|  |  |
| --- | --- |
|  | **Faculty Notebook**  **(Day 4)** |

**Table of Content**[**¶**](#gjdgxs)

1. [**Built-in Functions**](#30j0zll)
2. [**User Defined Functions**](#1fob9te)
   * 2.1 - [**User defined functions without arguments**](#3znysh7)
   * 2.2 - [**User defined functions with arguments**](#2et92p0)
   * 2.3 - [**Recursive Functions**](#tyjcwt)
3. [**Scope of Variables**](#3dy6vkm)
4. [**Lambda Functions**](#1t3h5sf)
5. [**Overview of map, reduce and filter functions**](#4d34og8)
   * 5.1 - [Function map()](#2s8eyo1)
   * 5.2 - [Function filter()](#17dp8vu)
   * 5.3 - [Function reduce()](#3rdcrjn)
   * 5.4 - [Function accumulate()](#26in1rg)

## **1. Built-in Functions**[**¶**](#lnxbz9)

|  |  |
| --- | --- |
|  | **The Python interpreter has functions instantly available for use, called *built-in functions*. For example, the print() function prints the given object.**  **In Python 3.6 there are 69 built-in function.** |

Some exapmles are:

|  |  |
| --- | --- |
| **Function** | **Description** |
| **abs()** | **It gives the absolute of a numeric value** |
| **sum()** | **It gives the sum of a numeric values values** |
| **len()** | **It gives the sum of a numeric values values** |
| **apply()** | **applied both to series and dataframes where function can be applied both series and individual elements based on the type of function provide** |
| **map()** | **only works on a pandas series where type of operation to be applied depends on argument passed as a function, dictionary or a list** |

As we progress with the course, we shall come across a few more built-in functions.

Get the absolute value of a number.

In [1]:

*# get the absolute value of -99.34*  
abs(-99.34)

Out[1]:

99.34

A list consists of elements 44, 45, -56, 44, 32,-54, 44, 56, 44, 87, 44, 56.

We can obtain the number of elements in list and sum of all these elements.

In [2]:

x = [44, 45, -56, 44, 32,-54, 44, 56, 44, 87, 44, 56]

In [3]:

*# len() is used to check length*  
len(x)

Out[3]:

12

In [4]:

*# sum() is used to obtain sum of all the elements in list*  
sum(x)

Out[4]:

386

In [5]:

*# create a string*  
string = "Data Science"

Get the length of the string

In [6]:

*# use len() to obtain the length of the string*  
len(string)

Out[6]:

12

The output is the number of characters in the string including the blank space.

Now let us find the number of words in the string.

To do so, we will first break the string at the blank space using the function split() and then with the help of the function len() we obtain the number of words in the string.

In [7]:

len(string.split())

Out[7]:

2

Now we create a string 'apple, kiwi, pineapple,mango,grapes, banana'. Is it possible to find the number of fruits in the string.

Yes, it is. We shall use the split function again and then function 'len'.

In [8]:

*# create a string*  
fruits = 'apple, kiwi, pineapple,mango,grapes, banana'  
  
print(fruits.split())  
print("The number of fruits in the list are: " , len(fruits.split()))

['apple,', 'kiwi,', 'pineapple,mango,grapes,', 'banana']  
The number of fruits in the list are: 4

It is however evident that there are more than 4 fruits in the list

**Note:** The function split(), breaks the string at a blank space (white space) if no argument is passed. So now we shall specify for it to break at a "," since the string is not spaced properly.

In [9]:

print(fruits.split(","))  
print("The number of fruits in the list are: " , len(fruits.split(",")))

['apple', ' kiwi', ' pineapple', 'mango', 'grapes', ' banana']  
The number of fruits in the list are: 6

This gives the correct output.

**Note:** We can specify any character we wish to split at. For instance, a letter, a number, "," , "'" , "." and so on. If nothing is specified, the function will split at a white space by default.

## **2. User Defined Functions**[**¶**](#35nkun2)

There are situations where in built functions are not availabe in a library. In such cases, we do the computations by using formulas/concepts

|  |  |
| --- | --- |
|  | **At times there are no built-in functions available, thus there arises a need for the user to define a function as desired. The user-defined functions are named blocks of code that are designed to do a specific job(s). When you want to perform a particular task that you’ve defined in a function, you call the name of the function responsible for it. This aids the scenarios where a chunk of code is required repeatedly; just call the function dedicated to handling that task, and the call tells Python to run the code inside the function. You’ll find that using functions makes your programs easier to write, read, test, and fix.** |

**Argument and Parameter**[**¶**](#1ksv4uv)

|  |  |
| --- | --- |
|  | **Parameters and arguments are subtly different. Lets us first define them.**  **1. Parameter: A value which is passed when defining a function**  **2. Argument: A value that is passed when invoking a function** |

Consider the following user defined function:

def fun(params): statement 1 statement 2 . . .

