

Standardised by W3C

Standardised by ECMA

HTML

CSS

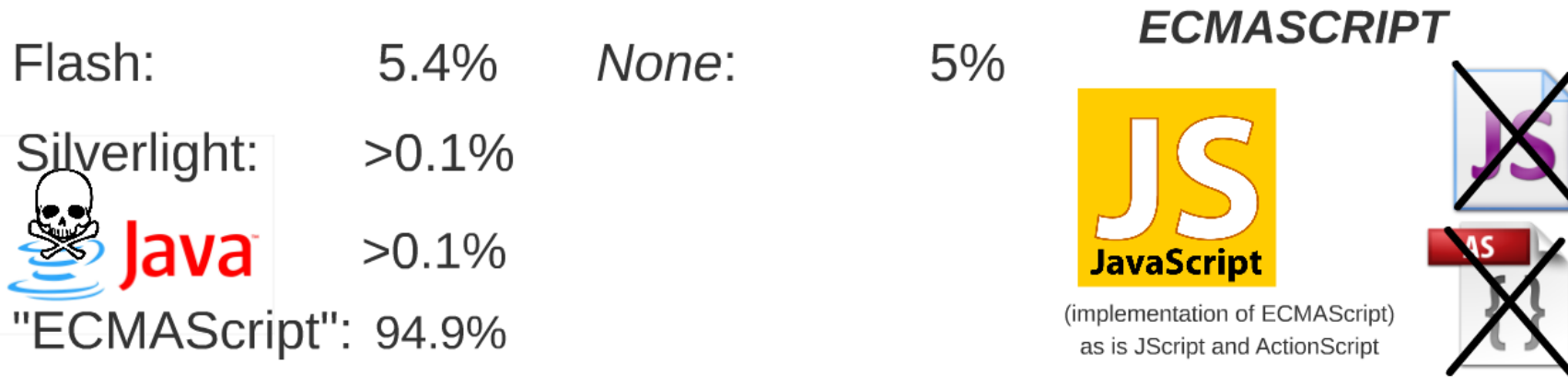
JS



DOM



Client Side Processing



Source: W3Techs.com ~ 31 December 2017

JavaScript attempts to work with the web page. There's a console; but the way in which the effects of JavaScript are felt by the user are the ways in which it interacts with the web page itself.

All the processing is done on the user's computer. This is outsourcing of computation from a design point of view

Overlap of W3C and ECMA is DOM

The overlap is in the Document Object Model

All browsers must parse HTML and create a tree like structure out of it.

For instance Chrome and Opera use Blink while Firefox uses Gecko and Internet Explorer, Trident.

But if you want to edit this structure after the fact? Some consistent model with which to work with is necessary.

DOM Standardisation

DOM Level 0 (Created by Netscape at same time as JavaScript) to be used directly by the language for use primarily with html forms and images.

Microsoft created its own DOM Level 0 (and flavour of JavaScript, namely JScript) as the two companies competed through the 90s.

W3C eventually helped create a standardised DOM Level 1. Later versions, with extended functionality (particularly in relation to XML) were subsequently released. We are currently on DOM Level 4.

Implications

A standardised DOM means you have a universally accepted tree structure for scripting to work with: so if you wish to get an array of every link on a web page, or insert additional tags in an existing web page, there's a guaranteed structure with which to work.

What will we cover?

JavaScript is huge.

JavaScript has several popular frameworks

- What ways is JavaScript similar to what you've seen before?
- What ways is JavaScript different from what you've seen before?
- What is the purpose of JavaScript?
- Avoiding the bad aspects of JavaScript

What won't we cover?

Development of full-blown web apps/games

Angular or React

AJAX (in practice)

JavaScript OOP

Intermediary languages (CoffeeScript etc.)

Creating custom HTML elements

Generic programming stuff

JavaScript is easy in all the wrong ways

Want to leave out a semicolon? *Sure*

Want to forget about scope? *Okay*

Mash together different types? *Er... is it okay if it makes no sense?*

The principle of encouraging development...
even if that development was awful

Is JavaScript a bad language?

