Advanced Business Decision Support Systems

Professor Deepu Philip
Department of Industrial Engineering and Management Engineering
Indian Institute of Technology, Kanpur
Professor Amandeep Singh
Imagineering Laboratory ac
Dr. Prabal Pratap Singh
Indian Institute of Technology, Kanpur

Lecture 30

Control Flow Statements & List Comprehensions

Hi everyone, welcome you all to the lecture series on Advanced Business Decision Support Systems.

Control Flow Statements & List Comprehensions

Advanced Business Decision Support Systems

Dr. Prabal Pratap Singh

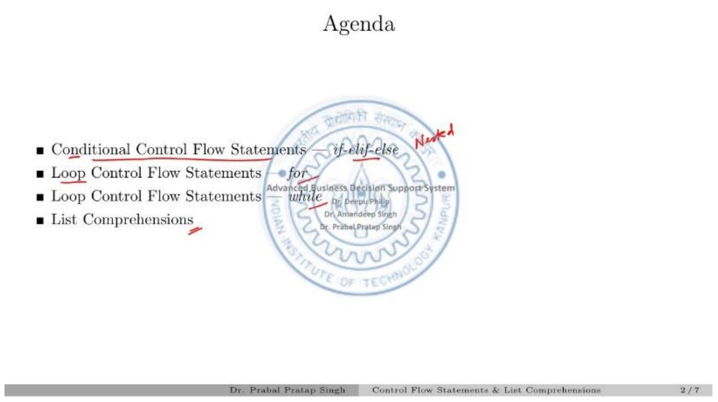
Indian Institute of Technology Kanpur

- Variable
- Statement
- Comment
- Data Types
- Numbers
- Strings
- List
- Slicing
- List Method

Dr. Prabal Pratap Singh | Control Flow Statements & List Comprehensions | 1 / 7

So, I am Prabal Pratap Singh and I am from IIT Kanpur, and we are discussing the Python programming languages and until now, we have covered Variables, how to initialize them and how to use them, then different kind of Statements, Comments, Single line comment, Multi line comment and different kinds of data types like Numbers, Strings, List and some operations like Slicing on these Sequential data types and methods on List. So, today before moving on to other different kinds of data types, we need to first see how can we manipulate the control flow in the program.

So, until now, we have seen that, we are writing the commands and the interpreter is executing all these commands line by line, but we can also skip the line, we can also skip the complete block of code by making some decisions on that code. So, these types of statements are called 'control flow' statements.



Agenda

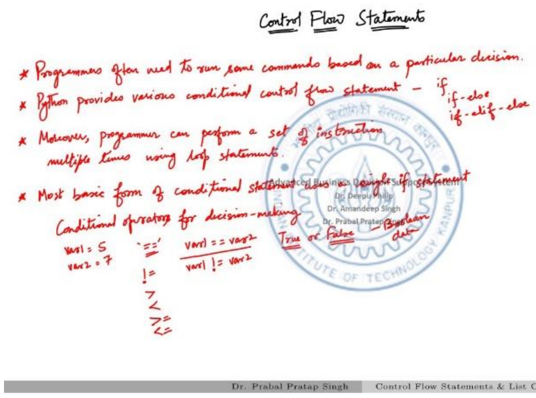
- Conditional Control Flow Statements — *if-elif-else* Nested
- Loop Control Flow Statements — *for*
- Loop Control Flow Statements — *while*
- List Comprehensions

Dr. Prabal Pratap Singh | Control Flow Statements & List Comprehensions | 2 / 7

So, the agenda for today is we will first see, what are these 'control flow' statements, the Conditional form of them like 'if', 'elif' and else and we will also see the nested version of these commands.

After that, we will move to other types of 'control flow' statements that are Loops. In these kinds of statements, what happens is, interpreter will keep on executing the same lines of code until and unless the programmer asks them to move to the other block of code. So, these kinds of statements come under 'for Loop'. After that, another Loop control statement is 'while'.

So, we will see, what are the differences between these two different kinds of statements and then using all these new information, we will see how can we use List Comprehension which is a beautiful way of creating a list which we will see in our earlier lecture.



Control Flow Statements

- Programmers often need to run some commands based on a particular decision.
- Python provides various conditional control flow statement – if, if-else, if-elif-else
- Moreover, programmer can perform a set of instruction multiple times using loop statements.
- Most basic form of conditional statement uses a single if statement

Conditional operators for decision-making

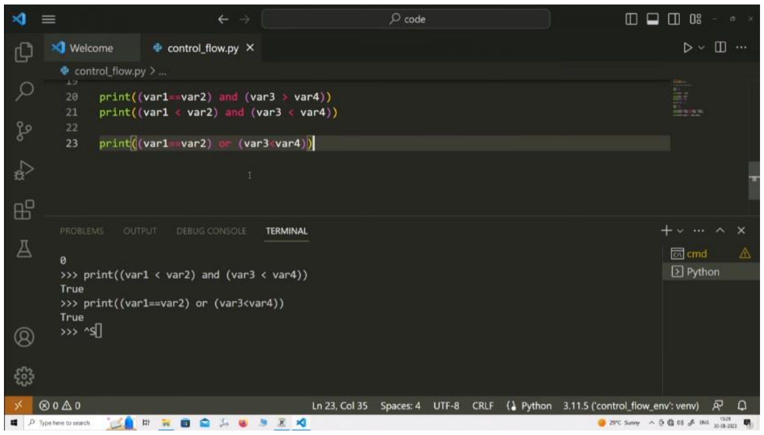
var1 = 5
var2 = 7

'=='
!=
>
<
>=
<=

var1 == var2
var1 != var2

True or False – Boolean data

Dr. Prabal Pratap Singh | Control Flow Statements & List Comprehensions | 3 / 7



```python
print((var1==var2) and (var3 > var4))
print((var1 < var2) and (var3 < var4))

print((var1==var2) or (var3<var4))
```

```
0
>>> print((var1 < var2) and (var3 < var4))
True
>>> print((var1==var2) or (var3<var4))
True
>>> ^S
```

So, moving on to the next. So, let us see what are 'control flow' statements. So, before looking at the python implementation of it, let us first understand what are these Statements. So, in real world example, let us say, your mother asks you to buy something from the market, let us say, vegetable.

