Partial

This lesson explains the partial mapped type.

WE'LL COVER THE FOLLOWING

- The `Partial` mapped type and setting properties to *optional*

The `Partial` mapped type and setting properties to *optional*

TypeScript has a few mapped types that you can explore by looking in *lib.d.t.s*. The lib definition file comes in different flavors depending on the TypeScript version installed.

The `Partial` mapped type sets every property to optional. Instead of having one interface with all but one of the types required, only one type is needed. The creation of the real type, even with a mix of optional and read-only ought to be built. Only in the scenario where information can be partially assigned to an object that it is correct to have optional members.

Without mapped type, the duplication of a second interface is needed with all the same type functions and properties but with a question mark to specify that `undefined` can be set (or that value can be forgotten). Nevertheless, it eases maintainability to always synchronize both interfaces. The solution is to use the built-in mapped type `partial`.

```
type Partial<T> = {
    [P in keyof T]?: T[P];
}
```

Reading the partial mapped type section is a good recall to delve into what was introduced previously. If you remember, with the `readonly` mapped type, the code was looping with the `in` to add the mention `readonly` in front of

each property. As a reminder, here was the mapped type:

```
type ReadonlyInterface<T> = { readonly [P in keyof T]: T[P] };
```

With partial, similarly, we add the question mark to make every property optional. The scenario of having a partial type comes when you have the capability to edit an entity. A user might edit only a few fields on the overall entity, and so you might only receive a *partial* object with only the modified fields.