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

**Week - 03**
**Lecture - 02**
**Manipulating lists**

(Refer Slide Time: 00:02)



So lets take a closer look at lists now. We said that lists are mutable objects. So, if we have a list called list1 whose values are [1, 3, 5, 6], and then we assigned list1 to the list named list2 then we said both list1 and list2 in this case because lists are mutable will be pointing to the same list [1, 3, 5, 6]. Now if I take the position 2 which is this position and replace it by the value 7 then clearly list1 is [1, 3, 7, 6], but because list2 and list1 were pointing to the same object we have that list2 also has the same value [1, 3, 7, 6].

On the other hand, if we made this change in a more roundabout way. So what we did was, we took this list and then we first took its slice 1, 3 from 0 up to position 1 not 2 so I get 1, 3. Then I insert a 7, and then I take from position 3 on wards which is 6. then I also get [1, 3, 7, 6] in list1. But on the other hand because I used plus, what I have done is I have created a new list and therefore list2 has not changed, in this case list2 remains [1, 3, 5, 6]; in other words, concatenation using plus results in producing a new list.

```
madhavan@dolphinair:...thon-2016-jul/week3/python$ python3.5
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> list1 = [1,3,5,6]
>>> list2 = list1
>>> list1[2] = 7
>>> list1
[1, 3, 7, 6]
>>> list2
[1, 3, 7, 6]
>>> list1 = [1,3,5,6]
>>> list2 = list1
>>> list1 = list1[0:2] + [7] + list[3:]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'type' object is not subscriptable
>>> list1 = list1[0:2] + [7] + list1[3:]
>>> list1
[1, 3, 7, 6]
>>> list2
[1, 3, 5, 6]
>>>
```

Let us check this in the python interpreter. So, if I say list1 in to 1, 3, 5, 6 for example, and I say list2 is equal to list1 and then I just change the position 2 of list1 then list1 and list2 are 1, 3, 7, 6. On the other hand if I say list1 is equal to 1, 3, 5, 6 as before and list2 is equal to list1 and now I change list1 in this slice plus concatenation way, so if I say take the first two positions then put a 7 and then take the rest of list1. Now, list1 is again 1, 3, 7, 6, but list2 which was pointing to list1 is no longer pointing to list1 because the plus has created a new list and so the new list is not the same as your old list, so list2 continues to point at the old list so it is 1, 3, 5, 6.

This is an important point that one has to keep in mind regarding mutability. If we start reassigning a list using plus we get a new list, this also applies when we do it inside a function. If inside a function we want to update a function list then so long as we do not reassign it we are OK, but if we put a reassignment using plus then the list that has been updated inside the function will not reflect outside the function. So, we always have to be very careful about this.

(Refer Slide Time: 03:13)



## Extending a list

* Adding an element to a list, in place

  * list1 = [1,3,5,6]
    list2 = list1
    list1.append(12)

  * list1 is now [1,3,5,6,12]

  * list2 is also [1,3,5,6,12]

Now, how would be go about extending a list? Suppose, we want to stick a new value 22 at the end of a list; one way to do this is to say l is l plus 22. But as we saw, this plus operator will create a new list, so if you wanted to append a value to a list and maintain the same list so that for instance if it is inside the function we do not lose the connection between the argument and the value will being manipulated inside the function this would not do. We saw this function append in passing when we did gcd in the very first week.

Append is a function which will take a list and add a value to it. So here we have said list1 is 1, 3, 5, 6 as in the previous examples. List2 is list1 and now we have said take list1 and append 12. So, list1 the way we have write it is list1 dot append and in append we give the argument the new value to be appended. So what this does is, of course it will make list1 now a five element list with the original 1, 3, 5, 6 and a new value 12 at the end, but importantly this is the old list1 it is not a new list in that sense. So, list2 has also changed. Append actually adds a value in place both list1 and list2 point to be new list with 12 at the end.

(Refer Slide Time: 04:37)



On the other hand, if we had done it like I mentioned using the plus operator then we would find that list1 changes, but list2 does not because as we saw before concatenation produces a new list. So, append is a function which extends a list with a new value without changing it.