No. Like all languages, you'd only use it in
certain circumstances.

When to use JavaScript?

Data visualisation

Web development

Full stack development

You can do **everything** in JavaScript.

- Dynamic parts of a website
 - A website can re-write itself using JavaScript
- Some visual aspects

What do you already know?

- Variables
- Scope, parentheses, statements end with a semicolon...
- Loops, logical statements, switch, *and so on*

unless you are a
Python head

```
if (flagIsTrue){  
    for (i = 0; i < 10; i++) {  
        j+=i;  
    }  
}  
else flagIsTrue = 1;
```

← C /
JavaScript

- Functions
- Objects

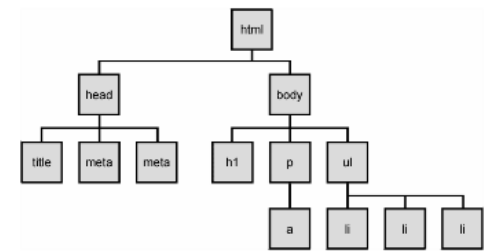
no pointers

Working with the DOM

One of the main purpose of JavaScript is to work with the Document Object Model API

This means that a JavaScript script can access any node on the DOM tree.

What is a node? HTML element

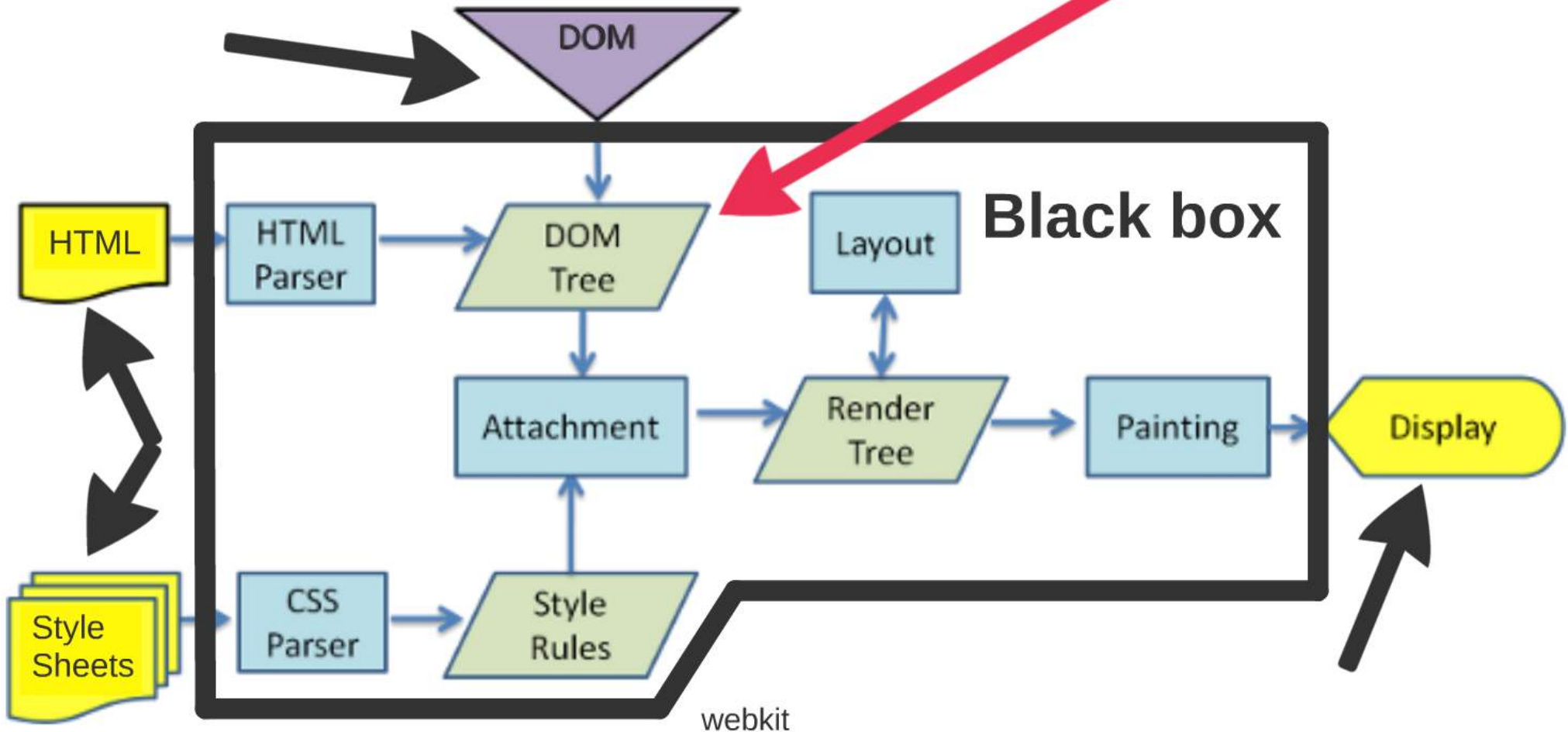


Possible to use the information of a node.

Possible to overwrite the information in a node!

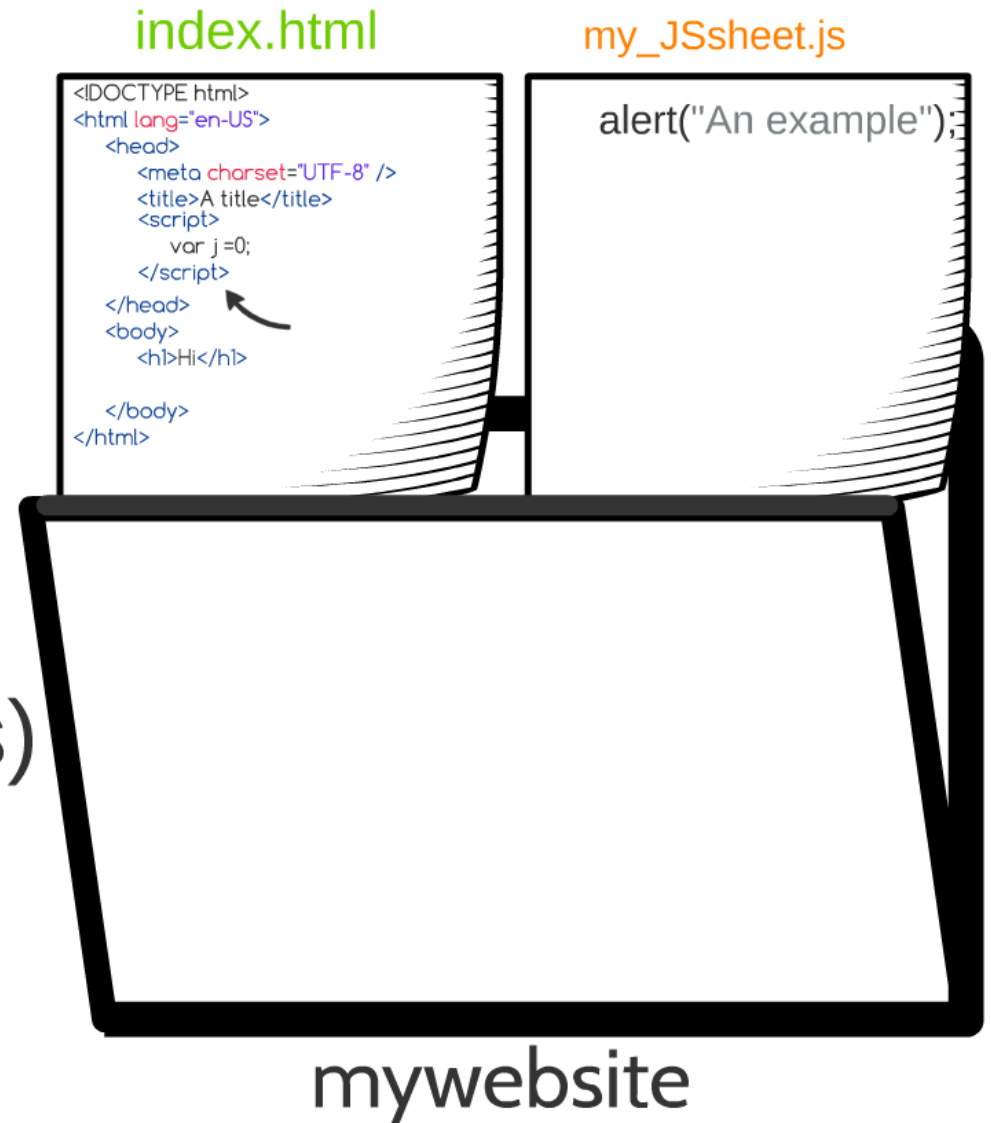
What does a browser do?

