Scientific Computing Using Python

Professor. Vivek Aggarwal and Professor. Mani Mehra

Department of Mathematics

Indian Institute of Technology, Delhi

Lecture No. 03

So, welcome to Scientific Computing Using Python. In the last two lectures, we have done a lot of Python commands. So today, we will do further work that will be useful for us in the future. Let's get started.

I have opened a new file, and I have named it Lecture 3. So, in this, today Let us start the Python operator. If we write it (Python operator) with # and mark it down: will get

Python operators

So, Python operators are mathematical operations like 3 + 4. We did 3 - 4, and -1 came. So, if we want to multiply, then in multiplication we will write this:

-2 * 5

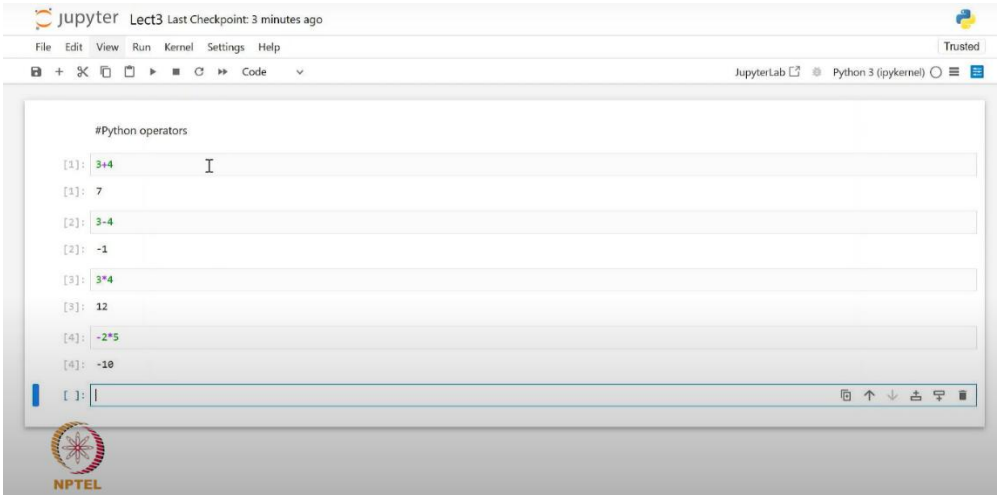
This is the answer(-10). Similarly, we have to see how to do division. So, we did division by:

3 / 4

Backslash is being used. So, this is:

24 / 5

(Refer slide time: 1:32)



Okay, so for division we will do this. Now, we have to take the power. So, if I want to square 3:

3 ** 2

So, if we put a double star, what does it mean? If we want to get power, then we have to take double star. 3 ** 4 will give you 81. Okay, so if we want to get power, then we have to put double star.

Now, there is another operator, like 3%2, let's see what comes out. 1 comes out. So, what does it give us? It gives us a remainder. We also call it the Modulus function. So, if we want to write it, we can write it here as a comment: # Modulus

So, we already know what it does. It gives us a remainder. If we divide 3 by 2, what does our remainder come out? Now, if I divided 4 by 2, then it got zero. Why? Because 2 completely divides 4. So, the remainder of this — the remaining part — will be zero. So, that's why we use it a lot.

Like:

10 % 3

1 comes out. So, if we are dividing 10 by 3, then we know that the remaining part will be 1 only, since 3 × 3 = 9. So, it came out like this. This is a mathematical operation which we will use.

We can define Python assignments:

I will also define this by putting a backslash, and I will mark it down here and press Enter:

Python Assignment operators

So, Python assignment means, like we have defined any of our variables x, I have defined it and given its value:

x = 5

So, what did I do? I assigned 5 to the variable x.

Now I write:

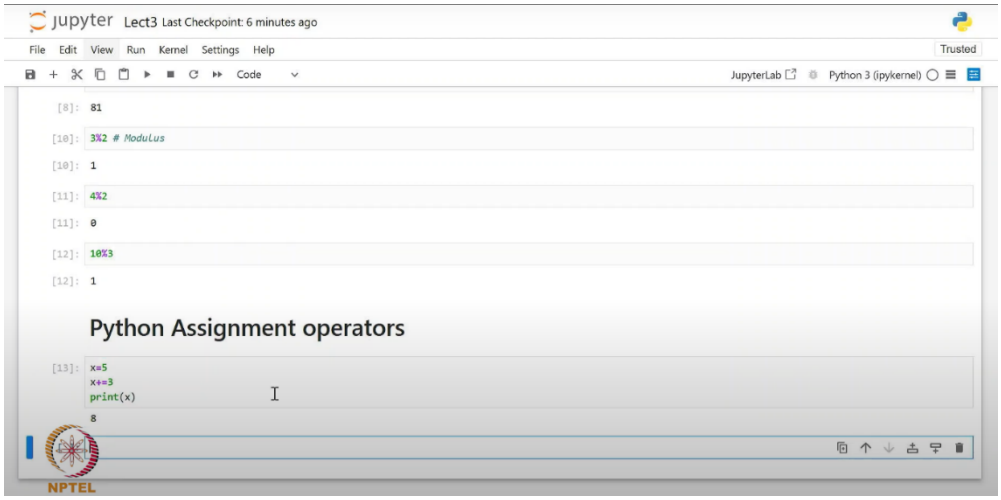
x += 3

So, after this, I wrote:

print(x)

and pressed Enter. So, it came to 8. What does it mean? It means that we are assigning this to the operation. So what does it mean? x += 3 means x = x + 3. So, it means that if you add 3 to any value of x, you will get a new x.

(Refer slide time: 4:33)



Now, the value of x has become 8. So, I will minus it:

x -= 3

Suppose, and press Enter. Then the same value will come. So now, the value that we have obtained will come to the same value. I assigned it to x = 5, then we changed that value and assigned it a new value. By writing this:

x += 3

After that, I minused it, and it turned out to be the same. So, we can do it anytime.

Like I wrote:

x = 10

I defined it, and I wrote:

x -= 4, So what would it mean? It would mean: x = x - 4

Assign any value to x and subtract 4 from it. So, now I want to show its output. So, I will print it:

print(x)

So, the earlier value of x was 10. Four got minused from it.

So, the new value that has been assigned to x has become 6. Now, if I do x + 2, it will become 8. Why? Because the new value was 6. Now, if I added 2 to it, it turned out to be 8. So, we can define it like this.

