

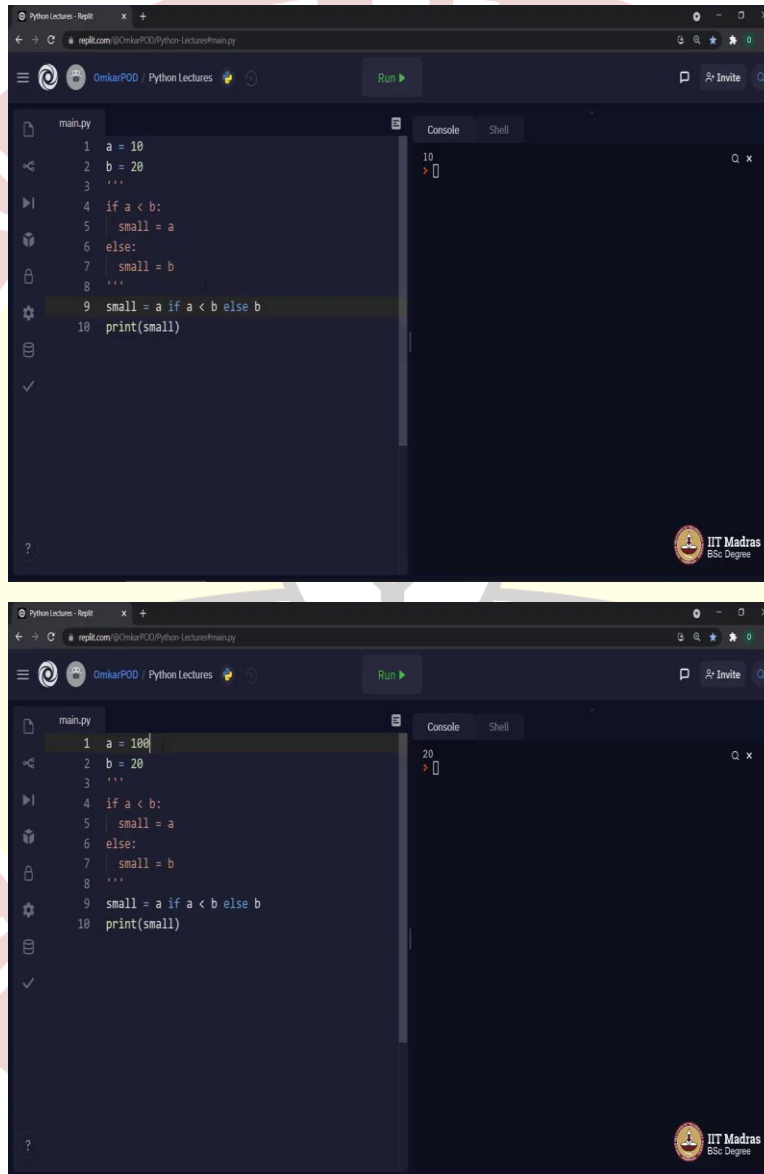


IIT Madras

ONLINE DEGREE

Programming in Python
Professor. Sudarshan Iyengar
Department of Computer Science & Engineering
Indian Institute of Technology, Ropar
Functional Programming (Part 2)

(Refer Slide Time: 0:16)



The image displays two screenshots of a Python IDE (likely Replit) showing a Python script and its execution results. The script defines variables `a` and `b`, and uses a conditional statement to assign the value of `small` based on whether `a` is less than `b`. The ternary operator is also used to assign `small`.

Top Screenshot: The code is as follows:

```
1 a = 10
2 b = 20
3 ...
4 if a < b:
5     small = a
6 else:
7     small = b
8 ...
9 small = a if a < b else b
10 print(small)
```

The console output shows the value `10`.

Bottom Screenshot: The code is the same, but the value of `a` has been changed to `100`:

```
1 a = 100
2 b = 20
3 ...
4 if a < b:
5     small = a
6 else:
7     small = b
8 ...
9 small = a if a < b else b
10 print(small)
```

The console output now shows the value `20`.

Hello Python students. In this lecture, we will continue where we left in previous lecture with respect to functional programming. Earlier we saw iterator and generator. Now, we will look into the third point with respect to functional programming which is inline statements. So, first

observe this code, a is equal to 10, b is equal to 20, if a is less than b, small is equal to a, else small is equal to b, print small and we have our output.

Once again, this is very easy program. What if I tell you that this particular block of the code, forget about this initialization and print statement, this particular block which is if-else block, this particular block I can write it in a single line. Shocked, but believe me it is possible using Python feature called functional programming. This entire code can be replaced by a single line. Still it will have same output.

