[Captioner standing by]
>> PROFESSOR: Morning. It is 8:30 AM. The midterm will be two weeks from today. You need to bring your ID. It will be in class. We will cover as exactly described as in the syllabus. Have basically three more lectures to go. Last time we spoke about two types of loops, why loops and for loops. Today we will discuss certain uses of certain Python statements. Related to functions. One thing I would like to mention is I notice somebody walked into my office and I noticed the way that this student was navigating these lectures was a little bit strange to me. The way that he was navigating was by staring down. That is not the most effective way to do this. The most effective way to do this, the way we have prepared for you. There is an index, the beginning of this lecture you see these topics covered. That is an index. To the material. Point number one. Look at the index. The second thing to do. In that index, there are certain things that happen to be highlighted. Maybe and, may be bold, may be black. Meaning from our viewpoint the most important items in the index are those ones. This, this, that. Giving you a high-level view of the concentrate. That is point number two. Point number three, I want to go here. Just click. And it takes you to that portion of the material. Okay. Questions? Okay. Next item. Some people have been asking me how do I prepare for the midterm? There are three steps. Step number one, go to the lecture. Step number two, look at the corresponding CUI book participation activities. You can choose what to practice. Step number three, look at previous quizzes. We have posted previous quizzes in Canvas. You should be able to go through. Okay. If you skip one of these steps and you are not doing well so far it will be difficult though you will do well in the midterm. Now in terms of what are the things that we expect you to have learned? For every lecture, you go and look at the index. The index tells you the most important points. And those are the most important points. Lecture one most important points. Boom, boom. ZY book exercise. Lecture two most important point. Blah blah blah. One of the things that has been proven effective through the years, it is good to start with some variables. Unless you believe you are the Python genius. And I have not found a Python genius yet. But maybe in this class we have some Python geniuses. It is good to start somewhere and discuss with somebody else. That is one of the purposes of the groups. If the group does well, we give bonus points for every member of the group. Yes, there will be different sets in each group. That is why we have this questionnaire. We balance the group in such a way that groups are balanced. So that is part of the experience in this class. Being able to start in your group. Get to know your group members. Go to a bar together, maybe. Go to a discotheque. That's fine. At the same time solving what you need to solve for the class as a group no programs did this by themselves. That is gone from the past. You will be assigned to a group and right. A human program. Human in the bar. Closing stop, what is this? Sometimes a function starts at this. But you are looking at these lectures you can do what I am doing here. There is something red here. Oh, okay. The most important piece is a function can be

subtle as placeholders. Placeholders in your code. How is this done in Python? I want to compute something about a number of steps, how the number of steps that I walk translate into feet. Whatever your stat may be, the number of steps is three. Then you have a parameter, number of steps. Multiply by that to give you how many feet you've walked. This is something that multiplies the perimeter by three. I'm interested in how these number of steps translate. Well, I don't really know how to do that. I would like to apply a function that converts steps to calories but I don't know how to write that in Python. You write the definition, steps to calories with number of steps. Right now you write that PAS. All you are telling the interpreter is I have this function. I want to call it steps to calories. But I don't have it yet. Yeah. What is the advantage of this? Why is this important? It is how you develop code. Especially when code becomes big, you develop a template of your code. Or the functions. And understanding how that template goes means you understand the program and process that you would like to implement. The way to do that, you don't have to type everything. Pass, pass, another one. How this is related. Treating that. So this is useful. Here is a great message. The important thing is you complete the message and complete the message for Whom? This function I don't know how to write. I put PSS, working in the group. Making it aware I don't have the functions yet. So you know the previous statement is not developed yet. That is how groups, developing, works. It is good practice, to return -1. Remember, you don't have this code. You are creating the wrapper, I cannot print a statement saying finish the steps to wherever this is. Then return -1. So this program is not going to cause even though you do not have the code for it. Is that clear? Raise your hand if you got this. Most of the time most of the time in big projects most of the time it is spent trying to identify what it is trying to do. The programs are the ones that do not follow these principles. This part was developed by JJ. He never puts the stone. The project says no, don't put JJ here. It is a program in practice. It is extremely useful. The interpreter is not going to be upset. Another statement. So many. Oh I see the stops can be used, - - something and for some reason it cannot be finished. If the required impact, you don't know the user is going to type the input. You really don't know that. Even your code is perfect. And you say under input. You are going to type the input into it. How do you protect that? How do you protect that? Here is a way to do it. You would like the program, this silly user did not enter the input. You entered the statement wrong. So you add or raise an exception. Remember we spoke about that. Implementing that you put that down. For example have a function who would like to get points from the users like this one. He is supposed to, you raise the exception to implement any interpreter is not going to complain. This is going to continue. Anyway, tell me what the heck is this? The expression. This is on a side point. But can anyone tell me what is that? Yeah.

