



IIT Madras
ONLINE DEGREE

Programming in Python
Professor. Sudarshan Iyengar
Department of Computer Science & Engineering
Indian Institute of Technology, Ropar
Mr. Omkar Joshi
Course Instructor
Functional Programming (Part 3)

(Refer Slide Time: 0:16)

The image displays two screenshots of a Python REPL environment, likely JupyterLab, showing the execution of Python code. The background features a large, semi-transparent watermark of the Indian Institute of Technology Madras logo.

Top Screenshot: The code defines four regular functions: `add`, `sub`, `mul`, and `div`. Each function takes two arguments, `x` and `y`, and returns the result of the corresponding operation. The code then prints the results of these functions applied to the arguments 10 and 20.

```
1 def add(x, y):  
2     return x + y  
3  
4 def sub(x, y):  
5     return x - y  
6  
7 def mul(x, y):  
8     return x * y  
9  
10 def div(x, y):  
11     return x / y  
12  
13 add = lambda x, y: x + y  
14 sub = lambda x, y: x - y  
15 mul = lambda x, y: x * y  
16 div = lambda x, y: x / y  
17 print(add(10, 20))  
18 print(sub(10, 20))  
19 print(mul(10, 20))  
20 print(div(10, 20))  
21
```

The console output shows the results of these operations: 30, -10, 200, and 0.5.

Bottom Screenshot: The code defines the same four operations using lambda functions. The code then prints the results of these lambda functions applied to the arguments 10 and 20, and finally prints the type of the `add` variable.

```
1 add = lambda x, y: x + y  
2 sub = lambda x, y: x - y  
3 mul = lambda x, y: x * y  
4 div = lambda x, y: x / y  
5 print(add(10, 20))  
6 print(sub(10, 20))  
7 print(mul(10, 20))  
8 print(div(10, 20))  
9  
10 print(type(add))
```

The console output shows the same results as the top screenshot, followed by the type of the `add` variable, which is `<class 'function'>`.

Hello Python students. So far we have seen two different lectures based on functional programming and this is third and last part of functional programming concepts. In this lecture, we will discuss about various different functions which are associated with functional

programming. And the first function is referred as lambda function. Lambda function is a type of function which is anonymous, as in without providing any function name, we can write a function in Python and such functions are referred as lambda functions.

Look at this particular code. Here, I have four different functions, add, subtract, multiply, divide, and I am executing with these values. As we all know, this will give us this kind of a output. If you observe all these four functions has only one line inside it and that too is the return statement. And one might think writing a function only for this one line does not make much sense. So, in such a case, we use this particular feature of functional programming called lambda functions.

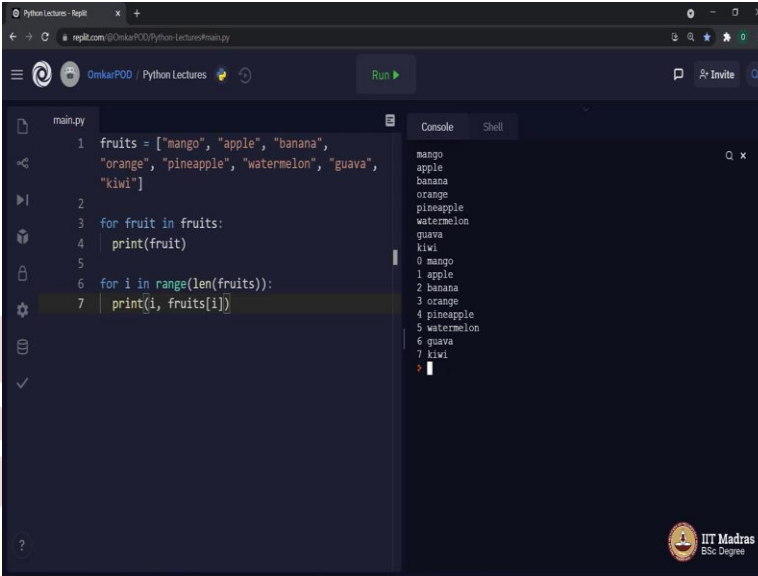
In case of lambda functions, instead of writing these functions, we can simply replace them by using four different variables like add is equal to lambda x comma y, just like these parameters, colon and then this particular statement which is our written statement in original function x plus y. In this case, what happens is we are creating one lambda function as in one anonymous function where there will be two parameters and the operation which we are supposed to perform is x plus y.

