Contents

[LISTS 1](#_Toc38548721)

[IMP ABOUT LISTS 4](#_Toc38548722)

[Examples 4](#_Toc38548723)

[Index greater than list length 5](#_Toc38548724)

[insert() vs append() 5](#_Toc38548725)

[insert() vs extend() 5](#_Toc38548726)

[**Formatting** 8](#_Toc38548727)

[Print Formatting 8](#_Toc38548728)

[**🡪To change working directory path from Jupiter use below command** 9](#_Toc38548729)

[**PYTHON map() function** 10](#_Toc38548730)

[**Python map() function** 10](#_Toc38548731)

[**Tuples addition element wise:** 13](#_Toc38548732)

[**reduce() in Python** 14](#_Toc38548733)

[**filter() in python** 16](#_Toc38548734)

[**FUNCTION ARGUMENTS** 18](#_Toc38548735)

[\*args and \*\*kwargs in Python 22](#_Toc38548736)

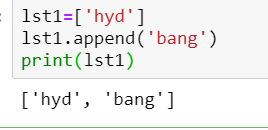
[thumb rule for list comprehension 25](#_Toc38548737)

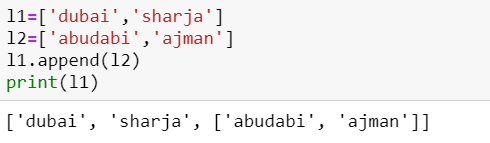
## LISTS

**Append Method**

The append() method in python adds a single item to the existing list. It doesn’t return a new list of items but will modify the original list by adding the item to the end of the list.

After executing the method append on the list the size of the list increases by one.

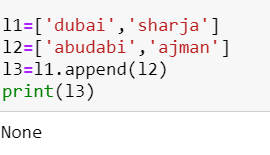


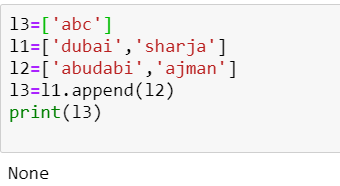


Length l1 is 3

Note : In above example l2 has 2 elements but as extend will extend only one element ,it will add entire l2 list as single element

Note: we can’t assign append output to new list



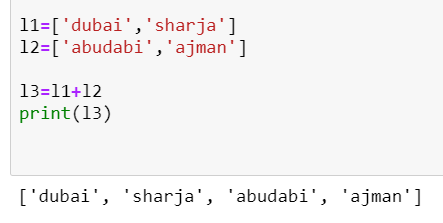


In above example we tried to assign output to l3 but ,it will give none

**Append using+ operator**

**+ operator will create new list ,it wont change current list.**

We can use + operator to append 2 lists



If we use + then it will unzip or raval all elements of l2 and append to l1 and gives new list. That new list we can assign to new variable. Variable l1, l2 are same after using + symbol.

So length of l1+l2 is 4

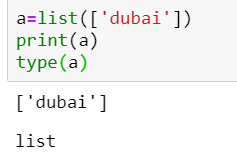
Note: if we use append then entire list added as single element but if we use + operator then it will raval all elements then add to first element.

|  |  |
| --- | --- |
| **append** | **operator '+'** |
| **The append() method in python adds a single item to the existing list** | **If we use + then it will unzip or raval all elements of l2 and append to l1 and gives new list** |
| **.It doesn’t return a new list but will modify the original list by adding the item to the end of the list.** | **it returns new list. That new list we can assign to new variable. Variable l1, l2 are same after using + symbol.** |

NOTE: if we use list to create it will split the data. If we use number in list then it will give error bcz number is not iterable.

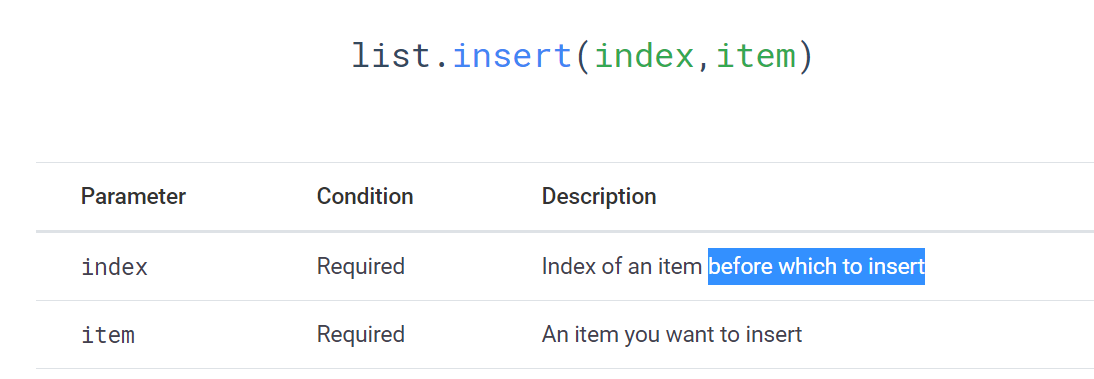


to use list to create list then we have to use []



## IMP ABOUT LISTS

<https://www.learnbyexample.org/python-list-insert-method/>



Examples

Example: Insert ‘yellow’ at 2nd position

L = ['red', 'green', 'blue']

L.insert(1,'yellow')

print(L) # ['red', 'yellow', 'green', 'blue']

You can also use [negative indexing](https://www.learnbyexample.org/python-list/#negative-list-indexing) with insert() method.

Example: Insert ‘yellow’ at 2nd position with Negative indexing

L = ['red', 'green', 'blue']

L.insert(-2,'yellow')

print(L) # ['red', 'yellow', 'green', 'blue']

in above -2 means green . so it will insert yellow before green element

Index greater than list length

When you specify an *index* greater than list length, you do not get any exception. Instead, the item is inserted at the end of the list.

Example: Index greater than list length inserts item at the end

L = ['red', 'green', 'blue']

L.insert(10,'yellow')

print(L) # ['red', 'green', 'blue', 'yellow']

## insert() vs append()

Inserting item at the end of the list with insert() method is equivalent to append() method.

Example: Inserting item at the end with insert()

L = ['red', 'green', 'blue']

L.insert(len(L),'yellow')

print(L) # ['red', 'green', 'blue', 'yellow']

is equivalent to

L.append('yellow')

## insert() vs extend()

insert() method treats its argument as a single object.

Example:

L = ['red', 'green']

L.insert(2,'blue')

print(L) # ['red', 'green', 'blue']

Use [extend()](https://www.learnbyexample.org/python-list-extend-method/) method, if you want to add every item of an iterable to a list.

Example:

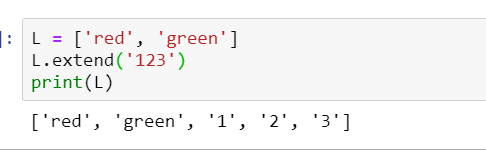
L = ['red', 'green']

L.extend('blue')

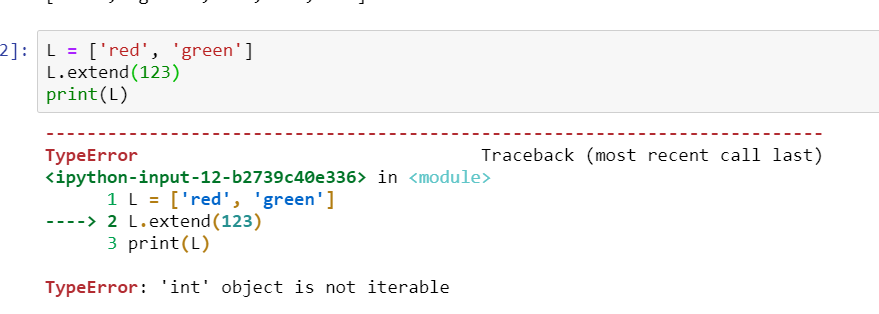
print(L) # ['red', 'green', 'b', 'l', 'u', 'e']