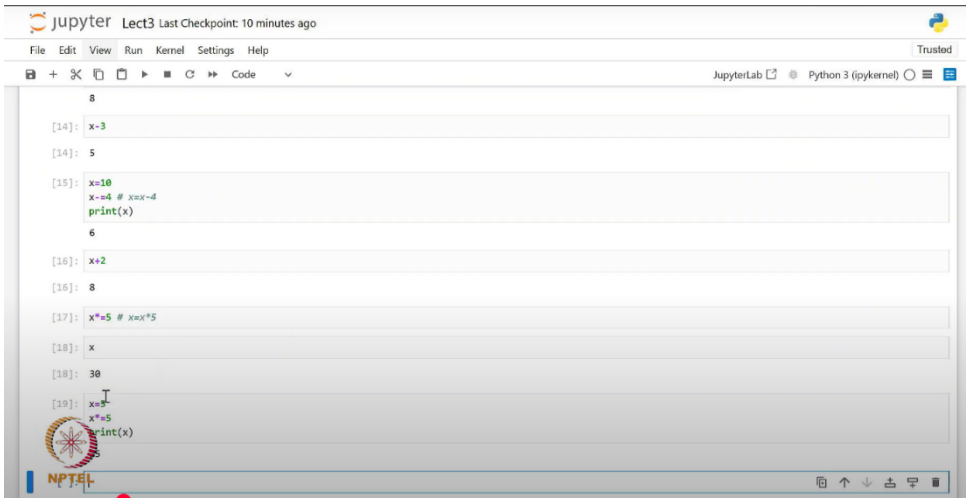
Now, I do addition and multiplication. Let me multiply and see what happens. So, we already have the value of x as 8.

I define it here: x* = 5. Here, so that I remember, I write that x = x * 5, it means this. Okay, I enter it. Now I write the value of x. What will x be? It will become 30. What does it mean? Earlier, the value of x was 6. I multiplied it by 5. It was okay. Here, I did x * 6, and 30 came out.

Now, suppose the value of x was 5 and I wrote x *= 5. I wrote 5 here also. Okay, and I printed this value — it came out. So, what did we do? We multiplied the value of x by 5.

Okay, so in the same way, the value of x was 6 earlier and I add 2 in that so it became 8. but x is only 6 . So, here I multiplied the value of x by 5. It came out to be 30. Here, I assigned the value of x to 5. Okay, this is an assignment. We are assigning it. So, what did we do? We multiplied it by 5. I assigned 5 to the value of x, then I changed the value of x with this command. So, this command told me what this command is showing. After that, I multiplied it and I got 25.

(Refer slide time: 8:05)



Now, what do I do? I put 8 in this x*=8 and see what comes out. So, 200 comes out. What did I do? I multiplied it by 8. So, 25 was the new value of x. See, the new value is because what is x? x means x * 5. So, we had multiplied some value earlier by 5, and the new value that came out will be saved here in x. Now, the value of x will change and will be the new value, and that is 25. I used the same value here, and I wrote here that x multiplied by 8. So, we got 200. So, the new value of x becomes 200. If I multiply x by 5, then the new value will be 200. If I divide it by 10, then the value 20 comes out. So, what happens here? The value becomes 20.

But the value of x is 200. I have performed an operation on this. So, the value of x, if I write x, it will show 200 only, because the value of x that has been assigned here is 200. Here, I just applied the operation on it and the result I got is 20. But the value of x is still 200, because it was assigned here. So, in the same way, we can assign any value by assigning it. So, I will write x = 40. Okay. Now, I will write x%=5. I will write this.

So, what does it mean to write % 5 over x? It means that x = x % 5. This means that if I print it here print(x) , then we will get the value of 0. Why? Because 40 is completely divisible by 5, and the remainder will come out to be 0. So, the value has come.

(Refer slide time: 11:02)

Now, what is the value of x? Zero here. Now, what do I do? I change the value of x and write it as x=x + 18. I wrote this. Now, what do I do? I assign a new value to x, whose value is 18. Now, when I write this to x, 18 comes out.

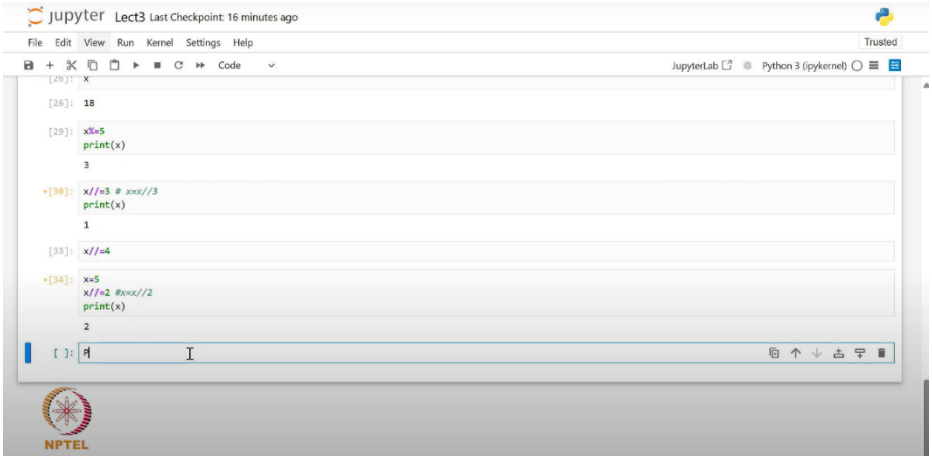
So, now after writing 18 to x, I write x%=5, so it will come from here. So, this value came. I did that to it. Now, what do I do? Let's print it here: print(x) — it came out as 3, because the new value was 18. I divided 18 by 5, and the remainder came out to be 3. So, this is the answer. From here, we get the remainder 3. So, like this, we can do any value; we can do division. Now, I write x //= 3 ,and here I write print(x), enter, then we get 1.

Okay. So, the x //= 3 means that x = x / 3. This means that any value x holds, I will divide it by 3, and we will get a new value of x. If I type its value, it will come to be 1. So, now we have x is 1. Now, what do I do? If I type x //= 4, let's see what comes out. It will be 0, because we have defined that x is a integer or floating value.

So, now I write x = 5, and after that I wrote x //= 2 , and printed it and we get 2. So, the value that we had, which was 5, we divided it by 2. So, I wrote it here. This means x = x / 2. So, the integer value came 2. From here, if we divide 5 by 2, we know that 2 point something will come out. So, the integer value will came out. It will give it to you 2.