>> STUDENT: [Away from mic]

>> PROFESSOR: The difference between two points. On the plane. The second is to

understand why. There is a method. What is the method? Being specified in that line of code. Just read. You're not thinking. Just open your eyes. Where is the method? Where is the. Yes? Thoughts are important. When you see the dog, what is before it? This MAT H. Before that this is something called MAT H. That something is bribery. What is after the? What do you see? SQ RT. What is that supposed to be? It is a method. A function that exceeds in that rivalry that does something. This particular one, how the heck do I know? You can check. In this case, SQ IT. That is a square root. The translation doesn't stop there. Math. SQ IT. There is a method in that rivalry function. Already implemented for you called SQ Martin. Or square root. What happens after that? There are parameters. What do you see inside? Tell me anything you see inside. The parameters of that function. Another question may be to help you. What is inside the parentheses? Whatever it is. That whole thing, whatever it is. What is that in that case? Is it a potato? This is a tomato. This is a man. This is a woman. This is a song. What the heck is that? In parentheses. What is that? This is a tomato? No. How the heck do you know it's a frog? Give me a higher-level answer. You are correct but give me a higher-level answer than that. It's an argument but what kind of argument? It's a number. Some kind of number. Whatever that is it is some kind of number. And that number we are going to apply there. Square root. This is what it is saying. Yes? What the heck is that number? Let's look at the first piece. This is how you pass these things. In the beginning you may be notified of this. It may be cumbersome but you will get it. You will have some parentheses over there. Whatever it is. Now*,*. Now there is another parentheses. Another*,*. Parentheses *,*. There is something between those two parentheses. Which is valid. I try to give you a hand that sometimes you see something very complicated and that is what you do. This is the whole argument. What are the pieces inside? For each one, what does it mean? You practice this and later on say oh, that is this. Yeah, yeah. What is in first parentheses? There is something that says P. Isn't it? That P has a number here. Whatever it is. But that is not important. Then this bracket, by the way, I am going to forget about this and just focus on one of them. There is a bracket. And inside of the, is zero. After that it says minus, then after that there is one, I guess. There are zeros. And this is inside the parentheses. And you are focusing on one parentheses. This is not essential at this point. What is that? Can you help? Between two brackets, zero. What the heck is that?

>> STUDENT: [Away from mic]

>> PROFESSOR: You say what? Is the X value for one of the points. But why is it written like that? These two brackets, zero.

>> STUDENT: The X value is still - -[away from mic]

>> PROFESSOR: Because the point is represented as what? What is a point? What is a point on the plane? This is why this goes with this. You will see and 35 seconds why. What is a point? On the plane. Mathematically speaking, it's two numbers put together. This is seven, this is nine. This is the interpretation. Seven here, nine here. And this is the .7, nine. That is the points on the plane. This is mathematics. How is

