JavaScript Overview

Lecture 2

ALEX HINDS | FRONTEND DEVELOPER | @AL_HINDS

Course Overview

Intro to the Web

Language Overview

Introduction to the DOM

DOM API and Events

Asynchronous JavaScript

Extended JavaScript

To recap

Modern Web

Three core building blocks to the modern web, HTML, CSS, and JavaScript

HTML and CSS

HTML and CSS provide a scaffold and help guide the layout and style of our content, but have limitations.

JavaScript

Complements HTML and CSS to drive dynamic web pages. We're yet to see how but we'll get there.

To recap

Modern Web

Three core building blocks to the modern web, HTML, CSS, and JavaScript

HTML and CSS

HTML and CSS provide a scaffold and help guide the layout and style of our content, but have limitations.

JavaScript

Complements HTML and CSS to drive dynamic web pages. We're yet to see how but we'll get there.

To recap

Modern Web

Three core building blocks to the modern web, HTML, CSS, and JavaScript

HTML and CSS

HTML and CSS provide a scaffold and help guide the layout and style of our content, but have limitations.

... JavaScript

Complements HTML and CSS to drive dynamic web pages. We're yet to see how but we'll get there.

Intro to JavaScript

Runtime, engine, and the event loop

JavaScript Introductions

Runtime

Can and Can't

Script Loading

DevTools

Plain text

Scripts are provided and executed as plain text. They don't need special preparation or compilation to run.

Engine

Scripts then rely on an 'engine' to compile them into byte code. Engines are environment dependent. Mozilla (SpiderMonkey) and Chrome (V8) have different engines for example.

Single threaded

JavaScript relies on a concept of run-to-completion. Tasks are executed synchronously, and queued if multiple tasks need to run at the same time.

JavaScript Introductions

Runtime

Can and Can’t

Script Loading

DevTools

JS in the browser

Javascript is only as powerful as its environment. In the browser its focus is UI manipulation.

It can..

Manipulate CSS, HTML and the web document directly and efficiently; react to user interaction; make web requests; store data in the browser; mine bitcoin.

It can’t..

Read or write from your operating system without explicit permission (eg. file upload). Arbitrarily request data from any site – permissions required.

JavaScript Introductions

Runtime

Can and Can't

Script Loading

DevTools

Loading a script

Scripts are included on an HTML page in three main ways.

Inline, from an external source, or from a module.

```
<!DOCTYPE html>
<html lang="en">
<body>
    <script>
      console.log('Hello Andrew!');
    </script>
</body>
</html>
```

JavaScript Introductions

Runtime

Can and Can't

Script Loading

DevTools

Loading a script

Scripts are included on an HTML page in three main ways.

Inline, from an external source, or from a module.

```
<!DOCTYPE html>
<html lang="en">
<body>
    <script src="script.js"></script>
</body>
</html>
```

```
// inside our script.js
console.log('Hello Andrew!');
```

JavaScript Introductions

Runtime

Can and Can't

Script Loading

DevTools

Loading a script

Scripts are included on an HTML page in three main ways.

Inline, from an external source, or from a module.

```
<!DOCTYPE html>
<html lang="en">
<body>
    <!— We'll see how this works a bit later —>
    <script type="module" src="script.js"></script>
</body>
</html>
```

