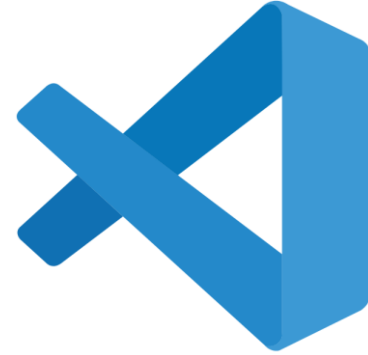


Bootcamp Full Stack – Módulo 01 – Slides

- <https://code.visualstudio.com/>
- Principais características:
 - Principal editor de código utilizado nos dias atuais.
 - Excelente suporte ao git.
 - Terminal de comandos integrado.
- Acompanhe o professor:
 - Instalação da ferramenta.
 - Instalação de extensões.
 - Configuração essencial.
 - Dicas de utilização.



☑ Visual Studio Code:

- Mais conhecido como VSCode.
- Principal editor de código JavaScript nos dias atuais.
- Multiplataforma.
- Excelente suporte ao git.
- Excelente conjunto de ferramentas e extensões.
- Suporte ao Emmet.
- Terminal integrado.

- <https://nodejs.org/>
- Ecosystema de desenvolvimento JavaScript.
- Permite, por exemplo:
 - Criação de scripts (CLI – Command Line Interface).
 - Manipulação de arquivos e pastas.
 - Criação de servidores web e API's (Back End).
 - Interação com Bancos de Dados.
 - Instalação de pacotes e bibliotecas através do NPM (Node Package Manager).
- Acompanhe o professor:
 - Instalação da ferramenta.
 - Testes de execução.



☑ Node.js:

- Ecossistema para desenvolvimento e execução de código JavaScript fora da web (computadores pessoais, servidores, etc.).
- Utilizado como CLI de diversas ferramentas (Angular, React, Vue, etc.).
- Utilizado para a criação de servidores web.



- Node Package Manager.
- Gerenciador de pacotes do Node.js.
- Possui também um site que é, na verdade, um repositório de pacotes.
- <https://www.npmjs.com/>
- Ferramenta de linha de comando.
- Pré-requisito: Node.js instalado e configurado.
- Instalação **padrão** de pacotes:
 - `npm install nome_do_pacote`
- Instalação **global** de pacotes:
 - `npm install -g nome_do_pacote`

Live-server

A simple development http server with live reload capability.

- <http://tapiov.net/live-server/>
- Biblioteca para o Node.js, instalável via NPM.
 - Comando: `npm install -g live-server`
- Cria um servidor web simples rapidamente.
- Muito útil para desenvolvimento.
- Auto refresh.
- Acompanhe o professor:
 - Instalação da ferramenta.
 - Testes.

☒ NPM:

- Grande repositório de bibliotecas construídas com Node.js.
- Comando para manipulação de pacotes.

☒ Live-server:

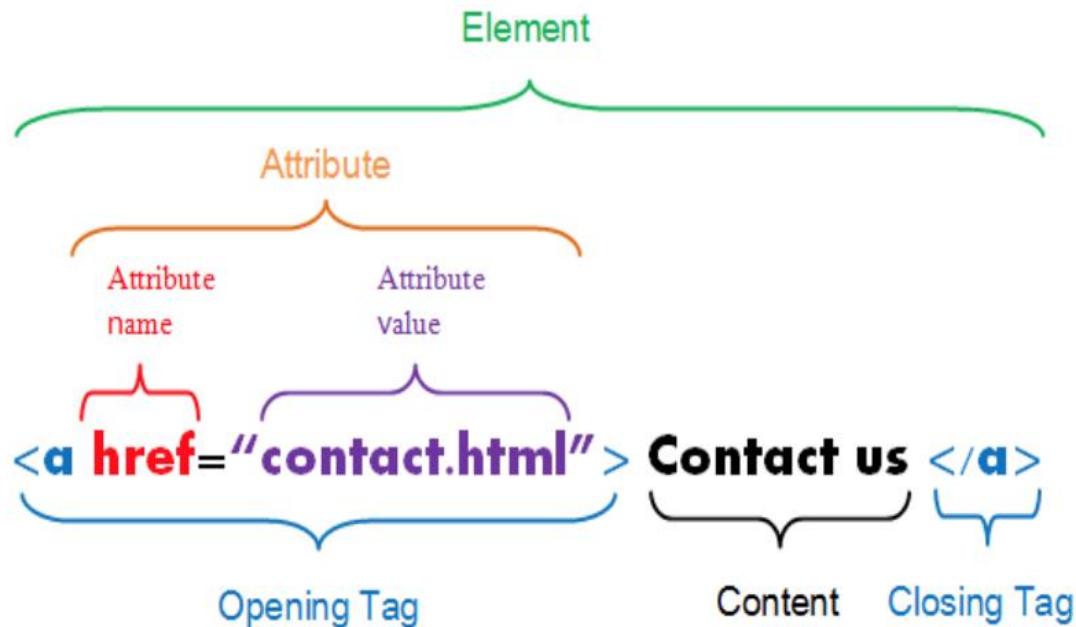
- Pacote do Node.js para a criação de servidor web para desenvolvimento.

