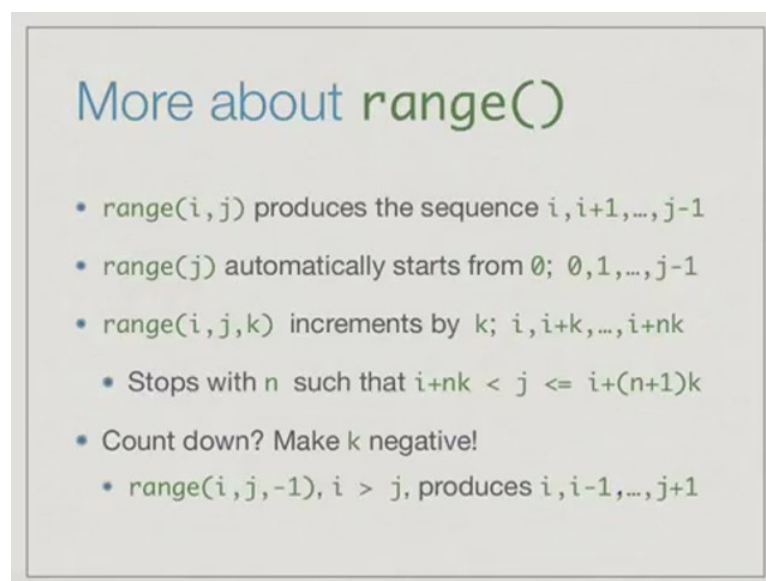**Programming, Data Structures and Algorithms in Python**
Prof. Madhavan Mukund
Department of Computer Science and Engineering
Chennai Mathematical Institute,Madras

Week - 03
Lecture - 01
More about Range ( )

(Refer Slide Time: 00:02)

We have seen the range function which produces a sequence of values. In general, if we write range i comma j, we get the sequence i, i plus 1 up to j minus 1.

Quite often we want to start with 0. So, if we will give only one argument if we just write range j, this is seen as the upper bound and the lower bound is 0. This is like a slice, where if you do not write the first argument of the slice, if we write for instance l colon n then it will run from 0 to n minus 1. In the same way if we just write range j, automatically we will get 0, 1 up to j minus 1.
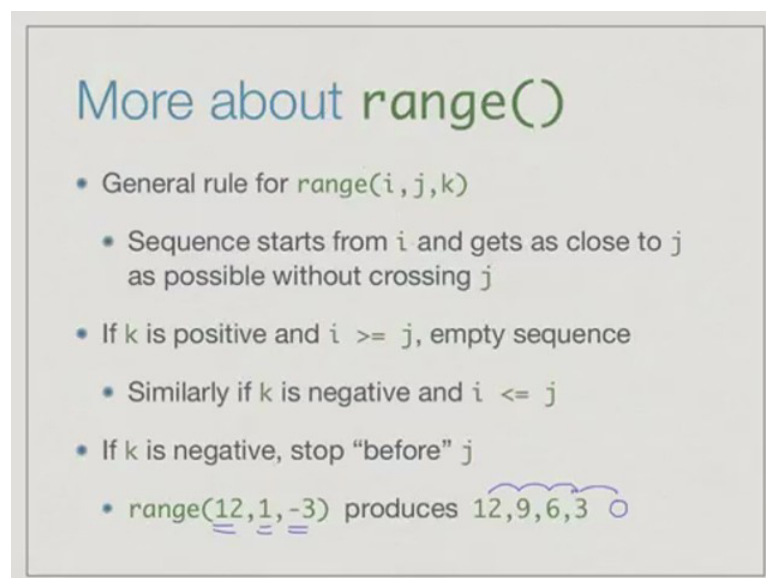
Often we may want to generate a sequence where we skip by a value other than 1. We want a kind of arithmetic progression if you are familiar with that. So, we want i, i plus k, i plus 2 k and so on, we do this by giving a third argument. The third argument if we

give it to range tells the range function to skip every k item. So, we have i then we go directly to i plus k. So, implicitly if we do not say anything it is like we put a 1 here right. So, the default value is 1 i, i plus 1 and so on.