JavaScript Introductions

Runtime

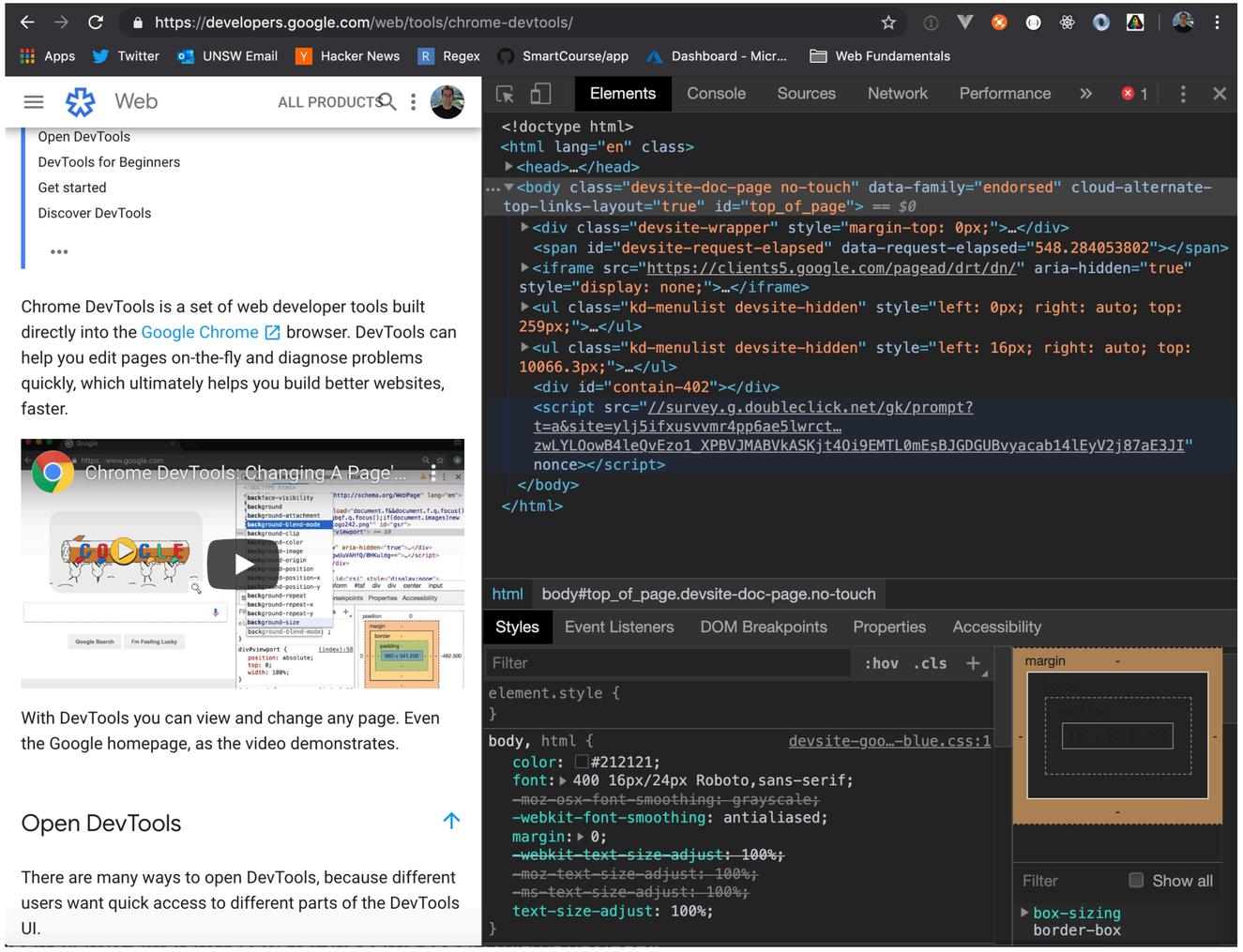
Can and Can't

Script Loading

DevTools

DevTools Demo

DevTools are integral to the WebDev experience.



Language and Syntax

Littered with lots of demos

Basics

console

prompt, alert

comments

semicolons

```
// functionally similar
console.log('Hello World');
console.warn('Some warning');
console.error('Some bad thing');
```

Basics

console

prompt, alert

comments

semicolons

```
// outputs a prompt to the browser window
const message = prompt('Can I have a sheep?');

// outputs a dialog message to the browser window
alert('alert!: ', message);

// returns a boolean
const yesOrNo = confirm('Do you want more soup?');
```

Basics

console

prompt, alert

comments

semicolons

```
/**
 * Multiline comments
 */

// single line comments

// c-like
```

Basics

console

prompt, alert

comments

semicolons

```
// functionally similar
console.log('Hello World');
console.warn('Some warning');
console.error('Some bad thing');

// outputs a prompt to the browser window
const message = prompt('Can I have a sheep?');

// outputs a dialog message to the browser window
alert('alert!: ', message);

// returns a boolean
const yesOrNo = confirm('Do you want more soup?');
```

Variables

Declaring variables

var, const, let... and an implicit fourth global.

‘let’

Can be re-assigned and declared ahead of time. Common way to declare a variable

‘const'

If a variable is declared with a const prefix it cannot be re-assigned. Its value cannot be directly mutated, but if it points to a reference, the underlying object still can. Important to note.

Variables

Declaring variables

var, const, let... and an implicit fourth global.

‘let’

Can be re-assigned and declared ahead of time.
Common way to declare a variable

‘const'

If a variable is declared with a const prefix it cannot be re-assigned. Its value cannot be directly mutated, but if it points to a reference, the underlying object still can. Important to note.

Variables

Declaring variables

var, const, let... and an implicit fourth global.