Okay, so if we want to use any operator like multiplication or division, then we can assign a new value to x, which we have just assigned by using it. Okay, so we can do all these operations very easily. So, after this, we do some more Python conditions.

(Refer slide time: 14:25)



In Python, there are conditions — if statements. So, let's do Python conditions and if statements.

What did I do to this? I put a hash and marked it down, and then pressed enter. Then it appeared written: Python Condition and If Statement.

What does condition mean? Like what did we do — define. I defined x = 5. After that, I defined y = 10. Okay. Now, I used the if statement. So, I used the if statement — if y is greater than x, I wrote the statement and applied the condition. Okay. If y > x, and if that is true, then its value will go inside the block. Okay. And I will write print there. Here, I will define that y is greater than x. I have defined this, and it will be executed only when the condition we have applied is true.

```
x=5

y=10

if y > x:

    print('y is greater than x')
```

So, as soon as I entered it, Then, "y is greater than x" came out correctly. Okay.

If I change this, I will copy it and change it to x = 11, and press Enter — nothing came out. What does this mean? It was not entered inside the if block. Why? Because the condition failed; it became false. That is why it did not enter the if block.

Now, This print statement we have. I'll write this outside the block, and write, "x is greater than y". Okay. Now let me see what will happen. It will print "x is greater than y", and it won't enter the if block. So, we did this. In the same way, we can use other operators as well. For example, this is the greater-than sign. We can use other signs too — like the equal-to sign.

Let me take

```
x = 10

y=12
```

if y > x:

print(‘y is greater than x’)

After that, if I want to check whether x is greater than y or if x is equal to y, then we use the elif command — meaning, if the previous condition wasn’t true, check this one.

So, I used elif and wrote:

elif x == y:

print(‘x is equal to y’)

(Refer slide time: 19:18)

I added this and pressed Enter. It’s written y is greater than x and it is true

Now, I change the value of x to 10. Let’s see what happens. When I did it, it printed "x is equal to y". Why? Because the first condition was false, so it didn’t go inside the if. Then it went to the elif, and checked whether x == y, and this was true, so the statement inside was executed.

Now suppose x is greater. I change the value of x to something greater. Nothing gets printed. Why? Because if became false and elif also became false. In that case, both conditions are false, so it won’t enter either of the blocks. And the output is nothing.

To handle such a situation, we can use else. So, I wrote:

if y > x:

print(‘y is greater than x’)

elif x == y:

print(‘y is equal to x’)

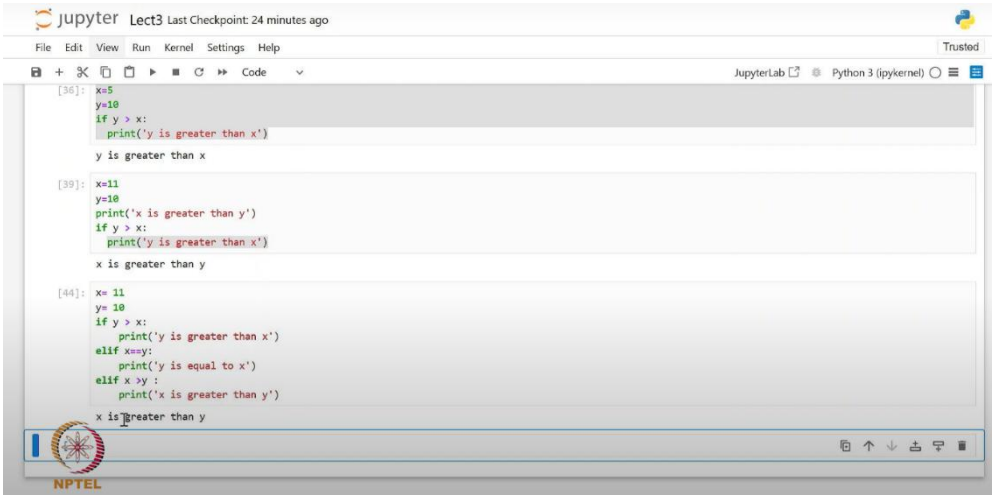
elif x>y :

print(‘x is greater than y’)

Now let's see what happens. It's written "x is greater than y ". So, what happens is,it goes to an if loop, its condition is false, so it will go into the elif block, and it was also false. So, it will go to the next elif and print the corresponding statement.

So from here, Python checks condition by condition — if if is false, it tries elif, and if that's also false, it goes to else.

(Refer slide time: 22:03)



Now, we can use elif multiple times. For example, I take:

a = 34

b = 30

Then I write:

if b > a:

print('b is greater than a')

elif a == b:

print('a is equal to b')

else:

print('b is less than a')

Here, the else part means: if neither if nor elif is satisfied, then this part will run. So in this example, it will print "b is less than a".

So, the values we had here — I had used elif. So elif means else if — go here, then go here, then go here. What happens in this is: if this is not true, and this is also not true, then it will have to go here. So what happens here is — your condition is not satisfied. For example, A didn't come here; it was never closed. This block was not closed, so we closed it. Now, it printed B, so the statement here became false, and this also became false, so it went here — and this is the else statement. So, I used if, elif, and else. Okay.

So, we use these values like this. If I don't want to use elif, what can I do? Now, for example, I took a= 34

b = 20

if b>a: print('b is greater than a')

else:

print(' a is greater than b')

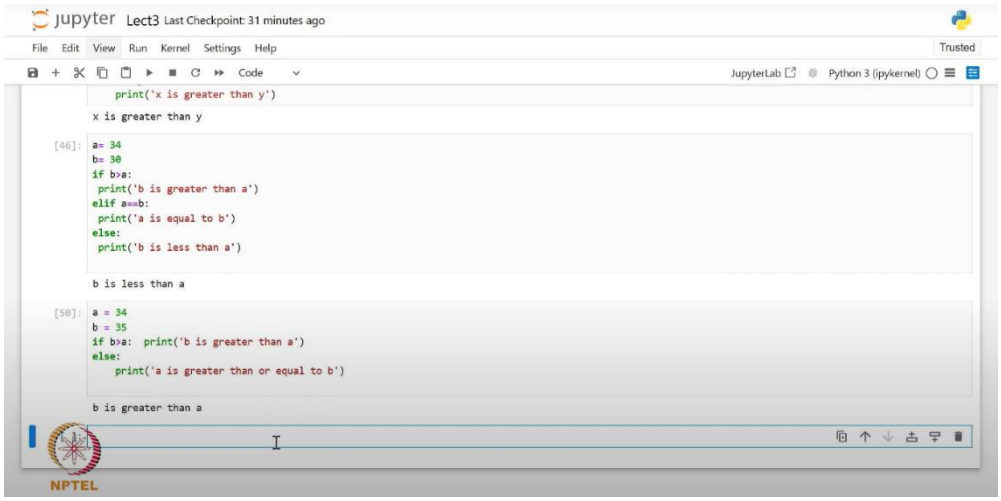
So in this case, we used else. else means: if the first condition is not true, then go here. If b is not greater than a, then it will come here and print the statement. Now, when I wrote this, the output was: "a is greater than b". But the problem is that if my a= 34 and b = 34, then let's see what happens — both are equal. So this is a false statement. If the condition fails, then it becomes false. Okay. But after the if, it will definitely go to else. It will either go here or go to else. It means if...else: either you go here and compile the command inside and implement it, or you go to else and implement that.