ex:1

below 123 is treated as string and iterable

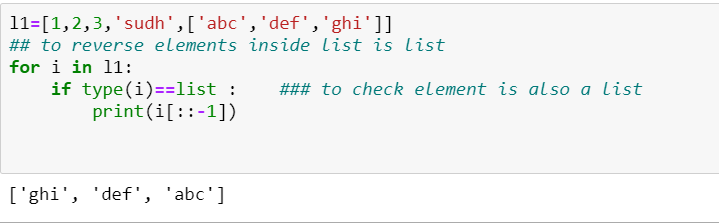


In below examples 123 is int data type and it is not iterable

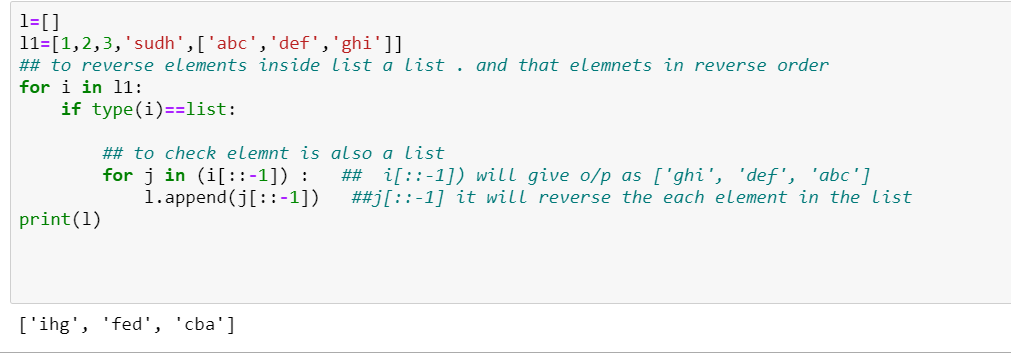


--🡪

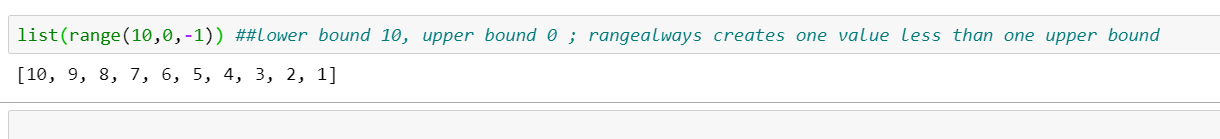
To reverse a list which is present in a list



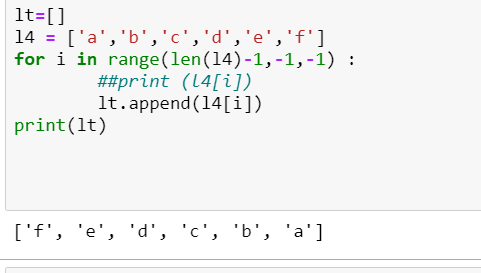
🡪:



-🡪



Reversing the list using range :



# **Formatting**

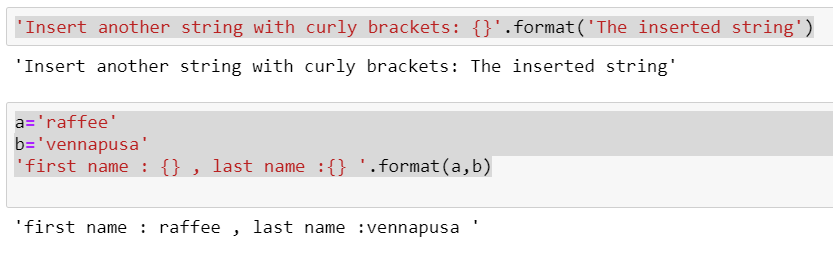
## Print Formatting

Print Formatting ".format()" method is used to add formatted objects to the printed string statements.

Let's see an example to clearly understand the concept.

Note: no of {} and the no of values in format should be same

if no of values in format is greter than no of {} then we wont get any error but if no of values in format is less than no of {} then python throw error





# **None data type**

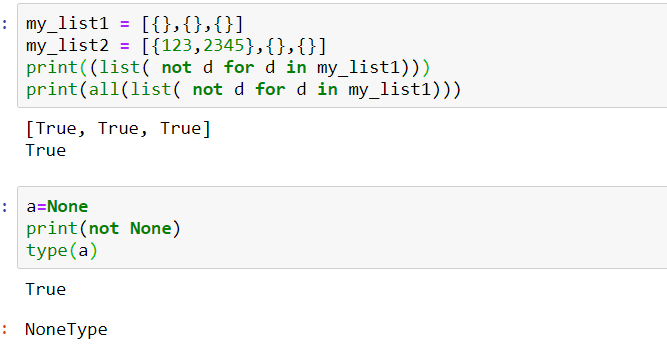
not None will give as True

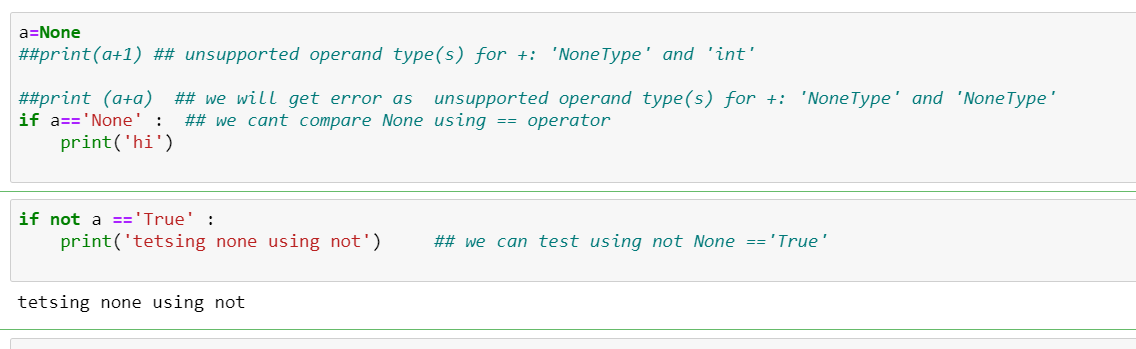
we can’t do arthemetic operation on None data tye

we cant tets None using == operator

we can test None using not None =='True'

print(not None) 🡪 gives True





🡪max function

Syntax:

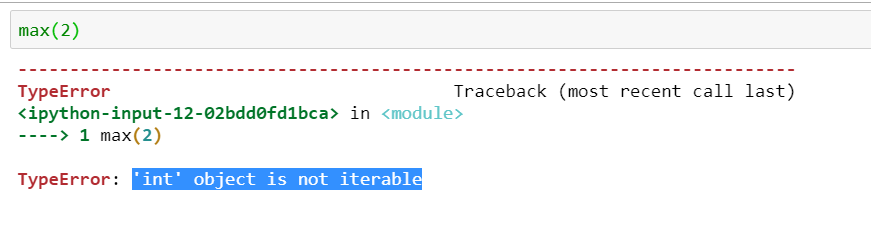
max(iterable[, default=obj, key=func]) -> value

or

max(a,b,c, ...[, key=func]) -> value

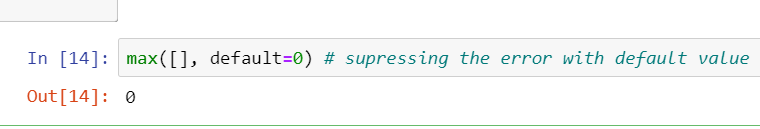
we cant use max(2) with single value.. it should have multiple values .if it ahs single value then it will give error as

'int' object is not iterable



max([]) *# empty iterable causes ValueError*

max([], default=0) *# supressing the error with default value*



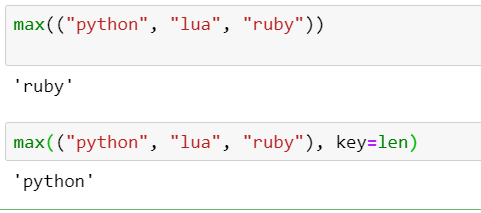
Trying to find largest value among the objects of different types causes an error.

max(10, "pypi") ## will give error as unorderable types: str() > int()

max(5, [-10, 55]) ## will give error as unorderable types: list() > int()

>>>

The following is another example where we compare strings based on its length instead of their ASCII values.



#Write a Python program to find the list in a list of lists whose sum of elements is th

e highest

#solution

num = [[1,2,3], [4,5,6], [10,11,12], [7,8,9]]

print(max(num, key=sum))

o/p is: [10,11,12]

# **🡪To change working directory path from Jupiter use below command**

cd C:\Users\s777593\Desktop\PYTHON\RAFFEEMDAI

* To find os path

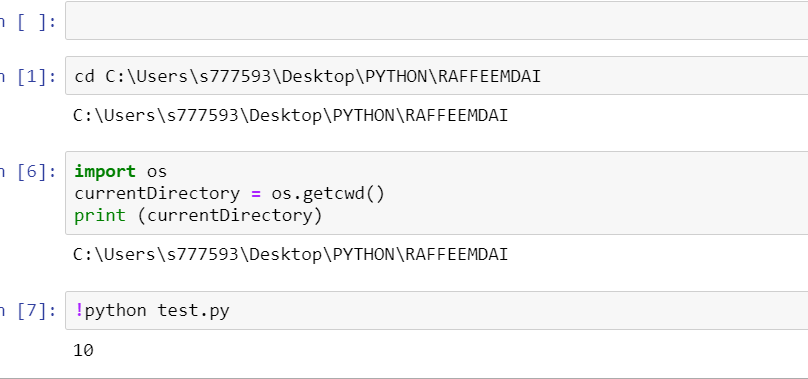
import os

currentDirectory = os.getcwd()

print (currentDirectory)

