

The Web (JS)

Interface Pessoa-Máquina - 25/26 - LEI / UM

Hugo Pacheco

hpacheco@di.uminho.pt

The Web: uma tríade atual

Tipografia, cores,
layout, efeitos
visuais, animação



Hierarquia, navegação,
significado, conteúdo

Interação e automação;
linguagem general-
purpose que corre no
cliente

HTML: interação limitada

- **HTML** de uma página web pode efetuar HTTP requests adicionais para:
 - **CSS:** `<link href="presentation.css" rel="stylesheet">`
 - **JS:** `<script src="behavior.js"></script>`
 - **Media:**
 - ``
 - `<video src="nyan-cat.mp4"></video>`
 - `<audio src="o-superman.mp3"></audio>`
 - **Outras páginas HTML:** `<iframe src="otherpage.html"></iframe>`
- ! Mas só o pode declarar estaticamente, executado “on page load”

JavaScript (JS)

- A única linguagem de programação que corre no browser **do lado do cliente**
- Qual o propósito?
 - **Interação:** “When click on button, validate the form and display errors”
 - **Automação:** “Add a `<time>` element inside every `<td>` in the schedule”
 - **APIs:**
 - **Externas (Web):** “Display latest Instagram posts in the sidebar”
 - **Internas (Browser):** “Get the geolocation of the mobile device”
 - ...
- Uma implementação do standard ECMAScript (ES)

JS

JavaScript = Java + Script?

- Criada em 1995 por Brendan Eich @ Netscape
- Bastante diferente de **Java**, mas alguma sintaxe similar
 - “I was under marketing orders to **make it look like Java** but not make it too big for its britches. It’s just this sort of silly little brother language, right?”
- Uma **scripting** language interpretada pelo browser
 - “When it started getting early traction, [...] I was a little surprised. But I wasn’t shocked, because on reflection what happened was **the people that wrote the web page could now program** the web page.”



JavaScript (JS)

- Uma general-purpose language, desenvolvida para ter...

- ...the functional programming of **Scheme**



- ...the prototype-based dynamic object-oriented programming of **Self**



- ...the syntax and appearance of **Java**

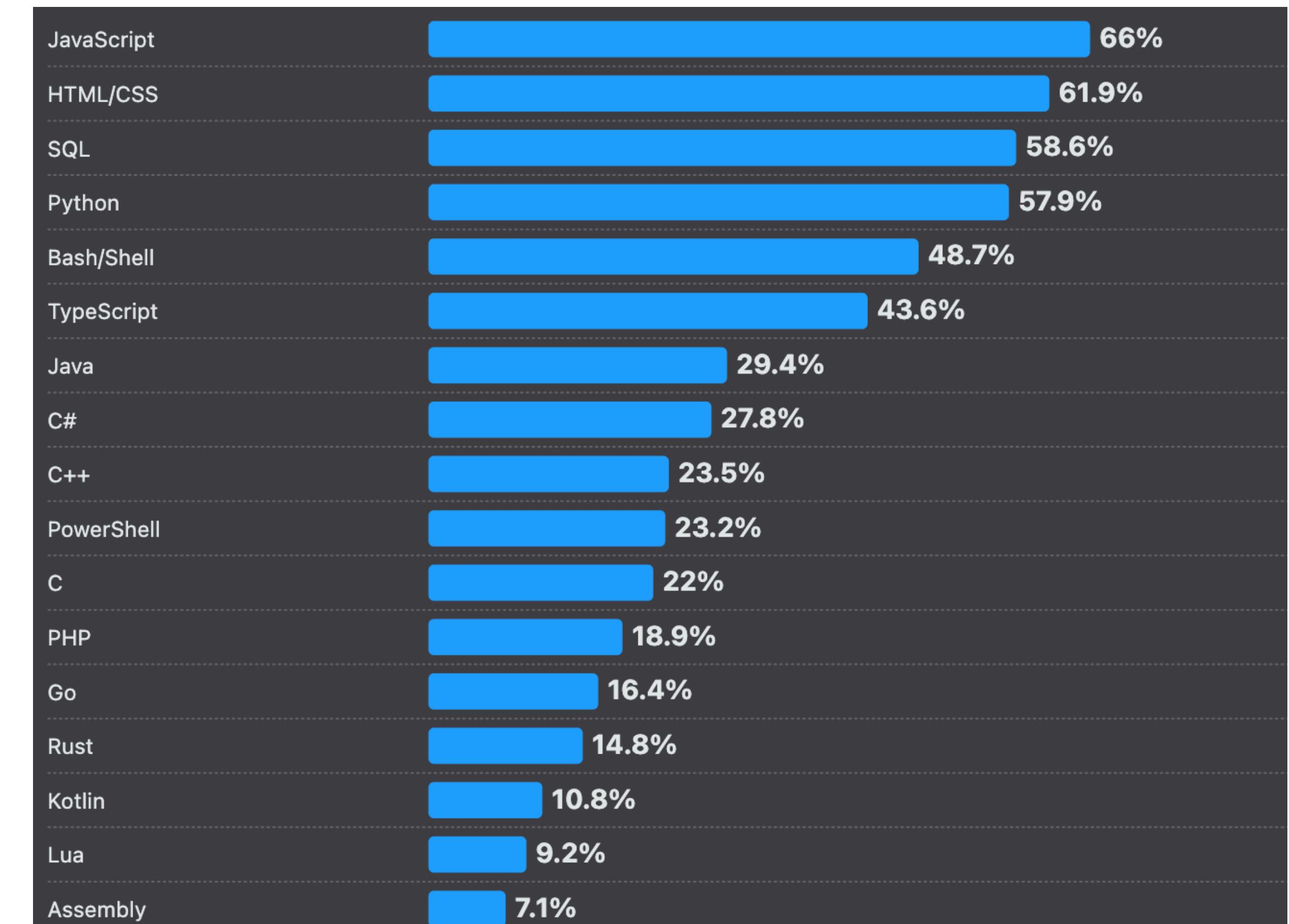


- Famosamente criada em apenas 10 dias!



