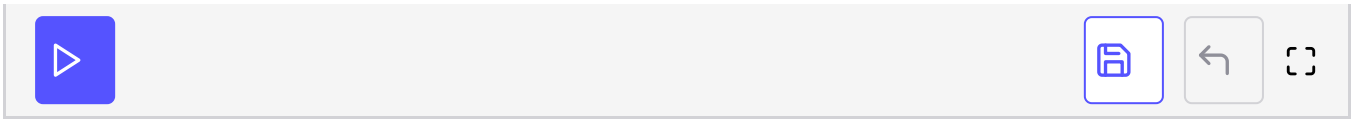
Function and Inference Variables

This lesson goes over functions and inference variables.

TypeScript can infer the type of a variable, and hence it is possible to avoid using the colon for anonymous functions by simply setting the variable to an unnamed function, that has type parameters and a return type. In the example below, all the `myAnonymous...` functions (**lines 5, 8, 11, 12**) have no type defined but they are all strongly typed by inference.

```
const inc = 1;
function myNamedFunction(p: number): number {
    return p + inc;
}
const myAnonymousFunc = function(p: number): number {
    return p + inc;
};
const myAnonymousFunc2 = (p: number): number => {
    return p + inc;
};
const myAnonymousFunc3 = (p: number): number => p + inc;
const myAnonymousFunc4 = (p: number) => p + inc;
```

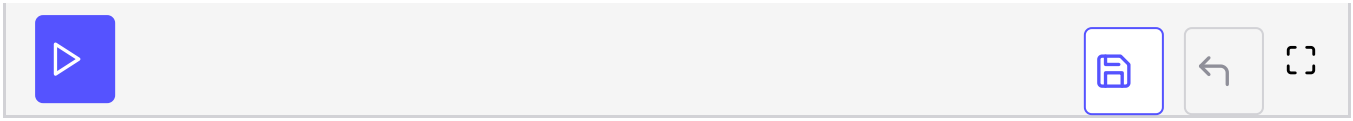


In the case of a function inside an interface, the name of a parameter doesn't need to match the definition and the function body of the implementation; only the type. In the following code, **line 2** defines a parameter named `p1` but the implementation name the same parameter `anotherNameForP1` at **line 6**. As long as the type is respected, the name does not matter.

```
interface MyInterface {
  myFunction: (p1: number) => void;
}

let myInterfaceWithDiffParams: MyInterface = {
  myFunction: (anotherNameForP1: number) => {
    console.log(`The parameter is ${anotherNameForP1}`);
  }
};
```

```
myInterfaceWithDiffParams.myFunction(100);
```



`String`, numeric, and `Boolean` literal types were not inferred; they needed an explicit declaration with a colon. Starting with **TypeScript 2.1**, literal types are always inferred for const variables and read-only properties.

```
const infHello: "hello" = "hello"; // Explicit
const infWord = "world"; // Implicit using inference to "world"

let infHello2: "hello" = "hello"; // Explicit
let infWord2 = "world"; // Implicit using inference to string

let worldString: string = "world";

let windeningToString: string = infHello; // Compile
```