

Informática
CURSO TÉCNICO

Programação WEB

Aula 01: Introdução ao JS

O que é o Javascript?

- Utilizada para prover interatividade e dinamismo a websites
- Permite programar o comportamento da página Web na ocorrência de eventos
- Permite alterar o documento HTML por meio da manipulação da árvore DOM
- Comumente referenciada como JS
- Pode ser executada tanto do lado do navegador como do servidor (Node.js)
- Não é necessário compilar explicitamente o código JavaScript
- Não confundir com a linguagem de programação Java

Javascript e ECMAScript

- Ecma International - Organização que desenvolve padrões
- ECMAScript é uma linguagem padronizada, uma especificação
- ECMA262 é o nome do padrão propriamente dito
- JavaScript é uma implementação da linguagem ECMAScript
- Outras implementações:
 - JScript e ActionScript
- JavaScript originalmente desenvolvida por Brendan Eich da Netscape

Javascript Embutido no HTML

```
<html>

  <head>
    <script>
      // Código JavaScript
    </script>
  </head>

  <body>
    ...
  </body>

</html>
```

*Código JavaScript embutido no
cabeçalho do documento HTML*

```
<html>

  <head>
    ...
  </head>

  <body>
    ...
    <script>
      // Código JavaScript
    </script>
    ...
  </body>

</html>
```

*Código JavaScript embutido no
corpo do documento HTML
(poderia ser depois de </body>)*

Javascript em Arquivo Separado

Arquivo HTML

```
<html>

  <head>

    <script src="arquivoJavaScript.js"></script>

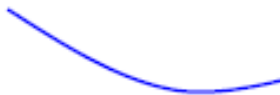
  </head>

  <body>

    ...

  </body>

</html>
```



Arquivo JS

```
/* arquivoJavaScript.js */
```

Observações Gerais

- JavaScript é sensível a maiúsculas e minúsculas (case sensitive)
- Declarações podem ou não terminar com o ponto e vírgula
- Os tipos das variáveis são definidos automaticamente
- Comentários de linha: `// comentário`
- Comentários de bloco: `/* comentário */`

Estruturas condicionais e de Repetição

```
if (expressão) {  
    // operações se verdadeiro  
}  
else {  
    // operações se falso  
}
```

```
switch (expressao) {  
    case condicao1:  
        // operações  
        break;  
  
    case condicaoN:  
        // operações  
        break;  
    ...  
    default:  
        // operações  
}
```

```
for (let i = 0; i < 10; i++)  
{  
    // operações  
}
```

```
while (expressao)  
{  
    // operações  
}
```

```
do {  
    // operações  
} while (expressao)
```

Declaração de variáveis

`var nomeDaVariável= valorInicial`

- Variável com escopo local se declarada dentro de uma função
- Variável com escopo global se declarada fora de funções
- Pode ser redeclarada e pode ter valor atualizado
- Variáveis globais também podem ser acessadas pelo objeto *window*

`let nomeDaVariável= valorInicial`

- Variável tem escopo restrito ao bloco de código
- Pode ser acessada e atualizada apenas dentro do bloco
- Não pode ser redeclarada no mesmo bloco

`const nomeDaConstante= valor`

- Semelhante a let
- Porém não pode ser atualizada
- Deve ser inicializada no momento da declaração

Exemplo de Variáveis

```
<script>
  const pi = 3.14
  var soma = 0; // soma é uma variável global
  for (let i = 1; i <= 10; i++) {
    soma += i
  }
  if (soma > 50) {
    let k = soma + pi; // k só pode ser acessada aqui
    var m = k + 1
    console.log(k)
  }
  console.log(m)
  console.log(k)
</script>
```

Operadores aritméticos, relacionais e lógicos

Operadores Aritméticos e Atribuição

Operador	Significado
+	Adição (e concatenação)
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira
++	Incremento
--	Decremento
=	Atribuição
+=	Atribuição com soma
-=	Atribuição com sub.

