Before we begin...

- Videos On!

Welcome

Agenda

- Browser Internals
- JavaScript and the Browser
- The Document Object Model (D.O.M.)
- DOM Selectors and Traversal
- Creating DOM Nodes
- Events
- Animations

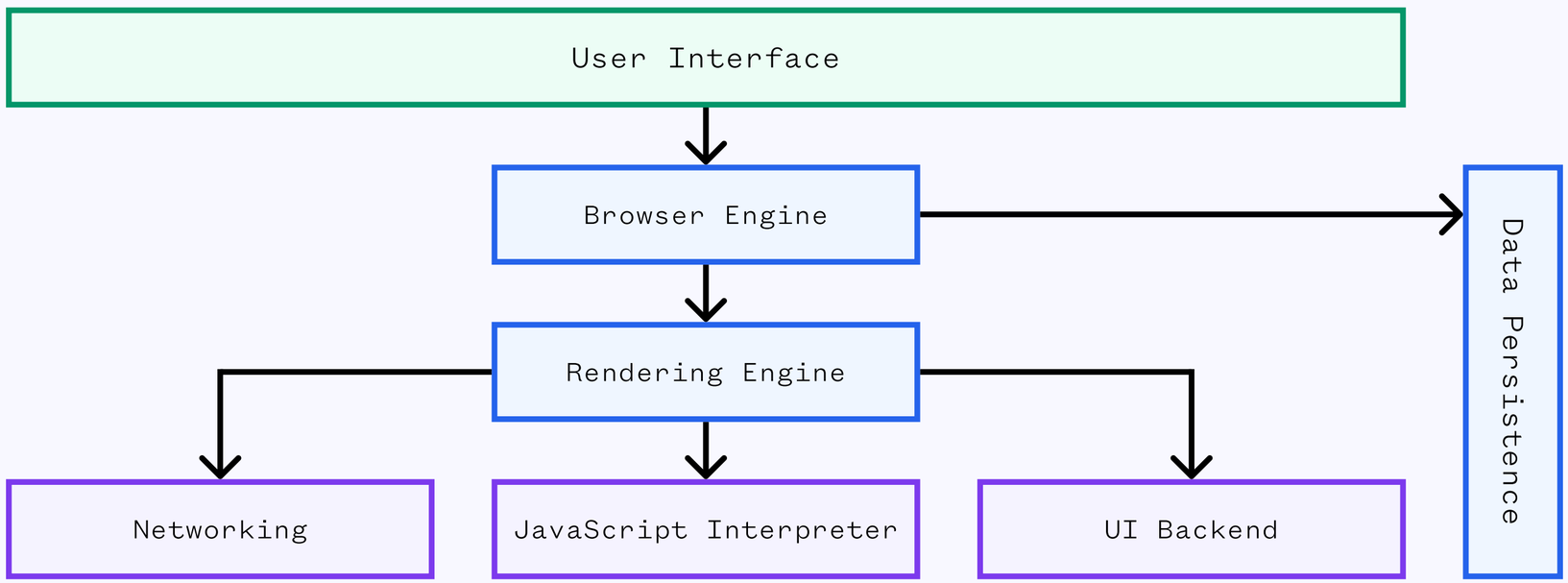
Review

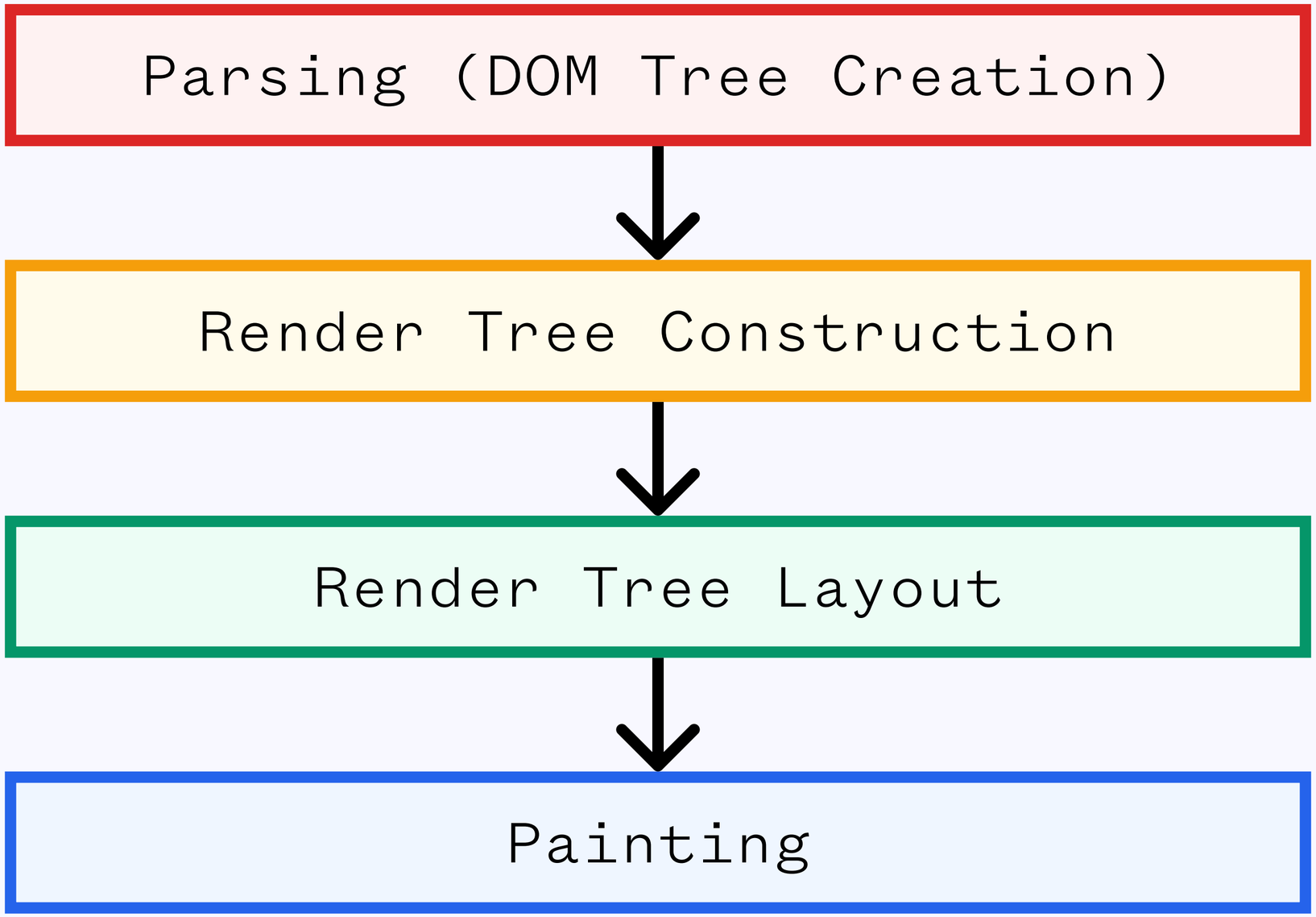
- Pseudocode
- Advanced Functions
  - Callbacks
  - Scope and Hoisting
  - Closures
  - Higher Order Functions
  - Rest Parameters
  - Spread Operators

Welcome

Browser Parts

- User Interface (search bar, menu, address bar etc.)
- Browser Engine (manipulates rendering engine)
- Rendering Engine (renders the page)
- Networking (retrieves URLs)
- UI Backend (draws basic widgets - not just for the browser)
- JavaScript Interpreter (executes JS)
- Data Storage (persistence layer)


User Interface
Browser Engine
Rendering Engine
Networking
JavaScript Interpreter
UI Backend
Data Persistence


Parsing (DOM Tree Creation)
Render Tree Construction
Render Tree Layout
Painting

Resources

- Lin Clark: How do browsers work?
  - Podcast by CodeNewbie
- HTML5 Rocks: How Browsers Work
- Moz://a Hacks: Building the DOM Faster
- Umar Hansa: An Introduction to Browser Rendering