Similarly, we can write sub is equal to lambda, once again, x comma y colon x minus y and then similar code for multiplication and division. And from this point onwards, we can use these four variables add, sub, mul, and div in order to execute them as functions. Hence, now we do not need all these function definitions at all. If we execute this code, still we will get the same output.

In this case, this particular lambda function is stored inside a single variable called add, whereas the calling of a function is exactly the way it was earlier. If we try to print the type of that particular variable which stores this lambda function, it says class function. So, this single variable is nothing but a function. And we can use this to refer to these lambda functions individually.

Once again, the advantage of using lambda function is it reduces the length of our program. If we are supposed to write functions which contains very few number of lines or a single statement like this, we can write it in a single line using lambda function.

(Refer Slide Time: 4:31)



The screenshot shows a Python REPL window with a file named 'main.py'. The code in the file is as follows:

```
1 fruits = ["mango", "apple", "banana",  
2 "orange", "pineapple", "watermelon", "guava",  
3 "kiwi"]  
4  
5 for fruit in fruits:  
6     print(fruit)  
7  
8 for i in range(len(fruits)):  
9     print(i, fruits[i])
```

The console output shows the fruits printed one by one, followed by the fruits printed with their indices:

```
mango  
apple  
banana  
orange  
pineapple  
watermelon  
guava  
kiwi  
0 mango  
1 apple  
2 banana  
3 orange  
4 pineapple  
5 watermelon  
6 guava  
7 kiwi
```

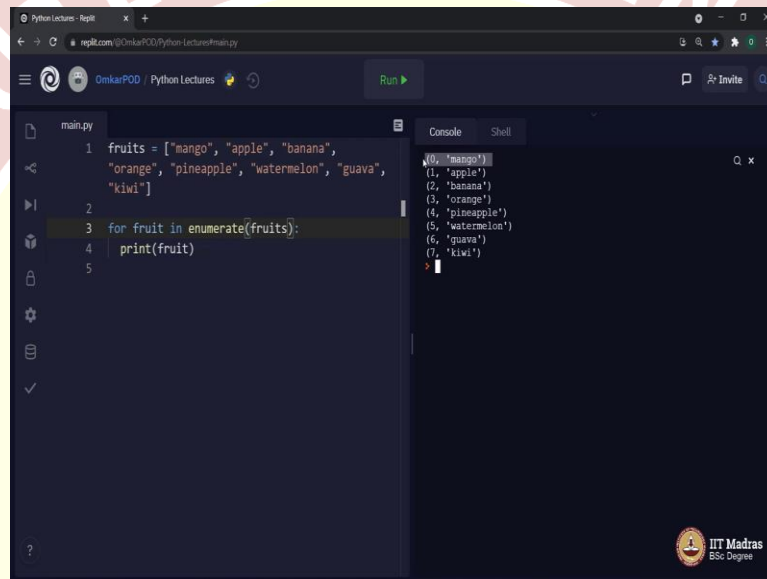
After lambda function, the next function associated with functional programming is something called as enumerate. So, look at this particular code. This is a list of fruits and I have written a loop to iterate over that list and this will be our output. But in this case, I am just getting all these values from the list. Right now it is a small list so that is fine. But what if I have a list with let us say 500 elements in it and I randomly select some element and I want to know what is the index of this element in the list.

Practically, it is not a good idea to count it from 0, 1, 2, 3 and so on, that is not an optimized solution, if you have a very big list. In such a case, we can modify this particular for loop and instead of iterating over this list like this, we will use the other way around using range function, where we will have this index *i* and every time we will print index *i* along with the value for it. And this will give us our expected output like this. And now it is easy to know watermelon is at fifth index, kiwi is at seventh index and so on and this will work for any list of any size.

But if you observe here, this value is 0 and this particular string mango are two different entities as in this *i* is just an integer which we are using for iteration, whereas fruits of *i* is the value in the list and there is no real connection between this value mango and this *i*. Similarly, this 3 and orange has no direct connection, 3 is a different integer and orange is a different entity. And if some calculation goes wrong somewhere, then these index values and these particular strings may not match.

So, somehow, we have to find a way to couple these things together so that whenever we say line number 2, it will give us apple and one together as a single entity. And that is the place this particular function called enumerate helps us. If we use enumerate function, we do not even have to write this particular code. Instead, we can add that enumerate function over here itself, enumerate fruits. Let us execute, and then you will understand the difference between the current output and output with enumerate function.

