JS | Nested data structures - arrays & objects

Learning Goals
Introduction
Nested data structures
Objects in objects
Array of arrays (two-di...
Array of arrays with ...
Time to practice
Summary
Extra Resources
Expand all
Back to top
Go to bottom

Learning Goals

After this lesson you will know:

- how to nest objects in objects and objects in arrays,
- how to nest arrays in objects and arrays in arrays,
- how to use these nested data structures.

Introduction

In the previous lessons, we learned how to work with objects and arrays. Let's shortly recap:

- Arrays:

```
let books = ['Eloquent JavaScript', 'Secrets of the JavaScript Ninja'];
console.log(books[0]); // => Eloquent JavaScript
console.log(books[1]); // => Secrets of the JavaScript Ninja
console.log(books[2]); // => undefined
```

- Objects:

```
let eloquentJS = {
  title: 'Eloquent JavaScript',
  author: 'Marijn Haverbeke'
};

console.log(eloquentJS.title); // => Eloquent JavaScript

let secretsOfNinja = {
  title: 'Secrets of the JavaScript Ninja',
  author0: 'John Resig',
  author1: 'Bear Bibeault' // two authors... interesting...
};

console.log(secretsOfNinja.author0); // => Bear Bibeault
```

Okay, now when we refreshed our memory how arrays and objects look like and how we can retrieve values from them, let's see how we can combine them. And why we should do that.

Nested data structures

Having all data is not always enough. We have to know how to **organize** that data so we can use it easily and maintain the state of our data clean and understandable as much as possible.

Objects in an array (aka array of objects)

Let's take a look into our previous examples. The array of books and objects that contain the details about each book are correlated, but they exist as separate variables. We can merge it into just one variable and have an array of objects instead:

```
const books = [
  {
    title: 'Eloquent JavaScript',
    author: 'Marijn Haverbeke'
  },
  {
    title: 'Secrets of the JavaScript Ninja',
    author0: 'John Resig',
    author1: 'Bear Bibeault'
  }
];
```

Looks good. How can we reach in and grab some data?

- Getting the data from arrays of objects

```
console.log(books[0]);
// { title: 'Eloquent JavaScript', author: 'Marijn Haverbeke' }
```

Since we are working with arrays, we use `[index]` notation to access the data. Based on the previous console.log(), we can see that we got the whole object back. Well, this is easy - when working with objects, we can just simply use `.` notation to get the data:

```
console.log(books[0].title);
// 'Eloquent JavaScript'
```

- Adding/removing the data to/from arrays of objects

We still can use the same methods we covered in one of the previous lessons:

- *push()*
- *unshift()*
- *pop()*
- *shift()*

Let's see the example:

```
let jsDesignPatterns = {
  title: 'Learning JavaScript Design Patterns',
  author0: 'Addy Osmani',
};
books.push(jsDesignPatterns);
console.log(books);

// [
//   { title: 'Eloquent JavaScript', author: 'Marijn Haverbeke' },
//   { title: 'Secrets of the JavaScript Ninja',
//     author0: 'John Resig',
//     author1: 'Bear Bibeault' },
//   { title: 'Learning JavaScript Design Patterns',
//     author0: 'Addy Osmani' }
// ]
```

🏋 If you would like to practice a bit before you move forward, go to the end of the lesson, on the *"Time to practice"* section and work on the first assignment.

Arrays in objects

Again, let's go and take a look into our previous books array. One of the books has two authors, so it would be logical to place them in the same property that will hold both of them. Well, that means we would have an array of authors instead of two separate properties:

```
const books = [
  {
    title: 'Eloquent JavaScript',
    author: 'Marijn Haverbeke'
  },
  {
    title: 'Secrets of the JavaScript Ninja',
    authors: ['John Resig', 'Bear Bibeault']
  }
];
```

And here is how we can access to these values:

```
console.log(books[1].authors);
// [ 'John Resig', 'Bear Bibeault' ]

console.log(books[1].authors[0]); // => John Resig
console.log(books[1].authors[1]); // => Bear Bibeault
```

> At the end of the road, the same rule has been applied all over again:
>
> - to access the property of the object use `.` (dot) notation, and
> - to access the element of the array, use `[index]` notation.

🏋 If you would like to practice a bit before you move forward, go to the end of the lesson, on the *"Time to practice"* section and work on the second assignment.

Objects in objects

Another way of organizing data could be to place objects inside objects. Let's see an example:

```
let currentGroup = {
  course: 'Web Development',
  type: 'full-time',
  squadNumber: 'squad-207',
  classroom: {
    teacher: {
      name: 'Rick',
      age: 27
    },
    classroom: {
      floor: 2,
      seats: 35,
      available: true
    }
  }
};
```

As we can see, *teacher* and *classroom* are both embedded objects inside *currentGroup* object. Here is how we can get the values we want:

```
console.log(currentGroup.teacher); // => { name: 'Rick', age: 27 }
console.log(currentGroup.teacher.name); // => Rick
console.log(currentGroup.classroom.available); // => true
```

> When having objects nested inside other objects, we use `.` (dot) notation to access the values.

🏋 If you would like to practice a bit before you move forward, go to the end of the lesson, on the *"Time to practice"* section and work on the third assignment.