Operadores Relacionais e Lógicos

Operador	Significado
==	Comparação por igualdade
===	Comparação por igualdade, incluindo valor e tipo
!=	Diferente
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a
&&	“E” lógico
	“Ou” lógico
!	Negação lógica

Operador de adição e concatenação

- O operador **+** deve ser utilizado com atenção
- Possibilita **somar** ou **concatenar**, dependendo dos operandos
- Se um dos operandos é uma **string** então será feita a **concatenação**.
 - o outro operando é convertido para string, caso não seja
- Se os dois operandos são **numéricos** então é realizada a **soma**
- Exemplos
 - `let x=5+5; // x terá o valor 10`
 - `let y="5"+5; // y terá a string '55'`

Diferença dos operadores == e ===

- Operador ==
 - Compara apenas valores
- Operandos de tipos diferentes são convertidos e valores comparados
- Operador ===
 - Compara o valor e o tipo dos operandos
 - Operandos de tipos diferentes sempre resulta em falso

```
1 == true;    // true;  
1 === true;   // false;  
10 == "10"    // true;  
10 === "10"   // false;
```

Objetos *window*, *navigator*, *document*

window

- Representa a aba do navegador que contém a página
- Possibilita obter informações ou realizar ações a respeito da janela, como:
 - Obter dimensões: `window.innerWidth`; e `window.innerHeight`
 - Executar uma ação quando a aba for carregada, fechada, etc.

navigator(ou `window.navigator`)

- Representa o navegador de Internet em uso (browser, user-agent)
- Fornece informações como idioma do navegador, geolocalização, memória, etc.
- Ex.: `alert(navigator.language);` // mostra “pt-BR”

document(ou `window.document`)

- Representa o documento HTML carregado na aba do navegador
- Possibilita a manipulação da árvore DOM

Métodos para E/S

`window.alert`

exibe uma caixa de diálogo para mensagens (botão Ok)

`window.confirm`

exibe uma caixa de diálogo para confirmação (Ok/Cancelar)

`window.prompt`

exibe uma caixa de diálogo para entrada de texto

`document.write`

adiciona conteúdo no documento HTML

`console.log`

registra conteúdo de *log* no **console** do navegador

`console.warn`

registra mensagem de *warning* no **console** do navegador

`console.error`

registra mensagem de *erro* no **console** do navegador

Métodos para E/S

`window.alert`

exibe uma caixa de diálogo para mensagens (botão Ok)

`window.confirm`

exibe uma caixa de diálogo para confirmação (Ok/Cancelar)

`window.prompt`

exibe uma caixa de diálogo para entrada de texto

`document.write`

adiciona conteúdo no documento HTML

`console.log`

registra conteúdo de *log* no **console** do navegador

`console.warn`

registra mensagem de *warning* no **console** do navegador

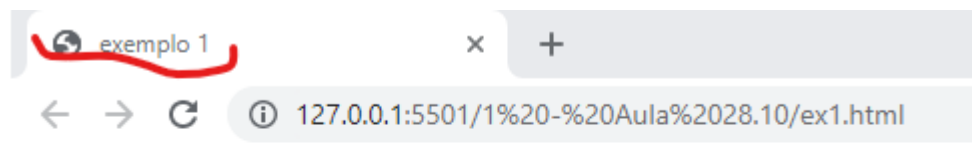
`console.error`

registra mensagem de *erro* no **console** do navegador

Exemplo 1

```
<h1>Titulo principal</h1>
<script>
  document.title = 'exemplo 1'
  document.write("<h2>" + document.title + "</h2>")
</script>
```

O código acima altera o título do site para “exemplo 1”. Também adiciona uma tag <h2> no documento. Veja o Resultado a seguir:



Titulo principal

exemplo 1

Declaração de funções

```
function nomeDaFuncao(par1, par2, par3, ...) {  
    // operações  
    // operações  
    // operações  
}
```

```
function max(a, b) {  
    if (a > b)  
        return a;  
    else  
        return b;  
}
```

```
let maior = max(2, 5);
```

Quando **'return'** não é utilizada, o valor **undefined** é automaticamente retornado

Recuperando elementos: getElementById()

O método **getElementById()** retorna o elemento cujo o atributo ID foi especificado.