‘let’

Can be re-assigned and declared ahead of time.
Common way to declare a variable

‘const'

If a variable is declared with a const prefix it cannot be re-assigned. Its value cannot be directly mutated, but if it points to a reference, the underlying object still can. Important to note.

Primitive Types

string

number

boolean

undefined/null

```
// strings can be declared like this
const aString = 'hello world'

// or like this
const anotherThing = "Hello World"

// or like this
const someAnotherThing = `hello world`
```

Primitive Types

string

number

boolean

undefined/null

```
// strings can be concatenated
// with the + operator (like python)
const firstName = ‘Alex',
      lastName  = 'Hinds'

const fullName  = 'Alex' + ' ' + 'Hinds' // Alex Hinds
// OR
const fullNameWithTemplate = `${firstName} ${lastName}`;

// strings also have a number of methods natively
available
anotherThing.toUpperCase() // HELLO WORLD
anotherThing.replace(/[aeiouAEIOU]/, ''); // Hll Wrld
```

Primitive Types

string

number

boolean

undefined/null

```
/** numbers */
const anInt = 1;

// floats
const aFloat = 1.2;

// exponents
const exp = 1e2;
```

Primitive Types

string

number

boolean

undefined/null

```
/** numbers */
const anInt = 123;

/** aFloat */
const aFloat = 1.2;

// exponents
const exp = 1e2;

const longFloat = 1.231215;

// Like stirngs numbers also have some handly
builtin
const shortFloat = longFloat.toFixed(2);

// toString translates to baseX representation
const binaryRepresentation  = anInt.toString(2);

// there are many more
```

Primitive Types

string

number

boolean

undefined/null

```
// standard declaration
const isTrue = true;
const isFalse = false;

// not operator to coerce a boolean
const isTrueAlso = !false;
const isFalseAlso = !true;

// most types can be coerced with a double !!
const emptyString = ''
const falseValue = !!'';
```

Primitive Types

string

number

boolean

undefined/null

```
const noValue; // undefined

// rarely something you do
const noValue = undefined;

// different to undefined,
const definitivelySet = null;
```

Example

Functions in JavaScript

Functions

basic

arrows

```
function checkAge(age) {
  if (age > 18) {
    return true;
  } else {
    return confirm(‘Can you legally drink?’);
  }
}

let age = prompt('How old are you?', 18);

if (checkAge(age)) {
  alert( 'Access granted' );
} else {
  alert( 'Access denied' );
}
```

Functions

basic

arrows

```
/* think about what this functions do */
const multiply = a => b => a * b;
const pluck = key => object => object[key];

// let's say tax of 10% for GST and a 5 % first customer discount
const discount = multiply(0.95);
const tax = multiply(1.10);

// the format required for sum
const sum = (num, secondNum) => num + secondNum;
```

Example

Arrays in JavaScript

Arrays

basic

methods

Declaration

```
let arr = new Array();
let arr = [];
```

Access

```
let fruits = ["Apple", "Orange", "Plum"];

alert( fruits[0] ); // Apple
alert( fruits[1] ); // Orange
alert( fruits[2] ); // Plum
```

Arrays

basic

methods

There are two core ways to manipulate arrays in JavaScript

1. Impure manipulations (the underlying array is effected)
2.Pure manipulations (the method returns a new array)

There is no ‘right’ way to use these methods, but just be aware of the above when you make your manipulations.

Arrays

basic

methods

```
// impure manipulaitons
const a = [];
a.push(1) // a = [ 1 ]
a.unshift(2) // a

// pure manipulations
const b = []
const c = b.concat(1, 2, 3)
// c = [ 1, 2, 3 ] b unchanged
const d = c.map((value) => value * 2);
// d = [ 2, 4, 6]
```

Example

Objects in JavaScript

Objects

basic

prototypes

classes

```
// note const limitations apply only to reassigning
// the Object. Object properties can still be altered
// with a const declaration.
const myObject = {};

// or like this
const myOtherObject = new Object();

const myUser = {
  id: 'UAAAAAAA',
  displayName: 'Jane Citizen',
  age: 25,
};

// You can also assign & read properties:
console.log(myUser.age);
// > 25
console.log(myUser['age']);
// > 25

// setting attributes
myUser.age = 29;
myUser.address = '123 Fake Street';
```

Objects

basic

prototypes

classes

```
// note the use of this in this special constructor
// also note the caps (a convention for constructor functions)
function Person(firstName, lastName, age) {
  this.firstName = firstName;
  this.lastName  = lastName;
  this.age       = age;
}

Person.prototype.getFullName = function() {
  return `${this.firstName} ${this.lastName}`;
};

Person.prototype.canDrinkAlcohol = function() {
  return this.age >= 18;
};

// now if we call the constructor function we get this
new Person('Jeff', 'Goldblum', 50);
// => Person { firstName: 'Jeff', lastName: 'Goldblum', age: 50 }
```

Objects

basic

prototypes

classes

```
class Person {
  constructor(firstName, lastName, age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
  }

  getFullName() {
    return `${this.firstName} ${this.lastName}`;
  }

  canDrinkAlcohol() {
    return this.age >= 18;
  }
}

new Person('Jeff', 'Goldblum', 50);
```

Example

QUESTION

What is ‘this’ in JavaScript?

