[Captioner standing by]
>> PROFESSOR: The intention is he types this. We want to convert from Fahrenheit to - -. He does that then the call will say okay, enter now. Some number. Remember when this input is
put into the consult this input is a string that has to be transformed to a number. When you
apply the function that gets converted to a number. In this case, F is what gets the value that has been typed here. After that there is some computation. If you don't remember this is
the formula to go from Fahrenheit to Celsius. - - when you are in Europe you have to be able
to say the temperature today is this. What would not be in the U.S.? The other way around, he
did not type this. He types all other stuff. The assumption is he did not type this. He may have written something else. New York Yankees, for example. Instead of that. Did he type something else? We will assume he wants to transform Celsius to Fahrenheit. The other way
around. This is the formula. Then we will talk about the value here. This is a very simple piece of code. What is important to understand is input from the user, shaking the input from
the user. If it's this from that, if it is something else do that. Then it is the value.
Okay. Yeah, but what happens is this user types in non-numerical value. What would happen?
For example, he types ABC for New York Yankees. We expect this or not? So he types this. What is the problem? You are asking him to type this or that. The first time he types That.
What is the problem? Yeah? ABC is what? It is a string so, okay. We would like to somehow be able to run the problem. That is the point. To run it to completion, it does not mean, well we do expect it to compute but this is a very simple combination. So what can you do? Here is, as he explains. This string is not supposed to happen. We use this so-called try and accept. Let me. The new thing here is we are running this section, TR wind. A Python instruction. With a column. Somewhere down there we are going to have another Python instruction called EAX CPT. The purpose of this diagram is not really everything else. The purpose of this diagram is to understand that you can enclose some piece of code. By putting
the - - saying that anything in between is, something happens in between then no matter what we
are going to, there is another error. The way to understand it is we have painted it and read for you. When you get to the statement that is labeled EXC EPT. So, to get confused here, you

have a statement called tri-column. Here is a block of code. Must be indented with respect to
that. Somewhere down here you have a code accept and hear we have the exception. What is
exemplified here is this is code. TRY Colin, that code. Indented. What that is saying is whatever happens here, whatever happens here I want you to handle whatever exceptions. In this
diagram this is encoded by these. You go here, go through, go here. If it happens to be that there is a problem then we go to the exception. You go this way, if there happens to be an exception. If there is no problem here, no problem here than continue. If I put in the wrapper around the code that is going to run exceptions. Compute it. This is exactly the same
code we had before. But now you have the label in this code block with TRY column. And here
you are matching God within accept. And this is saying an exception here. I want you, telling
the interpreter I want you to encode the difficulty you found. ER R. This means whatever it is, use a value called ER R to specify the arrow. Type whatever it is. This is not required but useful. The semantics of this, you have difficulty understanding the flow of semantics, look at the dialogue. Now I would like you to - - into these two rectangles here. This is transforming input. This is also transforming input. And in this case the computation was one
direction from Celsius to Fahrenheit and in this case it was from Fahrenheit to centimeters. Whatever the value is being stored here. Now that you see these two rectangles, tell me what
is the difference? With the corresponding two rectangles. Here. Everything else is - - up to this point. Tell me the difference between these rectangles and these. What do you see? Same
people? This is not even - - what is here that was in here before? I will show you this statement before. This, this, this is exactly the same. This was not there before. The only thing with this is before this happens we need something. Tell me what is it that this is doing? Yeah. Loud, loud. It is taking, the other one is taking this thing. This is more than that. The other one was asking for two different ones. If it is doing something more than that. This is very specific. But it matters. Yeah. You have a comment? What is the What is the
statement in the green rectangle doing? You should know that by now. That green thing was not
there before. Now it is. What is that statement?
>> STUDENT: And exception.
>> PROFESSOR: Say aloud. That is exactly the answer.
>> STUDENT: Exception.
>> PROFESSOR: Very well said, very good. What is your name? Very good, thank you. You

