Objects

Objects in JavaScript

Course Instructor

Laurence Svekis

Object

Group of values represented by name of key.

name:value pairs

Value can be any data type in JavaScript - number, string, boolean, array even functions also know as methods within the object. Arrays are in fact a type of object.

Objects can hold a collection of related data

```
var car = {};
car.color = ‘red’;
car.topSpeed = 300;
car.model = ‘mustang’;
car.company = ‘ford’;
car.year = 2012;
car.price = 50000;
```

Objects can contain many values

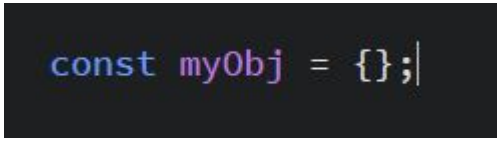
name:values pairs in JavaScript objects are called properties:

object literal

Object which uses curly brackets: {}

Using object literals is the more common and preferred method.

```
const myCar = {
        color: "red"
        , topSpeed: 300
        , model: "mustang"
        , company: "ford"
        , year: 2012
        , price: 50000
    };
```



object constructor

```
const myObj = new Object();

    myObj.one = "one";

    myObj.test = function(){

        console.log('testing works');

    }

    //myObj.test()
```

In Action

```
const myCar = {
        color: "red"
        , topSpeed: 300
        , model: "mustang"
        , company: "ford"
        , year: 2012
        , price: 50000
        , turnOn() { console.log('Your car is started');   }
        , drive() { console.log('You are driving');  }
    };
```

Dot notation vs Bracket notation

We accessed the object's properties and methods using dot notation.

Bracket notation is more dynamic

```
let a = 1;
const myBracket = {
    a1: "test 1"
    , a2: "test 2"
}

function test() {
    console.log(myBracket['a' + a]);
    a++;
}
```

```
top
> test()
  test 1
< undefined
> test()
  test 2
< undefined
>
```

```
> myCar
< ▶ {color: "red", topSpeed: 300, model: "mu
> myCar.color
< "red"
> myCar['color']
< "red"
> myCar["color"]
< "red"
>
```

Using Objects this methods and updating values

The this keyword refers to the current object the code is being written inside — so in this case this is equivalent to myCar. Using this it will ensure that the correct values are used within the context.

```
ipt>
const myCar = {
    color: "red"
    , topSpeed: 300
    , model: "mustang"
    , company: "ford"
    , miles :14000
    , year: 2012
    , price: 50000
    , turnOn() {
        console.log('Your car is started');
    }
    , drive() {
        console.log('You are driving');
    }
    , oilChange(){
        return `I needed that I have ${this.miles}`;

    }
};
```

```
myCar.oilChange()
"I needed that I have 14000"
myCar.miles += 5000
19000
myCar.oilChange()
"I needed that I have 19000"
>
```

Adding property to the object

myCar.miles = 40000

```
ipt>
const myCar = {
    color: "red"
    , topSpeed: 300
    , model: "mustang"
    , company: "ford"
    , miles :14000
    , year: 2012
    , price: 50000
    , turnOn() {
        console.log('Your car is started');
    }
    , drive() {
        console.log('You are driving');
    }
    , oilChange(){
        return `I needed that I have ${this.miles}`;

    }
};
```

```
> myCar.oilChange()
"I needed that I have 14000"
> myCar.miles += 5000
19000
> myCar.oilChange()
"I needed that I have 19000"
>
```

What Oooops.. Create Object with Constructor

They use the same construction values but are different.

