Record

This lesson explains the record mapped type.

WE'LL COVER THE FOLLOWING

Description of Record

The `Record` type is also one of TypeScript's default mapped types. The `Record` type is a way to build a new type with several members of a single type. For example, if you need an object with three members of type string, you could do like line 2. The first parameter takes a union of desired fields, the second parameter is the type of these fields. Line 2 defines `m1`, `m2` and `m3` that are all of type `string`.

```
// Create a type with three string fields
type RecordType1 = Record<"m1" | "m2" | "m3", string>;
// Instantiate a variable from the type
const x: RecordType1 = { m1: "s1", m2: "s2", m3: "s3"};
console.log(x);
```

When to use Record

A pragmatic usage of the `Record` mapped type is when you are receiving user input data and need to convert it into a strongly typed object. This scenario is quite common if you take input from a web form. The user enters data as a string, but the business logic requires a type such as a number, boolean, etc.

The following code shows a snippet of how `Record` can be used with that purpose in mind. Keep a close eye on lines 14 and 25.

```
// An interface with many fields of many types
interface Animal {
    age: number;
    name: string;

    maximumDeepness: number;

    numberOfLegs: number;
    canSwim: boolean;
    runningSpeed: number;
}

// A function that need to take all the animal fields but from a string type
function receiveInputFromUser(dataIn: Record<keyof Animal, string>): Animal{
    const wellTypedObject: Animal = {
        age: Number(dataIn.age),
        name: dataIn.name,
        maximumDeepness: Number(dataIn.maximumDeepness),
        numberOfLegs: Number(dataIn.numberOfLegs),
        canSwim: Boolean(dataIn.age),
        runningSpeed: Number(dataIn.runningSpeed),
    }
    return wellTypedObject;
}
console.log(receiveInputFromUser({
    age: "13",
    name:"Fish",
    numberOfLegs: "2",
    maximumDeepness : "123",
    canSwim : "true",
    runningSpeed : "0"
}));
```

How it works

The example leverages `keyof` that returns a list of strings representing the members of the specified type. In that case:

```
dataIn: Record<keyof Animal, string>
```

is identical to:

```
dataIn: Record<"age" | "name" | "maximumDeepness" | "numberOfLegs" | "canSwim" | "runningSpeed", string>
```

without the possibility of typos and without necessary to keep all the strings synchronized with the `Animal` interface.