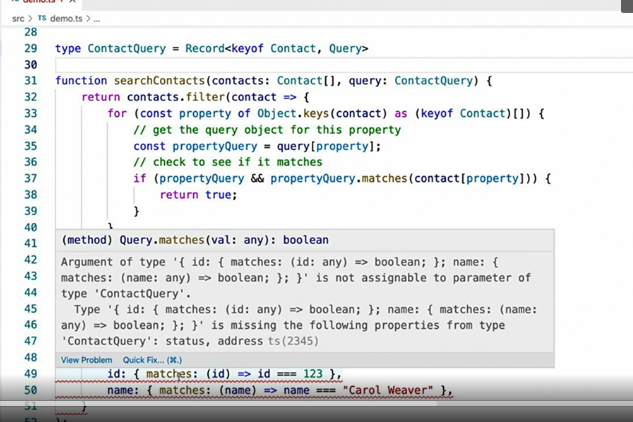
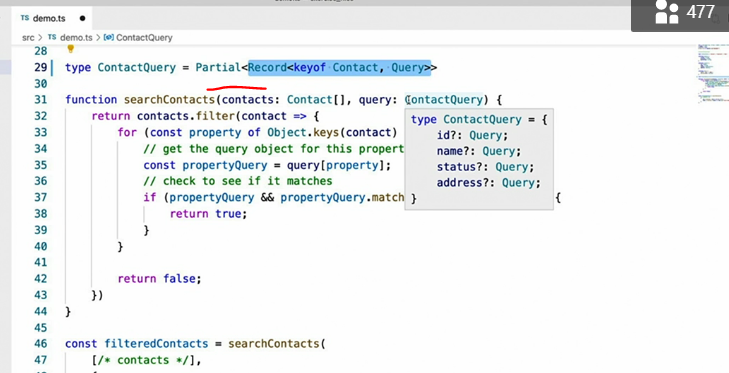
* - [Instructor] In a previous video, I left you hanging with a type script error that was caused by using the record type to define a query object whose properties were limited to the properties of the same name on the contact type.
* Now, that worked great to find the property that I previously had defined, which didn't match any of the properties on the contact type.
* Once I removed that property, however, I faced a new problem.
* Type script is now yelling at me that I haven't provided values for all of the properties of the contact type.



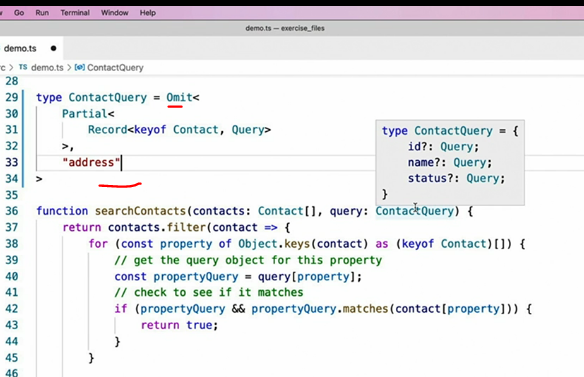
* The real problem here, though, is I don't think that this is a problem.
* I want to choose only a subset of those contact type properties in this query object.
* In this video, I'll show you a few utility types that allow you to extend from existing types, like record, but pick and choose which properties you want to include and exclude in your new type.
* First up, let's solve this current problem using the **partial helper type.**
* **The partial helper type is a generic type.**

****

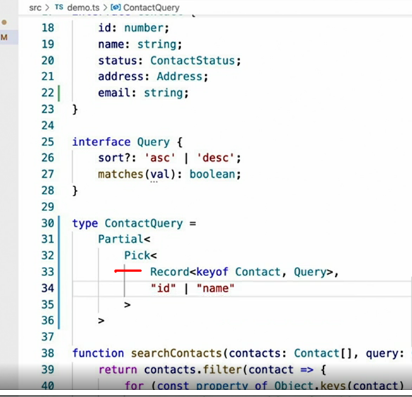
* So I'll use it to wrap my existing record-based type using this syntax.
* Notice how the error at the bottom immediately went away when I applied the partial type.
* This is because the partial helper type creates a new type that looks exactly like the type it wraps, but with all of its properties defined as optional.



* This means that you can define as many or as few of them, as you wish.
* In other words, now if I want to query only the ID and name properties and no other properties, like I'm doing on line 49 and 50, I can only define query objects for those specific properties.
* What if I want to do the opposite, however, and restrict the usage of certain properties.
* For example, what if I don't ever want to allow queries on the address property of the contact class.
* Well, there's another helper for that.
* And it's called **omit.**
* Like partial, the **omit helper** is a generic type that wraps another type and copies that type's definition, but, like its name suggests, it allows you to omit certain properties from that cloned definition.
* ***Unlike partial, however, the omit helper requires a second generic parameter, the list of property names that you'd like to omit.***
* For example, in order to omit the address property from this definition, I'd simply add it as the second generic parameter.



* And now I can see that it's no longer in the list of available properties.
* Of course, I can list as many properties as I like using a union type.
* With this definition in place, I can see that my contact query type now has only two possible properties, ID and name, and neither the address nor the status properties are available.
* And both of the remaining properties are optional due to the whole thing being wrapped in the partial helper.
* In this example***, omit is being used to exclude certain properties, leaving me with only a few.***
* This means as new properties are added to the wrapped type, the contact type, in this example, those new properties also become part of this new type.
* If that's not what I wanted, if I wanted to limit this type to only the ID and name properties, no matter what other properties get added to the contact type, I can use yet another helper named **pick.**
* As its name implies, **pick is the opposite of omit**.
* **It creates a new type using only the properties named by its second generic parameter.**
* In this example, I've limited the allowable fields to only ID and name, no matter what new fields may get added to the contact type in the future.



* And, finally, one last useful utility type is the **required** type, which is the opposite of the partial helper, turning all properties of a type to **required** properties, instead of optional.

Text

Description automatically generated

* Here, I've used the required type to wrap the contact query type that I've been demonstrating.
* And see that it has inherited the same properties as the contact query type, only those properties are now all required, instead of optional.
* I use these utility types all the time, especially the partial type, to quickly make new types based on existing ones, but modifying them slightly in order to make their properties optional or remove those properties all together.
* I expect these types will make plenty of appearances in your TypeScript code, as well.