Browser Environment

JavaScript and its Host Environment

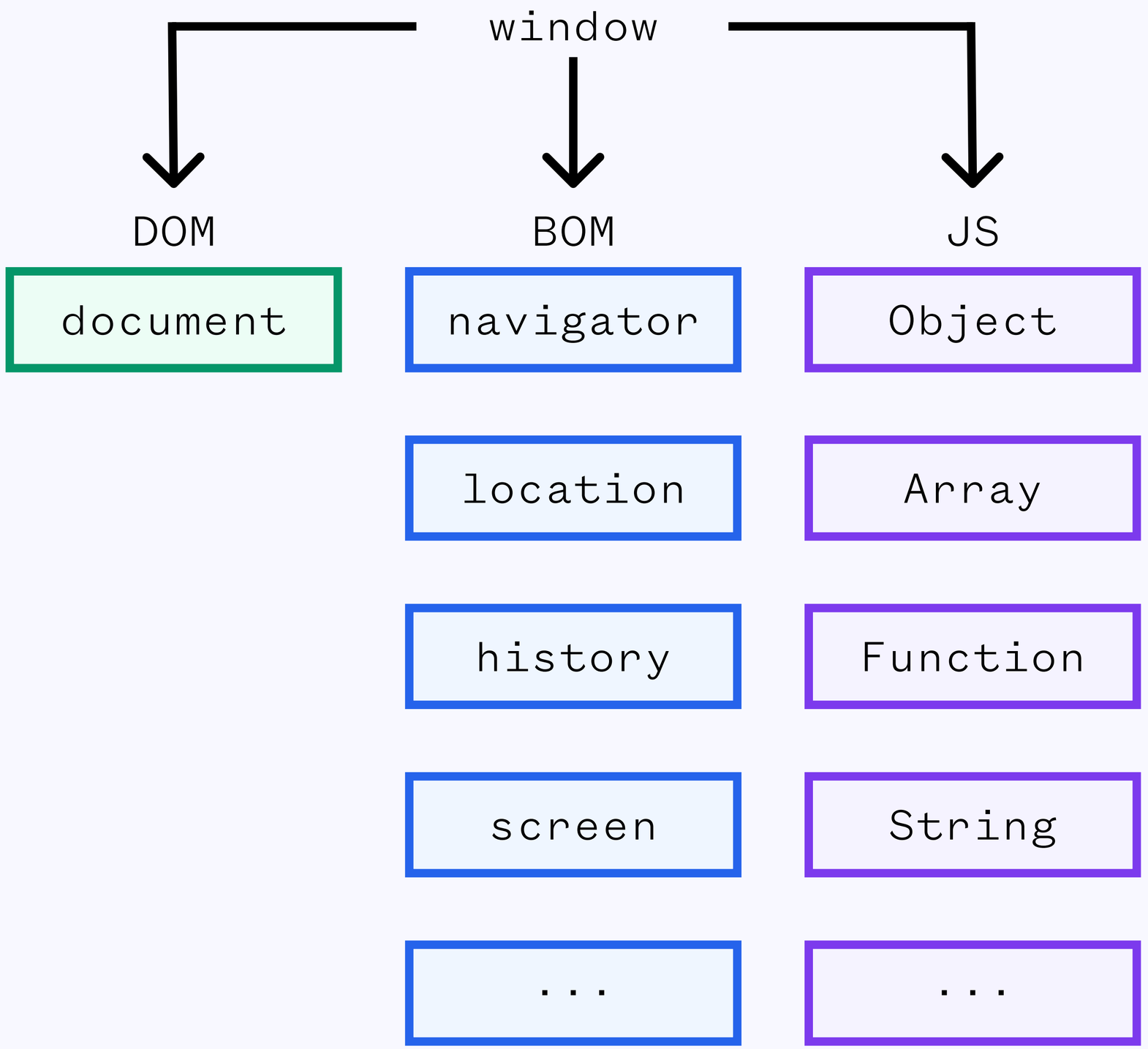
- JavaScript is a language with many uses and it runs on many platforms
- Examples of a platform may be a browser, or a web-server (Node.js). The JavaScript specification (*EMCA-262)*) calls this the **host environment**
- A host environment will always provide objects and functions relevant to that environment. Web pages gives a way to control the browser, Node provides server-side features etc.

The *root* object

In the browser, there is a root object called `window`. It has two main roles:

- It is a global object for JavaScript code (the global scope)
- It represents the browser window and provides methods to control it

So, what is the `window`?


window
DOM
BOM
JS
document
navigator
location
history
screen
...
Object
Array
Function
String
...

The 4 main parts of `window`

- The **Document Object Model**
  - Represents the entire page as an object that can be modified
- The **Browser Object Model**
  - Additional objects provided by the host environment (e.g. information about the browser and operating system, location etc.)
- The **JS Interpreter**
  - Provides all JS data types and syntax
- The **CSS Object Model**
  - Describes stylesheets and style rules

The Document Object Model

What is the **D.O.M**?