Browsers are able to parse HTML (along with other things like CSS and JavaScript).



3 "levels" of JavaScript

- external
- "inline"
- inline
(inside HTML elements)



Inline JavaScript

JavaScript embedded in HTML elements is to be avoided

```
<a href="#" onclick="alert('Hi')">Click Me</a>
```



- Difficult to read.
- Heavier HTML code.
- Lack of caching.
- Poor accessibility.
- Difficult code maintenance.

Getting your system running

Want to get your code running and doing stuff?

Make a HTML page.

HTML element: `<script> </script>`

Sometimes given attribute: `type="text/javascript"`

This is okay

Sometimes given attribute: `language="JavaScript"`

Not okay

Your standard code can go within script tags. No boilerplate it required.

Interpreted by the browser

Your console is found easily using "inspect element"

Hello world

Finally, amiright?

```
<script>    dynamic  
    var t = 5;
```

```
    console.log("Hello world" + t);
```

```
</script>
```

String concatenation

Hello world5

t (above) is a double

All numbers in JavaScript are doubles.

True and false

Many different things can equal **true** and **false**

```
console.log(1 == '1');
```

```
console.log(1 == [1]);
```

true

true

An empty string is false,
but an empty array is true.

```
var z= false;
```

```
if(z)
```

```
    console.log(z);
```

Never use == or !=

Always use === !==

NaN

NaN is only exception with ===

NaN is !== NaN

NaN means "Not a number" and is returned whenever a math function fails.


`isNaN(Math.sqrt(-1))` `true`

Equality comparisons with NaN will always yield false. This can be problematic.

isNaN() function can be used to test if something is a NaN!

Arrays

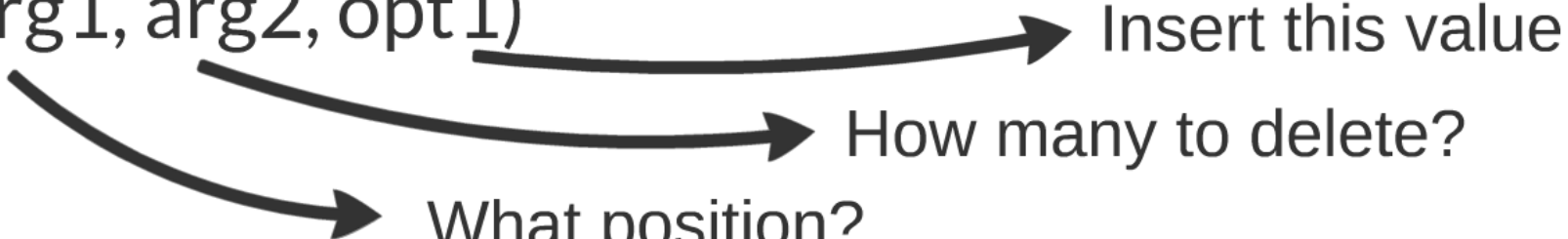
```
var dmt = ['Melzargard', 'Groribas', 'Geryuganshoop',  
'Boros'];  
var x = [10];  
var myList = [6, 21, 3];  
console.log(myList);
```

`.length` -> *length of array*

`.pop()` -> *remove from end of array*

`.push("a value")` -> *add to end of array*

`.splice(arg1, arg2, opt1)`



Insert this value

How many to delete?

What position?

Arrays II

Arrays automatically grow as additional content is added. Memory allocation is not something you ever have to worry about.