“Ten Days That Shook the World” Wide Web

- “**Back then**, the pace of Web innovation was furious [...]”
- **Hoje**, a linguagem mais popular do mundo



JS: uma revolução ~~de usabilidade~~?

- “One of the notorious **first in-house testers** was, ‘I’d like to compare a number to a string that contains that numeral. And I don’t want to have to change my code to convert the string to a number, or the number to a string. I just want it to work.’”
 - “And I did it. And that’s a **big regret**, because that breaks an important mathematical property [...] It led to the addition of a second kind of equality operator when we standardized JavaScript.”
 - <https://dorey.github.io/JavaScript-Equality-Table/>
- ! Grupo não heterogéneo de utilizadores durante prototipagem
- ! Sem tempo para avaliar a solução

JS: uma revolução ~~de usabilidade~~?

- “We all know that JavaScript is quite a funny language with tricky parts. Some of them can quickly turn our everyday job into hell, and some of them can make us laugh out loud.”
 - <https://github.com/denysdovhan/wtfjs>
- “JavaScript's popularity is almost completely independent of its qualities as a programming language” (Douglas Crockford, JavaScript: The Good Parts)
 - ! Não é a linguagem com melhor usabilidade para web developers

- $4 * "2" ** 3 + 10 = 4 * 8 + 10 = 32 + 10 = 42$
- $4 * 2 ** 3 + "10" = 4 * 8 + "10" = 32 + "10" = "3210"$
- $[] == ![] = true$



permanente

JS: uma revolução de funcionalidade

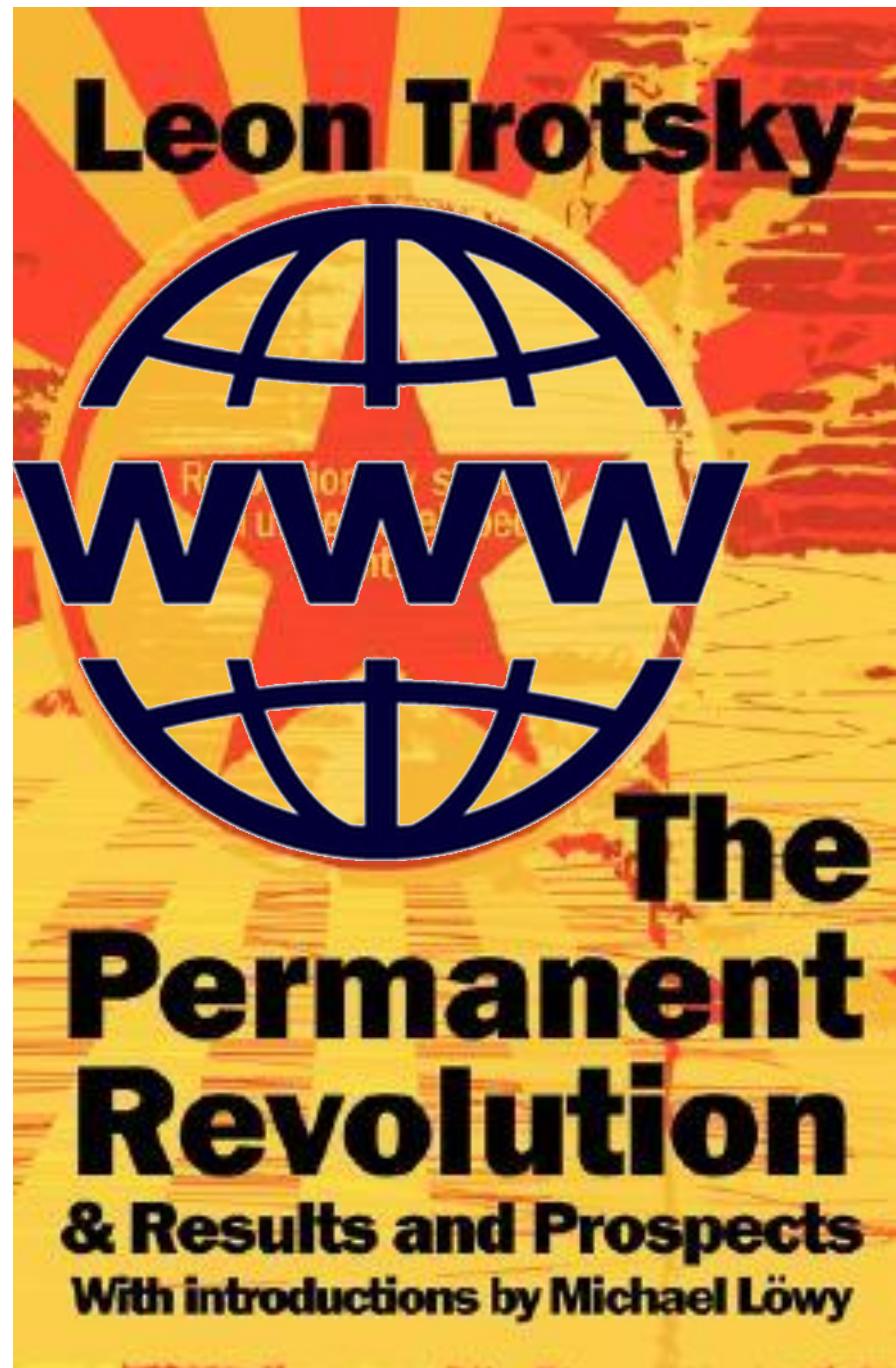
- **Backward Compatibility**

Erros ou imperfeições antigas são mantidos para que sites antigos não deixem de funcionar.

