

30 Days Of JavaScript: Objects



LinkedIn






Follow @asabeneh

Author: [Asabeneh Yetayeh](#)

January, 2020

[<< Day 7](#) | [Day 9 >>](#) Thirty Days Of JavaScript

-  [Day 8](#)
 - [Scope](#)
 - [Window Global Object](#)
 - [Global scope](#)
 - [Local scope](#)
 -  [Object](#)
 - [Creating an empty object](#)
 - [Creating an object with values](#)
 - [Getting values from an object](#)
 - [Creating object methods](#)
 - [Setting new key for an object](#)
 - [Object Methods](#)
 - [Getting object keys using Object.keys\(\)](#)
 - [Getting object values using Object.values\(\)](#)
 - [Getting object keys and values using Object.entries\(\)](#)
 - [Checking properties using hasOwnProperty\(\)](#)
 -  [Exercises](#)
 - [Exercises: Level 1](#)
 - [Exercises: Level 2](#)
 - [Exercises: Level 3](#)



Day 8

Scope

Variable is the fundamental part in programming. We declare variable to store different data types. To declare a variable we use the key word *var*, *let* and *const*. A variable can be declared at different scope. In this section, we will see the scope variables, scope of variables when we use *var* or *let*. Variables scopes can be:

- Global
- Local

Variable can be declared globally or locally scope. We will see both global and local scope. Anything declared without let, var or const is scoped at global level.

Let us imagine that we have a scope.js file.

Window Global Object

Without using console.log() open your browser and check, you will see the value of a and b if you write a or b on the browser. That means a and b are already available in the window.

```
//scope.js
a = 'JavaScript' // declaring a variable without let or const make it available in window object
b = 10 // this is a global scope variable and found in the window object
function letsLearnScope() {
  console.log(a, b)
  if (true) {
    console.log(a, b)
  }
}
console.log(a, b) // accessible
```

Global scope

A globally declared variable can be accessed everywhere in the same file. But the term global is relative. It can be global to the file or it can be global relative to some block of codes.

```
//scope.js
let a = 'JavaScript' // is a global scope it will be found anywhere in this file
let b = 10 // is a global scope it will be found anywhere in this file
function letsLearnScope() {
  console.log(a, b) // JavaScript 10, accessible
  if (true) {
    let a = 'Python'
    let b = 100
    console.log(a, b) // Python 100
  }
  console.log(a, b)
}
letsLearnScope()
console.log(a, b) // JavaScript 10, accessible
```

Local scope

A variable declared as local can be accessed only in certain block code.

- Block Scope
- Function Scope

```
//scope.js
let a = 'JavaScript' // is a global scope it will be found anywhere in this file
let b = 10 // is a global scope it will be found anywhere in this file
// Function scope
function letsLearnScope() {
  console.log(a, b) // JavaScript 10, accessible
  let value = false
// block scope
  if (true) {
    // we can access from the function and outside the function but
    // variables declared inside the if will not be accessed outside the if block
    let a = 'Python'
    let b = 20
    let c = 30
    let d = 40
    value = !value
    console.log(a, b, c, value) // Python 20 30 true
  }
  // we can not access c because c's scope is only the if block
  console.log(a, b, value) // JavaScript 10 true
}
letsLearnScope()
console.log(a, b) // JavaScript 10, accessible
```

Now, you have an understanding of scope. A variable declared with *var* only scoped to function but variable declared with *let* or *const* is block scope(function block, if block, loop block, etc). Block in JavaScript is a code in between two curly brackets ({}).

```
//scope.js
function letsLearnScope() {
  var gravity = 9.81
  console.log(gravity)

}
// console.log(gravity), Uncaught ReferenceError: gravity is not defined

if (true){
  var gravity = 9.81
  console.log(gravity) // 9.81
}
console.log(gravity) // 9.81

for(var i = 0; i < 3; i++){
  console.log(i) // 0, 1, 2
```

```
}  
console.log(i) // 3
```