HTML



- Hyper Text Markup Language.
- **Não** pode ser considerada uma **linguagem de programação**.
- É, na verdade, uma linguagem de **marcação**.
- Utilizada de forma **declarativa** para a estruturação de conteúdo na web.
- Define **elementos**, que são delimitados por **tags**.
- **Tags** podem possuir **atributos** e **conteúdo**.
- **Tags** com conteúdo devem ser encerradas.
- **Atributos** possuem **valores**.
- **Muita** importância **semântica**.
- **Pouca** importância **visual**.
- Excelente fonte de documentação - <https://developer.mozilla.org/pt-BR/docs/Web/HTML>

Estrutura de um elemento HTML



Estrutura de um elemento HTML – [Fonte](#)

Principais tags HTML para marcação

- `<p>` → parágrafos.
- `<h1>` a `<h6>` → títulos.
- `` → trechos a serem destacados.
- `<div>` → divisões da página.
- `` → definição de imagens.
- `<table>` `<tr>` `<td>` → definição de tabelas, linhas e colunas.
- `` `` → listas e itens de lista.
- `` e `` → ênfase no texto.
- `<a>` → âncoras (links).

- Muito utilizados em imagens e links.
- Caminho absoluto:
 - Local **absoluto** do recurso (arquivo, imagem, etc.) **em disco**.
 - Em geral, só funciona no ambiente original do site/página HTML.
 - Portanto, **evite**.
- Caminho relativo:
 - Local **relativo** do recurso (arquivo, imagem, etc.) **em relação aos demais arquivos do projeto**.
 - Pasta local → ' ./ '
 - Pasta pai/mãe → ' ../ '
 - Utilize sempre caminhos relativos.
 - Assim, as referências ao seu projeto funcionarão **em qualquer ambiente**.

- Acompanhe o professor.
- Serão demonstrados os seguintes conteúdos:
 - Estruturação de uma página HTML.
 - Utilização das principais tags.
 - Diferenciação entre caminho absoluto e caminho relativo.

- ✓ HTML é uma linguagem de marcação.
- ✓ Utilizada para a definição de conteúdo na web.
- ✓ Foco em semântica.
- ✓ Composta de elementos.
- ✓ Elementos são delimitados por tags.
- ✓ Elementos com conteúdo precisam de tag de fechamento.
- ✓ Tags podem possuir atributos.
- ✓ Atributos podem possuir valores.



- Cascading Style Sheets.
- Permite a estilização do conteúdo HTML.
- Foco no conteúdo visual.
- Utilizada de forma declarativa.
- Permite a alteração de cores, estilos de texto e posicionamento de elementos.
- Pode ser definido no HTML de três formas:
 - Atributo **style**.
 - Tag **style**.
 - Arquivo externo.

```
2  h1 selector
3  { declaration
4      padding: 10px;
5      margin: 0 auto;
6      color: blue;
7  } property value
8
```

Sintaxe do CSS – [Fonte](#)


- **Elemento:**

- Estiliza todos os elementos conforme identificador.
- Representado pelo nome da tag.

```
p {  
  color:  blue;  
}
```


- **Classe:**

- Estiliza todos os elementos que possuam a classe (atributo **class**).
- Representado no CSS por um ponto (.).

```
.destaque {  
  color:  green;  
}
```

- **Id:**

- Estiliza o elemento que possui o id (atributo **id**).
- Geralmente deve ser feito para um único elemento.
- Representado no CSS por uma hashtag (#).

```
#menu {  
  color:  yellow;  
}
```

- Técnica para remover o CSS padrão dos navegadores.
- Auxilia na padronização visual do site.
- Deve ser feita antes de qualquer outra declaração de CSS.
- Há diversas maneiras para se fazer o CSS Reset.
- Uma das técnicas mais utilizadas é a inclusão do arquivo reset.css, criado por Eric Meyer, que pode ser obtido [aqui](#).