- “Once something is released into the wild, bugs or imperfections quickly become essential features and are **nearly impossible to change.**”
(Brendan Eich)
- “Don’t break the web” (ECMAScript’s TC39)

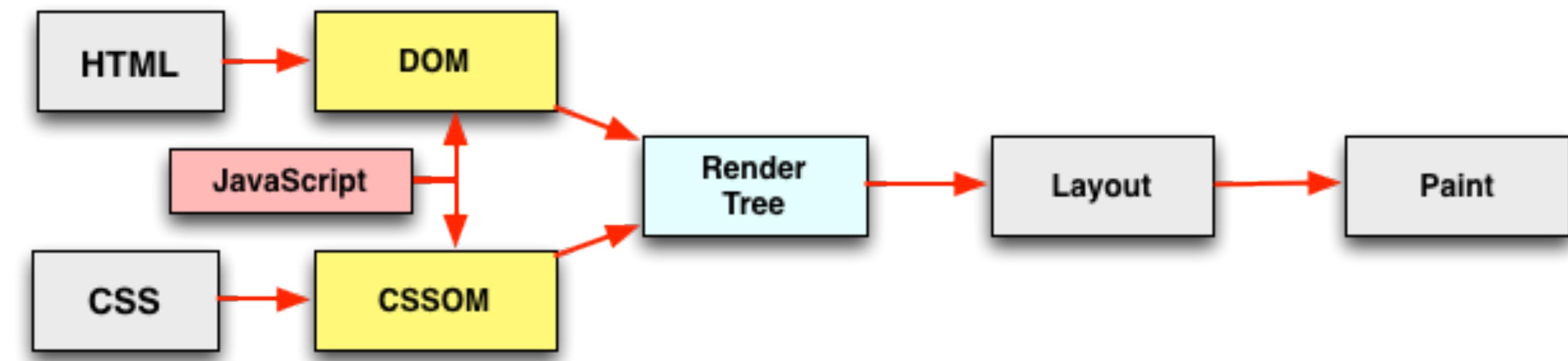
- **Essential JS subset for JS-only tasks**

- “There is a beautiful, elegant, highly expressive language that is buried under a steaming pile of good intentions and blunders. [...] JavaScript is a block of marble, and I **chip away the features that are not beautiful** until the language’s true nature reveals itself.” (Douglas Crockford, *JavaScript: The Good Parts*)
- “JavaScript should only do what only JavaScript can do.” (Jeremy Keith)



Arquitectura de uma página no browser

- Conteúdo da página web guardado no DOM
- Browser faz o rendering do DOM (considerando HTML + CSS)
- JS consegue manipular (entre muitos outros):
 - DOM: querying and manipulating the DOM tree, events
 - CSSOM: reading and manipulating stylesheets
 - Fetch: Sending HTTP requests



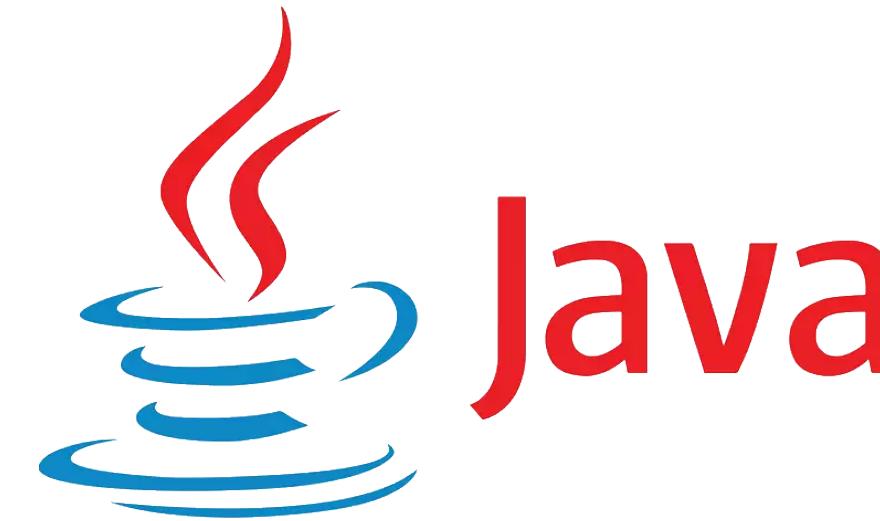
Exemplo: JS minimal

```
<script>
    alert('Hello world!');
</script>
```

1. Abrir Developer Tools > Console
2. Inspecionar o ambiente da página atual (diferente para cada window/tab)
3. E modificá-lo (variáveis globais no código, logging, ...):

```
myName = "Hugo";
console.log(myName);
```

JS: aparência de



- Classes, getters/setters, métodos, encapsulamento, ...

```
class Game {  
    #name; #year;  
    constructor() {  
        this.#name = "test";  
        this.#year = 2025;  
    }  
    get name() {  
        return this.#name;  
    }  
    set name(v) {  
        this.#name = v;  
    }  
    toString() {  
        return "Games(" + this.#name + ", " + this.#year + ")";  
    }  
}
```

JS: sintaxe

- “JavaScript borrows most of its syntax from Java, C, and C++, but it has also been influenced by Awk, Perl, and Python.”

```
// Declare variables
let x = 10;
const y = 20;
z = 30;

// Some types
let dqstr = "Hello"; // string
let sqstr = 'Hello'; // string
let num = 42; // number
let fpnum = 42.5; // number
let bool = true; // boolean
let person = {name: "Alice", age: 25}; // object
let arr = [1, 2, 3]; // array
let undef; // undefined
let n = null; // object

// Conditionals
if (x > 5) {
    console.log("Greater");
} else {
    console.log("Smaller");
}

switch (x) {
    case 10: console.log("Ten"); break;
    case 20: console.log("Twenty"); break;
    default: console.log("Unknown");
}

// Loops
let nums = [10, 20, 30];
for (let i = 0; i < nums.length; i++) {
    console.log(nums[i]);
}

nums.forEach(num => console.log(num));
for (let key in person) {
    console.log(key + ": " + person[key]);
}
```