Now, if we invoke this function:

fun(sample\_parameters)

Here the 'sample\_parameters' is called argument.

## **2.1 User defined functions without arguments**[**¶**](#44sinio)

Once a function is defined, it can be called anytime, without running the function again. Generally, a function returns a value or prints a value.

Let us see a few examples of user defined functions.

**1. We shall write a user defined function without argument to find the twice of a number.**

In [10]:

**def** twice():  
 a = int(input('enter a number: '))  
 print(a\*2)

In [11]:

*# In this function, no argument is passed*  
*# call the function*  
twice()

enter a number: 34  
68

**2. We shall write a user defined function hi() which take a string input. The output will greet the user displaying his/her name on the screen.**

In [12]:

**def** hi():  
 name = str(input('Enter your name: '))  
 print("Hey " + name +", how are you?")

In [13]:

*# In this function, no argument is passed*  
*# call the function*  
hi()

Enter your name: python  
Hey python, how are you?

In [14]:

*#create a function*  
**def** greetings():  
 print("hello")  
 print("How r u ?")

In [15]:

*# In this function, no argument is passed*  
*# call the function*  
greetings()

hello  
How r u ?

**3. To find area of a circle**

In [16]:

**def** area\_circle():  
 r = int(input('Enter the radius of circle: '))  
 area = 3.14 \* r\*\*2   
 print("The area of cicle with radius " + str(r) +" is " + str(area) + " sq. units")

In [17]:

*# call the function*   
area\_circle()

Enter the radius of circle: 3  
The area of cicle with radius 3 is 28.26 sq. units

## **2.2 User defined functions with arguments**[**¶**](#2jxsxqh)

Most of the function will either return a value or print an output. Take a note of this in the examples considered below.

**1. A user defined function to get the average of all numbers in a sequence.**

In [18]:

**def** average(x):  
 count = 0  
 total = 0  
 **for** i **in** x:  
 count += 1  
 total += i  
 avg = total/count  
 **return** avg

In [19]:

x = (44, 45, -56, 44, 32,-54, 44, 56, 44, 87, 44, 56)

In [20]:

*# call the function*   
average(x)

Out[20]:

32.166666666666664

The average of the number is 32.1667

If the function requires an argument to be passed and we do not provide it, then an error with be shown.

In [21]:

*# call the function*   
average()

**---------------------------------------------------------------------------**  
**TypeError** Traceback (most recent call last)  
**<ipython-input-21-817fa9574eec>** in <module>**()**  
 1 **# call the function**  
**----> 2** average**()**  
  
**TypeError**: average() missing 1 required positional argument: 'x'

**2. A user defined function with argument to greet a person by name.**

In [22]:

**def** greet(name):  
 name = str(name)  
 print("Hello, " + name)

In [23]:

*# call the function*   
greet('Dave')

Hello, Dave

Enter your name in the paranthese, and your notebook will greet you!.

**Also note that this function prints an output.**

|  |  |
| --- | --- |
|  | **TRY IT YOURSELF:**  **Assign a print() function to a variable. Is it possible?** |

**Answer:** The 'print' output can not be stored in a variable. Moreover, any function which returns a value can be stored in a variable.

**4. Consider the list given below and write a function to find the unique values of the list**

[44,45,56,44,32,54,44,56,44,87,44,56]

In [24]:

**def** unique(list1):   
 unique\_list = []   
 **for** x **in** list1:   
 **if** x **not** **in** unique\_list:   
 unique\_list.append(x)   
 **return** unique\_list

In [25]:

x = [44,45,56,44,32,54,44,56,44,87,44,56]  
  
*# call the function*   
*# store the output in a variable*  
unique\_values = unique(x)  
  
*# print the variable name*  
unique\_values

Out[25]:

[44, 45, 56, 32, 54, 87]

These are the unique values in the list.