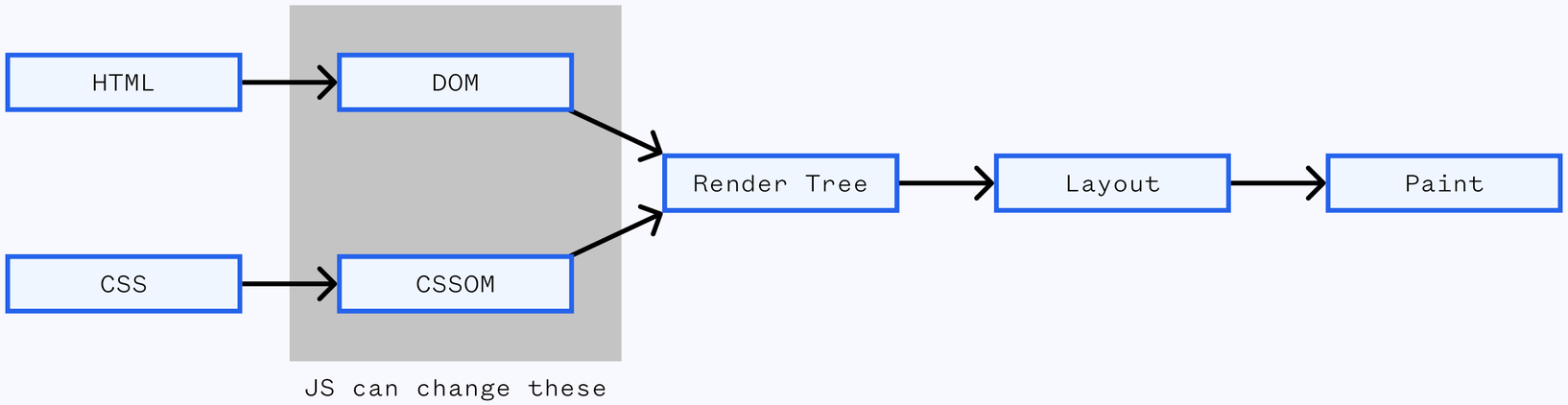
The **Document Object Model** represents the whole page as objects that can be modified, and is often called the **DOM Tree**

We can access, change, create, or delete anything on the page using it

It is represented by the globally available `document` variable

Using it, we can...

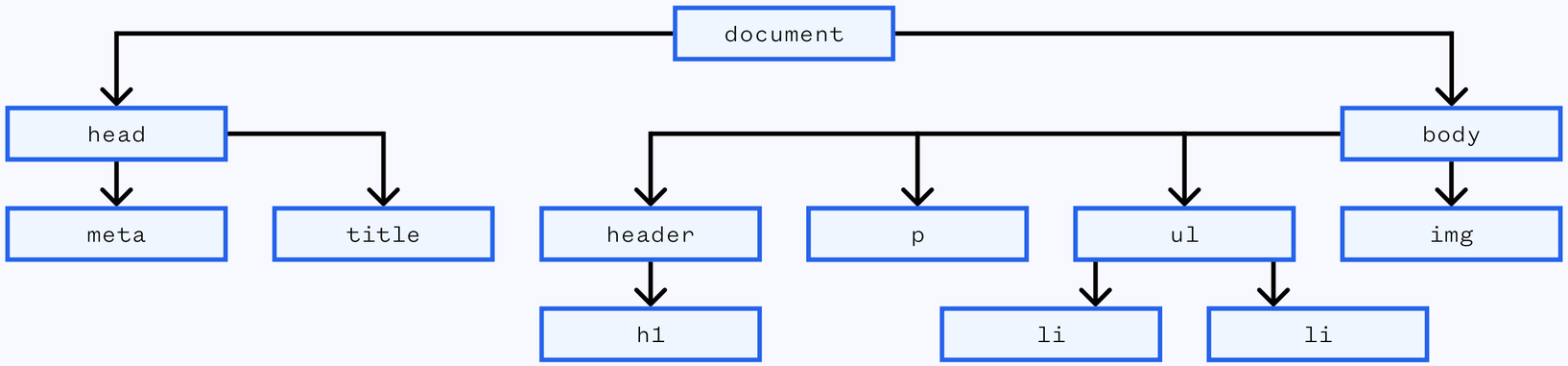
- Add, change or remove HTML elements and attributes
- Perform animations
- Add, change or remove CSS styles
- Add, change or remove event listeners