* To run .py scripts from Jupiter use below command

!python test.py



# **PYTHON map() function**

Ref:

<https://www.geeksforgeeks.org/python-map-function/>

<https://www.geeksforgeeks.org/python-map-vs-list-comprehension/>

<https://www.journaldev.com/22960/python-map-function>

# **Python 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.)

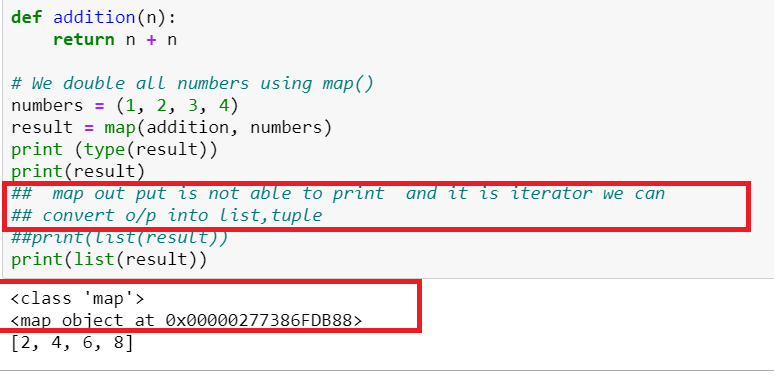
**Syntax :**

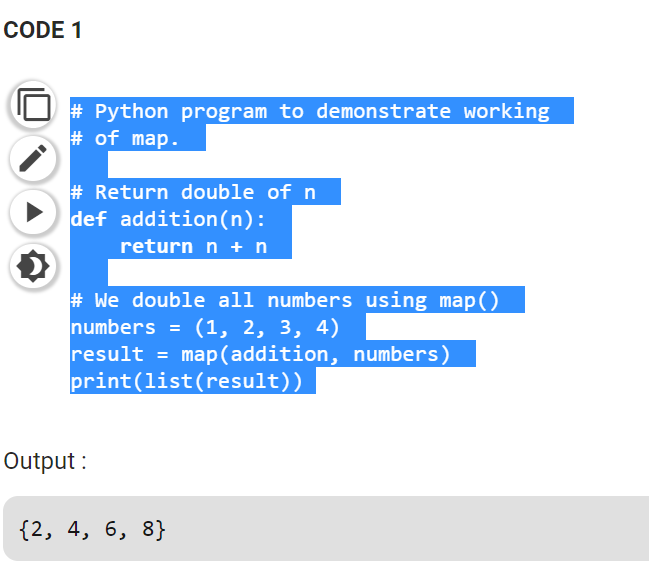
map(fun, iter)

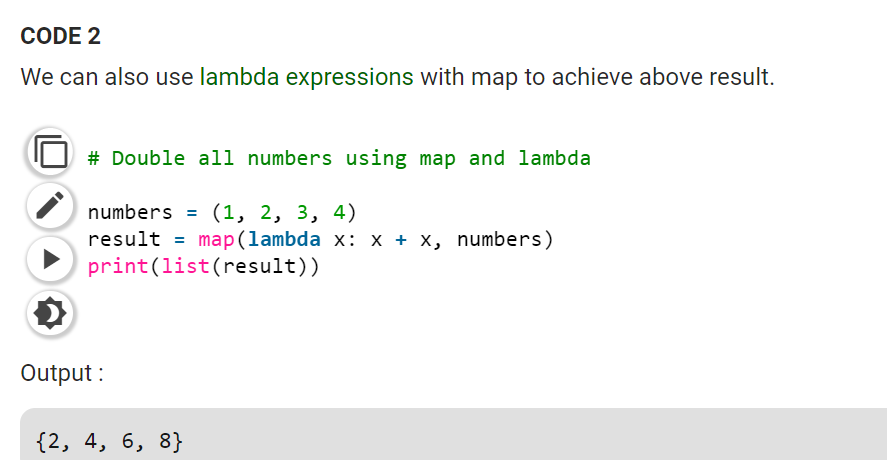
**Parameters :**

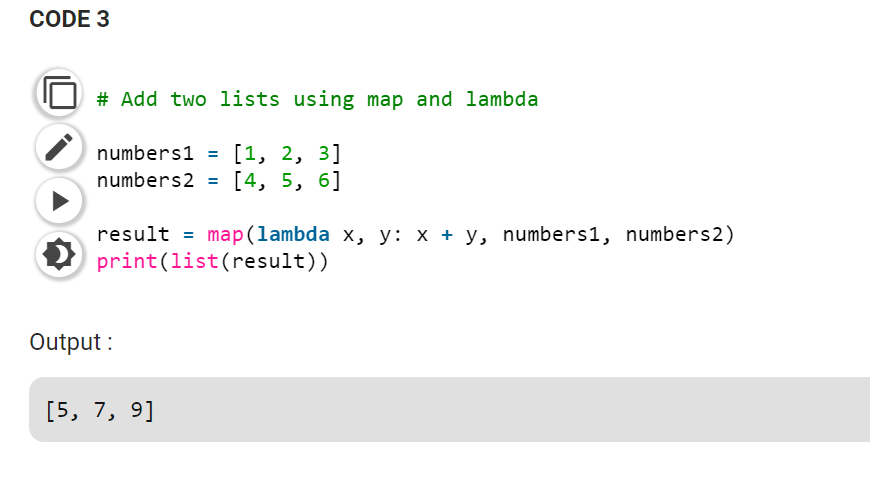


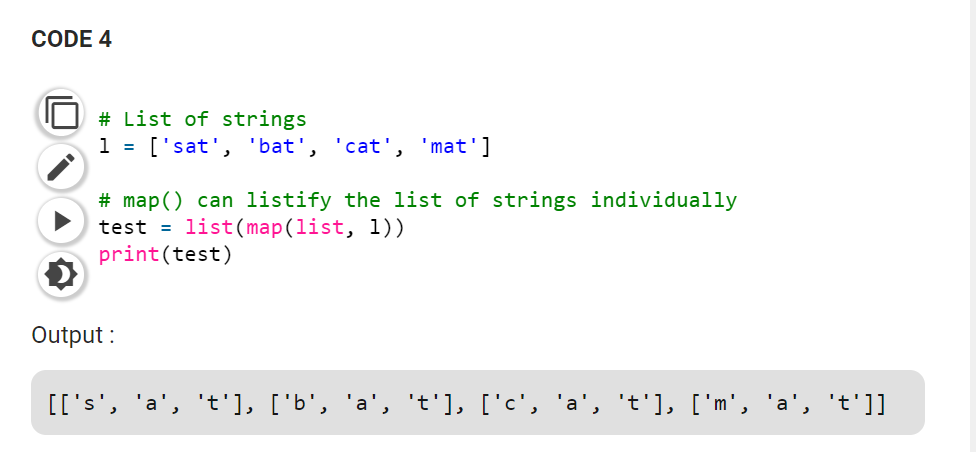
Note: as map function return map object so we cant print values directly we have to convert that using list,tuple .refer below example:







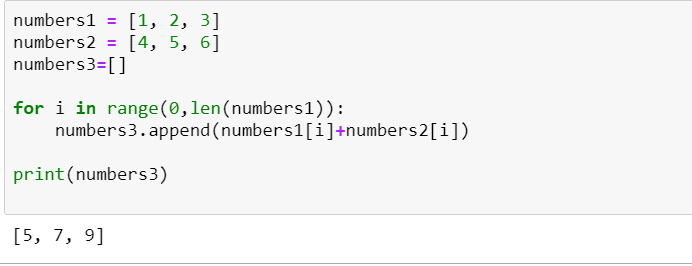


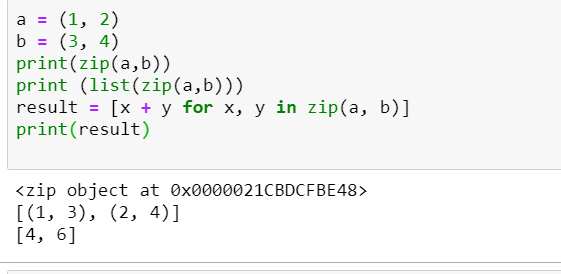


# **Tuples addition element wise:**

Refer:

<https://www.geeksforgeeks.org/python-adding-two-list-elements/>





# **reduce() in Python**

<https://www.geeksforgeeks.org/reduce-in-python/>

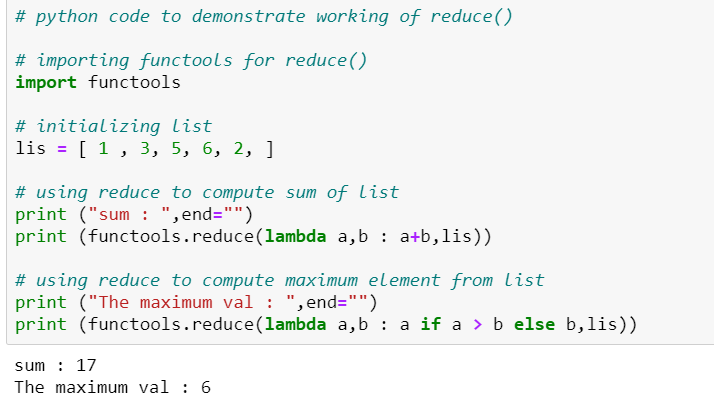
**The reduce(fun,seq) function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along.This function is defined in “functools” module.**

**Working :**

* **At first step, first two elements of sequence are picked and the result is obtained.**
* **Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.**
* **This process continues till no more elements are left in the container.**
* **The final returned result is returned and printed on console.**

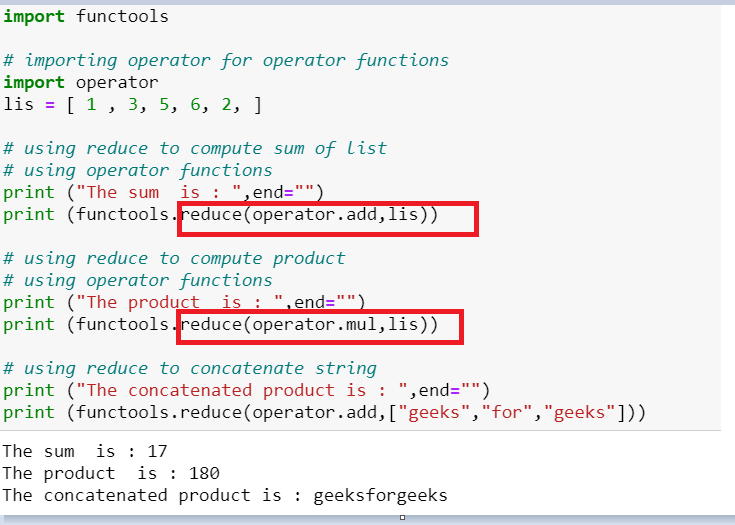
Note: in python 3 to use reduce() function we have to import functools

import functools



**Reduce with operator function**

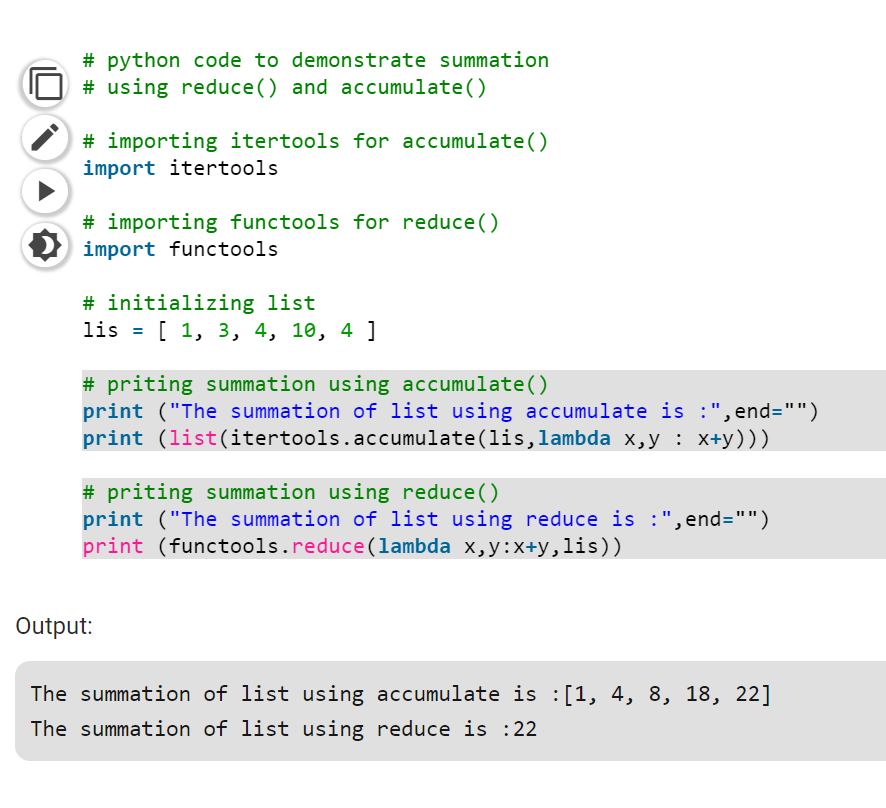
reduce() can also be combined with [operator functions](https://www.geeksforgeeks.org/operator-functions-in-python-set-1/) to achieve the similar functionality as with lambda functions and makes the code more readable.



**reduce() vs accumulate()**

Both reduce () and accumulate() can be used to calculate the summation of a sequence elements. But there are differences in the implementation aspects in both of these.

* reduce() is defined in “functools” module, accumulate() in “itertools” module.
* reduce() stores the intermediate result and only returns the final summation value. Whereas, accumulate() returns a list containing the intermediate results. The last number of the list returned is summation value of the list.
* reduce(fun,seq) takes function as 1st and sequence as 2nd argument. In contrast accumulate(seq,fun) takes sequence as 1st argument and function as 2nd argument.



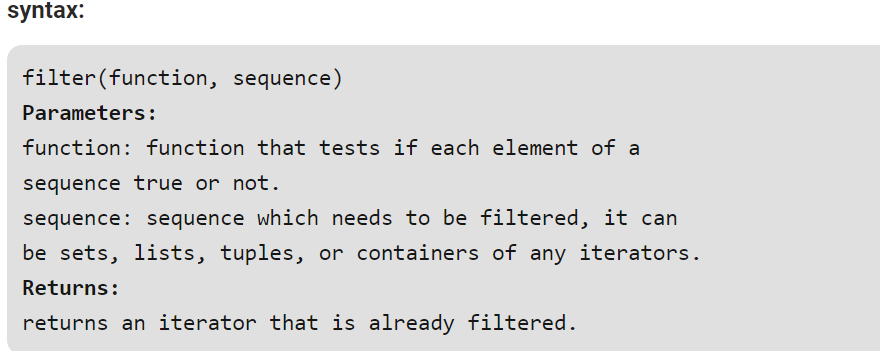
# **filter() in python**

refer:

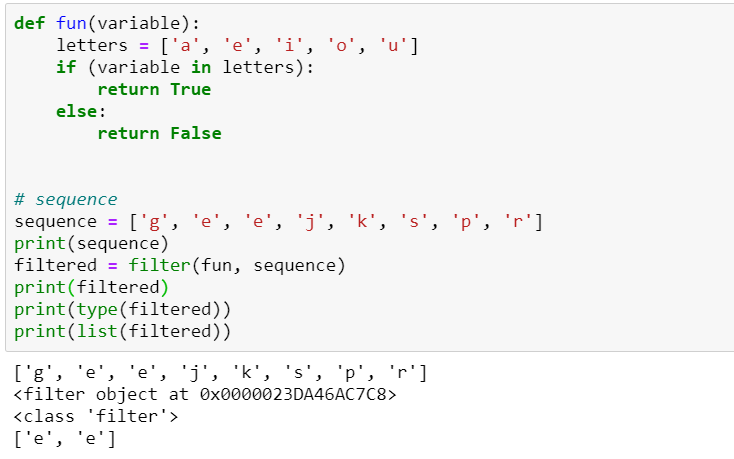
<https://www.geeksforgeeks.org/filter-in-python/>

<https://www.geeksforgeeks.org/lambda-filter-python-examples/>

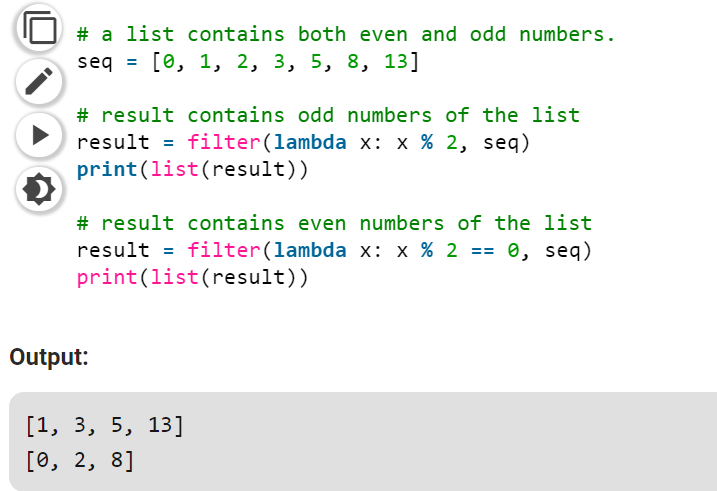
The filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.