JS: objetos de

- Orientação a objetos baseada em *prototypes*
 - Objetos herdam diretamente de outros objetos
 - **Classes são apenas açúcar sintáctico**
 - Métodos e atributos são funções partilhadas pela *prototype chain*, não são copiados entre classes
 - Possível criar novos objetos sem classes
 - Possível adicionar dinamicamente novos métodos e atributos
- + Abstração, modularidade, isolamento

```
let o = {  
    name: "José",  
    dizOlá: function()  
    { console.log("Olá  
"+this.name); }  
};  
  
let oo =  
Object.create(o);  
console.log(oo);  
oo.dizAdeus = function()  
{ console.log("Adeus  
"+this.name); };  
  
oo.dizAdeus();
```

JS: tipos de

- Tipagem dinâmica, como em Python
 - **Tipos associados a valores, não a variáveis:** uma variável pode ter valores de tipos diferentes em instantes diferentes
 - Variáveis (globais ao script por definição) vs let (no contexto de um bloco) vs const (não podem ser redefinidas)

```
let x = 10; // x is a number
x = "Hello"; // x is a string
x = true; // x is a boolean
```
 - **Conversões implícitas de tipos**
 - Em muito similar a Python (+, *, ...), mas ainda mais imprevisível
 - Conversão para boolean de *falsy* (`false`, `0`, `''`, `null`, `undefined`) ou *truthy values*
 - Propriedades não existentes de objetos retornam simplesmente `undefined`
 - Flexibilidade: reporta menos erros, não exige anotações de tipos

JS: funções de λ

- Suporte de *first-class functions*
 - **Funções são objetos**
 - Podem ser criadas e manipuladas dinamicamente
 - Podem ser passadas como argumentos ou o retorno de outras funções
- + Abstração, modularidade
- + Callbacks: código executado pelo browser
- + Execução assíncrona: *events, promises*

```
// Regular function
function add(a, b) {
  return a + b;
}

// Anonymous function
const add = function (a, b) {
  return a + b;
}

// Arrow function
const add = (a, b) => a + b;
```

JS: modelo de execução

- A linguagem é *single-threaded*; o browser controla a *thread* de execução
- HTML declara um conjunto de scripts
- Scripts executados por ordem de declaração, “on page load”

```
<script src="lib.js">  
<script>console.log("Hello");</script>
```

- Tag defer atira o script para o fim da ordem, quando a página acabar de carregar (útil se e.g. depender de elementos do DOM)

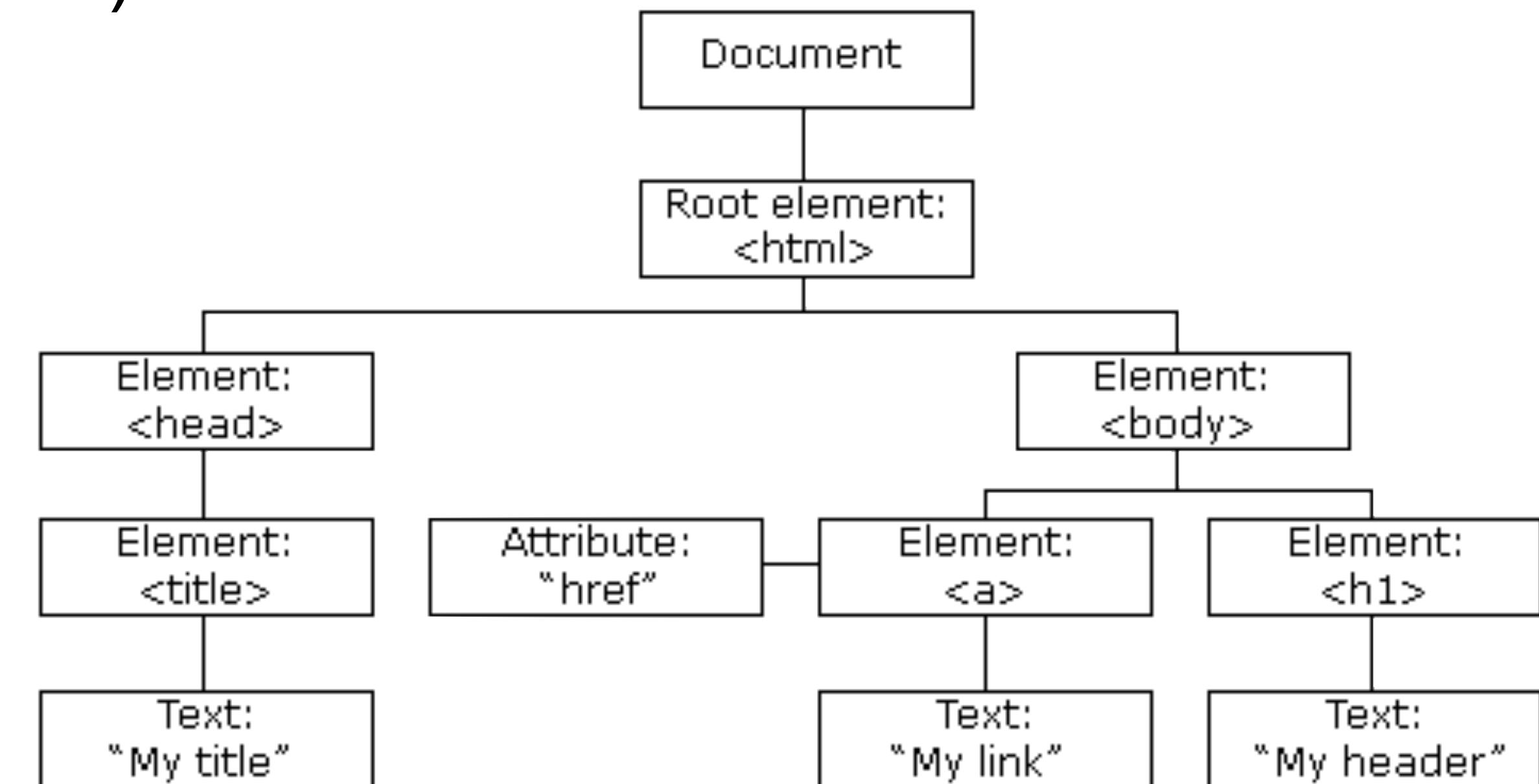
```
<script defer src="handler.js">
```

- Tag async executa o script em qualquer ordem, depois de outros scripts (útil para acelerar carregamento da página)

```
<script async src="pretty.js">
```