HTML
DOM
Render Tree
Layout
Paint
CSS
CSSOM
JS can change these

When the DOM changes

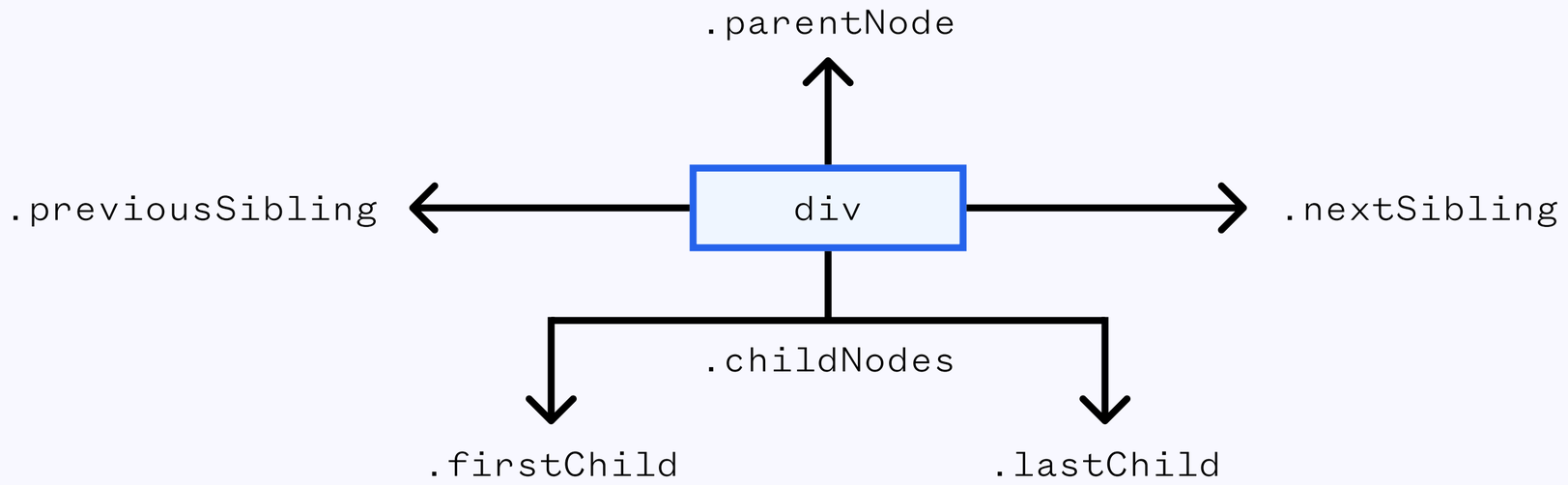
When the **DOM** changes, the page updates

- You make a change to the **DOM tree** using JS (through the `document` variable)
- The browser creates a new render tree
- The browser figures out the layout tree
- The browser re-paints the page


document
head
meta
title
body
header
h1
p
ul
li
li
img

Key Terms

- Each point of data is called a **node**
- Each node can have **parents**, **children** and **siblings**
- The **DOM** is accessed through the `document`
- We can call methods, and access, manipulate and delete properties (just like regular objects)
- It's called the *DOM tree*


.parentNode
.previousSibling
div
.nextSibling
.childNodes
.firstChild
.lastChild

Draw a DOM Tree

```
<!DOCTYPE html>
<html>
  <head>
    <title>Some website</title>
  </head>
  <body>
    <div class="container">
      <h1>Some heading</h1>
      <a href="http://www.google.com">Link</a>
    </div>
    <ul>
      <li>A list item</li>
    </ul>
  </body>
</html>
```

DOM Access

The `document` object gives us ways of accessing the DOM, finding elements, changing styles etc.