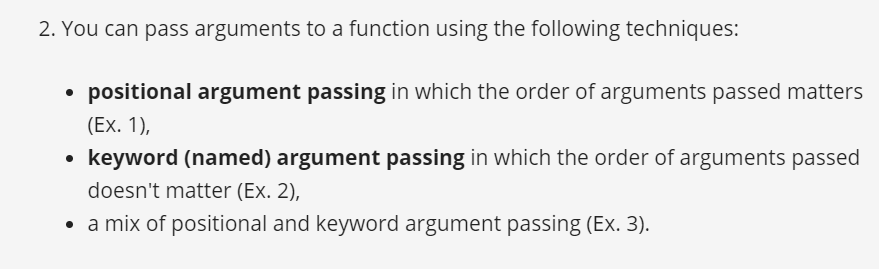
To print the output of a filter we have to convert that output into list or tuples

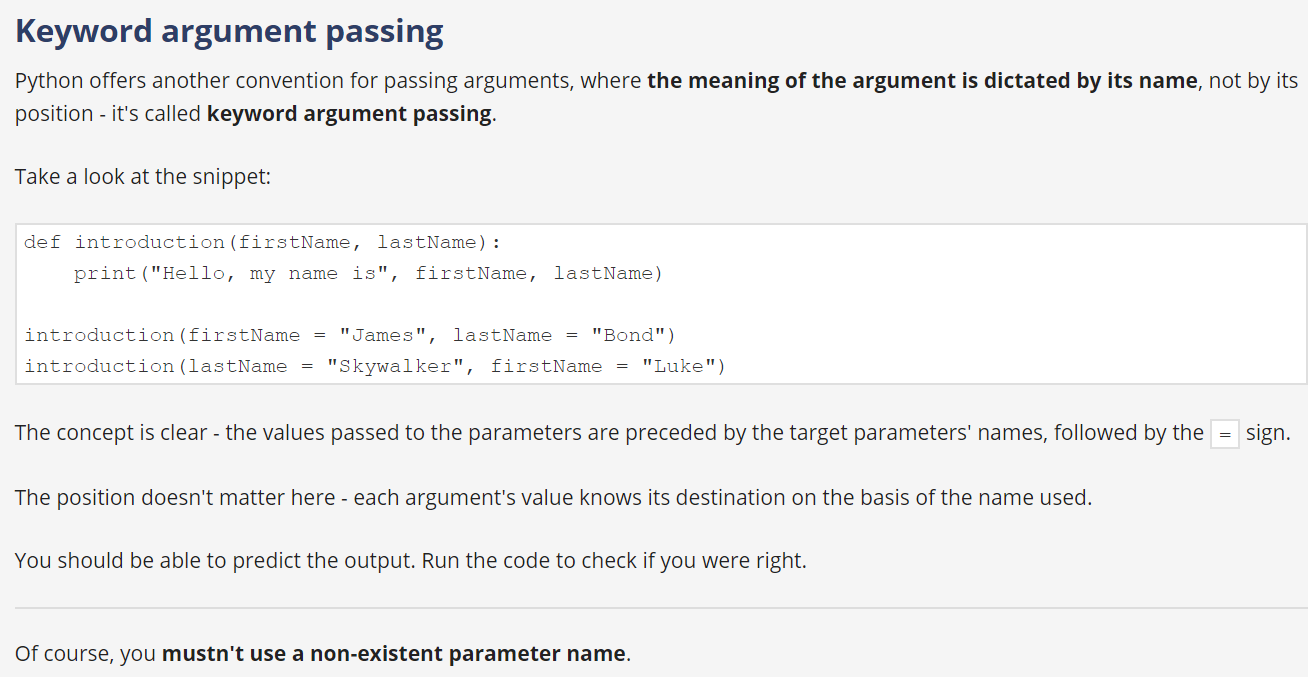


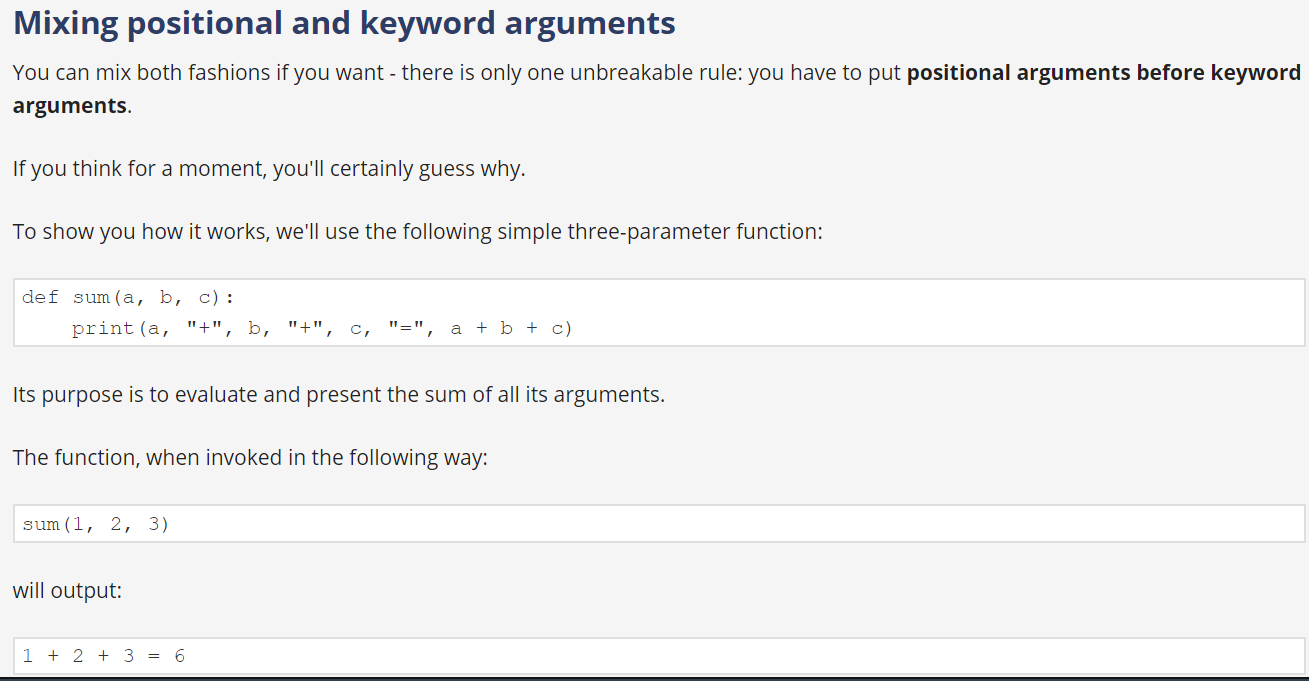
**Application:**  
It is normally used with [Lambda functions](https://www.geeksforgeeks.org/python-lambda-anonymous-functions-filter-map-reduce/) to separate list, tuple, or sets.