So, if she just asks you to buy some vegetable, then you can buy anything from the market, but if she asks you to, buy a particular vegetable until unless it is below 70 per kg, otherwise come home. So, this contains a particular kind of decision, you have to go to the market, see for yourself what the price of the vegetable is, and then, either you will get the vegetable or you will come back without buying the vegetable. So, this is how in real life we make decisions.

Similarly, Python as a programming language can also make decisions and programmers often need to take these kinds of decisions. So, that is why, in Python, programmers often need to run some commands based on a particular decision.

So, for these things to happen, Python provides various Conditional 'control flow' statements. Various versions of these like for example, only 'if' statements or 'if-else' statements or if you have different multiple decisions to take on, then you can use ‘if’, 'elif', it is nothing, but else if and the final else clause. So, these are the various variations in the ‘if-else’ statement available in python.

Moreover, programmers can perform a set of instructions multiple times using Multiple Loop statements. So, for example, kids in their early stages of schooling, they used to learn tables.

So, what if a teacher ask a student to tell a particular number of tables till 11 number. So, he or she will keep on saying the particular number times this changing number. So, here what he or she is doing is, he or she keeps repeating the same thing and with different kinds of computations. So, the same thing can be done with python as well. So, for that we need to use the 'for Loop' statements.

Most basic form of conditional statements uses a single 'if' statement. So, before using all these statements, we need to make decisions in python. Now, to make decisions, we need some conditional operators which we have seen earlier. So, let us see in detail, what are these conditional operators for decision-making. So, a very basic thing can be to check whether two things are equal or not in python.

So, let us say, we hold a value in variable 1 as 5 and variable 2 as 7. So, we can check whether these two variables hold equal value or not by using the equal to, equal to, operator. So, this is a conditional operator, what it will do is, it will check whether the container on the left and the container on the right have the same value or not. So, we can write this as var 1, equal to, equal to, var 2. So, the output of this statement can only be either true or false, and these values are predefined in python as a keyword and these are known as Boolean data.

So, either it can be equal or it cannot be equal. So, either the answer will be true or false. Similarly, we have other operators as well like, not equal to. So, if we ask where 1 is not equal to var 2, then this will be true. The first one will be false because 5 is not equal to 7.

Similarly, there are other operators like greater than, less than, greater than equal to, <=. So, these variables we can use for our computation purposes. So, let us see all these operators first in our Visual Studio code.

So, I have created a directory named code and I am opening the Visual Studio Code in that. Here is our Visual Studio Code.

Now, I am creating a new file named control _ flow dot py. So, this is a python file. So, this is the python and it is asking me for the location. So, this is the correct location. Let us create a file here.

Now, you can see, this is the file. So, let us first create our new environment. So, now, you can see that we have created a new environment named ‘control flow’ env. We can activate this environment by typing ‘control flow’ env. So, this is our environment.

So, this is the 'scripts' folder, and then activated. This is 'scripts'. So, here we have activated our environment. So, we can close this for the time being and let us now start writing our code. First of all, let me import a new module here today and I will show why I am calling it.

This is an OS module which comes as a standard library and it has a very useful system call called OS dot system and we can write CLS. and run this line. So, it has run import OS. Now, we will run this line. So, now we can clear our terminal by keep on running this line as many times as we want.

So, we will keep on coming to this line during our code today. So, let us start with the conditional operators. So, I am initializing my first variable as 5 and second variable as 7. Now, I can check whether these two things are equal or not by using == operator and it will print the output which will be either true or false. So, save this and shift enter.

So, it is saying, where one is not defined. Why? Because I have not yet run these two commands. So, since it is not in the memory. So, python interpreter is asking to first define this. So, let us run this and this.

Now, it is showing that these two things are not equal. Now, we can again clear our screen and we can check other things as well, like where one is not defined. So, let us run this is not <var 2. So, it is true that 5 is <7. Now, one more thing, I want to show you if the true keyword = 1 or not.

So, this is a Boolean data type and this is an integer data type. Now, let us see, whether it is equal or not. So, the interpreter is saying that these two things are equal. Why? Because the developers of python have created these things in a way that truly represents the value 1. So, false should represent value 0.

Let us see. So, this is also correct. Now, anything that you are checking with false, it is actually getting a value 0 and it has been stored as a Boolean type in a false keyword. So, let us also see whether anything other than 0 can be interpreted as true or not in python. So, for that to check, we will write true equals, let us use an integer like 7. So, this is false, that means, the interpreter will only interpret 1 as true. So, this is how we can use our conditional operators.

Now, there is one more thing like until now, we have seen that we can perform decision-making on two variables, but what if there are multiple decisions to make in a single statement? So, for that, we need some kind of conjunction operator. So, in Python, there are two operators that are available and that are ‘and’ and ‘or’. So, let us first create two more variables, var 3 equals 9 and var 4 equals 11.