**Also note that this function returns the output to the calling program.**

**5. A function to reverse a string**

In [26]:

**def** reverse(string):  
 *# set the empty string*  
 rev=''  
 **for** i **in** string:  
 rev = i+rev  
 print(rev)

In [28]:

*# call the function*  
string = str(input('Enter string:'))  
reverse(string)

Enter string:python for data science  
ecneics atad rof nohtyp

**6. A function to display multiplication table of an integer**

In [29]:

**def** table(n):  
 multi = 1  
 **for** i **in** range(1,11):  
 multi = n\*i  
 print(str(i) + " times " + str(n) + " is " + str(multi))

In [31]:

*# call the function*   
n=int(input('Enter number:'))  
table(n)

Enter number:23  
1 times 23 is 23  
2 times 23 is 46  
3 times 23 is 69  
4 times 23 is 92  
5 times 23 is 115  
6 times 23 is 138  
7 times 23 is 161  
8 times 23 is 184  
9 times 23 is 207  
10 times 23 is 230

**7. To find product of numbers in a list.**

In [32]:

**def** product(x):  
 prod = 1  
 **for** i **in** x:  
 prod \*= i  
 **return** prod

In [33]:

*#call the function*  
product([1,2,3,4,5])

Out[33]:

120

**8. A function which takes the order of a matrix and mutliplies it with a scalar. The function take two parameters- first the order of identity matrix and second the scaler to be multiplied.**

In [34]:

**def** scalar\_multiplication(n,m):  
 **for** i **in** range(0,n):  
 **for** j **in** range(0,n):  
 **if**(i==j):  
 print(m,sep=" ",end=" ")  
 **else**:  
 print("0",sep=" ",end=" ")  
 print()

In [35]:

*#call the function*  
scalar\_multiplication(4,5)

5 0 0 0   
0 5 0 0   
0 0 5 0   
0 0 0 5

**9. Usage of \*args.**

Usage : multiple arguments can be added without specifying the number of parameter defined in a function. It is used to pass a non-keyworded, variable-length argument list.

\*args in function definitions in python is used to pass a variable number of arguments to a function. It reads the value one by one and performs the required task.

The symbol \* is used to take in a variable number of arguments; by convention, it is often used with the word 'args', however we may use another word.

In [36]:

*# \* indicates that number of arguments can be taken*  
**def** Multiple(\*args):   
 **for** parameter **in** args:   
 print (parameter)

In [37]:

*#call the function*  
Multiple("Lets","learn","Python","for","Data","Science")

Lets  
learn  
Python  
for  
Data  
Science

**10. Another example on \*args.**

In [38]:

**def** to\_print( argument1, \*argument):  
 print(argument1)  
 **for** var **in** argument:  
 print(var)

In [39]:

*#call the function*   
to\_print( 10,20,30 )

10  
20  
30

**11. Usage of `** kwargs`.\*\*

Usage: \*\*kwargs is used to pass a keyworded, variable-length argument list to the function. The name 'kwargs' is used with the double star because it allows us to pass through keyword arguments (and any number of them).

In [40]:

**def** function(\*\*kwargs):  
 **for** a,b **in** kwargs.items():  
 print(a,"==",b)

In [41]:

*# call the function*  
a=function(A = "First letter", B = "Second letter", C = "Thirs letter")

A == First letter  
B == Second letter  
C == Thirs letter

**12. Another example of `** kwargs `.\*\*

\*\*kwargs is a dict of the keyword or arguments passed to the function

In [42]:

**def** print\_keyword\_args(\*\*kwargs):  
 **for** key, value **in** kwargs.items():  
 print( "**%s** = **%s**" % (key, value))

In [43]:

*# call the function*  
print\_keyword\_args(first\_name="John", last\_name="Doe")

first\_name = John  
last\_name = Doe

## **2.3 Recursive Functions**[**¶**](#z337ya)

A function that calls itself is a recursive function. Recursive function is used when a certain problem is defined in terms of itself.

In [44]:

*# calculate the sum of all numbers present in the list*  
**def** sum\_list(nums):  
 **if** len(nums) == 0:  
 **return** 0  
  
 last\_num = nums.pop()  
 **return** last\_num + sum\_list(nums)  
  
num = [1,2,3,4]  
sum\_list(num)

Out[44]:

10

In [45]:

*# implementation of the fibonacci series using recursion*  
**def** fib(n):  
 **if** n == 0:  
 **return** 0  
 **elif** n == 1:  
 **return** 1  
 **else**:  
 **return** fib(n - 1) + fib(n - 2)  
   
   
**for** i **in** range(15):  
 print(fib(i), end=" ")

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

<a id = 'scope'></a>

## **3. Scope of Variables:**[**¶**](#3j2qqm3)

|  |  |
| --- | --- |
|  | **The scope of variables is broadly classified as:**  **1. Global Variable: Global variables are the variables declared outside of the function or in global scope. This means, global variable can be accessed inside or outside of the function.**  **2. Local Variable : Local variables are the varibales declared inside the function's body or in the local scope** |

**Following are examples of Global variables:**

The variable 'string' is a global variable.