(Refer Slide Time: 04:48)

So, append takes a single value. Now, what if we wanted to append not a single value, but a list of values; we wanted to actually take a list and expand it by adding a list at the end, we had say 1, 3, 5 and we wanted to put 6, 8, 10. So, we want to take 1, 3, 5 and we wanted to expand this to have three more values, of course we can append each of these value one at a time. But there is a function which is provided which like append extends a list, but here this must be a list itself.

So extend takes a list as an argument, append takes a value as an argument. So, list1 extend list2 is the equivalent of saying list1 is equal to list1 plus list2, but remember that this must be a list it is not a single value it is not a sequence of value it is a list so it is must be given in square brackets you must give 6, 8, 10 as an argument to the extend function.

Now, this is to add elements to a list there is also a way to remove an element from a list. So, this is one way to remove it by specifying the value. We are not looking at a particular position we are looking for a value x and list1 removes the first occurrence of x in the list. Now, you may ask what happens if there is no occurrence x in the list. Well, in fact this will give us an error so you have to be careful to use remove only if you know that there is at least one copy of x and remember it only removes the very first occurrence, doesnot remove all the occurrences. So, if there are ten occurrences of x in list1 only the very first one will be removed.

```
>>> list1 = list(range(10))
>>> list1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list1.append(12)
>>> list1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12]
>>> list1.extend([13,14])
>>> list1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14]
>>> list2 = list1 + list1
>>> list2
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14]
>>> list2.remove(5)
>>> list2
[0, 1, 2, 3, 4, 6, 7, 8, 9, 12, 13, 14, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14]
>>> list2.remove(5)
>>> list2
[0, 1, 2, 3, 4, 6, 7, 8, 9, 12, 13, 14, 0, 1, 2, 3, 4, 6, 7, 8, 9, 12, 13, 14]
>>> list2.remove(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
>>>
```
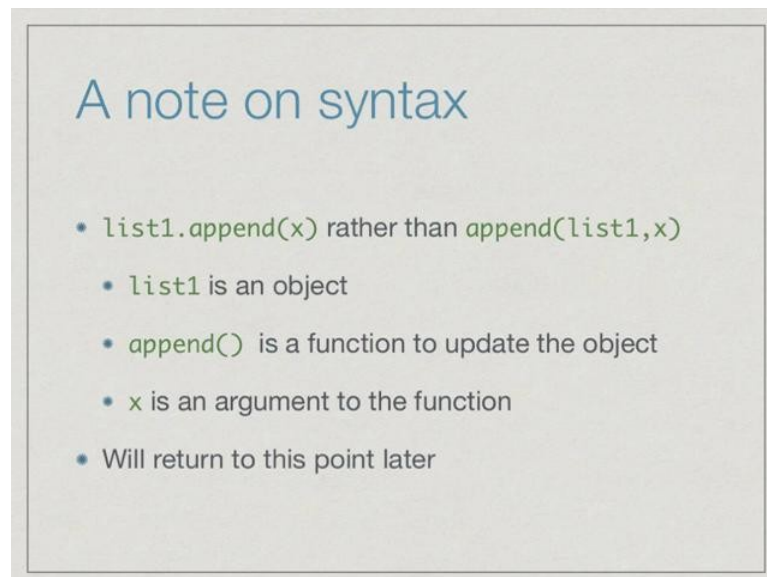
Let us explore these things. Let us start say with list1, so remember from the previous lecture we said we can take range and produce a list. Now if I do this I have list1 is 0, 1, 2 to 9, now if I say list1 dot append 12, then list1 is appended with 12. Now if I say list1 dot extend say 13, 14, then list1 now has 13, 14 appearing. So this is how append and extend work. Now supposing, just for the sake of argument I take list2 and I make two copies of list1. Now, list2 goes from 0 to 14 with a gap of course in between at 10 and 11, again from 0 to 14.

