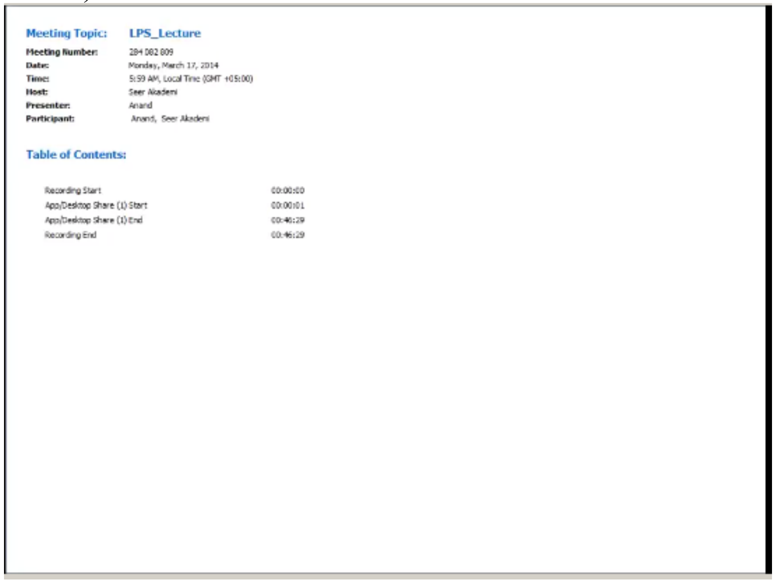
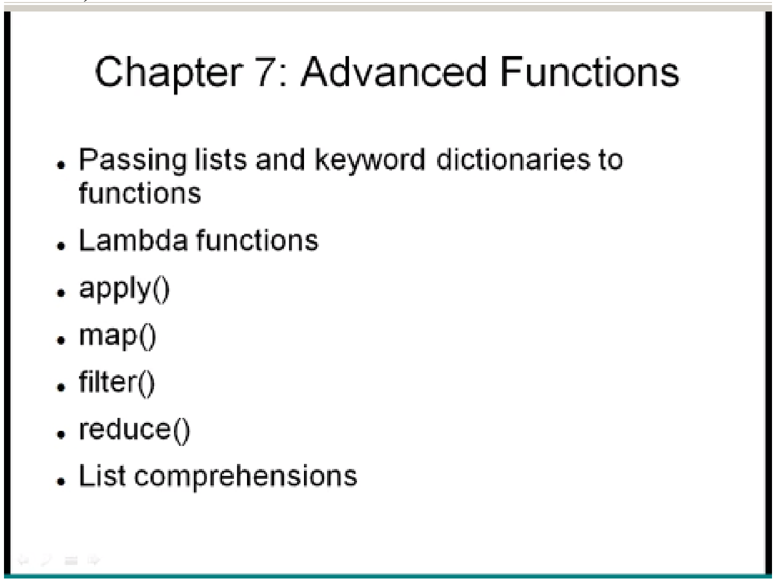
(Refer Slide Time: 00:00)



Meeting Topic:	**LPS_Lecture**
Meeting Number:	284 082 809
Date:	Monday, March 17, 2014
Time:	5:59 AM, Local Time (GMT +05:00)
Host:	Seer Akademi
Presenter:	Anand
Participant:	Anand, Seer Akademi

Table of Contents:

(Refer Slide Time: 00:06)



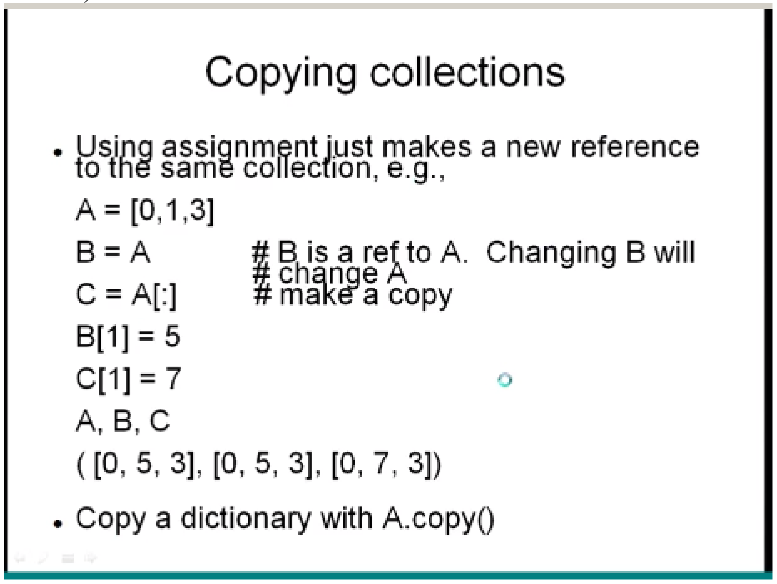
Chapter 7: Advanced Functions

- Passing lists and keyword dictionaries to functions
- Lambda functions
- apply()
- map()
- filter()
- reduce()
- List comprehensions

Okay so I planned is not it yeah and then everyone again welcome to the LPS lecture, we actually we went through several concepts on Python last time around, the important ones are first of all the kind of talked about the triples list as data structure and then how we can operate on them, that was the main focus from last time I think you guys remember that, today we will be continuing with that actually the on python programming essentially, so python programming itself we covered a lot of things so far.

So as you know we talked about the control flow then, the data flow itself with starting with various data structures with string or integer types, the floating point types things like that then in the last lecture the power ball the two other new types like the triples and more the lists and the

kind of we went through how to operate on them, or the index then things like that so I think right now we have a good understanding of how the Python programming actually works, so that we can take the next step of going to some of the advanced concepts.
(Refer Slide Time: 01:51)



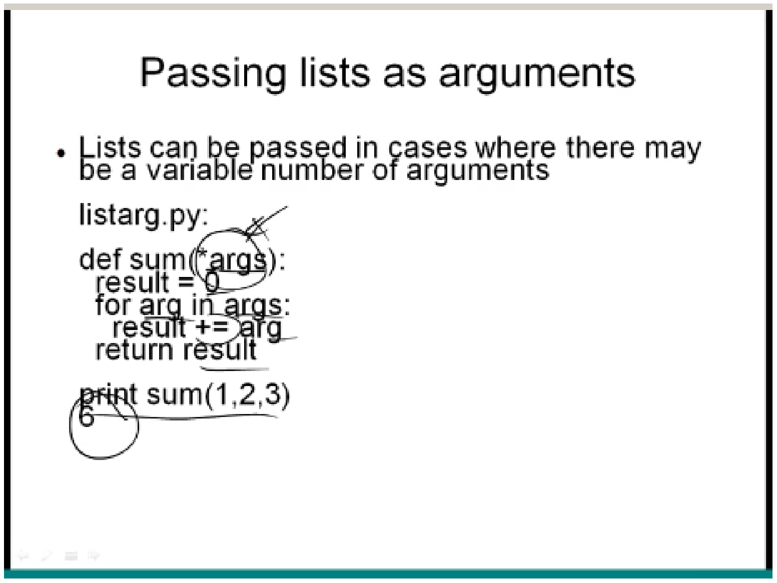
Copying collections

- Using assignment just makes a new reference to the same collection, e.g.,

```
A = [0,1,3]
B = A         # B is a ref to A.  Changing B will
              # change A
C = A[:]      # make a copy
B[1] = 5
C[1] = 7
A, B, C
( [0, 5, 3], [0, 5, 3], [0, 7, 3])
```