Sintaxe:

```
document.getElementById("elementoID");
```

Observações:

- Caso o elemento não exista ou o ID informado seja incorreto ele retornará *null*.
- Observe que o parâmetro *elementoID* é case-sensitive, então o **document.getElementById ("Main")** irá retornar *null* em vez do elemento `<div id = "main">` porque "M" e "m" são diferentes para os efeitos deste método.
- Não deve existir dois elementos em uma página com o mesmo valor de um atributo ID. Caso exista mais de um elemento com o ID especificado, o método getElementById () retorna o primeiro elemento do código-fonte.
- Este método é um dos métodos mais comuns no HTML DOM, e é usado quase que toda vez que você deseja manipular, ou obter informações a partir de um elemento HTML.

Evento onclick HTML

- O evento onclick executa determinada funcionalidade quando um botão é clicado
- Você coloca a função em JavaScript que você quer executar dentro da tag de abertura do botão

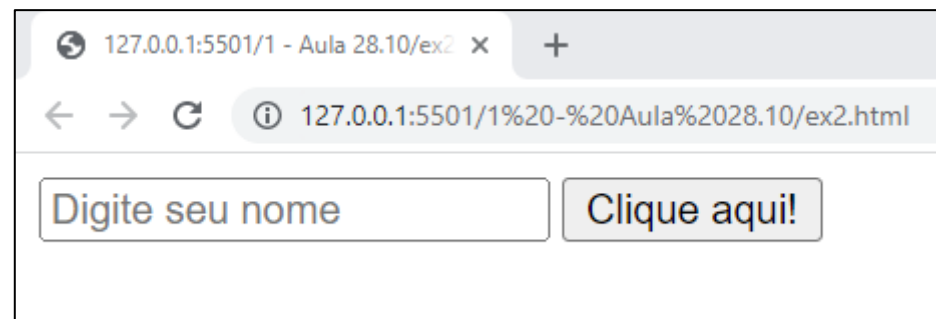
Sintaxe básica de onclick

```
<button onclick="funçãoQueSeráExecutada()">Clique aqui</button>
```

Observe que o atributo onclick é puramente JavaScript. O valor que ele recebe, que é a função que será executada, diz tudo, já que ela é invocada diretamente da tag de abertura.

Exemplo propriedade onclick

O exemplo tem dois elementos, um input e button. Ao clicar no button, é mostrado um alerta com o nome que foi digitado no input



Exemplo propriedade onclick

. A lógica do código é descrita a seguir:

- A função alerta() é chamada quando o botão é clicado (onclick)
- O elemento input é recuperado através da função getElementById
- Após recuperar o nome digitado, ele é mostrado como uma mensagem de alerta

```
<input placeholder="Digite seu nome" id="nome">
<button onclick="alerta()">Clique aqui!</button>

<script>
    function alerta() {
        //recupera o elemento input com id nome
        let input = document.getElementById('nome')
        console.log(input)
        //recupera o nome
        let nome = input.value
        alert('bem vindo ' + nome + '!')
    }
</script>
```

Exemplo adiciona +1

Agora vamos fazer um código JS que mostra um contador, que aumenta seu valor ao clicar no botão.