So, if I write: a is greater than or equal to b, then it will be absolutely correct. If this condition is written, then whether a is greater than or equal to b, the logic will hold. So in this case, the output comes correctly. This statement which we have written for a is absolutely correct.

So, when do we use elif and when do we use else in if statements? It is known that if is conditional — it will enter only if the condition is true. What happens in else is — if it becomes false, then it will definitely go to else. Okay.

Now, if I change the value — suppose I make b= 35 — then it will not go to else; it will go to the if block and execute that because it becomes true there. So I have written: b is greater than a. So we use conditions when we have any number and we need to compare it or do something based on it. Then we use this if, elif, else structure again and again.

(Refer slide time: 29:25)



Now, if you want to do a little more than that, then we can use else here too. So this is okay.

Now, we have talked about conditions — now I want to use AND..

You must have read about gates — in that, there are AND, OR, and NOT gates. So if we want to combine two conditions, then we use AND.

So I want to use AND. So, I write this to remember, this is a logical operator.

Logical operators are used when there are two conditions and we need to check that both the conditions are true — only then it should enter the block. Otherwise, the condition becomes false.

For example, I take:
a = 50
b = 30
c = 60

```
if a>b and c>a :
    print( 'both conditions are true')
```

, and I press Enter.

Now, this will happen only if both conditions are true. Is a > b? Yes — 50 is greater than 30, so that's true. Is c > z ? Yes — 60 is greater than 50, so that's also true. So if both these conditions become true, then the and statement means the whole condition becomes true. Then it will enter the block and the print statement will display the output.

Now, if I change it to 40 — okay, say a = 40 now — let's do it. Now, see — a > b is still true. But c > a is now 60 > 40, which is also true. So again, both are true, and the statement is executed.

But if a becomes 70, then a > b is true, but c > a becomes false (60 > 70 is false). So, as soon as even one condition fails, the AND block does not get executed, because AND requires both to be true. So like this, we can use and again and again. Similarly, there is also the OR operator, which is also a logical operator.

In the case of OR, even if one of the two statements becomes true, it will be considered the overall condition to be true

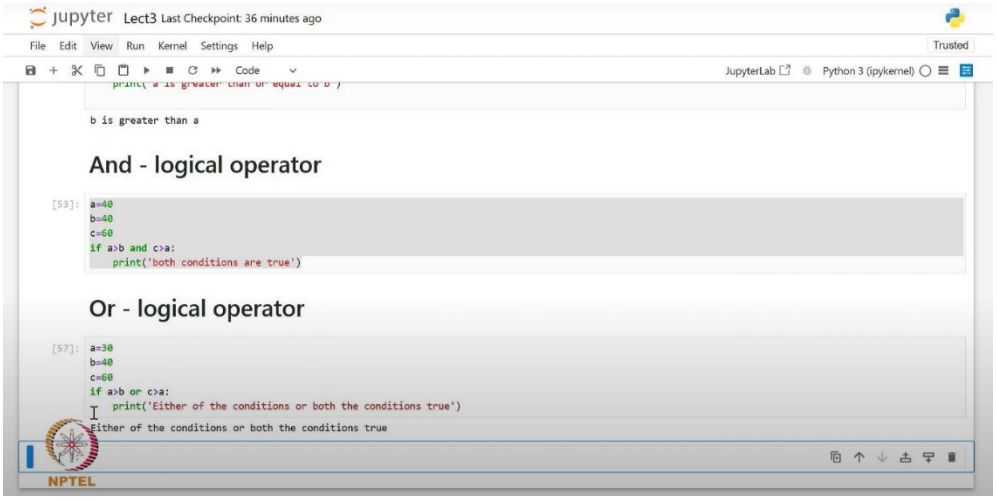
Like, if I had copied this, I copied it here, now I wrote it as both conditions. So let me change it — should I write as either of the conditions or both the conditions true. So if I press enter, then we can write it like this.

Now, a is 40. Okay, I will now write it. Let me make it 30. Now see, a > b is not true, but c > a is yes. So, if I write this and press enter, then this appears. And I have not written or here, I still have to write or.

```
a=30
b=40
c=60
if a>b or c>a:
    print( ' Either of the conditions or both the conditions true')
```

So what is the output? Either of the conditions or both the conditions true, and if one of the two conditions is true, then it will enter inside it. If both are satisfied, then there is no problem.

(Refer slide time: 34:00)



If I write here a < b, okay, a < b is true, c > a is true, then it will appear as written. If both are true, then it will also be true, but if even one of the two is true, then that value will become true, and the output inside will be displayed.

So in this way, we can use OR.

Similarly, we have Not equal to — we can also write it as Not equal to. So what did we do? The equal to statement is used — what is there in equal to? We have to put two equal signs, okay? Similarly, we have AND and OR. So what do we have to do in Not equal to? Not equal to is represented by this sign: !=.

So with this sign (!=), we will use Not equal to.

Now if I copy it here, I wrote Control + C, Control + V, and I gave the statement here: if a != b. Okay, this is written.

```
a=30

b=40

if a != b:

    print( ‘ a is not equal to b’)
```

So when I pressed enter, the output came: “a is not equal to b” because this is the sign — this one.

I will change it now, okay? So let me copy-paste this. I did Control + C. I wrote 30 here as well. Now when I hit enter, we did not get any display because a and b are equal. And when will it be true here? When a is not equal to b. That is why we do not have any output.

So the sign of not equal to will be used like this.

Now, if I want to do less than or equal to, so for less than or equal to, I do this

a=30

b=40

if a <= b:

print(‘ a is less than equal to b’)

Now, if a and b are there, then I have to write less than or equal to. So here it appears: less than or equal to.

