
- lamda funciton contains only one statement or Expression not like functions blocks of statements

- Lambda function returns the value implicitly or automatically when it executes
  - Syntax **varname=lambda params-list :expression**
    - OR
  - Syntax **varname=lambda params-list: Statement**

1.0.1 Explaination

- here lambda is keyword which is used to define lambda functions
- param-list represents list of variables used for storing inputs coming fucniton call
- Expression/Statements represents and executable statement and whose result returns automatically (no need of return statement)

Question: Define a function for cal addition of two numbers

[ ]:
```
# By Using Normal Function

def calc_add(a,b):
  c= a+b
  return c

#main program
result=calc_add(10,20),
print("Addition : {}".format(result))
```

```
Addition : 30
```

[ ]:
```
# by using lambda function
result = (lambda x,y : x+y)
print(result(10,20))
```

```
30
```

[ ]:
```
#program accepting Two value and find biggest among them by using Anonymous␣
 ↪Functions
##AnonymousFunsEx2.py
a=int(input('Enter a value'))
b=int(input('Enter second value'))

result = lambda x,y : x if x > y else y if y > x else 'Both are same'
print(result(a,b))
```

```
Enter a value25
Enter second value25
Both are same
```

[ ]:
```
#program accepting Three value and find biggest among them by using Anonymous␣
 ↪Functions
#AnonymousFunsEx3.py
```

```
a=int(input('Enter First value'))
b=int(input('Enter second value'))
c=int(input('Enter Third value'))

print("-" * 18 + "Normal Function" + "-" * 9)

def normal_big(x,y,z):
  if x > y and x > z:
    print("Big Number : {}".format(x))
  elif y > x and y > z:
    print("Big Number : {}".format(y))
  elif z > x and z > y :
    print("Big Number : {}".format(z))
  else:
    print('ALL ARE SAME : {} {} {}'.format(x,y,z))
#main program
normal_big(a,b,c)
print("-" * 18 + "Lambda Function" + "-" * 9)
big = lambda x,y,z : x if (x > y) and (x > z) else y if (y > x) and (y > z)␣
 ↪else z if (z > x) and (z > y) else 'All Are Same'
print(big(a,b,c))
```

```
Enter First value25
Enter second value25
Enter Third value25
------------------Normal Function---------
ALL ARE SAME : 25 25 25
------------------Lambda Function---------
All Are Same
```

[ ]:
```
#program accepting Three value and find biggest among them by using Anonymous␣
 ↪Functions
#AnonymousFunsEx3.py
a=int(input('Enter First value'))
b=int(input('Enter second value'))
c=int(input('Enter Third value'))

print("-" * 18 + "Normal Function" + "-" * 9)

def normal_big(x,y,z):
  if y<=x>y:
    print("Big Number : {}".format(x))
  elif z<=y>x:
    print("Big Number : {}".format(y))
  elif x <=z>y:
    print("Big Number : {}".format(z))
  else:
```

```
    print('ALL ARE SAME : {} {} {}'.format(x,y,z))
#main program
normal_big(a,b,c)
print("-" * 18 + "Lambda Function" + "-" * 9)
big = lambda x,y,z : x if (y<=x>z) else y if (z<=y>x) else z if (x <=z>y) else␣
 ↪'All Are Same'
print(big(a,b,c))
```

```
Enter First value25
Enter second value25
Enter Third value25
------------------Normal Function---------
ALL ARE SAME : 25 25 25
------------------Lambda Function---------
All Are Same
```

[ ]:
```
#program accepting word and convert into Upper case
#AnonymousFunsEx5.py
# HINT word="python" output= "PYTHON"
def conver_string(s):
  if s.isupper():
    s=s.lower()
  else:
    s=s.upper()
  return s
# main program
s=input('Enter string \t')
print("-" * 20 + "Normal Function" + "-" * 20,'\n')
res=conver_string(s)
print('Original String : {}\t Converted String : {} '.format(s,res))
print("-" * 20 + "Normal Function" + "-" * 20 ,'\n')

op = lambda x: s.lower() if s.isupper() else s.upper()
print('Original String : {}\t Converted String : {} '.format(s,op(s)))
```

```
Enter string    prashant sundge
--------------------Normal Function--------------------

Original String : prashant sundge        Converted String : PRASHANT SUNDGE
--------------------Normal Function--------------------

Original String : prashant sundge        Converted String : PRASHANT SUNDGE
```

