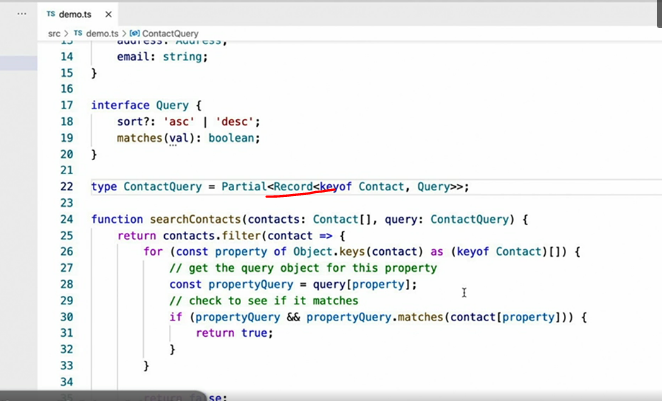
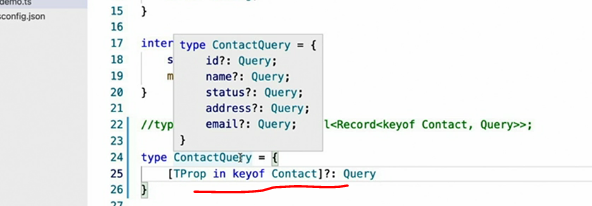
* - [Instructor] In the previous video, I introduced you to the record utility type, which allows you to construct a new type by referencing the fields of another type.



* Defining new types like this is an incredibly useful technique and using the record type to define them is a quick and easy way to create simple references, like the one I've created here.
* However, more often than not, the new type you're looking to create is more complex than what's shown here and in those cases, I like to use a different syntax called a **mapped** type.
* In its basic form, a mapped type definition looks like this.
* This mapped type definition uses a property indexer syntax, the open and close brackets, in conjunction with the key of syntax I used in the record based example, in order to enumerate all of the keys of the contact type.



* In other words, this example completely duplicates the previous type alias that used the record syntax, complete with the question mark at the end to make all of the mapped properties optional.
* *The reason I like the mapped type syntax better, however, is that while it does make the simple definitions like this one, look a little more complex, it actually allows you to make more complex definitions much more readable.*
* ***For instance, one of my all time favorite uses of this approach is to use the name of the key that I'm referencing to make one boiler plate line of code that adapts to each target property.***
* Let me show you what I mean.



* Let's revisit this query type that I created a while back, specifically, the type of parameter that gets passed into the matches method on line 19.
* Without an explicit type, TypeScript infers this parameter to be the **any** type.
* But remember, we want to avoid any wherever possible.
* So let's convert this to a generic type.
* And use that new generic type as the parameter type for the matches method.



* Once that's complete, I can go back to my mapped type to take advantage of this new generic parameter and pass the type of the mapped property like this.

Graphical user interface, text, application

Description automatically generated

* In order to explain what I just did, let me show you the resulting type.
* A mapped type where not only have all of the property names of the source type, contact in this case, been copied over but I've been able to reference the types of those properties as well.

Graphical user interface, text, application

Description automatically generated

* I did that using the same property index or syntax that I would use to access a value of an instance on the contact type, the open and close brackets.
* Why is all of this useful? Well, now I've got full static typing in my query objects.
* For example, when I provide a query function for the ID property, the query functions generic type parameter is number, the same type as the contact IDs property.
* Likewise, the parameter type of the query object for the name property is a string, the same as the contacts name property that it was originally mapped from.
* You may have noticed that TypeScript is now complaining that it can't adequately predict the type of value that will be retrieved when we iterate through these properties on line 34.
* I can fix that issue by telling it that it will be some kind of type from one of the properties on contact but it doesn't need to care about which one.
* Now that it's all working, you might be thinking that this approach seems a little complicated or advanced or maybe a little overkill.



* However, I find using mapped types in this way to be one of the most useful techniques to fill those static typing gaps where I might otherwise find myself defaulting back to the any type.
* I expect you'll be surprised at how often this will come in useful to you as well.