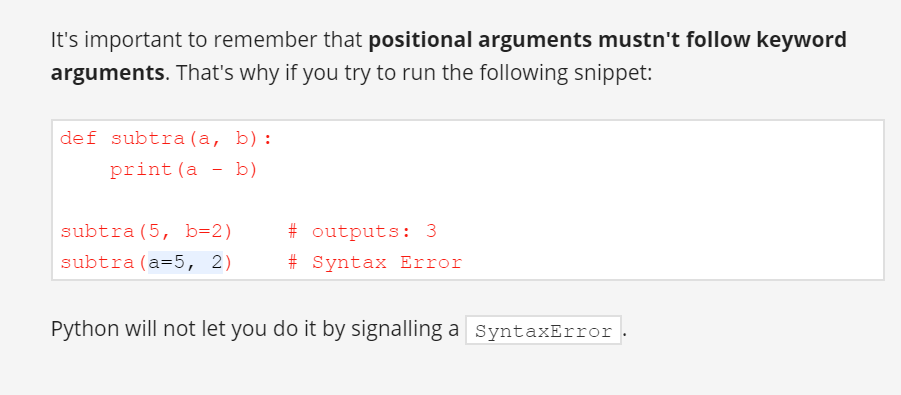


# **FUNCTION ARGUMENTS**









ACCORDING TO APLIED AI Mahesh python has below Forms of Arguments

Refer: <https://www.geeksforgeeks.org/args-kwargs-python/>

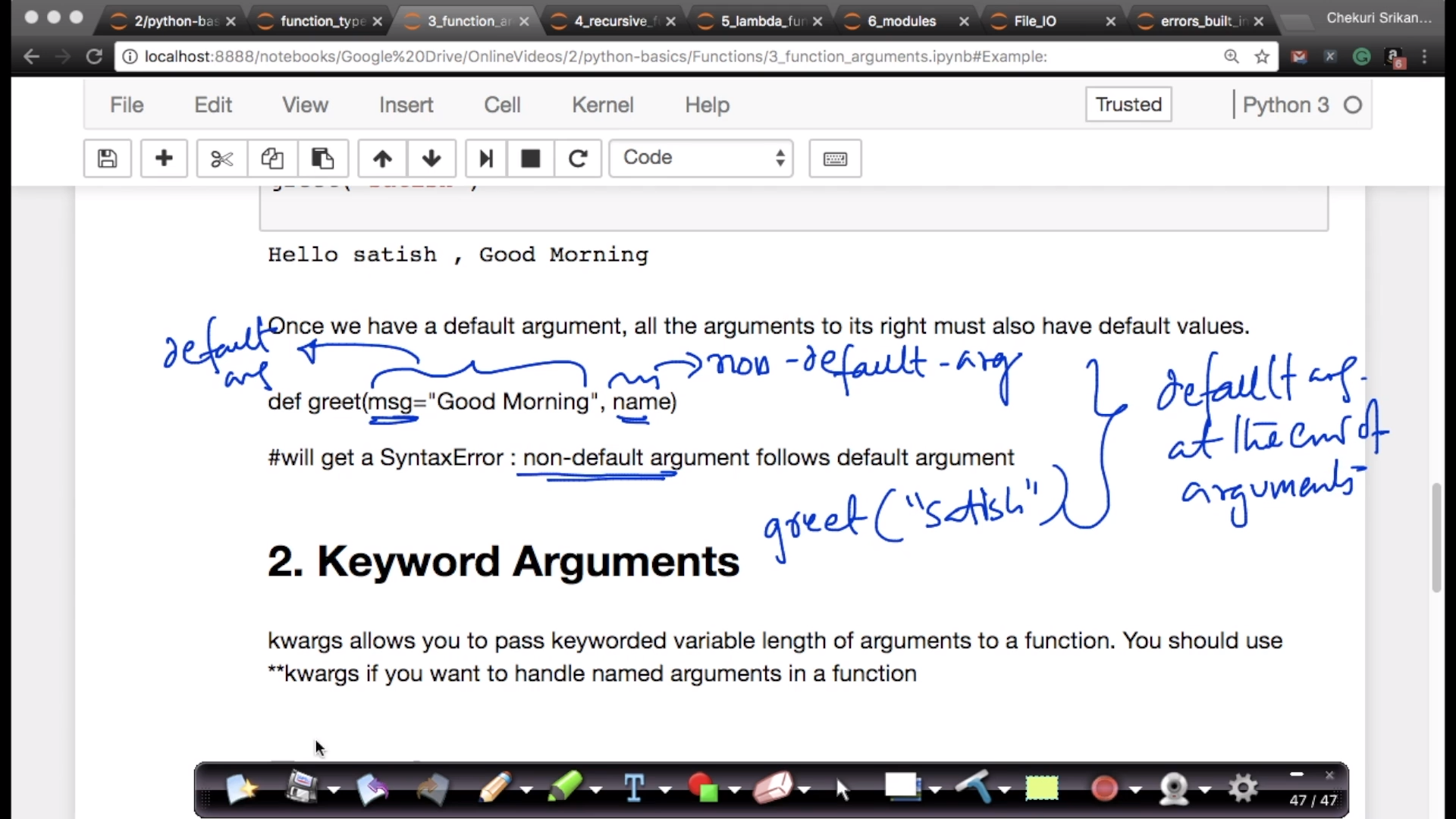
1. Default Arguments

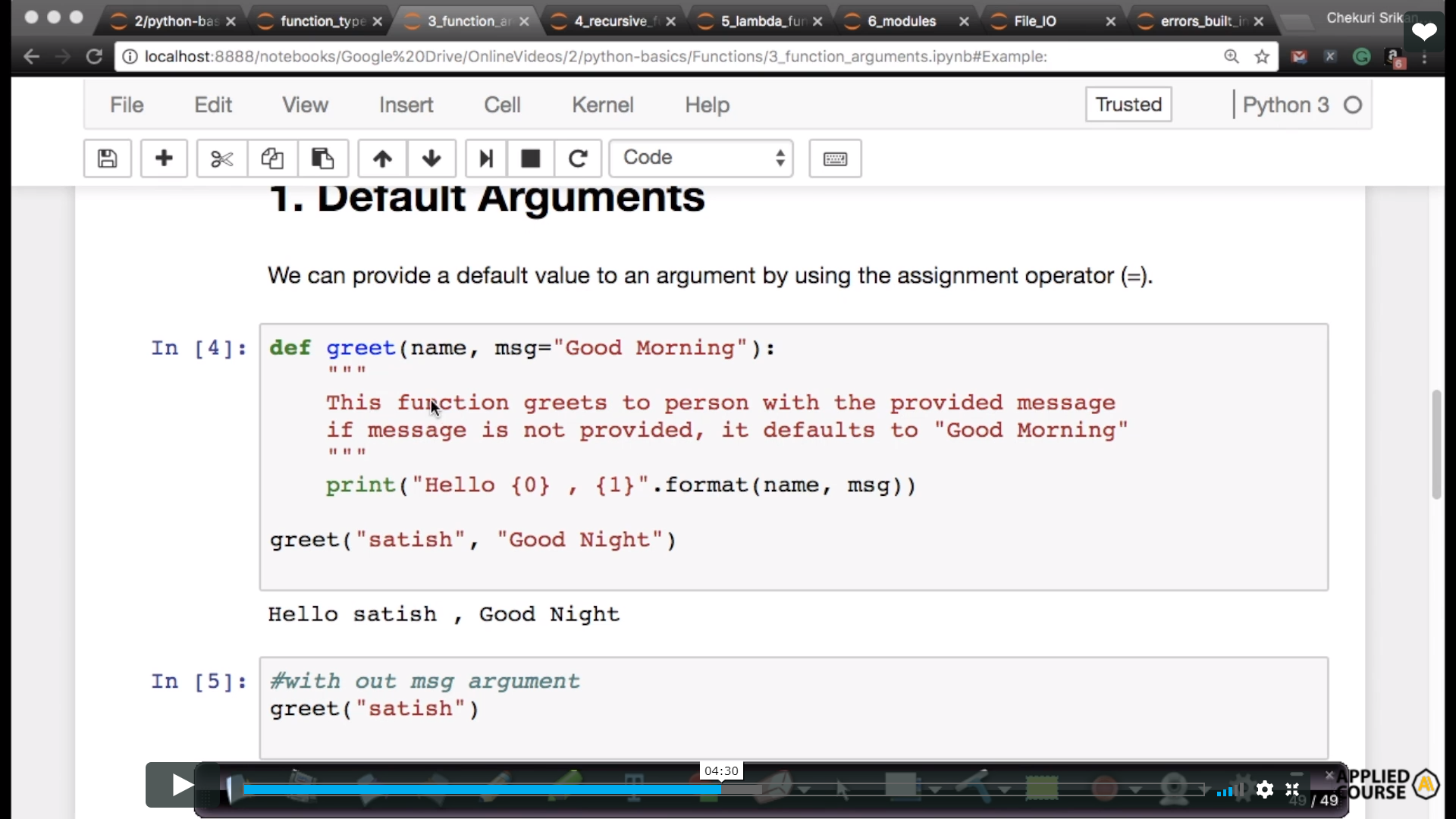
2. Keyword Arguments --(\*\*kwargs):

3. Arbitary Arguments -- \*args

Default arguments :