this represented in Python? As a what? If it is a point that has two components you have to use something to push the components together. Yes? If it is a point it will be a topper with two. That topper has to have a name. So that's P2. The name of a topper. That zero is the first components of that topper. You have something like P2. Not has two components, like seven and nine. That is the point. And I would like to access this . These things, this is the zero component. This is the first component. That one is P2 bracket zero and this one is P2 bracket one. The zero components of that code being two. This is the first component of that code, P2. Raise your hand if you got this? Because now I'm going to tell you in 10 seconds why this is very important. This is very important because look. This is the simplest kind of topic in mathematics. But in general, when you are dealing with higher-level data with higher dimensional data. You don't have to components. You only have 150 components. 300 components. 1000 components. And finally, the machine learning, we have collections that have millions of components. However, from the computer science point of view we don't get scared by this. So if this has a thousand components then we may decide to call this point P 1000. This has no meaning in the language. That has meaning for you as a programmer. That you are dealing with vectors with a thousand dimensions. You want to refer to this particular entry here. And this example 79 and this one is 25. You want to refer to this entity here. It will be P 1000. Bracket, this is the zero, first, second entity. That is done. Regardless of dimensions. You want to go to the 39 entity, this. Then 3000. 39. The importance of this is that it is generalizable to any dimension. The important thing is you are dealing with high dimensional data. Each object now mathematically is a vector of numbers. It can then be accessed by the name, P something. And each entity can be accessed by the corresponding index. Independent of the dimension. Is that clear? Are you happy? So so. Semi-happy. Okay. Coming back, we are taking as somebody told us before, this is the zero entry of point P2. This is the zero entry of point P1. We are taking two dimensional points. Each point has two components. This is the first component of that.2. This first component - - you really understand this after that. This is doing exactly what he said. And just to publish, I don't know what it is. Precalculus or whatever it is. In general you have two points. This is math, and Python. One, one, next 2 x 2. In our language this is .1 and this is.2. What the expression is saying is state the first component of this. Called P1, P2. I guess. Yeah. So in that expression you are taking these two. X1 -62, X squared. X2 minus wide to his wife squared. After that you are summoning them.), Some second parentheses. After that you are taking the square root. And this mathematical square root is precisely the difference between two points. Can you maybe see? Oh my gosh. Can you repeat here? Yes, yes. Why? Because this is so so. The realization of this to any dimension. So if you have, instead of two dimensions, you have points. X1, no, I don't want to confuse you. Let me just use a different letter because I don't want to confuse you. If you have, let's say C1, C2, up to see N. This is a vector. With a vector N. You have another vector, W1, W-2. W N made in Python language this is a point. This is .1 in dimensions. This is .2 in dimensions. Now what is the distance?

Between P1 and P2? This is not a two-dimensional problem. This is a - - dimensional problem. Anybody? By the way, there are people that have learned pre-Calc and have been focusing on two dimensions when they are the champions of the world. That's very little. The important thing is that many of these things have generalizations to any dimension and that is the power of mathematics and computer science. That definition here, for any dimension, can anybody try to generalize, let's say, let's go to three dimension. Suppose you have two points in three dimensions. C1, C2, C3. And the other one, W1, W-2, W3. There's points in three dimensions, this is .2 and three dimensions. Look at this expression here. I tried to generalize it to three dimensions. The way to think about this so you remember it your entire life is, when this is saying is take the first component here. Take the difference and square it. I can do the same thing here. Right? The first components are C1 W1. Take the difference. And square it. Yes? Now the second component, C2 minus W-2. Take the difference and square it. The next one, C3 minus W3 and square it. Is that clear? Now what do we do with all of these things? We add them together. Yes? What do we need to do to that whole thing? Take the square root. You know how to compute the distance between two points in three dimensions. What
about if it is 10 dimensions? Can you tell me? Yes, 10 dimensions would be basically the same. Except this will be - -. Over here I would add plus.
>> SPEAKER: Minister before and square, blah blah blah. And it is the same. Okay. The basic model of the story is 19th century thinking. Was that the world was a two or three dimensional place. It's okay. When we understand, two and three dimension are very little. You are dealing with data. Need N dimensions. This is something that somehow, is kind of hard for some people to swallow the first time. Let me give you an example. Suppose that each of us are going to describe by 20 parameters. One parameter is height. Another parameter is weight. Another parameter is classification of the color of the eyes. Another parameter is location, etc. So we are using time parameters. Each of us is described by a 10 dimensional vector. These are very popular. Do you like oranges or apples? What kind of music? The number of parameters was something like 215. When you fill this out information in this society become a vector of dimension 215. What is the data inside this? It is going to take his vector and it is going to take his vector and is going to compute the distance between those two vectors. And then the matching game becomes to find pairs of vectors about our a small distance. There are some caveats behind the description. But the point is the data is not two or three dimensional. This example teaches you how to access this little thing but in your mind what you should be thinking is wait a minute, this is only for two points. That is precalculus, high school stuff. Okay, so now you Python genius say that I'm going to generalize this to two dimensions. And you can write code that has not very easily. It is the moral of the story clear? The world is not two or three dimensions. The world of data. Okay. I hope this becomes clear. Okay, eBay. How does eBay charge you? Different trees. Different thresholds. They manipulate this depending on the number of users. It is very simple. If something is less than $50 you pay a fee. These thresholds are somewhere

