
# Intro to the Browser & the DOM

Nick Whyte | @nickw444 | @nickw444

# About Me

- Frontend Engineer / Technical Lead @ Canva
- Graduated UNSW in 2016 (Computer Science)
- COMP2041 student in 2014
- COMP2041 tutor in 2015

# Design any presentation. Publish anywhere.

Work or play, Canva is loved by beginners and experts, teams and individuals.

Get started, it's free!

I already have an account

# 5 Minute HTML Refresher

# Simple HTML5 Document

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Title Goes Here</title>
</head>
<body>
  Hello World
</body>
</html>
```

# Adding CSS

## Inline Styles

```
<head>
  <meta charset="UTF-8">
  <title>Title Goes Here</title>
  <style>
  /* your CSS here */
  </style>
</head>
```

## CSS File

```
<head>
  <meta charset="UTF-8">
  <title>Title Goes Here</title>
  <link rel="stylesheet" href="src/index.css">
</head>
```

# Adding Scripts

## Inline Scripts

```
<head>
  <meta charset="UTF-8">
  <title>Title Goes Here</title>
  <script>
  /* your code here */
  </script>
</head>
```

## Script Files

```
<head>
  <meta charset="UTF-8">
  <title>Title Goes Here</title>
  <script src="src/index.js"></script>
</head>
```

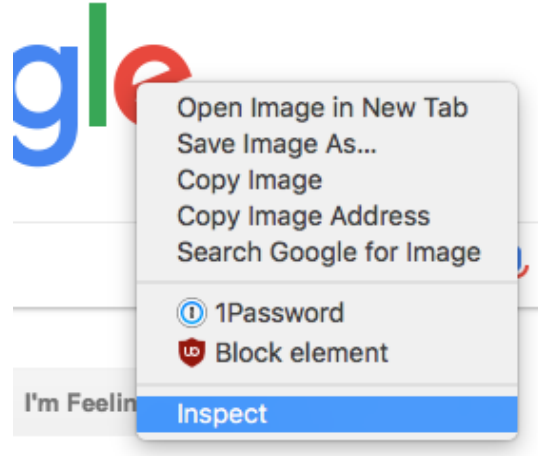
# Basic HTML Elements

- `div`: A container element that can contain zero or more child elements. A block level element.
- `p`: A paragraph element
