Three.js

Creating a scene

The goal of this section is to give a brief introduction to three.js. We will start by setting up a scene, with a spinning cube. A working example is provided at the bottom of the page in case you get stuck and need help.

Before we start

If you haven't yet, go through the Installation guide. We'll assume you've already set up the same project structure (including index.html and main.js), have installed three.js, and are either running a build tool, or using a local server with a CDN and import maps.

Creating the scene

To actually be able to display anything with three.js, we need three things: scene, camera and renderer, so that we can render the scene with camera.

main.js —

```
import * as THREE from 'three';

const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera( 75, window.innerWidth /
window.innerHeight, 0.1, 1000 );

const renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
```

Let's take a moment to explain what's going on here. We have now set up the scene, our camera and the renderer.

There are a few different cameras in three.js. For now, let's use a PerspectiveCamera.

The first attribute is the field of view. FOV is the extent of the scene that is seen on the display at any given moment. The value is in degrees.

The second one is the aspect ratio. You almost always want to use the width of the element divided by the height, or you'll get the same result as when you play old movies on a widescreen TV - the image looks squished.

The next two attributes are the near and far clipping plane. What that means, is that objects further away from the camera than the value of far or closer than near won't be rendered. You don't have to worry about this now, but you may want to use other values in your apps to get better performance.

Next up is the renderer. In addition to creating the renderer instance, we also need to set the size at which we want it to render our app. It's a good idea to use the width and height of the area we want to fill with our app - in this case, the width and height of the browser window. For performance intensive apps, you can also give setSize smaller values, like window.innerWidth/2 and window.innerHeight/2,

which will make the app render at quarter size.
If you wish to keep the size of your app but render it at a lower resolution, you can do so by calling setSize with false as updateStyle (the third argument). For example, setSize(window.innerWidth/2, window.innerHeight/2, false) will render your app at half resolution, given that your <canvas> has 100% width and height.
Last but not least, we add the renderer element to our HTML document. This is a <canvas> element the renderer uses to display the scene to us.
"That's all good, but where's that cube you promised?" Let's add it now.

```
const geometry = new THREE.BoxGeometry( 1, 1, 1 );
const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
const cube = new THREE.Mesh( geometry, material );
scene.add( cube );

camera.position.z = 5;
```

To create a cube, we need a BoxGeometry. This is an object that contains all the points (vertices) and fill (faces) of the cube. We'll explore this more in the future.
In addition to the geometry, we need a material to color it. Three.js comes with several materials, but we'll stick to the MeshBasicMaterial for now. All materials take an object of properties which will be applied to them. To keep things very simple, we only supply a color attribute of 0x00ff00, which is green. This works the same way that colors work in CSS or Photoshop (hex colors).
The third thing we need is a Mesh. A mesh is an object that takes a geometry, and applies a material to it, which we then can insert to our scene, and move freely around.
By default, when we call scene.add(), the thing we add will be added to the coordinates (0,0,0). This would cause both the camera and the cube to be inside each other. To avoid this, we simply move the camera out a bit.

Rendering the scene

If you copied the code from above into the main.js file we created earlier, you wouldn't be able to see anything. This is because we're not actually rendering anything yet. For that, we need what's called a render or animation loop.

```
function animate() {
	renderer.render( scene, camera );
}
renderer.setAnimationLoop( animate );
```

This will create a loop that causes the renderer to draw the scene every time the screen is refreshed (on a typical screen this means 60 times per second). If you're new to writing games in the browser, you might say "why don't we just create a setInterval ?" The thing is - we could, but requestAnimationFrame which is internally used in WebGLRenderer has a number of advantages. Perhaps the most important one is that it pauses when the user navigates to another browser tab, hence not wasting their precious processing power and battery life.

Animating the cube

If you insert all the code above into the file you created before we began, you should see a green box. Let's make it all a little more interesting by rotating it.
Add the following code right above the renderer.render call in your animate function:

```
cube.rotation.x += 0.01;
cube.rotation.y += 0.01;
```

This will be run every frame (normally 60 times per second), and give the cube a nice rotation animation. Basically, anything you want to move or change while the app is running has to go through the animation loop. You can of course call other functions from there, so that you don't end up with an animate function that's hundreds of lines.

The result

Congratulations! You have now completed your first three.js application. It's simple, but you have to start somewhere.
The full code is available below and as an editable live example. Play around with it to get a better understanding of how it works.
index.html —

```
<!DOCTYPE html>
<html lang="en">
	<head>
		<meta charset="utf-8">
		<title>My first three.js app</title>
		<style>
			body { margin: 0; }
		</style>
	</head>
	<body>
		<script type="module" src="/main.js"></script>
	</body>
</html>
```

main.js —

```
import * as THREE from 'three';

const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera( 75, window.innerWidth /
window.innerHeight, 0.1, 1000 );

const renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.setAnimationLoop( animate );
document.body.appendChild( renderer.domElement );
```

```
const geometry = new THREE.BoxGeometry( 1, 1, 1 );
const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
const cube = new THREE.Mesh( geometry, material );
scene.add( cube );

camera.position.z = 5;

function animate() {

	cube.rotation.x += 0.01;
	cube.rotation.y += 0.01;

	renderer.render( scene, camera );

}
```