- Acompanhe o professor.
- Serão demonstrados os seguintes conteúdos:
 - Integração de HTML e CSS.
 - CSS Reset.
 - Estilização de texto.
 - Estilização de cores.

- ☑ CSS é uma linguagem de marcação.
- ☑ Utilizada para a estilização de conteúdo HTML na web.
- ☑ Foco em estilos, cores e posicionamento.
- ☑ É possível aplicar estilização em elementos, classes e id's.
- ☑ É recomendada a utilização de CSS Reset antes de qualquer outra estilização.

- Linguagem de programação.
- A relação com Java é apenas no nome comercial.
- Nome formal da linguagem – ECMAScript.
- Principal utilização – Front End Web.
- Também pode ser utilizada em:
 - Backend, com Node.js.
 - Aplicações desktop, com Electron.
 - Dispositivos embarcados.
 - Internet das Coisas (IoT).
 - Machine learning.

- A maneira mais comum é a criação de um arquivo externo com extensão .js.
- Inclusão da tag `<script src="arquivo.js"></script>` antes do encerramento de `<body>`
- Acompanhe o professor.

- ☑ JavaScript é uma das principais linguagens de programação atualmente.
- ☑ É utilizada principalmente para interação do usuário em Front End Web.
- ☑ Possui diversas outras aplicações, tais como:
 - Backend com Node.js.
 - Desktop com Electron.
 - Machine Learning.
- ☑ Facilmente integrada ao HTML e CSS.

- Existem atualmente **8** tipos de dados em JavaScript.
- Vamos focar apenas nos principais:
 - Number → 1, -3, 8.56
 - String → "Teste", "3.14", 'Aspas simples'
 - Boolean → true, false
 - Null → null (explicitamente definido pelo programador)
 - Undefined → undefined (ausência de valor)
 - Object → [1, 3, 5], [6, 'sete', true], {id: 2, nome: 'Raphael'}
- Apesar dos tipos, JavaScript é considerada uma linguagem de programação com **tipagem fraca**.

- Criamos variáveis para guardar valores que serão reutilizados posteriormente.
- Esses valores são guardados em memória.
- Palavra-chave: `var`.
- Operador de atribuição de valores: `=` (igualdade).
- Exemplo de comando: `var pi = 3.14;`
- Acompanhe o professor na demonstração de:
 - Tipos, valores e variáveis.

- Soma → +
- Subtração → -
- Multiplicação → *
- Divisão → /
- Exponenciação → **
- Resto de divisão → %
- Incremento → ++
- Decremento → --

- Atribuição padrão → =
- Atribuição com operação → += -= *= /= %=

- Igualdade → ===
- Diferença → !==
- Maior → >
- Menor → <
- Maior ou igual → >=
- Menor ou igual → <=

- E → &&
- OU → ||
- Negação → !

- Muito útil para testes simples e depuração de código (debug).
- Principal comando: `console.log()`.

```
> var x = 10;
< undefined
> console.log(x);
10
< undefined
> console.log("O valor de x + 10 é " + (x + 10));
O valor de x + 10 é 20
< undefined
```

- Muito útil para descrever operações complexas.
- Algumas formas:
 - Comentários de linha: `// Comentário`
 - Comentários de bloco: `/* Comentário */`
- Acompanhe o professor na demonstração de:
 - Operadores.
 - Console.
 - Comentários.

☑ Principais tipos e valores:

- Number, String, Boolean, Object, Null e Undefined.

☑ Variáveis:

- Palavra-chave var.

☑ Operadores:

- Aritméticos, lógicos, de comparação, de atribuição.

☑ Console:

- O comando `console.log()`.

☑ Comentários:

- Comentários de linha e de bloco.

- Afirmativas que podem ser:
 - **Ou** verdadeiras.
 - **Ou** falsas.
 - **Jamais** serão **verdadeiras e falsas** ao mesmo tempo.
 - Podem ser aninhadas com operadores lógicos (&& / || / !)
- Exemplos:
 - `7 > 5` → `true`
 - `2 === 2 && 4 > 3` → `true`
 - `3 === '3' || 2 > 2` → `false`
 - `9 < 11 && !(9 < 11)` → `false`

- O comando **if/else**:
 - Principal estrutura de decisão.
 - Testa uma proposição lógica.
 - O bloco **if** é executado se a proposição lógica for **true**.
 - O bloco **else** é executado se a proposição lógica for **false**.
 - O bloco **else** é opcional.
 - Os comandos podem ser aninhados.
 - O comando executa **somente um dos blocos**.

