~~Modern~~ Javascript

Some fundamentals and a bit of "modern"

Javascript

JavaScript (JS) is a lightweight, interpreted, programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as node.js and Apache CouchDB. JS is a prototype-based, multi-paradigm, dynamic scripting language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles. - MDN

Do not confuse JavaScript with the Java programming language. Both "Java" and "JavaScript" are trademarks or registered trademarks of Oracle in the U.S. and other countries. However, the two programming languages have very different syntax, semantics, and uses. - MDN

The menu

- Functions and Objects
- Some "new" JS stuff
- Basic DOM+JS
- Simple app (With some REST stuff, continuation in the group sessions)

NodeJS

- For the second part if you want to follow along :)

https://nodejs.org/en/download/current/

"Basic" Javascript

Functions

- All functions return something, a function without a return statement returns undefined. (Unless it's a constructor)
- Functions are objects, this means that a function can be stored in a variable, array, or object.
- Also, a function can be passed to, and returned from, a function. (setTimeout)

http://jsbin.com/zugimaleno/edit?js,console

Calling functions

- Four different ways to call functions
  - Method
  - Function
  - Constructor
  - Apply/Call
- Bind (You will probably use this with React)

http://jsbin.com/bucavuzihi/edit?js,console

Function scope

- Everything used to be function scope
- With the new `let` and `const` keywords that changed
- Can still be useful however, e.g. when not using modules

http://jsbin.com/rubemahiqi/edit?js,console

Objects

- Object oriented but class-less
- Objects are created without classes

```
const myObject = {
    name: 'Mark'
};
```

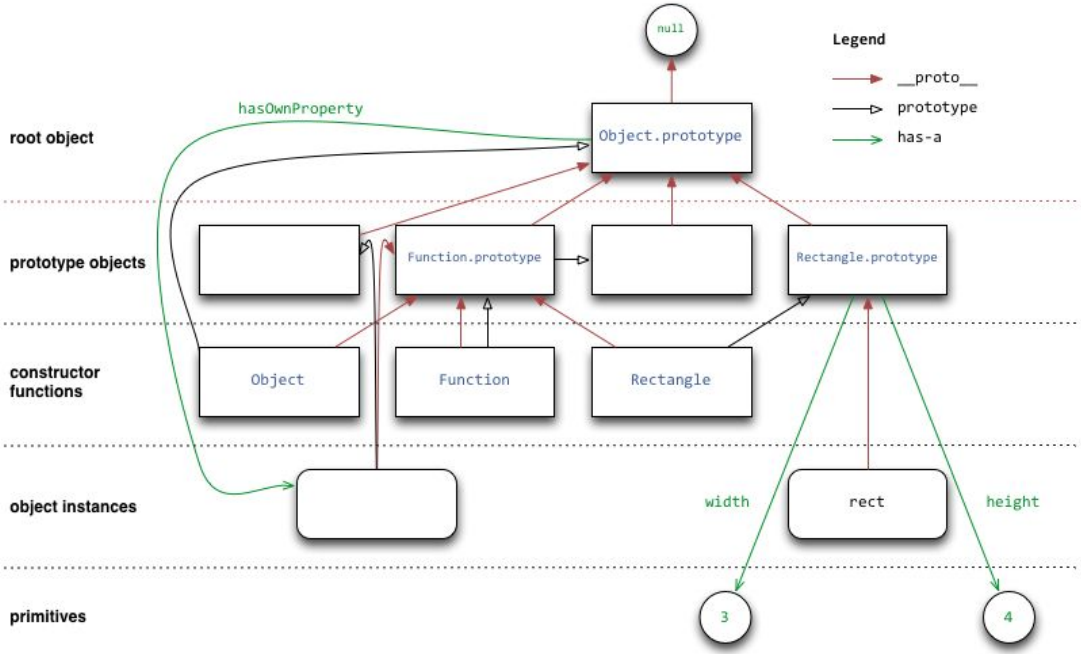
- "Inheritance" happens through the prototype chain.