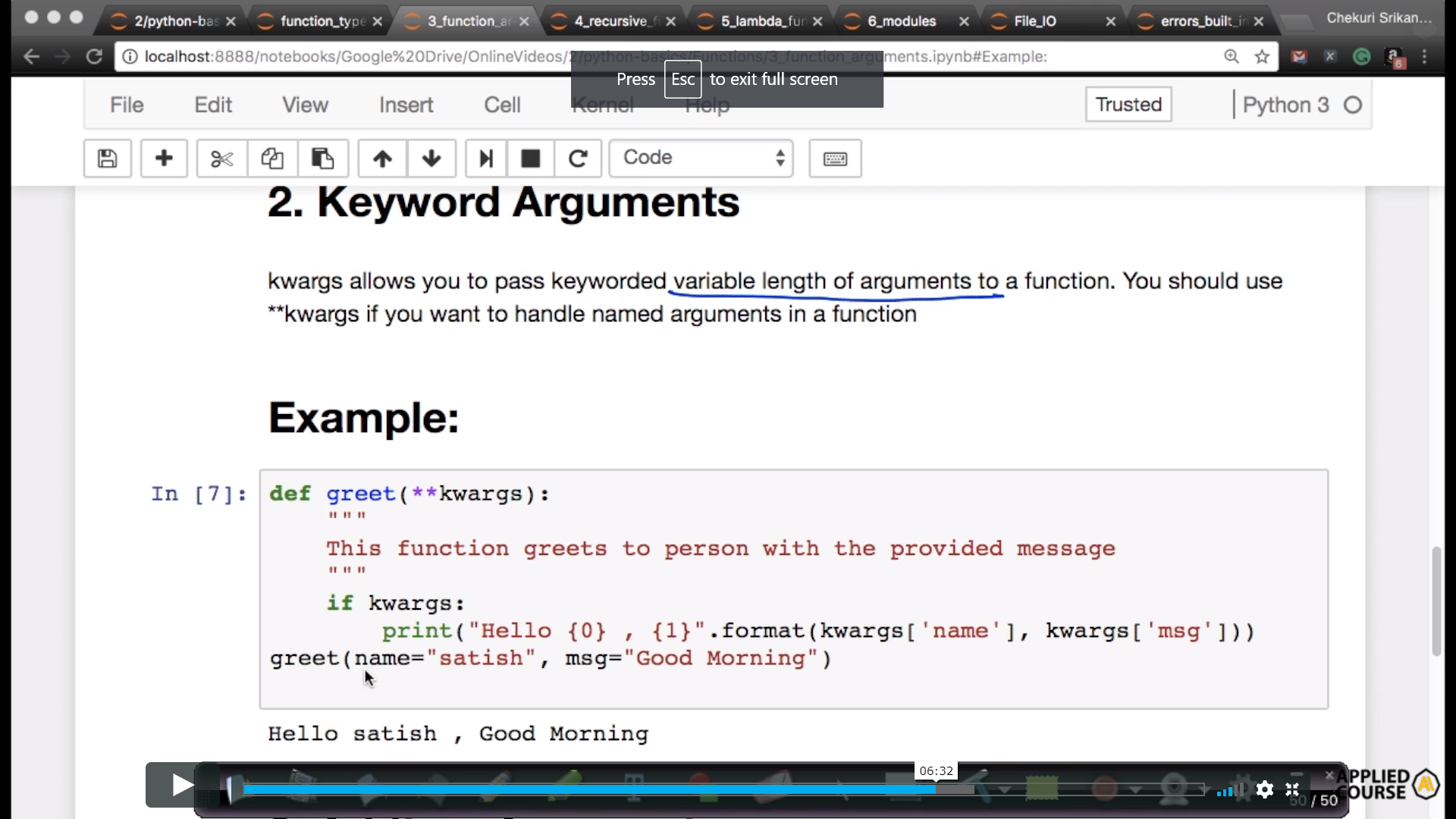
Always default arguments should be at the end i.e first non-default after that non default arguments will be there

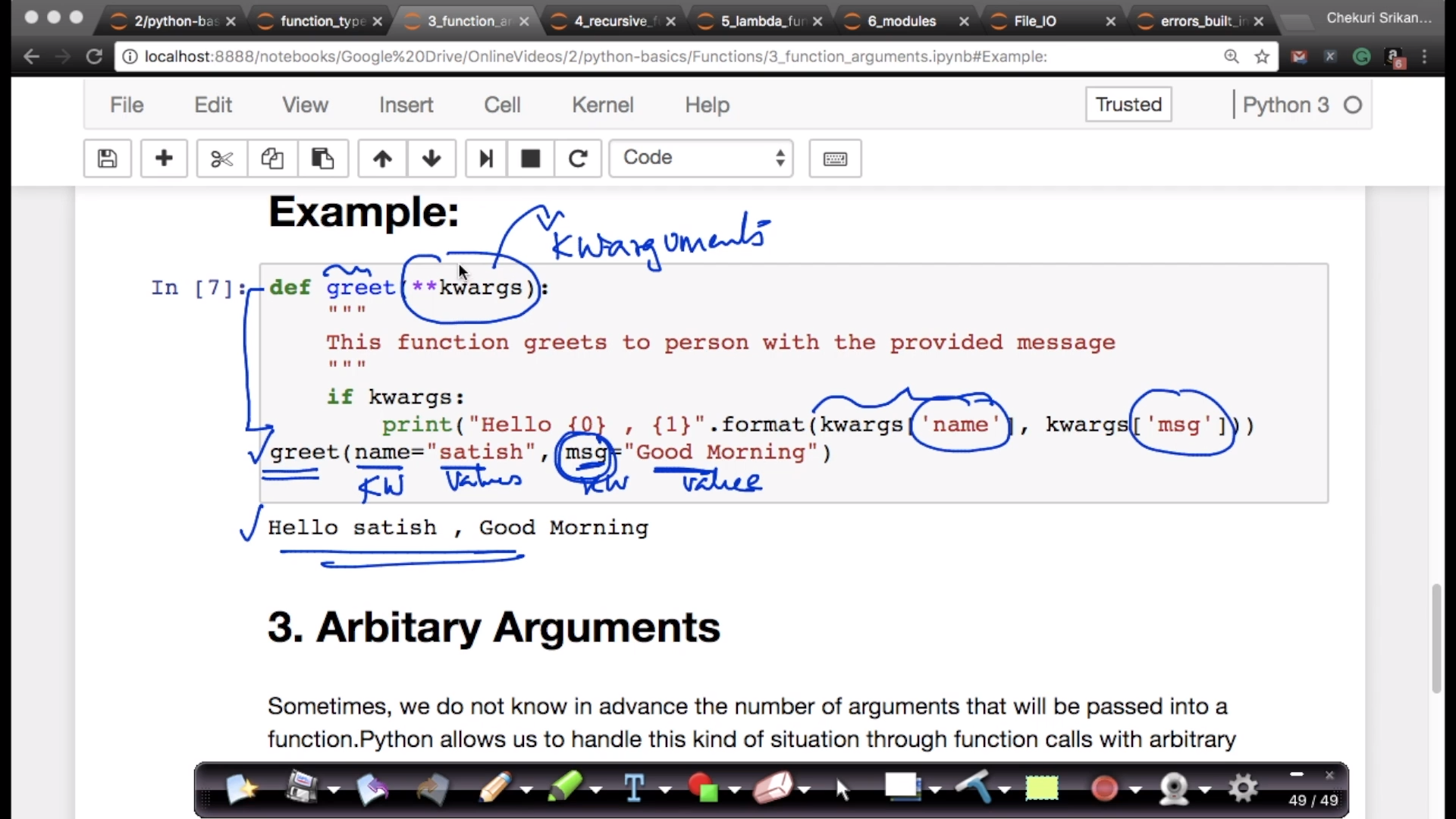




Keyward arguments:

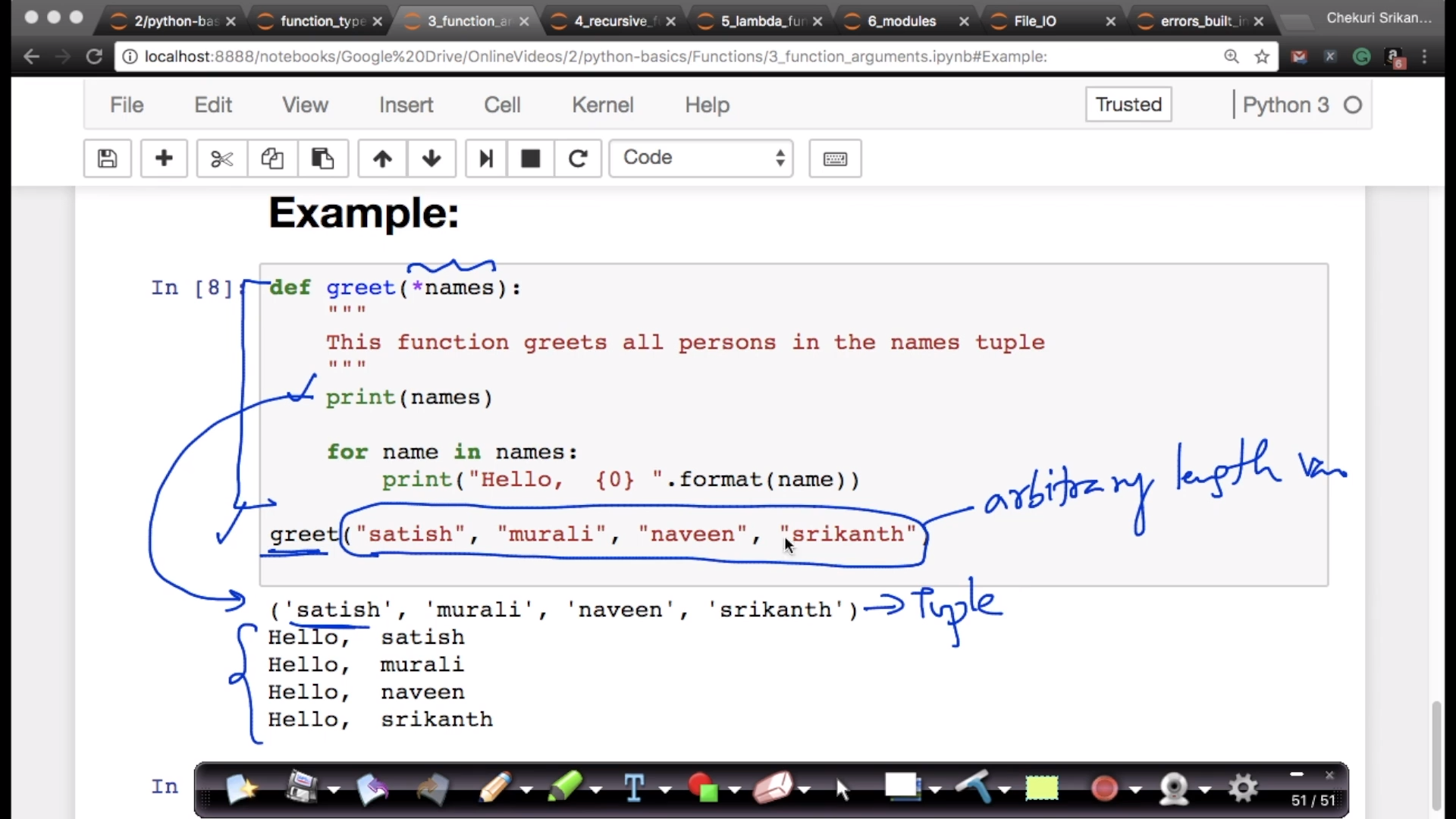
kwargs allows you to pass keyworded variable length of arguments to a function. You should use \*\*kwargs if you want to handle named arguments in a function