So, we have already seen how to make a single decision like where 1 equals where 2 right. Now, what if we want to check one more condition here? So, we can write and var 3 greater than var 4. So, what this statement is doing is, it will print the output of the decision that will be created by this statement. So, the decision will be either true or false only.

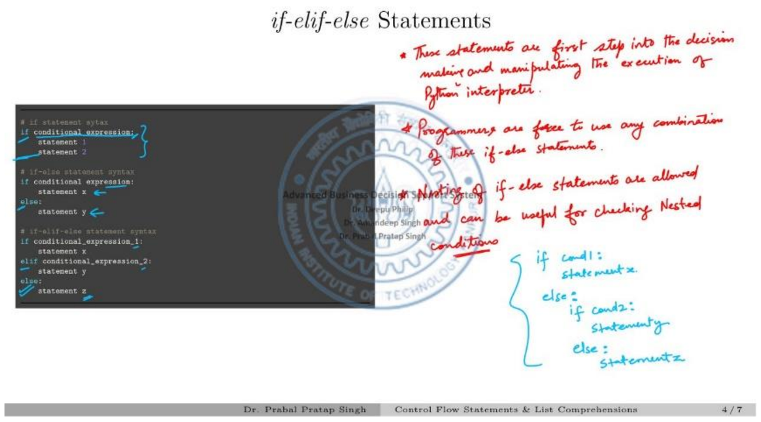
So, 'print' statement will either write true or false, but the first expression will be true, if these two variables hold equal value which is not the case because here, variable 1 is 5, variable 2 is 7. Similarly, variable 3 holds value 9 and variable 4 holds value 11. So, this expression will check whether variable 3 is greater than 4 or not, if it is not, then it will write false. So, the first decision will be true, the second will be false and from our basic knowledge, we know that AND will report the final decision to be true, if both of these are true.

So, since both of these are not true. So, it will print false. So, let us first initialize this variable in our interpreter variable 3, variable 4. Now, I am running this. So, it is saying false. If I just copy this statement and change this conditional operator to less, then the answer should be true. Since, our variable 1 holds value 5 and variable 2 holds value 7.

So, these are not equal. So, variable 1 is not equal to 2. So, we can check whether variable 1 is <2 or not. Now, this will be true and the next is also true. So, this should print true. Now, both hold the true value that is why the 'print' statement is getting the 'true' as answer.

So, these are the different kinds of conditional operators and we can also see the OR operator as well which will give the decision as false only if both are false, otherwise, if any of the expressions is true, then it will give true. So, we can write var 1 = var 2 whether they are equal or not or var 3 is <var 4. So, here var 3 is <4 is the correct statement. So, this will output true, but the first one will not output true because these are two different values, but since, we are using OR. So, if any one of these is true, then the whole statement becomes true.

So, this should print also true. So, this way you can use and you can add any number of decision expressions here to create this. So, let us move forward.



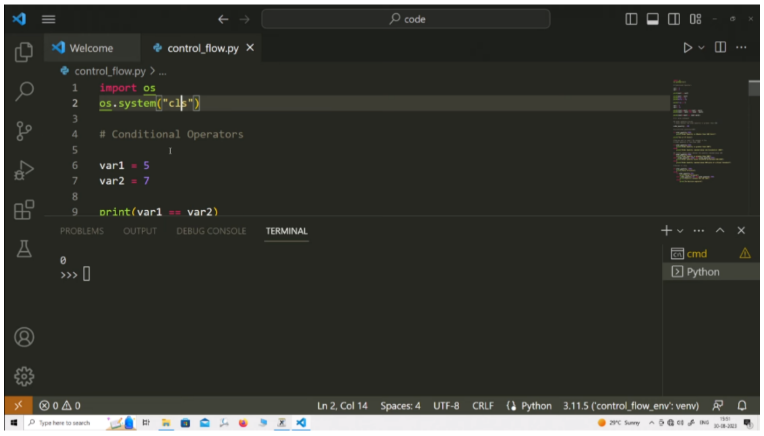
if-elif-else Statements

```
# if statement sytax
if conditional_expression:
    statement 1
    statement 2

# if-else statement syntax
if conditional expression:
    statement x
else:
    statement y

# if-elif-else statement syntax
if conditional_expression_1:
    statement x
elif conditional_expression_2:
    statement y
else:
    statement z
```

- These statements are first step into the decision making and manipulating the execution of Python interpreter.
- Programmers are force to use any combination of these if-else statements.
- Nesting of if-else statements are allowed and can be useful for checking Nested conditions

```
if cond1:
    statement x.
else:
    if cond2:
        statement y
    else:
        statement z
```

Dr. Prabal Pratap Singh Control Flow Statements & List Comprehensions 4 / 7



```python
import os
os.system("cls")

# Conditional Operators

var1 = 5
var2 = 7

print(var1 == var2)
```

```
0
>>> 
```

So, now let us start looking into the if and else statements in python. So, these statements are the first step into the decision-making and manipulating the execution of python interpreter and programmers are free to use any combination of these 'if-else' statements.

So, we can write any number of 'if-else' statements in our code until unless, we have a limit on the code base. So, let us see this basic syntax of 'if' statement. So, this is a predefined keyword if and syntax highlighting will always give you a different color for this word. After that, you can write your conditional expression which we have seen in our earlier slide, how to write a conditional expression and how it will get evaluated to either true or false.

So, if just after writing if a conditional statement should give either true or false as a output and then if the output is true then, the next lines will get executed otherwise, it will not get executed. So, after computing your conditional expression, you have to use these columns and then create a new line and now you can see the white space indentation here. So, these are blank things and these are four space indentation here we are using. So, python do not use curly braces or other things like in C language or other languages where, the punctuation marks will tell you whether a new block of code started or not. Here, python will use the white space indentation as we have already discussed.