(Refer Slide Time: 7:55)

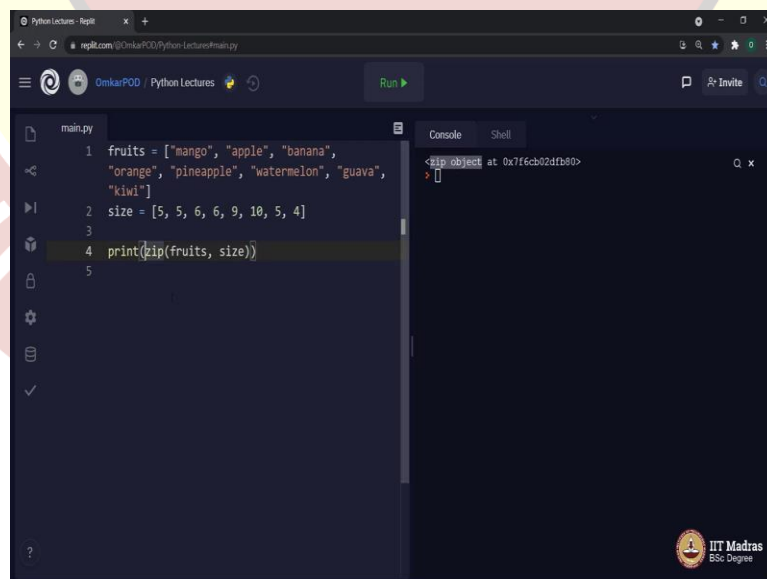


The screenshot shows a Python REPL window with a file named `main.py`. The code in the file is:

```
1 fruits = ["mango", "apple", "banana",  
2 "orange", "pineapple", "watermelon", "guava",  
3 "kiwi"]  
4 for fruit in enumerate(fruits):  
5     print(fruit)
```

The console output shows the following:

```
<0, 'mango'>  
<1, 'apple'>  
<2, 'banana'>  
<3, 'orange'>  
<4, 'pineapple'>  
<5, 'watermelon'>  
<6, 'guava'>  
<7, 'kiwi'>
```



The screenshot shows a Python REPL window with a file named `main.py`. The code in the file is:

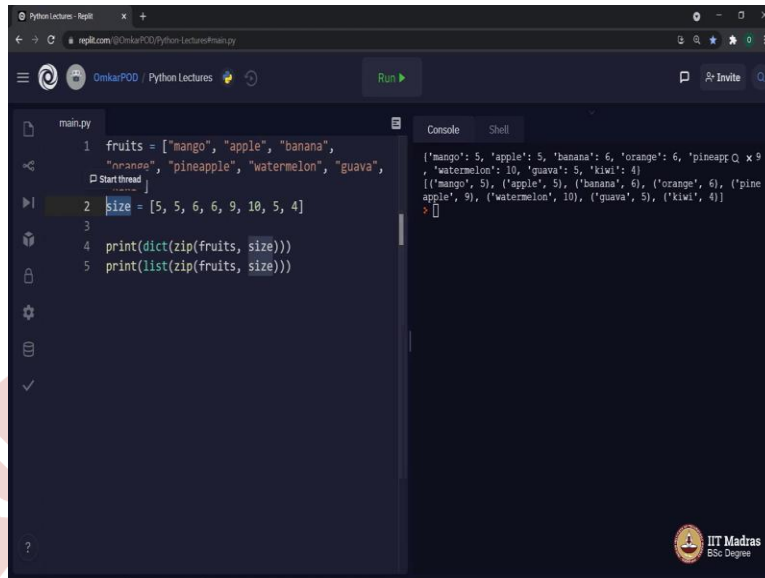
```
1 fruits = ["mango", "apple", "banana",  
2 "orange", "pineapple", "watermelon", "guava",  
3 "kiwi"]  
4 size = [5, 5, 6, 6, 9, 10, 5, 4]  
5 print(zip(fruits, size))
```