- O comando **switch**:
 - Útil para testes de **igualdade** com **vários valores possíveis**.
 - Sintaxe mais elegante e legível que if/else.
 - O **else** é implementado com a cláusula **default**.
- **Operador ternário**:
 - Útil para substituir if/else em comandos simples.
 - Sintaxe um pouco confusa, porém pode ser elegante.
 - Utiliza os símbolos **?** e **:**

- **Acompanhe o professor:**
 - Demonstração de if/else.
 - Demonstração de switch.
 - Demonstração de operador ternário.

- O comando **while**:
 - Executa o bloco **enquanto** a proposição lógica for **true**.
 - É necessário que tornemos a proposição **false** em algum momento.
 - Caso isso não ocorra, acontecerá o **loop infinito**.
 - Geralmente utilizamos uma variável de controle para garantir a correta execução do comando.

- O comando **do... while**:
 - Muito semelhante ao **while**.
 - A diferença é que, no **do... while**, a proposição é **testada ao final do bloco**.
 - Com isso, garante-se que o bloco seja **executado pelo menos uma vez**.

- O comando **for**:
 - Comando semelhante ao **while** e **do... while**.
 - Entretanto, possui uma **sintaxe mais elegante e menos propensa a erros**.
 - Inicialização da variável de controle, teste da proposição e incremento/decremento são feitos na mesma linha.
- Acompanhe o professor na demonstração de:
 - Comandos **while**, **do... while** e **for**.

- Bloco de código que executa algum tipo de algoritmo.
- **Não é executado imediatamente.**
- Deve ser **invocado posteriormente.**
- Permite a reutilização de lógica.
- Pode aceitar **parâmetros**, também conhecidos como **argumentos**.
- Pode retornar **zero** ou **apenas um valor**.
- Acompanhe o professor.

☑ Proposição lógica:

- Afirmativa que podem ser ou verdadeira ou falsa.

☑ Estruturas de decisão:

- Comandos if/else, switch e operador ternário (? :).

☑ Estruturas de repetição:

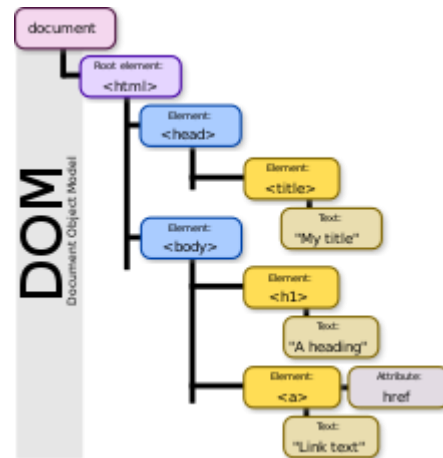
- Comandos while, do... while e for.

☑ Funções

- Blocos de código reutilizáveis que implementam algoritmos.

DOM – Introdução

- Document Object Model.
- Representa uma árvore de objetos em memória.
- No caso da web, mapeia documentos HTML em objetos.
- Com isso, pode-se facilmente recuperar e modificar dados dos documentos.
- Para mais informações sobre o DOM, acesse [este link](#).



O comando *querySelector()*

- Principal comando para acessar elementos do DOM um-a-um.
- Opções de parâmetros:
 - Elemento / Classes / Id's.

```
> var firstParagraph = document.querySelector('p');
< undefined
> var mediawiki = document.querySelector('.mediawiki');
< undefined
> var content = document.querySelector('#content');
< undefined
> firstParagraph
< ▶ <p>...</p>
> mediawiki
< ▶ {redefineFallbacksForTest: undefined, trackQueue: Array(0), now: f, track: f, trackError: f, ...}
> content
< ▶ <div id="content" class="mw-body" role="main">...</div>
```

O comando *querySelectorAll()*

- Principal comando para acessar elementos agrupados do DOM.
- O retorno é do tipo NodeList. Para converter para array (mais comum), use Array.from
- Opções de parâmetros:
 - Elemento / Classes / Id's.

```
> var allParagraphs = document.querySelectorAll('p');  
⏏ undefined  
  
> allParagraphs  
⏏ ▶ NodeList(16) [p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p]  
  
> var arrayParagraphs = Array.from(allParagraphs);  
⏏ undefined  
  
> arrayParagraphs  
⏏ ▶ (16) [p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, p]
```

A propriedade *textContent*

- Refere-se ao conteúdo textual de alguns elementos.
- Acompanhe o professor para demonstração de exemplos com:
 - document.querySelector
 - document.querySelectorAll

Parágrafo 1

```
> var element = |
```

- ✓ Na web, é possível manipular o DOM com JavaScript.
- ✓ **querySelector** é utilizado para se obter um único elemento.
- ✓ **querySelectorAll** é utilizado para se obter vários elementos semelhantes.
- ✓ **textContent** refere-se ao conteúdo textual de diversos elementos.