Array of arrays (two-dimensional array)

Sometimes it makes sense to organize the data in such a way, so it represents multiple arrays saved inside just one variable. And the type of that variable is an array as well.
Let's take a look:

```
const books = [
  ['Eloquent JavaScript', 'Secrets of the JavaScript Ninja'],
  ['Learn Python the hard way', 'Real Python Courses'],
  ['Effective Java', 'Java Generics and Collections']
];
```

> In two-dimensional arrays, to reference an element, is to reference an entire (inner) array.

```
console.log(books[1]);
// => [ 'Learn Python the hard way', 'Real Python Course' ]
```

Referencing the second element of the books array, gave us back the whole array with Python books. If we would like to get the elements of this array, we would have to keep using the same notation (`[]`) since we still working with an array:

```
console.log(books[1][0]); // => Learn Python the hard way
console.log(books[4][1]); // => undefined
```

🏋 If you would like to practice a bit before you move forward, go to the end of the lesson, on the *"Time to practice"* section and work on the fourth assignment.

Bonus: 2D arrays with nested objects

Very often, you will be dealing with some data structures that are beyond the regular level of complexity. This being said, you might run into an array of arrays that contains nested objects with other nested arrays and objects. Well... pretty much all the previously covered but in one!
Let's take a look and if this is not too clear, don't spend too much of your time on it now. Later in the course we will cover this again:)

```
let basic = {
  language: 'JavaScript',
  frameworks: [
    {
      end: 'back',
      list: [
        {
          name: 'Express2k',
          released: 2010
        },
        {
          name: 'MeteorJS',
          released: 2012
        }
      ],
    },
    {
      end: 'front',
      list: [
        {
          name: 'ReactJS',
          released: 2013
        },
        {
          name: 'VueJS',
          released: 2014
        }
      ]
    }
  ]
};
```

Let's destructure a bit:

- the variable named *basic* has a type *object*, and it has two properties:
  - *language* - type of string and
  - *frameworks* - type of array
- *frameworks* is array of objects
- each object has two properties:
  - *end* - type of string (back, front) and
  - *list* - type of array
- *list* has two objects, and each has two properties: one type of string (name) and one type of number (released)

Let's see how we can access the data:

```
console.log(basic.frameworks);
// gives us back the ARRAY with TWO OBJECTS
// => [
//   { end: 'back', list: [ [Object], [Object] ] },
//   { end: 'front', list: [ [Object], [Object] ] }
// ]

// DON'T WORRY BECAUSE OF [Object] syntax, it just represents more complexed structure inside. Will be covered later in the course.
```

Let's move deeper - we work with array so we know we need to use `[index]` to access to nested objects:

```
console.log(basic.frameworks[1]);
// => {
//   end: 'front',
//   list: [
//     { name: 'ReactJS', released: 2013 },
//     { name: 'VueJS', released: 2014 }
//   ]
// }
```

We see that the property *list* has type *array* so to access its elements we have to do the following:

```
console.log(basic.frameworks[1].list[0]);
// => { name: 'ReactJS', released: 2013 }
```

It might seem a bit overwhelming, but more you practice, easier it will be.
🏋 In case you want to go straight to practice, go to the *Iteration 5* of the following *Time to practice* section.

📝 Time to practice

Open a new CodePen and do the following:

1. Using the given array of objects:

```
let products = [
  {
    name: 'iPhone',
    price: 799.99
  },
  {
    name: 'Samsung Galaxy S10',
    price: 900.00
  }
];
```

2. Given the array, print:

- your course type (full-time or part-time)
- the most familiar topic
- the least familiar topic

```
let course = {
  name: 'Web Development',
  type: ['full-time', 'part-time'],
  topics: ['HTML/CSS & Responsive Design', 'JavaScript', 'MongoDB', 'Node', 'Express', 'React']
};
```

3. Given the object with nested objects in it, print:

```
let student = {
  firstName: 'Ana',
  lastName: 'Blair',
  course: {
    name: 'Web Development',
    type: 'part-time'
  },
  address1: 'Madrid',
  address2: {
    street: 'Happy Street',
    number: 123,
    city: 'Barcelona',
    zip: 08017,
    country: 'Spain'
  }
};

// console.log(???); // => Web Development
// console.log(???); // => Happy Street
// console.log(???); // => Spain
// => Ana moved from Barcelona to Madrid to take part-time Web Development course...
```

4. Given a 2D array, print the following:

```
const ironCompanies = [
  ['Mexico City', 'Miami', 'San Paulo'],
  ['Amsterdam', 'Barcelona', 'Berlin', 'Lisbon', 'Madrid', 'Paris']
];

console.log(ironCompanies[0][2]); // => Miami
console.log(ironCompanies[1][3]); // => Amsterdam
console.log(ironCompanies[1][2]); // => Paris
```

5. Use the example from the lesson with frameworks to retrieve the following:

```
// console.log(???); // => Express2K
// console.log(???); // => In IronHack, I'll learn Express2K and ReactJS.
```

Summary

In this lesson, you've learned how to use and access to data inside different combinations of nested data structures using arrays and objects. Now you know how to write and use:

- objects inside arrays,
- objects inside other objects,
- arrays inside objects,
- arrays inside other arrays and
- all of these mixed together and with strings, numbers, and booleans.

Extra Resources

- Accessing Nested Objects in JavaScript
- Basic JavaScript: Accessing Nested Objects - freeCodeCamp
- JavaScript Multidimensional Array