So, this is the first step where we are looking the indentation in python. So, this is a complete block of code and here after using indentation, we will write n number of statements we want. So, we are just writing two statements and if this condition becomes true then these statements will get executed. Similarly, we can use else also as a keyword here and what difference it will make is, if the programmer wants to run some commands based on this conditional expression, then these statement x will execute otherwise, for all other cases statement y will get executed here. In the third case, what if we have different types of conditional expression like conditional expression 1, 2 or n number of these statements.

So, we can use 'elif' this is a short form of 'elif'. So, if this statement 'elif' this statement, then we will run statement y and after all these expressions if the interpreter does not find any conditional expression to be true, then run this as statement that right and also nesting of 'if-else' statements are allowed and can be useful for checking nested conditions. So, nested block is nothing but there is an 'if' statement conditional expression 1 colon statement x else. Now, in this else, we can write another 'if' statement condition 2 statement y else statement z. So, this can be an example of a nested if-else statement.

So, we can see the examples on our VS code. So, let us write a short program for creating order quantity tracker. So, what it will do is, it will do a very basic task to check whether the program the order quantity is greater than 1000. So, to do this, we first need to capture this order quantity in a variable. So, let us now create a new variable order quantity equals and we are initializing it with 1100 value. Now, in advanced examples, you can take the input from the user for this order quantity variable.

So, first now, let us see the implementation of single conditional operators. So, order quantity greater than 1000. So, this will just write order quantity is not equal. So, let us run this order quantity = now this is printed. So, this is nothing true because order quantity is greater than 1000.

Now, this same conditional expression, we can use control and forward slash to make a active statement as a commented line. So, I did this. So, again I can toggle this comment to an active line or active line to a commented line. Now, we will write our first if expression and we will use the same thing, order quantity is greater than 1000. Now, you can see here that, just we are using this expression into the 'if' statement, this is the conditional expression.

Now, in a new line, the VS code will automatically indent your lines here because the code writer knows that you are writing a if block here. If you are using notepad then, this will not happen and you have to press space four times or use a tab character. So, these are the things that will help when you are using a dedicated code editor for your development environment. So, now let us print here, order quantity is greater than 1000 units.

So, let us run this 'if' statement by using shift enter. So, here you can see that first interpreter read that block of code and then printed this because since our order quantity is greater than 1000. So, it printed this if we change it to 900 and then, again override this variable and then again, run this 'if' statement, nothing will get printed. Why? Because our conditional expression reported false. So, if will only run this statement when it will the conditional expression report a true value. Since, there is nothing else to do.

So, it stayed in the same state nothing gets printed here. Let us see where whether we are doing it right by writing a simple 'print' statement out of if block. So, why I am writing this, we are confirming that whether our code actually code gets executed here and then moved and reached to this 'print' statement or not. Since, our order quantity is 900. So, this print will not work because of this 'if' statement, but our execution should reach to this point. So, let us again run this code block shift enter and now you can see that, after executing this 'if' statement nothing got printed and then the interpreter reached to this 'print' statement and printed out of if block.

So, our code is working fine. Now, enhance our order quantity tracker by adding an else clause. So, now we want to report the manager if the order quantity is <1000. So, our code should report the manager by printing a statement that order quantity is <1000. So, we will again write the if block by checking order quantity is greater than 1000, print “Order quantity is greater than 1000”.

So, I need to keep this in a string. So, I am using double quotes here. Now, I am writing the else block. So, your else block should stay below your if and your code editor automatically did that, but you have to be cautious that, your if and else should be on the same indentation level and the statements that they want to run should be on a four space indented level like this 'print' statement. Now, when I press enter after this colon, the indented line should be on the first indentation. So, here we want to report this to the manager, “Order quantity reached below the threshold of 1000”.

The threshold of 1000. So, what this code block will do is it will first check whether the order quantity is greater than 1000 and if it is not then it will print this statement and we first need to check what is our order quantity currently. So, it is 900. So, the else clause should run here right. Now, let us check shift enter.

So, it is printing order quantity reached below the threshold of 1000. So, this way you can use if and else clause. Now, let us also see the 'elif', how to use 'elif'. So, the tracker should check whether the quantity reached below 500. So, let us copy this. So, let us check 'elif' and here we can check 'elif' order quantity <= 1000 and order quantity greater than 500, print order quantity below threshold and between 500 to 1000.

So, this is the first step and here our 'print' statement should try gets modified and should tell the manager that the order quantity reached below 500 units of critical threshold. So, since we still have order quantity as I think 900 yes.

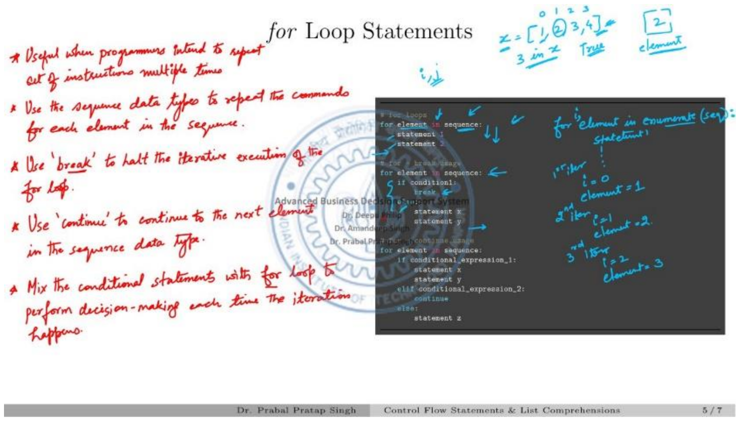
So, in this stage, this 'elif' clause will run right. Let us see. So, order quantity below threshold and between 500 to 1000 and if you just change the order quantity here. So, let us check if t = 400. Now, you again run this code and it will reach to the else clause that order quantity reached below 500 units of critical threshold.