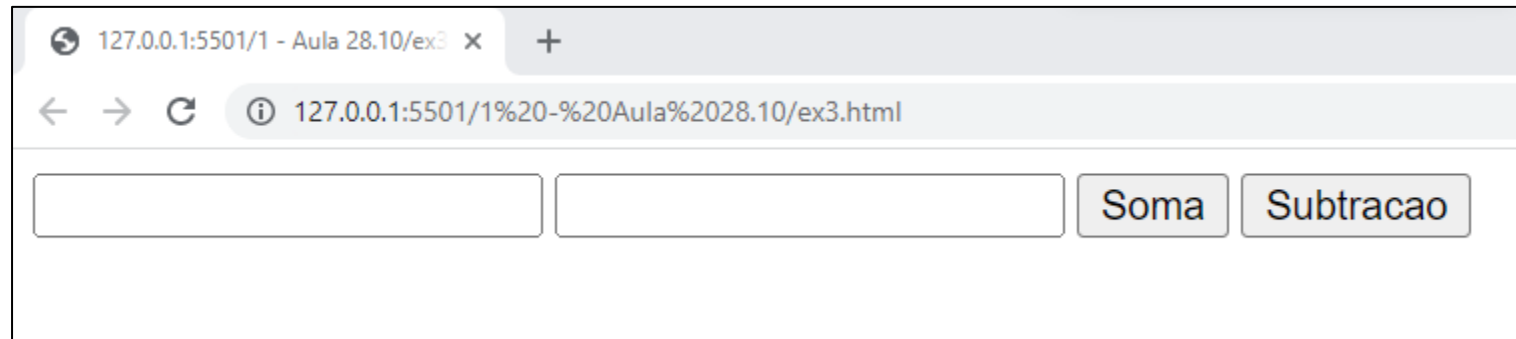
Exemplo adiciona +1

```
<h1 id="contador">Contador: 0</h1> <br>
<button onclick="adiciona()">Adicione +1</button>
<script>
  var cont = 0;
  function adiciona() {
    if (cont < 10) {
      cont += 1
      //recuperar elemento h1
      let elContador = document.getElementById('contador');
      elContador.innerHTML = 'Contador: ' + cont;
    }
    else {
      alert('Chegou no limite')
    }
  }
</script>
```

No exemplo acima, criamos uma variável cont, que armazena o valor do contador. Quando o botão é clicado, verifica se o valor do contador é menor que 10(limite). Com isso o contador é adicionado e o elemento é modificado

Exemplo soma, subtração

No próximo exemplo vamos ter dois input's que recebem números e realiza a soma ou subtração deles



The screenshot shows a web browser window with a single tab titled "127.0.0.1:5501/1 - Aula 28.10/ex3 x". The address bar displays the URL "127.0.0.1:5501/1%20-%20Aula%2028.10/ex3.html". Below the address bar, there are two empty text input fields side-by-side. To the right of these fields are two buttons: "Soma" and "Subtracao".

Exemplo soma/subtração

```
<input type="number" id="n1">
<input type="number" id="n2">
<button onclick="somar()">Soma</button>
<button onclick="subtracao()">Subtracao</button>
<script>
  //var : cria variavel global
  var input1 = document.getElementById('n1')
  var input2 = document.getElementById('n2')
  function somar() {
    //recuperar os valores
    num1 = parseInt(input1.value)
    num2 = parseInt(input2.value)
    soma = num1 + num2
    alert('A soma é: ' + soma)
  }
  function subtracao() {
    num1 = parseInt(input1.value)
    num2 = parseInt(input2.value)
    sub = num1 - num2
    alert('A subtração é: ' + sub)
  }
</script>
```

- No exemplo ao lado temos dois input's que recebem números e dois botões.
- Cada botão está configurado o evento de click com uma função diferente
- Observe que ao recuperar o número, é necessário convertê-lo para inteiro, pois o valor vem do input como string

Arrays em javascript

- Não são tipos de dados primitivos
- São tratados como objetos, com propriedades e métodos
- Podem armazenar valores de tipos diferentes
- Os elementos são acessados por índice numérico (e não por chaves)
 - Não são associados

Arrays em javascript

- Elementos colocados entre colchetes, separados por vírgula

```
let pares= [2, 4, 6, 8];  
  
let primeiroPar= pares[0]; // 1º elemento  
  
let nroElementos= pares.length;
```

- Elementos de diferentes tipos

```
let vetorMisto= [2, 'A', true];
```

- Pode ser iniciado com vazio

```
let pares= [];
```

Arrays em Javascript – Outros métodos

```
let vogais= ['E', 'I', 'O'];  
  
vogais.push('U')      // adiciona um item no final do vetor  
  
vogais.pop()          // remove e retorna o último item do vetor  
  
vogais.unshift('A')   // adiciona um item no início do vetor  
  
vogais.shift()         // remove e retorna o primeiro item do vetor  
  
vogais.indexOf('E')    // retorna a posição da 1ª ocorr. de um item (ou -1)  
  
vogais.length          // propriedade contendo o número de elem. do vetor
```

Percorrendo array com Estrutura **for**

```
let pares = [2, 4, 6, 8];  
for (let i = 0; i < pares.length; i++) {  
    console.log(pares[i]);  
};
```

```
let pares = [2, 4, 6, 8];  
for (let item of pares) {  
    console.log(item);  
};
```