[ ]:
```
#program accepting word/line and find number of words
#AnonymousFunsEx6.py

s='pams sundge'
```

```
count=0
for ch in s:
  if ch != ' ':
    count +=1


print(count)

scount = lambda x: sum(1 for ch in x if ch != ' ')
print(scount(s))
```

```
10
10
```

2 Special Function in Python

- filter()
- map()
- reduce()

2.1 Filter()

Syntax varname=filter(fucntion name,iterable_object) - Here varname is an object of type and we can convert into any iterable object by using casting functions - *FuncitonName* represents either normal function or anonymous functions (lambda) - *iterable objects* represents sequence list, set, tuple, dict types - The execution process of filter() is that each value of iterable object sends to funciton name if the function return True then elements will be filtered if the funciton returns False then that element will be neglected /not filtered this process will be continued until all the elments of iterable object completed

[ ]:
```
#program for obtaining / Filtering +Ve values from list of values
#FilterEx1.py
def positive(n):
  if n>0:
    return True
  else:
    return False
#main program
lst=[10,-20,30,-40,50,60,-70]
kvr=(filter(positive,lst))
print('-'*20 + 'Normal Function'+'-'*20)
print(type(kvr,))
#convert filter obejct in to list
positive_list=list(kvr)
print('Given Data:',lst)
print('Positive Data: ',positive_list)
print("+"*55)
```

```
print('-'*20 + 'Lambda Function'+'-'*20)

lam_positive_list=filter(lambda x : x>0 , lst)

print("Filter Object", lam_positive_list)
lam_positive_list=list(lam_positive_list)

print(lam_positive_list)
```

```
--------------------Normal Function--------------------
<class 'filter'>
Given Data: [10, -20, 30, -40, 50, 60, -70]
Positive Data:  [10, 30, 50, 60]
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--------------------Lambda Function--------------------
Filter Object <filter object at 0x780035435870>
[10, 30, 50, 60]
```

[ ]:
```
#program for obtaining / Filtering +Ve values from list of values
#FilterEx1.py
lst=[10,-20,30,-40,50,60,-70]
print(lst)
pos_list=filter(lambda x : x>0 , lst)
pos_list_convert=list(pos_list)
print(pos_list_convert)
neg_list=filter(lambda x : x<0 , lst)
neg_list_convert=list(neg_list)
print(neg_list_convert)
```

```
[10, -20, 30, -40, 50, 60, -70]
[10, 30, 50, 60]
[-20, -40, -70]
```

[ ]:
```
#program for obtaining / Filtering even from list
#FilterEx6.py
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
def even(n):
  even=list()
  odd=list()

  for i in n:
    if i % 2 ==0 :
      even.append(i)
    else:
      odd.append(i)

  return even , odd
```

```
#main program
even, odd = even(numbers)
print('Orginal list ', numbers)
print("Odd Number ", odd)
print('Even Number ', even)




print('-'*20 + 'Lambda Function'+'-'*20)

lambda_even = filter(lambda x: x%2==0 , numbers)
lambda_odd = filter(lambda x: x%2!=0 , numbers)

lambda_even = tuple (lambda_even)
lambda_odd = list(lambda_odd)
print('Orginal list ', numbers)
print(lambda_even, type(lambda_even))
print(lambda_odd, type(lambda_odd))
print("+"*50)



print('-'*20 + 'Lambda Function'+'-'*20)
result = filter(lambda x: x % 2 == 0, numbers), filter(lambda x: x % 2 != 0,␣
 ↪numbers)
even, odd = list(result[0]), list(result[1])
print(even, odd)
print("+"*50)
```

```
Orginal list  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Odd Number  [1, 3, 5, 7, 9]
Even Number  [2, 4, 6, 8, 10]
--------------------Lambda Function--------------------
Orginal list  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
(2, 4, 6, 8, 10) <class 'tuple'>
[1, 3, 5, 7, 9] <class 'list'>
++++++++++++++++++++++++++++++++++++++++++++++++++
--------------------Lambda Function--------------------
[2, 4, 6, 8, 10] [1, 3, 5, 7, 9]
++++++++++++++++++++++++++++++++++++++++++++++++++
```