Now if I say list2 dot remove say 5, now there are two copies of 5 remember the first copy which is here in the beginning and second copy which is later, so this will remove the first copy. Now, if I look at list2 the first one skip at 4 to 6, but the second copy is still there. If I say it again then both copies have gone, because I do not have this 4, 6 and I also do not have a 5 here again its 4, 6. Now what happens if I have remove it a third time, now I get an error saying x is not in the list.

Remember that remove works only if x is in the list, if it is not in the list you get an error. Now it is important we will see later that we get an error it also has a name, so it says a value error. This will be useful because later on we will find that within Python we can actually examine errors and take alternative action if an error occurs and we can signal

what type of error it is by looking at the value that the error returns.

(Refer Slide Time: 08:18)



The append function looks a little bit different from the other functions we have seen so far. We would normally expect the function append to take two arguments; the list and the value to be appended. So we would think that the correct way or the natural way to write append would be to say append to list1 the value x. On the other hand what we have is this funny notation it takes says to list1 apply the function append with value x. In a Python terminology list1 is an object and append is a function to update the object and x is supposed to be an argument to the function append.

In such a situation we have an object and we then apply a function to it, so we use three functions attached to the object by using the dot notation rather than passing the object to the function which is a more normal way in which you think of functions. We will come back to this point later on and may be two - three weeks from now and we look at what is called Object Oriented Programming within Python.

There is another way to expand and contract lists and place, this is by directly assigning new values to a slice. So, we go back to our old example: list1 is 1, 3, 5, 6 and list2 is list1. Now what we are saying is that take the slice from position 2 on wards and assign it the value 7, 8. So remember the positions are 0, 1, 2, 3. So what this is saying is, take this slice namely 5, 6 and replace it by 7, 8. What we get is that, of course list1 is the slice 5, 6 is replaced by 7, 8, but this slice replacement happens in place.

It is a bit like assigning a new value at a given position. If I say list 2 is equal to 7 we said that position two becomes 7. In the same way if I say that list1 from slice two to the end become 7, 8 it changes 5, 6 to 7, 8 both in list1, but it also does not change where it is pointing to, so list2 also get affected. So both of them now say 1, 3, 7, 8.

Now, here we had a slice of length two and we replaced it by a new list of length two. So, we preserved the structure of the list in terms of the number of positions. This is not required, Python allows you to both expand and shrink a slice. For instance, you could have taken that list, now let us say we have this is 1, 3, 7, 8 and again we want to take slice 2 onwards which has two positions and we can say replace it by a list with three values. We are saying take this list take the slice from 2 to 3, the last two positions and replace it with three values and what we get is the old 1, 3 and this slice has now become
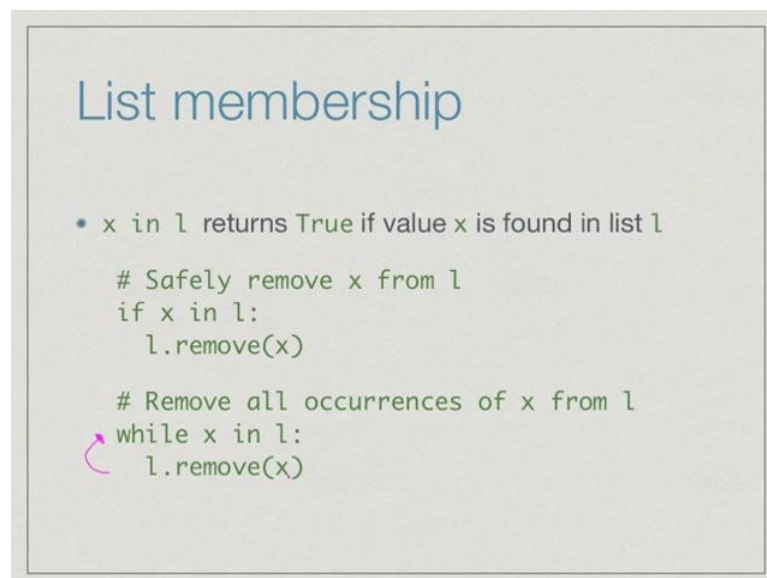
9, 10, 11. So, we had a four element list become a five element list. This is the one way to expand a list in place using a slice.

