**Q&A time**

Before we move on, I would like to address 3 questions which you may (or may not) be thinking ...

**Question 1: why did the first paint happen before the DOMContentLoaded event?**

I answer this a little later when I talk about “**Preload & images and how they affect the CRP**”. But the short answer is that the paint process happens in lots of small chunks.

Each browser has their own unique (and secret) way of painting content to a screen, but the important thing you need to know is that the browser IS allowed to render an incomplete DOM and CSSOM tree. Weird right! This is why in the lecture we saw 3 paint events and I was only concerned about the final paint time (which happened at around 200ms).

**Question 2: What does DOMContentLoaded event mean?**

The DOMContentLoaded event is fired after the HTML code has been fully retrieved from the server, the complete DOM tree has been created and scripts have access to all elements via the DOM API. It is fired on the *document*object (this is different to the load event that fires on the *window*object).

Remember, when the browser reads your HTML document and comes across a <script> tag, it needs to execute these scripts before continuing building the DOM. This makes sense, right? Sometimes you want JavaScript to modify the DOM (e.g. document.write), and so it makes sense that the DOMContentLoaded event has to wait for simple scripts.

In more simple terms, we can say then that the DOMContentLoaded event marks the point where the DOM is ready. And this means that your DOMContentLoaded event will typically fire when the HTML **and your scripts** have been executed (but not necessarily the images or CSS).

Note: there are exceptions to the rule, especially when you reference external scripts which we’ll see in this course (hint: async).

**Why is the DOMContentLoaded important?**

The DOMContentLoaded event typically means that the render tree can begin to be constructed. If our goal is to get to this point as quickly as possible (which it should be) then knowing when our page gets to this point is helpful.

Many developers want to only fire their JavaScript when the DOMContentLoaded event has fired, and this is why you often see JavaScript code wrapped like this: document.addEventListener('DOMContentLoaded', doThings);

**Question 3: If scripts have to execute before the DOMContentLoaded event fires, why does placing JavaScript at the bottom of your file help?**

Short answer: it doesn’t really help to reduce the time it takes for the entire render process. But it can help get the DOM ready faster. And this is important because the paint process happens in stages so the quicker you can get the DOM ready, the better. As I said in the lecture, we are more concerned about the “above-the-fold” content. This means we only care about the first meaningful paint (did you know that Google now ranks pages based on the first contentful paint (FCP). So getting to this point is rather key).

Now, if you put your JavaScript at the top of the page, the browser will start loading your JavaScript files before the html markup, images and other resources. **And this slows things down.**The user may see a blank page while JavaScript is loading (this is because the default behavior of JavaScript is synchronous – things happen sequentially one after the other, and everything will be put on pause until the script is executed).

And this is why you’ll often see developers place JavaScript at the bottom of the file. It actually has a very similar effect to what we saw above. So instead of writing document.addEventListener('DOMContentLoaded', doThings); developers can now include their JavaScript at the bottom of the file. Both of these approaches (including JavaScript at the bottom of the file **VS** executing your scripts after the DOMContentLoaded event has fired) do the same thing.

So why don’t we always place JavaScript at the bottom of our files then?

Good question.

Often we have code that is triggered by certain events being fired (such as the DOMContentLoaded event, or a load event), and we want to have more control as to WHEN we reach these events. Sometimes, for example, we don't mind waiting until much later before executing our JavaScript. This is why today there are better methods at controlling when exactly we want our JavaScript to execute. And for this reason, placing your JavaScript code at the bottom of your page has its own shortcomings and is not always the best solution.

But let me not spill the beans … we are going to be talking about this more in the lectures that follow.

**Bottom line: placing scripts at the bottom of the file is still done to improve the time it takes to get to the first contenful paint.**

Takeaway

Don’t feel overwhelmed.

Right now, all I want you to grasp is that you need to keep the page load process in your mind when writing your JavaScript code. Simple placement of your code (or more modern techniques such as async, defer, etc) can lead to large gains in overall perceived page load time.

Hope this is starting to make (at least a little) more sense.

Keep going and see you soon.