



ReturnType

This lesson explains the ReturnType mapped type.

We'll cover the following



- Extracting the return type of a function
- ReturnType with many return types
- ReturnType with asynchronous functions

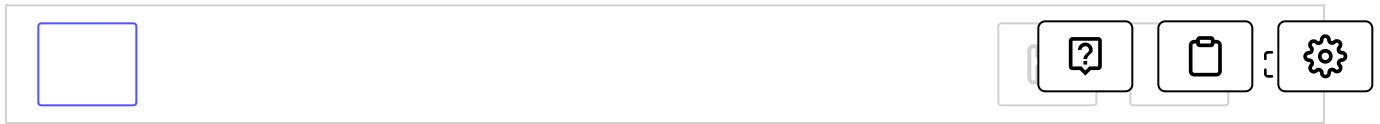
Extracting the return type of a function#

In some cases, you may want to extract the returned type of a function. TypeScript comes with a `ReturnType` mapping function that gives you this information.

For example, if you have a function that returns a `string`, you can use `ReturnType<yourFunction>` and it will return the type `string`. The following code on **line 5** assigns to a variable, the type which is a string because the return type of the function `getName` is a `string`.

```
1 function getName(): string {  
2     return "Name";  
3 }  
4 type FunctionType = ReturnType<typeof getName>;  
5 const varX:FunctionType = "This is a string";  
6 console.log(varX);
```





ReturnType becomes important in case the return type changes in the future. Then, the following code will not compile since TypeScript will change the **FunctionType** on **line 4** to the function's new return type. The following code does not compile as expected which demonstrates how safe TypeScript can be while maintaining source code.

```
function getName(): { firstName: string, lastName: string } {  
    return { firstName: "John", lastName: "Doe" };  
}  
type FunctionType = ReturnType<typeof getName>; // Not a string anymore  
const varX: FunctionType = "This is a string"; // TypeScript won't compile  
console.log(varX);
```



Return Type is modified, causing TypeScript to stop compiling

ReturnType with many return types#

The code above uses **typeof** to get the type signature of the function which is **()=>string**, and **ReturnType** gets the **string**. What about when the function does not explicitly specify a return type?

TypeScript can infer this information for you, even when the function can return several objects. In the example below, an object that is not defined is returned 50% of the time, and the other 50% of the time, an object with a similar field but different types, is returned.

```
function getSomething() {  
    if (Math.random() < 0.5) {  
        return {  
            cond: "under 0.5",  
            typeScript: true,  
        }  
    }  
}
```

```
    };  
  } else {  
    return {  
      cond: 1,  
      typeScript: "3.7",  
      moreField: true  
    };  
  }  
}  
type functionType2 = ReturnType<typeof getSomething>;
```



Moving your cursor on top on `functionType2` of line 15 shows:

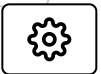
```
type functionType2 = {  
  cond: string;  
  typeScript: boolean;  
  moreField?: undefined;  
} | {  
  cond: number;  
  typeScript: string;  
  moreField: boolean;  
}
```

The interesting part is that TypeScript specified `moreField?: undefined` for the first part of the union. The reason is that the second part returns the member as `boolean`, but this is not in the first part. TypeScript returns a type that is balanced in terms of structure among the possible return values.

If a function returns two primitive types, a union of the values is returned. However, TypeScript is smart enough to narrow the type down if possible. For example, the following example does **not** return `number | string` but the literal `1 | "1"`.

```
function getSomething2() {  
  if (Math.random() < 0.5) {  
    return 1;  
  } else {  
    return "1";  
  }  
}  
type functionType3 = ReturnType<typeof getSomething2>;
```





ReturnType with asynchronous functions#

You may wonder what **ReturnType** returns in the case of an *asynchronous function*. An asynchronous function returns the *promise* of a type. For example:

```
async function asyncFunction(){  
    return await Math.random();  
}
```



This piece of code returns the type, **Promise<number>**. But, what if we wanted the type **number** which is the generic parameter? It is possible with a **conditional type**. The condition type is out of the scope of this lesson, but here is a glimpse of how it can be used in conjunction with **ReturnType**:

```
async function asyncFunction(){  
    return await Math.random();  
}  
  
type functionType4 = ReturnType<typeof asyncFunction>; // Promise<number>  
type functionType5 = ReturnTypeFromPromise<functionType4>; // number  
  
type ReturnTypeFromPromise<T> = T extends Promise<infer U> ? U : T;
```




Line 7 is where the *heavy lifting* happens. The condition type checks if the returned type **T** extends **Promise<?>**. If it does, it can **infer** the type into **U** and return it.

Otherwise, it returns the whole type. Without going any further into the conditional type, we can conclude that **ReturnType** is handy to extract a function's return type.





 **Back**

Exclude








Next 

Custom Mapped Type

☒ Mark as Completed

 Report an Issue

