W03 Readings

Object Methods. This.

Use ‘this’ in functions when doing something with a variable that allows you to be more modular.

Functions that are stored in object properties are called “methods”.
Methods allow objects to “act” like object.doSomething().
Methods can reference the object as this.

The value of this is defined at run-time.

When a function is declared, it may use this, but that this has no value until the function is called.
A function can be copied between objects.
When a function is called in the “method” syntax: object.method(), the value of this during the call is object.

Please note that arrow functions are special: they have no this. When this is accessed inside an arrow function, it is taken from outside.

Really cool example.

There’s a `ladder` object that allows to go up and down:

```
let ladder = {
  step: 0,
  up() {
    this.step++;
  },
  down() {
    this.step--;
  },
  showStep: function() { // shows the current step
    alert( this.step );
  }
};
```

Now, if we need to make several calls in sequence, can do it like this:

```
ladder.up();
ladder.up();
ladder.down();
ladder.showStep(); // 1
ladder.down();
ladder.showStep(); // 0
```

This in JavaScript reading:

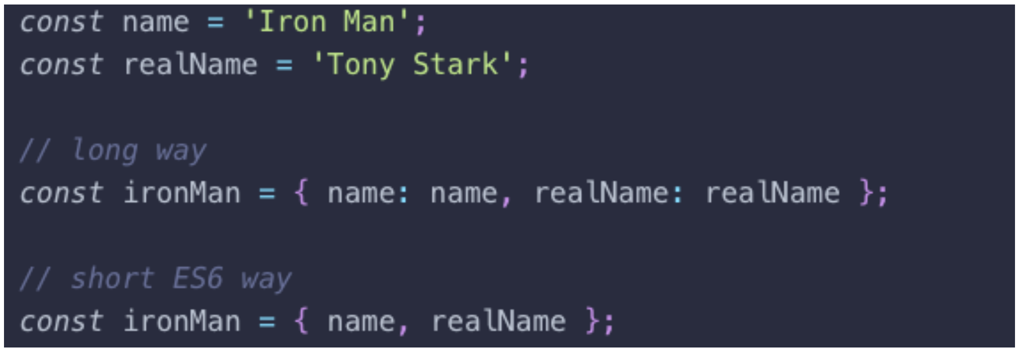
this in global context
this in object construction
this in an object method
this in a simple function
this in an arrow function
this in an event listener

Ch 5 Objects reading:

The appropriate way of creating objects:

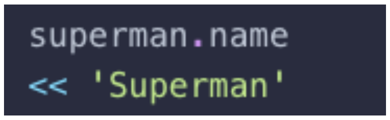
```
const spiderman = {};
```

Naming object properties:



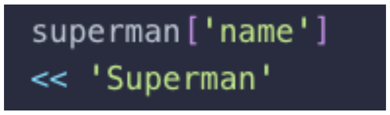
```
const name = 'Iron Man';
const realName = 'Tony Stark';

// long way
const ironMan = { name: name, realName: realName };

// short ES6 way
const ironMan = { name, realName };
```

Accessing properties



```
superman.name
<< 'Superman'
```

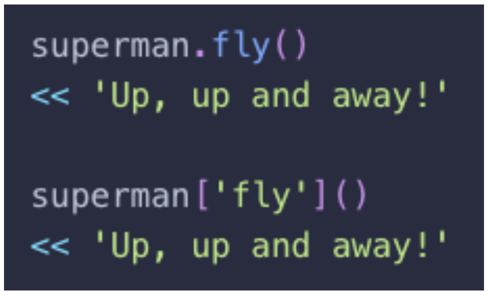
Also, can be accessed using brackets



```
superman['name']
<< 'Superman'
```

If you get undefined while trying to access a property then it doesn't exist. Check the spelling or other potential errors that would be preventing you from retrieving it.

