Feature detection in Javascript

One thing that makes the web difficult to work with is that a given feature may or may not be available in all your target browsers either because of the version of EcmaScript (the Javascript standard) it was introduced in or because browsers haven't gotten around to implement them and may not do so at all or worse, they may have poor implementations of a given API (the initial implementation of Indexed DB in Safari comes to mind).

In the old days we would do browser detection and assume that if the browser was Netscape they would support a given set of features (and not others). This worked in some cases but in others (like IE 3.5) there were differences between operating systems. Code like the one below was fairly common

```
let ua = navigator.userAgent;

if(ua.indexOf('Firefox') !== -1) {
  console.log('running Firefox specific code');
} else if(ua.indexOf('Chrome') !== -1) {
  console.log('running Chrome specific code');
}
```

But this type of browser sniffing has significant drawbacks. The biggest one is that, as written, it's an all-or-nothing solution, it doesn't take into account version numbers or that a feature may be supported in some versions of a browser and not others.

Rather than use browser detection, the current best practice is to use feature detection to see if a browser supports a given feature.

There are several ways to craft your own feature detection tools like Modernizr. We'll cover each method and then we'll take a quick look at Modernizr and how to use it.

The first way is to look for a property in a global object like window or navigator. If a browser supports a given feature, one of these globall objects will have methods that implement it. The following example creates a function to test

for geolocation and uses it to conditionally use geolocation features:

```
function supportsGeolocation() {
  return !!navigatpr.geolocation;
  // or return 'geolocation' in navigator
}

if supportsGeolocation() {
  console.log('Geolocation is supported')
} else {
  console.log('Geolocation is NOT supported');
}
```

Some times, during development, vendors would prefix a feature to differrentiate it from the standard finalized version. Different vendors used different prefixes:

- Mozilla used `mozObject`
- Chromium browsers and Safari used `webkitObject`
- Microsoft used `msObject`

In this example, we define Audio context to be either the standard (unprefixed) version or the WebKit prefixed one using a logical `or` operator.

```
function supportsAudioContext() {
  return window.AudioContext
    || window.webkitAudioContext;
}

if (supportsAudioContext()) {
  const audioCtx = new AudioContext();
}
```

The next way to test support for an HTML feature is to create an element in memory using `document.createElement()` and check if a property exists on it.

```
function supports_canvas() {
```

```
return
!!document.createElement('canvas').getContext;
}

if(supports_canvas()) {
  console.log('Canvas is suported');
}
```

We can test whether a feature is supported and then we can test for individual properties of that object. The functions below first test if the browser supports video at all (`supports_video`) and then check for each type of populat video used in browsers.

```
function supports_video() {
  return !!document.createElement('video').canPlayType;
}

function supportsH264() {
  if (!supports_video()) {
    return false;
  }

  var v = document.createElement("video");

  return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
}

function supportsOggTheora() {
  if (!supports_video()) {
    return false;
  }

  var v = document.createElement("video");

  return v.canPlayType('video/ogg;  codecs="theora, vorbis"');
}
```

```
function supportsWebm() {
  if (!supports_video()) {
    return false;
  }

  var v = document.createElement("video");

  return v.canPlayType('video/webm; codecs="vp8, vorbis"');
}
```

We can use the functions in nested if statements to get exactly what we need. Just be aware that you may get unexpected results: I wasn't expecting Chrome to support OGG video.

```
// do we support HTML video at all?
if (supports_video) {
  // If so let's look at which formats
  if (supportsH264) {
    console.log('supports h264 baseline');
  }

  if (supportsOggTheora) {
    console.log('supports Ogg');  'supports h264 baseline'sWebm) {
    console.log('supports WebM');
  }
}
```

So far we've seen that it is possible to create custom detection functions but that it takes more work than it may be necessary.

Enter Modernizr, a library that automates CSS and HTML feature detection for you at the cost of adding an additional script to your pages that require feature detection.

Using Modernizr, we can write the tests for video like this:

```
if (Modernizr.video) {
```

```
  // let's play some video! but what kind?
  if (Modernizr.video.webm) {
    // try WebM
  } else if (Modernizr.video.ogg) {
    // try Ogg Theora + Vorbis
    // in an Ogg container
  } else if (Modernizr.video.h264){
    // try H.264 video + AAC audio
    // in an MP4 container
   }
}
```

Which technique you use depends on what you're trying to accomplish. Most times, writing an if statement with the value of what you want to test will be enough but it's always good to know you have options :)