In ES6 and above there is *let* and *const*, so you will not suffer from the sneakiness of *var*. When we use *let* our variable is block scoped and it will not infect other parts of our code.

```
//scope.js  
function letsLearnScope() {  
  // you can use let or const, but gravity is constant I prefer to use const  
  const gravity = 9.81  
  console.log(gravity)  
  
}  
// console.log(gravity), Uncaught ReferenceError: gravity is not defined  
  
if (true){  
  const gravity = 9.81  
  console.log(gravity) // 9.81  
}  
// console.log(gravity), Uncaught ReferenceError: gravity is not defined  
  
for(let i = 0; i < 3; i++){  
  console.log(i) // 0, 1, 2  
}  
// console.log(i), Uncaught ReferenceError: i is not defined
```

The scope *let* and *const* are the same. The difference is only reassigning. We can not change or reassign the value of the *const* variable. I would strongly suggest you to use *let* and *const*, by using *let* and *const* you will write clean code and avoid hard to debug mistakes. As a rule of thumb, you can use *let* for any value which change, *const* for any constant value, and for an array, object, arrow function and function expression.



Object

Everything can be an object and objects do have properties and properties have values, so an object is a key value pair. The order of the key is not reserved, or there is no order. To create an object literal, we use two curly brackets.

Creating an empty object

An empty object

```
const person = {}
```

Creating an object with values

Now, the person object has firstName, lastName, age, location, skills and isMarried properties. The value of properties or keys could be a string, number, boolean, an object, null, undefined or a function.

Let us see some examples of object. Each key has a value in the object.

```
const rectangle = {  
  length: 20,  
  width: 20  
}  
console.log(rectangle) // {length: 20, width: 20}
```

```
const person = {  
  firstName: 'Asabeneh',  
  lastName: 'Yetayeh',  
  age: 250,  
  country: 'Finland',  
  city: 'Helsinki',  
  skills: [  
    'HTML',  
    'CSS',  
    'JavaScript',  
    'React',  
    'Node',  
    'MongoDB',  
    'Python',  
    'D3.js'  
  ],  
  isMarried: true  
}  
console.log(person)
```

Getting values from an object

We can access values of object using two methods:

- using . followed by key name if the key-name is a one word
- using square bracket and a quote

```
const person = {  
  firstName: 'Asabeneh',  
  lastName: 'Yetayeh',  
  age: 250,  
  country: 'Finland',  
  city: 'Helsinki',  
  skills: [  
    'HTML',  
    'CSS',  
    'JavaScript',  
    'React',  
    'Node',  
    'MongoDB',  
    'Python',  
    'D3.js'  
  ],  
  isMarried: true  
}
```

```
'CSS',
'JavaScript',
'React',
'Node',
'MongoDB',
'Python',
'D3.js'
],
getFullName: function() {
  return `${this.firstName}${this.lastName}`
},
'phone number': '+35845454545'
}

// accessing values using .
console.log(person.firstName)
console.log(person.lastName)
console.log(person.age)
console.log(person.location) // undefined

// value can be accessed using square bracket and key name
console.log(person['firstName'])
console.log(person['lastName'])
console.log(person['age'])
console.log(person['age'])
console.log(person['location']) // undefined

// for instance to access the phone number we only use the square bracket method
console.log(person['phone number'])
```

Creating object methods

Now, the person object has getFullName properties. The getFullName is function inside the person object and we call it an object method. The *this* key word refers to the object itself. We can use the word *this* to access the values of different properties of the object. We can not use an arrow function as object method because the word *this* refers to the window inside an arrow function instead of the object itself. Example of object:

```
const person = {
  firstName: 'Asabeneh',
  lastName: 'Yetayeh',
  age: 250,
  country: 'Finland',
  city: 'Helsinki',
  skills: [
    'HTML',
    'CSS',
    'JavaScript',
    'React',
    'Node',
```

```

    'MongoDB',
    'Python',
    'D3.js'
  ],
  getFullName: function() {
    return `${this.firstName} ${this.lastName}`
  }
}

console.log(person.getFullName())
// Asabeneh Yetayeh

```

Setting new key for an object

An object is a mutable data structure and we can modify the content of an object after it gets created.