So, this way you can use if, 'elif' and else. Further, we can also check the nested else clause. So, this by just changing few things like, if order quantity is greater than 1000, print above threshold or we can write the nested 'if' statement here. Order quantity is <= 1000, print below threshold else order quantity greater than 500 and order quantity <1000, print quantity between 500 and 1000.

We need to change this to greater than let us say, 900. So, now, what this code will do is, if the order quantity is greater than 1000, then the things are normal above threshold, but if the order quantity is greater than 900. So, since it is not greater than 1000. So, the interpreter will reach here, then it will check whether it is greater than 900, then it will just write below threshold, but if the order quantity is between 500 and 900, then it will write the quantity between 500 and 1000 which is a critical level here (let us say). You can again add one more else here by writing print no decision captured.

So, this is the code. So, this statement will run if all of the three decisions which we have written above will not being captured by the interpreter. So, we can run this code and it is saying no decision captured because the order quantity is 400. Now, if we change the order quantity to say, 700, then it will say that, the order quantity is 400. And then, run this code quantity between 500 and 1000.

So, this way you can use 'if', 'elif' and else clause in python. So, let us clear the screen again. Now, let us move forward.

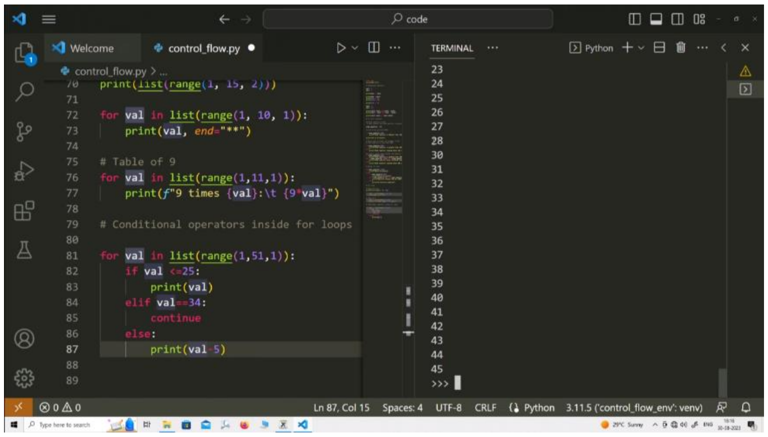


for Loop Statements

* Useful when programmers intend to repeat set of instructions multiple times
* Use the sequence data types to repeat the commands for each element in the sequence.
* Use 'break' to halt the iterative execution of the for loop.
* Use 'continue' to continue to the next element in the sequence data type.
* Mix the conditional statements with for loop to perform decision-making each time the iteration happens.

```
# for loops
for element in sequence:
    statement 1
    statement 2

# for - break usage
for element in sequence:
    if condition1:
        break
        statement x
        statement y

# for - continue usage
for element in sequence:
    if conditional_expression_1:
        statement x
        statement y
    elif conditional_expression_2:
        continue
    else:
        statement z
```

Dr. Prabal Pratap Singh Control Flow Statements & List Comprehensions 5 / 7



```
print(list(range(1, 15, 2)))

for val in list(range(1, 10, 1)):
    print(val, end="**")

# Table of 9
for val in list(range(1,11,1)):
    print(f"9 times {val}:\t {9*val}")

# Conditional operators inside for loops

for val in list(range(1,51,1)):
    if val <=25:
        print(val)
    elif val==34:
        continue
    else:
        print(val-5)
```

```
23
24
25
26
27
28
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
>>> 
```

So, the next statements are 'for Loop'. So, these are useful when programmers intend to repeat set of instructions multiple times.

It uses the sequence data types to repeat the instructions. So, we have seen the sequential data type like strings and list until now. So, what if we have a list and list contains the name of vegetables and you need to print all the elements of list in each new line.

So, the 'for Loop' can do that easily we will see how it will do. And, the next thing is we can use break statements. So, this is a different kind of statement that can halt the iterative execution of T 'for Loop'. So, with this statement you can or a programmer can halt the complete execution of the Loop.

So, it is very useful when you do not need to use the complete sequential data type and just want to print few things from the data and then break the complete Loop statement. So, the syntax is quite different here. So, let us first see the syntax. So, for a simple 'for Loop', this is a sequential data type, it can be a List, it can be a String or Dictionary, Tuples, but we have not yet learnt.

So, we will see in next lectures what are those, but any kind of sequence can be used here. Also, this is a new operator and what it will do is it will check whether there is an element in this sequence. So, let us say you have a list x which contains four numbers 1, 2, 3, 4. So, if you want to check whether 3 is in this list or not.

So, you can write 3 in x and it will either print true or false. So, since 3 is present. Now, for in 'for Loop's what happens is this in operator works quite differently the interpreter will first check the sequence and start iterating over it. So, this is a sequential data type. Now, what will interpreter do is, it will take each element out from this container and the element word here is not a keyword, this is a user defined word you can use i, j or anything here or any word you want to write here. So, any word except the keywords.