Arrays are objects, and can be indexed using the dot operator.

```
var myList = [6, 21, 3];
```

```
myList.1 === 21;
```

But wait, there's more!

The indexing of JavaScript can be messed around with.

You can treat JavaScript arrays as hashmaps.

```
car = { excellent_car: "Mercedes",  
        7: "Mazda",  
        too_hot_2_handle: "Ford Pinto"  
}
```

car[7] is the same as car.7

car.excellent_car is the same as
car["excellent_car"]

Functions

Functions exhibit a similar lack of boilerplate as seen already.

```
function thisIsTheFunction(theargument) {  
  theargument+=1;  
  return theargument;  
}
```

theargument is undefined

err... type checking?

typeof(theargument);

Note that objects, arrays, etc. (non-primitives) are passed by reference!

typeof Operator

Type	typeof Value
Undefined	undefined
Null	object
Boolean	boolean
Number	number
String	string
Array	object
Object	object

NaN

number

Hoisting && Scope

Declarations are moved to the top of the current scope by the JavaScript interpreter; meaning the top of the current function or script. All functions and variables are hoisted.

This means that functions can be called before they are declared

JavaScript does not have block level scope.

Nested blocks are always able to access variables in higher levels. YES! Seriously...

Importance of `var`. Never declare a variable without it.

```
stupid();
```

```
function stupid(){  
    function whatareyouactuallyserious(){  
        myFish.splice(2, 0, "MAGICMOONFISHY");  
    }  
    whatareyouactuallyserious();  
}
```

```
var myFish = ['cod', 'red herring', 'kipper', 'salmon'];  
console.log(myFish);  
(5) ["cod", "red herring", "MAGICMOONFISHY",  
"kipper", "salmon"]
```


Back to the DOM

As discussed already, the DOM consists of nodes.

Inserting, deleting, accessing, and editing nodes are all possible via JavaScript.

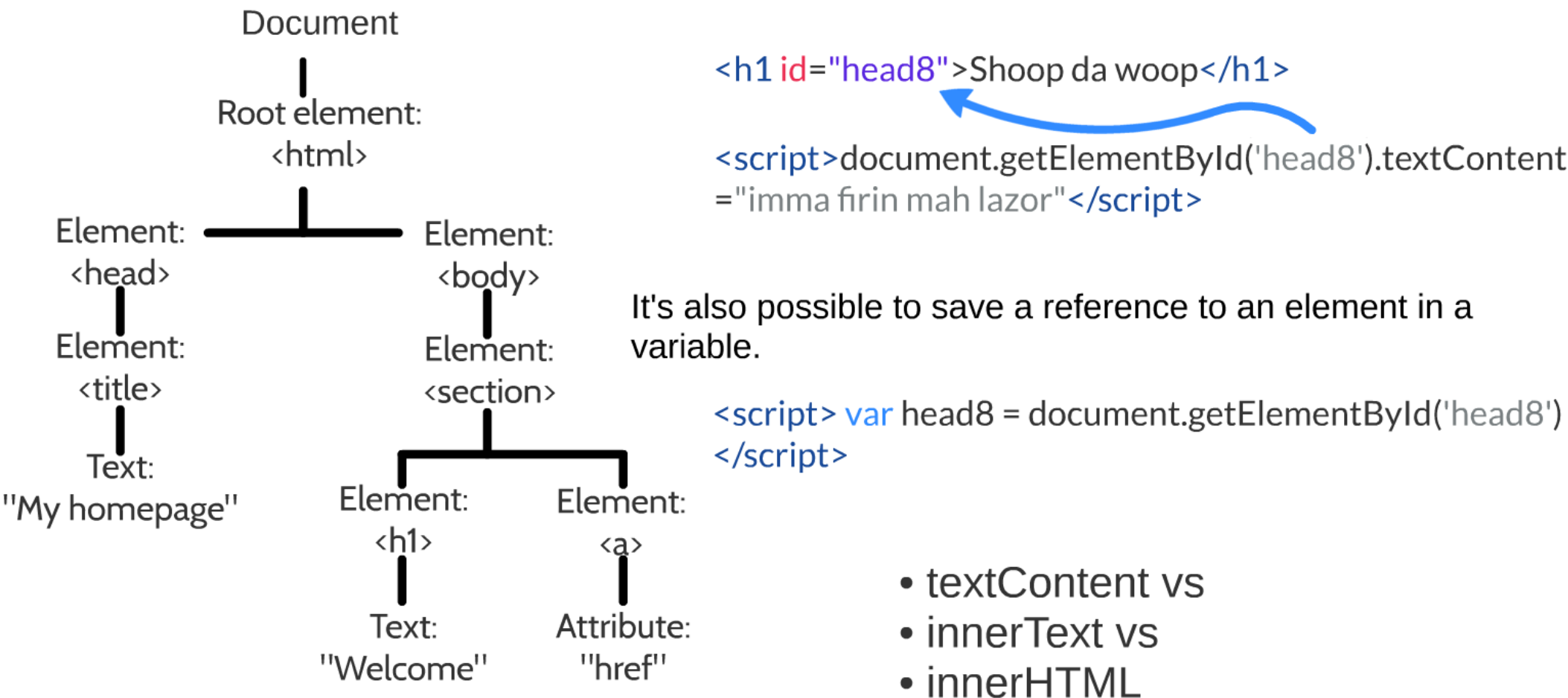
JavaScript can't access DOMs outside its own window (or tab) in the browser.