A propriedade *style*

- É possível modificar o CSS diretamente através da propriedade *style*.
- Entretanto, esta **não** é uma boa prática.
- Acompanhe o professor.
- Não é interessante injetarmos código CSS em nosso JavaScript.
- Uma boa prática seria a adição e remoção de classes.

Parágrafo 1

```
> var p |
```

O comando *classList.add()* e *classList.remove()*

- Com estes comandos, podemos adicionar e remover classes de elementos.

```
<style>
  .good {
    color: blue;
  }
  .great {
    color: green;
  }
  .awesome {
    color: purple;
  }
</style>
</head>
<body>
  <p id="p1">Parágrafo 1</p>
  <p id="p2">Parágrafo 2</p>
  <p id="p3">Parágrafo 3</p>
</body>
```

Parágrafo 1

Parágrafo 2

Parágrafo 3

```
> var p1 = document.getElementById('p1');
p1.classList.add('good');
p1.classList.add('great');
p1.classList.add('awesome');
```


- ☑ Na web, é possível manipular o CSS com JavaScript.
- ☑ Não é uma boa prática manipular o CSS diretamente.
- ☑ É melhor definir classes de CSS em arquivos isolados e invocar **classList.add** e/ou **classList.remove**.

- Principal interação com os usuários em sistemas web.
- Tag **<form>**.
- Funcionamento padrão:
 - Botão submit – envia os dados ao servidor;
 - Botão reset – limpa os dados do formulário.
- Funcionamento atual:
 - JavaScript intercepta os dados.
 - Envio de dados ao servidor feito de forma assíncrona.
 - Tag **<button>** também é muito utilizada.

Formulários – Campos de input

- **<textarea>** - usado para grandes quantidades de texto.
- **<input>** - elemento mais utilizado, contendo:
 - Caixas de texto simples.
 - Data/hora.
 - Checkboxes.
 - Cores.
 - Números.
 - E-mail.
 - Botões submit e reset.
 - Outros.
- Renderização varia conforme o navegador.
- Acompanhe o professor.

The screenshot shows a web form with several input fields and validation messages. The fields include:

- A text input with "gmailcom" and a validation message: "Insira um URL." (Insert a URL).
- A text input with "teste" and a close button (X).
- A text input with "raphaelyahoo.com" and a validation message: "Inclua um '@' no endereço de e-mail. 'raphaelyahoo.com' está com um '@' faltando." (Include an '@' in the email address. 'raphaelyahoo.com' is missing an '@').
- A dropdown menu with "3" and a close button (X).
- A text input with "agosto de 2016" and a close button (X).
- A text input with "Semana 02, 2016" and a close button (X).
- A text input with "03:01" and a close button (X).
- A text input with "06/08/2015 02:03" and a close button (X).

There is also a calendar widget showing "agosto de 2015" with a table of dates:

dom	seg	ter	qua	qui	sex	sáb
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

☑ Formulários:

- Principal forma de interação com o usuário.
- Vários elementos disponíveis.
- Diversos tipos de inputs e botões.
- Principais atributos utilizados pelo JavaScript: class/id e value.

- Representam o “quando” das aplicações.
- A implementação é feita através de “escutadores” (*listeners*), com a função **addEventListener**.
- Acompanhe o professor.

Evento	Ativação	Categoria
onload	Após o carregamento	document / window / body
onchange	Conteúdo do elemento alterado	form
onfocus	Elemento recebe foco	form
onblur	Elemento perde o foco	form
onselect	Elemento é selecionado	form
onsubmit	Dados do formulário são enviados ao servidor	form
onkeydown	Tecla pressionada	teclado
onkeypress	Tecla pressionada e solta	teclado
onkeyup	Tecla solta	teclado
onclick	Clique do mouse no elemento	mouse
ondblclick	Clique duplo do mouse no elemento	mouse
onmousemove	Mouse se moveu sobre o elemento	mouse
onmouseout	Mouse saiu do elemento	mouse
onmouseover	Mouse passou sobre o elemento	mouse
onmouseup	Botão do mouse solto sobre o elemento	mouse

JavaScript – Evitando envio de dados ao servidor



- Por padrão, ao clicar em um botão do tipo submit de um formulário HTML:
 - Os dados são enviados ao servidor.
 - A página é recarregada.
- Atualmente, grande parte dos sistemas web funciona como SPA (Single Page Application).
- Assim, deve-se evitar o **refresh** com **evento.preventDefault**.
- Dados **reagem** às interações do usuário instantaneamente.
- Acompanhe o professor.