The console output shows the following:

```
<zip object at 0x7f6cb02df80>  
>
```

```
PythonLecture - Repl  
repl.it.com/@OmkarPOD/PythonLecture/main.py  
OmkarPOD / Python Lectures  
Run  
main.py  
1 fruits = ["mango", "apple", "banana",  
2 "orange", "pineapple", "watermelon", "guava",  
3 "kiwi"]  
4 print(list(zip(fruits, size)))  
5  
size = [5, 5, 6, 6, 9, 10, 5, 4]  
Console  
[('mango', 5), ('apple', 5), ('banana', 6), ('orange', 6), ('pineapple', 9), ('watermelon', 10), ('guava', 5), ('kiwi', 4)]
```

```
PythonLecture - Repl  
repl.it.com/@OmkarPOD/PythonLecture/main.py  
OmkarPOD / Python Lectures  
Run  
main.py  
1 fruits = ["mango", "apple", "banana",  
2 "orange", "pineapple", "watermelon", "guava",  
3 "kiwi"]  
4 print(dict(zip(fruits, size)))  
5  
size = [5, 5, 6, 6, 9, 10, 5, 4]  
Console  
{'mango': 5, 'apple': 5, 'banana': 6, 'orange': 6, 'pineapple': 9, 'watermelon': 10, 'guava': 5, 'kiwi': 4}
```



```
PythonLecture - Repl
replit.com/@OnkarPOD/PythonLecture/main.py

main.py
1 fruits = ["mango", "apple", "banana",
2 "orange", "pineapple", "watermelon", "guava",
3 ]
4 size = [5, 5, 6, 6, 9, 10, 5, 4]
5 print(dict(zip(fruits, size)))
6 print(list(zip(fruits, size)))

Console
({'mango': 5, 'apple': 5, 'banana': 6, 'orange': 6, 'pineapple': 9, 'watermelon': 10, 'guava': 5, 'kiwi': 4})
[('mango', 5), ('apple', 5), ('banana', 6), ('orange', 6), ('pineapple', 9), ('watermelon', 10), ('guava', 5), ('kiwi', 4)]
```

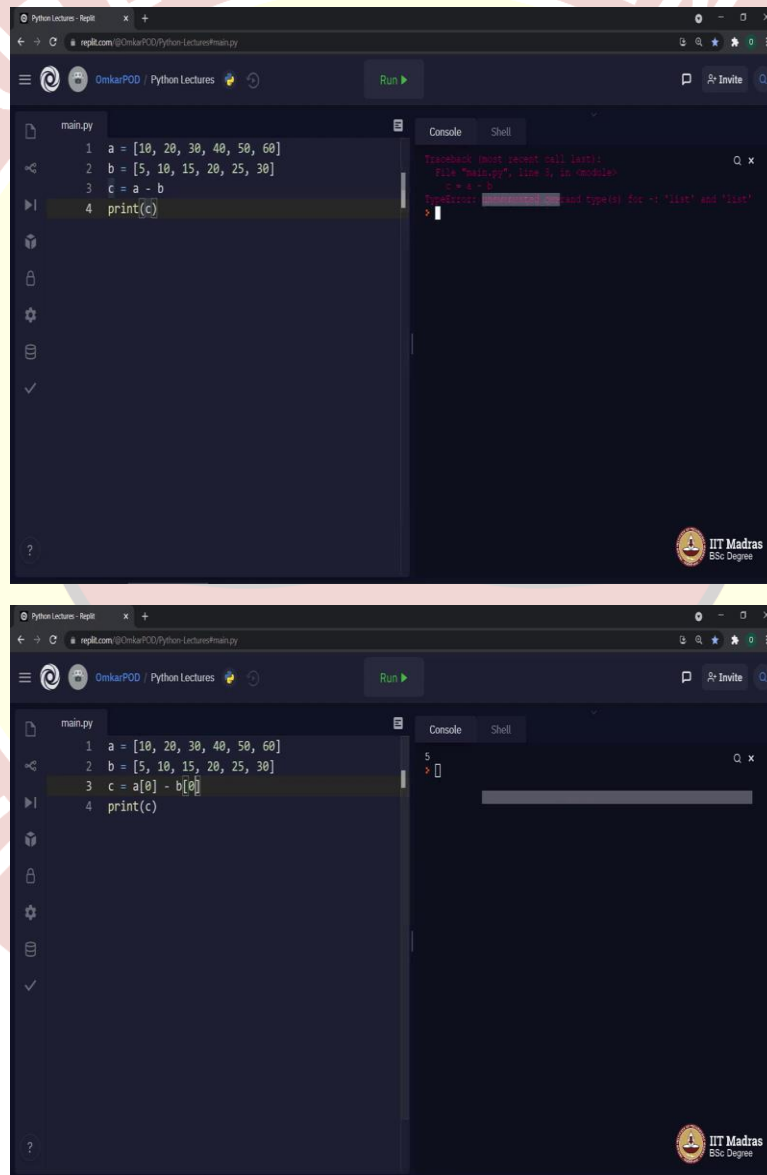
Now, as you can see, this value 0 and mango, it is coupled together inside a single tuple. And because of that, now we can say my fourth tuple is pineapple, my seventh tuple is kiwi and this is how they are coupled together as a single entity. And this particular coupling of index and value with each other leads to a next problem.