There are lots and lots of methods and properties associated with the DOM, including some experimental ones.

We'll be looking at some of the more basic ones

Back to the DOM

Valid text and attribute nodes are attached to html elements on the DOM.



Loading the script

JavaScript is parsed and executed as it is encountered. Traditionally no other action can take place as this is occurring, including the parsing of HTML and the construction of the DOM.

This means that a JavaScript script can refer to parts of the page that it does not yet know about.

It will fail, as the script will assume that the element being searched for does not exist. One solution is to put any JavaScript code at the bottom of the page.

Alternatively you can test to ensure the page has been fully loaded before execution using [window.onload](#)



```
<script>document.getElementById('head8');  
// huh? </script>
```

```
<h1 id="head8">Shoop da woop</h1>
```



More HTML tags: noscript, canvas

The `<noscript></noscript>` element has a simple purpose. It displays any HTML contained **within** it **iff** scripting is disabled.

People may have scripting disabled for increased security or to save resources.

Consider usability. Also, always have a fallback in case JavaScript fails.

canvas

The `<canvas></canvas>` tag is used to draw graphics, on the fly, via JavaScript.

`<canvas id="myCanvas">`This text displays if the user is using a potato`</canvas>`

`<script>`

`var canvas=document.getElementById('myCanvas');`

`var ctx=canvas.getContext('2d');`

`ctx.fillStyle='#FF0000';`

`ctx.fillRect(0,0,80,100);`

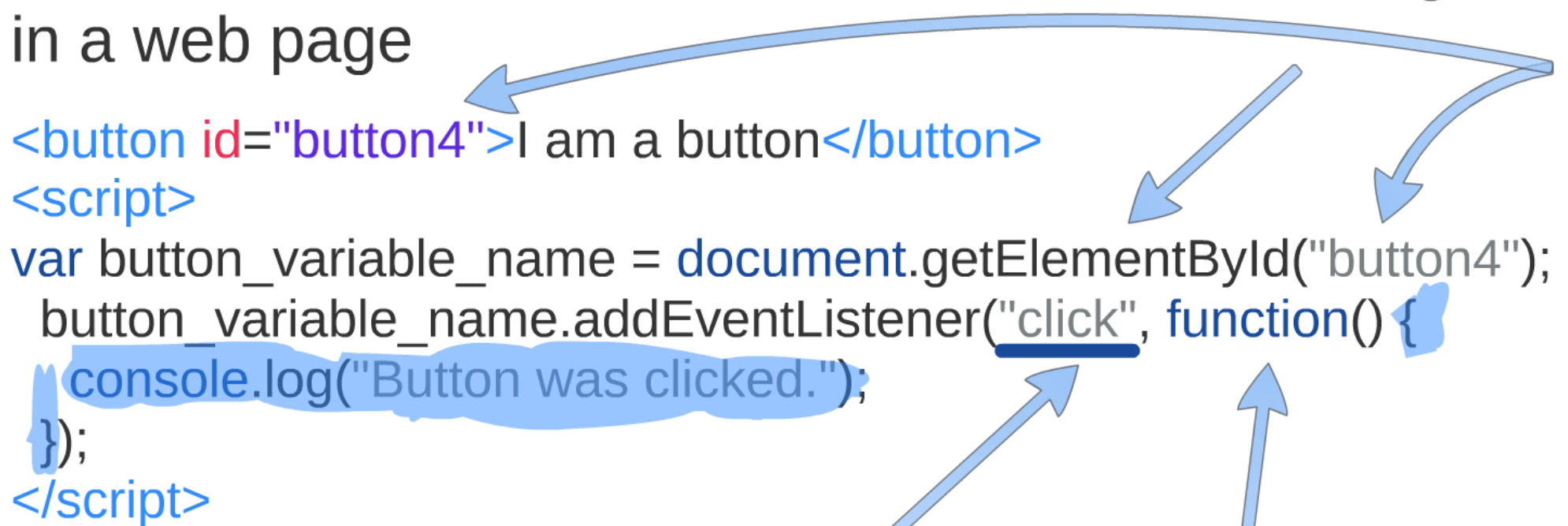
`</script>`



Event handlers

Event handlers can be created to monitor changes in a web page

```
<button id="button4">I am a button</button>
