Front-End v23.0

Back to 'Day 5 | Pre-work'

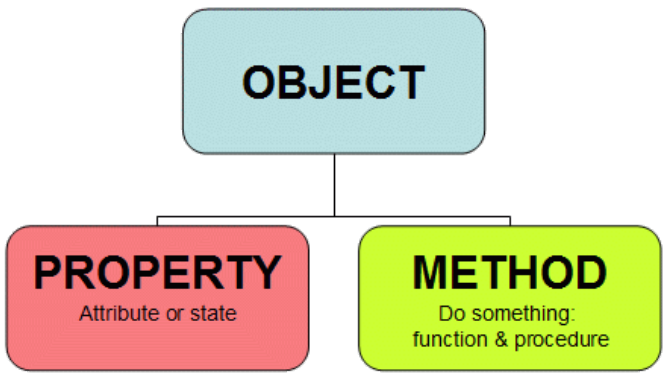
Javascript | Day 5 | Pre-Work

Table of contents:

JavaScript Objects

In JavaScript, most things are objects of some kind. All JavaScript values, except primitives (string, number, boolean, null and undefined) are objects.

When objects are used, a set of properties and methods are taken to create what we call a model. In an object, elements that you are already familiar with like variables and functions take on new names.
If a variable is part of an object, it is called a property. If we imagine a dog as a JavaScript object, both his name and age would be properties.
When we have multiple digitized "dog-objects" each individual dog might have a different name and a different age.
Similarly, if a function is part of an object, it is called a method **.** Methods represent tasks that are associated with the object. For instance, if you want to have your dog bark, i.e. print "bark" on your document, you can use a method to do this.



The object below represents a person. It has five properties (firstName, lastName, age, drivingLicense and hobbies) and one method (fullName). The method as it is a function, needs curly braces. The object is stored in a variable called person.

```
var person = {
    firstName: "john",
    lastName: "doe",
    age: 25,
    drivingLicense : true,
    hobbies: ['swim', 'read', 'coding'],
    fullName: function() {
        return this.firstName +' ' + this.lastName;
    }
}
```

The **this** syntax means that its reference to the object itself. As we are still in the object when we want to refer to a property inside this very object we need to use the syntax **this**.PropertyName.

Above you can see the person object. The object contains the following key/value pairs: firstName-john, lastName-doe… respectively. To see the entire object represented in the console, you may use console.table(). It will show a table with the object content. It can also be used for normal arrays.

```
console.table(person);
```

script.js:12

(index)	Value	0	1	2
firstName	"john"			
lastName	"doe"			
age	25			
drivingLicense	true			
hobbies		"swim"	"read"	"coding"

```
▼Object
    age: 25
    drivingLicense: true
    firstName: "john"
  ▶ fullName: ƒ ()
  ▶ hobbies: (3) ["swim", "read", "coding"]
    lastName: "doe"
  ▶ __proto__: Object
```

As you can see, **Objects consist of a set of key/value pairs** (previously we named them name/value pairs). This is just one of the ways you can create an object. Please keep in mind that in programming you can very often find key/value pairs.

If you want to reach a specific property like firstName you can type:

```
console.log(person.firstName);
```

If you want to call the fullName method you need to call it using parentheses:

```
console.log(person.fullName());
```

Let’s consider another way with which we can **create an object**:

```
var person = new Object();

person.firstName = "john";
person.lastName = "doe";
person.age = 25;
person.drivingLicense = true;
person.hobbies = ["swim", "read", "coding"];
person.fullName = function() {
  return this.firstName +' ' + this.lastName;
};
```

From the example above we see that we can **access the properties and methods of an object using dot (.) notation**. We can also access properties using square brackets:

```
console.log(person["firstName"]);
```

The two examples above do exactly the same thing. It's up to you to decide which one is clearer and better for you to use.
Let's consider one function program based on objects. First, we define an object, with its parameters and methods. And then, we call the methods to provide an output depending on the input value we give the method.

```
<html>
<body>
  <h3>info about a student</h3 >
  <h1 id="personName"></ h1>
  <h2 id="age"></ h2>
  <h3 id="message"></ h3>

  <script>
  var person = {
    firstName: "john",
    lastName: "doe",
    age: 25,
    drivingLicense : true,
    hobbies: ['swim', 'read', 'coding'],
    greetings: function() {
      return `Hi, My name is ${this.firstName } ${this.lastName}.`;
    }
  }

  // Print the first name of a person
  document.getElementById("personName").innerHTML = person.firstName;

  // Print the age property
  document.getElementById("age").innerHTML = person.age;

  // Call the greetings() method
  document.getElementById("message").innerHTML = person.greetings();

  </script>

</body>
</html>
```

What is Object Oriented Programming

Object Orientation is a paradigm used in software development paradigm that considers a program as a collection of objects. These objects can communicate and interact with each other through something called methods.
As you already know objects, we will now talk about another Element which can build on that: Classes.

