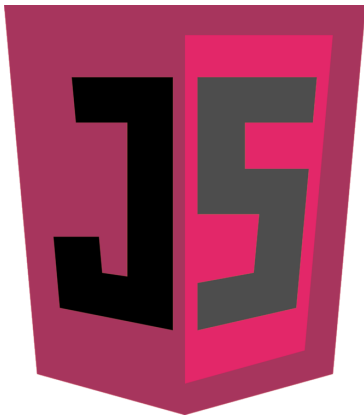


Functions are First Class Citizens

In this lesson, we'll study JavaScript functions and how they can be used in tandem with objects. Let's begin! :)

WE'LL COVER THE FOLLOWING

- [Listing-07-03](#): Assigning functions to objects
- [Listing-07-04](#): Defining functions with arguments



Assigning Functions to Objects



Object instances in JavaScript can have properties that **contain special objects, called functions.**

After you assign a function to an object, you can use the corresponding property to invoke that function

This is shown in Listing 7-3 below:

Listing-07-03: Assigning functions to objects #

```

<!DOCTYPE html>
<html>
<head>

<title>Functions</title>
<script>
  // Define a constructor for Car
  var Car = function (manuf, type, regno) {
    this.manufacturer = manuf;
    this.type = type;
    this.regno = regno;
    this.getHorsePower =
      function () { return 97; }
  }

  // Create a Car
  var car = new Car("Honda",
    "FR-V", "ABC-123");
  console.log("Horse power: " +
    car.getHorsePower());

  // It is a stronger car
  car.getHorsePower =
    function () { return 127; };
  console.log("Horse power: " +
    car.getHorsePower());
</script>
</head>
<body>
  Listing 7-3: View the console output
</body>
</html>

```

The `Car` constructor assigns a function to the `getHorsePower` property of a newly instantiated `Car`. This function returns `97`.

Later, as the highlighted code indicates, the `getHorsePower` property of `car` (a `Car` instance) is redefined to `return 127`. The code invokes this function *twice* with the `getHorsePower()` notation, the first time the one defined in the constructor, the second time the redefined one.

This is clearly shown in the console output:

JS console

```

97
127

```

JavaScript Console Output

Of course, you can define functions with arguments, as shown in Listing 7-4.

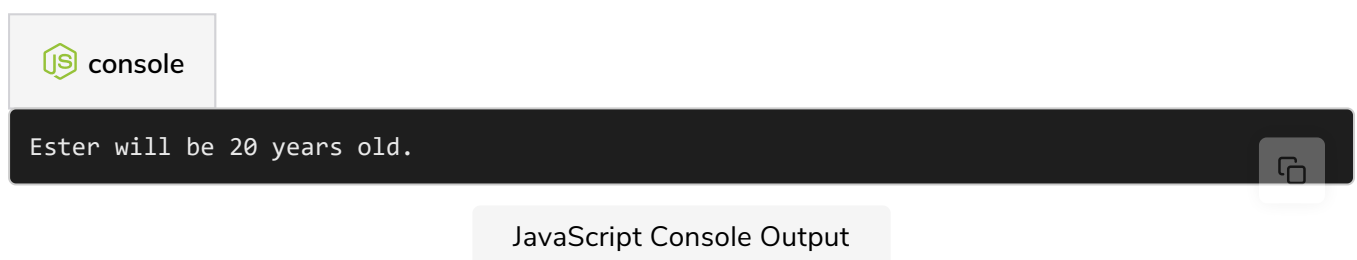
Listing-07-04: Defining functions with arguments

```
<!DOCTYPE html>
<html>
<head>
  <title>Functions with argument</title>
  <script>
    // Define a constructor for Child
    var Child = function (name, born) {
      this.name = name;
      this.born = born;
      this.ageInYear = function (year) {
        return year - born;
      };
    };

    // Create a Child
    var ester = new Child("Ester", 1996);
    console.log("Ester will be "
      + ester.ageInYear(2016)
      + " years old.");
  </script>
</head>
<body>
  Listing 7-4: View the console output
</body>
</html>
```

The `Child` constructor defines an `ageInYear` function that accepts a single argument. When you invoke this function on the `ester` instance, you can pass the argument, as shown in the highlighted code above.

This short code snippet produces the following console output:



As you can imagine, functions may have multiple arguments, and as you will learn later, those *arguments can be functions too*.

You probably won't be surprised if I tell you that functions

may retrieve functions.

But before going toward more abstract and advanced features, let's stay on the ground for now.

In the *next lesson*, we'll learn about regular expression literals and how to use these in JavaScript.

See you there!