So, you can write here and this will become a dynamic container for this 'for Loop'. So, what happens is, let us say, this is an element container. So, in the first step of the iteration, the 'for Loop' will fetch the first value which is 1 and keep this value in this dynamic container. Now, the value of element is 1 and it will execute this statement after getting this value. So, you can use this value of element in these statements as 'val'. This is the first execution. Now, in the second execution what happens is interpreter will again go to the sequence and fetch the second value which is 2.

So, it will remove the value 1 from this container and write value 2 here and now dynamic container contains value 2 and now you can again run these two statements, statement 1 and statement 2 and these statements can use this element value as 'val' which has been changed now. So, this happens for all the elements of this sequential data type and until 4, it will keep on changing the value of element. After executing for all these values, the control will come here that is for the next 'for Loop'. So, this is how a simple 'for Loop' executes. Now, what if we want to break the statement let us say, after 2?

So, we can use this ‘break’ statement here by providing a first conditional check. So, until this line, everything happens the same way as in the first the simple 'for Loop', but since interpreter will see that there is an if else clause, it will first check for this condition and if this condition becomes true, then the control will jump from the complete 'for Loop' to this location that is, it will not iterate on all other values otherwise, if this condition is not true, then this else statement will keep on running. So, this is how the ‘break’ statement works. Similarly, there is a different statement named ‘continue’. We can use the continue statement to continue.

So, this is how the ‘break’ statement works. So, simply continue to the next element in the sequence data type and as we have seen in this statement, we can mix the conditional statement with the 'for Loop' to perform various kinds of decision-making each time the iteration happens. So, this happens and now as I have told you that, here when we are running this simple 'for Loop' in the element container, we are only getting the value of this, but while we are discussing the sequence data type, we know that this sequence data type, which is a list has an index also.

This is at index 0, this is at index 1, this is at index 2 and this is at index 3. So, what if we want to fetch the index value also each time the iteration runs? So, we can write here for elements in this new function enumerate and your sequence colon and now you can write statement 1 and other statements. So, what will happen is, here you can use one more container, I am using ‘I’ here. So, for I comma element. So, what happens is this sequence will give these values 1, 2, 3 and this enumerate function will also get the index values of these values.

So, for each iteration, we will get two values, the first value will get stored in the I and the first value is the index value. So, for the first iteration, I will hold value 0 and element will hold value 1. This is the first iteration. So, the second iteration, I will hold value 1 and element will hold the next value which is 2, for the third iteration I will hold value 2 and element will hold value 3. So, this way our iterations will keep on happening and this is very useful, when we need to capture both of these values and we need to use the index as well for our Looping statements.

So, let us see all of these things in our VS code now. So, we know different kinds of sequence data types like string and list, but now I am introducing a new function name as ‘Range’. So, the Range function can create a list of values. So, let us just create our first sequence of list that contains a range of values from, let us say, 1 to 15 with a step of 1 which is by default. So, let us see the output of this statement. So, what happened here is the range function will create a list from 1 to the last value which is 14 because 15 is excluded. The last value remains excluded here and it is printing each value with a step of 1.

So, the difference between two consecutive values will be 1. Now, if we just change it to 2, the step will become the difference of 2. So, you can see here the number of values in sequence data type reduced because each value is a step of 2. So, 1, after that 2, then 3, 4 is missing, 5, 7, 9, 11, 13 like that.

So, this is a way to create quickly our sequence data type. So, we are creating a list here. Now, we can directly use this in our 'for Loop', for 'val', because I have told you that we can write any variable name here as a dynamic container. So, I can write a list. This list function will create the list out of the values of from range. Range reports, we return the generator value which we have not yet touched, but we will touch in our next classes. 1, 2, let us say 10 only, and then 1.

We can print all these values one by one here. So, this is a very simple 'for Loop' and so, you can see here that I have wrongly typed the indentation level here. So, I can change my indentation level. You can see this. I was writing in the indented level of this else block, but we have completed that and now we need to change it to the base indentation level of the complete code base. So, we can use control + open square bracket.

So, this is a shortcut key to change the indentation level for large lines of code. Now, let us run this command. So, you can see here that this 'for Loop' fetched the value from this list container which holds values from 1 to 10, but excluding 10, 1 to 9 only. And, it will print each value one by one. So, 1 2 3 4 5 6 7, now it is writing on each new line, why, because I have told you earlier that print function has by default an end argument, which contains a new line character, that is why print gets written on each new line.

If we change this to, let us say 2 stars, then after each value, 2 stars will get printed. So, let us run this command. So, now, you can see that after each fetch, it will print 2 stars, and then it will again fetch the next value and after all the fetching, it will bring the interpreter to the same horizontal level y, because there is no next value. So, let us now move to the next command, but first clear our screen by running the same command. And, you can also move this terminal window to your right, so that we can see things clearly. I just used the right click button on this tab bar, and then you can get the panel position. Initially, it was bottom by default, you can use left or right anything.

Now, let us use the 'for Loop' to create a table which we learned in our earlier classes. So, let us write a table of 9. So, for values in the list range from 1 to 11 with a step of 1, we should write a F string as I have discussed earlier that F string can be a dynamic string that can use dynamic values from variables which can change over execution which is happening here. So, since it is a table of 9, we will write 9 times and our value is 'val' here and we can use the tab stepscape character, and then write our computation here which is 9 times 1. So, what this code is doing simply is, first the 'for Loop' will first create this R sequence data type from 1 to 10 only because 11 is excluded. And then, in each iteration, the value, this 'val' variable will hold from 1 to 10, and this 'print' statement will write 9 times in each iteration, but this will change. This will hold the value which this 'val' will hold, and then it will create this 9 into which we can use 'val' here. So, these curly braces will hold the dynamic values that get changed or any kind of computation. You can do any arithmetic computation which we have discussed earlier.