So if a and b are equal, then it will be considered true. Either it is less or equal. So here, if it is equal, it will become true.

(Refer slide time: 37:44)

If I change the quantity, then what will happen? Now I have made a = 40. Now let's see what will happen. So, nothing has happened, right? a <= b is false. So, if the condition fails, then the statement will not be implemented. So here, no output will come.

Okay, so this less than or equal to has come. Like this, we write greater than or equal to. So, what will happen in greater than or equal to? Now, what did I do? I wrote less than or equal to. Now I want to take greater than or equal to, so this sign should come here (>=). Greater than or equal to has come. I wrote enter. Why did it come? Because a is automatically greater than b. So, here we will write that a is greater than or equal to b.

Now we have to write Not greater than or equal to, then what should we do for not? For that, we will have to write like this:

Now I want to write that if not a>b . What does it mean that the not which we were putting in not equal to? So we cannot put it like this in not equal to. I want to write: if a is not greater than b, then what should be written for this?

Okay, so if we do this:

a=20

b=30

if not a > b:

print (‘a is NOT greater than b’)

So if a is greater, right? And here if a is not greater than b, then what will happen? This statement will appear here. So, if I write it to be 20, then it is written as it is. What does it mean here? a is not greater than b. It means that if a is not greater than b, then it will be satisfied. So, if a is not greater than b, then b is greater.

In this case, it will be written here whatever condition becomes true and the output inside it will be displayed, right?

(Refer slide time: 40:58)

So we will put it like this: if we put it with the exclamation sign, then in that case you will get an error, okay? So we have used the logical operators that we had — how can we use them? How can we use them in if statements?

So now, let's see a little more. I write nested if. Okay, what does nested mean? If an if statement is inside another if statement, then what will be done? I write this and mark down here, and then — so what does nested mean? Inside it.

Now, I wrote x = 20. I wrote if x > 20 or let me write > 15: here. Let me write what should be written: print "x is greater than 15" will be printed if this happens; only then it will go inside it, okay? Now I write if x > 25, I write colon, then I wrote print "x is greater than 25". So, what did we do? We put the if statement inside another if. Okay, this is called nested .

I wrote else print "Not above 15 or Not above 25", I write this, okay?

x = 20

if x>15:

print (‘x is greater than 15’)

if x> 25:

print(‘ x is greater than 25’)

else:

print(‘Not above 25’)

So now let's see, if I run this, what comes out on running it? It was written that "x is greater than 15 and Not above 25".

So now let's see what happens. It was 20, so it became true here; so x is greater than 15, so 20 came. So here it is written "x is greater than 15".After it went here, this command will go here and it wrote that "x is greater than 25", so this will not go. If it is false, the else and the else of this if will go here and "not now 25" will be written. So first it went here, then here — this if, this is our nest. If we look at it, we will write it is nested, okay?

(Refer slide time: 44:33)

So it is the nested if, and the else is of this if, right? And if I write else: print(‘Not above 15’) , I will write this. Now let's see what happens — invalid syntax has come, okay?

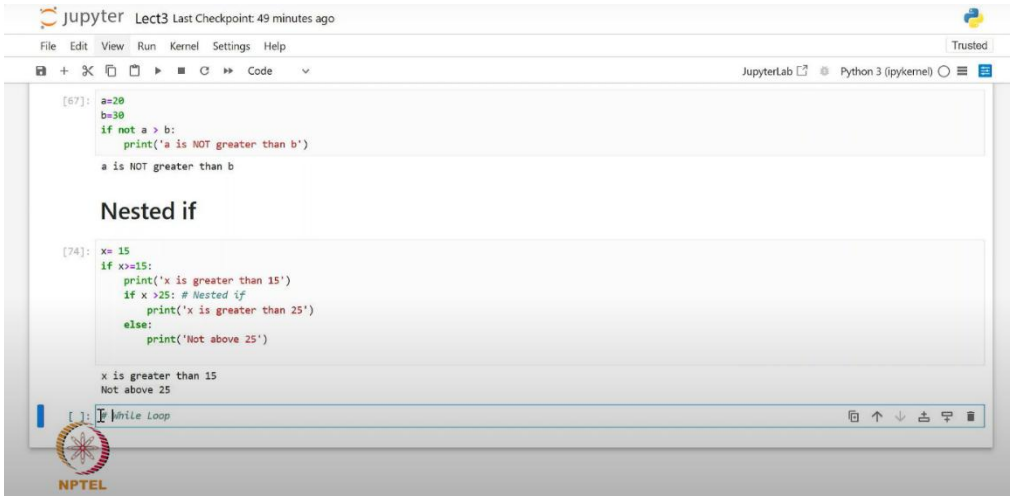
So here, the syntax is wrong. Two elses have come, so let's take it till here. What happened here? There are two else statements. So if the else statement was this, then it will not work, okay? So, we have used it in this way.

Now what do I do? Let's change x. x is greater than 15, so I write 10. Now let's see what happens.

Now see, it will not go here. So, when any statement does not go here, then the if statement which is inside it — means it is made inside it — then when it does not go in it, it will not go till here. If it does not go till here, then there will be no value, okay? So here, if this value is true, then only it will enter inside it.

Now I write here x is greater than or equal to — let's write this. Now what did I do? I changed it to greater than or equal to. So as soon as I do this, it becomes true. After that, I will press enter; it will print. And if it becomes false, okay, then it will go to else and will print true here.

(Refer slide time: 47:03)



So all these things — we can use nesting like this.

Similarly, if you remember, we have a While loop.

While loop means this will keep going on until we stop it. That is what we call a loop. Instead of writing this statement again and again, we use looping.

So we will use looping like this:

I defined an index; I wrote i=1.

And I wrote while i < =5, then print it. Let's print i, and i will get incremented by one, and this is a colon.

i=1

while i<=5:

print(i)

i+=1

So what is happening here is that we will go inside the loop, and whenever the condition i < 5 becomes true, it will go inside, it will print its value, and it will give an increment.So this has to be incremented as well, and we have to do indenting for it. And I pressed enter.

We get output. Like I started with i = 1. As soon as one went here, then 1 is less than 5, yes. So, it print i and added one in it. What does it mean? we have just done is i = i + 1.

So one got incremented in i. So, I become 2, then it will go back there.

So what is in the While loop? It will go on like this until the condition of the while does not become false. As soon as it reaches 5. So, i < 5 is true, so it went inside it, okay? What will happen after that?