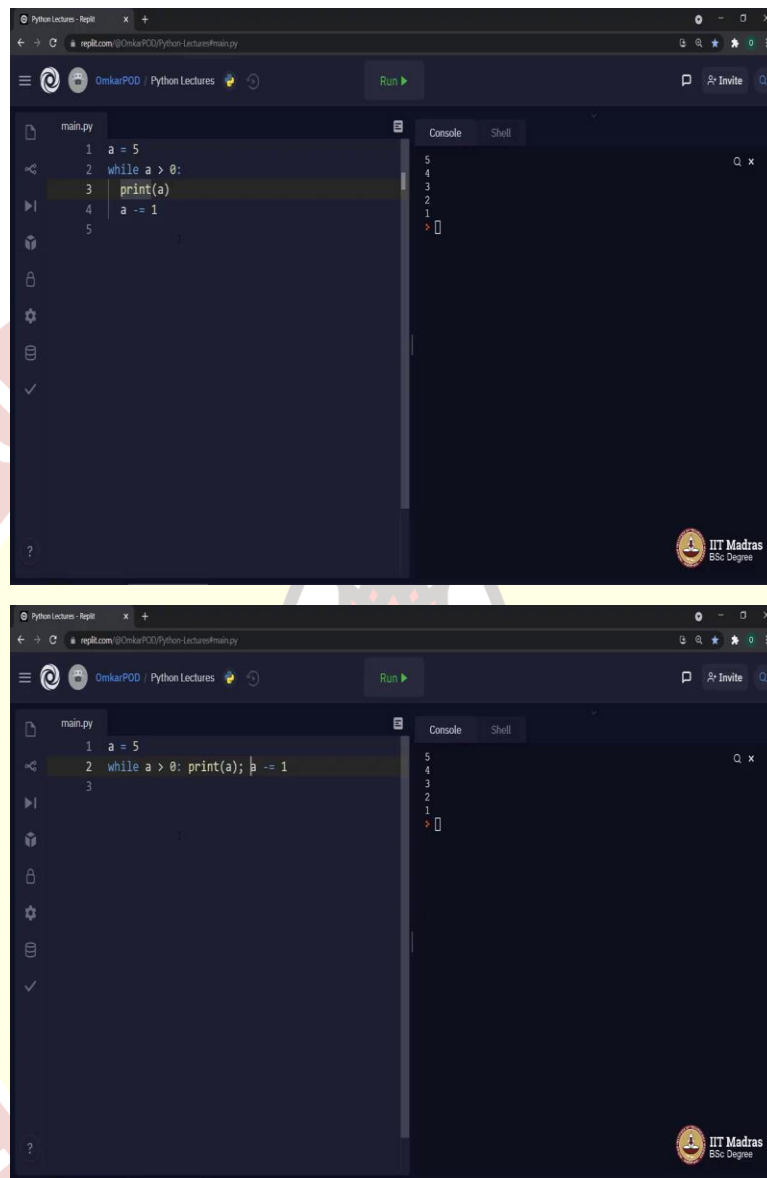
So, let me first commend this code. Now, I will write that single statement which has same effect just like these four lines. Small is equal to a if a is less than b, else b. Now, let us first execute and the output is 10 because a is less than b. What if I make it 100? Now, the output is 20. And that is exactly how even these four lines of code was working.

What happens over here is, we are assigning this value a to variable small if a is less than b. If that is not the case, which means if we are supposed to execute else, then we are assigning this variable b to this particular variable small. And this is how you can convert your four lines of code into a single line. And that is the only advantage we get over here.

With respect to the time taken or with respect to the performance of the code, these four lines and this one line has same output. Both will take same amount of time and both are equal with respect to Python or the computer. Only advantage is instead of writing this in four lines we can write it in a single line and reduce the length of our program by using this everywhere possible.

Now, I am sure you must be wondering, is it possible only with if-else or can we do something like this with loops as well? And the answer is yes we can write even our loops in single line.

(Refer Slide Time: 4:05)



```
main.py
1 a = 5
2 while a > 0:
3     print(a)
4     a -= 1
5
```

```
main.py
1 a = 5
2 while a > 0: print(a); a -= 1
3
```