are
taking the input, whatever was there before. And you are assigning that to a variable, IMP, does not sound like a value to you? What is this doing when you put this statement here? This
is allowing you to restore and in the value IMP whatever the user puts there. Before that, IMP
was not there. You as a programmer, you are being careful. Saying whatever is put there in the input I will store that in variable or IMP. Why is that useful?
>> STUDENT: I believe it is useful to have a separate statement because this one tests it as a
string. That is significant because if you are just trying to automatically convert it to a flow it might run into a difficulty when you type in a bunch of letters. Once it is a string you can see this is a string and it will not work with a flow.
>> PROFESSOR: Are you running for office? That is very well said. Very well said. You should become my assistant this semester. Very well said. Keeping track in IM P, as a variable. As a programmer you can manipulate that anyway you wish. In particular because that is stored
there. At the end you can actually print the value of that variable and tell the user this is what you print. And that is not correct. Raise your hand if you got this. Okay. Here in the exception, now you can print. It must be a number you enter. That. your program will crash.
The program is telling you what was the problem. That is one particular item here. It is not too much here except this new statement. The input capturing the variable. You can use it. Okay. Let's now try to do something, let's suppose this input, this numerical value. This is no different than what we saw before. Exactly the same thing you saw before. Capturing the
input into the variable. So, now this guy was asked to enter this or that and he typed 12. Any comments? He typed 12.5. What happened in this case? He is typing something, in complete
disagreement. You are asking to enter a, F to C or vice versa. He types ABC. He is not paying attention. We told you how to handle that. This is a different type of error. He is typing and integer F to C, or C to F. Or a number. It is a similar type of error that you can handle in a similar way. Okay. Now, that was one exception but in many cases you can have multiple exceptions. Your code becomes more complicated. It could be that type of error that
you would like to code yourself is - - to zero. Or maybe you were supposed to type a string and you type a number. These are two types of errors but there are many other possibilities.
It could very well be that you only have one exception, to exceptions. Maybe you have a exceptions to handle in your code. Any idea how to handle multiple exceptions? According to
the programming language constructs we have learned in this class until now? When you

have
something like this, this, this, or that. What is that? The Python construction that allows you to handle the sequence of conditions. What? I don't hear the answer. No. In code. You have condition one. In condition one you want to do something. If it is not condition one but
condition two and you want to do something. If it is not condition one, two but it is three. And you want to do something. Yeah?
>> STUDENT: If statement.
>> PROFESSOR: The if statement is if condition one, do this. Right? If condition two, do this. If condition three, do this. So the last one is correct. However, there is a better way to handle when you have a sequence of steps or a sequence of conditions. And you would
like to activate different pieces of code for each condition. If you raise the umbrella, switch. That is kind of a different name but you're close. Match case. But in some cases you cannot do that. That is a good answer. Very good. Match statement. To handle multiple exceptions you can use a match case. Is that clear? And again, that's no different to what we
described before. This is the pattern of a match case but in this case we are using the match
statement. With a try statement. Try, if the exception is one, do this. If this is the exception, do that. It is a match statement. And it happens to bethat Python allows you to do that very simplistic. This very different examples. But the bottom line is, if you write TRY column except if this is exception one, do this. If this is exception two, do that. Etc. As many as you want. It's a match case but now we try and accept. And these are examples for
that. There is another statement called raise, RAISE. Something you type in Python. Depending on the code you are implementing you can say you know what? This user sometimes
behaves knowing, and a non-intelligent way. What if this guy types something very strange? But I don't know exactly what is it. But he may. So you can type in your code, and exception,
RAISE and Tell the interpreter I want you to look at this. This becomes useful when code becomes big. Speaking of 100,000 lines of code. I'm talking about 200, then at some point depending on the logic you're using, this all becomes very useful. So here, again. The way to
lead these diagrams. The logic is very simple. Anytime you see a diamond it is what you are testing. Anytime you see a rectangle that is a piece of code that is going to do something. That is the way to understand this. The important thing here is that this Redpath tells you when are you going to get to the exception? So here in this example, this example is dealing
with body mass index which is something we mentioned. This is used when you go to the doctor.

It is your weight, your height, the ratio within your body mass index. So it is to numbers. Weight and height. What is the computation? Dividing two numbers. So to get a determinants,
this is not different to something like this. The two numbers, you should be able to tell me right now what are the exceptions that immediately you should take care of the program that
computes body mass index. The input is to numbers. Weight and height. What are the possible
exceptions you can think of? Come on, there is one written on the board. The denominator is
zero. That is an exception. You have to know how to take care of that exception. Tell me another one. Body mass index. Yeah. It's what? Okay, if it's a string than not is another case we can take care of. But suppose there's two numbers. You want? The number is, that is
an exception. Isn't it? You have to check if the number is negative. That's another exception. We have division by zero, and another exception for that case. This size is sleepy. He is representing the site. He's always answering. Another exception is excuse me. Body mass index. We assume body mass index is for dogs? Cows? Body mass index is for humans.
So what is a type of exception that is different to the first two? Yeah.
>> STUDENT: I noticed it may be unrelated but the weight and height being zero may raise an
exception.
>> PROFESSOR: Yes, the value of zero. Tell me this. The value is nonzero or negative. What is another type of exception? Give me an example. What? Try 300. Is that acceptable? No, there is a range of values. Does that make sense for this particular computation? So you can
check that out. Weight, 4000 pounds. As far as I know there is no human being like that. So weight has a range. Zero is an extreme case. Negative is an extreme case but for these computations you should be aware of what is the range for the values. And if you don't put this in the program will give you some cases of body mass index. For it to make sense you will
add this exception. So the way to do that is raise an exception. Type in Python RAISE. Weight must be positive. Height must be positive. There should be a nonzero somewhere. In
any case I will help you get the case of these diagrams. They are exactly the semantics of the
code. A way to see how this code will work. When problems start getting a little bit longer, it becomes a nuisance. Especially if you are not used to writing relatively complicated programs. It becomes a nuisance. Arrow here, and arrow here. I have 15 arrows. What am I going to do? It's hard to keep track of all of this. There is a wonderful tool not Python provides to you. To lock the arrows, so the system will lock the arrows in the - -. So it