If it reaches six, what will happen? Six — it will become false here, then it will stop. Okay, so here this condition has come.

If I change this a little bit, write Control + C, Control + I, then what will we do here? I just write less than. So see, it will come till four. Why? Because the i < 5 will become false and what will happen here? It will become false, okay? So, it will be 1 2 3 4 only.

(Refer slide time: 49:30)

Let's write i = 0. i is indexing, so I will start it from zero. So, I started indexing. In Python, indexing starts from zero, like in MATLAB, everything used to start from one.

I write i < 6. Don't forget to put a colon. Here I wrote. After that, I write i += 1. Okay, so what did I do? I first gave the increment and after that I do print i.

Now let's see what happens in this case. This is written . Starting with zero, so zero is less than 6 , yes. What happened here? The i which was earlier zero became one. So as soon as one was formed, it printed one, then it became two, three, four, five, six, okay, right?

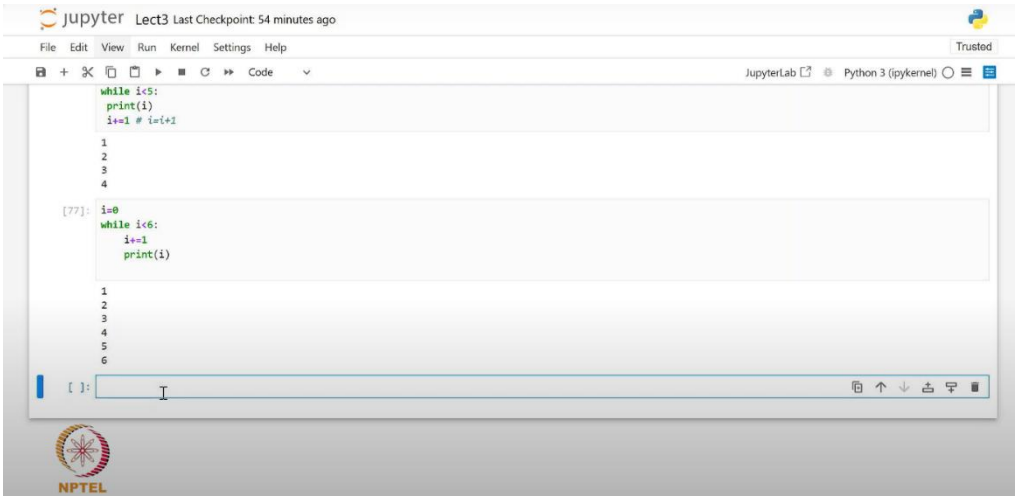
And if we look at this, here 5 became 5 < 6, yes, true. Here it became i=6 and it printed six. So if you see, in this one i started from zero; here it started from one, but what we had printed, we first printed it and then incremented it.

What did we do in this case? I incremented it. If I did it first and printed it later, okay? So the increment will happen first. So if zero went in, then zero is less than 6, yes. So, here it made zero into one and the increment got printed.

Now, like 5 goes inside, 5 is less than 6, it created six inside it and printed it. So, it depends on where we are writing the print value and where we write the increment.

So the while loop will keep running as long as its condition is true.

(Refer slide time: 51:54)



That's why sometimes what happens is that we have to stop the while loops. So, to stop it, what do we do? We put a condition for it.

A break statement comes.

So, like I wrote i = 0, start from zero. And I wrote while i < 6. I wrote this. After that, I printed it. So, here I printed it .Now, what did I do? If i == 4, I put the condition. So, as soon as that happens, I break. Okay, and here I wrote i += 1. So now, let's see what happens.

I=0

While i < 6:

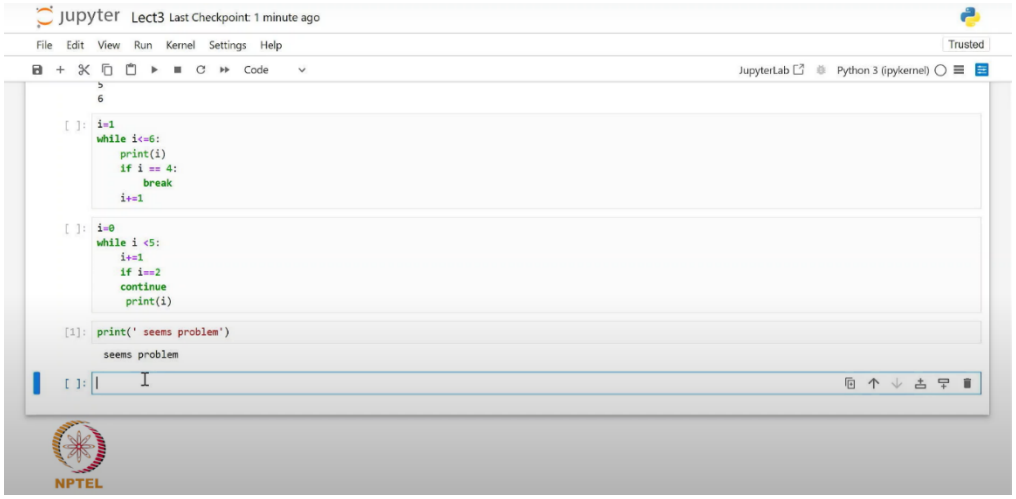
print(i)

if i==4:

break

i+=1

(Refer slide time: 52:54)



Now after this, our while loop is done. So now, what will we do? We will use for statement. To use the for statement, let's see how we can use it, okay?

For looping.

Now, like I took it, I defined a set defined set1. Okay, in that, I defined a listing. So I did listing. I did ‘mango’, okay, comma. Then I did ‘banana’, comma. And after that, after banana, I did ‘apple’. This we defined. "Listing" means I defined a vector.