Percorrendo array com Estrutura **forEach**

```
let pares = [2, 4, 6, 8];  
let soma = 0;  
pares.forEach( function (elemento) {  
    soma += elemento;  
});
```

Percorrendo array com método **forEach** e função **anônima**

Função anônima é uma função sem nome. No caso acima, a função é um parâmetro do método **forEach**. Para cada elemento do array, será executada a função

Percorrendo array com Estrutura **forEach**

```
let pares = [2, 4, 6, 8];  
let soma = 0;  
pares.forEach( elemento => soma += elemento );
```

```
let pares = [2, 4, 6, 8];  
pares.forEach( elemento => console.log(elemento) );
```

Percorrendo array com método **forEach** e *arrow function*

Arrow function é uma forma mais sucinta de se fazer uma função. Existem várias formas de se fazer, dependendo da quantidade de parâmetros e comandos da função.

No exemplo acima, **elemento** é o parâmetro da função.

Strings

Definida com aspas simples ou duplas

```
let msg= "JavaScript";
```

Acessando um caracter

```
let primLetra= msg[0];
```

```
let primLetra= msg.charAt(0);
```

Contra-barra para caracteres especiais

```
let msg= 'It\'s alright';
```

Strings com aspas duplas podem conter aspas simples e vice-versa

```
let msg= "It's alright";
```

Várias outras propriedades e métodos

```
length, indexOf, substr, split, etc.
```


Template String

- Strings definidas com o caractere crase (backtick): ``minha string``
- Suporta fácil interpolação de variáveis e expressões usando `${ }`
- Maior facilidade para definir strings de múltiplas linhas
- A string pode conter aspas simples ou duplas

```
<script>
  let a = 1
  let b = 2
  let c = 3
  const delta = b*b - 4*a*c
  console.log(`O discriminante da equação com
    coeficientes ${a}, ${b} e ${c} é ${delta}`)
</script>
```

Objetos simples (plain object, POJO)

- Contém apenas dados
- Comumente definido utilizando chaves { }
- Lista de pares do tipo *propriedade:valor*
- Criado como instância da classe **Object**

Objetos simples (plain object, POJO)

```
<script>
  let carro = {
    modelo: "Fusca",
    ano: 1970,
    cor: "bege",
    "motor-hp": 65
  }
  console.log(carro.ano)           // 1970
  console.log(carro["motor-hp"]); // 65
</script>
```

Informática
CURSO TÉCNICO

Programação WEB

Aula 02: DOM

Document Object Model

- Os navegadores criam árvore DOM da página
- Cada elemento é um nó
- Há uma relação de hierarquia entre os elementos. Eles podem ser filhos, pais, irmãos...
- Scripts acessam esses elementos através