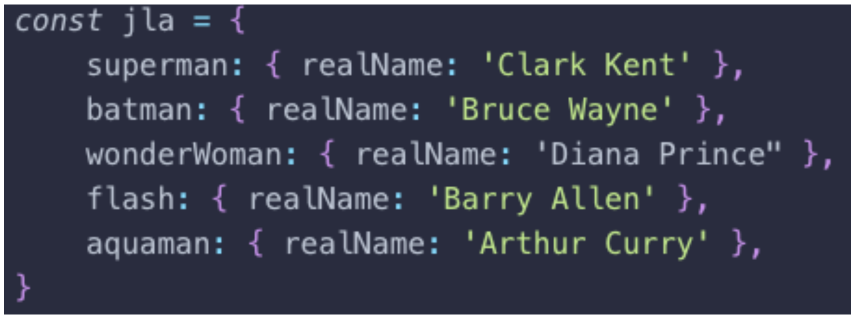
Different ways to call methods dot method or bracket method.



```
superman.fly()
<< 'Up, up and away!'

superman['fly']()
<< 'Up, up and away!'
```

Objects Are Copied by Reference
An important concept to get your head around is that objects are assigned by reference. This means that if a variable is assigned to an object that already exists, it will simply point to the exact same space in memory. So, any changes made using either reference will affect the same object.

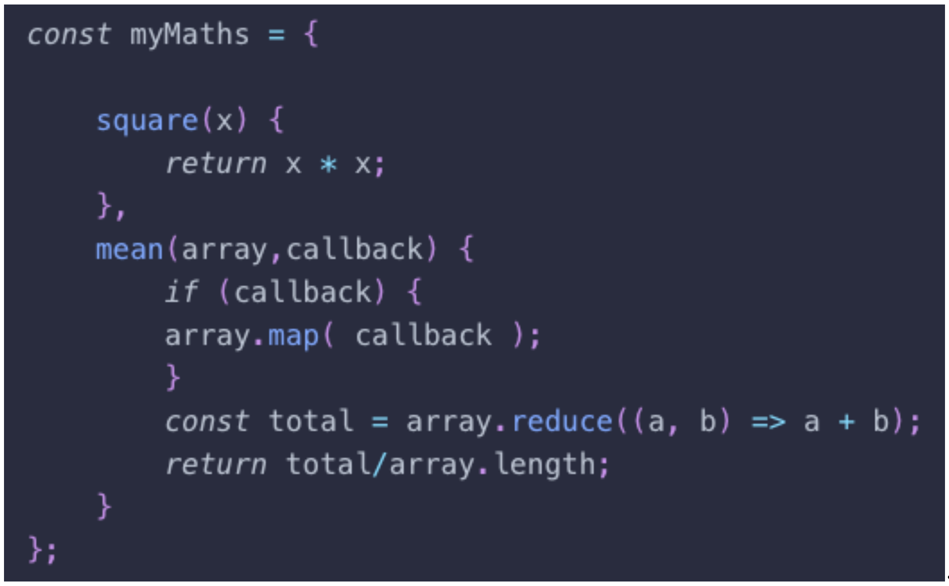
We can nest objects.



```
const jla = {
    superman: { realName: 'Clark Kent' },
    batman: { realName: 'Bruce Wayne' },
    wonderWoman: { realName: 'Diana Prince" },
    flash: { realName: 'Barry Allen' },
    aquaman: { realName: 'Arthur Curry' },
}
```

Namespacing:

Namespacing is just creating an object to carry functions just in case you start getting name collisions as others work on the project you could be colliding with other names that others are trying to name methods.



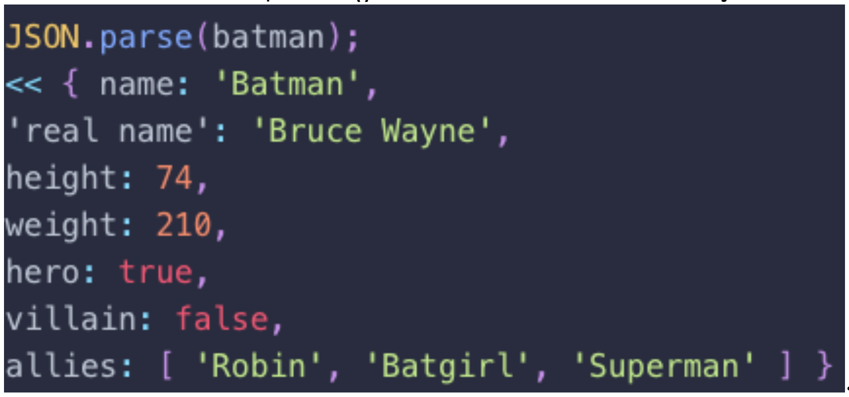
```
const myMaths = {

    square(x) {
        return x * x;
    },
    mean(array,callback) {
        if (callback) {
        array.map( callback );
        }
        const total = array.reduce((a, b) => a + b);
        return total/array.length;
    }
};
```