JS: ambiente de execução

- Scripts de uma página correm num ambiente definido para a janela
- Dado pela variável global `window` (atributos são também vistos como variáveis globais: `window.attr ≡ attr`)
 - Contém informação sobre a janela (`innerWidth`, `innerHeight`, ...)
 - Contém a raiz da DOM tree (`document`)
 - Cada elemento da página é um objeto na DOM tree



JS: ambiente de execução

- Identificadores de elementos criam variáveis globais

```
<button id="submit">Submit</button>
```

```
console.log(submit);  
console.log(window.submit);  
console.log(window['submit']);
```

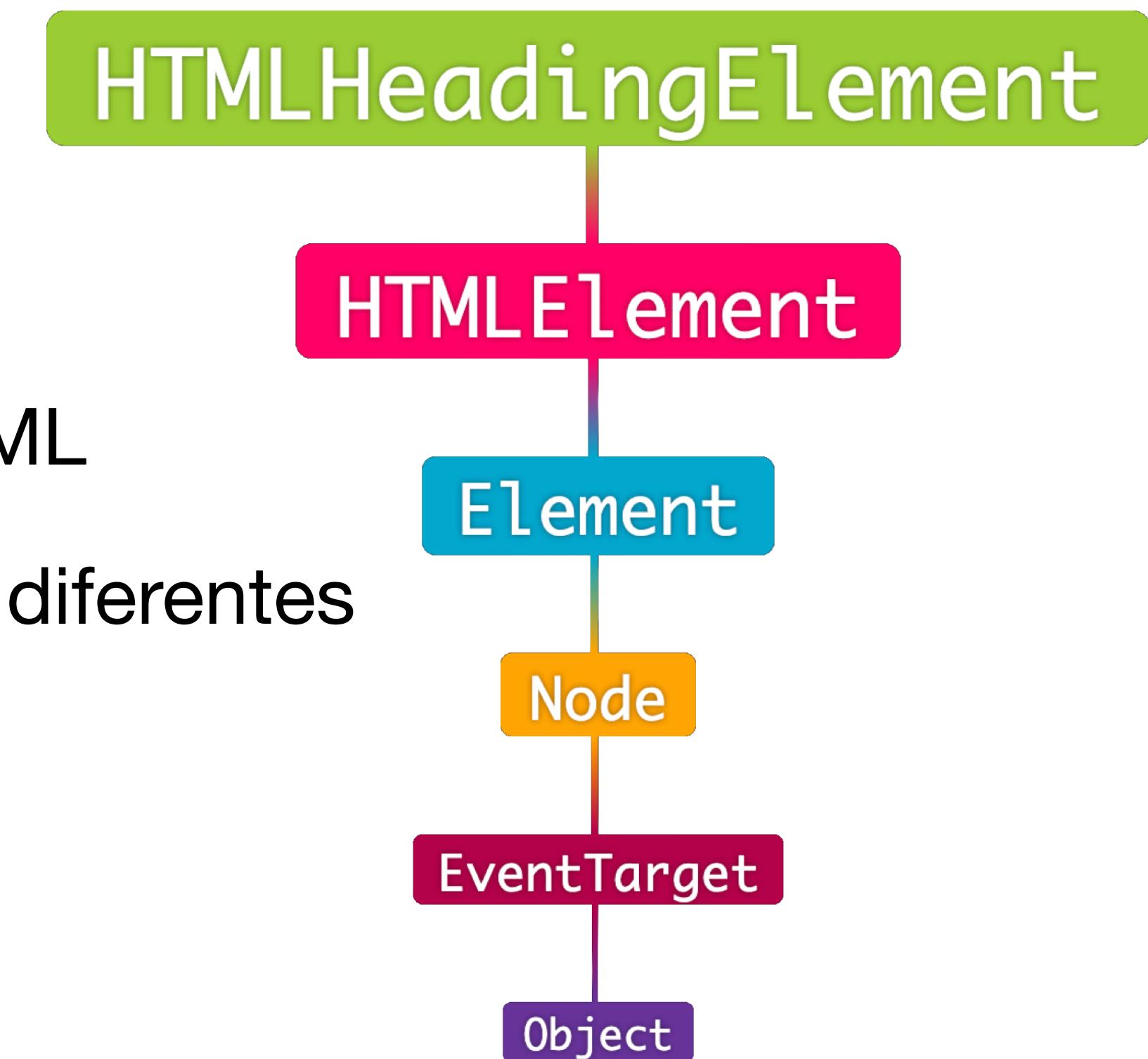
- O que acontece neste caso?

```
<button id="submit">Submit1</button>  
<button id="submit">Submit2</button>
```

- Variável `submit` torna-se uma `HTMLCollection`

JS: DOM tree

- Cada objeto de cada elemento HTML tem uma classe específica
 - `HTMLDocument`: o HTML document da raiz
 - `HTMLHeadingElement`: cabeçalhos (`<h1>`, `<h2>`, ...)
 - `HTMLImageElement`: imagens (``)
 - `HTMLElement`: partilhada por todos os elementos HTML
- Objetos herdam hierarquia de classes com propriedades diferentes
 - E.g., `Element`: `parentNode`, `appendChild()`, `removeChild()`
 - E.g., `Node`: `innerHTML`