Console

```
5
4
3
2
1
>
```

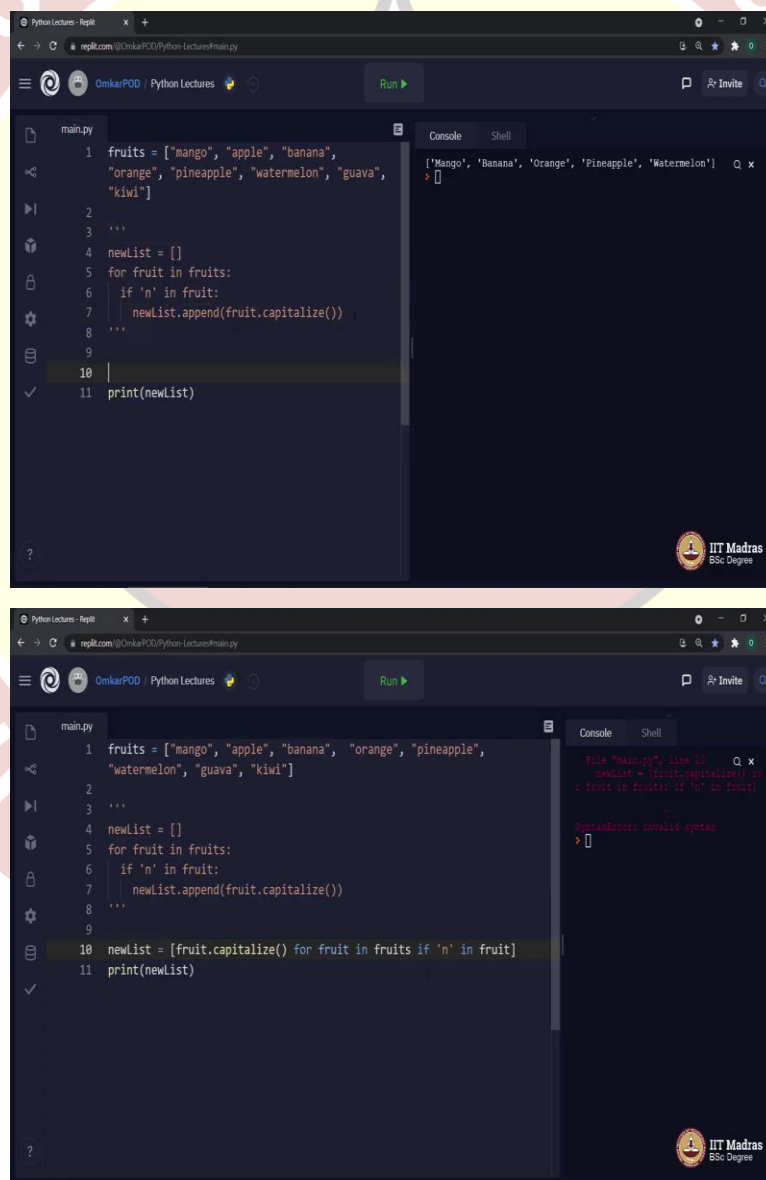
So, let me first write one simple code, let us say, a is equal to 5, while a is greater than 0, print a, a is equal to a minus 1, as in a minus equal to 1 and we will get our output from 5 to 1. This same code can be written in a single line by making small change like this. You bring print over here, add a semicolon and then get this statement as well in front of it. If we execute it, still it will give us the same output.

This is very useful if we have such small statements and only few lines of code inside while. But if you have a very lengthy code and all the statements are very big, it becomes very difficult to read that code if it is in single line. So, this kind of code we can write, but it is up to us to make a

decision whether to write it in multiple lines or write it in a single line. Once again, it reduces the number of lines. Otherwise, there is no difference between this code and the code which we had earlier with those three lines of while loop.

Now, the next question is, we did it with while, can we do this with for loop as well? And the answer is, yes, we can do that. But instead of doing it directly with for loop, let me introduce new concept where we will anyway use this particular inline for. And that next concept is called as list comprehension. Let me repeat, it is called list comprehension.

(Refer Slide Time: 6:04)



The image displays two screenshots of a Python IDE, likely JupyterLab, showing the implementation of a list comprehension. The IDE interface includes a file explorer on the left, a code editor in the center, and a console on the right. The code is written in a dark-themed editor.

Top Screenshot: The code in `main.py` uses a `for` loop with an `if` condition to filter and capitalize elements from a list.

