CS 4220

- Current Trends in Web Development -

Application Design and Development with Node.js

Prof. Cydney Auman

AGENDA

01 Intro to Objects

02 Object Iteration & Comparison

03 Object Destructuring

04 ES6 Spread versus Rest

Review

We discussed operators and type coercion. We talked about implicit and explicit conversion.

We looked JavaScript Strings focusing on both the use of template literals as well as various methods we can use with Strings.

We learned the meaning of mutation in JavaScript. This was pair with a code demo showing JavaScript primitive types are immutable but that arrays are mutable. We also worked on a code demo showing several array methods.

We took our first look at functions - comparing function declaration with function expression and also the function expression with arrow notation.

Objects

An **object** is a collection of related data consisting of key/value pairs. These key/value pairs can be added or removed at anytime.

One way to create an object is by using a curly brace notation. An object like this is referred to as an object literal — we are writing out the object contents as we create it.

To create an object literal, you only need to declare a variable and then set it equal to `{}`. Because this is object literal notation, properties and values can be added on declaration. The key/property is on the left side of the : and the value for that key is on the right side.

```
const course = {
    department: 'Computer Science',
    number: 4220
};
```

Object Binding and Mutation

When using the keyword const to declare an object. The variable will continue to point and reference the same object, *however* the contents of that object can change. Objects and arrays are mutable - the state can be modified after they are created.

```
const course = {
    department: 'Computer Science',
    number: 4221
};

// this change is allowed and perfectly acceptable
course.number = 4220

// this change is NOT allowed as this is re-assigning the object
course = {
    department: 'Computer Science',
    number: 4220
};
```

Objects Access

There are two ways to access a key's value from an object. **Dot notation** and **bracket notation**.

Dot notation - specify the variable/name given to the object followed by a dot and then by the property/key name.

Bracket notation - specify the variable/name given to the object , then a bracket followed by a variable or string which represents the property/key name and then close the bracket.

```
const course = {
   department: 'Computer Science',
   number: 4220
};
// uses dot notation
console.log(course.department)

// uses bracket notation with string
console.log(course['department'])
```

Bracket Notation Notes

In JavaScript **bracket notation** is used for both Array access using the index OR object access by using the key.

As we start working with both objects and arrays - it is important to keep in mind when using bracket notation which data structure we are working with.

Arrays will use the index in order to access a value inside an array.

```
const arr = ['mercury', 'venus', 'earth']
console.log(arr[0]) // mercury
```

Objects will use the key in order to access the value inside the object.

```
const course = {
   department: 'Computer Science',
   number: 4220
};
console.log(course['department']) // Computer Science
```

Objects Add & Remove Properties/Keys

Adding Properties/Keys

To add properties to an object the format is generally the object name followed by a dot and then by the key name. Alternatively, bracket notation can be used as well.

```
course.room = 'KH2005';
course['startTime'] = '6:00pm';
```

Removing Properties/Keys

To remove properties from an object use the keyword **delete** followed by the object name dot key. Alternatively, bracket notation can be used as well.

```
delete course.room;
delete course['startTime'];
```

ES5 Object Iterating

A **`for...in`** loop iterates over the properties of an object in an arbitrary order. Because order is not guaranteed, it is best not to add, modify, or remove properties from the object during iteration. Finally, the `for...in` loop is slower due to checks it must make on the Prototype chain.

Additionally, `for...in` loops should ***not*** be used with Arrays anytime index order is important.

```
const course = {
    department: 'Computer Science',
    number: 4220
};

for (const key in course) {
    console.log(key);
    console.log(course[key]);
}
```

ES6 Object Iterating

The **`Object.keys()`** method gets the keys of an object and returns them as an array.

Because we have the object's keys we can therefor loop over the array of keys to access the object's value for each given key in the object.

```
const course = {
   department: 'Computer Science',
   number: 4220
};

const courseKeys = Object.keys(course);  // ['department', 'number']

for (let i = 0; i < courseKeys.length; i++) {
   const key = courseKeys[i];
   const value = course[key];
   console.log(key, value);
}
```

Object Assign

The `Object.assign(target, source)` method is used to copy the values and properties from one or more source objects to a target object. It then returns the target object which has properties and values copied from the source object.

Target is the object to which values and properties will be copied into.

Source(s) is the object or objects from which values and properties are copied from.

```
const course = {
   department: 'Computer Science',
   number: 5220
};
Object.assign(course, { number: 4220, year: 2023 });
console.log(course);
// { department: 'Computer Science', number: 4220, year: 2023 }
```

Objects Comparison

In JavaScript when comparing objects - this is done by **reference to the object itself** and not the keys/values inside of it. Therefore we **cannot** use standard equality (== or ===) to compare two objects.

```
const obj1 = { a: 1 };
const obj2 = { a: 1 };

console.log(obj1 === obj1);  // true
console.log(obj1 === obj2);  // false
```

In order to compare objects for equality there are a few approaches - shallow equality and deep equality.

Currently, in **JavaScript there is no built in method** to perform shallow or deep equality. To perform either check we must write a function or use a library which has already written the appropriate function.