Now, how far does this go? Well, we want to go until we reach normally j minus 1. In general, we do not want to cross j. So, what we will do is we will get i plus n k for the largest n such that i plus n k is smaller than j, but if i go one more step, if i go to i plus n plus 1 times k then i will cross j right. So, if we have a step then we will keep going until we cross j and we will stop at the last value that is before j.

Having a step also allows us to count down. All we have to do is make the step negative. So, if we say i comma j comma minus 1 then provided we start with the value which is bigger than the final value, we will start with i produce, i minus 1, i minus 2 and so on and we will stop with j plus 1.

(Refer Slide Time: 02:07)

More about range()

- General rule for range(i,j,k)
  - Sequence starts from i and gets as close to j as possible without crossing j
- If k is positive and i >= j, empty sequence
  - Similarly if k is negative and i <= j
- If k is negative, stop "before" j
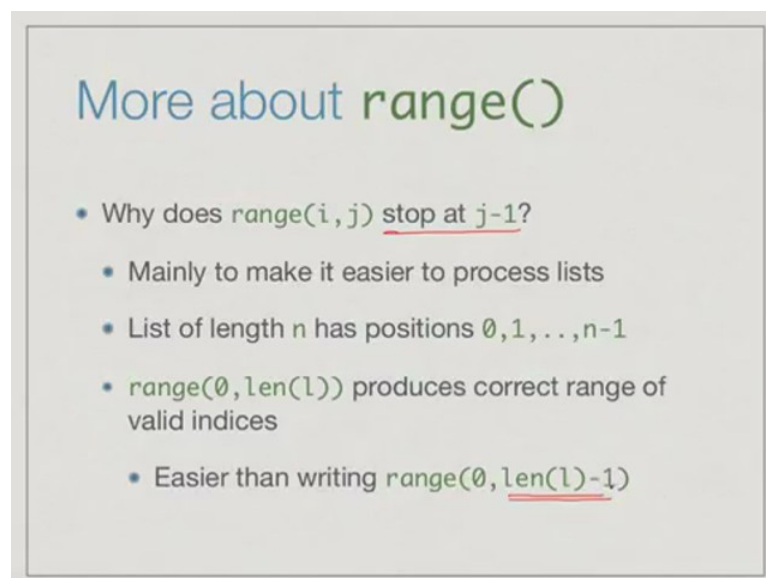 - range(12,1,-3) produces 12,9,6,3

The general rule for the range function is that you start with i and you increment or decrement if k is negative, in steps of k such that you keep going as far as possible without crossing j. In particular, what this means is that if you are going forward, if you are crossing, if you have positive, if your increment is positive then if you start with the

value which is too large then you will generate the empty sequence because you cannot even generate i because i itself is bigger than j or equal to j then that would not be allowed.

Conversely in the negative direction what happens is that if we start with the value which is smaller than the target value, we are already below j and so we cannot proceed, we get an empty sequence. This idea about not crossing j it is not same as saying stops smaller than j because if you are going in the negative direction you want to stop at a value larger than j. So, you can think of it as before and before means different things depending on whether you are going forwards or backwards.

Just to see an example, suppose we want to have a range which starts from 12 and whose limit is 1, but the increment is minus 3. So, we will start with 12 and then we will go to 9, then we will go to 6, then we will go to 3 and if we were to go one more step you would go to 0, but since 0 crosses 1 in the negative direction we would stop at 3 itself, we would not cross over to 0.