Setting a new keys in an object

```

const person = {
  firstName: 'Asabeneh',
  lastName: 'Yetayeh',
  age: 250,
  country: 'Finland',
  city: 'Helsinki',
  skills: [
    'HTML',
    'CSS',
    'JavaScript',
    'React',
    'Node',
    'MongoDB',
    'Python',
    'D3.js'
  ],
  getFullName: function() {
    return `${this.firstName} ${this.lastName}`
  }
}

person.nationality = 'Ethiopian'
person.country = 'Finland'
person.title = 'teacher'
person.skills.push('Meteor')
person.skills.push('SasS')
person.isMarried = true

person.getPersonInfo = function() {
  let skillsWithoutLastSkill = this.skills
    .splice(0, this.skills.length - 1)
    .join(', ')
  let lastSkill = this.skills.splice(this.skills.length - 1)[0]

```

```

    let skills = `${skillsWithoutLastSkill}, and ${lastSkill}`
    let fullName = this.getFullName()
    let statement = `${fullName} is a ${this.title}.\nHe lives in ${this.country}.\nHe teaches $
    return statement
  }
  console.log(person)
  console.log(person.getPersonInfo())

```

Asabeneh Yetayeh is a teacher.

He lives in Finland.

He teaches HTML, CSS, JavaScript, React, Node, MongoDB, Python, D3.js, Meteor, and SasS.

Object Methods

There are different methods to manipulate an object. Let us see some of the available methods.

Object.assign: To copy an object without modifying the original object

```

const person = {
  firstName: 'Asabeneh',
  age: 250,
  country: 'Finland',
  city: 'Helsinki',
  skills: ['HTML', 'CSS', 'JS'],
  title: 'teacher',
  address: {
    street: 'Heitamienkatu 16',
    pobox: 2002,
    city: 'Helsinki'
  },
  getPersonInfo: function() {
    return `I am ${this.firstName} and I live in ${this.city}, ${this.country}. I am ${this.ag
  }
}

```

```

//Object methods: Object.assign, Object.keys, Object.values, Object.entries
//hasOwnProperty

```

```

const copyPerson = Object.assign({}, person)
console.log(copyPerson)

```

Getting object keys using Object.keys()

Object.keys: To get the keys or properties of an object as an array

```

const keys = Object.keys(copyPerson)
console.log(keys) //['firstName', 'age', 'country','city', 'skills','title', 'address', 'getPe

```



```
const address = Object.keys(copyPerson.address)
console.log(address) //['street', 'pobox', 'city']
```

Getting object values using Object.values()

Object.values: To get values of an object as an array

```
const values = Object.values(copyPerson)
console.log(values)
```

Getting object keys and values using Object.entries()

Object.entries: To get the keys and values in an array

```
const entries = Object.entries(copyPerson)
console.log(entries)
```

Checking properties using hasOwnProperty()

hasOwnProperty: To check if a specific key or property exist in an object

```
console.log(copyPerson.hasOwnProperty('name'))
console.log(copyPerson.hasOwnProperty('score'))
```

🟡 You are astonishing. Now, you are super charged with the power of objects. You have just completed day 8 challenges and you are 8 steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.



Exercises

Exercises: Level 1

1. Create an empty object called dog
2. Print the the dog object on the console
3. Add name, legs, color, age and bark properties for the dog object. The bark property is a method which return *woof woof*
4. Get name, legs, color, age and bark value from the dog object
5. Set new properties the dog object: breed, getDogInfo