In [46]:

*# This function uses global variable 'string'*  
**def** function():   
 print (string)   
   
  
string = "Python for Data Science"  
function()

Python for Data Science

Another example where x is a global variable

In [47]:

**def** f():  
 y = x  
 print(y)

In [48]:

x = 8  
f()

8

Consider another scenario where x is declared in the function after y is declared.

In [49]:

**def** f():  
 y = x  
 print(y)  
 x = 22

In [50]:

f()

**---------------------------------------------------------------------------**  
**UnboundLocalError** Traceback (most recent call last)  
**<ipython-input-50-0ec059b9bfe1>** in <module>**()**  
**----> 1** f**()**  
  
**<ipython-input-49-0efb2420edee>** in f**()**  
 1 **def** f**():**  
**----> 2** y **=** x  
 3 print**(**y**)**  
 4 x **=** **22**  
  
**UnboundLocalError**: local variable 'x' referenced before assignment

The output will be an error since x is declared after y and y is x.

**2. Local Variable**[**¶**](#1y810tw)

Example of local variables.

In [51]:

**def** sum(x,y):  
 sum = x + y  
 **return** sum  
  
print(sum(4,7) )

11

Here, x and y are local variables. Their value is not callable outside the definition.

Let us print x and check it value.

In [52]:

x

Out[52]:

8

**Here x takes its global value and not the one defined in the function.**

Another example of local variable:

In [53]:

*# set a local variable*  
**def** local():  
 y = "This is a Local variable"  
 print(y)  
  
local()

This is a Local variable

Now if we print just y

In [54]:

*# local variable y cannot be accessed outside the function*  
print(y)

**---------------------------------------------------------------------------**  
**NameError** Traceback (most recent call last)  
**<ipython-input-54-5b64542eefae>** in <module>**()**  
 1 **# local variable y cannot be accessed outside the function**  
**----> 2** print**(**y**)**  
  
**NameError**: name 'y' is not defined

We are bound to get an error since it. The variable 'y' is a local variable.

<a id = 'lambda' > </a>

## **4. Lambda Functions**[**¶**](#4i7ojhp)

|  |  |
| --- | --- |
|  | **A lambda operator or lambda function is used for creating small, one-time, anonymous function objects in Python.**  **The general expression for a lambda function is**  **lambda argument(s): expression**  **A lambda function can take any number of arguments, but they contain only a single expression. An expression is a piece of code executed by the lambda function, which may or may not return any value.** |

Let us look at some examples using on lambda functions.

**1. Obtain the minimum among two numbers**

In [55]:

min\_value = (**lambda** x, y: x **if** x < y **else** y)  
min\_value(34, 78)

Out[55]:

34

**2. Check whether two strings are equal**

In [56]:

result = (**lambda** x, y: 'True' **if** x == y **else** 'False')  
result("Data", "DaTa")

Out[56]:

'False'

**3. Demonstrating wrong usuage of variable name**

In [57]:

*# multiplication - wrong usage of variable*  
x = **lambda** a, c : a + b  
print(x(5, 6))

**---------------------------------------------------------------------------**  
**NameError** Traceback (most recent call last)  
**<ipython-input-57-f2865e686470>** in <module>**()**  
 1 **# multiplication - wrong usage of variable**  
 2 x **=** **lambda** a**,** c **:** a **+** b  
**----> 3** print**(**x**(5,** **6))**  
  
**<ipython-input-57-f2865e686470>** in <lambda>**(a, c)**  
 1 **# multiplication - wrong usage of variable**  
**----> 2** x **=** **lambda** a**,** c **:** a **+** b  
 3 print**(**x**(5,** **6))**  
  
**NameError**: name 'b' is not defined

**If variable a and c are used, then the operations should be on a and c.**

## **5. Overview of map, reduce and filter functions**[**¶**](#2xcytpi)

|  |  |
| --- | --- |
|  | **The function map(f,l) applies a function 'f' to each element of list 'l'**  **The output of map is not a list, to convert it to a list use list(map(f,l))**  **The function filter(p,l) where property p is checked for every element in list l. The output of fliter(p, l) is a sublist of list 'l' that satistifies the property 'p'.**  **The reduce(fun,seq) function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along and returns only one value.This function is defined in “functools” module.**  **The accumulate() in “itertools” module. The syntax for it is accumulate(l,f). It works similar to the reduce(), however there is a little difference in the output.** |