The **new** keyword turns the function call into constructor call

```
const mySecondCar = myCar;
mySecondCar.miles = 100000;
console.log(mySecondCar.oilChange());
console.log(myCar.oilChange());

function Car(model,company,color){
    this.color = color;
    this.company = company;
    this.model = model;
    this.miles = 0;
    this.oilChange = function(){
        return `I needed that I have ${this.miles}`;

    }
}

let car1 = new Car('F150','ford','blue');
let car2 = new Car('Civic','honda','green');

car1.miles += 5000;
console.log(car1.oilChange());
console.log(car2.oilChange());
```

```
I needed that I have 100000
I needed that I have 100000
>
I needed that I have 100000
I needed that I have 5000
I needed that I have 0
> car1.miles = 500000
< 500000
> car1.oilChange()
< "I needed that I have 500000"
>
```

ES6 class object

```
class Car {
    constructor(model, company, color) {
        this.color = color;
        this.company = company;
        this.model = model;
        this.miles = 0;
        this.oilChange = function () {
            return `I needed that I have
            ${this.miles}`;
        }
    }
}
let car1 = new Car('F150', 'ford', 'blue');
let car2 = new Car('Civic', 'honda', 'green');
car1.miles += 5000;
console.log(car1.oilChange());
console.log(car2.oilChange());
```

```
I needed that I have 5000
I needed that I have 0
> car1
< ▼Car {color: "blue", company: "ford", model: "F150",
    color: "blue"
    company: "ford"
    miles: 5000
    model: "F150"
  ▶oilChange: ƒ ()
  ▶__proto__: Object
> car2
< ▼Car {color: "green", company: "honda", model: "Civi
    color: "green"
    company: "honda"
    miles: 0
    model: "Civic"
  ▶oilChange: ƒ ()
```

Methods in the Class

```
class Car {
    constructor(model, company, color) {
        this.color = color;
        this.company = company;
        this.model = model;
        this.miles = 0;
    }
    oilChange() {
        return `I needed that I have ${this.miles}`;
    }
}
let car1 = new Car('F150', 'ford', 'blue');
let car2 = new Car('Civic', 'honda', 'green');
car1.miles += 5000;
console.log(car1.oilChange());
console.log(car2.oilChange());
```

```
color: "blue"
company: "ford"
miles: 5000
model: "F150"
▶ __proto__: Object
> car1.oilChange()
< "I needed that I have 5000"
>
```

Better format notice that it is not in the values but you can still access it.

Arrays and Objects together

```
let myArray = ['Name', 50, false, 'First Name'];

const myObj = {

    list: myArray

};
```

```
let myArray = ['Name', 50, false, 'First Name'];
const myObj = {
   list: myArray
};
let test1 = [{
       person: {
          age: 40
          , name: "John"
       }
}, {
       person: {
          age: 25
          , name: "Jane"
       }
}
          ];
test1.push({
   person: {
      age: 50
      , name: "Jimmy"
   }
});
for (let x = 0; x < test1.length; x++) {
   console.log(test1[x].person.name);
}
```

Removing Object Properties

You can use delete to remove an object property

Looping values

```
const myCar = {
        color: "red"
        , topSpeed: 300
        , model: "mustang"
        , company: "ford"
        , miles: 14000
        , year: 2012
        , price: 50000
        , turnOn() {
            console.log('Your car is started');
        }
        , drive() {
            console.log('You are driving');
        }
        , oilChange() {
            return `I needed that I have ${this.miles}`;
        }
    };
for (let key in myCar) {
        console.log(myCar[key])
    }
```

```
for (let key in myCar) {
        console.log(myCar[key])
    }
```

See keys and values

You can get the keys and/or values this way

```
Object.keys(myCar)
▼ (10) ["color", "topSpeed", "model",
    0: "color"
    1: "topSpeed"
    2: "model"
    3: "company"
    4: "miles"
    5: "year"
    6: "price"
    7: "turnOn"
    8: "drive"
    9: "oilChange"
    length: 10
  ▶ __proto__: Array(0)
```

```
Object.values(myCar)
▼ (10) ["red", 300, "mustang", "ford", 14000, 2012, 50000, ƒ, ƒ, ƒ]
    0: "red"
    1: 300
    2: "mustang"
    3: "ford"
    4: 14000
    5: 2012
    6: 50000
  ▶ 7: ƒ turnOn()
  ▶ 8: ƒ drive()
  ▶ 9: ƒ oilChange()
    length: 10
  ▶ __proto__: Array(0)
```

JSON methods on JavaScript Objects

You can turn your object into a string but notice you lose you methods. JSON is based on JavaScript objects but not 100% the same.