Prototypal inheritance

- Objects linked together
- Chain of objects
- "Shadowing" of properties/methods

http://jsbin.com/gekinunolu/1/edit?js,console

Prototypal inheritance



"Modern" Javascript

Preparing for React/Angular

Classes (prototypes made easy?..)

```
class A {
     constructor() {
          this.name = 'The A class';
          this.course = '5750';
     }

     // methods
}

class B extends A {
     constructor() {
          super();
          this.name = 'The B class';
     }

     // additional methods
}
```

Rest and spread

```
function soRest(a,b, ...rest) {
        console.log(a,b, rest);
}

const args = ['a', 'b', 'c', 'd'];

soRest(...args); // 'a', 'b', ['c', 'd']
```

Template literals

- Templating made easy
- ${ } is used to inject stuff into the template string

```
const a = 1;
const b = 2;
console.log(`${a} + ${b} = ${a + b}`); // 1 + 2 = 3
```

Destructuring (Array)

- Assign values from an array to variables directly

const [a, b] = ['Hello', 2]; // a = 'Hello, b = '2'

Instead of:

const values = ['Hello', 2];
const a = values[0];
const b = values[1];

- Default values

Destructuring (Object)

- Assign values from an object to variables directly

const {a, b} = {a: 'Hello', b: 2, c: 'Something else' }; // a = 'Hello, b = '2'

Instead of:

const values = {a: 'Hello', b: 2, c: 'Something else' };
const a = values.a;
const b = values.b;

Destructuring (Object)

- More useful when using default values

```
const { name = 'John', age = 25 } = {};

const { name = 'John', age = 25 } = { name: 'Sven' };
```

Destructure function arguments

```
function doRequest({ url, method = 'GET', data }) {
	return request(url, method, data);
}

doRequest({
	url: 'http://google.com',
});
```

Promises

A promise to some value from the future... maybe. :)

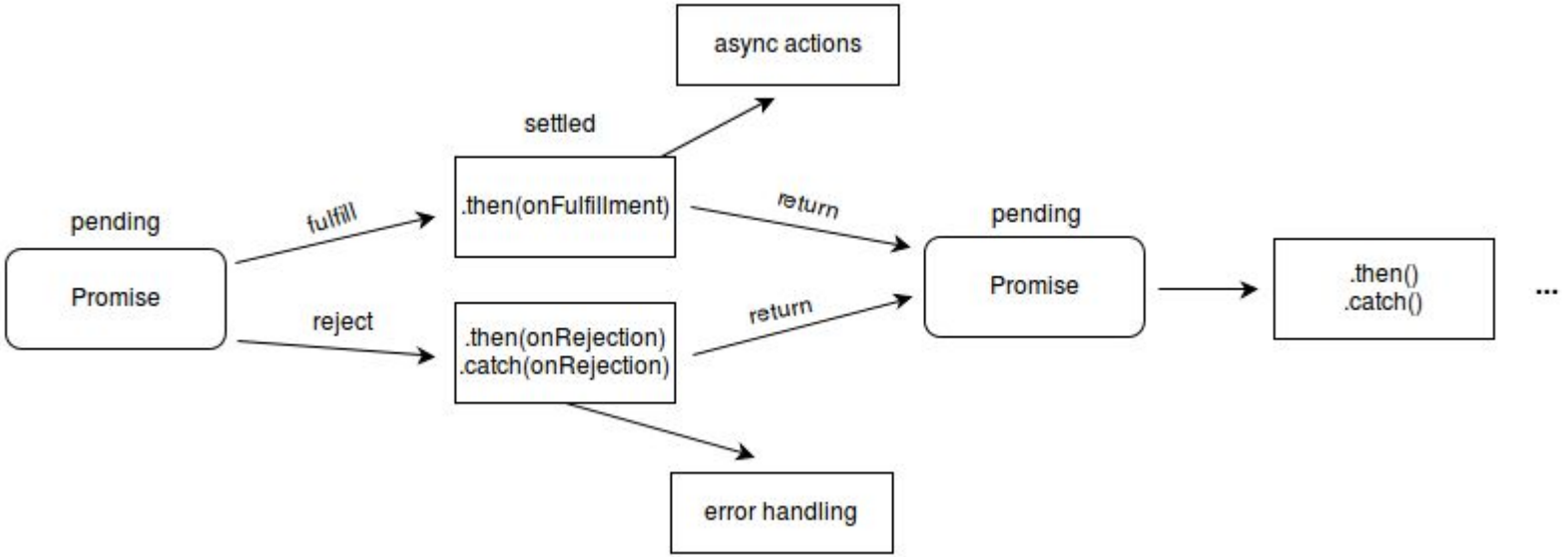
- Can be used for async things
  - Network requests
  - User interaction
  - Long running tasks
- Can help you make sense of callback hell

