The DOM

Debugging

🎓FUNdamentals

A programmer spends almost all of her time writing code.

TRUE or **FALSE**

Writing code is only one of the many things that a programmer does.

We spend lots of time doing other things as well.

- Reading documentation
- Reading code
- Researching (googling)
- **Debugging code**

🎓FUNdamentals: debugging

Bugs! Where do they come from?

- Typos
- Forgot to pass an argument
- Pass the wrong type of data as an argument
- Make wrong assumptions
- A million other things

🎓FUNdamentals: debugging

Exceptions

In certain cases, a bug will cause your code to crash. **This is a GOOD thing**.

It will usually contain an error message that *tells you*

- where the problem is
- where to start looking for the problem

🎓FUNdamentals: debugging

Exceptions

In certain cases, a bug will cause your code to crash. **This is a GOOD thing**.

It will usually contain an error message that *tells you*

- where the problem is
- where to start looking for the problem

The actual programmer mistake can very well be elsewhere.

This is similar to how a human error in a factory will manifest itself only in the final product.

🎓FUNdamentals: debugging

Finding Exceptions

The error message is missing perhaps 5% of the time (rough).

This makes it hard to find the bug.

This is where using **console.log()** can really help.

🎓FUNdamentals: debugging

console.log

Learning to **console.log** effectively is an *essential* part of becoming a developer.

It allows you to be independent.

It's one of the most important parts of this course.

🎓FUNdamentals: debugging

Example 1

```
const x = 5;
const y = [1, 2, 3]

y.map(x);
```

```
> y.map(x);
Uncaught TypeError: 5 is not a function
    at Array.map (<anonymous>)
```

Example 2

```
function getCartTotal(data) {
  let salesTax = 1.14;
  return data.cart.subtotal * salesTax;
}
getCartTotal({
  items: ['banana'],
  subtotal: 5
});
```

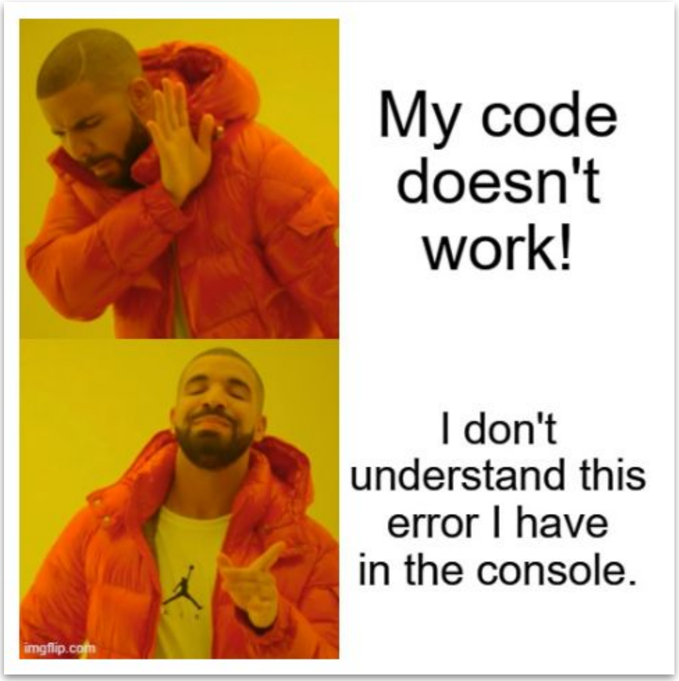
```
> ../test.js:3
  return data.cart.subtotal * salesTax;
                   ^
TypeError: Cannot read property
'subtotal' of undefined
```

