



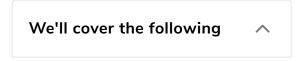






Exclude

This lesson explains the exclude mapped type.



- Exclude vs Extract
- Exclude vs Pick

Exclude VS Extract#

The Exclude mapped type is similar to Extract in the sense that it builds a type by selecting several properties. However, contrary to Extract, Exclude takes all the properties from a type and removes the specified one, instead of starting from nothing and adding the specified properties.

The following code snippet produces the exact same code as the one created in the previous lesson with <code>Extract</code>.

```
interface Animal {
 1
 2
        name: string;
 3
        gender: string;
 4
        sound: string;
    interface Human {
 7
        name: string;
 8
        gender: string;
 9
        nickname: string;
    }
10
11
12
    // type LivingThing = Extract<keyof Animal, keyof Human>;
    type LivingThing = Exclude<keyof Animal, "sound">;
    function sayMyName(who: Record<LivingThing, string>): void {
```

```
15 console.log(who.name +" is of type " + who.gender);
16 }
17 const animal: Animal = { name: "Lion", sound: "Rawwwhhh", gender: "Male };
18 const human: Human = { name: "Jacob", nickname: "Jaco-bee", gender: "Boy" };
19 sayMyName(animal);
20 sayMyName(human);
```

The commented-out **line 12** is the code from the Extract mapped type example. In this code, **line 13** replaces the functionality of **line 12**. The **line 12** was getting the intersection of properties between the Animal and the Human.

As you can see, all Animal type properties are provided with the keyof of the first generic parameter and the second parameter subtracts the sound property. The end result is a new type called LivingThing that has two properties called name and gender. Instead of specifying these two types from the Human interface and using Extract to get them from Animal, Exclude goes the other way around by removing what is specified.

That being said, we could also use keyof on the second generic parameter and achieve the same result.

```
interface Animal {
    name: string;
    gender: string;
    sound: string;
}
interface Human {
    name: string;
    gender: string;
    nickname: string;
}
interface NoisyLivingSpecies{
    sound: string;
}
```

```
}
type LivingThing = Exclude<keyof Animal, keyof NoisyLivingSpecies>;
```

Exclude VS Pick#

While the examples of Extract and Exclude borrow the Record mapped type, it is not a necessity. For example, we can create a HumanWithoutNickname type by using Exclude and Pick.

```
interface Animal {
    name: string;
    gender: string;
    sound: string;
}
interface Human {
    name: string;
    gender: string;
    nickname: string;
}
interface NoisyLivingSpecies{
    sound: string;
}

type LivingThing = Exclude<keyof Animal, keyof NoisyLivingSpecies>;
type HumanWithoutNickname = Pick<Human, LivingThing>;
```

The Exclude statement strips off the sound property of Animal. The LivingThing type is a union of the two remainings properties: "name" | "gender".

Then, using these two strings, Pick creates a type composed of two named properties from the associated types in the Human interface. In this case, two strings.

This is interesting, isn't it? The dynamic feature of mapped types allows for the creation of several types, depending on which mapped type is used and which type they are used against.

For example, the previous example created a <code>HumanWithoutNickname</code> of two properties defined by the <code>Exclude</code> with the type of <code>string</code>, because it was









built upon type Human that has these two string properties. Let's modify the code to use Pick on a different human type.

```
interface Animal {
    name: string;
    gender: string;
    sound: string;
interface Human {
    name: string;
    gender: string;
    nickname: string;
}
interface Human2 {
    name: string;
    gender: boolean;
    intelligenceScore: number
}
interface NoisyLivingSpecies{
    sound: string;
type LivingThing = Exclude<keyof Animal, keyof NoisyLivingSpecies>;
type HumanWithoutNickname = Pick<Human, LivingThing>;
type HumanWithoutNickname2 = Pick<Human2, LivingThing>;
```

In that example, Human2 has a property gender that is a boolean instead of a string. Pick takes this type along with the property and creates a HumanWithoutNickname2 without the two string properties as before, but with Record.

Instead, Pick maps a string for the name and a boolean for the gender. Because the Exclude from Animal took only the name and gender members, the intelligenceScore is not mapped. Move your cursor over the two types (lines 22 and 23) to see the difference.





Extract



! Report an Issue