☒ Eventos:

- Implementam o quando.
- Melhoram a experiência do usuário em sistemas web.

☒ Sistemas web geralmente devem se comportar como SPA's (Single Page Applications).

- Acompanhe o professor.

- ☑ Já possível construir o front end de diversos sistemas web com a informação vista até aqui.
- ☑ Entretanto, o JavaScript tem muito mais a oferecer.
- ☑ O JavaScript moderno melhora a experiência do desenvolvedor.

- ECMAScript em constante evolução.
- Navegadores acompanham de forma mais lenta.
- Risco de “quebra” da internet.
- A evolução do JavaScript é regulada pelo [TC39](#).
- Conseguimos utilizar ES6+ com transpiladores.
 - Babel.
 - TypeScript.

- **var** já foi visto anteriormente.
- **var** possui escopo amplo.
- É melhor utilizar **let**, que possui escopo reduzido.
- Utilizamos **const** para garantir imutabilidade.
- Com **const**, não é possível atribuir um novo valor à variável.
- Entretanto, é possível modificar os dados de **arrays** e **objetos**.
- Uma boa prática é sempre começar a declaração com **const** e trocar por **let** se for realmente necessário.
- **var** tem sido raramente utilizado atualmente.

Arrow functions

- Sintaxe mais simples.
- Escrita mais declarativa e funcional.
- Exemplo:

```
const makeBeer = function beerFun(qty) {  
  return '🍺'.repeat(qty);  
}  
  
const makeWine = (qty) => '🍷'.repeat(qty);
```

Fonte: fireship.io

Template literals

- Maneira mais simples e elegante de criar Strings a partir de expressões.
- Utiliza o símbolo ` (crase).
- Expressões JavaScript ficam entre \${}.

```
> const a = 1, b = 2, c = 3;  
> a = 1, b = 2, c = 3;  
> array = [1,2,3,4]  
> array3 = [...array1, ...array2];  
> array2 = [4, 5, 6];  
> array1 = [1, 2, 3];  
> array3 = [...array1, array2];  
> accounts = temp1.map(item => {return {id: ...  
> accounts = temp1.map(item => {id: item.id,...
```

- É possível atribuir valores padrão para parâmetros de funções.
- Em geral, parâmetros com valores padrão devem se situar no **final** da função.
- Acompanhe o professor para exemplos de:
 - var x const x let.
 - Arrow functions.
 - Template literals.
 - Default parameters.

- ☑ **var** deve ser evitado, pois é propenso a bugs de escopo.
- ☑ Utilize **const** por padrão para declarar variáveis.
- ☑ Utilize **let** caso seja necessário reatribuir um novo valor à variável.
- ☑ **Arrow functions** permitem escrita mais simples e funcional em comparação a funções.
- ☑ **Template literals** facilitam a escrita de Strings com expressões JavaScript.
- ☑ **Default parameters** flexibilizam a utilização de parâmetros em funções.

- **map** → gera um novo array transformando os dados.
- **filter** → gera um novo array filtrando elementos com base em proposição.
- **forEach** → percorre todos os elementos do array, aplicando lógica.
- **reduce** → realiza cálculo iterativo com base nos elementos.
- **find** → encontra elementos com base em proposições.
- **some** → verifica se há pelo menos um elemento que atenda à proposição.
- **every** → verifica se todos os elementos atendem à proposição.
- **sort** → ordena os elementos com base em um critério.
- Acompanhe o professor.

- ☑ O JavaScript possui excelentes métodos para manipulação de arrays.
- ☑ Métodos podem ser customizáveis através de **callbacks**.
- ☑ É uma boa prática a utilização de arrow functions nesses métodos, para simplificar a escrita.

Operador ... (spread)

- Muito útil para trabalhar com arrays e objetos.
- Em arrays, este operador **espalha** os itens do array, que podem ser recuperados para compor outro array, por exemplo.
- Acompanhe o professor.

```
> const array|  
  > array3 = [...array1, ...array2];  
  > array2 = [4, 5, 6];  
  > array1 = [1, 2, 3];  
  > array3 = [...array1, array2];
```

Operador ... (rest)

- Muito útil para trabalhar com arrays e objetos.
- Como **rest**, é comum a utilização em funções, **agrupando** os parâmetros em um array.
- Principal aplicação → permitir funções com número infinito de parâmetros.
- Acompanhe o professor.

```
> const super|
```

Destructuring

- Facilita a escrita ao trabalhar com objetos.
- Torna o código mais claro.
- É também possível utilizar a técnica de *destructuring* com arrays, usando [].
- Acompanhe o professor.

```
> //sem des|  
← undefined
```

```
> //|  
← undefined
```

☑ Rest/spread:

- Espalha elementos de vetores (spread).
- Agrupa elementos em funções (rest).