The other thing we can do is shrink a list; we can put a smaller thing. Supposing, we want the list to have just one value in the position 0 on 1, so we take the slice 0 to 2 which will give us these two positions so now you have a slice of length two, but we assign it a list of length one. So, this 1, 3 is replaced by just the single 7. Now we had a list of length five after the previous expansions which has now become a list of length four after this contraction.

With slices you can replace a slice in place, this can produce a bigger list or a smaller list depending on what you put in, but as you can imagine this can be very confusing. So, you should be very careful that you know what you are doing if you are trying to directly updates slices in the list.

(Refer Slide Time: 12:07)
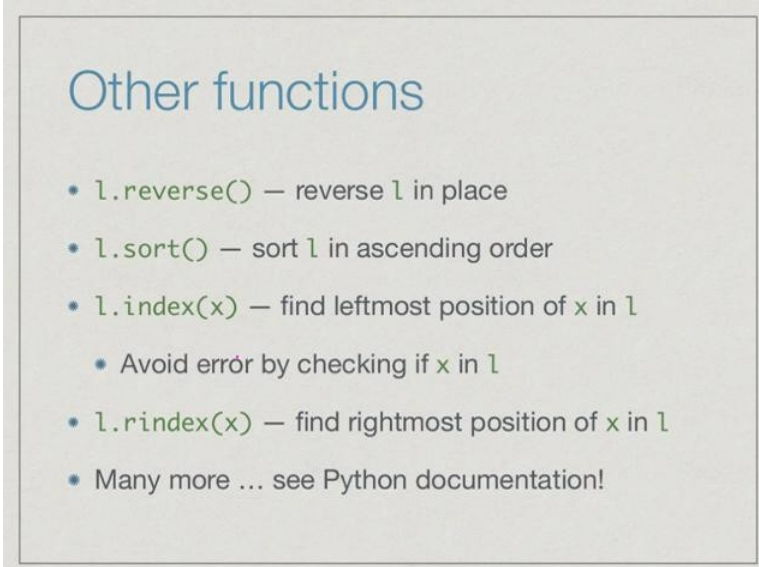


One of the very common things that we want to know about a list is whether a value exists in a list. So, Python has a very simple expression called x in l. So, x in l returns true if the value x is found in the list l. Now we can use this for instance to make our remove a safe operation; before we invoke l dot remove x we first check that x actually is

in l. So, if x is in l then the condition will be true and only then will we try to remove it, if x is not in l then we would not remove x. In this case we are guaranteed that l dot remove will not be called in an error prone context where it will say there is no x in l.

Also recall that remove removes only the first element. We can replace this if by a while and say that so long as there is a value x in l keep applying remove. This will in one short remove all the x's in l because every time we remove an x we go back and check if there is still an x in l if there are still on x in l we remove it, so from left to right this loop will remove all the x's in l.

(Refer Slide Time: 13:14)



## Other functions

- `l.reverse()` — reverse l in place
- `l.sort()` — sort l in ascending order
- `l.index(x)` — find leftmost position of x in l
  - Avoid error by checking if x in l
- `l.rindex(x)` — find rightmost position of x in l
- Many more ... see Python documentation!