JS: uma linguagem orientada a eventos

- **Events** sinalizam um acontecimento que acabou de ocorrer
- Scripts podem registar **event handlers**: código executado assincronamente pelo browser quando um evento acontece
- Duas classes de eventos
 - **Browser**: página acabou de carregar, redimensionamento da janela, mudança na conexão de rede
 - **User**: click num botão, movimento do rato
- ! Eventos não são despoletados nem processados instantaneamente
 - ! São colocados numa *event queue* e potencialmente *coalesced* ou limitados a *framerate* controlada pelo browser

JS: Event object

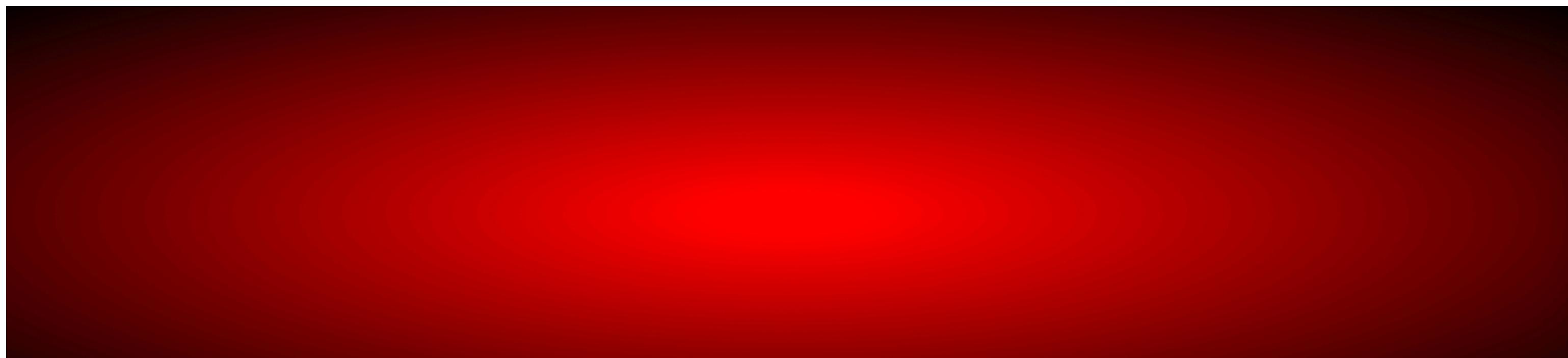
- Event handlers recebem um objeto com o evento associado
- Event object fornece metadados associados ao evento em causa:
 - target element (para o qual o evento foi disparado)
 - mouse position
 - mouse button
 - keyboard modifiers
 - keys pressed
 - ...

JS: User events

- Exemplo de um user event (mouse) que manipula o DOM

```
//CSS
body {
  background-repeat: no-repeat;
  background-size: 100% 100%;
  background-attachment: fixed;
  background-color: red;
  background-image: radial-gradient(
    at calc(var(--mouse-x, .5) * 100%)
    calc(var(--mouse-y, .5) * 100%) ,
    transparent, black
  );
}
```

```
// JS
document.addEventListener("mousemove", evt => {
  let x = evt.x / innerWidth;
  let y = evt.y / innerHeight;
  let root = document.documentElement;
  root.style.setProperty("--mouse-x", x);
  root.style.setProperty("--mouse-y", y);
});
```



JS: User events (primitive)

- Key pressed (`keydown`) or released (`keyup`)
- Mouse moved (`mousemove`)
- Mouse button pressed (`mousedown`) or released (`mouseup`)
- Mouse enters (`mouseover`) or leaves (`mouseout`) an element
- Element gained focus (`focus`), lost focus (`blur`), or was modified (`change`)
- Form was submitted (`submit`)
- ...

JS: User events (composite)

- Eventos compostos que combinam eventos primitivos
 - Mais complexos do que parecem
 - Usar preferencialmente aos eventos primitivos
- Clicking (`click`)
- Double-clicking (`dblclick`)
- Character held down (`keypress`)
- Form element value changed (`input`)
- Entering (`mouseenter`) or exiting (`mouseleave`) an object's bounding box

JS: Browser events

- Eventos para além de interação com a UI
 - `HTMLElement`:
 - `error`: loading de recurso falhou
 - `load`: loading de recurso terminou com sucesso
 - `Window`:
 - `online`: browser ganhou acesso à rede
 - `offline`: browser perdeu acesso à rede
 - `message`: nova mensagem colocada na consola, e.g. usando `postMessage()`
 - `close`: janela fechada
 - `load`: loading da página completa terminou

Exemplo (Animação)

- Podemos criar uma animação usando JS (<https://tgdwyer.github.io/html/>)
 1. Criando um rectângulo em SVG (gráficos declarativos como tags **HTML**)

```
<svg><rect id="rect" x="10" y="5" width="20"  
height="20"/></svg>
```

2. Atualizando os seus atributos para mover o rectângulo
 1. Definindo um ciclo que incrementa uma variável **x** com uma dada *framerate*
 2. Atualizando o atributo do rectângulo com o valor de **x**

```
rect.setAttribute('x', x);
```

Exemplo (Animação)

- Podemos criar a mesma animação usando CSS

? Quais as vantagens ou desvantagens?

+ Bastante mais simples

+ Declarativo

```
#rect {  
    animation-name: moveX;  
}  
  
@keyframes moveX {  
    0% { x: 0; }  
    100% { x: calc(100vw - 40px); }  
}
```

The blue rectangle is animated declaratively through CSS.



The red rectangle is animated by a custom imperative JavaScript program.