- `span`: generic inline container for phrasing content. Similar to div, except span is an inline element
- `h1`, `h2`, `h3`, `h4`, `h5`, `h6`: A heading element
- ...and many more!

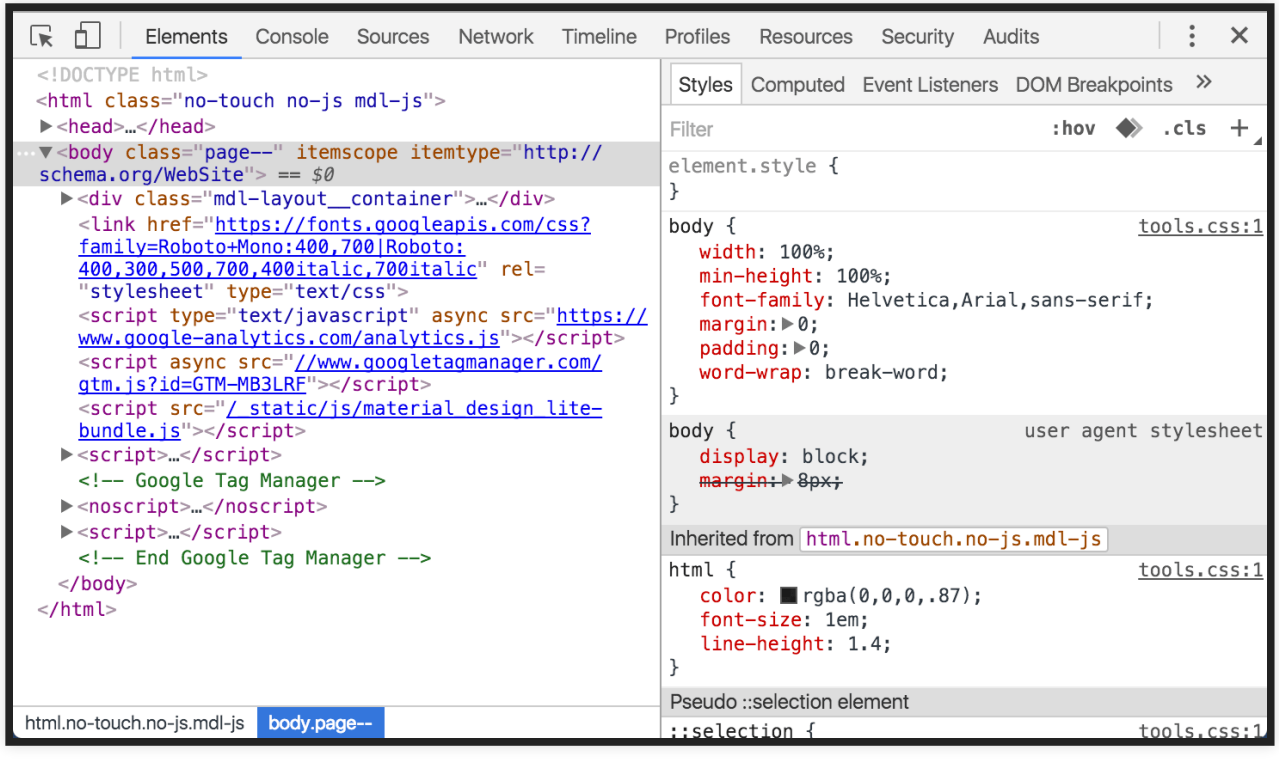
# Let's start building our app!

# Devtools

(Google Chrome)

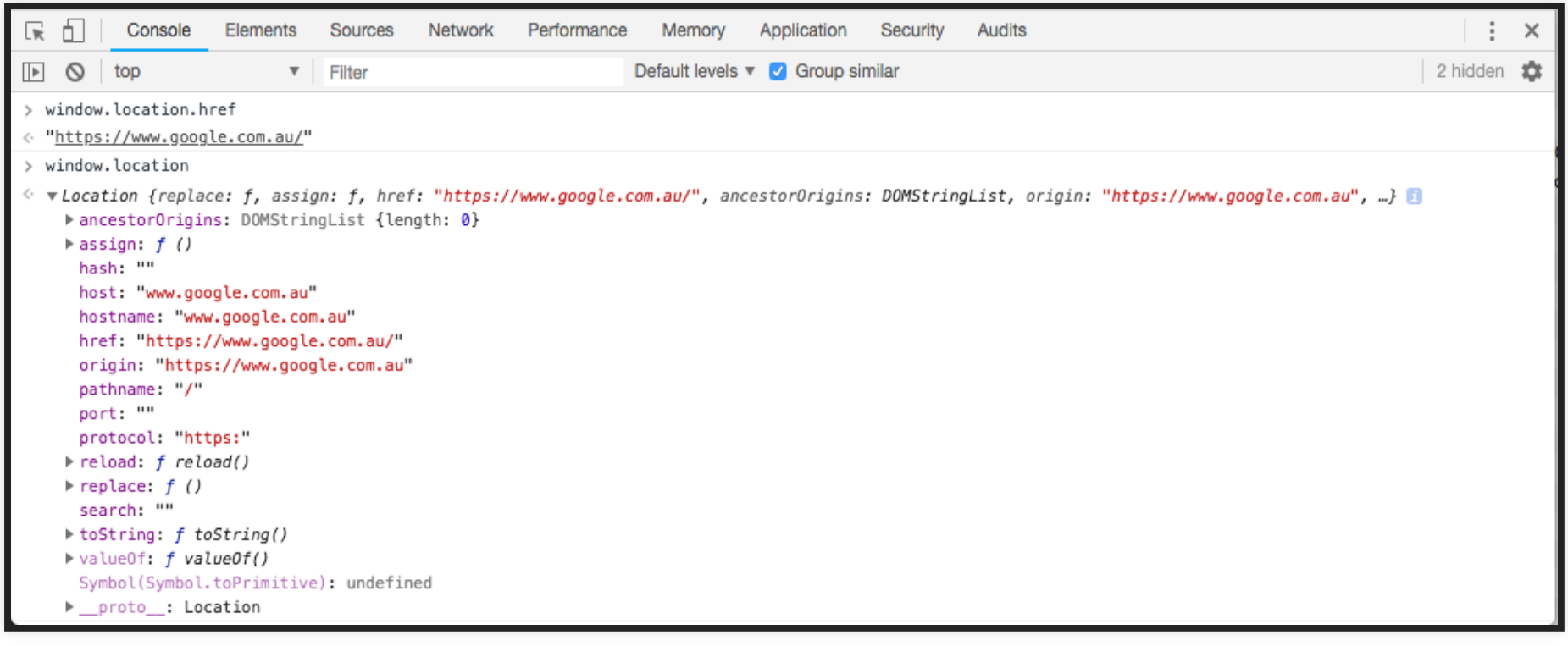


# Elements Panel



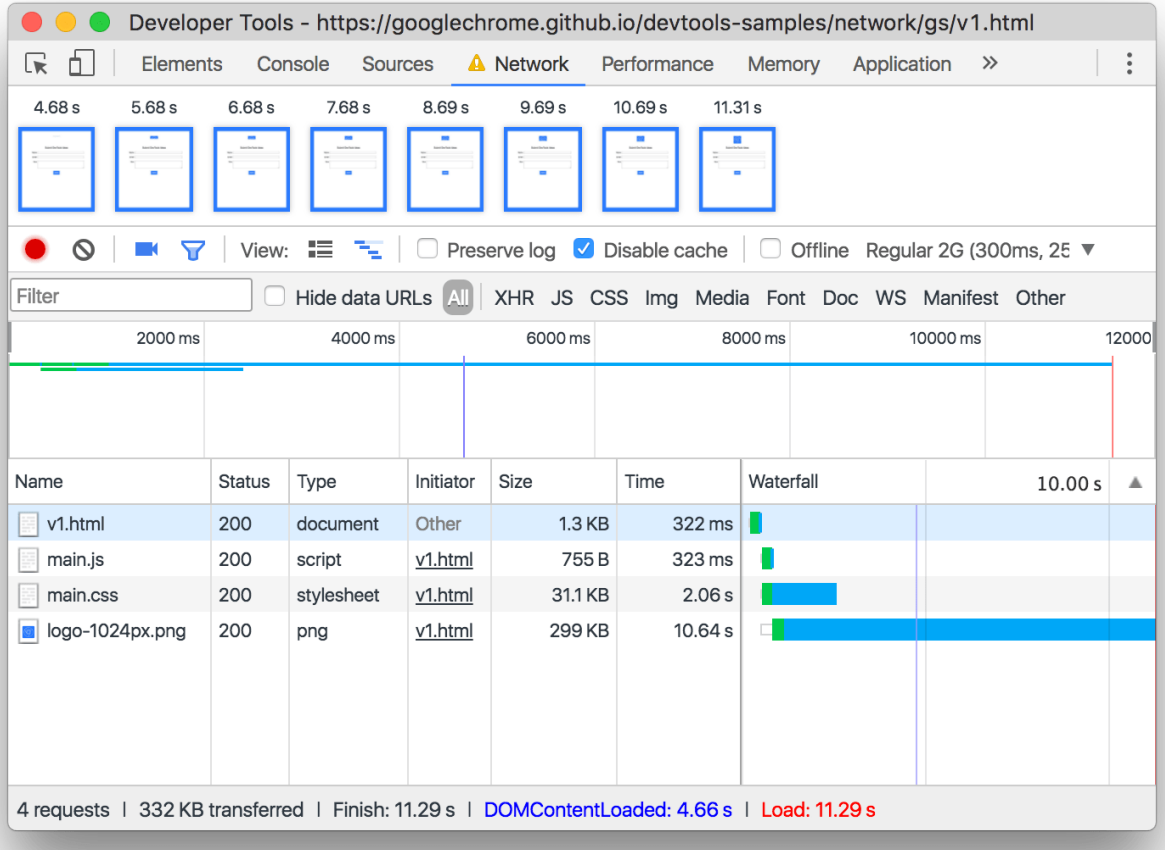

https://developers.google.com/web/tools/chrome-devtools/inspect-styles/

# Console



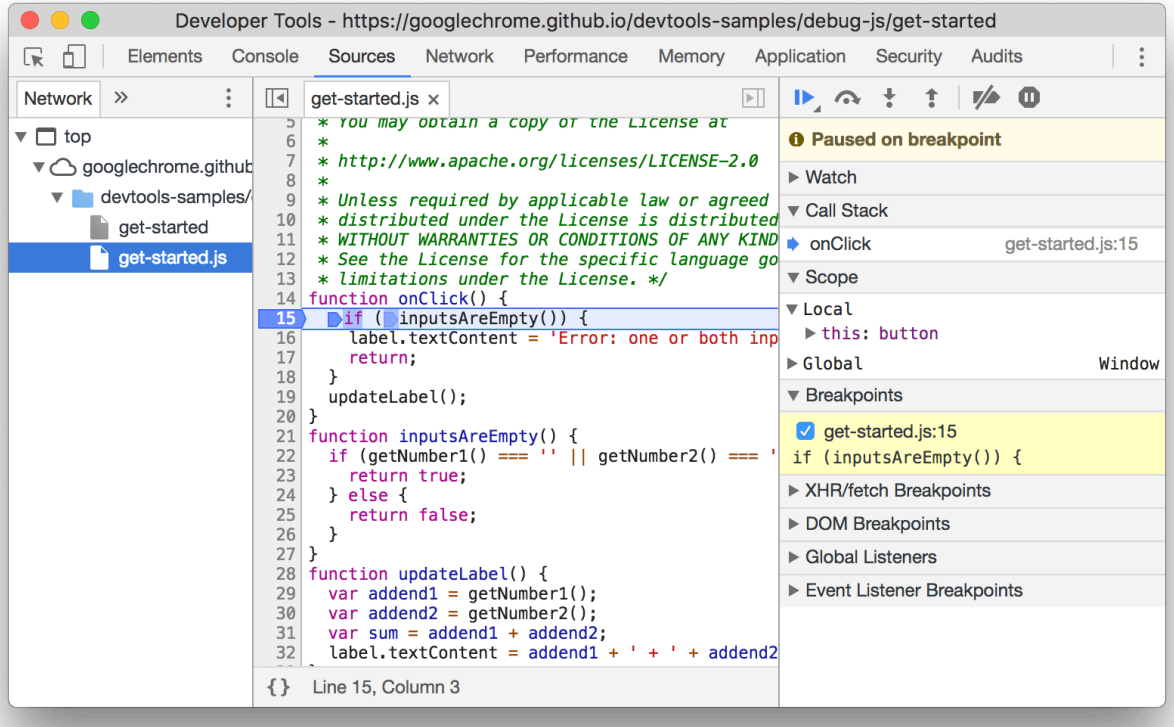

https://developers.google.com/web/tools/chrome-devtools/console/

# Network Inspector




https://developers.google.com/web/tools/chrome-devtools/network-performance/

