keyof to Validate a Member's Name

This lesson shows how to use keyof to extract information from a type or an interface.

WE'LL COVER THE FOLLOWING

Definition and goals

`keyof` is a subtype of String that allows extracting members' names from a type or from an interface. The main goal of `keyof` is to define a set of valid members' names. For example, you may have a public function that takes the direction as a string for convenience. Defining this parameter as a string would open the door to accepting anything. However, using the keyword `keyof` followed by the type that contains a union of the four cardinal points would limit the allowed string to those four values.

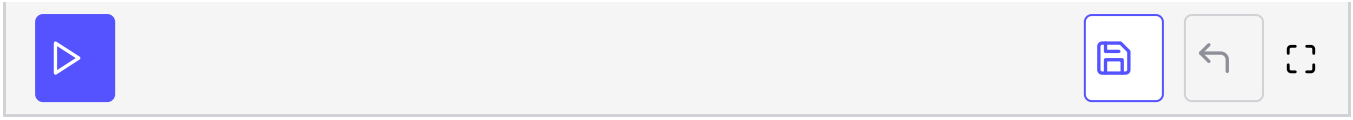
```
// Interface's members
interface InterfaceWithMembers {
    id: number;
    title: string;
    createdBy: Date;
}

const members: keyof InterfaceWithMembers = "id"; // Only accept id, title or createdBy

// Type's values
type TypeToKeyOf = "north" | "south" | "east" | "west";
function fKeyOfParameter(direction: TypeToKeyOf) {}
//fKeyOfParameter("no"); // Doesn't compile
fKeyOfParameter("north");
```

Use of generic

Generic can use `keyof`. For example, you can have a function that takes the first parameter of type `T` and the second parameter of type `R` that extends `keyof T`. The extend `keyof` of the second parameter means that it narrows down the potential value of the first argument. It is dynamic, and TypeScript provides full validation at design and transpilation time.

In the following example, at **line 3-5** the interface contains three members that are the only possible string values to pass as the second parameter. The `K` types are `K extends keyof T`, **line 8**. The return type of the function at **line 8** is the type of the member `K` of the object `T` which are both passed by parameter.

```
// Interface's members
interface InterfaceWithMembers {
    id: number;
    title: string;
    createdBy: Date;
}
const iWithMembersForKeyOf: InterfaceWithMembers = { id: 1, title: "1", createdBy: new Date()
function prop<T, K extends keyof T>(obj: T, key: K): T[K] {
    return obj[key];
}
const id = prop(iWithMembersForKeyOf, "id"); //the value 1 of type number
```



Anotherway to see what is happening with the previous example is to remove the generic. In the interface defined at **line 2** using `K extends keyof T` is similar to write `K extends keyof InterfaceWithMembers` if we remove the generic. So, `K` can be any of the three members defined by the interface at **line 2**: `id`, `title`, `createBy`. The return type `T[K]` means to take the type of the object passed in the first parameter and select the member `K`. If `id` is passed, the type is `number`, if `title` is passed, the type will be `string`.