The object that ‘this’ refers changes every time execution context is changed. (huh?!)

https://medium.com/quick-code/understanding-the-this-keyword-in-javascript-cb76d4c7c5e8

Example

Control Flow & Loops

if/else

switch

loops

errors

```
if (condition) {
    // do something
}

if (condition) {
    // do something
} else {
    // do something
}

// as with c, one liners don't require
brackets. (don't do this though!)
if (condition)
    // do something
else
    // something else

// And of course ternary
const x = condition ? 22 : 0;
```

Control Flow & Loops

if/else

switch

loops

errors

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Control Flow & Loops

if/else

switch

loops

errors

```
// same as c
let index = 0;

while (index < array.length) {
    let value = array[index];
    index++;
}

do {
    let value = array[index];
    index++;
} while (index < array.length);


// Iteration over an array
for (let index = 0; index < array.length; index++) {
    // do something with item
    // very similar to c
    let value = array[index];
}
```

Control Flow & Loops

if/else

switch

loops

errors

```
// Iteration over an object/array
for (const property in items) {
    // do something with item
    // very similar to python
    let value = items[property];
}
```

```
let string1 = "";
const object1 = {a: 1, b: 2, c: 3};

for (let property1 in object1) {
  string1 += object1[property1];
}

console.log(string1);
// expected output: "123"
```

Control Flow & Loops

if/else

switch

loops

errors

```
let iterable = [10, 20, 30];

for (let value of iterable) {
  value += 1;
  console.log(value);
}
// 11
// 21
// 31


// OR like this.. other ways too
for (let char of "test") {
  // triggers 4 times: once for each character
  alert( char ); // t, then e, then s, then t
}
```

Control Flow & Loops

if/else

switch

loops

errors

I didn't cover this in the lecture, but try {} catch (e) {} is your go-to error handler. Note, unhandled errors in JS will cause your scripts to crash!

```
try {
  throw new Error('Whoops!');
} catch (e) {
  console.log(e.name + ': ' + e.message);
}
```

Example

import/ export

basic

import

export

```
//  lib/math.js
export function sum (x, y) { return x + y }
export var pi = 3.141593

//  someApp.js
import * as math from "lib/math"
console.log(“2*Pi = " + math.sum(math.pi, math.pi))

//  otherApp.js
import { sum, pi } from "lib/math"
console.log(“2*Pi = " + sum(pi, pi))
```

import/ export

basic

import

export

```
// relative path
import defaultExport from "module-name";

// alias the default export
import { default as alias } from "module-name";

// import all exports under the 'name' namespace
import * as name from "module-name";

// import namedExport
import { namedExport } from "module-name";

// alias using 'as'
import { namedExport as alias } from "module-name";

// combining a few imports from the same module
import defaultExport, { export1 , export2 } from "module-name";
```

import/ export

basic

import

export

```
// AND The export versions
export default Export;

// re-export a default import as an alias
export { default as alias } from "module-name";

// export all exports under the 'name' namespace
export * from "module-name";

// export namedExport
export { namedExport } from "module-name";

// alias using 'as' from an external module
export { namedExport as alias } from "module-name";

// combining a few exports from the same module
export { defaultExport as default, export1 , export2 };
```

Leveraging Built-in Libraries

Batteries included

You don’t need to reinvent the wheel. A lot of functionality comes for free in the libraries automatically exposed by the JS engine.

Less code to maintain

Libraries are maintained by professional developers working on browsers. They also don’t need to be downloaded by your users.

Leveraging Built-in Libraries

Batteries included

You don’t need to reinvent the wheel. A lot of functionality comes for free in the libraries automatically exposed by the JS engine.

Less code to maintain

Libraries are maintained by professional developers working on browsers. They also don’t need to be downloaded by your users.

Example

Thanks!

And good luck

ALEX HINDS | FRONTEND DEVELOPER | ATLASSIAN