What if I want to couple values from two different lists, as in what if I have another list here called size and in that particular list, I am storing the length of every single element in this list fruits. So, mango has 5 characters, hence 5, watermelon has 10 characters, hence 10, kiwi has 4 characters, hence 4. So, if you see, all these are lengths of those strings stored in the same order in which these fruit names are stored. And now I have to find a way to couple values from two different lists, fruits and size.

Now, this enumerate function will not be helpful. Instead, I have to use another function which is there in Python's functional programming and the function is called zip. Print zip fruits comma sites. And if we execute this, it will return our object called zip. It is a zip object. We can convert this zip object into a list. Now, you will get it. This mango is coupled with its length five, watermelon is coupled with its length 10, guava is coupled with its length five and so on. Now, we have a list of tuples. Similarly, if your programming requirement is a dictionary instead of a list, you can simply convert it into a dictionary and it will give you a dictionary with these fruit names which are strings as keys and all these lengths of those strings as its value.

Once again, you can achieve all this in a single line. Now, this time, I will not show you how you can do this without zip. So, first, write a program to get this dictionary and this list with coupled values from list fruits and size. Try to get this dictionary and then this list of tuples without using this zip function. Write that code then you will understand how many numbers of lines it takes, how much effort you have to put in to create these particular two different data structures against the use of zip in single line. Now, let us move to our next function.

(Refer Slide Time: 11:54)



The image contains two screenshots of a Python IDE (replit.com) showing a program execution. The top screenshot shows the code being executed, and the bottom screenshot shows the resulting error message.

Top Screenshot (Code):

```
main.py
1 a = [10, 20, 30, 40, 50, 60]
2 b = [5, 10, 15, 20, 25, 30]
3 c = a - b
4 print(c)
```

Bottom Screenshot (Error Message):