Rule of Thumb



- “JavaScript should only do what only JavaScript can do”
- **HTML + CSS**
 - “Declarative approaches can feel like giving up control: I’m going to leave the details to the web browser.”
- **JS**
 - “But I feel like this control comes at a cost. I’m not sure you really gain control. I think you get the feeling of control. But it always involves making assumptions, whereas a declarative mindset is about avoiding assumptions.”

(Jeremy Keith)

No geral

- 💡 **Promover a separação de conceitos**

- 💡 Evitar manipular elementos **HTML** a partir de **JS**

- ❗ Por detrás de várias vulnerabilidades de segurança Web, e.g. XSS

- Se possível, manter a estrutura **HTML**
 - Apenas modificar valores, e.g., **textContent** em vez de **innerHTML**
 - Apenas alterar **visibility** de elementos com **JS**

- 💡 Evitar manipular propriedades **CSS** a partir de **JS**

- Se for uma questão de estilo estático, organizar com classes **CSS**
 - Se for uma questão de estilo dinâmico, utilizar variáveis **CSS**

Exemplo (click 😊)

- Se o botão for clickado, muda o seu conteúdo textual para “Thank you 😊”

```
<button id=clickme>Click me</button>

clickme.addEventListener("click", function (event)
{
    event.target.textContent = "Thank you 😊";
});
```

- ? Igual ao seguinte?

```
<button onclick="this.textContent = 'Thank you 😊'">Click me</button>
```

- ✓ Sim, formulação mais simples
- ! Diferenças em quando o *event handler* é registrado

Exemplo (click 😊)

? Solução HTML melhor?

- ✖ Não, mistura comportamento com estrutura
- Algumas limitações comportamentais
 - Só permite registar um *event handler* de cada tipo
 - Não permite especificar características do *event handler*

```
dom.addEventListener(type, handler, options)
```

```
options = { capture : bool, once : bool, passive :  
bool }
```

Exemplo (click 🦄)

- Novamente, um botão com um *event handler* “on click”

```
<button id=button>Click <mark>me</mark>! !</button>  
  
button.addEventListener("click", evt => {  
    evt.target.innerHTML += "🦄";  
});
```

Carregamento de Scripts:

Padrão: Executados por ordem de declaração.

Defer: Executa apenas no fim, após o carregamento da página (útil quando depende do DOM).

Async: Executa de forma assíncrona, sem ordem fixa (útil para performance/scripts externos).

- Mesmo comportamento que o exemplo anterior?

- ✗ Não, há dois comportamentos possíveis para um só evento!
- ✗ De quem é a culpa?

Bug

Browser

JavaScript

innerHTML

JS: Event processing

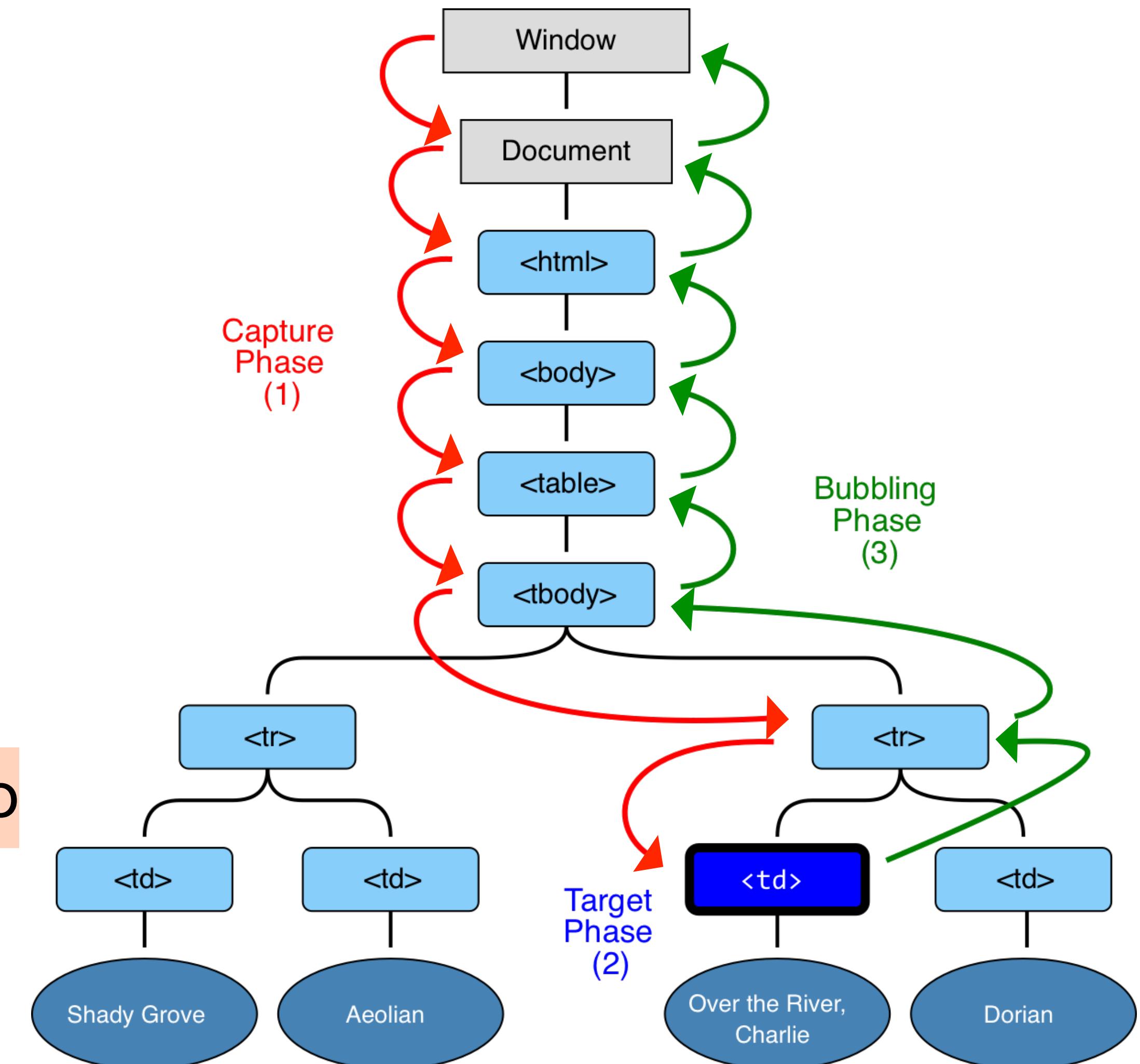
1. **Capture:** Evento é disparado para a janela e propagado, descendo a DOM tree até ao target element