# Sources Panel

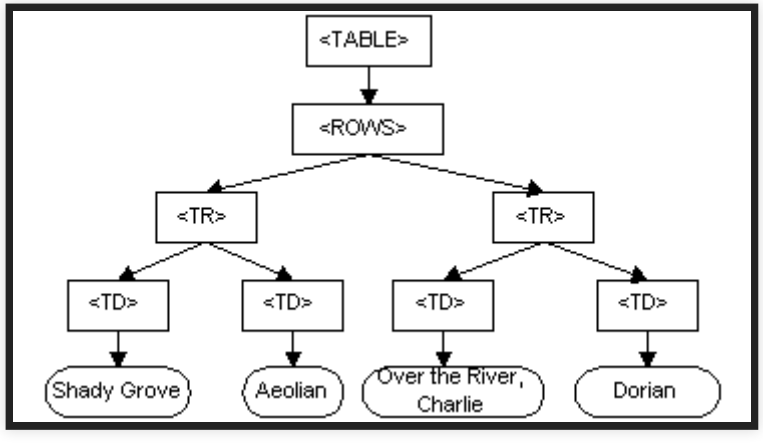



https://developers.google.com/web/tools/chrome-devtools/sources

# Programming For the Browser

# What is the DOM

- An API for HTML (or XML) documents.
- Represents the page so that Javascript can change the structure, style and content of it.
- The DOM represents the document as nodes and objects in a tree.




https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

# `document.?`

- document represents the web page loaded in the window and serves as an entrypoint into the DOM tree.
- `document.head`: Returns the `<head>` element of the current document.
- `document.body`: Returns the `<body>` element of the current document.

# Query the DOM