will interpret, you know what? This is my coat in my exception but whatever you find write that into the file. Clear? Then in that way you can actually take a five and you can do your code in a better way because after that you take care of one arrow. This is not five, this is - -. Taking care of distance. A wonderful mechanism. Now you have specified that in Python.
So there is a library code, logging. You import that library. In that library, you specify whether it is a basic configuration. There is a method. Any time somebody does something or
some model does something that means this is a function inside the library. Something called
basic configuration. Something that is specified. In parentheses you specify certain parameters the important parameters for you to understand, this is the file name you wanted to
understand and open for you. You want, this is just too technical at this point. Write this down. This is a specification indicating I will be containing a bunch of errors. Now you have specified the format in which you want the system to tie the arrows in that five. This is just a specification of that. And now that is at the beginning of your code. When you put this utility interpreter I need and arrow. I need you to log out for me. So the system expects you to have something called tri-column as before. You have the exceptions as before. But whenever there is a pattern in the code that gets to an exception the system automatically, one
of this is on except oil. The valuable here. You can say log the arrow. The system will log the arrow. After that you can print, look at it, etc. A very simple mechanism. Again, in code, input, here, logging basic configuration with these parameters. Here is format, name of
the file, code. And you just have to say at the end, I want you to log the arrows. In that. And I want you to print the arrow for me. This is one way to use try, except to deal with multiple arrows. And this takes care of it. There is also called final. It is an area, no matter how horrible my coat is I don't know what the heck I'm doing. But you know what? At
the end of the day, finally I want you to do this for me. That is a statement you usually put at the end of your code so when you have something like try, accept, you have code finally and
it will always be executed no matter what happens here. It is a very useful thing. The same thing before, finally, we want to put thank you for using my body mass index calculator. After
that you can actually do whatever. Before - - when you put finally than this is what is going to happen. Welcome to CS 210. It worked more than expected but okay. Now, there are more
complicated ways to use finally. Stating that for completeness in the following patient. You don't have to, you just jump to the next section. When I asked the question before about this,

somebody said I can use if statements. Yes, you can simulate all of this and you were the one
that said document the idea is the language offers to you more elegant, more powerful mechanisms to deal with this distance. That is why we are introducing this to you. You may tell me you want to compare. When to do this or that. And the important aspects about this try accept mechanism is that it handles exceptions that are raised by the interpreter that you
don't see necessarily. The code you can see, you can see the distance can happen. Depending
how the code is, there may be many crazy things happening that you are unaware of. Try, accept. Allows you to get the benefit from this interpreter. Because it is going to tell you that. You use if else there are many things that could be hidden. Because those programs are
defined exceptions. But there are other things hidden and that is why it is better to use - -. So you can look at the example there and today, we are speaking about - - in mathematics. Which is a statement that usually related to inductive proofs. Inductive proofs, there is a statement that is true and if that statement is true then you can prove the statement is also true when you implement the value of a parameter. There are many cases in which certain statements are true and you very that to be checked by the interpreter. So you say a search condition and put the conditions. You are instructing the interpreter, no matter who you are,
Mr. interpreter. I want you to check this condition. If that is not then do this. How you express that, for example. You have something like this, float as assigned to the AL and you want this to be assert. I want this to be positive. If this code is doing something funny in the positive, it will be printed. Assert is saying I want this to be positive no matter what. You can put this in more useful places. This example is simple. If you are generating random
integers on the plane by using the random library and the metal in the library which generates
for you a random integer. Between these two parameters. In fact, that was the first, it was one I think that we introduced. And you want to put try before that. An exception before that. You can put here and assert statement. How do you interpret this? I want whatever's going to happen. In the code. I want the coordinates to be positive. You are telling the interpreter I want this to happen. If this is true my coat will be happy. It is going to go
to the end. If that is not true I want you to accept this is the construction, same with the Try. Indicate the exception has a problem. This is very simple. And very useful. Okay. So, I guess the quiz is ready. Can you put the, okay. Now the quiz is ready. It should be 10 or 12 minutes. 10 minutes, quiz. You can go to Canvas. If this is a question about this I will not answer now. It has to be after. First, Canvas. No? You should be in Canvas now. 10 minutes for the quiz. After you finish the quiz. Okay. The quiz now in Canvas. Quiz three. I prefer if during the quiz you don't talk to your neighbor. You have 10 minutes. You should already be going. It's anyone having trouble accessing the quiz? Anybody? It's

[Students taking quiz]
>> PROFESSOR: After you finish your quiz, don't highlight it. That is the link to go to sim alive. If you want bonus points there are going to be some questions. When you get to sim alive you have to declare yourself. Send a message to sim alive bot. With your net ID backwards.