2. **Target:** Evento atinge o target element

3. **Bubbling:** Evento sobe a DOM tree, executando event handlers para cada elemento

💡 Variável `target` guarda o elemento atual

💡 Variável `currentTarget` guarda o elemento para o qual o event handler foi registrado



Exemplo (click

- Agora utilizando `currentTarget`

```
<button id=button>Click <mark>me</mark>! !</button>

button.addEventListener("click", evt => {
    evt.currentTarget.innerHTML += "🦄";
});
```

- ?
- Qual o comportamento esperado?

- ✓ Só há um comportamento para um evento: alterar o botão
- ! Notem que o evento é na mesma processado sobre o elemento `mark`

Exemplo (click 🦄)

- Considerem agora outro tipo de evento (rato dentro de elemento)

```
<button id=button>Click <mark>me</mark>! !</button>  
:hover {background: red;}  
  
button.addEventListener("mouseenter", evt => {  
    evt.currentTarget.innerHTML += "🦄"; });
```

- ? Qual o comportamento esperado?

- ✗ O elemento `mark` está *hovered* no **CSS** mas evento **JS** não reage?
- ? E se trocarmos `mouseenter` por `mouseover`? ([link](#))
- ✓ *Hovered effects* de **CSS** e **JS** consistentes

JS: Event processing

- 💡 Por defeito, nem todos os eventos são sujeitos a *bubbling*
- 💡 Sem *bubbling*, event handler é apenas disparado para o elemento para o qual foi registado
 - Em termos de comportamento: `target = currentTarget`

Sem bubbling	Com bubbling
<code>focus</code>	<code>focusin</code>
<code>blur</code>	<code>focusout</code>
<code>mouseenter</code>	<code>mouseover</code>
<code>mouseleave</code>	<code>mouseout</code>
<code>load</code>	
<code>error</code>	

Exemplo (Form validation)

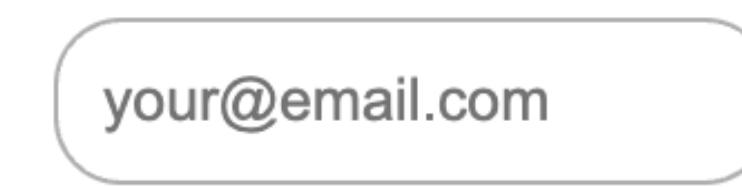
- **HTML** oferece algumas funcionalidades para validação de forms
 - Form fields podem ter input types diferentes (**text, email, url, date, ...**)
 - Form fields podem usar atributos (**placeholder, required, minlength, maxlength, ...**)
 - **CSS** oferece pseudo-class :**invalid**
 - Permite visualizar campos inválidos de forma distinta
 - Validação efetuada “on submit”
 - Browser reporta mensagens de erro
-
- The image contains three screenshots of a mobile application's sign-in screen. The first screenshot shows an email input field with the placeholder "your@email.com". A red error message above the field reads "Email must have pattern string@string.string". The second screenshot shows a password input field with the placeholder "Password". A red error message above the field reads "Password must be at least 6 characters". The third screenshot shows a "Sign in" button with a red outline, indicating it is disabled or invalid due to the previous errors.

DOM tree

- Elementos **HTML** na DOM tree são objetos ⇒ **têm estado interno!**
 - Registar event handlers
 - Ao remover um elemento, removem-se todos os seus event handlers
 - Guardar meta-information de validação (form fields)
 - Atributo `validity` : `ValidityState` contém estado de validação detalhado
 - `checkValidity()`: retorna um booleano com validade
 - `reportValidity()`: reporta validade ao utilizador
 - `setCustomValidity(message)`: corre validação definida pelo utilizador, string não vazia = inválido
 - ...

Exemplo (Form validation)

- Separação de conceitos
- Validação pode ser toda efetuada do lado do **JS** ⇒ mais flexibilidade
 - Implementar lógica de validação arbitrária (e.g., combinar múltiplos campos)
 - Integração e aspetto nativo utilizando a Constraint Validation API
- Para comportamentos simples, tradeoff entre modularidade e conveniência
 - ! Mais tarde, ao utilizar web frameworks, estado interno do **HTML** pode entrar em conflito com estado da framework ⇒ validação é um exemplo clássico



Exemplo (Calculadora)

- Queremos construir uma calculadora bidirecional
 - Alterar número → Atualizar soma
 - Atualizar números (divide por 2) ← Alterar soma

2 + 3.5 = 5.5 Equalize

? Precisamos do botão “Equalize”?

- Apenas se não quisermos fazer retype do número do lado direito

Exemplo (Semaforo)

- Queremos construir um semáforo
 - Tem 3 cores (**verde**, **amarelo**, **vermelho**), apenas uma ativa
 - Botão “Mudar” avança o semáforo
 - **verde** → **amarelo** → **vermelho** → **verde**
 - Botão “Modo teste” ativa/desativa animação automática
 - Semáforo avança a cada 1 segundo
 - Botão “Mudar” inativo se “Modo teste” ativo