The general approach for DOM manipulation:

- Find the DOM Node by using an access method and store it in a variable
- Manipulate the DOM Node by changing its attributes, style, inner HTML or by appending nodes to it

Access Methods

- `document.querySelector()`
- `document.querySelectorAll()`
- `document.getElementById()`
- `document.getElementsByClassName()`
- `document.getElementsByTagName()`

I'd suggest just using the first two!

```
document.querySelector("CSS")
```

`document.querySelector`

Returns the **first** DOM Node that matches a given CSS Selector (or `null`)

```
<h1>Our App</h1>
<p>Welcome</p>

<ul>
  <li>Item</li>
</ul>
```

```
let heading = document.querySelector("h1");
let para = document.querySelector("p");
let item = document.querySelector("ul li");
```

```
document.querySelectorAll("CSS")
```

`document.querySelector`

Returns **all** DOM Nodes that matches a given CSS Selector as a `NodeList` (very similar to an `Array`)

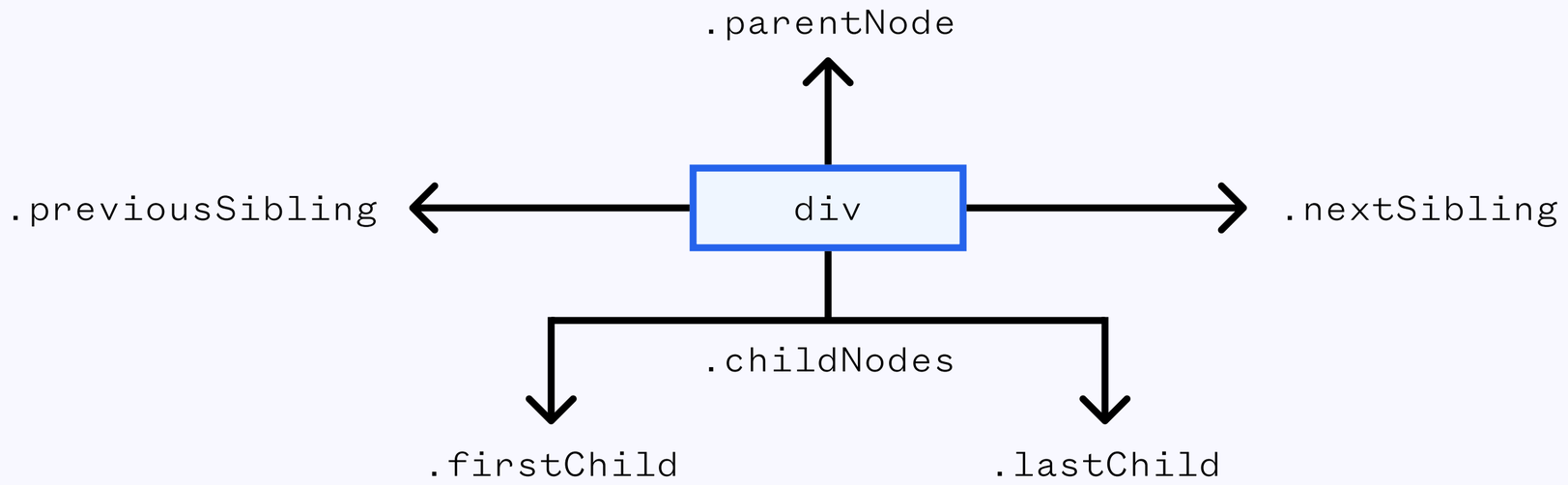
```html
<p>First para</p>
<p>Second para</p>

<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```

```javascript
let paras = document.querySelectorAll("p");
let items = document.querySelectorAll("li");
```

Start the exercises here, please!

See you in 10 minutes


.parentNode
.previousSibling
div
.nextSibling
.childNodes
.firstChild
.lastChild

The DOM Traversal

```html
<div>
  <h1>Hi</h1>
  <p>P tag</p>
  <h3>Heading</h3>
</div>
```

```js
const div = document.querySelector("div");

console.log(div.children);
console.log(div.childNodes);
console.log(div.parentNode);
```

node.getAttribute

```html
<img src="https://picsum.photos/400/300" alt="A nice image" />

<a href="https://ga.co" id="general-assembly">A link to GA</a>
```

```js
let image = document.querySelector("img");
let srcText = image.getAttribute("src");
let altText = image.getAttribute("alt");

let aTag = document.querySelector("a");
let href = aTag.getAttribute("href");
let id = aTag.getAttribute("id");
```

node.setAttribute

```
<img src="https://picsum.photos/400/300" alt="A nice image" />

<a href="https://ga.co" id="general-assembly">A link to GA</a>
```

```
let image = document.querySelector("img");
image.setAttribute("src", "http://picsum.photos/300");
image.setAttribute("alt", "Another image");

let aTag = document.querySelector("a");
aTag.setAttribute("href", "/home");
aTag.setAttribute("id", "home");
```

Working with HTML

