Definition

In this lesson, you will see a high level definition of a function in TypeScript.

WE'LL COVER THE FOLLOWING

Functions and TypeScript

Functions are at the core of JavaScript. The language is function-scoped. TypeScript doesn't make any changes in this regard and embraces the use of a class to delimit scope, which is also a feature of **ECMAScript 2015**.

In this chapter, you will:

- Review how you can define functions in TypeScript and draw parallels to JavaScript.
- See how TypeScript enhances functions by providing strong signatures that define parameters and return types.
- Explore the outline of the `this` pointer which is often confusing but simplified with TypeScript.

Functions and keywords

The keyword `function` followed by the name of the function defines a new function. "Named function" is the explicit use of "function" to identify a function. With the same keyword, you can create an anonymous function

with no name. The anonymous function can use a variable for future

invocation. In both cases, the function can be used multiple times and return a single value.

The following example has at **line 1** a named function. It is invoked in **line 17** by using the name with parentheses. Similarly, the named function can be invoked by the variable name `f1`. Note that you cannot invoke by name. The only role of naming like **line 5** is to provide additional detail when debugging and having the stack trace naming the function.

The function defined at **line 9** is anonymous and can be invoked with the variable name `f2` like at **line 20**. The last example is an anonymous function that does not have a name, neither a variable to hold a reference to the function: it is invoked immediately at **line 15**.

```
function fctNamedFunction1() { // Named Function
  console.log("Named Function 1");
}

let f1 = function fctNamedFunction2() { // Named Function
  console.log("Named Function 2");
};

let f2 = function () { // Anonymous Function
  console.log("Anonymous Function 1");
};

(function () { // Anonymous Function + Automatically invoked
  console.log("Anonymous Function 2");
})();

fctNamedFunction1();
// fctNamedFunction2(); // Cannot call by name
f1();
f2();
```

In all previous examples, the keyword `function` was used. However, a function could be used without the keyword, for example, using the **"fat arrow"** `=>` or when used in a class. Both will be covered in this chapter.

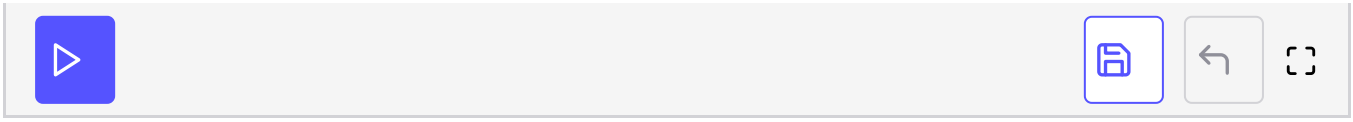
Functions and scope

TypeScript does not differ from JavaScript in defining the scope of a variable

consumed by a function. A function can access all defined functions within the function as well as all the ones outside of any parent scope.

Functions can be nested, which creates a whole set of possibilities around encapsulation strategies. For example, you can have a function that has two child functions. These two children can access the main function variables but cannot access each other's inner variables. Because both functions create a boundary since they define sibling scopes instead of parent-child scopes.

```
var function1 = () => { // Example of "fat arrow function"
  let variable1 = 1;
  var function2 = () => {
    let variable2 = 2;
    console.log(variable1 + variable2); // Access variable from function 1
  };
  function2();
};
function1();
```



In the upcoming lessons, you will see that functions can be categorized in two different families: *function expressions* and *function declarations*.