```
<button id="my-button">Hello</button>
```

## Using `getElementById`

```
const button = document.getElementById('my-button')
```

## Using `querySelector`:

```
const button = document.querySelector('#my-button')
```

Both of these find the element inside the DOM tree with `id="my-button"` and returns it.

# `window.?`

- similar to `document`, represents the window containing the DOM document
- The `Window` interface is home to a variety of functions, namespaces, objects, and constructors

https://developer.mozilla.org/en-US/docs/Web/API/Window

# window.alert

```
window.alert('Hello World!')
```

Run Code

# window.location


```
{
  "href": "http://localhost:8000/01-intro-to-the-browser/?print-p
  "ancestorOrigins": {},
  "origin": "http://localhost:8000",
  "protocol": "http:",
  "host": "localhost:8000",
  "hostname": "localhost",
  "port": "8000",
  "pathname": "/01-intro-to-the-browser/",
  "search": "?print-pdf",
  "hash": ""
}
```

# console.log()

```
console.log('Hello World!')
```

Run Code

n.b. You'll need to open your console to see the output of this demo!

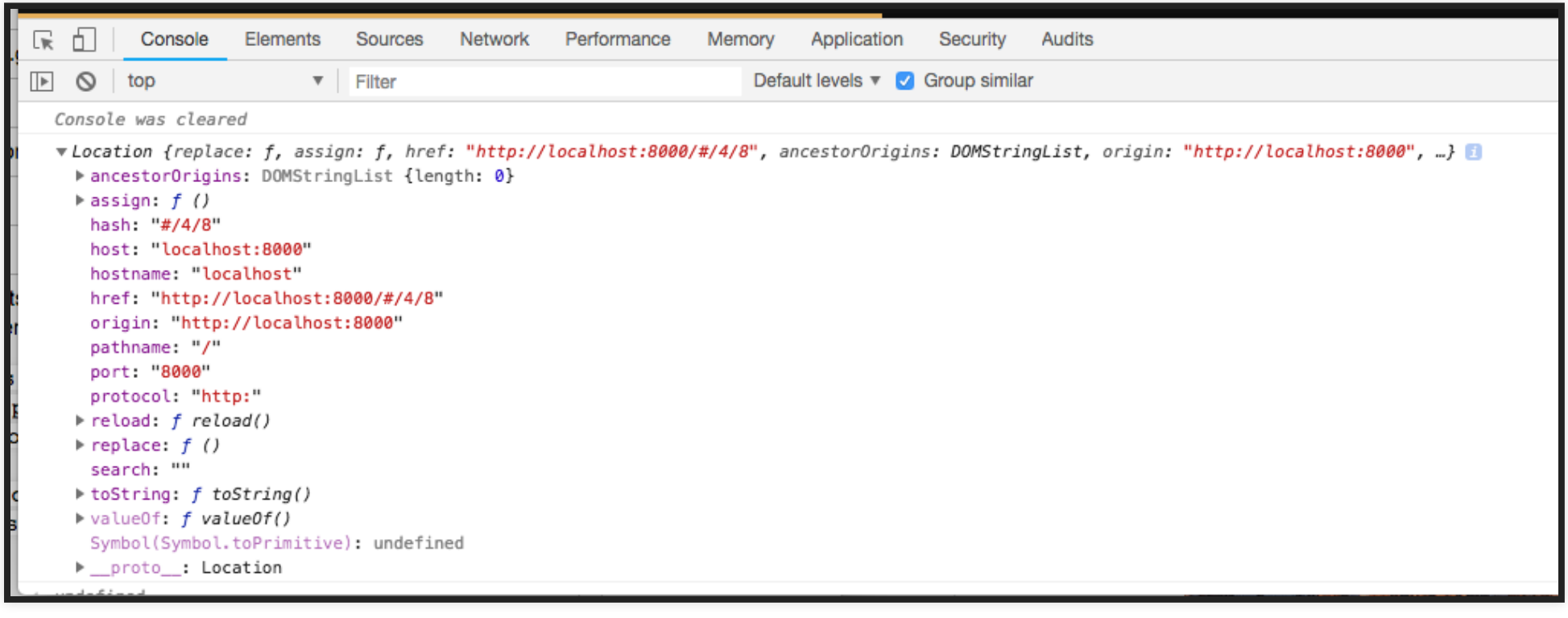
# `console.?`

The console can do a lot more than just plain old strings:

```
console.clear()
console.log()
console.error()
console.warn()
console.table()
console.group()
```

# Console Introspection

You can also log JS objects to the console and introspect them:

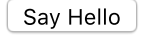


# With this knowledge, lets add some JS to our HTML

# Events & Interactions

# `onClick` event (html)

```
<button onClick="window.alert('Hello COMP2041!')">
  Say Hello
</button>
```



# Binding an event

```
target.addEventListener(type, listener[, options]);
```

- `type`: `'click'`, `'focus'`, `'blur'`, `'keydown'`, etc
- `listener`: The function to execute when the event occurs.

https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

# `onClick` event (html/js)

```
<button id="hello-button">Say Hello</button>
```

```
const helloButton = document.getElementById('hello-button');
helloButton.addEventListener('click', () => {
  window.alert('Hello COMP2041 from JS!');
});
```

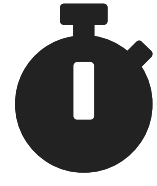
# mouseMove **event**

```
window.addEventListener('mousemove', ({x, y}) => {
  console.log({x, y});
});
```

```
{
  "x": 803,
  "y": 493
}
```

# Let's make our app interactive

# Timers



# setTimeout / clearTimeout

```
document.setTimeout(() => {
  window.alert('Done!')
}, 1000);
```

Run Code

https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setTimeout

# setInterval / clearInterval

```
let timer = undefined;
let counter = document.getElementById('counter');

function tick() {
  counter.innerText = parseInt(counter.innerText) + 1;
}

function onStartClick() {
  timer = window.setInterval(tick, 1000);
}
function onStopClick() {
  window.clearInterval(timer);
}
```

Counter: 0

Start Stop

# Time to make our stopwatch tick!

# Accuracy of Timers

- Timers in JS are not accurate and can become skewed.
- If we made a timer with just setTimeout/setInterval we would find the timer would drift if there was other computational work to perform.
- Browsers will typically pause timers when you switch tabs to conserve resources 😎

# Clock Drift Demo

## Actual Time

0.000

## Interval Time

0.000

Start | Run Long / Blocking Task

https://johnresig.com/blog/how-javascript-timers-work/
https://www.sitepoint.com/creating-accurate-timers-in-javascript/
https://www.reddit.com/r/learnjavascript/comments/3aqtzf/issue_with_setinterval_function_losing_accuracy/

# Solution To Clock Drift

- Store the start time and calculate the time elapsed on each render cycle
- Use setInterval/setTimeout to invoke a render cycle (or better yet; use `requestAnimationFrame` for easy 60FPS updates)

# Lets fix the clock drift in our application

# Behind the Scenes

(How JS gets executed in your browser)

# Scope

- When your browser encounters a script tag it (downloads and) executes it, placing all global variables onto `window`.
- This can pollute the global scope and cause conflicts.
- We can solve this with an IIFE:

```
(function() {
  let foo = 'whatever'
  // More code here
})();
// foo is unreachable here
```

# IIFE's

- Immediately-invoked function expression
- Produces a lexical scope using JavaScript's function scoping
- Stops global `window` pollution, since all variables sit within the function scope.