[ ]:
```
strings = ["Hello", "world", "PYTHON", "Programming", "LANGUAGE"]

uppercase_strings = list(filter(lambda s: s.isupper(), strings))
print("Uppercase strings:", uppercase_strings)
```

```
Uppercase strings: ['PYTHON', 'LANGUAGE']
```

3 Map()

Syntax varname = map(functionName, iterable_object)

- The Execution process of map() is that map() sends every element of iterable object to the specified function process it and returns the modified value(result) and new list of elements will be obtained the process will be continued until all the eleements of iterable_object completed
- map() is used for obtaining new iterable objects from existing iterable object by applying old iterable elements to the function

[3]:
```
#write  a python program which will hike the employee salary by 15 % and store␣
 ↪it in new list
#program for obtaining newsal list from oldlist

def sal_hike(lst):
  new_sal=list()
  for i in lst:
    new_sal.append(i+i*.15)
  return new_sal
#main program
emp_sal=[20000,30000,40000,50000,60000]
new_sal=sal_hike(emp_sal)
print("Employee Present Salary ")
print(emp_sal)
print("Employee 15% hike Salary ")
print(new_sal)
```

```
Employee Present Salary
[20000, 30000, 40000, 50000, 60000]
Employee 15% hike Salary
[23000.0, 34500.0, 46000.0, 57500.0, 69000.0]
```

[5]:
```
# write a program to find their squares

def square(l):
  sqr=[]
  for i in l:
    sqr.append(i**2)
  return sqr

#main program
lst=[1,2,3,4,5,6,7,8]
sqr=square(lst)
print('Orignal List ',lst)
print('Square list',sqr)
```

```
print('-'*20 + 'Lambda Function'+'-'*20)

l_sqr= list(map(lambda x : x**2 , lst))
print('Orignal List ',lst)
print('Square list',l_sqr)
```

```
Orignal List  [1, 2, 3, 4, 5, 6, 7, 8]
Square list [1, 4, 9, 16, 25, 36, 49, 64]
--------------------Lambda Function--------------------
Orignal List  [1, 2, 3, 4, 5, 6, 7, 8]
Square list [1, 4, 9, 16, 25, 36, 49, 64]
```

[6]:
```
#Filter rows based on a condition in a DataFrame column:
import pandas as pd

# Create a DataFrame
data = {'Numbers': [10, 20, 30, 40, 50]}
df = pd.DataFrame(data)

# Filter rows where 'Numbers' column is greater than 30
filtered_df = df[df['Numbers'].map(lambda x: x > 30)]

print(filtered_df)
```

```
   Numbers
3       40
4       50
```

4 reduce()

- reduce() is used for obtaining a single element/result from given iterable object by applying to a function

Syntax varname=reduce(function-name, iterable-object)

- step1: initially reduce() selects first two values of iterable object and place them in first var and second var
- step2: The function-name (lambda or noramal function) utilize the values of first var and second var and applied to the specified logic and obtains the result.
- step3: reduce() places the result of function-name in first variable and reduce() select the succeeding element of iterable object and places in second variable
- step4: Repeat step 2 and step3 until all the elements completed in iterable object and return of first variable