Exercises: Level 2

1. Find the person who has many skills in the users object.

2. Count logged in users, count users having greater than equal to 50 points from the following object.

```
const users = {
  Alex: {
    email: 'alex@alex.com',
    skills: ['HTML', 'CSS', 'JavaScript'],
    age: 20,
    isLoggedIn: false,
    points: 30
  },
  Asab: {
    email: 'asab@asab.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'Redux', 'MongoDB', 'Express', 'React', 'Node'],
    age: 25,
    isLoggedIn: false,
    points: 50
  },
  Brook: {
    email: 'daniel@daniel.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'React', 'Redux'],
    age: 30,
    isLoggedIn: true,
    points: 50
  },
  Daniel: {
    email: 'daniel@alex.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'Python'],
    age: 20,
    isLoggedIn: false,
    points: 40
  },
  John: {
    email: 'john@john.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'React', 'Redux', 'Node.js'],
    age: 20,
    isLoggedIn: true,
    points: 50
  },
  Thomas: {
    email: 'thomas@thomas.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'React'],
    age: 20,
    isLoggedIn: false,
    points: 40
  },
  Paul: {
    email: 'paul@paul.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'MongoDB', 'Express', 'React', 'Node'],
    age: 20,
    isLoggedIn: false,
    points: 40
  }
}
```

```

    }
  }
}

```

3. Find people who are MERN stack developer from the users object
4. Set your name in the users object without modifying the original users object
5. Get all keys or properties of users object
6. Get all the values of users object
7. Use the countries object to print a country name, capital, populations and languages.

Exercises: Level 3

1. Create an object literal called *personAccount*. It has *firstName*, *lastName*, *incomes*, *expenses* properties and it has *totalIncome*, *totalExpense*, *accountInfo*, *addIncome*, *addExpense* and *accountBalance* methods. Incomes is a set of incomes and its description and expenses is a set of incomes and its description.
2. **** Questions:2, 3 and 4 are based on the following two arrays:users and products ()

```

const users = [
  {
    _id: 'ab12ex',
    username: 'Alex',
    email: 'alex@alex.com',
    password: '123123',
    createdAt: '08/01/2020 9:00 AM',
    isLoggedIn: false
  },
  {
    _id: 'fg12cy',
    username: 'Asab',
    email: 'asab@asab.com',
    password: '123456',
    createdAt: '08/01/2020 9:30 AM',
    isLoggedIn: true
  },
  {
    _id: 'zwf8md',
    username: 'Brook',
    email: 'brook@brook.com',
    password: '123111',
    createdAt: '08/01/2020 9:45 AM',
    isLoggedIn: true
  },
  {
    _id: 'eefamr',
    username: 'Martha',

```

```

    email: 'martha@martha.com',
    password: '123222',
    createdAt: '08/01/2020 9:50 AM',
    isLoggedIn: false
  },
  {
    _id: 'ghderc',
    username: 'Thomas',
    email: 'thomas@thomas.com',
    password: '123333',
    createdAt: '08/01/2020 10:00 AM',
    isLoggedIn: false
  }
];

const products = [
  {
    _id: 'eedfcf',
    name: 'mobile phone',
    description: 'Huawei Honor',
    price: 200,
    ratings: [
      { userId: 'fg12cy', rate: 5 },
      { userId: 'zwf8md', rate: 4.5 }
    ],
    likes: []
  },
  {
    _id: 'aegfal',
    name: 'Laptop',
    description: 'MacPro: System Darwin',
    price: 2500,
    ratings: [],
    likes: ['fg12cy']
  },
  {
    _id: 'hedfcg',
    name: 'TV',
    description: 'Smart TV:Procaster',
    price: 400,
    ratings: [{ userId: 'fg12cy', rate: 5 }],
    likes: ['fg12cy']
  }
]

```

Imagine you are getting the above users collection from a MongoDB database. a. Create a function called `signUp` which allows user to add to the collection. If user exists, inform the user that he has already an account.

b. Create a function called `signIn` which allows user to sign in to the application

3. The products array has three elements and each of them has six properties. a. Create a function called rateProduct which rates the product b. Create a function called averageRating which calculate the average rating of a product
4. Create a function called likeProduct. This function will helps to like to the product if it is not liked and remove like if it was liked.

🎉 CONGRATULATIONS ! 🎉

<< [Day 7](#) | [Day 9](#) >>