```
(function() {
  let foo = 'whatever'
  // More code here
})();
// foo is unreachable here
```

https://en.wikipedia.org/wiki/Immediately-invoked_function_expression

# Execution Order

- A `<script>` tag is parsed & executed as it is encountered when the DOM is parsed.
- This means it may execute too soon, before all HTML elements are even available.
- If the script is large or complex it may block HTML rendering

https://eager.io/blog/everything-I-know-about-the-script-tag/

# Execution Order

This is an easy problem to solve in a few different ways;

- Move the script below the HTML elements it targets (or to the bottom of the source)
- Use an event listener in JS to listen to when the document is ready (`DOMContentLoaded`):

```
function init() {
  console.log('Document loaded!');
}
document.addEventListener('DOMContentLoaded', init);
```

https://developer.mozilla.org/en-US/docs/Web/Events/DOMContentLoaded

# Run to Completion

- Each message is processed completely before any other message is processed.
- Your functions cannot be preempted by another
- Your function will run entirely before any other code runs

https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop#Run-to-completion

# Example

```
const s = new Date().getSeconds();

setTimeout(function () {
  // prints out "2", meaning that the callback is not called immediately after 50
  console.log("Ran after " + (new Date().getSeconds() - s) + " seconds");
}, 500);

while (true) {
  if (new Date().getSeconds() - s >= 2) {
    console.log("Good, looped for 2 seconds");
    break;
  }
}
```

Run Code

https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop#Run-to-completion

# Querying the DOM

# Finding Elements

- `getElementById`: Finds an element by it's `id` attribute. Returns an `Element` object.
- `getElementsByClassName`: Finds elements with a matching `class` attribute. Returns an array-like object.
- `getElementsByTagName`: Finds elements with a matching tag name.
- `querySelector`: Magical jQuery-like DOM query method. Returns the first element matching the CSS selector given.

## Using `querySelector`

```
// id="my-button"
const button = document.querySelector('#my-button')
// class="my-button"
const button = document.querySelector('.my-button')
// class="my-button" nested inside <body>
const button = document.querySelector('body > .my-button')
```

# Querying Element Content

Sometimes you may want to read the content of an element

Hello World!

```
<p id="text">Hello World!</p>
```

```
const text = document.getElementById('text');
console.log(text.innerText);
```

Run Code

# Querying Element Bounds

Hello World!

```
<p id="text">Hello World!</p>
```

```
const text = document.getElementById('text');
console.log(text.getBoundingClientRect());
```

Run Code

# Manipulating the DOM

# Adding a CSS class

This demo adds a class to a `span` node.

Hello World!

```
.red { color: red; }
```

```
<p>Hello <span id="worldSpan">World!</span></p>
```

```
const worldSpan = document.getElementById('worldSpan');
span.classList.add('red');
```

Run Code

# Modifying Inline Styles

Hello World!

```
<p>Hello <span id="worldSpan">World!</span></p>
```

```
const worldSpan = document.getElementById('worldSpan');
span.style.fontSize = '2em';
```

Run Code

# Modifying innertext

It's also easy to update the content of a HTML Element

Hello World!

```
<p id="text">Hello World!</p>
```

```
const text = document.getElementById('text');
text.innerText = 'Goodbye World!';
```

Run Code

**Now that we know how to manipulate the DOM, lets render the stopwatch on the page.**

60FPS

# `window.requestAnimationFrame()`

- Tells the browser that you wish to perform an animation and to call the given function before the next repaint.
- Call this method whenever you're ready to update your animation onscreen.
- The number of callbacks is usually 60 times per second, but should match the display refresh rate.

https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

# requestAnimationFrame demo

```
const box = document.getElementById('box');
let offset = 0;

function renderFrame() {
  requestAnimationFrame(() => {
    offset += 10;
    box.style.transform = `translateX(${offset}px)`;

    // Enqueue a subsequent render cycle for the next frame.
    renderFrame();
  });
}

renderFrame();
```

Run Code

**Using `requestAnimationFrame` lets improve our application to have 60fps updates**


# Thanks

Nick Whyte | @nickw444 | @nickw444

p.s. we are looking for summer interns and 2019 graduates!
Please email nick@canva.com if you are interested