Okay, so after that, what do I do here? Then I write for x in set1: like this I defined. So, for x in set1 means what to do when x is in set1. Okay, so after that, I wrote set1, and after that, I put a colon and then printed x.

```
set1 = [‘mango’ , ‘banana’ , ‘apple’ ]

for x in set1

   print(x)
```

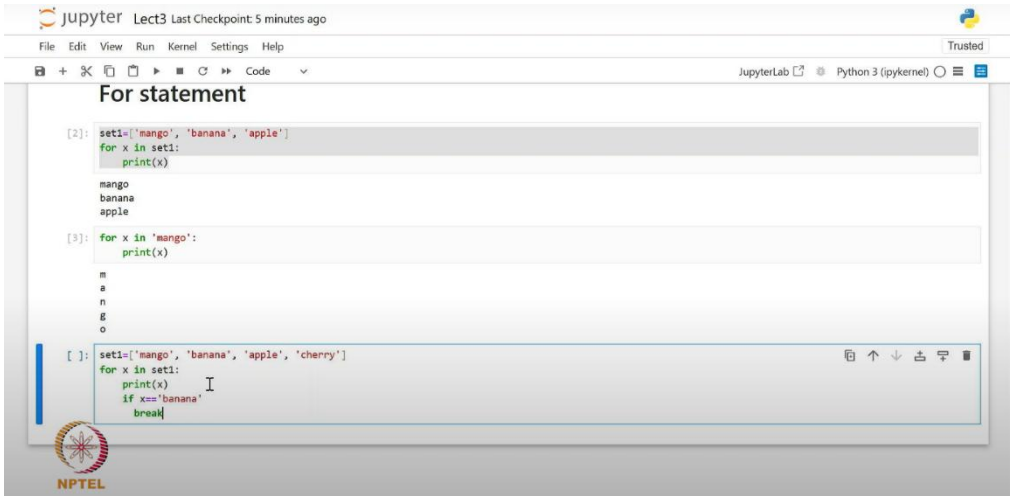
Now let's see what will happen here. So it is written mango, banana, apple. Okay, and what is happening in this? That set1 is a list. Now what will we do? I define x in set1 . So, as soon as x goes into the set, what will x in set1 do now? It will start printing the values that are in the listing.

Okay, it will keep printing those values, and it will keep doing this until it reaches the last element of the vector or the listing. So the last element was "apple", so it will print till "apple" and then it will end. Okay, so this loop — we call it a for loop. Okay, so a for loop is used when we have the quantities that are given in the listing.

Now, like this, I have defined for x in ‘mango’. Okay, I have defined this, and after that, I have printed x. So what will it do? It will put x in "mango", which means it will look for the characters and print them. I have written enter. "Mango", okay, so the for loop will enter our string in this character, and the values that are inside it, it will type them one by one here, and print it.

Now like this, I have set1. Now suppose I change it a little bit . Now I wrote mango, banana, apple, and here I wrote cherry. "cherry." Now what did I do?
for x in set1: print(x)

(Refer slide time: 57:10)



After that, what do I do in this applying if statement? I put if x == 'banana', so as soon as it does this, it will break it, okay? So as soon as that x is "banana", it will stop. I entered colon, yes it came. Okay, we will put colon here and enter. So see, first I wrote "mango", so mango came. Then I wrote "banana". Okay, and as soon as x is printed, and where this statement x == banana, so it means this statement is true. Then enter will be in the for loop and as soon as enter will break it.

set1 = [‘mango’ , ‘banana’ , ‘apple’]

for x in set1

print(x)

if x== ‘banana’

break

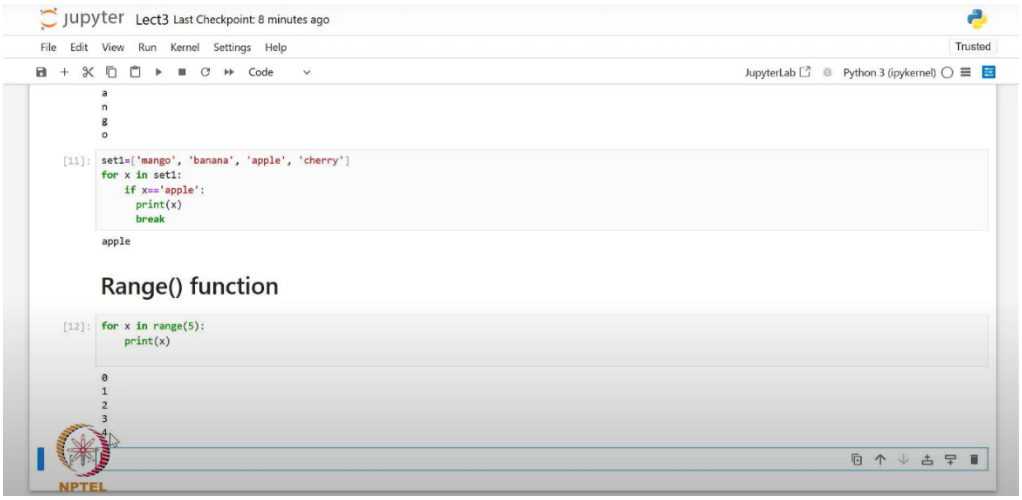
So this banana will be printed. I write here apple, it will be written till apple l. So like this, we can stop whatever we want to print. We can get it printed. If suppose I want to get "apple" printed, and I don't want to print anything else, so I will do this: remove print from here, control x, and write it here. Now what I have to do is ,I have to print only "apple". I pressed enter, it came.

Okay, so what did we do? This was in set1. Now we made x continue in set1, and I gave this statement that if x == 'apple' then only it will print, otherwise it will not. So as soon as it went to "apple", it printed and after that I broke and it stopped. "Break" means that just do not implement further; the for loop has to stop here.

Okay, so now I am using it, and like when I used it earlier, every word use was written — mango, banana — and after banana, it stopped. In this only apple will be used . Okay, so we can use this thing.

Like this, I can use range — range function, okay. And what is the range function? Now let's see what the range function will do for us. Now see what I have to do: for x in range(5): I wrote this.

(Refer slide time: 1:00:13)



Okay, so what is the range function? It returns a sequence of numbers starting from zero. It will start from there by default and will increment by one. So if I wrote x in range(5) and suppose after that I write a colon here and I printed it.

So what is written is print(x). So x is in range, in range means from zero to 5. Alright, it will keep printing 0 1 2 3 4. So it will exclude the last one, i.e., five. So what is happening is that x is getting printed and is incrementing automatically.

Last time whatever we did in the if loop or while loop, what were we doing in the while loop? We were printing this and were incrementing here. And in the starting, we had defined that i is starting from zero. Like we did above that i is starting from one. Like it was here, i is starting from zero while it is this. Now we gave the increment and printed it. So it got printed from here.

Now we did not do this work. What we did was that we defined for x in range(5), and the range starts from zero and up to 5 so exclude the last element. Okay, and the increment will keep happening by one and it will be printed. So first it printed 0, then 1, then 2, 3, 4. Okay, so the range of the five elements was there, which it displayed.