<script>
var button_variable_name = document.getElementById("button4");
button_variable_name.addEventListener("click", function() {
  console.log("Button was clicked.");
});
</script>
```



copy
load
change
mouseover
play

callback is a function that
you want to call when an
event is triggered

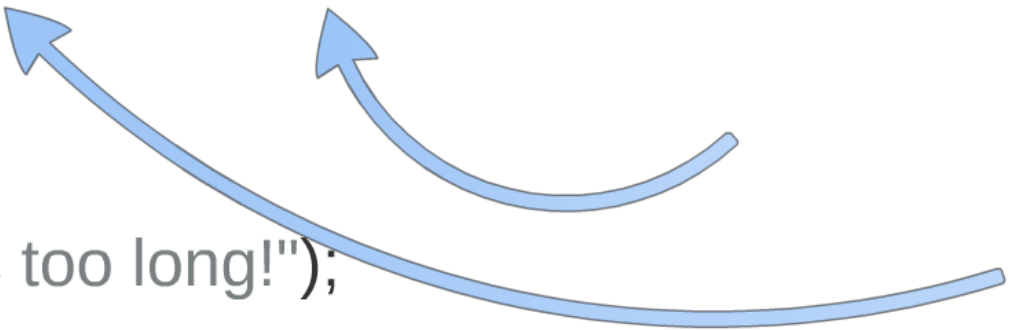
Form validation

Event handlers can give direction to users

```
<form>
<input id="firstname" type="text"/>
<input type="submit"/> </form>
<script>
var first_name = document.getElementById("firstname");
first_name.addEventListener("change", howLong);

function howLong() {
    var p = this.value.length;
    if (p>8) alert("hey, your name is too long!");
}

</script>
```



The diagram consists of two blue curved arrows. The first arrow starts from the `howLong` function and points to the `addEventListener` method in the `first_name` variable assignment. The second arrow starts from the `addEventListener` method and points to the `id="firstname"` attribute in the `<input>` tag, illustrating the connection between the event handler and the specific form element it monitors.

Form validation

Never **depend** on JavaScript for form validation.

JavaScript can be disabled, or rewritten!

But it is still much better to use client side scripting to give **information** to users.

- Updates, dynamically.
- Less frustrating for users
- Less resource use for servers

Libraries and frameworks

Development originally spurred by aim to promote cross-compatibility between browsers.

Remember: JavaScript only exists as code that has to be executed by browsers.

Later development focused on principle of "do more, with less code"



Maintainability, power, features, and subjective "feel" separates different frameworks.