Shallow Equality

Shallow equality will look at both objects to see if they have the same keys and values. This is typically done using the `Object.keys()` method. However there are limitations, objects in JavaScript can be nested and when this happens shallow equality no longer works as expected.

```
const shallowComparison = (obj1, obj2) => {
    const objKeys1 = Object.keys(obj1);
    const objKeys2 = Object.keys(obj2);

    if (objKeys1.length !== objKeys2.length) {
        return false;
    }

    for (let i = 0; i < objKeys1.length; i++) {
        const key = objKeys1[i];

        if (obj1[key] !== obj2[key]) {
            return false;
        }
    }

    return true;
};
```

Deep Equality

Deep equality solves this limitation by adding an additional check to see if the compared values are objects. If so, then a recursive equality check is performed on these nested objects.

```
const deepComparison = (obj1, obj2) => {
    const objKeys1 = Object.keys(obj1);
    const objKeys2 = Object.keys(obj2);

    if (objKeys1.length !== objKeys2.length) {
        return false;
    }

    for (let i = 0; i < objKeys1.length; i++) {
        const key = objKeys1[i];

        const objValue1 = obj1[key];
        const objValue2 = obj2[key];

        if (typeof objValue1 === 'object' && typeof objValue2 === 'object') {
            const isEqual = deepComparison(objValue1, objValue2);
            if (!isEqual) {
                return false;
            }
        } else if (objValue1 !== objValue2) {
            return false;
        }
    }

    return true;
};
```

Basic Object Methods

JavaScript provides many methods on objects. A comprehensive list can be found at:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object

`Object.assign(`*`target, source`*`)`
copies all properties from one or more source objects to a target object. returned the mutated target object.

`Object.keys(`*`obj`*`)`
returns an array of a given object's keys

`Object.values(`*`obj`*`)`
returns an array of a given object's values

`Object.entries(`*`obj`*`)`
returns an array of a given object's key values pairs as an array `[key, value]`

Object Destructuring

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to extract data from objects into distinct variables. By using destructuring we can unpack the key and assign its value in a single line. A **default value** can be assigned in the case that the key unpacked from the object is not defined.

```
const transformer = {
   name: 'Optimus Prime',
   team: 'Autobots',
   colors: ['red', 'blue', 'silver']
};

const { name, team, colors, type = 'truck' } = transformer;
```

Spread Operator vs Rest Parameter

JavaScript uses three dots (`...`) for both the spread operators and rest parameter. However, these two are not the same.

The **rest parameter** puts the rest of some specified values into a JavaScript array.

```
// rest
function loop(...args) {
    for (let i = 0; i < args.length; i++) {
        console.log(args[i]);
    }
}
loop('hello', 'world', 1);
```

The **spread operator** expands iterables (arrays, strings) into individual elements.

```
// spread
const start = ['a', 'b', 'c'];
const end = ['x', 'y', 'z'];
const alpha = [...start, ...end];
```

Object: ... (spread)

Previously we saw the **`Object.assign()`** method. This is used to merge two objects. And this method copies properties from one or more source objects to a target object.

Spread syntax **`...`** also works with objects and is typically used to make copies and allows for adding properties. The **spread syntax** does not mutate the original object.

```
const transformer = {
  name: 'Optimus Prime',
  team: 'Autobots',
  colors: ['red', 'blue', 'silver']
};

const updateTransformer = { ...transformer, homeWorld: 'Cybertrom' };
```

Array: ... (spread)

Previously we saw the Array `concat()` method. This is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

Spread syntax `...` allows an iterable such as an array to be expanded. And this syntax can be used to concatenate arrays and return a new array. This Spread operator is unpacking collected elements like arrays into a single element.

```
const a1 = [1, 2, 3];
const a2 = [3, 4, 5];
const concat = a1.concat(a2);
console.log(concat);  // [1, 2, 3, 3, 4, 5]

const spread = [...a1, ...a2];
console.log(spread);  // [1, 2, 3, 3, 4, 5]
```

Array: Destructuring and ... (rest)

The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

The rest syntax for arrays `...` allows for collecting all of the remaining elements (meaning - “the rest of the elements”) into an array.

```
const arr = [10, 20, 30, 40, 50];
const [first, second, ...restof] = arr;


console.log(first);    // 10
console.log(second);   // 20
console.log(restof);   // [30, 40, 50]
```

Resources

ES6 Features

- https://github.com/lukehoban/es6features

Modern JavaScript Tutorial

- https://javascript.info/object
- https://javascript.info/object-copy
- https://javascript.info/rest-parameters-spread

Mozilla Introduction to JavaScript

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters

Review and Prep

Review

- Review Slides
- Review and Run Code Examples

Homework 1

- Open on Canvas - Wednesday 02/08
- Due on Canvas - Wednesday 02/15 at 11:59pm PST
- All topics from 1/25 to 2/08 Lectures
- 2x Lab Time to work on Homework 1

Preparation for Next Class

- Read Eloquent Javascript - Chapters 5
- Work on Homework 1

```
console.log('Week 03');
console.log('Code Examples');
```