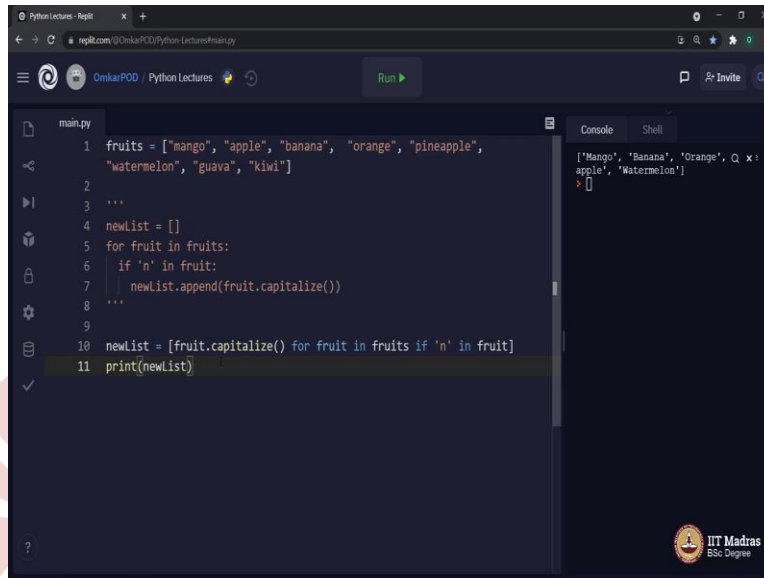
```
1 fruits = ["mango", "apple", "banana", "orange", "pineapple", "watermelon", "guava", "kiwi"]
2
3 '''
4 newList = []
5 for fruit in fruits:
6     if 'n' in fruit:
7         newList.append(fruit.capitalize())
8 '''
9
10
11 print(newList)
```

The console output shows the resulting list: `['Mango', 'Banana', 'Orange', 'Pineapple', 'Watermelon']`.

Bottom Screenshot: The code in `main.py` uses list comprehension to achieve the same result in a single line.

```
1 fruits = ["mango", "apple", "banana", "orange", "pineapple", "watermelon", "guava", "kiwi"]
2
3 '''
4 newList = []
5 for fruit in fruits:
6     if 'n' in fruit:
7         newList.append(fruit.capitalize())
8 '''
9
10 newList = [fruit.capitalize() for fruit in fruits if 'n' in fruit]
11 print(newList)
```

The console output shows the resulting list: `['Mango', 'Banana', 'Orange', 'Pineapple', 'Watermelon']`.

A screenshot of a web-based Python IDE. The editor shows a file named 'main.py' with the following code:

```
1 fruits = ["mango", "apple", "banana", "orange", "pineapple",  
2 "watermelon", "guava", "kiwi"]  
3 ...  
4 newList = []  
5 for fruit in fruits:  
6     if 'n' in fruit:  
7         newList.append(fruit.capitalize())  
8 ...  
9  
10 newList = [fruit.capitalize() for fruit in fruits if 'n' in fruit]  
11 print(newList)
```

The 'Console' tab on the right shows the output:

```
['Mango', 'Banana', 'Orange', 'apple', 'Watermelon']
```

 The IIT Madras logo is visible in the bottom right corner of the IDE interface.

Let us clear the screen and now look at this particular code. Once again, I am using that same list called fruits. I created a new list called new list, then I am iterating over this particular fruits list. And if the character n exists in any of the fruits, I am appending that particular fruit to this particular list. In addition to that, at the time of adding that fruit into a new list, I am also capitalizing that particular word, as in the first letter of each, this fruits will become an uppercase letter. Let us execute.

As you can see, only these are the fruits which contains this particular letter n. And now this small m became capital M, this small b became capital B and so on. So, in this particular code block we are doing two things. We are filtering over this particular list with this condition. And if the condition is satisfied, we are adding that element into a new list with small change in its value.

Now, using this new concept which I just said, which is called as list comprehension and the concept which we saw earlier, which was that inline statements, we will try to implement a code block which will do all these things once again in a single line. So, let me commend this code. And this is the place where now I will write that particular single line, new list is equal to, it has to be a list, let me use this blocks exactly from here, so that you will know how it works, fruits dot capitalize and we are supposed to do this for every fruit in fruits if n in fruits.

Let us execute. It says there is a syntax error. The mistake is this particular colon should not be there. Now, let us execute. And we have our output exactly the way it was when we executed

these few lines of code. Now, once again, operationally, it reduces the number of lines. Other than that, functionally, there is no difference between this line number 10 and these four lines over here.

So, what happens is, we say new list is equal to fruit dot capitalize for every fruit in this particular list, if n in fruits, and that is how you should read it or that is how you should interpret this particular line of code. This is how we wrote inline for, inline if and putting all this together in a list, creating a new list. This is what that list comprehension is all about.

I hope you have understood these concepts. I do not want to elaborate too much using many examples for this. You can try any of the Python programs which you have written before and convert them into new codes using these functional programming features. I am sure even have a lot of Python programs where you have used lists, loops, if-else blocks and so on. So, pick any of those previously done codes or any previous problem statements and now this time try to write a Python program with same logic but with a functional programming approach.

And then you will see earlier if the code was taking 10 lines, maybe now you can finish it in 3 or 4 lines. And that is the biggest advantage of this particular feature called functional programming in Python. Thank you for watching this lecture. Happy learning.