between 50 and a thousand. Now the fee is different. Something more than 1000 is a different fee. So you will like to check how much you pay, and eBay. This is very popular on universities. Making good money by selling different stuff. These records, whatever. You know better than I do. Bottom line is companies like this have certain structures. And that structure says in this case, the specification we have in this example is given by for Constance. Something up to $50 we are going to pay a fee. Paid to them. This is between 15.01 up to 1000. Now the fee is.5. Something higher than not, the fee is.02. But for every dollar you are going to pay, this to sense in this case. Okay. And you have the right to use the service. They charge you a fee. This is the information. The important part is not the numbers. The important part is to understand that there is a collection of categories. That describes the fee structure you are paying. So how will you write something that will compute in the same way. Going to 50 and you can do your computation. The fee you have to pay - - if this is not the case then you have these instructions. And in that case it becomes different but I hope you understand you have to pay a fee. For the first 50 items you apply this fee. For the other items you apply the other fee. If this is not the case then you have the other possibility. It is not that computing this mathematic is anything incredible and fantastic. The point of this exercise is for any structure you can develop basically the same kind of code. It means you have a collection of statements. If this is the case, do this computation. If not, don't do this computation. If that is not the case, if that is not the case, etc. That is the point of this exercise. To learn more coding patterns rather than specific pieces of code. You can check the details. Okay. Another point in the material is functions are objects. A function is an object. What the heck that makes? Let's make it simple. It is an integer an object? Yes, sir no? Is a plot and object? Is a - - value and object? These are the most atomic objects. Others are more complicated. Not atomic. They are collections. Give me an example. Of an object of - - a string. A string is an object. These are also objects. lists. Something different to list is also an object. Dictionaries. Something different all of those objects. What? Give me something different to all of this. Sets. You want to be a programmer. This is the most elementary thing you should have. They are atomic objects in any language. You can put them into collections to understand what are those collections. What are the collections we have seen? Who can tell me that? Atomic objects, integers. Jobs, bullets. Next is strings. Next? Come on guys, wake up. Wake up. Lists. This one? Sets. Don't forget sets. We are missing something very important. Not sets, not doubles, not sequences. Very powerful for expiration. What is that? Dictionaries. It is important to have that very clearly in your head. Dictionaries are not lists or sequences. Dictionaries are mappings. Books to order. Students, blah blah blah. So when you have a function the function has its parameters. Objects passed to that function. And a function produces objects as results. But now this is saying something higher-level. It is a higher level of abstraction. Functions, produce objects. Something higher-level. If the functions themselves are objects. It was not in the list, first it was atomic, no collections, no dictionaries, sets. Now

something higher-level. Functions. Functions are also objects. This is hard to swallow. I remember the first time I saw this. What defines an object? Can you tell us what defines an object? You're not a politician. That's what you are. Yeah. To be an object you have to have a value. What else? Right. What else? An identity. What are the types? What is the value? It depends. What is the identity. In memory, it is not usual on this time for you but this is what defines an object. Now type. Think of integers as a good example of floats. I understand the value or this. And identity. A location. That objects. Think of atomic objects as something similar to understand this concept. But now for the function, what the heck is a value of a function? It's an object. This is very. Maybe this is the most advanced concept. Until now we are using it in this class. It is a concept. You don't have to dig into it. But the function to exist. Must have some code that executes the function. If you have a function name, you have that piece of code. Yes that is a quote of Python. The function you want to execute. That's not. The value and the problem in a language sense of that function. This is why makeup, when you write these pieces of code that say in Python in any language it's like makeup. It is a makeup. It is on the surface. For that function to execute the computer has to do something. At a very low level. Inside the computer. That is going to simulate these instructions. That lower level in the computer, machine or assembly language. What is the point we are trying to make? The point is when you write this functions, in Python. The function depends on the low-level code. That implements that function in the machine. And that low-level code is called bytes code. That is the value of a function. Do you need to compute that? No. But it is good to be aware of that. There is a translation process. When you write the Python code to something, for example, for the very simple piece of code. This is the parameter X +1, this is a very simple piece of code. You go and look at the machine code that simulates this. It looks something like this. This, something like that. You need to understand this? No. When you to computer architecture you may like this but the point is the value of the function is determining when the interpreter reads the function you are defining and translates that into machine code. And that machine code is really the one doing the work. That is when the function exists. That is the bite code of the function. It is important to be aware of that. Execution by the interpreter, understanding immediately, they are executed, etc. So why all of this? Why all of this? Because the fact that functions are objects. That means you can pass a function as a parameter. Two other functions. Functions, other objects. The function now can be functions as parameters. And that is what makes the language is extremely powerful. Functions can be passed as parameters to other functions. For example you have a function called print phase, that Prince some kind of face looking symbol. Something like this. You type this in Python and say print phase. You will see two dots. This and less. That is called a phase. A piece of code that is simple like that. Now, don't take this lightly. Here this statement is taking this object that is a function and is assigning that object to a variable that is named F UC, this variable is dysfunction also, that assignment. You involve that function and get this result. Let me ask a