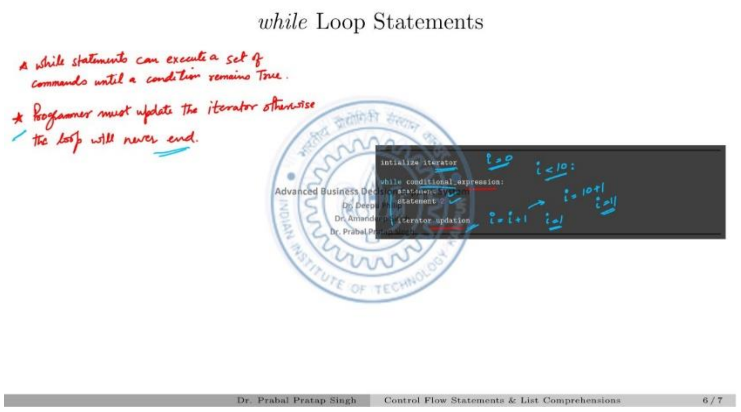
So, let us run this command. So, now you can see here that this is a complete table from 1 to 10. And now, let us move further ahead and use conditional operators inside 'for Loop'. So, here we can see what we can do for 'val'. And, let us move until 51 or 50 in our case, and then let us say if our value is <= 25, then ‘print’ our value, but otherwise ‘print’ value - 5.

So, what we are doing in this 'for Loop' is, we will first run this 'for Loop' for all values between 1 to 50. After that the interpreter will check whether this value is < equal to 25 or not. If it is <25, then it will simply print this value, otherwise it will decrease 5 from this value. So, at the iteration, when value holds 26, then it will print 26 - 5 which is 21. So, let us run this code.

So, let us confirm whether it is working right. So, until 25, it is working fine. After that, it starts with 21, then the same happens 21, 22, 23 until 45. So, when the value was 50, 50 - 5 is 45. So, it worked correctly. Now, if you just write here, ‘'elif’, ‘val’, equal to, equal to, let us say 34 ‘break’. So, now what happens is, whenever this 'for Loop' reaches this value 34, it will break out of this complete iteration, and it will not work anymore.

So, until 33, the last printed value will be 33 - 5 which is 28, and then it will stop. The complete 'for Loop' will come to a hold. So, let us see here, this was our 'for Loop' and then until 25, it worked well, and then for the next few values, until this 25, it worked well, and then for the next few values, it will keep on iterating until here, when the value reaches 34. So, it breaked.

The last value was 33 - 5 which is 28. So, it is going correctly. Now, we can do one more thing here. Let us remove this ‘break’ and write ‘continue’ here. So, what will happen is, ‘continue’ will skip this value iteration, and will keep on doing all other iterations. So, as soon as the iteration level reaches this 34, it will skip this value and will continue for all the code. So, let us just write this, and on 34, the value printed should be 34 - 5 which is 29. So, we do not have any value of 29 here, 28 then 30. So, this is how ‘continue’ works.



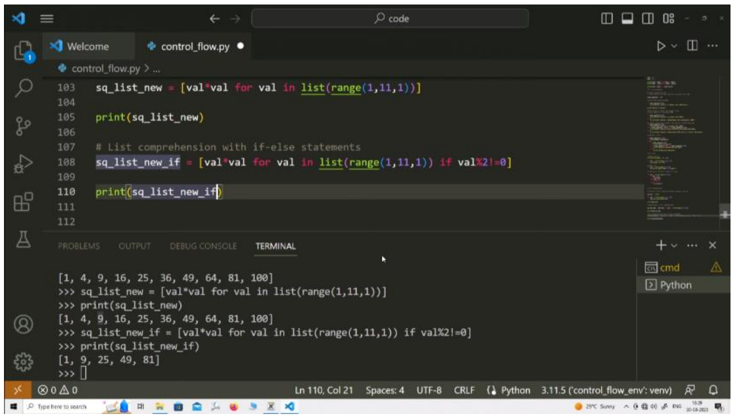
while Loop Statements

* A while statements can execute a set of commands until a condition remains True.
* Programmer must update the iterator otherwise the loop will never end.

```
intialize iterator          i=0
while conditional_expression:        i<10:
    statement
    statement 2
    iterator updation       i=i+1   i=1   i=10+1   i=11
```

Advanced Business Decision Support System
Dr. Deepu Philip
Dr. Amandeep Singh
Dr. Prabal Pratap Singh

Dr. Prabal Pratap Singh | Control Flow Statements & List Comprehensions | 6 / 7



```python
sq_list_new = [val*val for val in list(range(1,11,1))]

print(sq_list_new)

# List comprehension with if-else statements
sq_list_new_if = [val*val for val in list(range(1,11,1)) if val%2!=0]

print(sq_list_new_if)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> sq_list_new = [val*val for val in list(range(1,11,1))]
>>> print(sq_list_new)
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> sq_list_new_if = [val*val for val in list(range(1,11,1)) if val%2!=0]
>>> print(sq_list_new_if)
[1, 9, 25, 49, 81]
>>> 
```

Ln 110, Col 21 Spaces: 4 UTF-8 CRLF Python 3.11.5 ('control_flow_env': venv)

So, now let us move to our topic which is 'while Loop'. So, ‘while’ is also a Looping statement. And, 'while’ statements can execute a set of commands until a condition remains true. So, 'for Loop' runs until and unless the container contains some value which has some length, but a ‘while’ works on the conditional expressions. So, until and unless this expression will give you a true value, it will keep on iterating and keep on updating this iterator.

