* - [Instructor] Way back in the beginning of the course, I showed you how to use the any type to make TypeScript revert back to JavaScript's dynamic and unpredictable behavior of dealing with types.

Graphical user interface, text, application, email

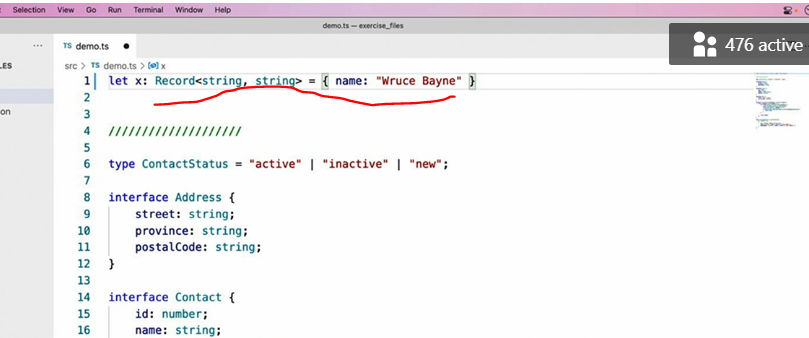
Description automatically generated

* It's particularly useful in dealing with situations like this, where you want to extend an object with properties that don't exist on the type that TypeScript originally inferred from the initialization of the variable on line one.
* In this case, just add any, and TypeScript is happy to stop yelling at you and let you go back to manipulating the object however you want.

Graphical user interface

Description automatically generated with low confidence

* The problem is that taking this approach is effectively opting out of type safety altogether.
* In other words, **describing a variable with the any type means it can quite literally be anything and TypeScript will never get upset.**
* I generally try to avoid using the any keyword whenever possible.
* In fact, there's a ***TypeScript configuration setting which allows you to trigger a compile error whenever any is applied.***
* One alternative that you can use to still retain some of the flexibility and dynamicism of the any type is the **record type.**
* The **record type** *is a very flexible type definition that allows you to define some structure and even some typing, without having to detail every possible property of the type you're trying to describe.*
* The record syntax is very simple.
* It's a generic syntax with *two generic parameters.*
* ***The first parameter is the possible property values, while the second is the possible property types.***
* For example, I can define a record with string properties and string values, like this.



* Note that my initialization continues to work just fine because the property name, name, is a valid string and it's assigned to a string value.
* If I try to assign one of these properties to a numeric value, however, TypeScript tells me that number is not assignable to string.
* It didn't have a problem with the property name I used.
* Number is a perfectly valid string value.

Graphical user interface, text, application, email

Description automatically generated

* However, it doesn't like that I'm trying to assign a numeric value to this property since I've only defined string values in my list of accepted property types.
* That's fine.
* I could just use a union to add number to that list.
* And then that error goes away too.
* And I can do the same with additional types as well.
* For example, I can add a boolean valued property and a function valued one too.
* And then add those types to the list of accepted types and watch the errors go away.

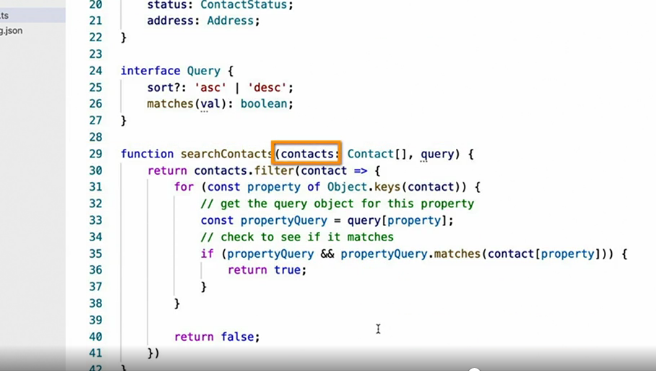


* Notice that other errors like when I try to assign the X variable to some completely different type are still very much there and still very legitimate warnings.