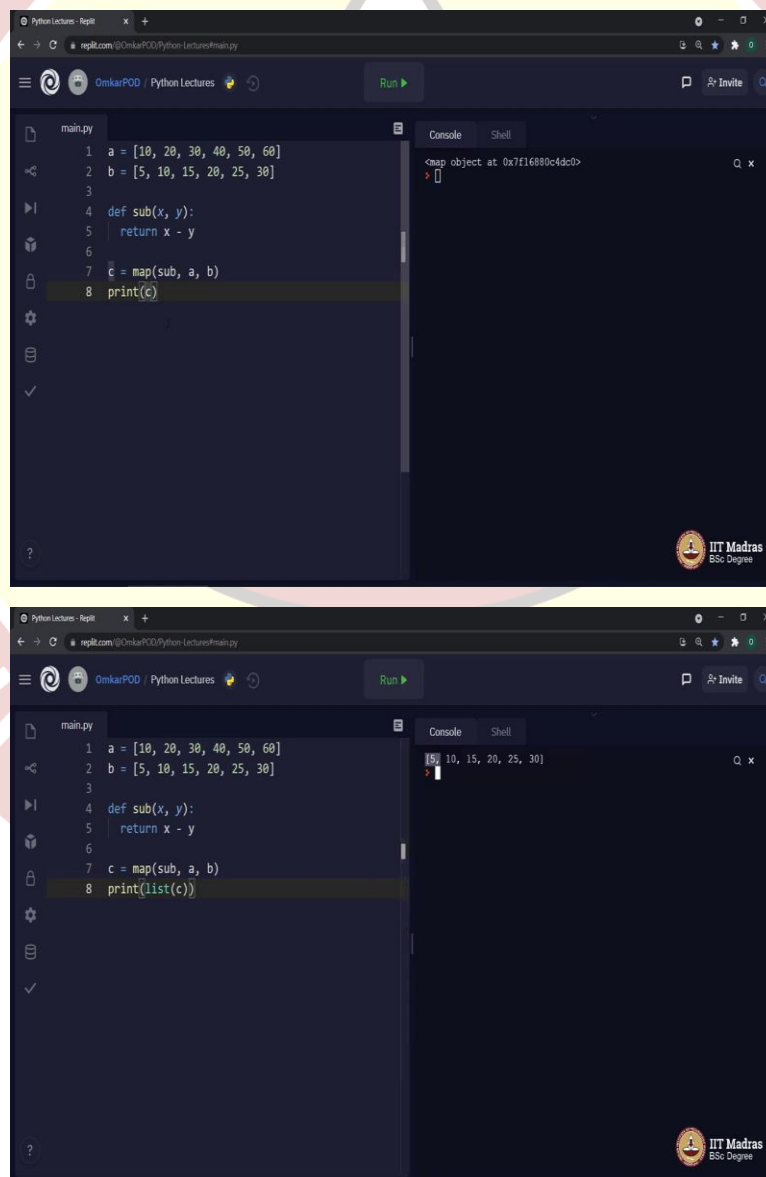
```
Traceback (most recent call last):
  File "main.py", line 3, in module
    c = a - b
TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

Consider these two lists, a and b. And now I want to create third list called c with a minus b, print c. If we execute this particular code, we all know computer will give us an error message

saying unsupported operand types for list and list. You cannot use this minus operator, this subtraction operator with two lists a and b, because you cannot say list minus list. You can say some element in list minus some element in list.

So, instead of this, you can say a of 0 minus b of 0 and then you will get the output as 5 that is possible, but you cannot say list minus list, which means in this case you will have to write a loop which will iterate over two different lists and then only you can create this third list. And in order to avoid this particular problem, there is a function available in Python called map. This map function will help us do exactly this, c is equal to a minus b. Let us see how that works.

(Refer Slide Time: 13:37)



The image displays two screenshots of a Python IDE, likely Replit, showing a script named 'main.py' and its execution output in the console.

Top Screenshot:

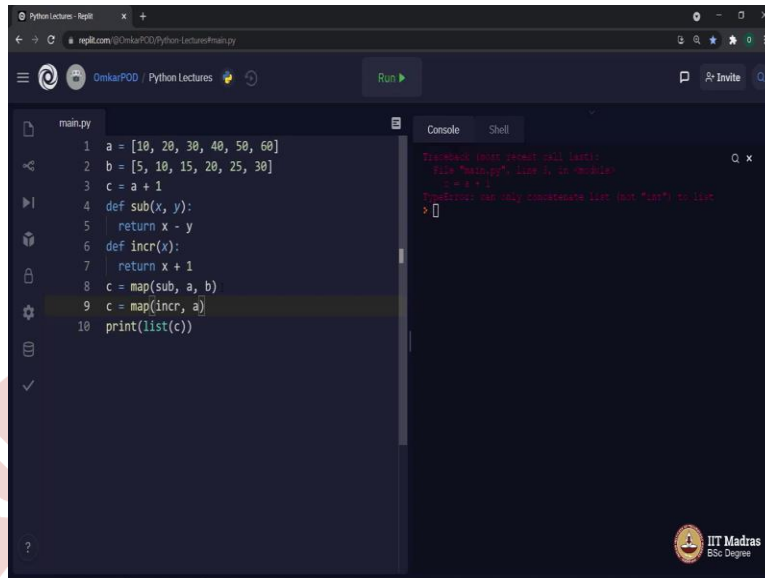
```
1 a = [10, 20, 30, 40, 50, 60]
2 b = [5, 10, 15, 20, 25, 30]
3
4 def sub(x, y):
5     return x - y
6
7 c = map(sub, a, b)
8 print(c)
```

The console output shows: `<map object at 0x7f16880c4dc0>` followed by a cursor, indicating that the code has been executed but the result is not yet printed.

Bottom Screenshot:

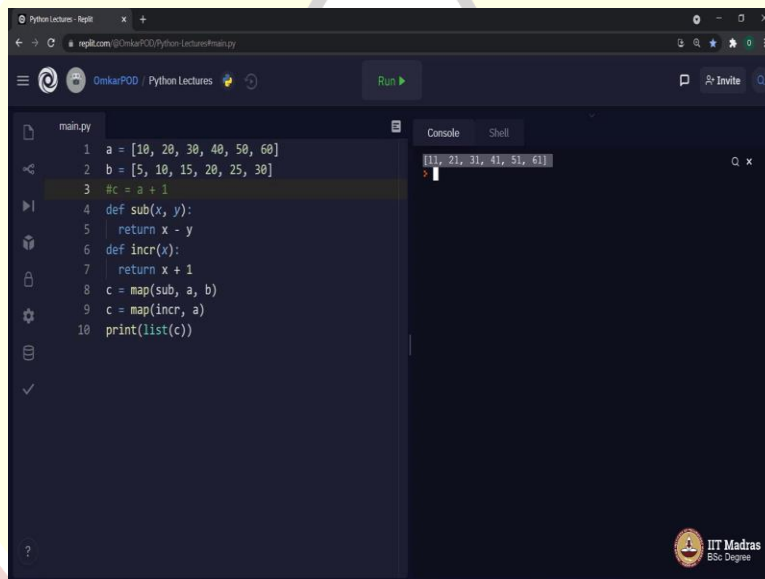
```
1 a = [10, 20, 30, 40, 50, 60]
2 b = [5, 10, 15, 20, 25, 30]
3
4 def sub(x, y):
5     return x - y
6
7 c = map(sub, a, b)
8 print(list(c))
```

The console output now shows: `[5] 10, 15, 20, 25, 30]`, indicating that the code has been executed and the result of the subtraction is displayed.