🎓FUNdamentals: debugging

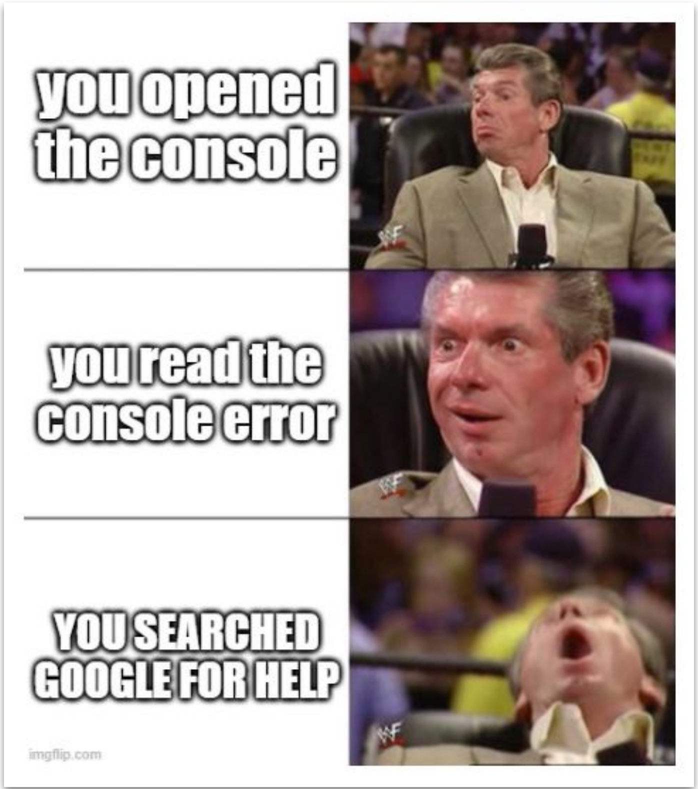
Always check the console.

Always read error messages.

It should be your FIRST reflex.



🎓FUNdamentals: debugging



Use your Google Fu!

- Search for the error message, without any custom variable names, and "javascript"
- Search for the problem domain (breaking into smaller pieces)
- Be skeptical
- Skim results

🎓FUNdamentals: testing

What is testing?

Why do we test?

When do we test?

🎓FUNdamentals: testing

Code

```
function strLength(str) {
  if (typeof str !== "string" || str.length === 0) {
    return undefined;
  }
  return str.length;
};
```

Test

```
test("Exercise 0", function () {
  expect(strLength("max")).toBe(3);
  expect(strLength("abcdefghijklmnop")).toBe(16);
  expect(strLength("This is a test case.")).toBe(20);
  expect(strLength("")).toBe(undefined);
  expect(strLength(256)).toBe(undefined);
  expect(strLength(["abcdefghijklmnop"])).toBe(undefined);
});
```

The DOM

Manipulating the DOM

The DOM

When you load a web page in the browser...

1. Retrieves the HTML text and parses it.

2. Builds a *model* of the document structure

3. Uses this model to render the page on the screen.

This is the **D**ocument **O**bject **M**odel.

The DOM

The DOM is a data structure that we can read and modify.

It acts as a *live* data structure. When it's modified, the page on the screen is updated. 🤯

The DOM

You can see it in your developer tools in the browser.

It looks *almost* identical to the HTML you wrote...

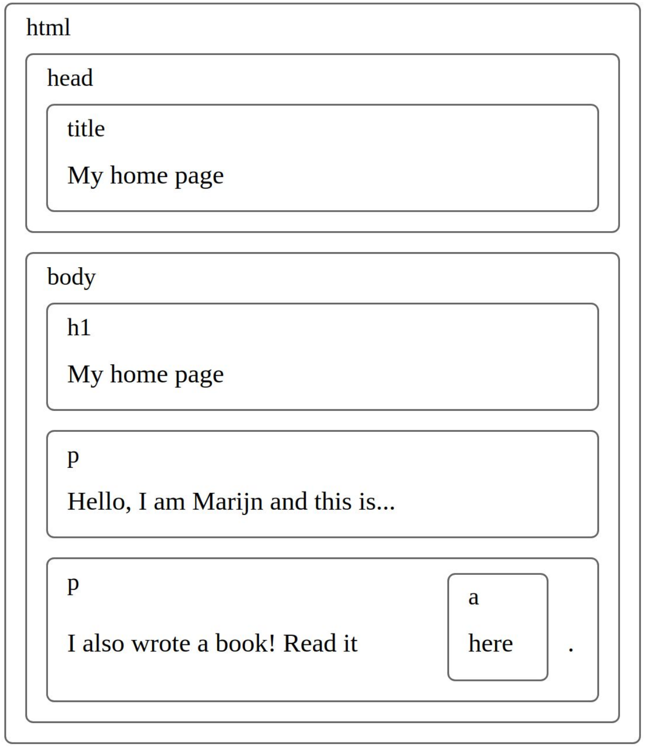
The DOM is actually your _corrected_ HTML.

⚠️ This means that it is impossible to debug your HTML with the dev tools.