Now there are a host of other functions defined for list, for instance l dot reverse will reverse a list in place, l dot sort will sort a list in ascending order. You can also sort it in other orders you can look up and see how to do that. If we only want to know whether an element is in l we say x in l, but if we want to know where it occurs then we use index it will find the leftmost position, but again it will give us an error if there is no x in the list, so we should first check if x in l and then find the index of the leftmost position.
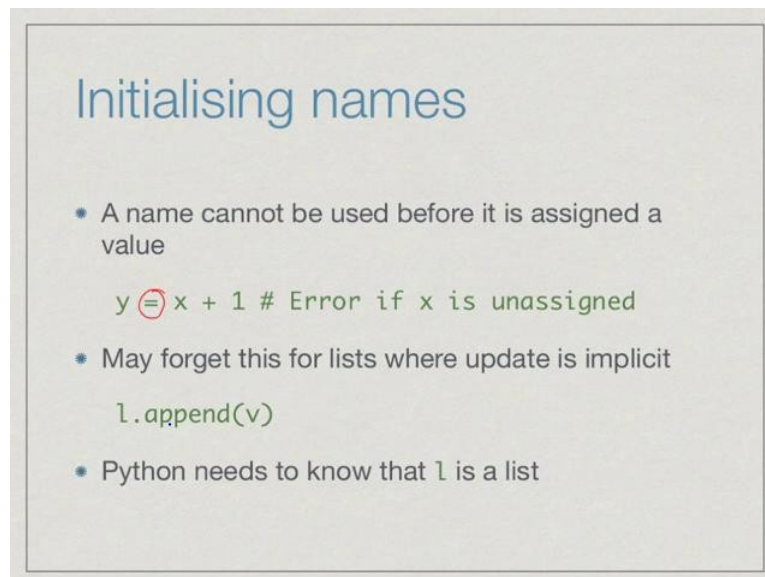
Now you might want not the leftmost or the rightmost position so there is an r index and there is a host of other functions and you must look up the Python documentation there is

no way that this course or any course can cover every function which is defined in Python for every type.

So you do have to look up the documentation and if you think that there should be a function that which does something natural very often there will be. So, try and look it up and see for yourself how it works and try to use it. If you have a question like what happens if I do this well Python is an interactive language. What happens if I do this? Just try it out and see and try to figure out from what you see in the interpreter, how the function works, in case there seems to be some ambiguity in the documentation. But above all do not be afraid to see in documentation only by looking up a documentation will you be able to learn the functions that you need because it is very difficult as I said to say up front every possible function that is there.
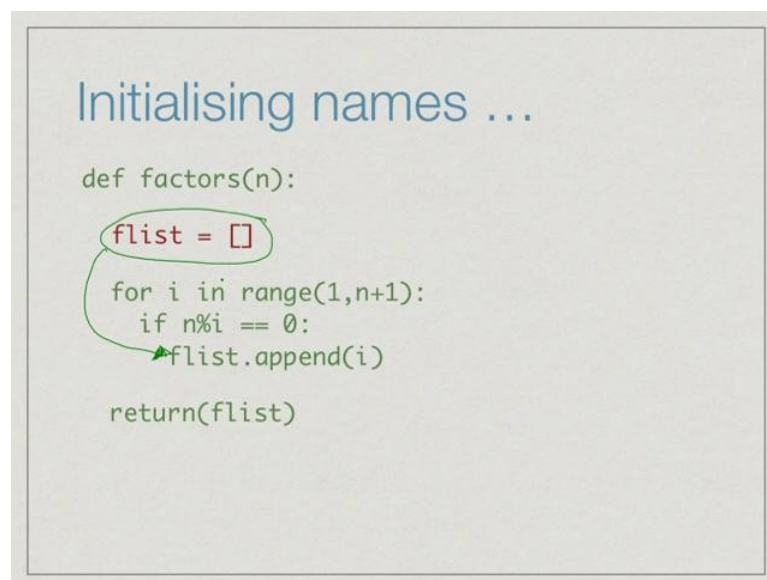
Final point regarding list is something we talked about in passing, which is that since names do not have types in Python, we do not have to announce the name, names just pop up as the code progresses. So every time a name pops us Python needs to know what value it is. Typically the first time we use a name we have to put it as part of an assignment, we have to assign a value to it and that value has to be something which is computable given the current names. So if we want to assign for instance to the name y

the expression x plus 1, at this point implicitly x must have a value, otherwise the x plus 1 cannot be evaluated.