```
let heading = document.querySelector("h1");
let currentText = heading.innerText;
let currentHTML = heading.innerHTML;

heading.innerText = "This is the text";
heading.innerHTML = "<u>Hi there</u>";
heading.innerHTML += "!!!";
```

Can anyone think of a reason as to why you need to be careful when changing the text using `.innerHTML`?

Getting Values

```
<input type="text" value="User types here" />
```

```
let input = document.querySelector("input");

let currentValue = input.value;
input.value = "Something else";
let newValue = input.value;
```

Working with Styles

```
<h1>Hello World</h1>
```

```
let heading = document.querySelector("h1");

// Getting Styles
let currentStyles = getComputedStyle(heading);
let fontSize = currentStyles.fontSize;

// Setting Styles
heading.style.width = "400px";
heading.style.fontSize = "24px";
```

Working with Styles

- CSS properties that normally have a hyphen in it, you must camelCase it
- Number properties must have a unit - they won't default to pixels

Start the exercises here, please!

See you in 10 minutes

Creating DOM Nodes

We can make our own DOM Nodes as well

```
let myParagraph = document.createElement("p");
myParagraph.innerText = "Created with JS";
myParagraph.style.fontSize = "24px";
myParagraph.style.color = "hotpink";

// Put it on the page
document.body.appendChild(myParagraph);
// Or...
document.body.insertBefore(myParagraph, document.body.firstChild);
// Or...
document.body.innerHTML += newPara;
```

Events

Some Terminology

- **Event**: something that happens
- **Callback**: a function that executes after the event has happened
- **Event Listener**: a method that binds an event to a callback

Events with JavaScript

There are three important things with events:

- The **target DOM Node** that is going to be interacted with (body, h1, p etc.)
- The **event type** (click, hover, scroll etc.)
- The **callback function** that runs as a response to the event

Events Pseudocode

```
WHEN the element with ID of toggle is CLICKED
    SELECT the body tag and save as body
    CHANGE the body CSS to have a hotpink background

WHEN the element with ID of toggle is CLICKED
SELECT the body tag and save as body
STORE the currentBackground of body

IF currentBackground === "hotpink"
    CHANGE the body CSS to have a ghostwhite background
ELSE
    CHANGE the body CSS to have a hotpink background
```

Creating Events

- Find a **DOM Node** using an access method
- Create a **callback function**
- Create the **event listener**, by binding the event type, the target DOM Node and the callback function)

node.addEventListener

```
let myButton = document.querySelector("button");

function myCallback() {
  console.log("The button was clicked");
}

myButton.addEventListener("click", myCallback);
```

node.removeEventListener

```
let myButton = document.querySelector("button");

function myCallback() {
  console.log("The button was clicked");
}

myButton.addEventListener("click", myCallback);
// Later on...
myButton.removeEventListener("click", myCallback);
```

What events are there?

Given that an event is a signal that something has taken place, there are lots of different events occurring all of the time. We always create event listeners in the same way!

- **Mouse Events** (click, contextmenu, mouseover/mouseout, mousedown/mouseup, mousemove etc.)
- **Keyboard Events** (keydown, keyup etc.)
- **Browser Events** (submit, focus etc.)
- **Form Events** (DOMContentLoaded etc.)
- **Window Events** (scroll etc.)

Callbacks

What are callbacks?

A callback function is really just a regular function passed into another function as an argument

They are very useful because they allow us to schedule asynchronous actions - they are functions that serve as a response (could be an event, or an interaction with an API - or anything, really)

Callbacks

```
function runCallback(cb) {
  // Wait a second...
  cb();
}

function delayedFunction() {
  console.log("I was delayed");
}

runCallback(delayedFunction);
```

Callbacks

```
function sayHi(name) {
  alert("Hello " + name);
}

function processInput(cb) {
  let name = prompt("Please enter your name.");
  cb(name);
}

processInput(greeting);
```

Scheduling

Scheduling

Occasionally, we don't need to run a function straight away - we want to run it after some time has elapsed, or at some regular interval.

`setTimeout` - Delays a function's execution by some amount of milliseconds

`setInterval` - Repeats the execution of a function continuously with an interval in between each call

setTimeout

setTimeout

`setTimeout` allows us to delay the execution of a function by a given number of milliseconds

