In this section, we'll focus on tasks to process inputs, i.e. the tasks that take the input and generate a useful output, often through a series of steps. It is sometimes important for the model to reason in detail about a problem before answering a specific question, and if you took our previous course, ChatGPT Prompt Engineering for Developers, you will have seen a number of examples of this. Sometimes a model might make reasoning errors by rushing to an incorrect conclusion, so we can reframe the query to request a series of relevant reasoning steps before the model provides a final answer, so that it can think longer and more methodically about the problem. And in general, we call this strategy of asking the model to reason about a problem in steps, "Chain of Thought Reasoning". For some applications, the reasoning process that a model uses to arrive at a final answer would be inappropriate to share with the user. For example, in tutoring applications, we may want to encourage students to work on their own answers, but a model's reasoning process about the student's solution could reveal the answer to the student. Inner monologue is a tactic that can be used to mitigate this, and this is just a fancy way of saying hiding the model's reasoning from the user. The idea of inner monologue is to instruct the model to put parts of the output that are meant to be hidden from the user into a structured format that makes passing them easy. Then, before presenting the output to the user, the output is passed and only part of the output is made visible. So, remember the classification problem from a previous video, where we asked the model to classify a customer query into a primary and secondary category. And based on that classification, we might want to take different instructions. Imagine the customer query had been classified into the product information category. In the next instructions, we'll want to include information about the products we have available. And so, in this case, the classification would have been primary, general inquiry, secondary, product information. And so, let's dive into an example starting from there. So, let's start with our usual setup. So, for this inner monologue example, we'll start with our same delimiters that we've been using. And now, let's go through our system message. And so, what we're doing here is asking the model to reason about the answer before coming to its conclusion. So, the instruction is, "Follow these steps to answer the customer queries. The customer query will be delimited with four hashtags.", our delimiter. So, then we've split this up into steps. So, the first step is to "Decide whether the user is asking a question about a specific product or products. And a product category doesn't count.". Step two, so, "If the user is asking about specific products, identify whether the products are in the following list.". And now we've included a list of available products. So, here we have five available products. They're all varieties of laptops and these are all made up products. They were actually generated by GPT-4. And step three, "If the message contains products in the list above, list any assumptions that the user is making in their message. For example, that laptop X is bigger than laptop Y or that laptop Z has a 2 year warranty." for example. Step four is, "If the user made any assumptions, figure out whether the assumption is true based on your product information.". And step five is, "First, politely correct the customer's incorrect assumptions, if applicable. Only mention or reference products in the list of 5 available products, as these are the only five products that the store sells. And answer the customer in a friendly tone." And these kind of very pedantic instructions are probably unnecessary for a more advanced language model like GPT-4. And then we'll ask the model to use the following format. So, step one, delimiter, it's reasoning. Step two, delimiter, reasoning and so on. And using the delimiters will mean that it will be easier for us later to get just this response to the customer. And kind of cut off everything before. So, now let's try an example user message. So, our message is, "by how much is the BlueWave Chromebook more expensive than the TechPro desktop?" So, let's take a look at these two products. The BlueWave Chromebook is 249.99. And the TechPro desktop is actually 999.99. This is not actually true. And so, let's see how the model handles this user request. So, we'll format into our messages array. And we'll get our response. And then we'll print it. And so what we're hoping for is that the model takes all of these different steps and realises that the user has made an incorrect assumption and then follows the final step to politely correct the user. And so, within this one prompt we've actually maintained a number of different complex states that the system could be in. So, you know, at any given point there could be a different output from the previous step and we would want to do something different. For example, if the user hadn't made any assumptions in step 3, then in step 4 we wouldn't actually have any output. So this is a pretty complicated instruction for the model. So let's see if it did it right. So step 1, the user is asking a question about specific products. They're asking about the price difference between these two products. The user assumes that the BlueWave Chromebook is more expensive than the TechBook Pro and this assumption is actually incorrect. It's reasoning through, taking longer to think about the problem. In the same way that a human would also take some time to reason about an answer to any given question, the model performs better if it also has time to think. And so the final response to the user is the BlueWave Chromebook is actually less expensive than the TechBook Pro. The TechBook Pro desktop costs $999.99, while the BlueWave Chromebook costs $249.99. And so let's see another example of a user message. And also at this point, feel free to pause the video and try your own messages. So let's format this user message. So the question is, "do you sell TVs?". And if you remember in our product list, we've only listed different computers. So let's see what the model says. So in this case, step one, the user is asking if the store sells TVs, but TVs are not listed in the available products. So as you can see, the model then skips to the response to user step because it realizes that the intermediary steps are not actually necessary. I will say that we did ask for the output in this specific format. So technically, the model hasn't exactly followed our request. Again, more advanced models will be better at doing that. And so in this case, our response to the user is, "I'm sorry, but we do not sell TVs at the store." And then it lists the available products. So again, feel free to try some of your own responses. And so now, we only really want this part of the response. We wouldn't want to show the earlier parts to the user. So what we can do is actually just cut the string at the last occurrence of this delimiter token or string of four hashtags and then only print the final part of the model output. So let's write some code to get only the final part of this string. So we're going to use a try except block. To gracefully handle errors in case the model has some kind of unpredictable output and doesn't actually use these characters. And so we're going to say our final response is the response and then we're going to split the string at the delimiter string. And because we want the final occurrence, we just want to get the last item in the output list. And then we're going to strip any white space. Because as you can see, there might be white space after the characters. Then we're going to catch any errors and have a fallback response which is, "Sorry, I'm having trouble right now. Please try asking another question.". And then let's print our final response. And so as you can see, we just cut the string to get this final output. And so this is what we would show to the user if we were building this into an application. And overall, I just want to call out this prompt might be slightly convoluted for this task. You might not actually need all of these intermediate steps. And so why don't you try and see if you can find an easier way to do the same task in your own prompt. And in general, finding the optimal trade-off in prompt complexity requires some experimentation. So definitely good to try a number of different prompts before deciding to use one. And in the next video, we'll learn another strategy to handle complex tasks by splitting these complex tasks into a series of simpler subtasks rather than trying to do the whole task in one prompt.