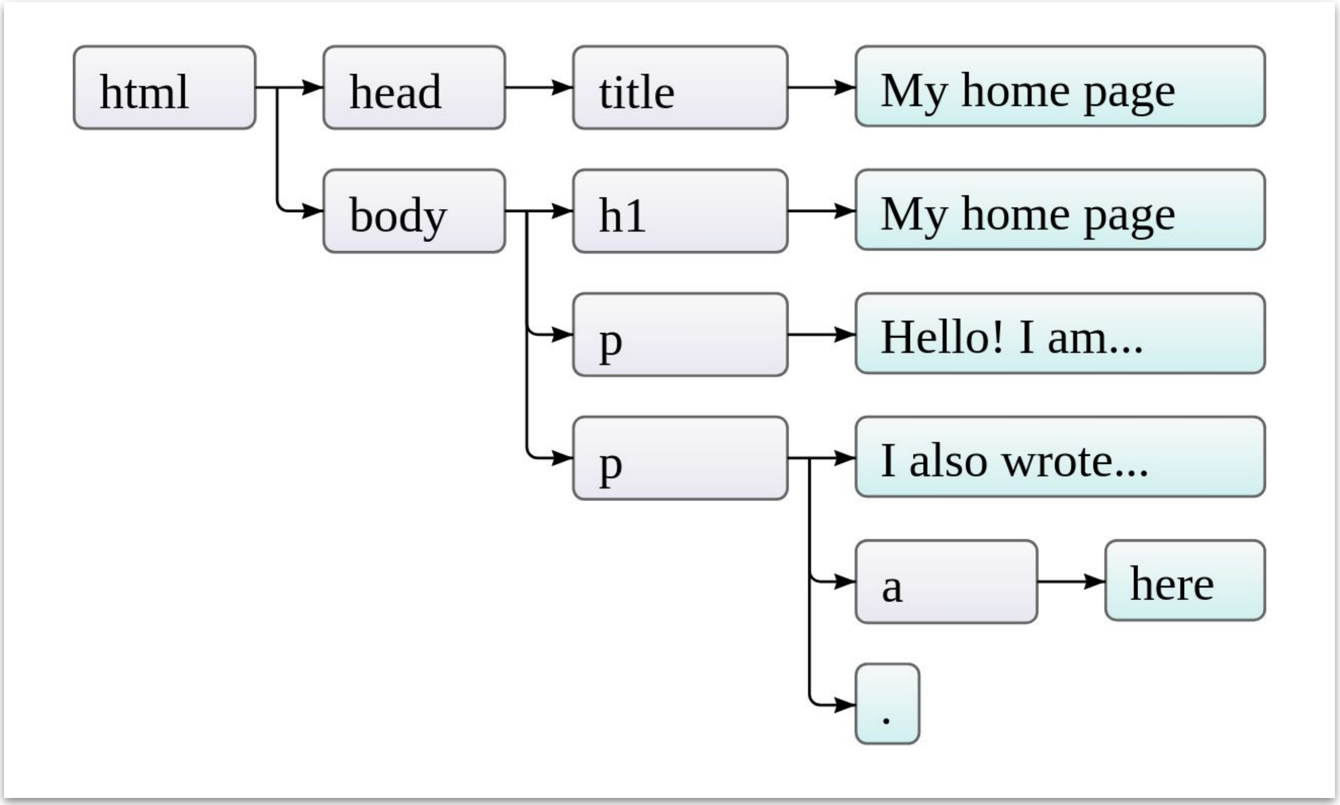
The DOM

A nested set of boxes

- For each box, there is an object that we can interact with.
- Each node may contain/refer to other nodes that we call *children*.
- Similar to a tree.
- End nodes usually called *leaves*.