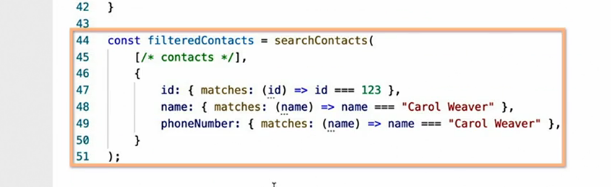
Graphical user interface, application, Word

Description automatically generated

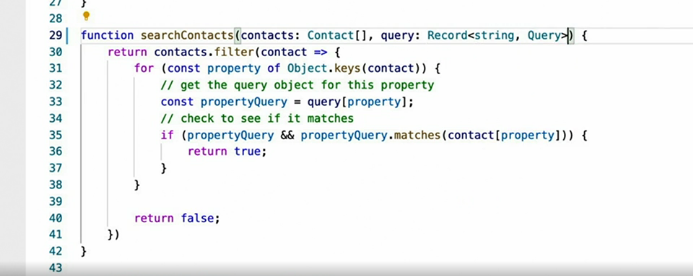
* I don't want to be able to assign X to some other type of value.
* Now that we've made the property types dynamic, let's take a look at the property names themselves.



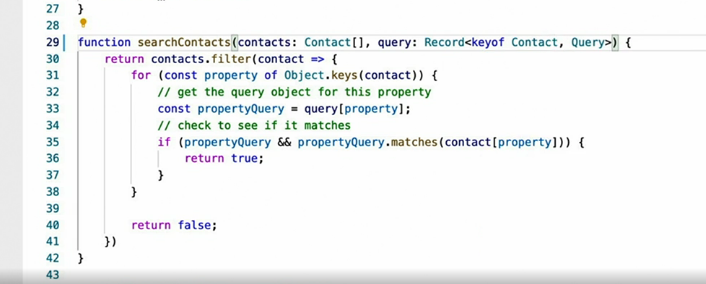
* Say I have this function that accepts an array of contact objects and a query object on which each property on the query object matches the name of a property on the contact type and contains metadata to filter the contacts by.
* This function iterates through the properties of each contact and compares them to a filter defined on the property with a matching name on the query object, if there is one.
* For instance, the contacts name property is compared with the name property on the query object.



* An example call to this method is shown below, right here.
* Now how can record help in this scenario? First I'll use the record syntax that I already showed, in order to define an object who can have properties of any name, but all of those properties values must be a query object.



* Honestly, this is really great and we could probably stop right here but let's make it specific to the properties on contact but let's limit the properties of the query object to the properties on the contact object by using our old friend, the keyof operator.

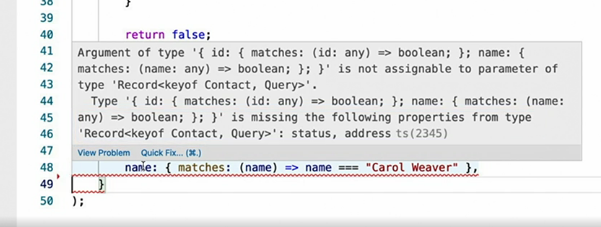


* And I can see as soon as I do this, TypeScript warns me that phone number is not a valid property on contact.

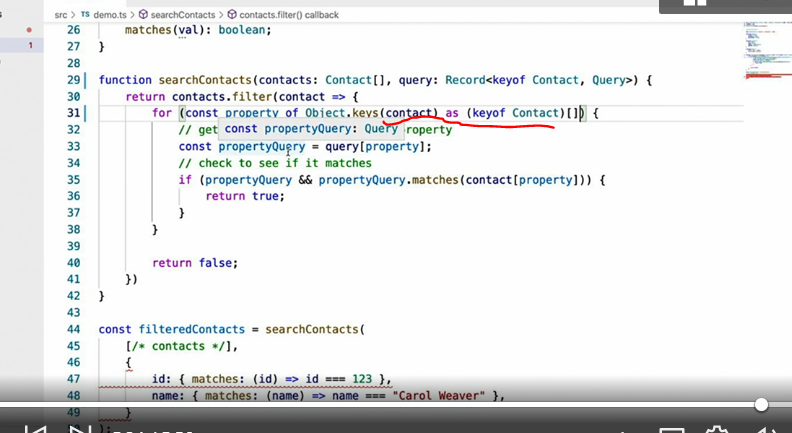
Graphical user interface, text, application

Description automatically generated with medium confidence

* So I'll remove it and now TypeScript yells at me with this not so straightforward error that I haven't provided a value for every property of the contact type.



* I'll get back to that later.
* The next thing I need to do is cast the call to object keys to the same type because for some reason, TypeScript is not yet smart enough to infer that.



* Once I do that, I can see that TypeScript can now infer that the type of the property query object is query, which means that matches must be a function.
* From these examples, I hope that I've proven just how simple and useful the record type can be in replacing the majority of places where you might otherwise give up and use the any type instead.
* And don't worry, I'll show you how to fix this error in another video.