[14]:
```
# find sum of lst values by using reduce()
from functools import reduce
def sum(a,b):
  return (a+b)

#main program
lst=[10,20,30,40,50]
sm=reduce(sum, lst)
print(sm)


l_sum=reduce(lambda x , y: x+y , lst)
print("Lambda reduce :", l_sum)
```

```
150
Lambda reduce : 150
```

[19]:
```
#Finding Max and min from list of elements by using reduce()
#ReduceEx3.py
import functools
print('Enter list of values seperated by space')
lst=[int(val) for val in input().split()]
maxv=functools.reduce(lambda x, y :x if  x > y else y, lst)
minv=functools.reduce(lambda x, y :x if x < y else y, lst)
print('Orignal list', lst)
print('Min value ', minv)
print('Max value ', maxv)
```

```
Enter list of values seperated by space
5 4 6 7 8 4 3 0
Orignal list [5, 4, 6, 7, 8, 4, 3, 0]
Min value  0
Max value  8
```

[25]:
```
#write a python program to accept the list of words from keyboard and make a␣
 ↪line of text using normal function
#reduceex04.py
from functools import reduce
def line_words(lst):
  line=''
  for i in lst:
    line = line+' '+i
  return line

print('Enter the list of words ')
lst=[word for word in input().split()]
print(lst)
```

```
line=line_words(lst)
print(line)

l_line=reduce(lambda x ,y : x +' '+ y , lst)
print(l_line)
```

```
Enter the list of words
pams is good boy he obey the orders of his senior
['pams', 'is', 'good', 'boy', 'he', 'obey', 'the', 'orders', 'of', 'his',
'senior']
 pams is good boy he obey the orders of his senior
pams is good boy he obey the orders of his senior
```

5 List Comprehension

Syntax:- list_object =[expression for varname in iterable_object Test_cond]

- The purpose of List Comprehension is that to read the values dynamically from key board seperated by delimeter (space, comma, colon etc)
- list comprehension is most effective way for reading the data for list instead traditional readting the data
- here Epression represents either type casting or mathematical expression

[27]:
```
lst=[3,2,4,5,6]
newlist=[val*2 for val in lst]
print("Original list",lst)
print("new list", newlist)
```

```
Original list [3, 2, 4, 5, 6]
new list [6, 4, 8, 10, 12]
```

[28]:
```
#Program accepting list of values seperate by commmas
#EvenOddListEx.py
print('Enter the values to find Even numbers seperated by comma')
lst_val=[float(val) for val in input().split(',')]
print(lst_val)
```

```
Enter the values to find Even numbers seperated by comma
4,3,5,7,8,9,0
[4.0, 3.0, 5.0, 7.0, 8.0, 9.0, 0.0]
```

[29]:
```
#Program accepting list of valuee and get the even values from the list
#EvenOddListEx.py
print('Enter the values to find Even numbers seperated by comma')
lst_val=[int(val) for val in input().split(',') if int(val) % 2 ==0 ]
print(lst_val)
```

```
Enter the values to find Even numbers seperated by comma
```

```
3,5,7,8,6,4,2
[8, 6, 4, 2]
```

[33]:
```
#write a program to print the squar values of a given list using comprehension
lst=[1,2,3,4, 5, 6, 7, 8, 9, 10]

sqr_val=[i**2 for i in lst]
print(sqr_val)
print('Values \t Squar')
for i , v in zip(lst, sqr_val):
  print(i ,'\t', v)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
Values   Squar
1        1
2        4
3        9
4        16
5        25
6        36
7        49
8        64
9        81
10       100
```

[37]:
```
#write a program to print the postive and negative  values
print('Accepct postive negative numbers from user')
lst=[int(n) for n in input().split()]
positive = [pos for pos in lst if pos < 0]
negative =[neg for neg in lst if neg > 0]
print('Positive \t Negative')
for p , n in zip(positive, negative):
  print(p,'\t\t',n)
```

```
Accepct postive negative numbers from user
3 -2 4 -4 5 -5 -6 7
Positive         Negative
-2               3
-4               4
-5               5
-6               7
```