```
1 a = [10, 20, 30, 40, 50, 60]
2 b = [5, 10, 15, 20, 25, 30]
3 c = a + 1
4 def sub(x, y):
5     return x - y
6 def incr(x):
7     return x + 1
8 c = map(sub, a, b)
9 c = map(incr, a)
10 print(list(c))
```

Traceback (most recent call last):
File "/main.py", line 1, in module:
c = a + 1
Problem: use only iterable list (not "list") to list
> []



```
1 a = [10, 20, 30, 40, 50, 60]
2 b = [5, 10, 15, 20, 25, 30]
3 #c = a + 1
4 def sub(x, y):
5     return x - y
6 def incr(x):
7     return x + 1
8 c = map(sub, a, b)
9 c = map(incr, a)
10 print(list(c))
```

[11, 21, 31, 41, 51, 61]

def subtract x comma y return x minus y. And now c is equal to map, map what, this particular function called subtract with two values which is a and b which are these two lists. Now, if you execute this code, c is a map object. We can convert this into a list. And if we execute again, we will get the expected list with subtraction of each element inside it. I assume you all must have understood what map is doing. But still let me explain it.

The map function takes two different types of arguments; first is a function name which it supposed to execute and then all those parameters which that particular function accepts. So, if this particular function takes only one parameter, then over here this map function will also take

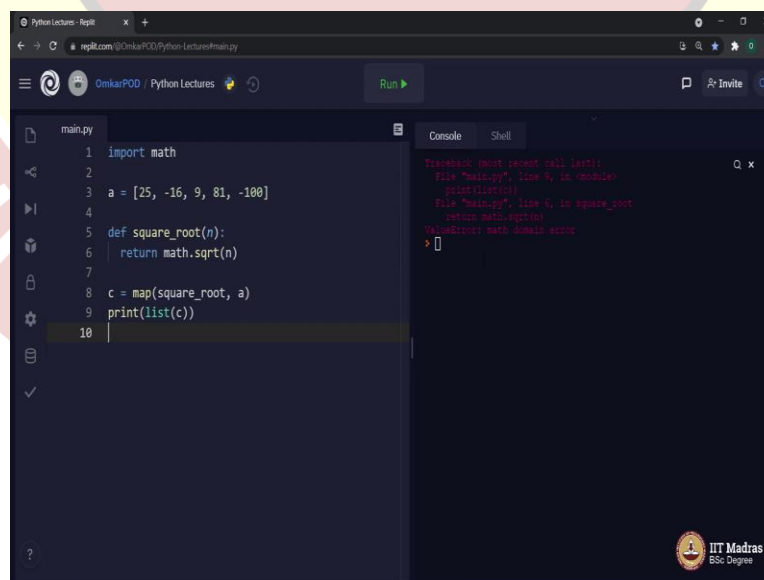
only one parameter along with function name. If function takes three parameters, then map will also take three parameters along with function name.

So, what this map function do is, map function implements a concept of mapping of function with its iterators. So, in this case, this map function will iterate over these two lists and at the same time it will execute this function for every single entry inside these two lists. And that is why it is very helpful when we want to repeat some operation for every single element inside a list.

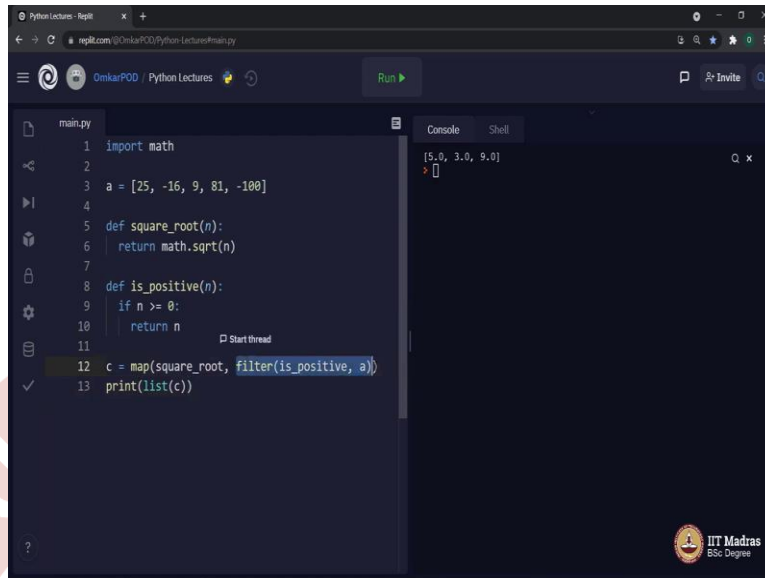
For example, I want to do something like c is equal to a plus 1. Of course, this is not a valid Python statement. But what I am trying to say over here is, I want to increment every single value in this list a without using any iteration. In such a case, I can simply write one more function called def increment and simply return x plus 1. And this time, I will make it c is equal to map of incr comma a.

Let us execute. I should comment in this code and we got our expected output. And this is something which you can do for a list of any sides without even writing any kind of a loop over it. And this is possible because this particular function called map. Now, let us move to our last function from functional programming in Python.

(Refer Slide Time: 17:20)