- Copy a dictionary with A.copy()

so yeah we also talked about the collections and basically like what collections mean, which is another data structure that we talked about, when we talk and we discussed TCL essentially in collections as a data structure work synopsis supports that we talked about, so similar collections are available in Python we talked about them so today what we are going to go through is basically some of the advanced functions, essentially we are going to talk about how to pass lists and keyword dictionaries two functions.
Then we also will talk about something called the lambda function it is a clear it is a function generator so we will talk about that, then we will talk about some of the functions like apply, map, filter, reduce and we also will go through some more understanding of the list like my list comprehensions, so let us begin todays lecture.
(Refer Slide Time: 02:51)



Passing lists as arguments

- Lists can be passed in cases where there may be a variable number of arguments

listarg.py:

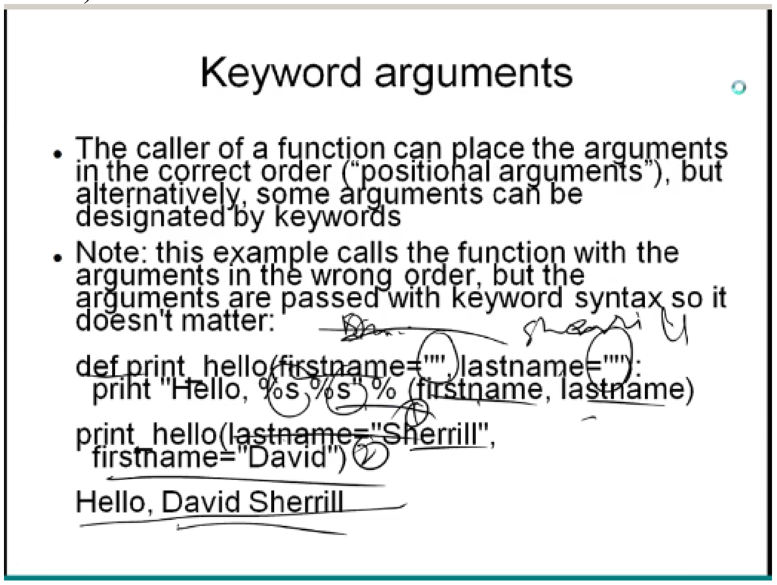
```
def sum(*args):
  result = 0
  for arg in args:
    result += arg
  return result

print sum(1,2,3)
6
```

So how do we pass lists as arguments, so the list can be passed in cases where there may be a variable number of arguments, so this is something that we saw in TCL as well or like the specify this specific special parameter in the as a function and then can pass the arguments, so here the list are got py here we are defining this function has some the functions name is come and, then we say basic the arguments are we denote it like this basically and say like the result is 0 or arg in args result + R and then the return results.

So here since the star ours is a itself is a pointer we can actually then parse a list into this, so now like if you say like print sum of 1, 2, 3 the answer is 6 so notice in tax of how do we pass the argument.

(Refer Slide Time: 04:08)



Keyword arguments

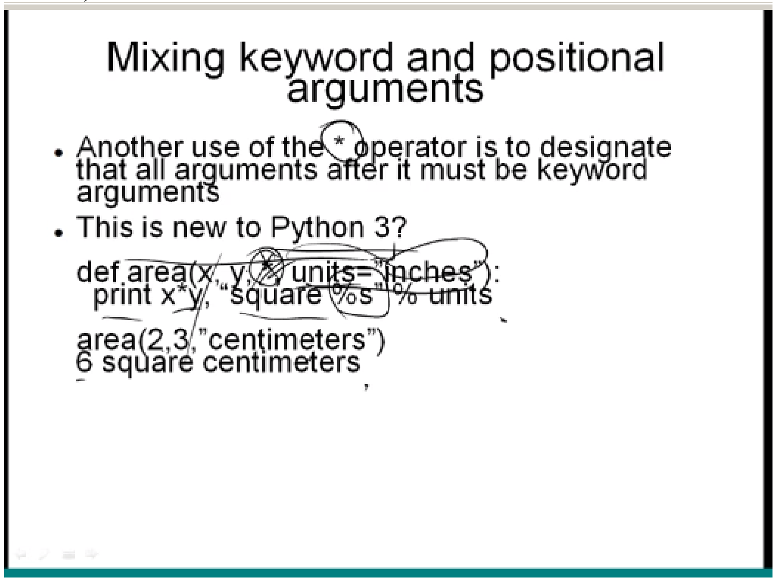
- The caller of a function can place the arguments in the correct order ("positional arguments"), but alternatively, some arguments can be designated by keywords
- Note: this example calls the function with the arguments in the wrong order, but the arguments are passed with keyword syntax so it doesn't matter:

```
def print_hello(firstname="",lastname=""):
  print "Hello, %s %s" % (firstname, lastname)

print_hello(lastname="Sherrill",
  firstname="David")

Hello, David Sherrill
```

So then now the keyword arguments essentially ligament so the caller of a function can play the arguments in the correct order, so that is the positional arguments, so that is this is something that we saw even in a TCL as the positional versus the non position stuff, so some sometimes we can actually change the order by assigning keywords ,so how do we pass that, so for example so here like I mean in this one with example it is a function with arguments in the wrong order.
But the arguments are passed with the keyword syntax so that it does not help so how do we do that so in TCL we saw some newer construction to handle this form of an issue we talked about that here, let us see basically like oh um print hello basically again and then we have the first name equals, and then the last name equals, and then we can say like okay pass the hello Prince the first name last name, and then those are the two names that are passed into the function, now if you want to so standard one is basically like I mean you can just write it like I mean what is.
So if you just say hello Sam or David Sherrill and channel then it will print exactly the same, but if you want to actually interchange the arguments you can just say that last name equal to Sherrill and personal David as a hard assignment with this function names, then it will actually print appropriately, so even though here the order is interchanged basically the last name and first this is first, and then this is second, but it prints correctly which is the first name and second.
So this is one way to pass the keyword arguments essentially so on when we have like specific like when we change the order then this may be useful.
(Refer Slide Time: 06:38)



Mixing keyword and positional arguments

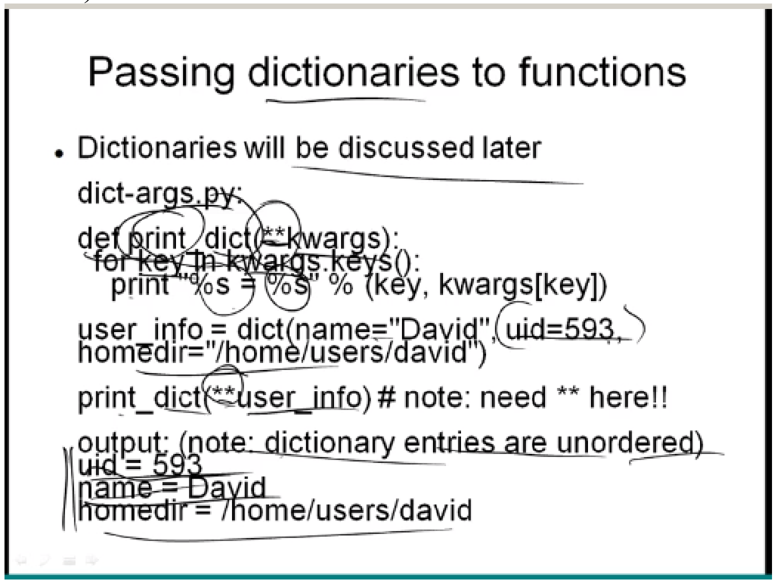
- Another use of the * operator is to designate that all arguments after it must be keyword arguments
- This is new to Python 3?

```
def area(x, y, *, units="inches"):
  print x*y, "square %s" % units

area(2,3,"centimeters")
6 square centimeters
```

So what happens if you are missing key words and the positional arguments, so this is also like something that we saw how do we pass, when we talked about TCL what is some arguments are missing and we would not pass it appropriately, there we use the use of the TCL understanding of

what strict substitution strings be curly braces and the coats, and then we made use of it to pass these kind of patterns, so here how do we do it like in Python, so these are already like the star operator which is used as a variable length of argument.
So we can use the same concept to designate all arguments after this as the arguments, so and this is something that is new in Python 3, so here we have an example again the area we say basically XY and then we say like the start of paper, and then they say units equal 2 inches, so now when we do print X times Y and then we say X square and then we say basically X square this particular units.
So if you say basically like 2, 3 centimeters it gives 6 Square centimeters, so the star essentially like an captures this so you omit this then the units is taken as inches, if you put it then the units become whatever that star that we specify.
(Refer Slide Time: 08:31)



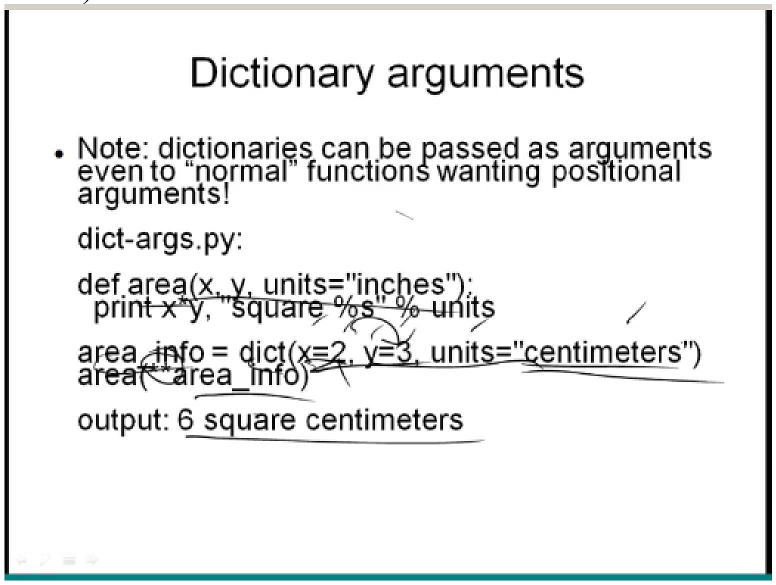
Passing dictionaries to functions

- Dictionaries will be discussed later

```
dict-args.py:
def print_dict(**kwargs):
  for key in kwargs.keys():
    print "%s = %s" % (key, kwargs[key])
user_info = dict(name="David", uid=593,
homedir="/home/users/david")
print_dict(**user_info) # note: need ** here!!
output: (note: dictionary entries are unordered)
uid = 593
name = David
homedir = /home/users/david
```

And now the other important thing is passing the dictionaries two functions as you know like dictionaries are the associative arrays that we saw in Perl, and then we also saw similar concept in TCL and in Python they are all dictionaries, I hope you remember that and we are going to be talking about dictionaries given in more detail go wrong, but here in this one essentially likely have this particular function all print, and then we specify it now we specify two stars and then the KWargs.
So this is the keyword argument so basically for each key in keyword args doc Keys essentially and then we can print what is the key and what if we value, so here we say basically like okay so we did an example of users in police dictionary name user ID and home directory, there are three fields here, so if you really say like I mean print date test our user info, I think this one it will

print underscore ,and then the output is essentially like UID 593 name is David and then home directory is particularly like / home / users / David.
So one thing that you want to notice we talked about this earlier also we talked about the dictionary database or dictionary data structure, the dictionary entries are unordered unlike lists or even tipple for the matter, so that is the reason why we even though we specified like this basically making use ready in second the name of course when it printed out if you use ready can be first, and then name is second.
So since it is unordered this is what you may expect and the other specific thing is basically the when you pass the dictionaries as an argument you need to specify with another double star.
(Refer Slide Time: 11:13)



Dictionary arguments

- Note: dictionaries can be passed as arguments even to "normal" functions wanting positional arguments!

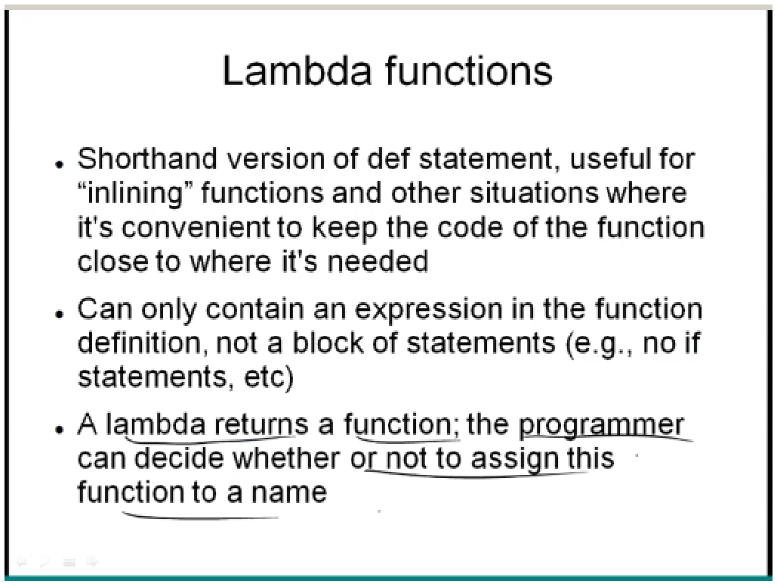
dict-args.py:

```
def area(x, y, units="inches"):
  print x*y, "square %s" % units

area_info = dict(x=2, y=3, units="centimeters")
area(**area_info)
```

output: 6 square centimeters

You now how do we pass the dictionary arguments essentially to function, so the dictionary can be passed as argument even to normal function wanting the positional arguments, so how do we do that essentially like so here again define the area XY units inches, and then here now we have this data in a dictionary, so how do we pass that information, so here we say basically area info is there is dictionary that they create which is X = 2 Y = 3 and then unity.
So the three different keys basically XY and units and then their values at 23 and centimeters, now we can actually even though like this is a normal function, we can pass this dictionary into this, this normal function by just specifying the double star area in for the dictionary name, and then the output is just 6 square centimeters, which is essentially the same 2 times 3 and then put centimeters square centimeters okay.
(Refer Slide Time: 12:40)

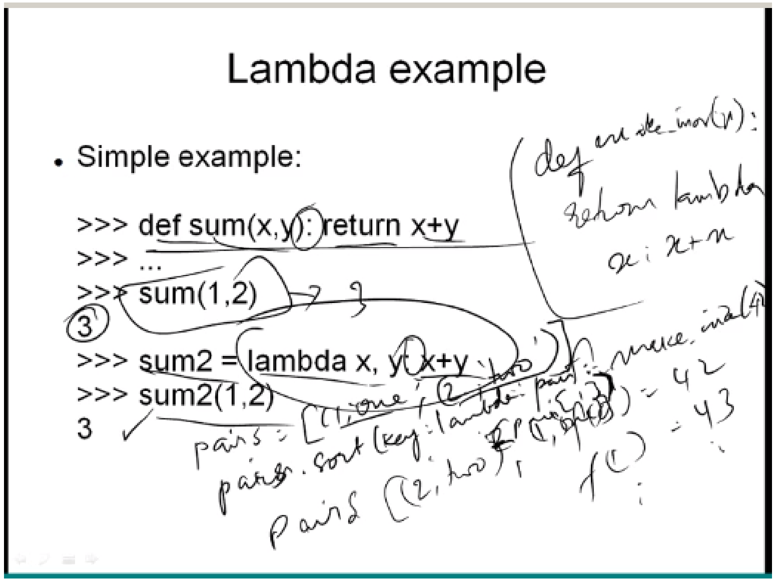

Lambda functions

- Shorthand version of def statement, useful for "inlining" functions and other situations where it's convenient to keep the code of the function close to where it's needed
- Can only contain an expression in the function definition, not a block of statements (e.g., no if statements, etc)
- A lambda returns a function; the programmer can decide whether or not to assign this function to a name

So now we come to the another major topic that we want to talk about today all the lambda functions, so the lambda functions are essentially their shorthand version of def statement or a different statement one function it is useful for in lining functions that means that basically you can define function from the book, and also in other situations where it is convenient to keep the code or function close to where it is needed so where it is getting boosted.

One thing to notice it can contain an expression in the function can only contain an expression in the function definition, but not block of statements there is only one expression position, so we cannot specify like if statements or orloops etcetera inside the lambda function, and then a lambda returns a function programmer can decide whether or not to assign the function and so this is another priority, even though the lambda function the programmer can decide whether to assign it to function subject to any so let us see how we can use this for lambda function.

(Refer Slide Time: 14:02)



Lambda example

- Simple example:

```
>>> def sum(x,y): return x+y
>>> ...
>>> sum(1,2)
3
>>> sum2 = lambda x, y: x+y
>>> sum2(1,2)
3
```

def make_incr(n):
return lambda
x: x+n
f = make_incr(42)
f(0) = 42
f(1) = 43
pairs = [(1, one), (2, two)]
pairs.sort(key: lambda pair: pair[1])
pairs [(2, two),

So here is one example this is a very simple example, so in normal course we define a function called sum X Y and then we say return X + Y, and then if you use it you can say like someone 12 and then this will result in three as it shows here, if you want to use the lambda function we can directly create sum 2 = lambda X Y, and then you need to specify this column same as here, this X +Y, so this is like just a shorthand representation of this, this line to here.

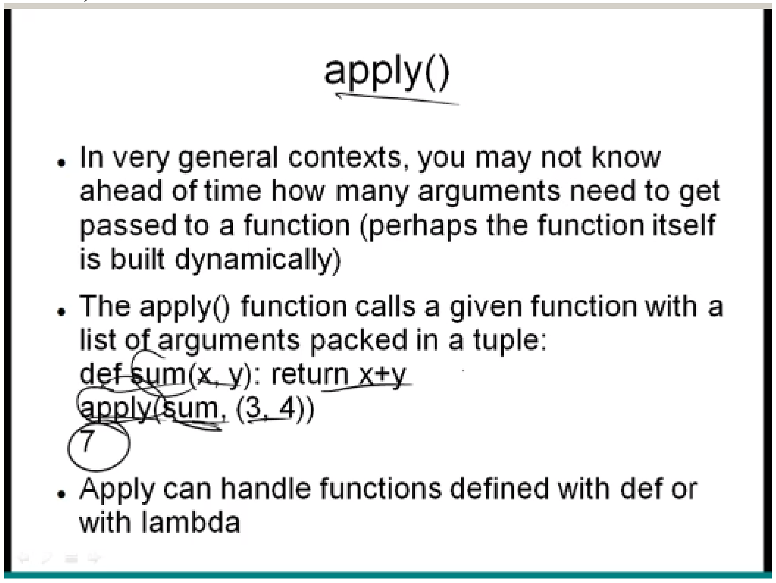
And then when we say like some 2(1,2) that is also the same, so we can use these lambda functions wherever the function objects are required, they are syntactically as I mentioned like the suitable single expression, semantically they are just syntactic sugar for the normal function, another thing is basically the lambda functions can reference variable from the containing scope, so again like we can commit to that.

So another thing is basically we can say neft makes incremental and with N then this function, we can define it as the term lambda X, X+, so if you define it function like this then if you say like you can say F= make incremental A 42 F 0 will become 42 and F1 is 43 etcetera, so you can call it this kind of the lambda function, so you can use this lambda not just as a function but I mean in between functions or quick rather than like I mean actually using a function inside a function you can use it like a lambda inside a function.

And get better coding no other way, so another example is like you can add this small function as an argument so how do we do that, so let us see like I mean we define a dictionary actually a list dictionary basic four pairs and then detail of the pairs is 1, one 2, two this is pair and then we say assault the pairs and the assault and we defined key = and here we can specify a lambda pair and a pair is actually pair one.

Now if you execute like pairs you can actually now sort the this thing, in this case actually it will give you 11 and 22 but you can actually sort it with actually like never sort from this one so basically letting their will now become going to the best it is 2, two and 1,one , so this is one, one application essentially we have been specified by the lambda expression to return a function central.

(Refer Slide Time: 19:47)



apply()

- In very general contexts, you may not know ahead of time how many arguments need to get passed to a function (perhaps the function itself is built dynamically)
- The apply() function calls a given function with a list of arguments packed in a tuple:

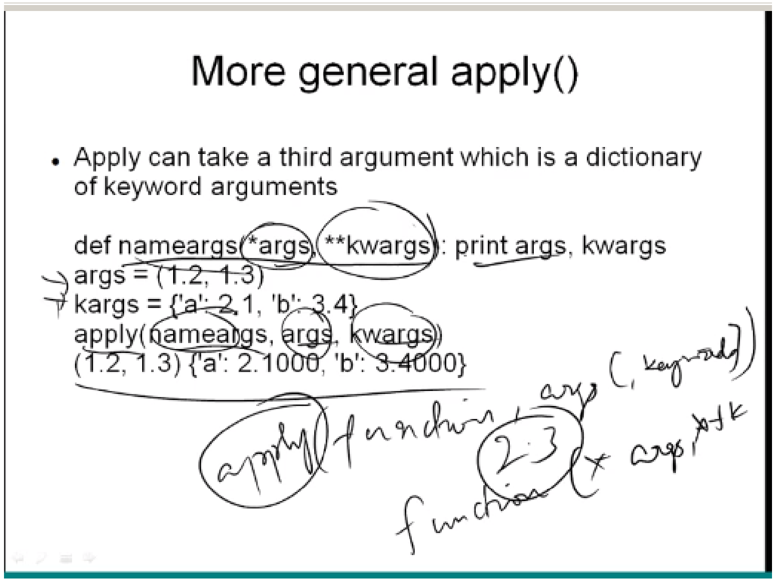
```
def sum(x, y): return x+y
apply(sum, (3, 4))
7
```

- Apply can handle functions defined with def or with lambda

So now we go into some more details actually and other functions essentially, so here we talk about applied, so in very general context you may not know ahead of time how many arguments need to get passed to a function, this is like a typical scenario essential variable arguments and also like I mean if the function itself is built dynamically then you would not know essentially.

So the apply function also given function with a list of arguments, packed in a triple so that is the usefulness of applying, so here we can say basically like a function that sum X Y return XY, now we can actually call, call this to execute this function using applied so that it is basically we call we pass this simple function as a triple that is some 3,4 and then the result is 7, the apply cannot handle functions defined with F or with lambda, so that those things are possible as well okay.

(Refer Slide Time: 22:18)



More general apply()

- Apply can take a third argument which is a dictionary of keyword arguments

```
def nameargs(*args, **kwargs): print args, kwargs
args = (1.2, 1.3)
kargs = {'a': 2.1, 'b': 3.4}
apply(nameargs, args, kwargs)
(1.2, 1.3) {'a': 2.1000, 'b': 3.4000}
```

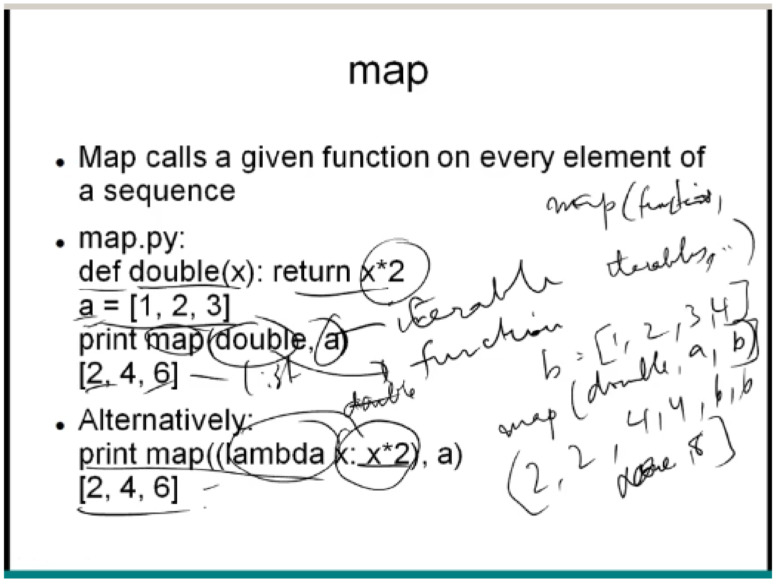
apply(function, args (, keywords))
2.3
function(* args, **k

So now also the apply can make the third argument which is the big which is a dictionaries keyword arguments essential, so here an example name arguments, start arguments we know that this list and then dictionary we want to print this whole thing, and again here we have defined these two variables, and if you want to print this essentially we can now they say apply name arguments args and kwargs and then it prints the whole thing.

Again you can see that basically you can park the function name the its arguments all in one shop, and then you can get it executed, so here it is a variable name variable length of the arguments you can still get executed with this function, so essentially like I mean the key thing to note with apply is basically It is the format is essentially like a function and then args basically here all in have keywords.

This is the general format of apply, and one thing to note is actually like this apply itself is deprecated since version 2.3, and an equivalent function is just use function same function with star args basically and ** keywords like directly this will actually work better in this context.

(Refer Slide Time: 24:33)



map

- Map calls a given function on every element of a sequence
- map.py:

```
def double(x): return x*2
a = [1, 2, 3]
print map(double, a)
[2, 4, 6]
```

- Alternatively:

```
print map((lambda x: x*2), a)
[2, 4, 6]
```

map(function, iterables, ...)

iterable

function

[1, 2, 3]

double

b = [1, 2, 3, 4]

map(double, a, b)

[2, 2, 4, 4, 6, 6, none 8]

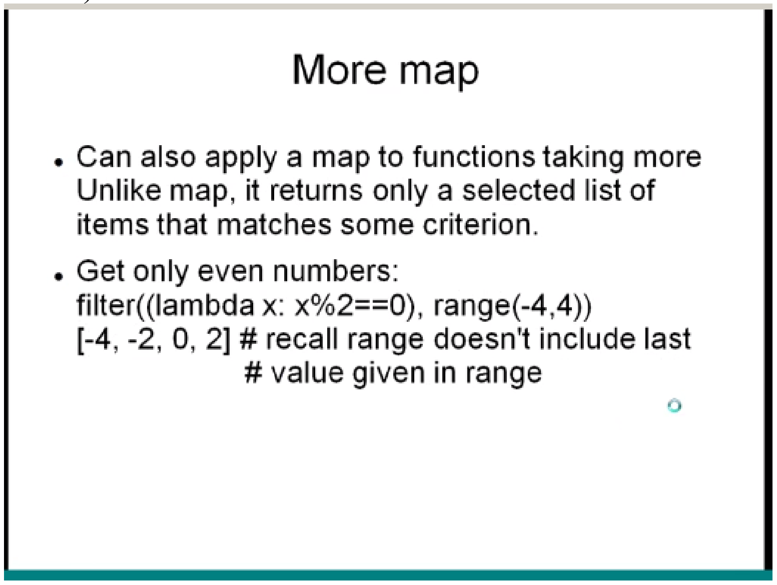
So now let us look at some other function this is what is called a map function, so the map calls a given function on every element of the sequence essential, so once you specify the sequence essentially like communicate basically it is a kind of iterate through that sequence, so essentially like in another words basically applying the function every item of the iterable and return a list of results, so you can have a single function which will take a single argument but you can apply that to a sequence and then get the result.

You can think of this as like a vector function if additional iterable arguments are passed the function must take the take that many arguments take that many arguments and is applied to the items from all iterable in parallel, if one iterable is shorter than the another it is assumed to be extended with none items so we bother none actually previously and so the remaining will be treated as a none particular in trouble.

And if function is none the identity function is assume so if there are multiple arguments map again returns the list containing an a consisting of triples, containing the corresponding items from all the intervals it is just a transpose at that point the iterable arguments maybe sequence are any iterable arguments and the result is always a list, so there is another involvement look at here is an example it is called double X return X 2.

So if you want to apply to this sequence like 1, 2, 3 all we got to do is the map and then the function name followed by the triple which is here in this case it is little sequence A and then we did like 2,4,6 at the list so this is the list this is the iterable that is the function, so the maps a general thing is basically the map you have a function, and then set a fixable you okay so what this means is like I mean if we have another set for B = say 1, 2, 3, 4, and then if you say map double A, B.

So now the answer is going to be able 2, 2 4, 4 6, 6 then 0 or none 8, so that is the list that you can get you and then we can also like add the lambda inside this map function, so here directly we write lambda X, X times 2 and then A then the thing goes, so here we do not have to specify this X equal to X + X times 2 this is a another function for double, so omit this double and then directly with map list and get the same answer.
(Refer Slide Time: 29:03)

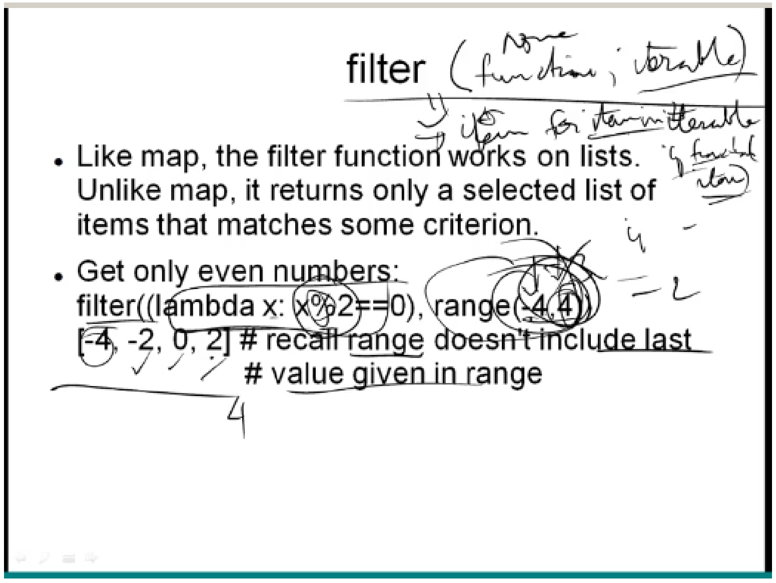


More map

- Can also apply a map to functions taking more Unlike map, it returns only a selected list of items that matches some criterion.
- Get only even numbers:

```
filter((lambda x: x%2==0), range(-4,4))
[-4, -2, 0, 2] # recall range doesn't include last
               # value given in range
```

So some more on map so here this is something that we saw earlier we can also like add these two essentially like, so here we specify two variables with X and Y, X + Y is the info here this sequence is assumed to be X, and this sequences are going to be Y, so you get like every pair added and that will be the result, so 1+ 4 is 5 2 + 5 is 7 and then 3 + 6 is 9, again here if you omit one of them say like 1, 2 and then this 4 5 6 then our answer will be like 4 I am sorry 5,7 and then 6.
Because third argument is as soon as, so it would not error out saying that there are two different sized arguments, it will just generate that this is result.
(Refer Slide Time: 30:16)



filter (function, iterable)

None

item for item in iterable if function item

- Like map, the filter function works on lists. Unlike map, it returns only a selected list of items that matches some criterion.
- Get only even numbers:

```
filter((lambda x: x%2==0), range(-4,4))
[-4, -2, 0, 2] # recall range doesn't include last
                # value given in range
```

4

So now the next one is filtered, so these are all like built in functions there are many, many, many built in functions in more in Python we will go through like some of them, let us see like I mean we may be able to go to some more later on, so the next one is the filter, the filter command essentially for it works on lists unlike map it returns only the selected list of items, so we saw the filter commands in TCL, to specify like I mean some way to restrict the data, so similarly like we can also use simple Python.

And then the filter command essentially like I mean it is, it is basically like so you can suppose the filters filter function and then they trouble, so this is a typical specification or the bar max or filter command, basically what it says is to construct a list from those elements of the iterable for which the function returns a true, iterable will must be either a sequence at container which supports an iteration or just an iterator.

The iterable is a string or a triple the result also has the same type, otherwise it always returns a list the function if the function is none basically here there is nothing none, then the identity function is not resume, and basically just copied that so the same thing is system return so all elements actually like I mean the all elements that are fall, they are just removed and then you get only the on the true, so that is something that you may want to know, so the other thing about filter is also basically when you specify like function and Google.

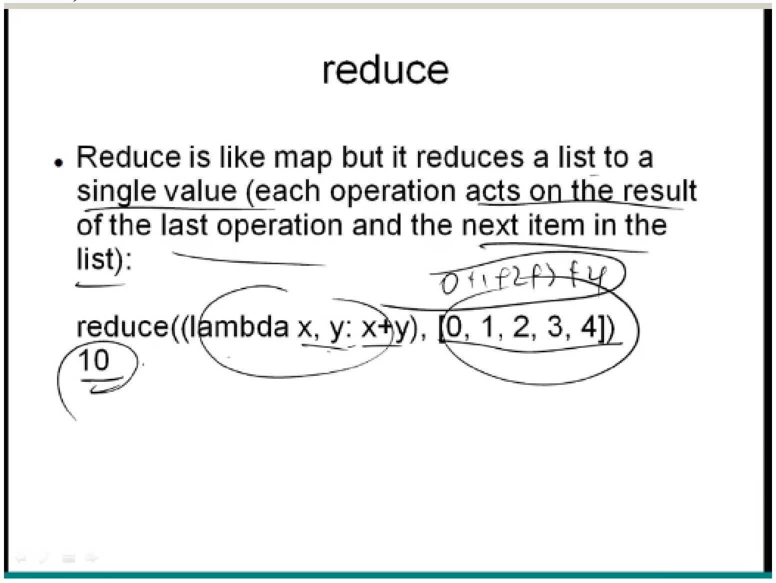
Is equivalent to item for item in iterable if function item ,so this is the same as item or item in iterable, if function item so you can think of filter that the shortcut for this command basically item for item in iterable if function item, so the function item returned to the true or false so if this is true then that becomes the item from four, and you iterate inside this iterable all, all the items look at the online.

So that is kind of you can you can think of this basically has oh it is actually written you and you can also add additional condition basically that is even say like if function is not none and the item for item in the iterable if item basically like so it also test for whether item is true or not if function is none okay, so that is the second condition that I talked about, so here we have an example it is basically filter lambda X, X you know this one basically if the, the remainder essentially link from that division is form equal to 0.

And then we give this range that is minus 4, 2, 4 that is your iterable the range is a literal and then the function is yet, so now we will see like I mean how this thing is, this thing works essentially like so it iterates from - this 424 and each one and then basically it towards by 2 and then compares the reminder, so whether that is equal to zero so for 4-4 the remainder is 0, so this becomes true and that is filtered out, next one is - 3 and the - 3 actually like now the remainder is not 0.

It is actually minus 1 so that is omitted and then when you go -2 again it is 0 for it pins 2 and then same like 0 and then 2, so we print out all the even numbers within this range, and if you look at this one this one thing is basically it is not the last one is not listed here, basically only up to it is not the poor is not there, that is because if you recall the range function does not include the last value given in the range, so it is like N – 1, so you may want to pay some attention to this essentially like when you are actually writing the program, because if you want to include that for then you need to extend the range past though before.

Refer Slide Time: 37:31)



reduce

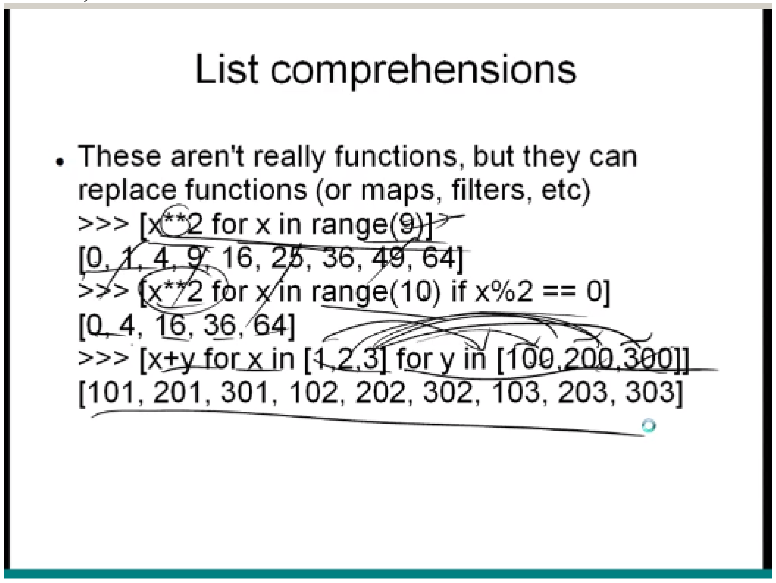
- Reduce is like map but it reduces a list to a single value (each operation acts on the result of the last operation and the next item in the list):

$0+1+2+3+4$

```
reduce((lambda x, y: x+y), [0, 1, 2, 3, 4])
10
```

So now we go to another function called the reduce, reduce is just like map but it reduces the list to a single value, and then this is using some operation, so again it is a function and the iterable,

but here the function is essentially like I mean this go ahead and add the or basically like a minute it is essentially like gives the last, so, so the so it reduces this list intellect distance single value, so each operation acts on the result of the last operation and then the next item in the in the list.
So here we say like lambda X Y X + Y so basically just first one is like I mean it is a give 0, so it is basically if 0 and then it adds + 1 + 2 + 3 + 4 and then the final result is 10.
(Refer Slide Time: 38:39)

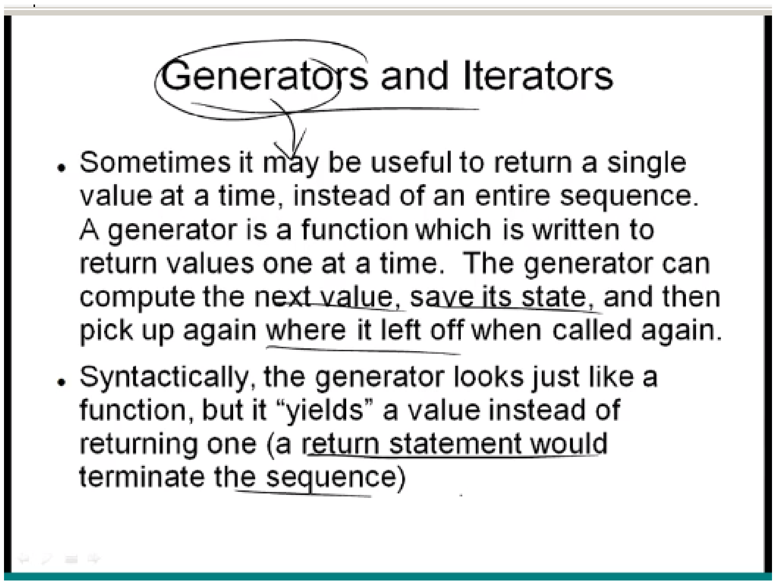


List comprehensions

- These aren't really functions, but they can replace functions (or maps, filters, etc)

```
>>> [x**2 for x in range(9)]
[0, 1, 4, 9, 16, 25, 36, 49, 64]
>>> [x**2 for x in range(10) if x%2 == 0]
[0, 4, 16, 36, 64]
>>> [x+y for x in [1,2,3] for y in [100,200,300]]
[101, 201, 301, 102, 202, 302, 103, 203, 303]
```

So now let us see some the list comprehensions, and these are really not functions but they can replace function basically maps filters exit form etcetera, so essentially we can specify the list s essentially leveling X power 2 or X in range 9, so it gives you all the square numbers from 0 to 8 as you know like a 9 is excluded from the range, so that is what you will get, and then we can also like to get the idea of actually like a square number and then which is also divisible by 2, and or even square numbers.
That for that we just specify this, this is like all the way to up to nine so it brings a 0 for 16, 36 and 64 to mix 1, 9, 25 and 49, so essentially I can, so it is like functions that you are putting it inside a list but it is expanded and difficult expand it as a function and then the bigger the real operation that goes on, now the other one is X plus y 4x in 1 2 3 and 4 y in 100 200 300 the result is basically with adds of X + Y for each of the X, so 101, 201, 301 then 102, 202, 302 and then finally 103, 203, 303 so this is the result.
(Refer Slide Time: 41:12)



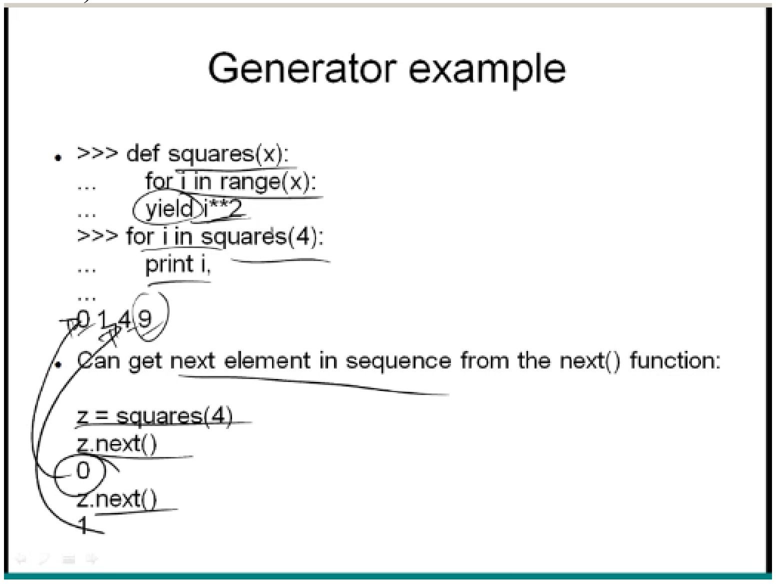
Generators and Iterators

- Sometimes it may be useful to return a single value at a time, instead of an entire sequence. A generator is a function which is written to return values one at a time. The generator can compute the next value, save its state, and then pick up again where it left off when called again.
- Syntactically, the generator looks just like a function, but it "yields" a value instead of returning one (a return statement would terminate the sequence)

So we, we talk about the iterables, so there is a concept of generators and Iterators sometimes it may be useful to return a single value, at a time instead of the entire sequence, so for that we use the generator essential generator is the function which is written to return values one at a time, the generator can compute the next value and save its state and then pick up again there is left of main called again, so syntactically the generator look just like a function, but it yields a value instead of returning one.

So and a return statement will actually terminate that sequence at all, the sequence all together, one until we see a written statement basically keeps that single value around.

(Refer Slide Time: 42:13)



Generator example

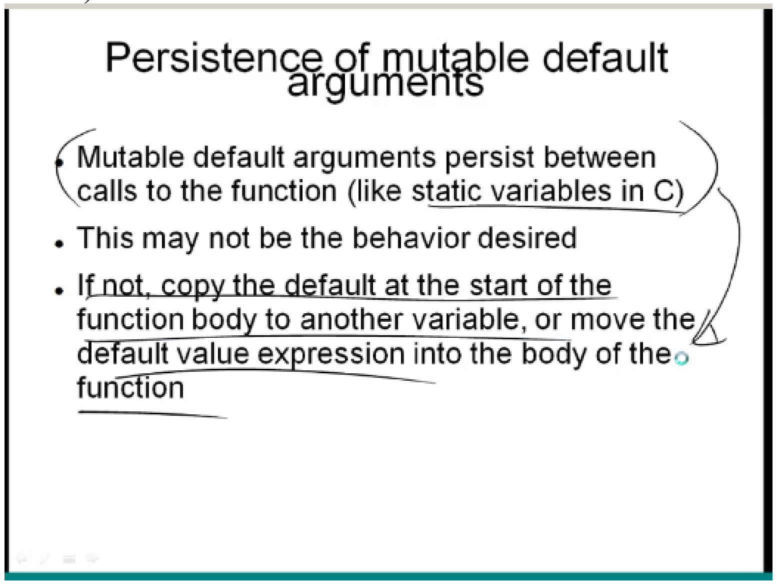
```
>>> def squares(x):
...     for i in range(x):
...     yield i**2
>>> for i in squares(4):
...     print i,
...
0 1 4 9
```

- Can get next element in sequence from the next() function:

```
z = squares(4)
z.next()
0
z.next()
1
```

so here is a generator example so we define squares essentially then for I in range X yield that is the new one with the generator, and then I pod and then we basically use this and so for IM

squares X squares for print I, so now it is going to print just one value at a time essentially that is 0, 1, 4, 9 and it keeps the last value at this point and then we can get the next element in the sequence from the next function, Z squares 4 and the first one and first time next when we use it, it becomes 0 which is the first one and then that we next will become one, and then so and so.
(Refer Slide Time: 43:17)



Persistence of mutable default arguments

- Mutable default arguments persist between calls to the function (like static variables in C)
- This may not be the behavior desired
- If not, copy the default at the start of the function body to another variable, or move the default value expression into the body of the function

So now the position of the mutable default arguments, so the default arguments which are mutable persist between the function calls so essentially, so these are like static variables so if they are like mutable default arguments they continue between the functions, and this may not be the behavior that we want, so if we do not want this behavior, we need to copy the defaults at the start of a function body to another variable, or move the default value expression into a body and the body of the function.
So this is basically like something that you may want to pay attention again this is another code of python, because of this you want to do this okay so now that finishes this, this lecture so in this lecture we actually went to a couple of things one is mainly the lambda function, then we also looked at some of the default functions, that are available inside or Python, the few ones are like the apply which is kind of deprecated right now in just the function call just like what is specified here.
As directly and then the map function then we also look at the filter and then reduce, we also kind of learned how to write functions in form of lists basically if they are not less than we actually write functions and be this comprehended as a function, and it is applied as a function, so we can actually like write these kind of commands into a list syntax, and then call it as a

function, and then finally we actually learnt about the generator as a concept we still need to look at the Isolators which we will do the majesty.

But so and then we saw this example of how to use the generator, and then finally one of the rules of python is essentially like any kind of form default arguments, they persist between the function calls if they are mutable, so if you do not want this behavior then we need to copy the default and the start of the function body to another variable, all we have to move the default value expression into a body of a function, so inside or outside change to another video, so that is pretty much the lesson for today basically the lecture and again we will pick it up in the next class thank you.