JavaScript-Part 3

Almost done – welcome to week 7!
When you load a page into the browser, the browser parses the HTML and creates an internal model of your document, that contains all the elements of your HTML markup.

Your JavaScript can interact with the DOM to get access to the elements and the content in them. JavaScript can also use the DOM to create or remove elements.

When JavaScript modifies the DOM, the browser updates the page dynamically, so you see new content on your page.
1. Start by creating a document node at the top.

```
<doctype html>
<html lang="en">
<head>
  <title>My blog</title>
  <meta charset="utf-8">
  <script src="blog.js"></script>
</head>
<body>
  <h1>My blog</h1>
  <div id="entry1">
    <h2>Great day bird watching</h2>
    <p>Today I saw three ducks! I named them Huey, Louie, and Dewey.</p>
    <p>I took a couple of photos...</p>
  </div>
</body>
</html>
```
document is always at the top. document is a special part of the tree that you can use in JavaScript to get access to the entire DOM.

document is also like the root of an upside down tree.

These are like the branches of the tree.

These are like the leaves of the tree (because there are no elements inside them, just text).

The DOM includes the content of the page as well as the elements. (We don’t always show all the text content when we draw the DOM, but it’s there).

We compare this structure to a tree because a “tree” is a data structure that comes from computer science.
DRAW THE DOM FOR THIS

```
<html lang="en">
  <head>
    <title>Movies</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Movie Showtimes</h1>
    <h2 id="movie1">Plan 9 from Outer Space</h2>
    <p>
      Playing at 3:00pm, 7:00pm.
      <span>
        Special showing tonight at <em>midnight</em>!
      </span>
    </p>
    <h2 id="movie2">Forbidden Planet</h2>
    <p>
      Playing at 5:00pm, 9:00pm.
    </p>
  </body>
</html>
```
JS & HTML

→ communicate via DOM

JS can get access to DOM via: `getElementById`

`getElementById`

Cool! Wow!

LOVE IT!
```html
<doctyple html>
<html lang="en">
<head>
<title>Planets</title>
<meta charset="utf-8">
<script>
window.onload = init;
function init() {
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red Alert: hit by phaser fire!";
}
</script>
</head>
<body>
<h1>Green Planet</h1>
<p id="greenplanet">All is well</p>
<h1>Red Planet</h1>
<p id="redplanet">Nothing to report</p>
<h1>Blue Planet</h1>
<p id="blueplanet">All systems A-OK</p>
</body>
</html>
```
Here's a DOM with a secret message hidden in it. Evaluate the code below to reveal the secret!

```
document.getElementById("e7")
document.getElementById("e8")
document.getElementById("e16")
document.getElementById("e9")
document.getElementById("e18")
document.getElementById("e13")
document.getElementById("e12")
document.getElementById("e2")
```

Write the element each line of code selects, as well as the content of the element to reveal the secret message!
Green Planet
All is well

Red Planet
Nothing to report

Blue Planet
All systems A-OK
getElementById gets access to an element – use `innerHTML` to change the content of that element.
We're assigning the element to a variable named planet.

```javascript
var planet = document.getElementById("greenplanet");
```

Here's our call to `getElementById`, which seeks out the "greenplanet" element and returns it.

And in our code we can now just use the variable `planet` to refer to our element.

```javascript
planet.innerHTML = "Red Alert: hit by phaser fire!";
```

We can use the `innerHTML` property of our planet element to change the content of the element.

We change the content of the `greenplanet` element to our new text... which results in the DOM (and your page) being updated with the new text.
```javascript
var planet = document.getElementById("greenplanet");

planet.innerHTML = "Red Alert: hit by phaser fire!";
```

We change the content of the `greenplanet` element to our new text... which results in the DOM (and your page) being updated with the new text.

Any changes to the DOM are reflected in the browser's rendering of the page, so you'll see the paragraph change to contain the new content!
Your DOM is created when the page is fully loaded.
If the JS is executed before the DOM is created, then it cannot change anything.
Need to tell the browser:
“run my code after you’ve fully loaded in the page and created the DOM”
First, create a function named `init` and put your existing code in the function.

Notice that your code goes between an opening `{` and a closing `}`.

Here, we’re setting the value of the window.onload property to the function name.

This says when the page is fully loaded, execute the code that is in `init`. 
Objects—a collection of properties

- All dogs have a list of activities they are doing
  - Walking
  - Fetching a ball
  - Sleeping

- Have a weight
- Cutest dog object (ever!)

- "Fido" have a name
object representing a dog

name: "Fido"

weight: 40

breed: "Mixed"

loves: ["walks", "fetching balls"]
We assign the object to the variable fido

```javascript
var fido = {
  name: "Fido",
  weight: 40,
  breed: "mixed",
  loves: ["walks", "fetching balls"]
};
```
Access object properties with “dot” notation:

```javascript
if (fido.weight > 25) {
    alert("WOOF");
} else {
    alert("yip");
}
```

- Use the object `fido.weight`
- Use the object along with a “.” and a property name to access the value of that property.
Access properties using a string with [] notation:

```javascript
var breed = fido['breed'];
if (breed == "mixed") {
    alert("Best in show");
}
```

Use the object along with the property value wrapped in quotes and in brackets to access the value of that property.