.

Big differences between JSON and JS objects.

- Functions are not permitted
- Permitted values are double-quoted strings, numbers, true, false, null, arrays, and objects
- Property names must be double quoted.

You can use the parse() method to a JSON object and return a JS object.



```
JSON.parse(batman);
<< { name: 'Batman',
'real name': 'Bruce Wayne',
height: 74,
weight: 210,
hero: true,
villain: false,
allies: [ 'Robin', 'Batgirl', 'Superman' ] }
```

.

Stringify() method does the opposite
Math object is awesome.

Math.PI // The ratio of the circumference and diameter of a circle
<< 3.141592653589793

Math.SQRT2 // The square root of 2
<< 1.4142135623730951

Math.SQRT1_2 // The reciprocal of the square root of 2
<< 0.7071067811865476

Math.E // Euler's constant
<< 2.718281828459045

Math.LN2 // The natural logarithm of 2

```
<< 0.6931471805599453

Math.LN10 // The natural logarithm of 10
<< 2.302585092994046

Math.LOG2E // Log base 2 of Euler's constant
<< 1.4426950408889634

Math.LOG10E // Log base 10 of Euler's constant
<< 0.4342944819032518
```

Chapter Summary

Objects are a collection of key-value pairs placed inside curly braces {}.

Objects have properties that can be any JavaScript value. If it's a function, it's known as a method.

An object's properties and methods can be accessed using either dot notation or square bracket notation.

Objects are mutable, which means their properties and methods can be changed or removed.

Objects can be used as parameters to functions, which allows arguments to be entered in any order, or omitted.

Nested objects can be created by placing objects inside objects.

JSON is a portable data format that uses JavaScript object literals to exchange information.

The Math object gives access to several mathematical constants.

The Math object can be used to perform mathematical calculations.

The Date object can be used to create date objects.

Once you've created a Date object, you can use the getter methods to access information about that date.

Once you've created a Date object, setter methods can be used to change information about that date.

The Regex object can be used to create regular expressions.

Chapter 6 DOM

It is a way we can interact with the webpage.

Chapter Summary

The Document Object Model is a way of representing a page of HTML as a tree of nodes.

The document.getElementById(), document.getElementsByClassName(), document.getElementsByTagNames() and document.querySelector() can be used to access elements on a page.

The parentNode(), previousSibling(), nextSibling(), childNodes() and children() methods can be used to navigate around the DOM tree.

An element's attributes can be accessed using the getAttribute() method and updated using the setAttribute() method.

The createElement() and createTextNode() methods can be used to create dynamic markup on the fly.

Markup can be added to the page using the appendChild() and insertBefore() methods.

Elements can be replaced using the replaceChild() method and removed using the removeChild() method.

The innerHTML property can be used to insert raw HTML directly into the DOM.

The CSS properties of an element can be changed by accessing the style property.

Chapter 7 Events:

Chapter Summary

Events occur when a user interacts with a web page.

An event listener is attached to an element, then invokes a callback function when the event occurs.

The event object is passed to the callback function as an argument and contains lots of properties and methods relating to the event.

There are many types of events, including mouse events, keyboard events, and touch events.

You can remove an event using the removeEventListener method.

The default behavior of elements can be prevented using the preventDefault() function.

Event propagation is the order in which the events fire on each element.

Event delegation is when an event listener is added to a parent element to capture events that happen to its children elements.