String or Number Indexes

This lesson discusses the two ways to access an index.

The index signature can define members of an object with an identifier with square brackets.

```
myObject["memberName"] = "Value Here";
```

The identifier type can only be a `string` or a `number`. The identifier is the name between the bracket. In the previous example, it was `memberName`. In the following example, we can see an identifier of string and one with number;

```
myObject["String Here"] = {name:"anything"}; // string identifier
myObject[100] = true; // number identifier
```

Index signatures can be useful if you want to control the type of key for a dynamic assignment as well as the type to be returned.

For example, imagine having a custom map type that allows you to have a string key and return a specific type of object. Having a defined index signature allows for having a strongly typed key (string or number) that can hold an explicit type. In the following example, **line 2** defines the type of index signature. The identifier must be a `number` while the value is a `string`. **Line 4** demonstrates how to create a new object with two values with the identifier `100` and `200`.

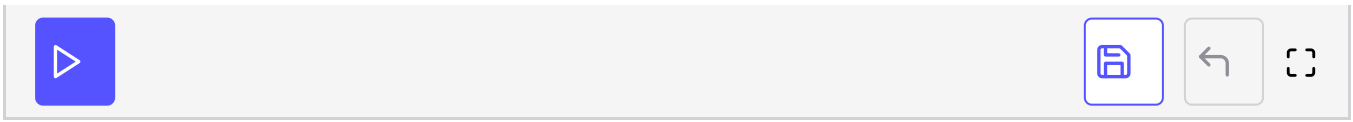
```
interface MyStringDictionary {
    [key: number]: string; // Only string is accepted for each number (identifier)
}
const dict1: MyStringDictionary = { [100]: "hundred", [200]: "two hundreds" };

console.log(dict1);
```

Index signature support generic type. The following example shows that the interface takes a `T` which can be any type. The `T` is then assigned to the expected value of the member at **line 2**.

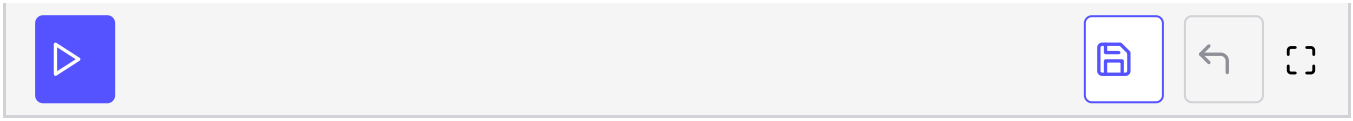
```
interface MyGenericDictionary<T> { // Generic for the value
    [id: string]: T; // The type T is the only accepted value
}
const dict2: MyGenericDictionary<string> = { ["100"]: "hundred", ["200"]: "two hundreds" };
const dict3: MyGenericDictionary<number> = { ["100"]: 100, ["200"]: 200 };

console.log(dict2);
console.log(dict3);
```



It makes it possible to define two indexes: one with a `string` key, one with a `number` key. However, beware that you'll have to change the return type of the `string` key to return not only the type you want the map to take, but also all the types of the object's members. TypeScript uses an index type of string, and is stricter.

```
interface MyStringDictionaryWithTwoMap {
    [key: number]: string;
    [key: string]: string;
    // [key: string]: number; // Doesn't compile
}
```



Example commented that demonstrate a mixed value type. It does not transpile.