Use [ ] around the property name.
3. Change a property's value:

```javascript
fido.weight = 27;
fido.breed = "Chawalla/Great Dane mix";
fido.loves.push("chewing bones");
```

- Changing Fido's weight
- Changing his breed...

Adding a new item to loves to do...

push simply adds a new item to the end of an array.
this means to go through all of the properties of the object.

4. Enumerate all an object’s properties:

```javascript
var prop;
for (prop in fido) {
    alert("Fido has a " + prop + " property ");
    if (prop == "name") {
        alert("This is " + fido[prop]);
    }
}
```

Note: the order of the properties is arbitrary.

We are using the [ ] notation to access the value of the property.

Each time through the loop, the variable prop gets the string value of the next property value.

can use a for-loop to enumerate the properties.
Have fun with an object’s array:

```javascript
var likes = fido.loves;
var likesString = "Fido likes";

for (var i = 0; i < likes.length; i++) {
    likesString += " " + likes[i];
}
alert(likesString);
```

Here we are assigning the value of fido’s loves array to the variable likes.

We can loop through the likes array and create a likesString of all fido’s interest.

And we can alert the string.
Pass an object to a function:

```javascript
function bark(dog) {
    if (dog.weight > 25) {
        alert("WOOF");
    } else {
        alert("yip");
    }
}

bark(fido);
```

We can pass an object to a function just like any other variable.

In the function, we can access the object's properties like normal, using the parameter name for the object.

We are passing fido as our argument to the function bark, which expects a dog object.
The Dot Operator .

The dot operator (.) gives you access to an object’s properties. In general it’s easier to read than the ["string"] notation:

- `fido.weight` is the size of fido.
- `fido.breed` is the breed of fido.
- `fido.name` is the name of fido.
- `fido.interests` is an array containing fido’s interests.
Can we add properties to objects after we've defined them?

- To add a property:
  ```javascript
  fido.age = 5;
  ```
  - Simply assign a new value.

- To delete a property:
  ```javascript
  delete fido.age;
  ```
  - Use delete keyword.
When an object is assigned to a variable, the variable is given a reference to the object. It doesn’t hold the object itself.

When we call `bark` and pass `fido` as an argument, we get a copy of the reference to the dog object.
1. We've defined an object, `fido`, and we are passing that object into a function, `loseWeight`.

```javascript

fido.weight = 48;

loseWeight(fido);
```

`fido` is a reference to an object, which means the object does not live in `fido` variable, but is pointed to by the `fido` variable.

When we pass `fido` to a fn, we are passing the reference to the object.
2. The `dog` parameter of the `loseWeight` function gets a copy of the reference to `fido`. So, any changes to the properties of the parameter affect the object that was passed in.

```javascript
function loseWeight(dog) {
  dog.weight = dog.weight - 10;
}

alert (fido.name + " now weights " + fido.weight);
```

When we pass `fido` into `loseWeight`, what gets assigned to the `dog` parameter is a copy of the reference.

So, `fido` and `dog` point to the same object.

```
function loseWeight(dog) {
  dog.weight = dog.weight - 10;
}
```

When we subtract 10 lbs from `dog.weight`, we are changing the value of `fido.weight`. This affects `fido` because they point to the same object.
Recall Excel

2D

A   B
1   2
2   3
3   4

A1 A2 A3 A4

49 36 24 70 56

ARRAY
temp[0] = 39;

int i = 0;

for (i; i < 20; i++) {
    alert(temp[i]);
    temp[i] += 2;
}
**type: ARRAY**

- Holds a collection of values
  - Each value has an index number that starts with 0!

Here's the variable for the array:

```javascript
var tempByHour = new Array();
tempByHour[0] = 59.2;
tempByHour[1] = 60.1;
tempByHour[2] = 63;
tempByHour[3] = 65;
tempByHour[4] = 62;
```

You can add more values to the array as needed:

For every index in the array, there is a corresponding value.

To add new values to the array, we just reference the index number of the array item, and give it a value.
```javascript
var tempByHour = new Array();
tempByHour[0] = 59.2;
tempByHour[1] = 60.1;
tempByHour[2] = 63;
tempByHour[3] = 65;
tempByHour[4] = 62;
```

is the same as

```javascript
var tempByHour = [59.2, 60.1, 63, 65, 62];
```
var tempByHour = [59.2, 60.1, 63, 65, 62];

tempByHour[5] = 61;

By using a new index, we get a new item in the array.

var message = "The temperature at 5 was " + tempByHour[5];
alert(message);

to access the value of the temperature at index 5, we just reference the array at index 5.

var numItems = tempByHour.length;

Ya gotta know the size of your array!
var x = funcName(arg1, arg2, ... argN);

function funcName(param1, param2, ... paramN) {
    statements;
    ... return result;
}
button when selected operator

```javascript
onclick doIt("\*")
```

```javascript
oncuck doIt("/")
```
```javascript
var a = 10;
a + 2 = 12

function abc() {
    var a = 20;
    alert(a);
    alert(b);
}

abc;
```