Now I write for x in range(2, 6). I wrote this. Okay, so what am I doing in this? The range that I am taking is starting from 2, and I printed x and press enter and then 2 3 4 5 came. Starting from 2 and will go up to 5. So, in this case, what did we get? That the values have to start from 2 and we printed it.

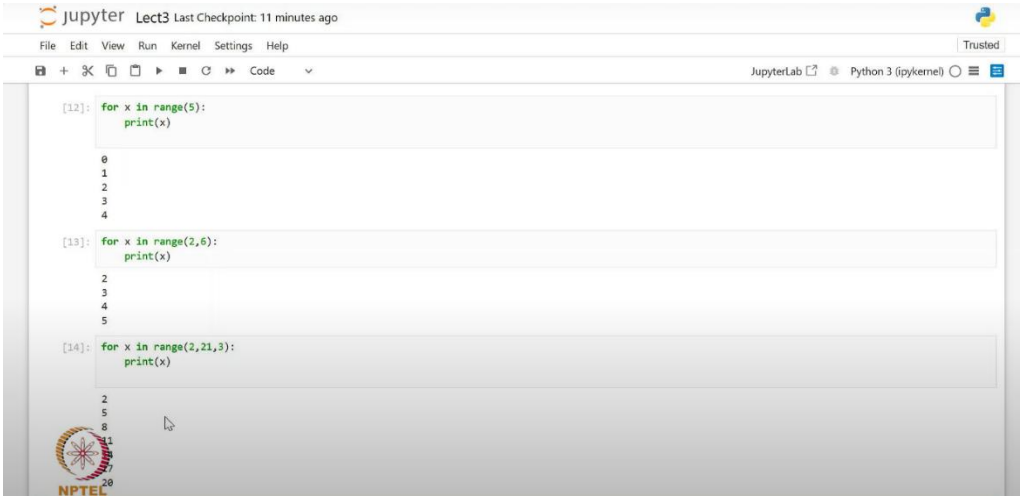
Now I suppose the increment, which was here was automatically 1 , I want to change the increment as well. So I wrote for x in range. Now what did I do? I started from, suppose, 2 and I have to go up to 21 and take an increment of 3. So,I printed this.

```
for x in range(2,21,3):
    print(x)
```

What will it do? It will start from 2, 2+3= 5 then 8 plus 3 like this, I will print it. So this has come, see — 2 has come, then 5 has come, then 8 has come, then 11 has come, then 14 has come, then 17 has come, then 20 has come. Okay. So it will print till the last value.

So what did we do in this? We took increment in three . By default, our increment is of one, like we wrote here. Otherwise, if we want to take different increments, then we can also write it like this, okay? So we wrote it in this form.

(Refer slide time: 1:03:03)



Now like I am taking this, for x in range. Now sometimes what has to be done is that our value starts from zero. We have to run a program, we have to do 20 iterations, okay, and we have to see the iterations after 3. So, what do we do? We print it and printed it, pressed enter.

```
for x in range(0,20,3):
    print(x)
```

So, yeah, okay, it starts from zero in this, then zero again, then six, then nine, then twelve, then fifteen, then eighteen. So it will print it till here.

So we will use this type of command a lot in the future when we will do coding for the methods of scientific computing. So like this, we can define a range.

And now as if I write this, I have defined set1. I have taken any listing, okay. I have taken the listing, I have defined it in it. I have defined the color "blue", then I have defined "red", then I have defined "green", okay. I have taken this like this.

I have taken set2. I have defined it, and in two, I have taken a combination — some "apple" like this. I have defined something, no logical "mango" and "banana". I have defined this, okay.

Now what do I do?
for x in set1 : I wrote that. Now, what did I do inside this? I have taken another for loop — for y in set2. I have defined it, and after that, I have written print(x,y). It will print.

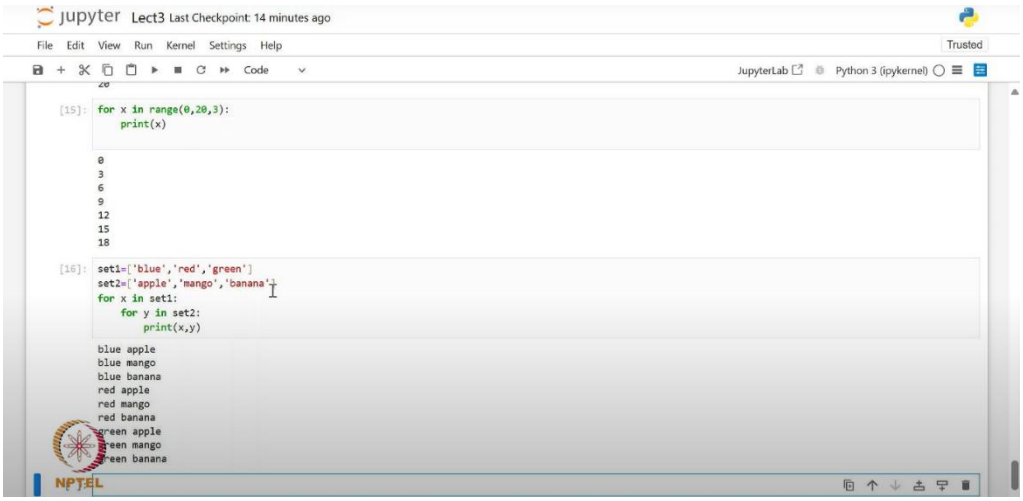
```
set1= [‘blue’, ’red’, ‘green’]

set2= [‘apple’, ‘mango’, ‘banana’]

for x in set1:
    for y in set2:
```

print(x,y)

Now, what will it do to us? What are we doing in this case? We used the for loop and then we also made a nested for loop. Just like we made a nested if loop, we made a nested for loop and then we printed it.

So, if I do this, let's see what comes out. It comes out. So what did I do in this? We will make "blue apple", "blue mango", "blue banana" element-wise. Then we will make "red apple", "red mango", "red banana". We will make "green apple", "green mango", "green banana".

(Refer slide time: 1:05:48)



So we will type all of them by writing print here. Similarly, we will call it a nested for loop. Just like we did a nested if loop, we did a nested for loop in the same way. So, we can use it like this.

Till here, we have used the if. So, the commands that we did today, that is, the if statement, range, and for, are the statements that we did today. So, these are very important statements, and they will be used a lot in the future. And in the upcoming lectures, we will use more statements and more Python commands than this. And after that, we will start scientific computing and will use it to write down the codes in the form of Python.

Thank you guys for watching this lecture. Namaste.