Promise methods

- Has three states "fulfilled/resolved", "rejected", "pending"
- Can only be fulfilled once
- resolve(value)
  - Create a 'resolved' promise from a value
- reject(value)
  - Create a 'rejected' promise from a value
- race(iterable)
- all(iterable)

Promise

- .then and .catch create new Promise objects
- Can be chained



More...

- Dynamic properties
- Map + Set + WeakMap + WeakSet
- Proxies
- Symbols
- Etc..

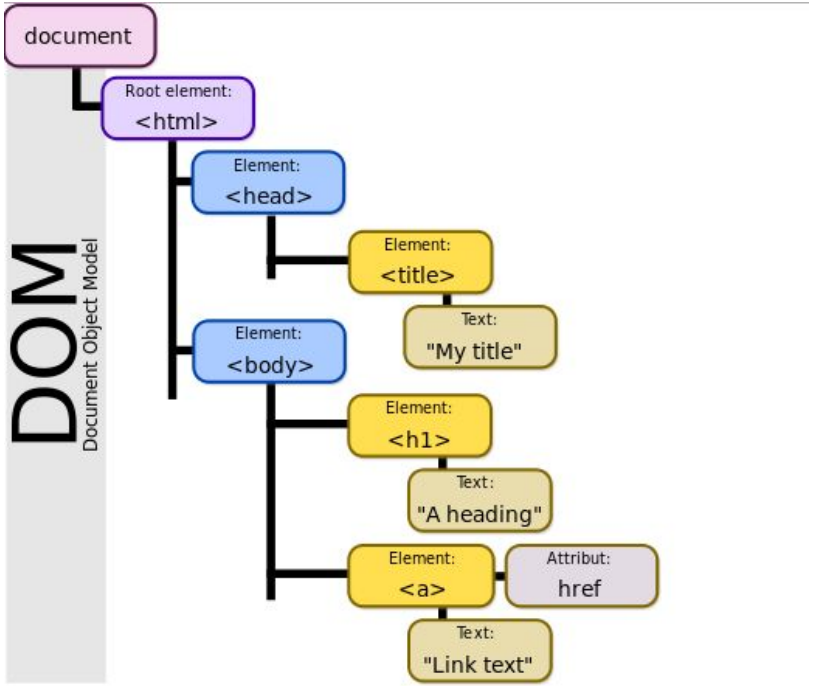
https://babeljs.io/docs/learn-es2015/

DOM

DOM

- "The **Document Object Model** (**DOM**) is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML **document** as a tree structure wherein each node is an **object** representing a part of the **document**." - *Wikipedia*
- The DOM is what you program against when doing web development
- When a web page is loaded, the browser creates a Document Object Model of the page.

DOM Tree


DOM
Document Object Model
document
Root element:
<html>
Element:
<head>
Element:
<title>
Text:
"My title"
Element:
<body>
Element:
<h1>
Text:
"A heading"
Element:
<a>
Attribut:
href
Text:
"Link text"

Working with the DOM

Using JavaScript we can:

- add, change, and remove all the HTML elements and attributes in the page.
- change all the CSS styles in the page.
- react to all existing events in the page.
- can create new events in the page.

Events

- addEventListener(handler: function)
- Handler will be called each time the event happens
- https://developer.mozilla.org/en-US/docs/Web/Events
- https://jsbin.com/?html,js,console

Coding an "app"

App

- Using webpack 2 to do bundling of the app and gives the ability to use JS modules.
- Implement some basic app that loads and creates DHIS2 Organisation Units

Organisation units in DHIS2

- Tree structure of nodes
  - Generally represent geographical locations, but could basically be any hierarchy.
- I only care about one level today ;)