Classes

In order to understand the idea behind JavaScript classes we need to understand the constructor functions and prototype. If we understand the constructor functions and prototype, it will be easy to understand the classes. This is because JavaScript classes are a new way for us to write the constructor functions by utilizing "the power of

prototype".

Let's take a look at an example:

```
//Define a class Person
class Person {
      firstName;
       lastName;
       //Define a constructor inside the class
      constructor (fName, lName) {
           this.firstName = fName;
          this.lastName = lName;
      }

        //Define the method inside the class
      fullName() {
          return  this.firstName + " " + this.lastName;
      }
  }
```

And in order to create a new instance of class Person, we do this:

```
//Define new instance of class Person
var person1 = new Person("Jenny" , "Doe");
```

```
<!DOCTYPE html>
<html>

< head>
  <title> Example</ title>
</head>

< body>
   <script type ="text/javascript">

  class  Person   {
       firstName;
       lastName;
      constructor(firstName, lastName) {
          this   .firstName = firstName;
           this.lastName = lastName;
      }

      fullName() {
          return  this.firstName + " " + this.lastName;
      }
  }

    var person1 = new Person("Jenny", "Doe"   );

   console.log(person1.fullName());

   </script >
</body>

</html >
```

This example proves that a class is a new way of implementing the constructor functions. Still, we need a few rules to work with that:

1. constructor requires the "new" keyword to work. To call the constructor, we need the following syntax:

```
var person1 = new Person("Jenny" , "Doe");
```

2. If you don’t add a constructor to a class, a default empty constructor will be added automatically. Same as the following:

```
constructor() {}
```

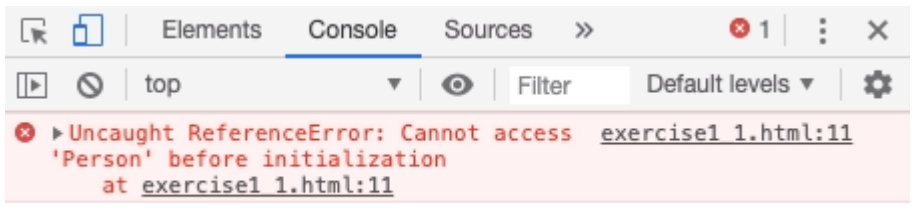
3. "Hoisting" in JavaScript means that certain elements are automatically moved to the top of the current scope, so for example you can use a variable before it is actually declared. This does not happen for classes and will return a Reference error if you try:

```
var person1 = new Person("Jenny", "Doe");

  class Person {
   constructor(firstName, lastName) {
        this.firstName = firstName;
       this.lastName = lastName;
   }

   fullName() {
        return this.firstName + " " + this.lastName;
   }
}
```

The code above will throw an error:



Class features

Constructor

Constructor is a function that is part of the class declaration. It defines a function that represents the class itself and is automatically called when you create the class.

```
var person1 = new Person("Jenny" , "Doe");
```

A class can not have more than 1 constructor function.

Subclassing

Subclassing is a way you can implement inheritance in JavaScript classes. The keyword "extends" is used to create a child class of a class.
Like in this example:

```
class Person {
  constructor (firstName, lastName) {
    this.firstName = firstName;
    this .lastName = lastName;
  }

  fullName() {
    return  this .firstName + " "  + this.lastName;
  }

}

class NewClass extends Person{
   age;

  constructor(firstName, lastName, age) {
    super(firstName, lastName);
    this.age = age;
  }

  fullName(){
    return "I am " + super.fullName() + ", and i am "+ this.age+ " years old";
  }
}

var person1 = new NewClass( "Jenny", "Doe" , 30);

console.log(person1.fullName());
```

As you can see we add a new property called **age** inside the NewClass. In the constructor we will add all the parameters from the parent class and the child class. From the total of 3 parameters, two will come from the parent class (**firstName** and **lastName**) and one (**age**) will be the **new parameter** . To use the parent class parameters in the constructor, the keyword **super()** needs to be applied as in the example above.

We will learn more about inheritance later. If you want to read up more about it however, you can find a deeper introduction here .

When we want to call a property or a method from the parent class in our new class, we need to use **super.**propertyname **or super.** methodname()**.**

◀ Solutions | JS Day 4

Jump to...

Javascript | Day 5 | Quiz ▶