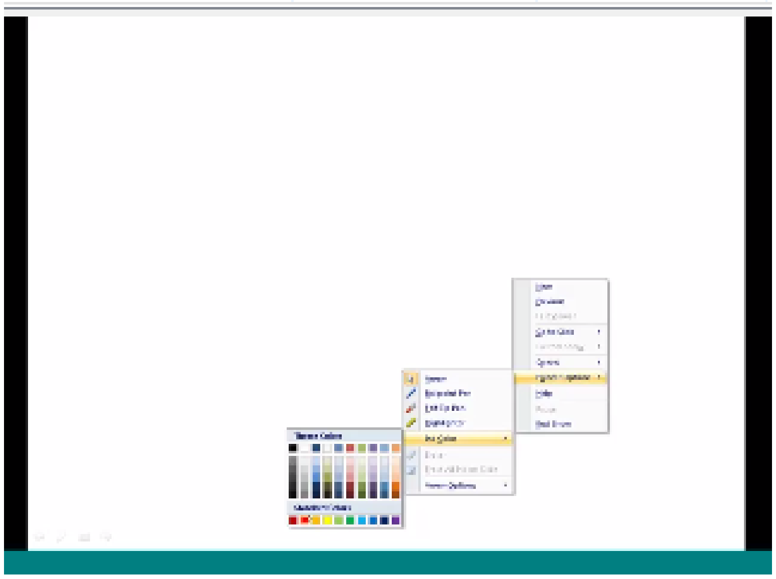
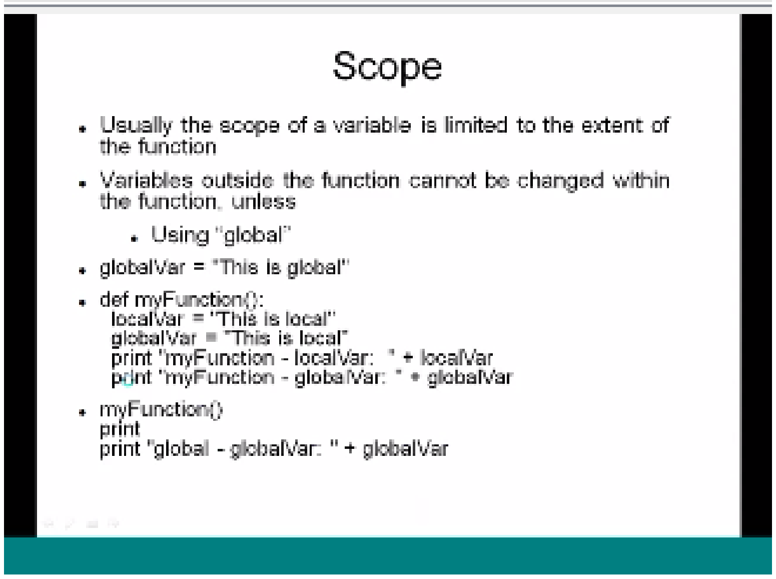
(Refer Slide Time: 00:02)



Again welcome to discussion hi everyone welcome to this session again we will be continuing our look into the Python programming. Last time the last lecture we introduced the Python programming today, I will continue in my the topics that I will be talking about I mentioned one thing which I did not cover during last time. So I will plan to cover that + some more topics for today so last time I talked about the scope of variables in Python.

The scope is very similar to what we have for tickle. So I just wanted to just give you an example.

(Refer Slide Time: 00:57)



Scope

- Usually the scope of a variable is limited to the extent of the function
- Variables outside the function cannot be changed within the function, unless
  - Using "global"
- globalVar = "This is global"
- def myFunction():

```
  localVar = "This is local"
  globalVar = "This is local"
  print "myFunction - localVar:  " + localVar
  print "myFunction - globalVar:  " + globalVar
```

- myFunction()

```
print
print "global - globalVar: " + globalVar
```

So first of all the scope of a variable is limited to the extent of a function so using a particular variable. That variable does not go outside the function once you make without the function that variable is gone pretty much. And the variable outside the function, are not also taken into the function. So if you are changing any variable which was declared outside the function in within the function the Python creates a brand new variable form even though it is the same name and just overloads the name.

And then it assigns the value what you are assigning and once you leave the scope of function that particular variable is gone. And it uses the global variable once again, so if you try to change anything within the function and then size print outside you will not see any changes it will this be the same of people. So here is one quick example I am going to pause it for a minute okay is limited to the extent of function. Once you change the variables essentially ligaments or any variables outside the function cannot be changed within the function.

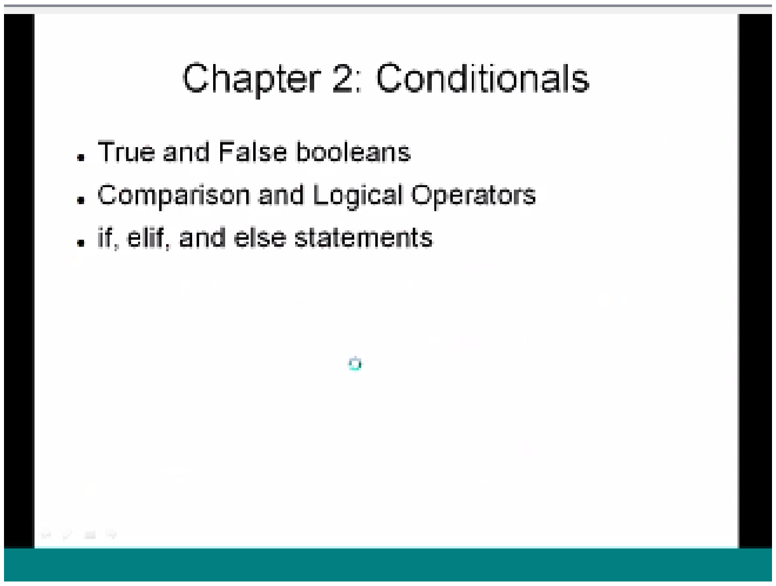
So Python usually assigns a new variable or the same name it overloads that name and then uses that within the scope and then once you go out again you do not feed or stakes. And the same thing, so if you want to have a variable defined outside and then you want to use it within the function you need to declare that as with global keyword. This is shown here essentially, so it is the global diamond, so here is one example and this section is actually a left Python code.

Let us review this one so here, I am declaring a global variable is called global war and then it is assigned to a string this is global. Now I am defining my function where I am defining a local war called and we basically assign this is local to that local war and then global war I am changing from that this is global to this is local. So if I print, within the function it will be still the same the global war will be same as this is local and the local war is also the same. So that is when I exit with this now if I print outside the global war in a tell me lag event water it will print.

It is still going to print this value as if it never got changed by my function this is again because when you define it. Here this global war is a different global war it actually creates this variable assigns the local this is local. And then it prints out this but once you go out it uses this so in order for us to change the global war within the function. We need to declare here global and then global war so this basically like this statement enables Python to actually get through this issue and then actually change battle this is local and that is visible outside the function.

So now when you print this one instead of this is global it will print the fist local so that explains the scopes of variables are the variable scope.

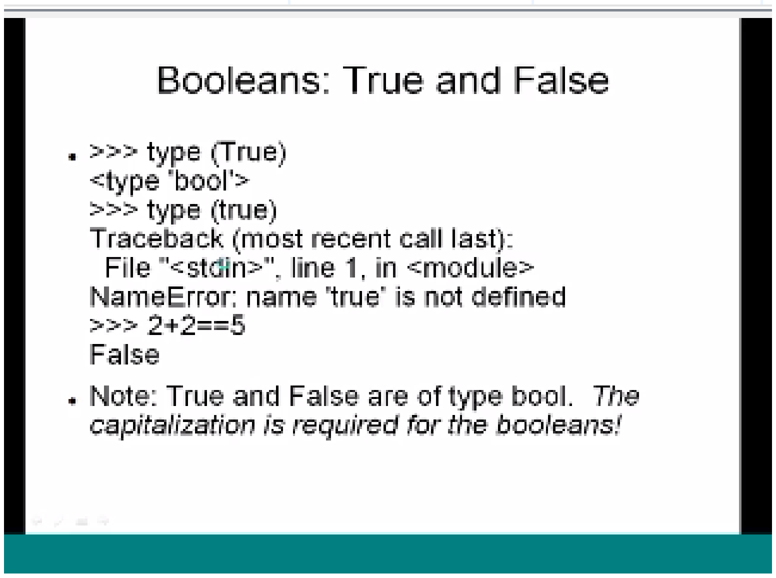
(Refer Slide Time: 05:26)



Chapter 2: Conditionals

- True and False booleans
- Comparison and Logical Operators
- if, elif, and else statements

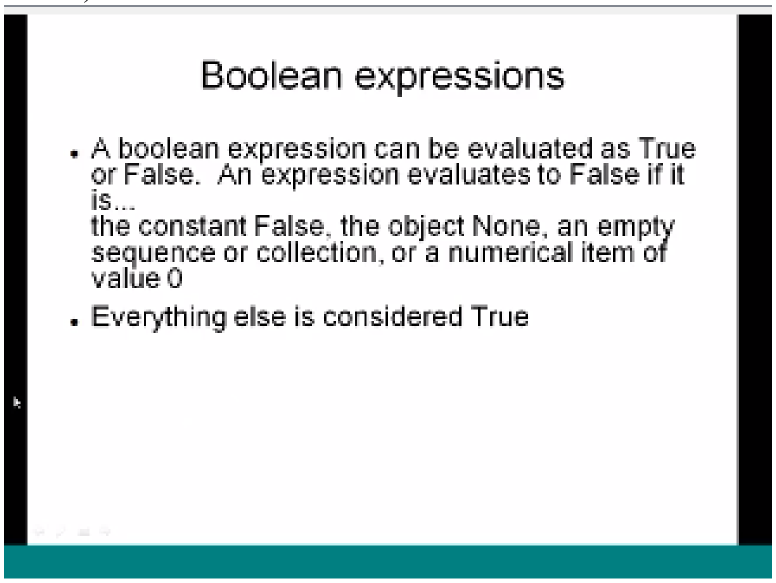
So today we will be continuing our discussion so just a recap in the first lecture we saw several topics basically. I just want this run to several of them I hope you remember all these things we started with like some basic stuff how to run Python programs. We talked about variables you talked about how to print and then what are the operators in Python some of the band keywords also like that we talked about and then. What are the rules for having it keyword then?
We talked about how can we input a variable into the into the in Python or into the program and then how to write comments and then now today we saw the scope. Okay so I hope those things are very clear for you now let us go to today's lecture. Basically we today we are going to talk about the conditionals basically true or false Boolean you truer for Boolean comparison and logical operators. And then we are also like going to look at some of you for if L is students.
Again remember Python is a shorthand programming language we want to minimize and get the most effective things in Python.
(Refer Slide Time: 07:24)



Booleans: True and False

-
  ```
  >>> type (True)
  <type 'bool'>
  >>> type (true)
  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  NameError: name 'true' is not defined
  >>> 2+2==5
  False
  ```
- Note: True and False are of type bool. *The capitalization is required for the booleans!*

So here, the main thing is like that type is what is called the main thing and type Boolean or Boolean and then we say like type true that is of type Boolean it does not matter like I mean if you specify the lowercase T here that is not defined. So essentially like that just does not in error so the Boolean are defined as per case TRUE and is FALSE. So make sure that you remember the case and use that. And if you specify something like this 2 + 2 = 5 this is a conditional checking basically whether the same and then the result is a false.
So the true or false are of type Boolean and the capitalization is required Boolean. So just remember that you.
(Refer Slide Time: 08:57)

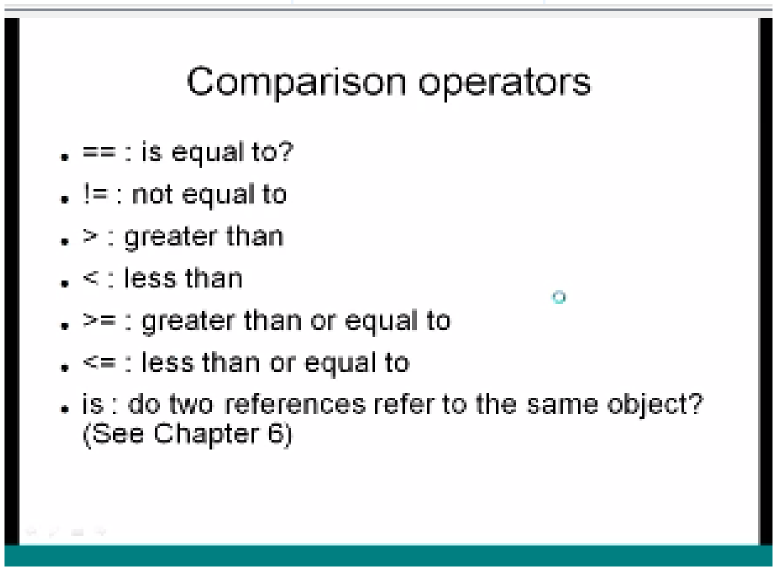


Boolean expressions

- A boolean expression can be evaluated as True or False. An expression evaluates to False if it is...
  the constant False, the object None, an empty sequence or collection, or a numerical item of value 0
- Everything else is considered True

A Boolean expression is an expression can be evaluated true or false the expression evaluates to false if it is the it is a numerical value of 0. Essentially given there are different ones one is the

constant false the object none and empty sequence or a collection or the numerical item of value 0. Think of this basically like a mean this is the primary one you have also the constant false. That we saw earlier they evaluate the Boolean expression evaluates to false then also = to answer calls the object is none it again written sub calls as an empty sequence.
Are a collection which opens all and then finally any numerical value of 0? Is also a possible forms and everything else other than these four conditions it is actually it is true for the Boolean expression.
(Refer Slide Time: 10:03)

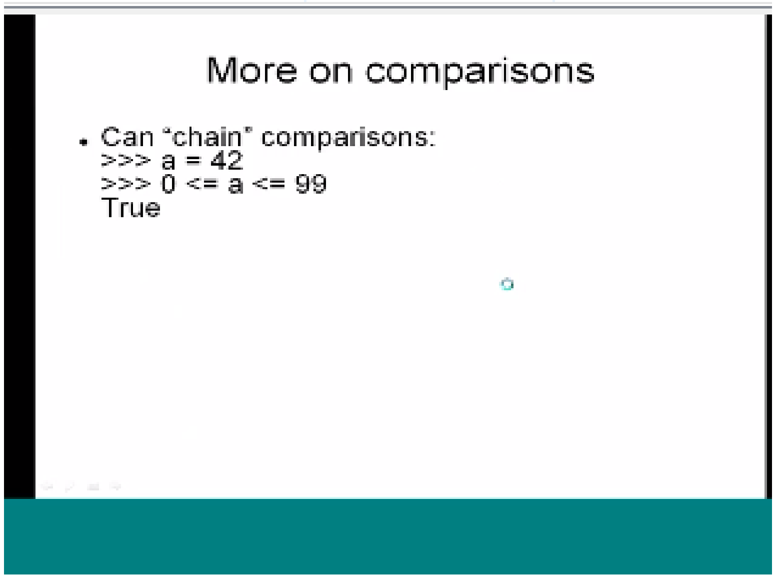


Comparison operators

- == : is equal to?
- != : not equal to
- > : greater than
- < : less than
- >= : greater than or equal to
- <= : less than or equal to
- is : do two references refer to the same object? (See Chapter 6)

So let us look at some of the comparison operators so this one we saw in the first example that is = that is actually it is a comparison to find out whether two things are =. And then the not = to is again it is a easy one it checks if the two values two value operands are = or not and these values are not = then the condition becomes true. So in other words, so this one is evaluates to A = = B you can say basically, and then if this basically and then this basically like whether checks whether it is true or not in their = or not.
Here, it is basically like A not = to P and if that becomes true then the Institute and then you have a > and you have < and > or = to is another operator and < or = to is and go on then we also have the similar kind of operator. That we saw in more the tickle or even in Pearl which is like a board operator this essentially checks if the value of two operands are = or not if the values are not = then the condition becomes 2. So in other words this is similar to the not = to anything.
And then we also have something ease the two references refer to the same objects this one we will cover in sub 6. So wait a couple more lectures and then we will go through that basically this

is like one level of indirection in probable references address is the same you know values need before. So that is another thing that find out.
(Refer Slide Time: 13:20)

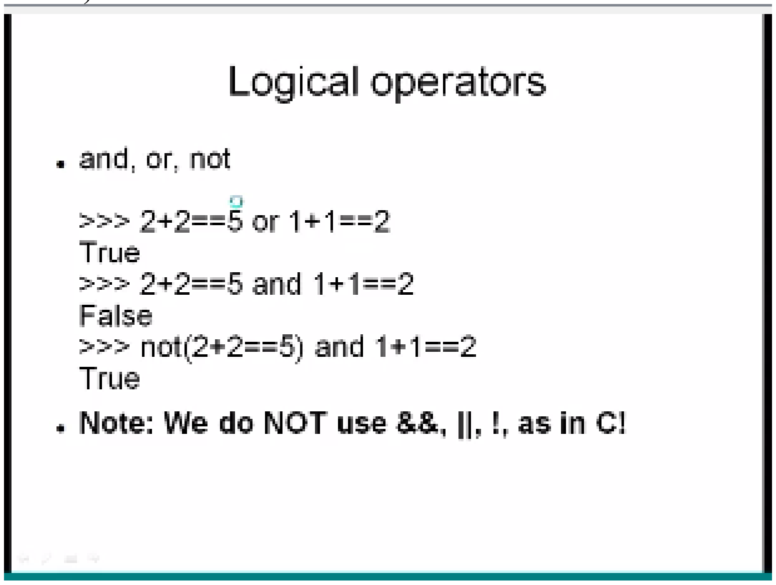


More on comparisons

- Can "chain" comparisons:

```
>>> a = 42
>>> 0 <= a <= 99
True
```

And in Python actually one thing that you can do is you can actually chain these comparisons. So you can define like the multiple equality must be = these multiple comparisons in the same line mean this goes to that shorthand notation that talked about. So here the simple thing basically 0 < or = to a < or = to 99 and if a 42 this expression will return true. But if a is say minus 10 this evaluates to false even though like this side will be true. And if it is 100 then also it returns false because this will evaluate true but this side is going to be false. So you can you can actually specify these shortcuts this actually saves a lot of time and of code.
(Refer Slide Time: 14:12)



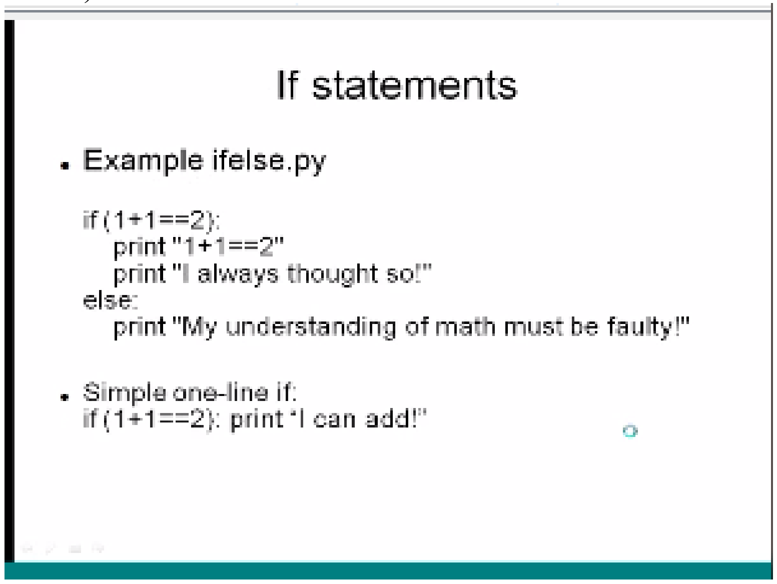
Logical operators

- and, or, not

```
>>> 2+2==5 or 1+1==2
True
>>> 2+2==5 and 1+1==2
False
>>> not(2+2==5) and 1+1==2
True
```

- **Note: We do NOT use &&, ||, !, as in C!**

Now let us look at some other operators like logical operators are and, or not essentially. So the and, or, not is essentially like through the real typing the numbers typing the actual words themselves out. So we can specify say like two + two = = five R one + one = = to this will return true because this side is true and essentially like I mean this may be false but false or true is actually true. If you say and now false and true that is actually false, then you can also specify not of this which means basically like this is now true.
And then this and this and that become true as well so one thing you note that actually we do not use the ampersand bar bar this bang S and C. So we do not do the C level stuff for using the logical operators. They are explicitly specified in the list form.
(Refer Slide Time: 15:47)


If statements

- Example ifelse.py

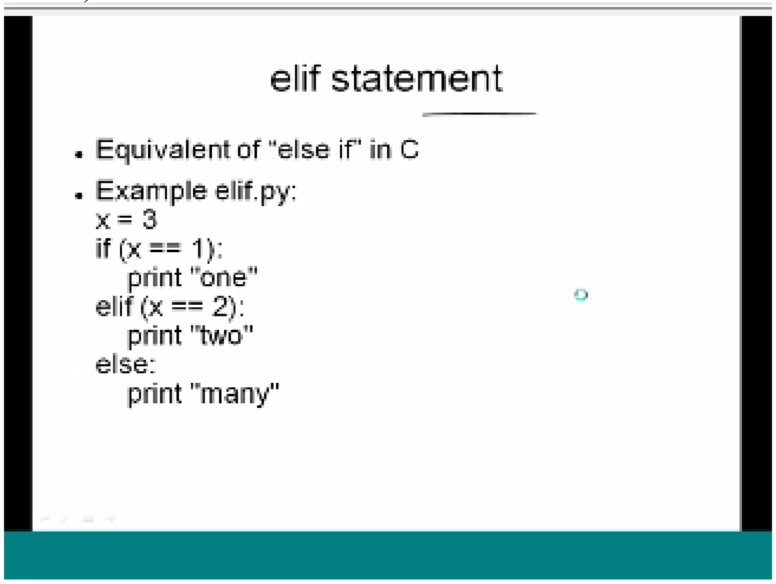
```
if (1+1==2):
    print "1+1==2"
    print "I always thought so!"
else:
    print "My understanding of math must be faulty!"
```

- Simple one-line if:

```
if (1+1==2): print "I can add!"
```

So even though we said basically we would not specify the form the double ampersand and all these things. There will be like single ones that are that are used which are mostly used for the bitwise operators. So we will talk about that in a little later. Now we go to the conditional ones if then else so here just one example on if else the pie so you can think of you can see this code actually which is 1+1 , 2 then print that and then print I always thought so and else bring to my understanding of the math is so not be faulty.
And a simple one-line ad is useless so we notice that X will go for column here or both if and else and which is quite important then you specify the if statement. The simple one will be in basic mail again with one and then again I can add variable one so all these cases actually like I mean it evaluates true. So the first set gets executed and then the second set is ignored. You feel like two things one is I mean of course this column here and then there is an indentation here in it for spin statements in else the indentation is quite important in Python.

The indentation gives it the frameworks of what is the it is almost similar to a function call or pick a function statement.
(Refer Slide Time: 18:39)

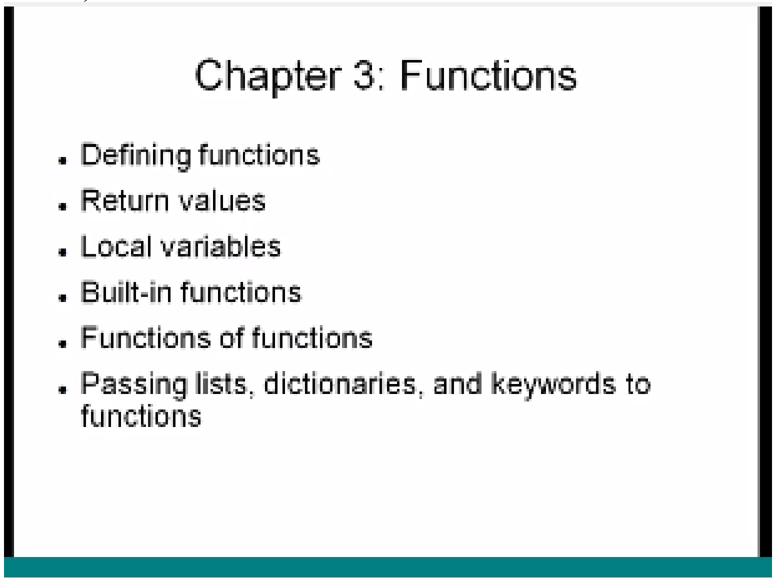


elif statement

- Equivalent of "else if" in C
- Example elif.py:

```
x = 3
if (x == 1):
    print "one"
elif (x == 2):
    print "two"
else:
    print "many"
```

The ellif statement is actually it is very similar to else if in your else is in tickle or even pearl so here is a simple one there X = to = 1 then pin 1 is = = to 2 LLL save or LS then - and then else print me so it is 2 then this particular thing catch and then preventer. So now let us go into the forum the next level.
(Refer Slide Time: 19:45)

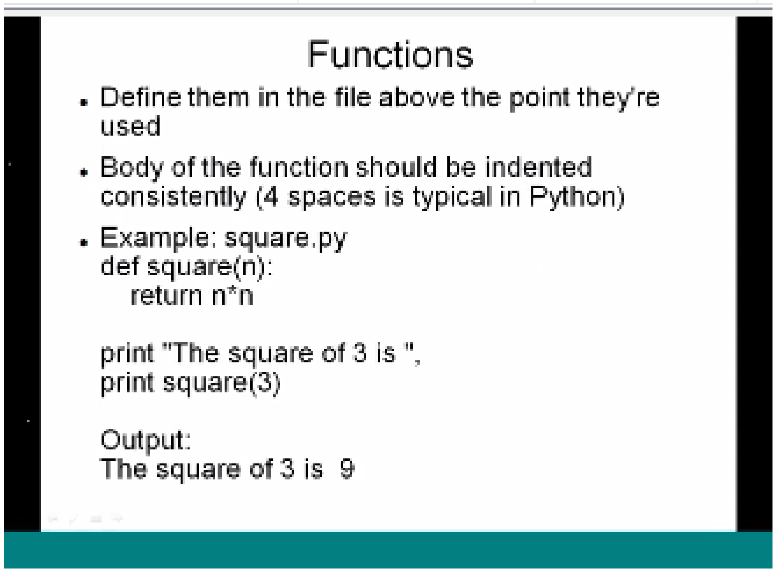


Chapter 3: Functions

- Defining functions
- Return values
- Local variables
- Built-in functions
- Functions of functions
- Passing lists, dictionaries, and keywords to functions

So a one thing on the ellif is basically like you can have as many elifcess possible in this one that is A first one colleen and LS second condition of the Collins. And then the maybe several eliefs before can actually go to else. So this is one thing that you may want to this say me something.

Now Python also supports loops which we will talk about in the next section there are while loop and for loop are supported, so we will talk about that. Now let us look at functions.
And in function we will talk about how do we define function we already tall like some of the basic basics of functions. And then we also see how to turn values local variables which we will cover once again the scope and then some built-in functions some functions of functions and then how do we pass lists dictionaries. And keywords and into the function so let us begin.
(Refer Slide Time: 22:09)



Functions

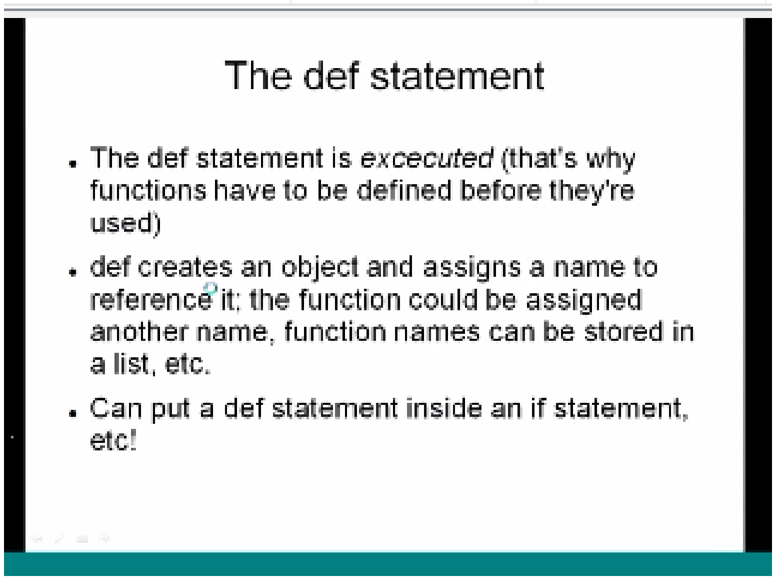
- Define them in the file above the point they're used
- Body of the function should be indented consistently (4 spaces is typical in Python)
- Example: square.py

```
def square(n):
    return n*n

print "The square of 3 is ",
print square(3)

Output:
The square of 3 is  9
```

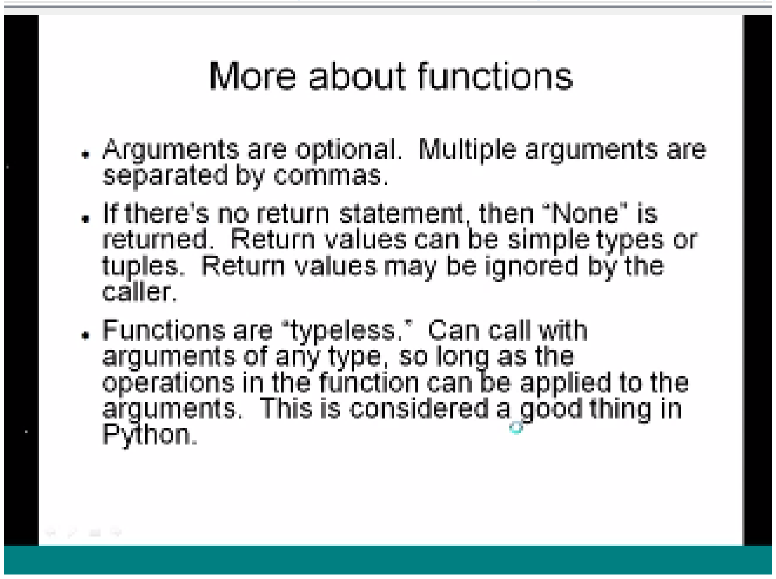
The functions basically you define them in a file about the point their value. So you need to define it before you are using the main reason is python is interpretive language. So you need to know what it is before you can use it. And the body of a function should be intended consistently so for spaces is typically by console within 4b increment. So here, we declare a function called and you can see basically like it is a definition the function name followed by column and then we leave four spaces and then we basically define the function.
So the usage will be you can say like print square of three is and then call go square functions for a three and then doubt will be an X square of 3 is 9. So couple of things one is that DEF statement, which is essentially the first one for stable function how we define it the DEF statement is executed.
(Refer Slide Time: 23:52)



The def statement

- The def statement is *excecuted* (that's why functions have to be defined before they're used)
- def creates an object and assigns a name to reference it; the function could be assigned another name, function names can be stored in a list, etc.
- Can put a def statement inside an if statement, etc!

Basically it is not like a parse and thrown away basically it is executed and that is why we need to define the functions. Before we can use them and what happens is when the def gets into the DEF creates an object and assigns a name to reference it. The function could be assigned to another new the function names can be stored in a list and we can put a def statement inside and if statement and it can be like a nested then the basically like solve.
(Refer Slide Time: 24:33)

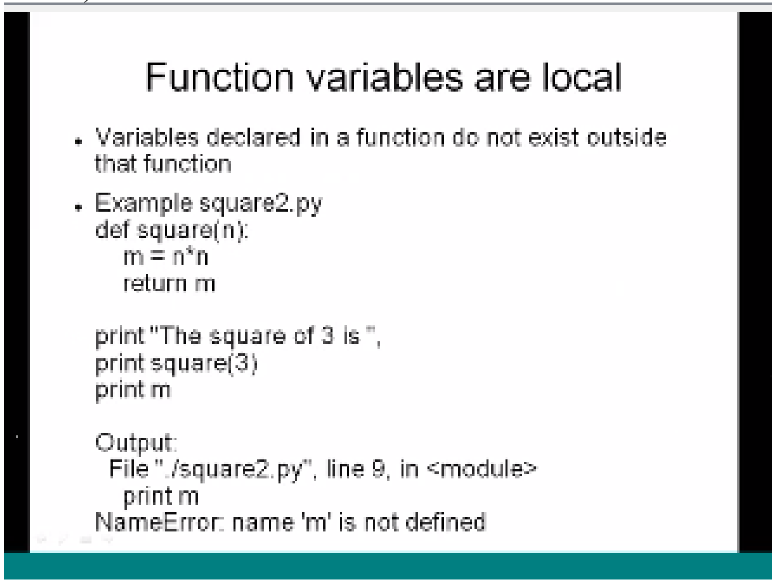


More about functions

- Arguments are optional. Multiple arguments are separated by commas.
- If there's no return statement, then "None" is returned. Return values can be simple types or tuples. Return values may be ignored by the caller.
- Functions are "typeless." Can call with arguments of any type, so long as the operations in the function can be applied to the arguments. This is considered a good thing in Python.

And more this ability to follow you arguments to a function optional if you have multiple arguments in basic accepted Commerce. And if there is no return statement then none in certain the return values can be simple types are tuples or basically two element array this selector A , B you and with a function returns the values they may be ignored by the color itself. One thing to

note is the functions themselves all type less that means that we can call it call with arguments of any type as long as the operations in the function can be applied to those arguments.
(Refer Slide Time: 26:19)



Function variables are local

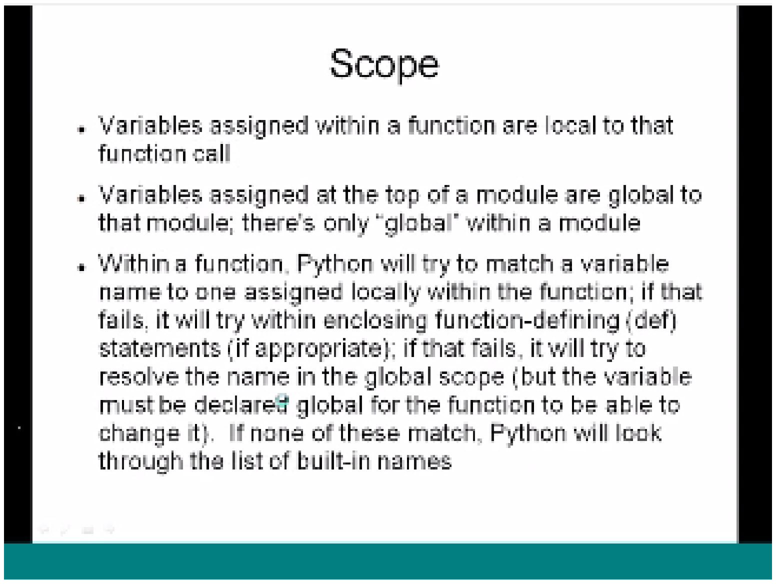
- Variables declared in a function do not exist outside that function
- Example square2.py

```
def square(n):
    m = n*n
    return m

print "The square of 3 is ",
print square(3)
print m

Output:
  File "./square2.py", line 9, in <module>
    print m
NameError: name 'm' is not defined
```

And this is considered as a good thing because now you do not have that dependency of the type of the argument anymore. Now this is something that we saw earlier as I mentioned variables declared in a function do not exist outside the function. So here is one example def square n which is M is defined as in then return M. So if you say LX square of three and then you class blueprint for a three and then you print M Python will generate this error.
So this is the opposite of what we saw in the first slide regarding the scope because this M does not live outside of this function which is these two statements. So it does not it cannot actually live also we can capture the value of square three because it is returning the M we can capture this as this is a go variable called M itself M = square 9 square 3. And then it will be there but this way even though this can actually this will work this does not work. So file comes up an error.
(Refer Slide Time: 27:31)



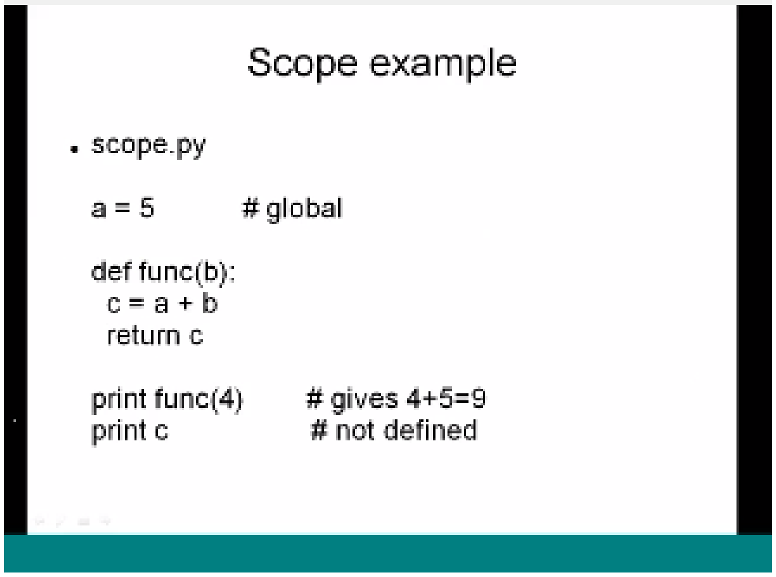
Scope

- Variables assigned within a function are local to that function call
- Variables assigned at the top of a module are global to that module; there's only "global" within a module
- Within a function, Python will try to match a variable name to one assigned locally within the function; if that fails, it will try within enclosing function-defining (def) statements (if appropriate); if that fails, it will try to resolve the name in the global scope (but the variable must be declared global for the function to be able to change it). If none of these match, Python will look through the list of built-in names

The variables are signed within a function are local to that function call that double saw and then variables are signed to the top of the module are global to that module. There is only global within a module, so this is the use of global to represent the global value of form the way. So within a function Python will try to match a variable one to one assigned locally within the function if it fails it will try within the enclosing function that defining statements that is DF if appropriate.

If that also fails then it will try to resolve a name in the global scope if none of this match then Python will look through the list of built-in names.

(Refer Slide Time: 28:41)



Scope example

- scope.py

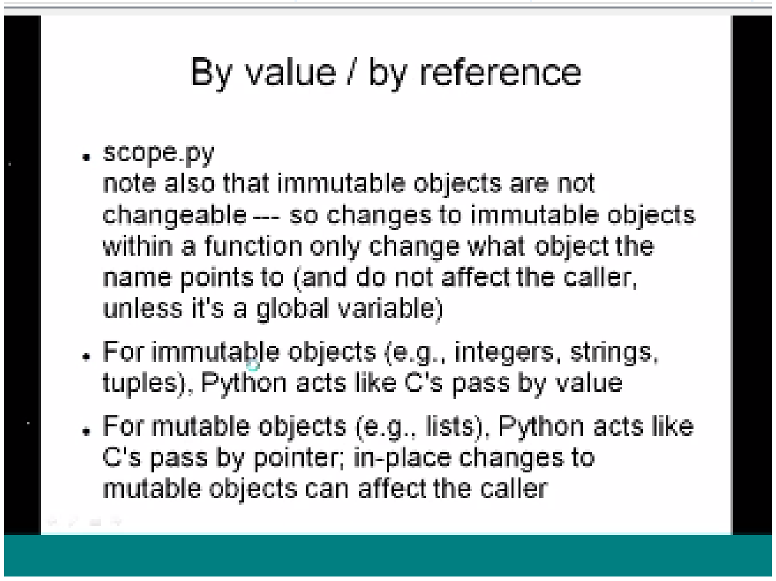
```
a = 5          # global

def func(b):
  c = a + b
  return c

print func(4)     # gives 4+5=9
print c           # not defined
```

So what are the built-in names that we will see so on the scope essentially Legume in another example here, A is 5, C is a + B and then within C then if you say funk for order hit return. And

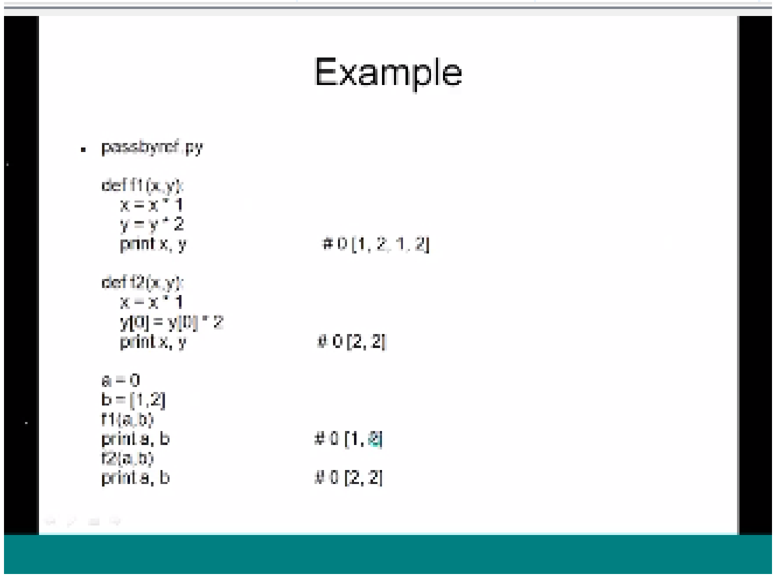
we say print see is not blue then it does not even but it basically like pins 4 + 5 is 9 so within the function we can still make it print but once it is outside the function it becomes a non-event. So here is another example A is = to 5 and then we say like global C and then C is a + B general return 5.
So outside when we say funk 4 it is actually now use the result and then if you say like the print C then now it is also defined it is defined to the 9. So this global variable actually helps in taking out some of the deficiencies earlier. And notice the racks in the evaluation takes place within function mode as this I addition and then sends that to C.
(Refer Slide Time: 30:23)



By value / by reference

- scope.py
  note also that immutable objects are not changeable --- so changes to immutable objects within a function only change what object the name points to (and do not affect the caller, unless it's a global variable)
- For immutable objects (e.g., integers, strings, tuples), Python acts like C's pass by value
- For mutable objects (e.g., lists), Python acts like C's pass by pointer; in-place changes to mutable objects can affect the caller

So unlike other programming languages like Perl Python in a in Python everything is my reference but please also note that there are mutable objects that are not changeable. So changes to immutable objects within a function only change what the object name points to and it does not affect the value. And it also does not affect who calls the program so the immutable objects like integer's strings tuples. Python acts like SC act like the sea stars the value. And for mutable an object that is example it lists Python acts like a cease pass by pointer. In place changes to the mutable object demo it compilations.
(Refer Slide Time: 31:33)



Example

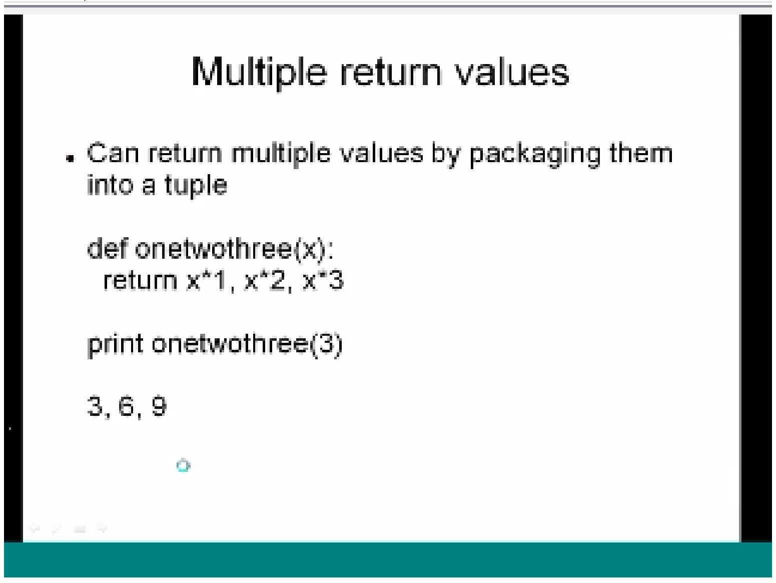
- passbyref.py

```
def f1(x,y):
   x = x * 1
   y = y * 2
   print x, y                 # 0 [1, 2, 1, 2]

def f2(x,y):
   x = x * 1
   y[0] = y[0] * 2
   print x, y                 # 0 [2, 2]

a = 0
b = [1,2]
f1(a,b)
print a, b                    # 0 [1, 2]
f2(a,b)
print a, b                    # 0 [2, 2]
```

So here are some examples one is the pathway records here we define the function f1 X and purpose of X and Y and then we just print out 1 + 1 is + 2 + 1 and whether one is posting and then we just print out all the values. Now we take the next function x f2 and we declare x = to x times 1 + y 0 is y 0 x 2. So this is presumably like a move or an array function. So now when we ask you to print out the figure eight pins are zero and then A 2/2 now we continue basically we make it = to 0 B is 1 and 2 again it needs to be in array for it.
We use inside then F 1 a B that will be a Paul so that returns this value and then actually this and strangle # so 2. And then we say basically like print a B and then difference this and then we use apply the second function it is been 0 and 2 inch. So here, we are just passing the repine to the point to the function and then getting back many things.
(Refer Slide Time: 33:36)



Multiple return values

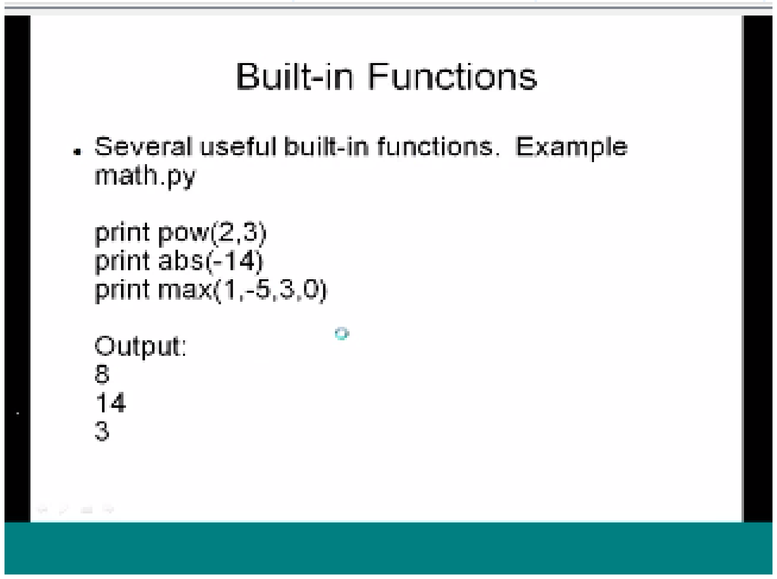
- Can return multiple values by packaging them into a tuple

```
def onetwothree(x):
  return x*1, x*2, x*3

print onetwothree(3)

3, 6, 9
```

And multiple values can be returned by a Python program but usually they happen with packaging int into a tuple or array. So there is a small packet basically 1 ,2 , 3 X times 1 x times 2x times, 3 and then if you print the 1, 2 ,3 program with the value of 3 then now there is a gulf between 6 more. Because now you can return 3 values instead of 1 or 2.
(Refer Slide Time: 34:22)



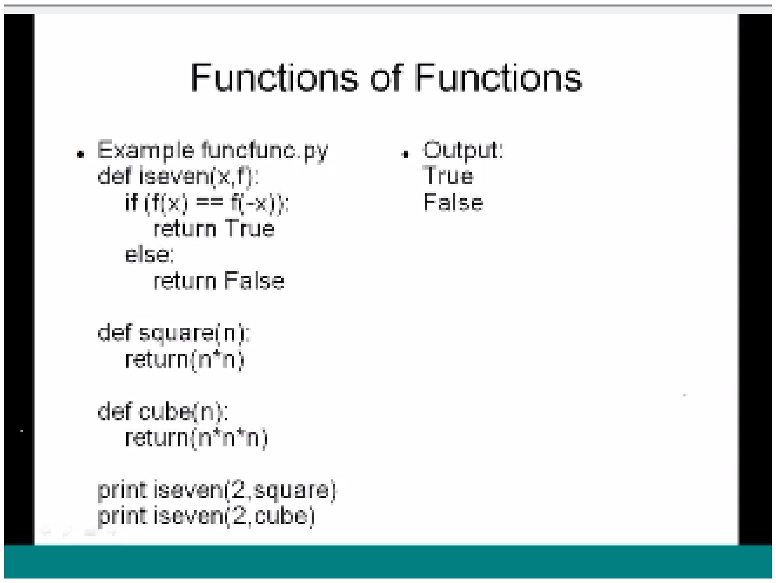
Built-in Functions

- Several useful built-in functions. Example math.py

```
print pow(2,3)
print abs(-14)
print max(1,-5,3,0)

Output:
8
14
3
```

Now Python has a rich set of built-in functions for example math dot PI installation we should be able to see there are several functions available there are all pretty much free why the point is for POW. So when we say like POW 2, 3 that is basically like the stands for power and basically it computes the so if you say like I mean this is A and B this computes A ^ B and a B S stands for absolute. So this is - 14 and then these are some of the nifty commands find the maximum or amongst these numbers and then here without says 3 it is the highest integer then. So these are some of the fun things to do with Python for your entities.
(Refer Slide Time: 35:35)



Functions of Functions

- Example funcfunc.py

```
def iseven(x,f):
    if (f(x) == f(-x)):
        return True
    else:
        return False

def square(n):
    return(n*n)

def cube(n):
    return(n*n*n)

print iseven(2,square)
print iseven(2,cube)
```

- Output:

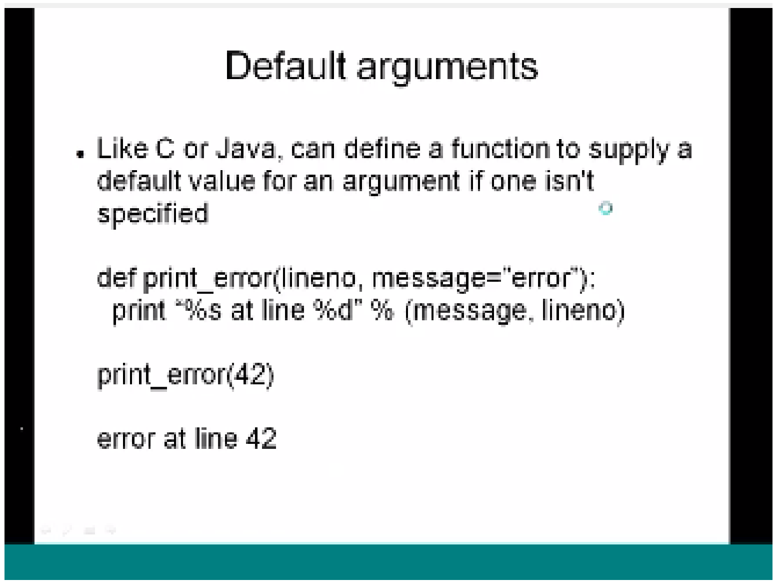
```
True
False
```

Now a function of a function essentially you so here is even is a function basically this is just X M F and then we do a test basically for X is the same as f - X then we have been through else given false. And how do we determine this is because we now go square missing so we are depending on under function call square N and then which returns basically this the multiplication. And then we also define another one for cube and then that returns and n times n times N.

So now we said okay print even two square and then print is even 2 cube you so the output of these two are the first one is it true and then the second one is a false because it simply basically. So first thing to notice this is one function and basically the top log function and here essentially always say X is so same as F - X then we will go through and Fox now when we talk about in square this will give you only return and n or n times n and the same thing here. So now if you look at this one actually this is a function of a function.

Because you are calling the easy one with an argument to and then basically also we are calling this one. So in four squares you need only one model so basically what happens is it knows that okay it is a square and it passes this number into that square abused answer and two becomes X square becomes F and then it is again the is even is evaluated with these numbers and then possibly this answer.

(Refer Slide Time: 38:18)



Default arguments

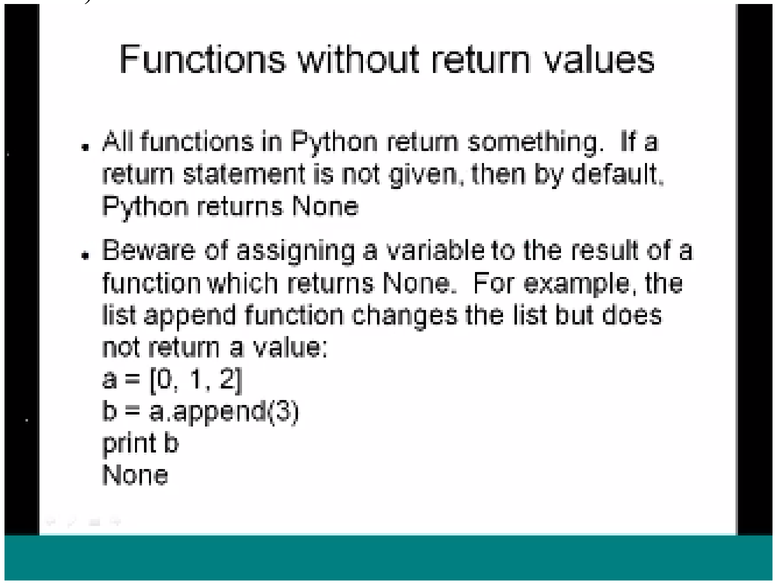
- Like C or Java, can define a function to supply a default value for an argument if one isn't specified

```
def print_error(lineno, message="error"):
  print "%s at line %d" % (message, lineno)

print_error(42)

error at line 42
```

So now some default arguments like in C and Java we can define a function to supply the default value for an argument if one is not specified. So these are useful for printing out the error messages and things like that. So here, we have defined a function or print error and it gets line number and some message the message is error. Already negative speed sign here so and then we just print basically the error message error at line number so and so and then that line number. So when we say like print error 42 it just stands physical error at magnitude.
So these are like predefined a value that you can supply it into a function and essentially they can that is what we do.
(Refer Slide Time: 39:26)



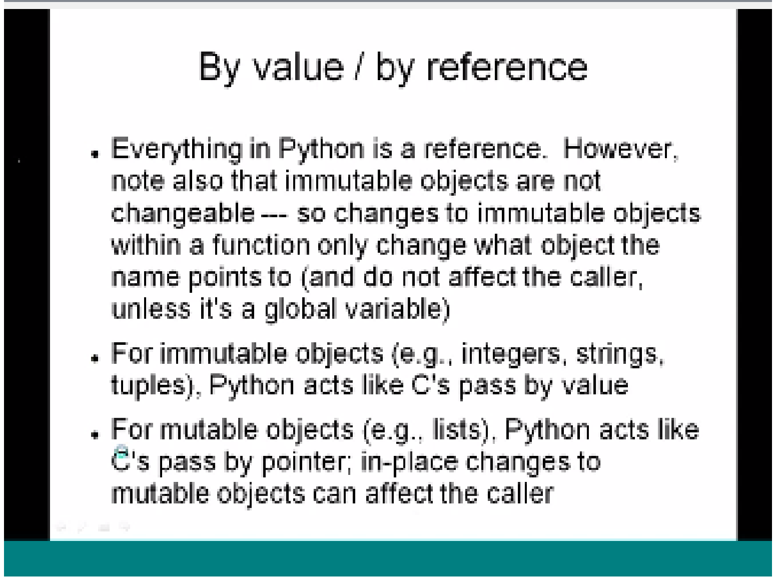
Functions without return values

- All functions in Python return something. If a return statement is not given, then by default, Python returns None
- Beware of assigning a variable to the result of a function which returns None. For example, the list append function changes the list but does not return a value:

```
a = [0, 1, 2]
b = a.append(3)
print b
None
```

So now let us talk about the function code in return value in general all the functions in Python return something if the return value is not given then by default Python returns none. So this is a

principle difference between Python and tickle as you know tickle actually returns last evaluated expression. So when you are writing like Python code the way of assigning a variable to the result of a function which returns number for example here.
The list append function changes the list but it does not return a value so when we say basically likes okay. A 0, 1, 2 and then we say this way about append 3 and that is a sign to be and it says that be it since done even though this is executed incorrectly and basically like the move this will be the 0, 0 , 1, 2, 3 but it only prints none so you may assume that no now that this is correct it should get a value of true but the actual value that you get this one is now. So this is basically like what I wanted to cover this lecture.
So we talked about several things I hope you remember we talked about the scope of the variables. We talked about the conditional system see that is what we started with basically fluent or true and false Boolean's then. We went into like how to compare all vaudeville operatives comparison and the logical operators and then. We also talked about if L it is an else statement then we continued our looking functions or we define function how do we actually execute the function.
(Refer Slide Time: 42:18)



By value / by reference

- Everything in Python is a reference. However, note also that immutable objects are not changeable --- so changes to immutable objects within a function only change what object the name points to (and do not affect the caller, unless it's a global variable)
- For immutable objects (e.g., integers, strings, tuples), Python acts like C's pass by value
- For mutable objects (e.g., lists), Python acts like C's pass by pointer; in-place changes to mutable objects can affect the caller

And then what are the other things associated with it one of which is the scope and the went into more depth rather than this one mentioned in passing last thing and then we went into like these values the various scope examples. And then we talked about how the works in Python and then we saw some examples how do we use values so functions it is the top okay. So thank you once again and we will see you in the next class thanks.