```
PythonLectures - Repl  
replit.com/@OmkarPOD/PythonLectures/main.py  
Run  
main.py  
1 import math  
2  
3 a = [25, -16, 9, 81, -100]  
4  
5 def square_root(n):  
6     return math.sqrt(n)  
7  
8 c = map(square_root, a)  
9 print(list(c))  
10  
Console  
Traceback (most recent call last):  
  File "main.py", line 6, in module:  
    print(list(c))  
  File "main.py", line 6, in square_root:  
    return math.sqrt(n)  
ValueError: math domain error  
IIT Madras  
BSc Degree
```

A screenshot of a Python IDE window titled 'PythonLecture - Replit'. The code editor shows a file named 'main.py' with the following code:

```
1 import math
2
3 a = [25, -16, 9, 81, -100]
4
5 def square_root(n):
6     return math.sqrt(n)
7
8 def is_positive(n):
9     if n >= 0:
10         return n
11
12 c = map(square_root, filter(is_positive, a))
13 print(list(c))
```

The console on the right shows the output:

```
[5.0, 3.0, 9.0]
```

 The IDE interface includes a 'Run' button and a 'Shell' tab. A watermark for 'IIT Madras BSc Degree' is visible in the background.

Let us look at this particular code block. Here once again, I am using this function called map. I am mapping this square root function with list a, as in I have one list called a and I want to take a square root of every single value inside this list. I am using this math dot square root method from math library. Let us execute and then we will see.

As expected, it is giving us some error. It is value error, math domain error. This is happening, because whenever we execute this math dot square root method using this minus 16 value, we are getting this error, because we are trying to compute square root of a negative number. And because of that, for this minus 16 and minus 100, we will get this error.

So, now the question is how to avoid these negative values while computing square root. And answer is that last function which I mentioned called filter, we can filter over this list and if we find a particular value is a valid entry, then only we will execute this math dot square root. And if it is a negative number, then we will not execute this particular method for that particular value using map function.

So, now let us add that filter function for this particular list a. It says filter is positive a. Just like map function, filter function also takes two types of parameters; first type is function name and second type is a list of parameters to this particular function. Which means now we have to write a function called is positive. Def is positive, parameter n, if n is greater than equal to 0, return n. Once again, over here, we are performing this particular operation over entire list in a single go using filter function, just the way we were doing it using map function.

So, this filter function will create a list which is filtered version of this original list and then that particular filtered list will be given to the square root function which will execute math dot square root function to get us square roots of only 25, 9 and 81 skipping minus 16 and minus 100. Let us execute and now we have those values. And that is how this filter function helps in the execution of map function. Therefore, most of the times you will end up using both these functions together. And that is the last concept from functional programming.

Thank you for watching this lecture. Happy learning.