### **5.1 Function map()**[**¶**](#1ci93xb)

**1. A test was conducted in a class, the list of all the scores is given. Tha scores are out of 45. Obtain the percentage of each score using lambda function.**

marks = [ 20, 23, 34, 45, 33, 10, 43, 23, 27, 34, 37, 44, 42, 43 ]

In [58]:

marks = [ 20, 23, 34, 45, 33, 10, 43, 23, 27, 34, 37, 44, 42, 43 ]  
  
print(list(map(**lambda** x, y=100 :x\*y/45 , marks)))

[44.44444444444444, 51.111111111111114, 75.55555555555556, 100.0, 73.33333333333333, 22.22222222222222, 95.55555555555556, 51.111111111111114, 60.0, 75.55555555555556, 82.22222222222223, 97.77777777777777, 93.33333333333333, 95.55555555555556]

The output is the percentage scores.

**2. Create two lists and add corresponding elements of each list**

In [59]:

*# iteration using list*  
list\_a = [30, 35, 40]  
list\_b = [12, 5, 70]  
   
print(list(map(**lambda** x, y: x + y, list\_a, list\_b)))

[42, 40, 110]

### **5.2 Function filter()**[**¶**](#3whwml4)

**1. To obtain the sum of cubes of odd number from 1 to 50**

In [1]:

*# sum of cubes of odd numers from 1 to 50*  
  
*# user defined function to obtain the cube*  
**def** cube(x):  
 **return**(x\*x)  
  
*# user defined function to obtain odds numbers from a list*  
**def** isodd(x):  
 **return**(x%**2** == 1)  
  
*# create a list of all cubes*  
list\_cubes = list(map(cube,filter(isodd,range(1,50))))  
  
*# sum of all elements in the in the list\_cube*  
sum(list\_cubes)

Out[1]:

20825

The same code above can be done using list comprehension in the follwing way:

In [2]:

*# map - the function defined above is called*   
*# generator - the range for numbers is fixed*  
*# filter - cube is calculated for the numbers which satisfy the 'if' condition*  
*# built-in function sum() is used to obtain the sum*  
  
sum ( [cube(x) **for** x **in** range(50) **if** isodd(x)] ) *# Map - Generator - Filter*

Out[2]:

20825

**2. To obtain the odd numbers from a list**

In [3]:

numbers = [121, 443, 477, 125, 34, 655, 458, 141, 363, 125, 456 , 578, 123]  
  
*# function that filters the numbers*  
**def** oddnumber(in\_num):  
 **if**(in\_num % 2) == 0:  
 **return** **True**  
 **else**:  
 **return** **False**  
  
*# Use filter() to remove odd numbers*  
out\_filter\_number = filter(oddnumber, numbers)  
  
print("Type of filter object: ", type(out\_filter\_number))  
print("Even number is as follows: ", list(out\_filter\_number))

Type of filter object: <class 'filter'>  
Even number is as follows: [34, 458, 456, 578]

### **4.3 Function reduce()**[**¶**](#2bn6wsx)

**1. List of sales in a shop are noted. Obtain the total sales**

In [4]:

*# import reduce function from functools*  
**from** **functools** **import** reduce  
  
sale = [14565,65460,74450,54546,23245,14543,84355,54566]  
total\_sale = reduce(**lambda** a, b: a + b, sale)  
print(total\_sale)

385730

**Thus we may say the total sale is 385730 INR.**

**2. Factorial of a 5**

In [5]:

*# import reduce function from functools*  
**from** **functools** **import** reduce  
  
factorial = reduce(**lambda** p,q : p\*q , range(1,7))  
  
print("The factorial of 6 is ",factorial)

The factorial of 6 is 720

### **4.3 Function accumulate()**[**¶**](#qsh70q)

**1. Sum of first ten whole numbers**

In [6]:

*# importing itertools for accumulate()*   
**import** **itertools**   
  
*# priting summation using accumulate()*   
print ("The summation of list using accumulate is :",end="")   
print (list(itertools.accumulate(range(10),**lambda** x,y : x+y)))

The summation of list using accumulate is :[0, 1, 3, 6, 10, 15, 21, 28, 36, 45]

The output of reduce() will be:

In [7]:

*# import reduce function from functools*  
**from** **functools** **import** reduce  
  
sum = reduce(**lambda** a,b : a+b , range(10))  
print(sum)

45

The function accumulate() displays the iterations whereas the function reduce() displays the final output.