[56]:
```
#Write a python program which will seperate the palindrom words from given list␣
 ↪of words
words =['mom' ,'shyam', 'madam', 'dad', 'liril', 'malayalam' ,'rajesh',␣
 ↪'redder', "racecar", "refer", "civic", "deified" ,"level"]
palindrom_words=[word for word in words if word==word[::-1]]
```

```
print(palindrom_words)
```

```
['mom', 'madam', 'dad', 'liril', 'malayalam', 'redder', 'racecar', 'refer',
'civic', 'deified', 'level']
```

[86]:
```
#find the palindrom word lengths and also find the highest length parlindrom␣
 ↪word with length and word
words =['mom' ,'shyam', 'madam', 'dad', 'liril', 'malayalam' ,'rajesh',␣
 ↪'redder', "racecar", "refer", "civic", "deified" ,"level"]
palindrom_words=[word for word in words if word==word[::-1]]

dict_pal=dict([pw,(len(pw))] for pw in palindrom_words)
for p ,l in dict_pal.items():
  print(p , '\t\t\t', l)

# #find max values from dict

max_val=0
max_key = None
for k, v in dict_pal.items():
  if v > max_val:
    max_val = v
    max_key = k

print('MAX length Palindrom word is {} and length is {}'.
 ↪format(max_key,max_val))
```

```
mom                     3
madam                   5
dad                     3
liril                   5
malayalam                       9
redder                  6
racecar                         7
refer                   5
civic                   5
deified                         7
level                   5
MAX length Palindrom word is malayalam and length is 9
```

5.1 Using lambda fucntion

[88]:
```
num_list=[1,2,3,4,5]

sqr_list=[n**2 for n in num_list ]
print(sqr_list)
sqr_lam={num : (lambda x: x**2) (num) for num in num_list}
```

```
print(sqr_lam)
```

```
[1, 4, 9, 16, 25]
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

[90]:
```
'''
using a function to define the key or value expressng you can use a function to␣
 ↪define
the key or value expression in the dict comprehension
'''
def sqr(num):
  return num * num

num1=[1,2,3]
sqr= {num : sqr(num) for num in num1}
print(sqr)
```

```
{1: 1, 2: 4, 3: 9}
```

5.2 dict Comprehension

** syntaxt :- new_dict = {key_expression: value_expression for item in iterable_if_condition }**

[120]:
```
name = ['pams', 'rajesh', 'shweta']
profs = ['Engineer', 'Teacher', 'Doctor']

# #Method 1
my_dict= {}
my_dict1 ={}
for (key, value) in zip(name, profs):
  my_dict[key] = value
print("USING ZIP")
print(my_dict)

#method2
for i in range(len(name)):
  my_dict1[name[i]] = profs[i]
print("USING RANGE")
print(my_dict1)

# dict Comprehension

my_dict_com={key:value for (key, value) in zip(name, profs)}
print("DICT COMPRESNSION")
print(my_dict_com)

# range comprehension
my_dict_com_range={name[i]:profs[i] for i in range(len(name))}
```

```
print("DICT COMPRESNSION RANGE")
print(my_dict_com_range)
```

```
USING ZIP
{'pams': 'Engineer', 'rajesh': 'Teacher', 'shweta': 'Doctor'}
USING RANGE
{'pams': 'Engineer', 'rajesh': 'Teacher', 'shweta': 'Doctor'}
DICT COMPRESNSION
{'pams': 'Engineer', 'rajesh': 'Teacher', 'shweta': 'Doctor'}
DICT COMPRESNSION RANGE
{'pams': 'Engineer', 'rajesh': 'Teacher', 'shweta': 'Doctor'}
```

[124]:
```
import pandas as pd

name =pd.DataFrame( ['pams', 'rajesh', 'shweta'])
profs = pd.DataFrame(['Engineer', 'Teacher', 'Doctor'])
# #Method 1
my_dict_pd= {}
my_dict1_pd ={}
for (key, value) in zip(name[0], profs[0]):
  my_dict_pd[key] = value
print("USING ZIP")
print(my_dict_pd)