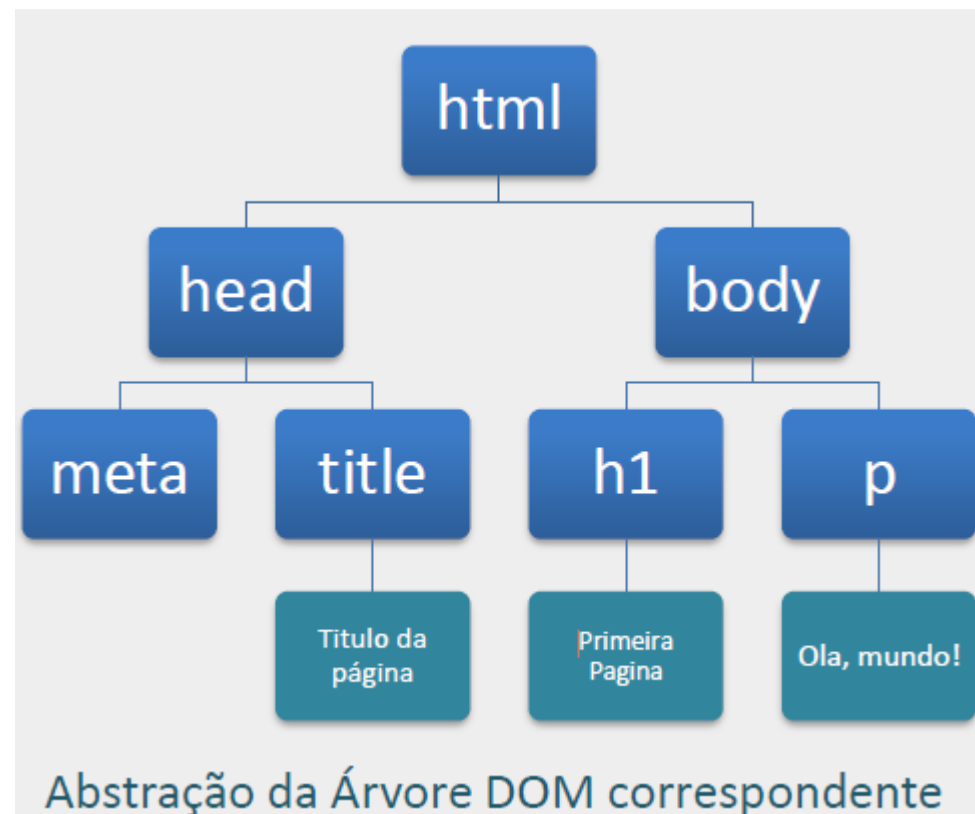
Manipulação da Árvore DOM

```
<!DOCTYPE html>
<html lang="pt-BR">

  <head>
    <meta charset="UTF-8">
    <title>Titulo da Pagina</title>
  </head>

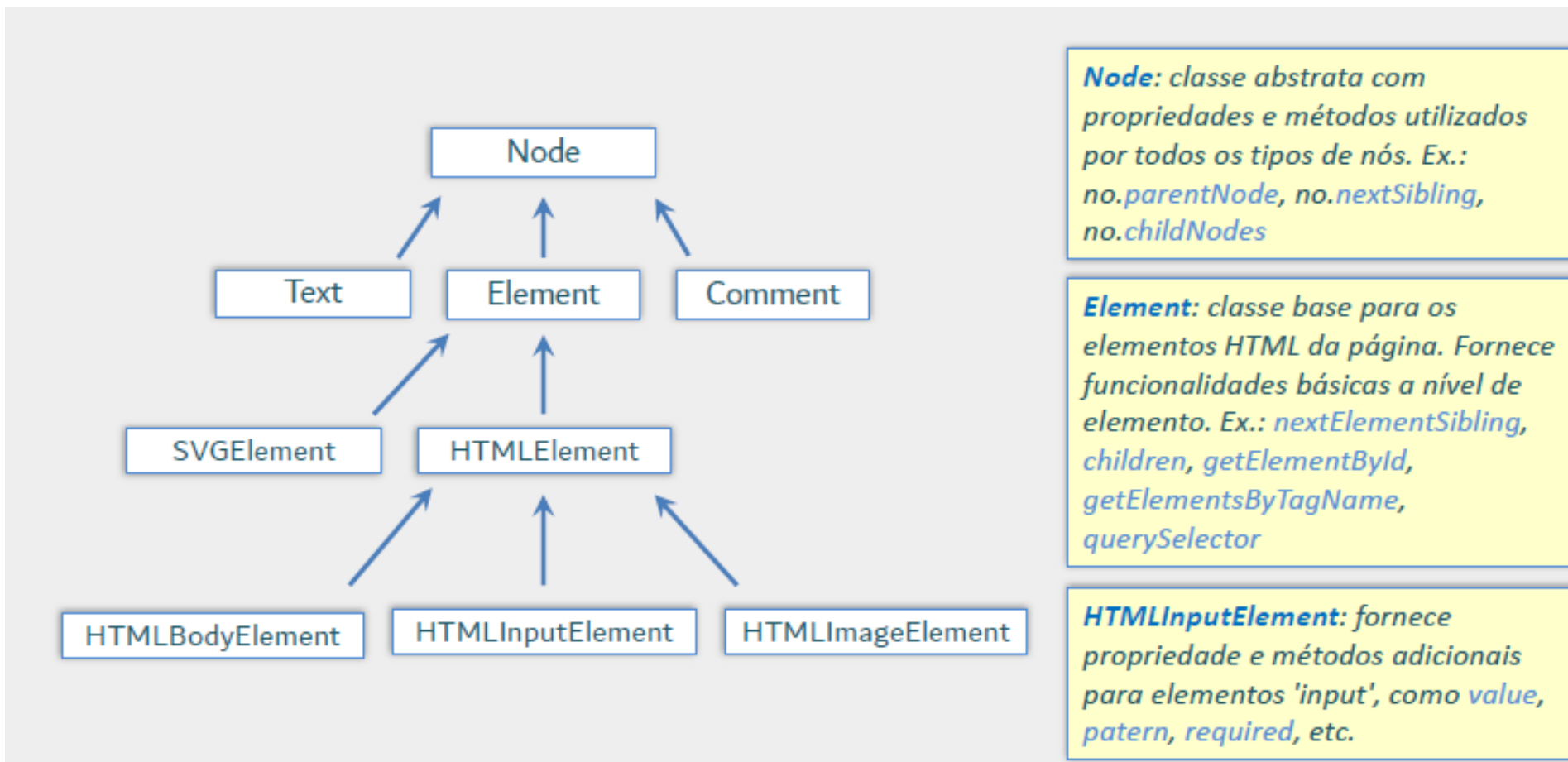
  <body>
    <h1>Primeira Pagina</h1>
    <p>Ola, mundo!</p>
  </body>

</html>
```