☑ Destructuring:

- Permite uma melhor escrita e legibilidade de código.
- Compatível com arrays e objetos.

Refatoração do projeto do Capítulo 3

- Acompanhe o professor.
- O projeto será refatorado com ES6+.

Conclusão

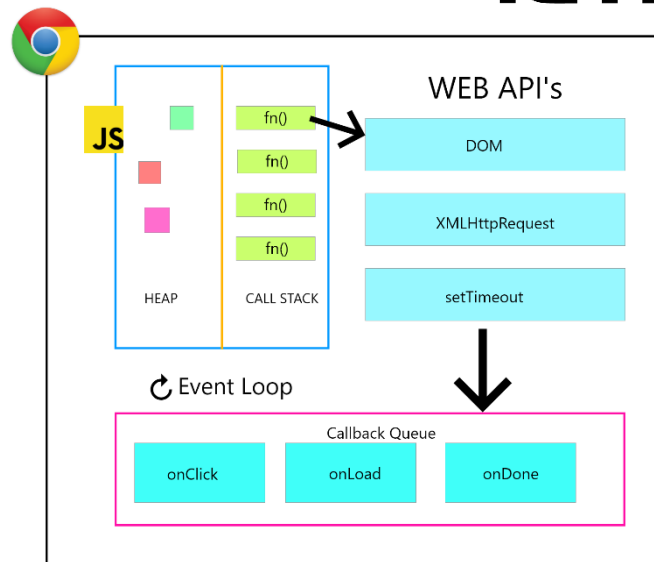
- ✓ Código **mais** organizado.
- ✓ Código **mais** elegante.
- ✓ Código **menos** compatível.

- Em JavaScript existem operações que podem ser lentas, como por exemplo:
 - Requisição de dados à APIs.
 - Processamento intenso de dados.
 - Comunicação com bancos de dados (Node.js).
- É extremamente importante que o JavaScript **não espere o término de instruções lentas.**
- A principal técnica para garantir a afirmação acima é a utilização do **event loop**.

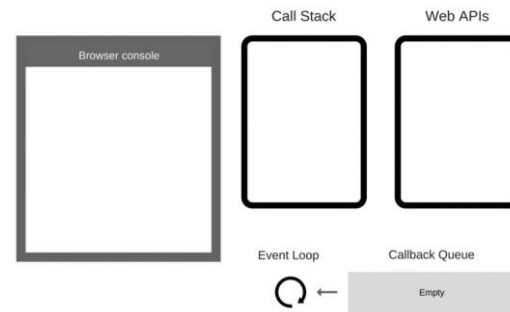
JavaScript – Event loop

- Funções a serem executadas ficam em uma pilha lógica de invocações (call stack).
- Quando a função utiliza Web APIs, ela precisa passar pelo event loop, pois está sujeita à lentidões.
- O event loop executa uma função por vez e faz a orquestração que permite execução assíncrona.
- Em geral, funções que usam WEB APIs possuem **callbacks** (funções passadas por parâmetro).
- Mais informações [aqui](#).

```
1 console.log('Hi');
2 setTimeout(function cb1() {
3   console.log('cb1');
4 }, 5000);
5 console.log('Bye');
```



1 / 16



- ☑ O JavaScript é **síncrono** por padrão.
- ☑ Entretanto, os ambientes de execução possuem o **event loop** para prover comportamento **assíncrono**.
- ☑ Funções que utilizam Web APIs utilizam o **event loop**.
- ☑ Em geral, após a execução no **event loop**, a função executa uma nova função de completude, denominada **callback**.
- ☑ **Callbacks** são **funções** passadas por **parâmetros** em funções que irão utilizar de alguma forma a Web API (Eventos de DOM, requisições HTTP, timeouts, etc.)

- **`setTimeout`:**

- Utilizada para postergar a execução de uma função.
- Tempo de atraso configurável em milissegundos.

- **`setInterval`:**

- Semelhante ao **`setTimeout`**, mas repete a execução a cada x milissegundos.
- Pode ser cancelada com **`clearInterval`**.
- Para isso, devemos guardar a referência em uma variável.