(Refer Slide Time: 03:42)

More about range()

- Why does range(i,j) stop at j-1?
  - Mainly to make it easier to process lists
  - List of length n has positions 0,1,..,n-1
  - range(0,len(l)) produces correct range of valid indices
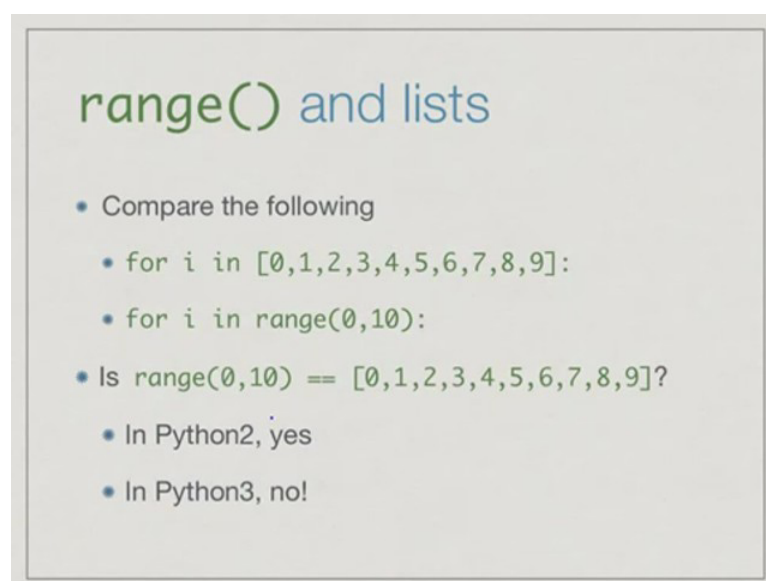   - Easier than writing range(0,len(l)-1)

It is often confusing to new programmers that range i comma j is actually from i to j minus 1 and not to j itself. So, there is no real reason why it should be this way, it is just

a convenience and the main convenience is that this makes it easier to process lists. Remember that if we have a list of length n, the positions in the list are numbered 0, 1 up to n minus 1. So, very often what we want to do is range over the indices from 0 to n minus 1 and if we do not know n in advance, we get it using the length function. We would like to range from 0 to the length of the list.

If the range is defined as it is now, where the actual value stops one less than the upper limit then range 0 comma length of l will produce the current range of valid indices. If on the other hand it did, what we would perhaps think is more natural and it will do i, i plus 1 up to j, if this were the case then every time when we wanted to actually range over the list positions, we would have to go from 0 to length of l minus 1. So, we have to awkwardly say minus 1 every time just to remind ourselves that the position stops one short. It mainly for this convenience that we can freely use the length of the list as the upper bound that the list stops, that the range stops at j minus 1.

As I said this is not, I mean, required you could easily define a range function which does the natural thing which is i, i plus 1 up to j, but then you have to keep remembering to put a minus 1 whenever you want the indices to stop with the correct place.

(Refer Slide Time: 05:23)

A range is a sequence and it is tempting to think of range as a list. We saw that for comes in 2 flavors, we can either say for i in a list or we can say for i in range something. So, range 0 to 10 generates a sequence 0, 1 up to 9. So, is this the same as saying for i in the list 0, 1, 2, 3, 4 up to 9. In other words, if we ask Python the following comparison, is range 0 comma 10 equal to the list 0, 1, 2, 3 up to 9 then the question is the result of this comparison true or false.

Here, we encounter a difference between Python 2 and Python 3. In Python 2, the output of range is in fact, the list. So, if you run this equality check, in Python 2 the answer would be true, but for us in Python 3 range is not a list. So, range produces something which is a sequence of values which can be used in context like a 'for', but technically the range is not a list, we cannot manipulate the output of range the way we manipulate the list.