Arbitrary arguments:

Sometimes, we do not know in advance the number of arguments that will be passed into a function.Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.



# \*args and \*\*kwargs in Python

**\*args**

The special syntax \*args in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-keyworded, variable-length argument list.

* The syntax is to use the symbol \* to take in a variable number of arguments; by convention, it is often used with the word args.
* What \*args allows you to do is take in more arguments than the number of formal arguments that you previously defined. With \*args, any number of extra arguments can be tacked on to your current formal parameters (including zero extra arguments).
* For example : we want to make a multiply function that takes any number of arguments and able to multiply them all together. It can be done using \*args.
* Using the \*, the variable that we associate with the \* becomes an iterable meaning you can do things like iterate over it, run some higher order functions such as map and filter, etc

**Example for usage of \*arg:**

|  |
| --- |
| # Python program to illustrate  # \*args for variable number of arguments  def myFun(\*argv):      for arg in argv:          print (arg)    myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks') |

**Output:**

Hello

Welcome

to

GeeksforGeeks

# Python program to illustrate

# \*args with first extra argument

def myFun(arg1, \*argv):

    print ("First argument :", arg1)

    for arg in argv:

        print("Next argument through \*argv :", arg)

myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks')

* **Output:**
* First argument : Hello
* Next argument through \*argv : Welcome
* Next argument through \*argv : to
* Next argument through \*argv : GeeksforGeeks

**\*\*kwargs**

The special syntax *\*\*kwargs* in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name *kwargs* with the double star. The reason is because the double star allows us to pass through keyword arguments (and any number of them).

* A keyword argument is where you provide a name to the variable as you pass it into the function.
* One can think of the *kwargs* as being a dictionary that maps each keyword to the value that we pass alongside it. That is why when we iterate over the *kwargs* there doesn’t seem to be any order in which they were printed out.
* **Example for usage of \*\*kwargs:**

|  |
| --- |
| # Python program to illustrate  # \*kargs for variable number of keyword arguments    def myFun(\*\*kwargs):      for key, value in kwargs.items():          print ("%s == %s" %(key, value))    # Driver code  myFun(first ='Geeks', mid ='for', last='Geeks') |

**Output:**

last == Geeks

mid == for

first == Geeks

|  |
| --- |
| # Python program to illustrate  \*\*kargs for  # variable number of keyword arguments with  # one extra argument.    def myFun(arg1, \*\*kwargs):      for key, value in kwargs.items():          print ("%s == %s" %(key, value))    # Driver code  myFun("Hi", first ='Geeks', mid ='for', last='Geeks') |

**Output:**

last == Geeks

mid == for

first == Geeks

**Using \*args and \*\*kwargs to call a function**

Examples:

|  |
| --- |
| def myFun(arg1, arg2, arg3):      print("arg1:", arg1)      print("arg2:", arg2)      print("arg3:", arg3)    # Now we can use \*args or \*\*kwargs to  # pass arguments to this function :  args = ("Geeks", "for", "Geeks")  myFun(\*args)    kwargs = {"arg1" : "Geeks", "arg2" : "for", "arg3" : "Geeks"}  myFun(\*\*kwargs) |

**Output:**

arg1: Geeks

arg2: for

arg3: Geeks

arg1: Geeks

arg2: for

arg3: Geeks

List comprehension:

# thumb rule for list comprehension

**Conditionas in List Comprehension**

only one if condition

number\_list = [ x for x in range(20) if x % 2 == 0]

print(number\_list)

**Nested IF with List Comprehension**

multiple if conditions

num\_list = [y for y in range(100) if y % 2 == 0 if y % 5 == 0]

print(num\_list)

**if...else With List Comprehension**

obj = ["Even" if i%2==0 else "Odd" for i in range(10)]

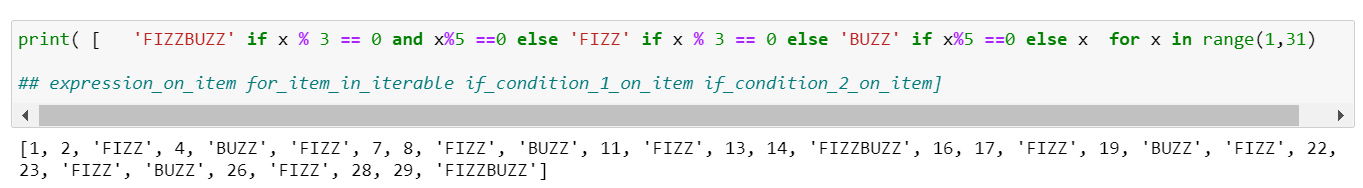
print(obj)

here if i%2==0 it will print ‘EVEN’

🡪 in below code if x%3==0 and x%5==0 satisfy then it will print ‘FIZZBUZZ’ ;

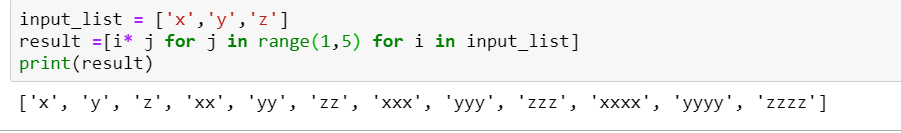
If x%3==0 then ‘FIZZ’

If x%5==0 then BUZZ



refer : <https://www.programiz.com/python-programming/list-comprehension>

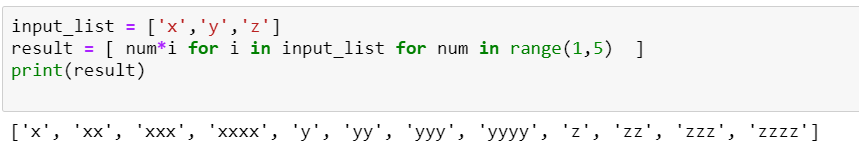
multiple for loops:



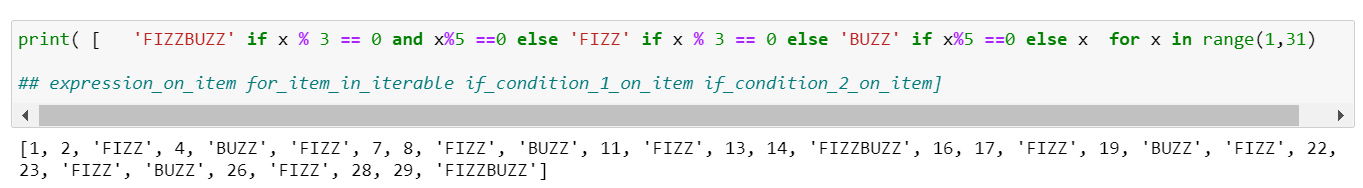
For j in range(1,5) will generate numbers from 1 to 4 i.e 1,2,3,4

For each element in input list we have to multiply with o/p of range(1,5) then

['x', 'y', 'z', 'xx', 'yy', 'zz', 'xxx', 'yyy', 'zzz', 'xxxx', 'yyyy', 'zzzz']



🡪 in below code if x%3==0 and x%5==0 satisfy then it will print ‘FIZZBUZZ’



🡪

String is iterable but number is not iterable

To convert iterable to iterator we will use iter() function

Refer ineuron session 2.1 material