- Acompanhe o professor.

- ☑ A função **setTimeout** é utilizada para postergar execuções.
- ☑ A função **setInterval** posterga e repete as execuções a cada x milissegundos.
- ☑ Em geral, é importante parar execuções de **setInterval**. Isso é feito com **clearInterval**.

- Utilizado para requisições HTTP.
- Trabalha internamente com promises.
- O primeiro retorno de fetch são dados binários.
- Em geral, convertemos esses dados para JSON, que retorna outra promise.
- Acompanhe o professor.

Promises

- Promises são construções cuja execução **retorna algo no futuro**, ou seja, é uma **promessa de execução**.
- A execução pode ser **resolvida (ok)**, ou **rejeitada (erro)**.
- A promise resolvida é interceptada com **then**.
- A promise rejeitada é interceptada com **catch**.
- Resolve parcialmente o problema do ***callback hell***.



```
1 function hell(win) {
2   // for listener purpose
3   return function() {
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                  loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                    loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                      async.eachSeries(SCRIPTS, function(src, callback) {
14                        loadScript(win, BASE_URL+src, callback);
15                      });
16                    });
17                  });
18                });
19              });
20            });
21          });
22        });
23      });
24    });
25  });
26 }
```

Async/await

- Açúcar sintático (syntax sugar) sobre promises.
- Melhoram a **legibilidade** do código.
- Dá a impressão de código síncrono.
- Deve-se decorar a função com **async**.
- Toda instrução relacionada à promise deve ser precedida de **await**.
- Acompanhe o professor.

#CALLBACK

```
GetUser(function(err, user){
  GetProfile(user, function(err, profile){
    GetAccount(profile, function(err, acc){
      GetReport(acc, function(err, report){
        SendStatistics(report, function(e){
          ...
        });
      });
    });
  });
});
```

#PROMISE

```
GetUser()
  .then(GetProfile)
  .then(GetAccount)
  .then(GetReport)
  .then(SendStatistics)
  .then(function (success) {
    console.log(success)
  })
  .catch(function (e) {
    console.error(e)
  })
```

#ASYNC/AWAIT

```
async function SendAsync() {
  let user = await GetUser(1)
  let profile = await GetProfile(user);
  let account = await GetAccount(profile);
  let report = await GetReport(account);

  let send = SendStatistic(report);

  console.log(send)
}
```



- ☑ O comando **fetch** é utilizado para requisições HTTP.
- ☑ Trabalha internamente com **promises**.
- ☑ O mais comum é que os dados sejam retornados no formato JSON.
- ☑ Promises.
 - Muito utilizadas no JavaScript assíncrono.
 - Resolvem parcialmente o problema do callback hell.
- ☑ Async/await:
 - Açúcar sintático de **promises**.
 - Melhora a legibilidade do código.

Desafio 2

- Construir código JavaScript para listar países e marca-los como favoritos.
 1. Busque os dados em <https://restcountries.eu/rest/v2/all> com fetch.
 2. Transforme os dados de forma que somente os seguintes itens sejam disponíveis:
 - Nome
 - Bandeira
 - População
 - Id (numericCode)
 3. Mostre a quantidade de países e o total de população na lista da esquerda e da direita.

- Construir código JavaScript para listar países e marca-los como favorito.
 4. Ao adicionar aos favoritos, o país deve ser movido da esquerda para a direita.
 5. Ao remover dos favoritos, o país deve ser reinserido na lista da esquerda.
 6. As listas de países devem ser **sempre** exibidas em **ordem alfabética**.
 7. Um **desafio extra** – formatar os números com JavaScript puro (pesquise por **Intl**).



















Desafio 2

- Desafio proposto em execução.

Desafio 02

Países (248)

População total: 7.321.402.072

		Åland Islands 28.875
		Albania 2.886.026
		Algeria 40.400.000
		American Samoa 57.100
		Angola 25.868.000
		Anguilla 13.452
		Antarctica 1.000
		Antigua and Barbuda 86.295
		Argentina 43.590.400

Países (2)

População total: 27.735.159

		Afghanistan 27.657.145
		Andorra 78.014

Resolução do desafio

- ☐ Acompanhe o professor.

- ✓ Organização do código em funções.
- ✓ Código auto-documentado.
- ✓ Utilização de fetch com async/await.
- ✓ Utilização de template literals.

- ☑ Introdução ao JavaScript.
- ☑ Variáveis, tipos e valores.
- ☑ Comandos de bloco.
- ☑ JavaScript moderno.
- ☑ Programação assíncrona.