```
function delayedFunction() {
  console.log("I was delayed!");
}
setTimeout(delayedFunction, 1000);

setTimeout(function () {
  console.log("I was also delayed - but I am anonymous");
}, 2000);
```

setInterval

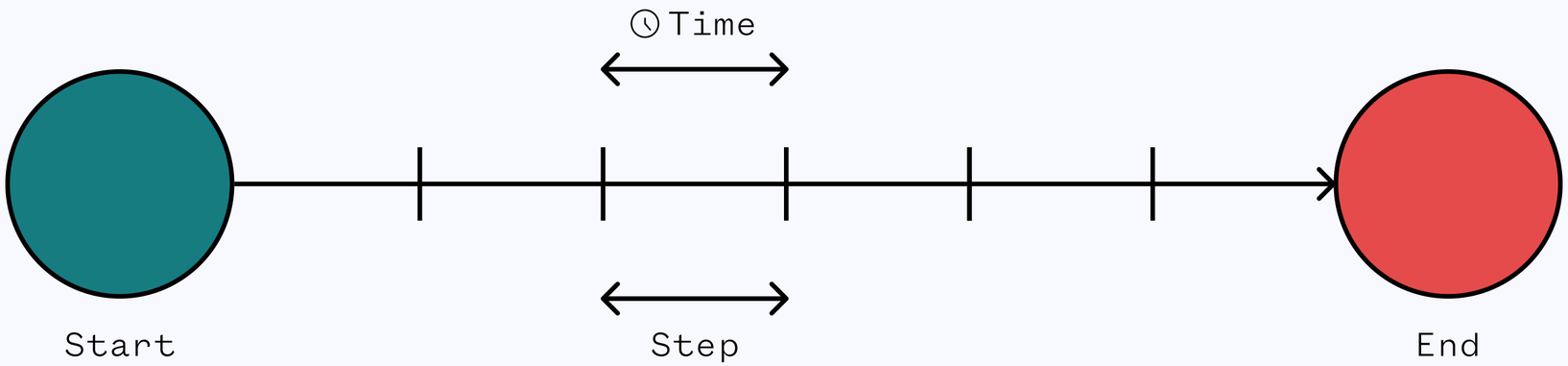
setInterval

`setTimeout` allows us to repeatedly execute a function, with a given number of milliseconds between calls

```
function regularlyCalledFunction() {
  console.log("I am called regularly");
}
let timer = setInterval(regularlyCalledFunction, 1200);
clearInterval(timer); // You can cancel the interval too!

setInterval(function () {
  console.log("I am also called regularly - but I am anonymous");
}, 2000);
```

Animations


Time
Start
Step
End

Animations

Things you need to define:

- Starting Point
- Step
- Time between steps
- Ending Condition

Fade Out: Pseudocode

```
SELECT and STORE the image as myImg

CREATE a function called fadeImgAway
  GET the current opacity and store as currentOpacityAsString
  GET the current opacity as a number and store as currentOpacity

  CREATE newOpacity by subtracting 0.01 from currentOpacity
  UPDATE myImg opacity to be newOpacity

  IF the currentOpacity is >= 0
    CALL fadeImgAway in 10ms

CALL fadeImgAway to start the animation
```

Fade Out

```
let img = document.querySelector("img");

function fadeImgAway() {
  let currentOpacityAsString = getComputedStyle(img).opacity;
  let currentOpacity = parseFloat(currentOpacityAsString, 10);
  img.style.opacity = currentOpacity - 0.01;
  if (currentOpacity >= 0) {
    setTimeout(fadeImgAway, 50);
  }
}

setTimeout(fadeImgAway, 1000);
```

That's all for tonight!

Homework

- Finish off the following exercises:
  - The DOM Detective
  - Replace the Logo
  - More DOM Manipulation
- Go through as much of JavaScript.info's Browser: Documents, Events and Interfaces section as you can

Homework: Extra

- Watch this course on DOM Events and go through DOM Events
- Work on your CSS Selectors using FlukeOut

What's Next?

- JavaScript and the Browser
  - More Events
  - More Animations
  - Bubbling and Capturing
  - Event Propagation
  - Event Delegation
  - Preventing Default Behaviour

Thank You!