(Refer Slide Time: 06:37)

range() and lists

- Can convert `range()` to a list using `list()`
  - `list(range(0,5))` `==` `[0,1,2,3,4]`
- Other type conversion functions using type names
  - `str(78)` `=` `"78"`
  - `int("321")` `=` `321`
    - But `int("32x")` yields error

Now, it is possible to use range to generate a list using the function list. So, name of the function is actually list. What we do, for example, is give the range as an argument of list. So, the sequence produced by a range will be converted to a list. If I want the list 0 to 4, I can say give me the range 0 up to 5. Remember, the range was stopped at 4
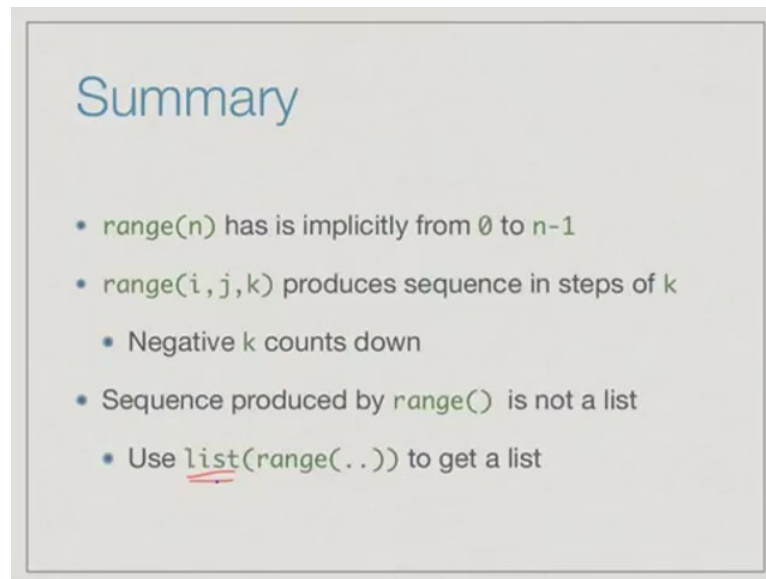
because 5 is the upper bound and this sequence will be converted to a list by the function list.

This is an example of a type conversion; we are converting the sequence generated by a range, if we said is not a list into a list by saying, make it a list. So, the function list takes something and makes it a list if it is possible to make it a list. If it is something which does not make sense it will not give you a valid value, now this is a general feature. So, we can use the names that Python internally uses for types also as functions to convert one type to another, for example, if we have the number 78 and we want to think of it as a string then the function str will take its argument and convert it to a string. So, str will take any value and convert it to a string representing that value.

This happens implicitly for instance as we will see when we want to display a value using a print function. So, what print will do is take a value, convert it to a string and only strings can actually be displayed, because strings are texts what we see on the screen or when we print out something is text. So, str is implicitly used very often. Sometimes we want to do the reverse we want to take a string and convert it to a value. So, for instance if the string consists only of digits then we should get a value corresponding to that string. If we give it the string 321 then it should give us back the integer 321. So, the value, remember the name of the function is the same as the name of the type to which you want the conversion to be done. So, we want to take a string and make it into an int, we use the name int.

Now, in all these things the function will not produce a valid value if it cannot do so. If I give the function int a string which does not represent the number then it will just give me an error. So, long as a type conversion is possible it will do it, if it does not it returns an error. We will see later on that actually the fact that you get an error is not a disaster there are ways within Python to recover from an error or to check what error it is and proceed accordingly.

To summarize what we have seen is that our simple notion of range from i to j has some variants. In particular, if we do not give it a starting point we just give it one value it is interpreted as an upper bound. So, range n is a short way of writing 0, 1, 2 up to n minus 1. Also we can use the third argument which is a step in order to produce sequences which proceed in steps i, i plus k, i plus 2 k and so on. In particular, if k is a negative step then we can produce decreasing sequences, and the last thing to remember is though range produces a sequence which can be used exactly like a list in things like for, in Python 3 a range is not a list.

If we want to get a list from a range output we must use this type conversion called list. In general, we can use type names to convert from one type to another type, provided the value we are converting is compatible with the type we are trying to convert to.