question before we go there because I want to make sure that you define this function. This is not function. Now you take that function, assign it to another function and invoke that new function. What is the output? What is the output? What is the output, though? Can anybody tell me the output? He said a face. No, wrong. Two faces. Explain why. To face.
>> STUDENT: [Away from mic] So you have two of the same functions.
>> PROFESSOR: Excellent. What's your name? Very good. Very good. This is two faces. This will print that. Now you assign it out to another and invoke the other one that happens to be the same thing so you get to faces. Suppose you have, functions being arguments to other functions. Suppose you have a function, you printed this statement and it would look like a human head, maybe. Or you have some other kind of statements that create a monkey head. Something that looks like that. You print that and it gives you a monkey head. Now you have two things. Human head and monkey head. And now you
have something that is print, another thing with parameter face. So this one is going to give you something like the body. This is just functions. And you put them together. Now you have two use it. Input, enter the one monkey, enter two, you want to do human. If the choice is one you involve the one that is going to print the monkey head. If it's too, you have the other one. You get something like that. What is the point? This, what is that? It's a function. Being past. As a parameter. To a Function. Okay.
Also with functions, people love to do copy and paste. What is the problem with copy and paste? I'm not saying don't do copy and paste. I find it very useful. When you are developing code and have developed a piece of code you can copy and paste the code somewhere else. What is it you have to be careful with? You have to be careful when you do copy and paste that the return values of the function are not messed up. For example here you have something, Fahrenheit. By the way, always, always, always, always, always. When you define functions, have a value. It will make you write a lot nicer, don't forget. For example you may have something here Fahrenheit., Here you have the code something else and then here you have Fahrenheit. There is nothing wrong with this code, nothing wrong. The problem is you copy and paste in this other statement that you have copy and pasted is there so this is going to return something strange. So be careful with what you do. Copy and paste. Another common errors is to determine the wrong variable. Be careful with that. Another common error is not to determine the function, always the value from the function. It's a lot more pressing. To finish this, today I would like to make sure you understand the notion of scope. Very important in program languages. The notion of scope of a variable. The scope or function. Let's focus on scope over variable. The important thing to understand is that depending on where the variable is in your code, there is a piece of your code where that variable makes sense. And if it is outside a piece of your code, I'm going to give you an example. That is a different variable. The important visual representation in your head for this is to think of areas of the code. Think of code. Here is a variable code, CS 210. Somehow you do something, being assigned, blah blah blah. That happens to be different outside

of that. There is another indentation and here there is CS 210. Something is being done to it. This. Is different to that. The scope of this variable is the visible part of the code in which the variable is used. Going outside of that, yes the variable is there but you say CS February 10, 2025 and here is CS 210 70. This is completely different. One example, and we go to the quiz. This piece of code, here is a definition of some function. You look inside the function and say oh, wait, wait. - - Inside the definition of that function. This one is defined beside this function. So this variable is local to this function. This one is outside. So this one, even if it is the same name is different to that one. It is very confusing. You can keep not very simple. Have something called gravel. You want a variable to be visible. Use the statement, global. That means it is the same. You don't say global. It is going to have a limited scope. The scope is where that variable is defined first use. Okay, quiz time. The same thing, after the quiz, go to similar. You want to gain points and we will play again for seven minutes or something.