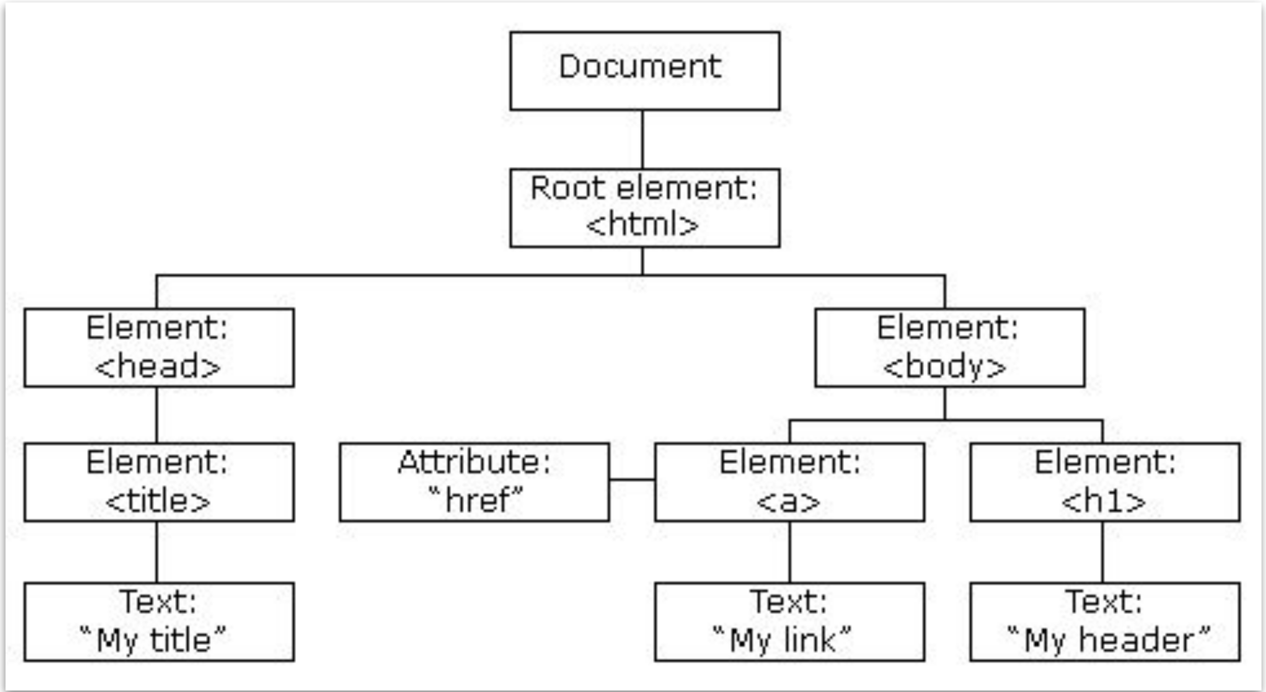

html
head
title
My home page
body
h1
My home page
p
Hello, I am Marijn and this is...
p
I also wrote a book! Read it
a
here
.

The DOM


html
head
title
My home page
body
h1
My home page
p
Hello! I am...
p
I also wrote...
a
here
.

The DOM

JavaScript can modify *all* of the HTML elements on the page.



The DOM: Target element

Get/grab an element

You can access a single DOM node using **document.querySelector()**

This takes a CSS selector as an argument.

```html
<div class="container">
  <h1 id="title">The title</h1>
</div>
```

```javascript
const container = document.querySelector('.container');

const title = document.querySelector('#title');

// or

const title = document.getElementById('title');
// notice no # is necessary
```

The DOM: Modify

Modify an element (node)

You can modify the content of a node with

- **.innerText** 🔗
- **.innerHTML** 🔗

The DOM: Create

Create an element (node)

To add a new node to an HTML page, you need to do it in 3 steps.

1. Create the new node
2. Add content to that node
3. Add that node to an existing node.

- **.document.createElement()** 🔗
- **.appendChild()** 🔗

The DOM: Style

Style an element (node)

1. Target the element using one of the methods we've just learned.
2. Modify its **style** attribute with *.style*.

```
const container = document.querySelector('.container');

container.style.background = “purple”;
```

This adds inline CSS.

The DOM: Style

You can modify a node's class attribute with .classList

myDiv.classList returns a DOMTokenList that is *read only*. 😭

But it modifiable with various methods! 😃

- **.add()**
- **.remove()**
- **.toggle()**

The DOM

FUNdamentals: Timing and delay

🎓FUNdamentals - Timing and delay

```
setTimeout(function () {
  // do something
}, time_in_milliseconds);
```

```
setInterval(function () {
  // do something
}, time_in_milliseconds);
```

```
const doSomething = function () {
  // do something
};

setTimeout(doSomething, 3000);
```

```
const makeBacon = function () {
  const amount = Math.floor(Math.random() * 6);
  let output = '';
  for (let i = 0; i < amount; i++) {
    output += '🥓';
  }
  console.log(output);
};

setInterval(makeBacon, 3000);
```

🎓FUNdamentals - Timing and delay

💡 Always use **clearInterval** to stop your **setInterval** loop.

This will require the **setInterval** to be declared.

```
const makeBacon = function () {
  const amount = Math.floor(Math.random() * 6);
  let output = '';
  for (let i = 0; i < amount; i++) {
    output += '🥓';
  }
  console.log(output);
};

// Declaring the interval also triggers
// the interval
const baconInterval = setInterval(makeBacon,
3000);

// allows us to do
clearInterval(baconInterval);
```