So, if x has not been seen before and for the first time in my code I see it on the right hand side of an assignment it means that I am expected to produce a value for x but no value has been assigned so far and this will give you an error. This is quite easy to spot, so when you write something and you see something on the right hand side and you have not seen it before then it means a Python will flag an error and it is not very difficult to understand why this is so.

Now the kind of list functions we saw now, it is bit more subtle. When I say l dot append v there is no equal to sign. So it is not immediately obvious that l dot append v requires l to already be having a list value, why cannot I just append v for example to an empty list. Well, of course I can append v to an empty list how does Python know that l is in empty list. So, python needs to know that l is a list, before it can apply this append function.
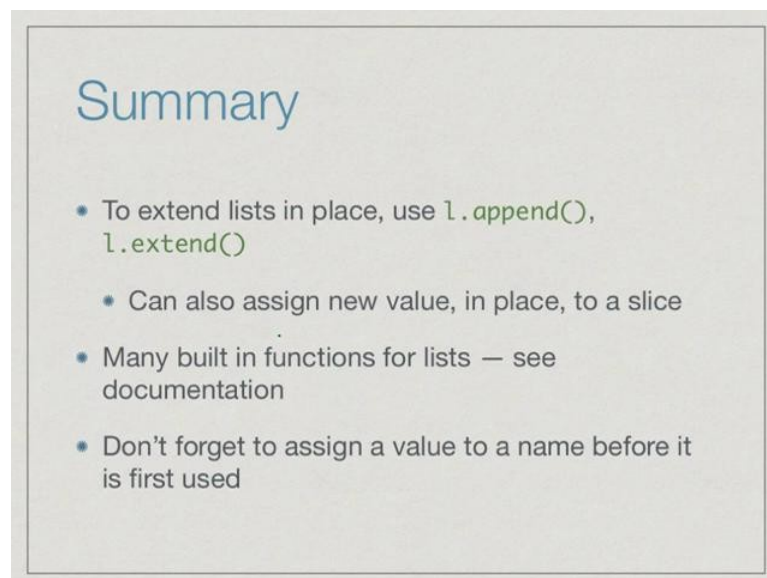
(Refer Slide Time: 16:21)



```
Initialising names ...

def factors(n):

    flist = []

    for i in range(1,n+1):
        if n%i == 0:
            flist.append(i)

    return(flist)
```

So, we saw this small function earlier which computes factors of n. So essentially what it does is, it takes all numbers in the range 1 to n. I take 1 to n plus 1 so that I run through the sequence 1 to n. And if a number divides n evenly if there is no remainder I have

used the append function now to append i to the list of factors which I will return. Now, the catch with this is that when I come for first time to this statement the first factor which will be 1 of course because 1 will have always be a factor.

Python will have to ask why flist has the ability to append a value, because flist has never been encountered to this point. We were careful when we wrote the code, of course we used plus because we did not use append in that code but it is the same thing. We have to be careful to insert this initialization. This initialization is only needed to tell Python when this first append happens that it is indeed the case that flist is of type list and therefore the append function is a valid function to apply to this name, without this you will get an error.

Just remember that you always have to make sure that every name that you use is initialized to a value the first time, so that whenever it appears later on, the value is clear and therefore what operations are allowed for this name are also clear to Python.

(Refer Slide Time: 17:47)

## Summary

- To extend lists in place, use `l.append()`, `l.extend()`
  - Can also assign new value, in place, to a slice
- Many built in functions for lists — see documentation
- Don't forget to assign a value to a name before it is first used

To summarize, what we saw is that we can extend lists in place using functions like append, extend and so on. We can also assign a new value in place to a slice of a list and in the process expand or contract the list, but this is something to be done with care; you

must make sure you know what you are doing. There are several built in functions on list; we will see some of them as we go along and use them and describe them as we see them, but it is impossible to document all of them and to go through all of them and it is also a very boring to just list out of a bunch of functions.

So, do look up the tutorial and other documentation which is available which I mentioned in the earlier weeks, so that you can find out what kind of functions are available. And finally, do not forget that you must assign a value to a name before it is first used otherwise, because names do not themselves have types, Python will not know what to do with the given name.