So, we will see what it is, and the programmer must update the iterator. Otherwise, the Loop will never end. So, to understand this, let us first see the syntax here. So, for a 'while Loop', we will first initialize an iterator. It could be anything, like let us say ‘I’ = 0.

Now, again we will write a conditional expression here. So, let us check whether ‘I’ is < 10 or not. So, since the current value of ‘I’ for the interpreter is 0. So, ‘I’ is < 10. So, this expression is true, and this 'while Loop' will allow the interpreter to move into these statements. So, for this, it will run this statement 1 and statement 2, then it will go and reach to this statement which will update the iterator.

So, how to update an iterator? Let us say, the programmer wants to increment the iterator. So, ‘I’ = ‘I’ + 1. So, when the interpreter reaches here, ‘I’ will become from 0 to 1. So, now ‘I’ is 1.

Now, the 'while Loop' will again check and see whether ‘I’ is still < 10 or not. So, it is still < 10. So, it will again do the same thing and it will keep on doing these things until ‘I’ this expression becomes false. So, when ‘I’ will become 10, then this iterator updation will say that ‘I’ = 10 + 1.

Now, ‘I’ will hold value 11, and this will become false. So, these statements will not run here, but, what if, a programmer forgets to update this iteration updation expression. So, that will make the interpreter keep on doing this Loop without stopping.

So, this is what this statement means, that a programmer must update the iterator, otherwise the Loop will never end. So, this is how a 'while Loop' iterates over values. So, let us move to the application of these 'for Loops’ and 'while Loops’. So, in the last lecture I talked about, lists are a very useful and general purpose sequence data type, and to get the benefits of a list by quickly creating it, we need to learn the ‘control flow’ statements. Now, what are these ‘control flow’ statements?

So, we can see what are the things that python provides using these 'control flow' statements to create a list quickly. So, list comprehension can reduce the lines of code significantly, and is a pythonic way of initializing a list, and these list comprehensions can also use conditional checks inside them. So, you may be wondering what all these things are, like what is the difference between list and list comprehension.

So, let us now see the application of these things first. So, let us say we need to create a list of squares. So, create a list of squares of numbers between 1 and 10. So, to create a list, what we can do now is, we will first initialize an empty list by declaring a name Square List equals list.

This is how we have seen that we can create an empty list. Now, we can run a 'for Loop'. For 'val' new in the list range 1 to 11 with a step of 1. We can update the Square List by appending something. So, we have seen that append is a method of list that can keep on adding new values into this list ,since it is empty. So, it will fill out these values.

So, since we want a square of the number. So, we can write 'val' into 'val'. So, this will square it. And after that, we can print this Square List. So, this is how we know that we can create a list. Let us run this code, and you can see here that we first initialize the list, then we run this 'for Loop', and now it is printing the Square List as 1 square of 2 is 4, 9, 16, similarly, until 10. But what if I tell you that with list comprehension, I can do all of this stuff in a single line.

So, let us create this section Application of List Comprehension. So, create a new variable that will hold the same list by using list comprehension. So, we just open and close a blank list here and start writing your 'for Loop', for 'val' in list range 1, 11, 1 and whatever statements you want to write for this 'for Loop', you can write here. So, 'val' contains your value. So, you can write ‘val’ into ‘val’ and with a space bar.

Now, what happens here is this 'for Loop' will keep on iterating until this container will have some value. And for each iteration, it will compute this value, and store it in this list at a particular index location.

So, now run this line and print this. Now you can see the output is the same, each index location contains a square of the value. So, this is how list comprehension has reduced your lines of code from 1, 2, 3, 4, 5, 6 to only a single line. So, this is the beauty of this. Now, let us move this terminal again to the bottom and let us see how we can write conditional statements inside this list comprehension. So, list comprehension with 'if-else' statements.

So, copy this, change this to a new variable and what if you just want to get the numbers of odd values inside this container. So, you can filter this by writing an 'if' statement after this. After defining all of these computations and your 'for Loop', you can create a filter ‘if’, so that it can filter these values from this list that is your sequence data type that we are working for our 'for Loop'.

So, you can write here ‘if’ ‘val’. So, we are looking only for two values that are odd. So, we can write ‘val’ not ‘val’ modulus 2. So, this computation will say that, what is the remainder when the value gets divided by 2 and if it is not equal to 0. So, if the remainder by dividing with 2 is not equal to 0, then it will be an odd value. So, for all these odd values, run this computation, and store it in R list.

So, now, we will only get these things 1, 9, 25. Now, let us see, we can write the 'print' statement as well, and run this line and this. So, now, you can see, we are getting this square for only odd values. So, this way you can perform the conditional checks also inside your list comprehension.

So, just one more thing is, this is the basic syntax, like we have sequence data type here, this element we have seen here, then this is the expression which gets computed to fill your list, and this is the conditional check which can filter elements from this sequence. And, a general list comprehension can also include some conditional checks here as well. For checking the conditions that are valid after filtering using this 'if' statement.

So, this will also be useful, but very rarely people use these things. We can frequently use this 'if' statement to filter the elements from the sequence.

So, I hope all these statements are useful for you in while creating your Decision Support Systems and we will now see other kinds of data types like tuples, sets and dictionaries in python which will heavily use all these statements like 'for Loop' to fetch the data from your dictionaries and we will also see how we can use comprehensions in dictionaries as well. So, thank you everyone and we will meet in the next lecture. Thank you.