#method2
for i in range(len(name)):
  my_dict1_pd[name[0][i]] = profs[0][i]
print("USING RANGE")
print(my_dict1_pd)


my_dict_com_pd={key:value for (key, value) in zip(name[0], profs[0])}
print("DICT COMPRESNSION")
print(my_dict_com_pd)

# range comprehension
my_dict_com_range_pd={name[0][i]:profs[0][i] for i in range(len(name))}
print("DICT COMPRESNSION RANGE")
print(my_dict_com_range_pd)
```

```
USING ZIP
{'pams': 'Engineer', 'rajesh': 'Teacher', 'shweta': 'Doctor'}
USING RANGE
{'pams': 'Engineer', 'rajesh': 'Teacher', 'shweta': 'Doctor'}
DICT COMPRESNSION
{'pams': 'Engineer', 'rajesh': 'Teacher', 'shweta': 'Doctor'}
DICT COMPRESNSION RANGE
{'pams': 'Engineer', 'rajesh': 'Teacher', 'shweta': 'Doctor'}
```

[132]:
```
my_heros={'Spider' : 'photographer', 'Bat':'philanthropist', 'Wonder Wo':␣
 ↪'Nurse'}
my_heros_man={key+'man': value for (key , value) in my_heros.items()}
print(my_heros_man)

# update Sider to super using conditon
# lets change the spidermans profession using condition
my_heros_super={(key+'man' if key != 'Spider' else 'Superman'):(value if value !
 ↪='photographer' else 'journalist') for (key, value) in my_heros.items()}
print(my_heros_super)
```

```
{'Spiderman': 'photographer', 'Batman': 'philanthropist', 'Wonder Woman':
'Nurse'}
{'Superman': 'journalist', 'Batman': 'philanthropist', 'Wonder Woman': 'Nurse'}
```

[91]:
```
squar_dict={x: x**2 for x in range(1,11)}
print(squar_dict)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

[94]:
```
#find the key suppose 36 from the dict

key={k for k,v in squar_dict.items() if v == 36}
print(key)
```

```
{6}
```

6 Set Comprehension

[95]:
```
squar_dict={x**2 for x in range(1,11)}
print(squar_dict)
print(type(squar_dict))
```

```
{64, 1, 4, 36, 100, 9, 16, 49, 81, 25}
<class 'set'>
```

6.0.1 PROGRAM

Manipulting Stings : we can use list comprehnsion to manipulate strings and create a new list with the results

for example to create a new list with the lenghts of the words in an existing list of strings

list_words= [‘mom’ ,‘shyam’, ‘madam’, ‘dad’, ‘liril’, ‘malayalam’ ,‘rajesh’, ‘redder’ ,“level”]

[108]:
```
#words =['mom' ,'shyam', 'madam', 'dad', 'liril', 'malayalam' ,'rajesh',␣
 ↪'redder' ,"level"]
```

```
print('Enter words seperated by comma')
# list comprehension to get inputs from the user
word_list=[word for word in input().split()]
print(word_list)
# find length
word_len=[len(wlen) for wlen in word_list]
print(word_len)
word_dict=dict(zip(word_list, word_len))
print(word_dict)
for i, v in word_dict.items():
  print(i, '\t',v)
```

```
Enter words seperated by comma
mam shyam madam dad liril malayalam rajesh redder level
['mam', 'shyam', 'madam', 'dad', 'liril', 'malayalam', 'rajesh', 'redder',
'level']
[3, 5, 5, 3, 5, 9, 6, 6, 5]
{'mam': 3, 'shyam': 5, 'madam': 5, 'dad': 3, 'liril': 5, 'malayalam': 9,
'rajesh': 6, 'redder': 6, 'level': 5}
mam      3
shyam    5
madam    5
dad      3
liril    5
malayalam        9
rajesh   6
redder   6
level    5
```

[ ]: