[Captioner standing by]

>> PROFESSOR: Good morning. Just to make sure you have an overview of what we're doing, today we are finishing up section 6. Then how to manipulate lists. Then we will start with dictionary manipulation. These topics are fundamental. Last time, last part of lecture six. We began discussing the scope. For example, look at the definition of this function. Look at the body of the function. There is a variable named total inches that appears he does not appear outside of the function. Okay. You have a function, a bunch of stuff. It happens to be a variable here. This appears here. It does not appear before that. So, the scope for the particular variable is this function you have defined variables outside of a function called a global function. The way to understand this is the variable is seen in the code. Locally, and this is the case, total inches, or if it is to find out side of the function it may appear outside. Also here, so it is not local, this is a global variable. Now, you have something like this. You notice here the name appears in the function. Visually it is here. However, that same variable appears outside. So this variable is not local to this function because it does not appear outside. Is that clear? Okay. Now the question is what happened? You want to change the variable and you want to tell the interpreter this variable is a global variable. Then what you do is - - the name of the variable with this Python. The SSL key is global. By the way this name is going to be here. But this variable is global. So when you change that variable here because you told the interpreter it is global, and will be seen outside. So in order for a global variable to feel the effect of a local change you have to indicate in that change that the variable is global. So the change will be seen outside. It happens to be when programs start becoming big, it is good practice. It is only mobile variables used for those things that are very constant. Throughout the whole process. Don't start mixing global with local if you are not sure what the effects are going to be. There is this notion or name which is just jargon to indicate that for every variable there is some kind of context in which the variable is meaningful. It is just jargon but good to be aware of. It is probably Megan you may ask yourself, how the heck? Am I going to keep track which ones are global and which ones are not. The interpreter are the ones that track all objects in the problem. For example when something is being executed, the interpreter is going to look at some name and space and we tell you how to maintain it. For the interpreter it is very easy to tell you how. When it is executing this it has to find somewhere. What is the object referenced in

the first two? What is the object why and the first two? These objects, that operation, it is going to update. Some variable called Z. Some name and space with the results of the expression. How this is maintained, it is maintained by the interpreter. As a dictionary. Remember what the dictionary is. Something that has visually two columns. One column is keys. In the other column contains values of those keys. Thinking of an example, names of students. The second column called B, grade of the student. This is good to have some mental view of this. In this case it is something the interpreter maintains. You don't maintain and the interpreter has a dictionary. Where the key of the dictionary, names of the objects and values are the objects themselves. The interpreter itself uses dictionaries to keep track of all of the names of objects in the code you develop now you have a very large program. This could become a nightmare, but it will be easy to handle if at some point you are handling sophisticated code and I imagine in your sophisticated code you will be doing that. A very simple thing, Python, you can invoke something called locals. Something called globals. Built in functions in the language. And you really want to see all objects, you just say print globals. You get a whole bunch of stuff telling you what all global objects are and the names for your code. Same thing, you can say Robles and it will print for you all global objects. How often are you going to use this? Not very often. But in real life development, when programs are pieces of code that are very important for all of the pieces of code. And in your professional life that is what will be doing. An application with your group that has 100,000 lines of code. You are not going to be developing 100,000 lines of code yourself. These applications have a template for development and you design three, four, five, six, 10 pieces. It could be that what you are doing is having a problem that has influence in many other pieces of code. At that point what is done? You use globals and locals to identify where the objects created exist. This is good to keep in mind. This is the kind of printout you get. I'm going to ask you to understand this? No. These are just mappings from names to objects. Okay. So why name spaces? Because these are the mechanisms interpreter uses to detect where is the scope of the variable. The interpreter uses all of this information to develop the scope of the variable. If something is going wrong it will tell you. The three types of scopes to be aware of, active in any point of the execution of the program. One is called a built in scope. Built in scope is all of the names of Python. Range, - - is built in. Mobile scope is all defined names of design functions and local scope are those local variables. It is important to be aware of this. Especially as you develop complicated pieces of code. That is the purpose of this. Just to make sure that you are getting the just of this global versus locality for variables, suppose you have a function competing the outage of two numbers, a and B. You divide by two and keep some variable called TMP. And then determine the value. It is a very simple function. The important thing here is this creates a variable called TMP. And the local space of the function. Suppose you assign to a five and two B 10. An A+ B is assigned to 10. And now you print the average and you print 10. Well, wait a minute. This one was the average. And here is 10 in TMP. Here starring in TMP, is there any conflict? I am using the same name? So this TMP is local here, not local to hear. These things are different. This is the same thing. That is where you print now, there is no conflict. Maybe the most interesting aspect of this lecture is what is called pass by assignment. How many of you know what is passing an argument to a function by reference? The word reference is the key. How many of you know that? Okay. In general. In any program language you have two types of mechanisms. Continuing with variables. If I have a variable and I would like to pass it to the function I can do it in two ways. By reference and by value. So if I have a variable this is the value of that variable. I can pass the value to the function but not the variable itself.

These things are different. That value is 10. I am passing the value, 10. If the name of this variable is Peter and I am passing Peter, that is different than passing Peter. This is called a pass by assignment. I know the description is not clean enough. The way to understand this is by looking at this particular - - for you. Suppose there is some piece of code, something called TBH, a variable. The interpretation is that this is the name of a variable. The way we represent this as this is like a tag to this particular location and memory. That particular location and memory has a certain story. Nothing new like that. Suppose there is a function somewhere above called definition, with a variable called H, that is the parameter, the argument of the function. This age and - - have different strengths, different names. So if I want to call this function - - and call this function with this variable, TBH I am doing the following. I am saying right now I want to use this variable called TBH to be passed to this function. TBH is a tag with a value seven. But the name, variable is not TBH, it's age when I call this the value of this TBH it is assigned to the variable H here. The effect of this is that after you call this function with some variable, the variable is assigned to age. So this memory location now has two tags. The first tag that was there before and the second tag is creating that function with the value name age. So the identification of these things by passing this variable here is really creating two tags to the same memory location. This is very important because now you say okay, for whatever reason you say I have an assignment statement and I'm going to implement the variable by one. While age is going to get evaluate. But TBH will not be affected. Raise your hand if you got this. If you don't know what is happening, print. TBH, you should get seven. This is called pass by assignment. And this is usually a very common route of mistakes and coding. Not understanding what pass by assignment means. This, and other programming languages, refers to pass by reference. But in Python they referred to this bypass by design. One thing that's important to be aware of, there are certain objects that cannot be changed. And objects that can be changed. So if an object cannot be changed and you happen to wish to change something associated with that object, what happens is a new object is created in the function's local scope in which you are making the change. But the original argument is not change. This is important. If it is new to the group the system creates a new object. This is not the same. However if an object is something that can be changed and you change it, then the object is seen outside of the scope of the function. This is a tricky relationship between objects that can be changed, objects that cannot be changed and a scope. Another important concept in terms of arguments and functions is keyword arguments. In fact, there are these conventions. Usually when you look at code you want to hide somebody for developing code. One of the things that is seen and analysis do this, naming conventions. You should use names that are meaningful. You are computing with dollars, well, dollars. You are computing with euros, euros. Your converting something from dollars to euros, conversion from dollars to euros. This is how good programmers are identified. Naming conventions used should convey the semantics for the computations that you are making. One thing is to use keyword arguments. For example, suppose you were looking at this function with a number of chapters. That we are missing from parentheses. You should be aware of this because of a project. Using math, title to be the title. Published to be published. Year to be the year. Version 2 Beta version. Chapters to be that, etc. So you are matching what you are printing with this arguments. What could be the problem? This looks natural. But that could be the problem. There is a possible, very common problem. Which one is it? It is assuming that disposition of the arguments matter. This is assuming the first argument is title. That the second argument is auto. The third argument is publisher. So this positional convention is problematic.

You have a function with too many arguments. Because, when you are going to do this print you may mess up and switch arguments. And you can confuse the title with the auto, etc. How to take care of that. Very easy. Use keyword arguments. There is nothing different here. The same. But you say print book description title equal to this, this title, you are specifying that the name of the argument is title. It is not the first position. Then you don't have to follow this order. You can type publisher equal to whatever the publisher is. You can change the order of the arguments by using keyword arguments. Notice the difference between this and this. This is printing directly the values of the arguments and of the order specified herein. This one is doing a little bit more. It is matching what you want to print with the name of the argument. And in that way you can print this in different orders. So this is also an important thing. The parameters for the values, you should make sure that for any function you are aware of what RV for values. Moral of the story in the business of objects that can be changed and not changed. Do not use objects that can be changed as parameters. You have a parameter, it better be something that cannot be changed. And specify whenever you write all of this. Okay, I think that the, those are the items I wanted to point out. Now we go to lecture seven, manipulation. The way to understand all of this stuff, the way to keep it in your head is to have a visual picture of what the operation does. So what is this business of analyzing? It happens to be that Python allows you to have negative indices. This is just what is called a syntax - -. Where you use a negative index you can traverse at least backwards. If you use -1, -1, always refer to the last element in the list. So you have this list and you insert -1, that is the last element in the list. You want -2, that is the second to last element in the list. So you get used to manipulation of indices. When you are using negative indices, the item in the position is really the item in the left of the list plus I if I is negative. That is the way that you write these things down. If I is negative you take the length of the list and add a negative number. Then you are pointing out to something before the end. As the first. Length of the list and now remove something, some offset. And then that is the element you are accessing. That is what this line is saying it is good to get used to this manipulations. Okay. Now you can assign to an element in the list and it will change the list in place. The first element is 11. First element to be 11. What do we change here? If from accessing the first element in index one, which is the one that I'm changing? This or this? In this assignment. You have this list enter accessing the first element in that list. And you are taking the value 11 and inserting the value 11. Which one are you changing? Which one? This one, right? Yeah.

>> STUDENT: [Away from mic]

>> PROFESSOR: Minus?

>> STUDENT: Length of X times -1.

>> PROFESSOR: Yes. Okay. Another important operation with list is slides. Waterslides? You have a whole list and you would like to go and take a chunk in the middle. That is what slides is. The second type of manipulation. You have some list and you are interested in the elements in that piece. Okay. How can you do that? Well there is this syntax in which you indicate what is the index of the first element you want to include? What is the index of the first element that you do not want to include in the list? This is usually a Points of confusion. This is the index of the first element not to include. Why the developer found this useful? That is not the issue. Syntax is first element to include, blah blah blah. This is the first element not to include. It happens to be that through a list

in different steps. What do I mean? You could want to walk through a list, element by element. Or you want to work a list by starting here, jumping here, jumping here. So these are steps. That you can use. This is extremely useful for text processing in particular. You can have something gigantic and you can say every element etc. And you can specify by using this. But before this step is, the value is one step at a time. If it is forced to zero, submitted, forced the length of the sequence. You have to be aware of these things. Look at how simple this is. You want to extract from a list from the first index of the fourth index. Use a greater list bracket, one through four. From index 1 to 3 because you are saying four, the first element not in the list and that would distract this piece of list. Suppose you want the second to the fourth. Well, the second to the four is actually this. These are just interpretations of the specification you are given. There is another very powerful thing called stride. The syntax is these four dogs. You take a list and put four dogs, three. This is saying walk through that list every three elements. You want to walk through that list every 10 elements you just, these four dogs, 10. And that is it. That creates another list which contains these elements, every third in this particular case. In this case, this was the list. Every third will be this one, three, six, nine. Very simple specification. Another important operation is concatenation and we spoke about that. I would like to single thought out. It is also very powerful, this concatenation over this. Remember that concatenation in Python is syntactically expressed. But you have to be careful in my dealing with lists or numbers? If you are dealing with numbers you are adding numbers. If you are dealing with lists concatenating. This becomes extremely powerful. When you have especially large data. You want to create large data. This is a good example of syntax. Using X access equal to one or three, it is important to understand this. It means there is an object, refer with the name X access, the list containing the elements 1 to 3. I would like to create another list that is going to use this as a piece of that list. You just write to create a new list from X to X. Meaning concatenation with some list. By doing that you get this. X X does not change. You printed, it is the same thing you started with. You are creating a new list that happens to have as the first components were piece of that list. The elements of XX. There is another operator very useful, start operator that allows you to repeat. Sometimes like to repeat stuff. You learn something called formal languages in computer science. The store operator allows you to define context free languages, which are the languages, the theory behind program languages. Suppose you have the same list with these objects, one, two, three. Indentation operating in another list and you would like that list to produce a new one which is X repeated three times. So you get one, two, three. One, two, three. XX is the change. Is that clear? Another thing that is very useful, especially when dealing with financial data. One example is being able to take a list and got the largest element without having to write any complicated piece of code that, the largest element. There is a function called Max. As long as you have something that can be iterated you specify like this and get the largest element. Whatever it is. Can I specify like that? Using this motorcycle. More interesting is this one. This one allows you to get the sum - - this is the value, blah blah blah. Well, that is going to be a gigantic collection of members in a day of training. And what is truly people like to get the average value during that thing. The average, the first thing to do when computing the average. He would like to sum those numbers and divide by collection of numbers. To get the sum, Python gives you a function called the sum. In which data may be coming in, blah blah blah. And you will add all of those. Starting with, specified where you want to start. Or you can specify it later on. Mix, or the trade. You may have a list of integers which are positive or negative. In financial markets this is very useful. You may be representing it went up by three points. It went down by two points. Etc. So now it goes up and down. Up and down. And you would like to find out where is the maximum. You take the maximum, what is the maximum? What is the maximum? Is it Three? What is the meaning of this? -2. Yeah? What is the sum of these numbers? -2+2 Is zero. +00+ -1+3. Sum is two. When you have an idea that things are growing, but when numbers are positive and negative, some of them cancel each other. So it is important to understand the meaning of this operation. They are very simple. But you have to be aware of what is happening. Okay. There is this business of unpacking and Python and some people become in dealing with this gigantic list. And sometimes they want to unpack it into a list of variables. Why? Because sometimes you have a list in these values, you would like to treat this more as variables. What you can do is take a list and assign it to a whole bunch of variables. Then you can deal with those as variables. Regardless of the values here. So they give you power to generalize whatever processing you are doing. You can operate the rows used to unpack these into different pieces. In this is a very nice way to understand this. And this is maybe something that is not common in other programming languages. It is very specific to this language. You may have a list being referred by this variable. After you can take that variable and treat it differently. You can take that whole list and treated different. This creates views of a list that are very useful for manipulations. In this particular case, what we're doing when you have this statement, on the left-hand side you have this funny thing called first start arrest. What are the semantics? The semantics is this is my list. You are going to create a viewing. Where you want to keep track of the first elements of the list. This says variable first, we have value five in this list. This says the rest I'm going to treat that as a variable called rest. The rest is the rest. Now you are creating two variables with some semantics. The first elements and the rest. Notice this is independent of the context of this. It gives you a lot of manipulation power for list. How do you know that? Another example. Here, you have that list. First in this case is five. The rest is whatever it is. Print first. Now you can treat this as a variable. Rest. Print the rest. The rest is the rest. This allows you to take maybe a very gigantic list that you don't even know how gigantic it is. But you might be interested in extracting a particular element and treating the rest as another variable to which you would like to apply some other process. This one is an extension of the previous one but we would like to keep track of the first. And the last. And while this is going to do, this rest is going to be constrained to go from the first to the last. So this variable first, let's find. This variable last, and the rest is only between this and this. It is done. How about something like this? You can actually say I would like to specify the first meeting the first is the beginning of the list by default. Then the last. Giving you from here, to the elements immediately the last one. What about something like this? What will be the effect of this? We take this, defer to that object. And now you have four, five, six, seven. Nothing that there is no. Can anybody explain to me what this means? Is there anything different here? You have a list referred by XX. Here we refer to that list. One, two, three. This is going to get the elements of that list, one, two, three. It is going to inject them here. Inject them here in a new list. You can simulate this concatenation. Nobody is saying no, there are no fast tricks to manipulate lists. This is now more interesting. When you have these ranges in which you say from here to here I want to do something from here to hear them. It happens to be that in many processes, there are specifications. You have something from 1 to 4. Step one. How well it be the interpretation of this if you apply to a list? If this is a range you go from the first up to the four elements and steps of one. What is this interpretation? From the first element of the 11th element. Jumping by twos. From the 11th element to the 33rd jumping by threes. What is it we are trying to exemplify? These ranges are specifying some chunks and you are going to apply to some list. Each specifying some particular chunks. Now you would like to use them. Use this as parameters to some other complicated process. You can now make a list of these ranges. This is a list of one, two, three ranges. And the beauty is now you can do a for loop. With this list of parameters. The four loop is going to take each element which is now a list in this list of parameters. And is going to do something with it. Let me just pause for a second. If this is a range of some list. This is a range. Of some list. A range of some list, you can put those together in a list. This is a list of ranges. It is that clear? When you have the specification you have a list of ranges. When you have a list you can apply the for loop. Right? For each item in that list, do something. But the items in this list are not atomic objects. Each one is a range. Raise your hand if you got this. Very few. What happened back there? Was that clear? No? This is an example of how powerful a language like Python is. Why? A list you can put inside a whole bunch of stuff. When you have a list you have controlling of the process for each of those complicated pieces, in this case a list. For each of these specifications. This is what it means for each specification of the list. You are going to do something. In this case to exemplify to you what it is doing. This is the first one. This is the beginning. Then you are going to do something. Now inside of that you can have another loop. Can anybody tell me what this means? To understand this you have to - - distance. This says range program and this star means I want to look at something being referred by this range perimeter. You go up here and that is this. This is referring to that. What is this range of parentheses, though? Do you remember what range is? Range is something that tells you, start here. Go up to here. Start here. Go up to here. These steps, range is the piece that specifies somehow in that particular example continuous piece. But you can have a range like this, too. Jumping the steps, you got the range of this, this and this. You are distracting that answer that you are doing something else. So what this is saying is I'm going to use this object range perimeter. I'm going to apply the range functions have that. The first value of this is this. That is a range. And to that, whatever it is, I'm going to apply some function. In this case it is simple. Print the value. This is now a local loop. Nothing that is print as part of the local loop. This print is that the external print. The Prince belongs to this external loop. These are two loops. Controlled by these range parameters. The internal one is doing something local to each one of the specifications here. To this one, that is that. This value is this one. To that one, do that. This is this. To that. So this is extremely powerful. And you should follow Y with those statements. This is the answer. It is becoming more powerful. Your main take now is something like this. A list, X, Y, C. you may decide to inject that here, here and here. There was a list before that. Four, five, six. And there is a way to inject something anywhere that you wish. That is the point of this. In this example you take X, Y, Z and inject that. And this new list called W. How was this expressed? Take that object referred by X, take the object referred by X injected here. This is take the object here, not here. It is positionally. Take the object by C and inject here. That achieves this for you. What is the view of this? You have a collection of lists. You can use those lists to inject them into some other bigger list by just using the star operator by then. And specify the position in which you want to do that. One, two, three. Inject. Leave this. As it is. And inject here. This is going to achieve the injection for two copies of X. And to something that was there before. Inject one copy here, inject another copy. This gives you that. Raise your hand if you got that. Extremely powerful. Extremely powerful. There are other uses for these methods. One is counting. In many processes you like to find out how much a particular object appears. So you have a list. Remember when you have something like this, this object in Python. You have a count, the way to understand this, it is a method that is going to be applied to a list. This method count applied to a list is going to return the number of times that X appears. This is the list. You're going to apply the method count to the list. The perimeter is a, we count the number of times A appears. Clear? In other programming languages you have the right piece of code to achieve this. You can do it and every element of the list. Looking for elements A. When you compare with a. Keep going until you found another one encountered by one, yes, you can simulate that. But now you can evaluate one - -. Index. In many cases you would like to return the index of an element you are looking for. We are dealing with lists Karen. This means what comes afterwards is certain metals called index. The syntax, the semantics of this method is you specify here is a perimeter. What you want to do because in this case you want to extract an index. So you have this list called X. The first component is not specified. After that you have the start. In this gives you the first time where the value appears. So here. You can look at C appears in zero, one, two, three. It would take a little bit for you to get the syntax down. You practice with those exercises. This is nothing difficult. There is also another method. Dealing with lists, this is going to allow you to insert a particular item out of position. You say I want position I to insert this item, X. There is a little bit of displacement. This first argument in the element before you want to insert. I will ask you to tell me if you understand why the heck when you have this list and you go to say apply the method insert. Two, which is an index. Look at this list, the result and tell me what is the effect that you see? Only one? Yeah? Back there? Yes?

>> STUDENT: [Away from mic]

>> PROFESSOR: So where the mouse is pointing. It's saying that in position to you insert this. And it's good to be aware of what's happening. What happened in position two? Something different? It was A, right? So this method is doing a lot more than what you expect. In general. It is not replacing the elements in position two. It is pushing everything. To the right. Pushing everything to the right. To allow this method to use the position where a was to replace that value for X. It is not replacing the previous value. The previous value has moved. In that new position the value is - -. Remove. This is a list, this is a method. Remove. Some item is going to remove an item in a list whose value is equal to X. We may speak of the first, second, 10th. Because we assume this method is going to be applied to a list and the list is sequenced. You may speak of the first, second, third. It is a sequence of elements. That is why we can use this terminology. Remove the first. This one is gone. Reverse, another important operation in computer size. You have a list, method called reverse. You reverse elements of the list in place. Corresponding that list, you apply the method reverse. These elements are going to be reversed. I'm sure you have these, palindrome. What is a palindrome? A word, reading from left to right. From right to left, you get the same thing. That is a palindrome. So you ask to build palindromes. How do you use this? You want to check if it's a palindrome, reverse it. Compare with a list. If it is the same that is a palindrome. I will leave this for you to check. It is a little bit cumbersome. Except I would like to mention to you that this particular, why don't I tell you? The way this is written, parentheses, a list. It is different. To this. Here. Reverse list. Versus list reverse. Can you tell me when you see that with your eyes. What is the difference syntactically? What does it mean when you see list.reverse. How do you take that? You take a list and apply the method to the list. Yes? The other is saying reverse parentheses list. What does that tell you? That reverse is.

>> STUDENT: List.reverse is a method that applies to a list object.

>> PROFESSOR: What about the first one?

>> STUDENT: I believe reverse is a function that takes list as a perimeter.

>> PROFESSOR: Very good. The main difference is the second one is a method that applies on the list. The first one is a function. That is applied to lists. There's two things that are different. What is the difference? The easiest way to think about this is that this is simply a function. That is going to apply a process to a list. And that process, the best way to think about this is in this way. This process applies to this list. It is going to go from the end and will look at F and is going to consider F a list by itself. Then goes to the next one, B. And is going to consider that another list from one to the rest. The other one, excuse me. The next step is going to take to which is going to take elements a and the third position going backwards. And is going to consider that the core objects. So it is an iterator. It is iterating over the elements but backwards. And the usual, there are many processes in computer science, that is exactly what they do. I think that this might be good enough explanation for now. Okay. The last one is reference and copy. Remember we use that term. When you use that value you would reference the list. When you change something like lists, by one reference, all references would change. Memorizing them, this is a practice. In the second project you will have a chance to practice them. This is a variable, a list. Why? Because I say, a is a tag for the list and the list contains three. Suppose now you made the assignment, a, assigned to variable B. What is going to happen? You create another tag, pointing to the same list. Is this clear? The meaning of the list. You assign that list. The variable, to another variable, B, creating another tag. That is a different tag. The list has a way to refer to it. One by a, one by B. You can now look out the elements in position one in the list aim. And you want to assign that to 99. That is going to change the same thing to 99. That is what this means. The question is how that affects B. B is still referring to that list. Therefore in position one, B, we have value 99. That is the moral of the story. There are other examples that will help you so you understand. When you assign a variable to another variable pointing to a list there are certain changes propagating other changes that are not propagating. Okay. Last thing I would like to mention, something called copying. Remember, before we have a list, here the statement we were using was a is assigned to being. That was the statement we had before. This is different. Now we say we want to make a copy. A copy of letter a. And specify certain indices. After that extraction is done we assign it to B. Do you see what happened here? Just open your eyes, look at the numbers you see here. Copy A2. Look what you have. What effect do you notice? Look at the next stop. Remember, we are building B, based on two. These two elements, that's a. Now we believe B. We are going to have here the list. The first piece is a copy of a. The second elements would be a, one third element would be two. But I haven't told you the results of this. Do you see what happened? This is a copy of aim. A copy of a. This first piece. What is the final results? Can anyone tell me? What should be next? Once more and then we finish. You have a list called a. I'm going to create another list called being. That list is going to contain internally a copy of A with other elements, two. This copy is going to put here in the first position of B. So this first position is a copying. Of A. The other will be in its place. I think you should check the by yourself.