Argument Passing

In this lesson we will see how to pass arguments to functions in JavaScript.
Let's begin!

WE'LL COVER THE FOLLOWING

When you invoke functions, **all arguments, independently of their type, are passed by value**. The value passed to an argument is stored in a local variable that has meaning *only in the scope of the function*.

As soon as the function returns, the local variable goes out of the scope and is discarded. It means that even if you change the value of arguments with the function, only the local copies change, and the original values remain intact.

Listing 7-10 demonstrates this:

Listing 7-10: Passing arguments by value to

functions

```
<!DOCTYPE html>
<html>
<head>
  <title>Passing arguments #1</title>
  <script>
    function doubleMe(arg) {
      arg = arg + arg;
      return arg;
    }

    var num1 = 5;
    var num2 = doubleMe(num1);
    console.log(num1); // displays 5
    console.log(num2); // displays 10
  </script>
</head>
<body>
  Listing 7-10: View the console output
</body>
</html>
```

Although the highlighted code line in the body of `doubleMe()` changes the argument (arg) of the function, it changes only a local copy; it has no effect on `num1` when passing this variable to `doubleMe()`. So, the first log output will display 5 (the original value of `num1`) and the second will show 10 (the result of `doubleMe()` invoked with `num1`).

Passing values by arguments does not mean that functions cannot cause side effects on values passed to them. When you pass reference values (objects), you can easily check this, as shown in **Listing 7-11**.

Listing 7-11: Passing arguments by reference to functions

```
<!DOCTYPE html>
<html>
<head>
  <title>Passing arguments #2</title>
  <script>
    function setYearOfBirth(obj, year) {
      obj.yearOfBirth = year;
    }

    var me = new Object();
    setYearOfBirth(me, 1968);
    console.log(me.yearOfBirth);
  </script>
</head>
```

```
<body>
  Listing 7-11: View the console output


</body>
</html>
```

Although the `obj` and year arguments are passed by values in this listing, the `setYearOfBirth()` function causes a side effect (setting the `yearOfBirth` property of the argument), and the console output displays 1968.

Nevertheless, it did not change the value passed in the `obj` argument. The side effect is caused by the fact that objects are reference values, and even if the value is copied, the copy points to the original content of the object, which is stored on the heap.

To prove that the value of `obj` is really copied when it is passed to the function, add the highlighted code to `setYearOfBirth()`, as shown in **Listing 7-12**.

Listing 7-12: Passing arguments #2

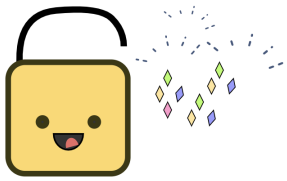
```
<!DOCTYPE html>
<html>
<head>
  <title>Passing arguments #2</title>
  <script>
    function setYearOfBirth(obj, year) {
      obj.yearOfBirth = year;
      obj = new Object();
      obj.yearOfBirth = 2001;
    }

    var me = new Object();
    setYearOfBirth(me, 1968);
    console.log(me.yearOfBirth);
  </script>
</head>
<body>
  Listing 7-12: View the console output
</body>
</html>
```

The first highlighted line instantiates a new object, stores it back in `obj`, and then the second line sets the `yearOfBirth` property of the newly created instance. This code snippet still displays 1968, because the `obj` argument is a copy of the reference value (`me`) passed to the function.

After the new object has been created within the body of `setYearOfBirth()`, the side effect (setting the `yearOfBirth` property) affects this new object, which is separate from the one passed to the function.

If the `me` instance would have been passed not by its value, but by reference, the `me` would have been automatically changed to point to the new instance created within the function, and then `me.yearOfBirth` would have been 2001.

Achievement unlocked! 🎉



Congratulations! You've learned about argument passing in JavaScript.

Amazing work!

Give yourself a round of applause! :)

In the *next lesson*, we will see what exactly a scope is in JavaScript.