Nota: Ao carregar uma página, o navegador percorre o respectivo código HTML e monta uma estrutura de dados internamente denominada **árvore DOM**, que é uma representação em memória de toda a estrutura do documento HTML. Nessa estrutura, cada elemento, comentário ou texto do documento HTML é representado como um objeto, denominado **nó**. A estrutura DOM é utilizada para manipular o documento HTML dinamicamente, utilizando programação, com a **DOM API** e a JavaScript.

Tipos de objetos na Árvore DOM



Hierarquia de nós na estrutura DOM

Nó **Root**: nó representando o elemento raiz `<html>`

Nó **Filho**: nó representando um elemento diretamente dentro de outro

Nó **Pai**: nó representando o elemento que contém o nó filho

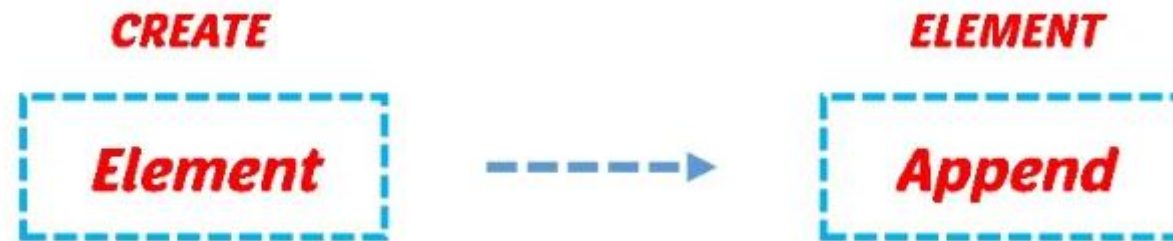
Nós **Irmãos**: nós representando elementos filhos do mesmo pai

Manipulação Árvore DOM

- Criar novos elementos
- Selecionar elementos
- Atualizar elementos
- Remover Elementos

A manipulação da árvore DOM correspondente ao documento HTML é possível graças a uma Web API denominada DOM API, que pode ser utilizada pelo desenvolvedor por meio da linguagem JavaScript e do navegador de Internet.

Criar Elementos



A criação de novos elementos HTML no DOM geralmente ocorre em dois passos:

- Criação do elemento
- Adicionar o elemento criado no DOM (em outro elemento)

Exemplo criação de elemento

```
<body></body>
<script>
  var el = document.createElement('h1')
  el.innerHTML = 'Criação de elemento'
  document.body.appendChild(el)
  console.log(el)
</script>
```

- O exemplo ao lado, cria um elemento <h1> pelo JS e adiciona no <body>

Sintaxe para criar elementos

```
var el = document.createElement('new_el');
```



**or Random DOM
element**



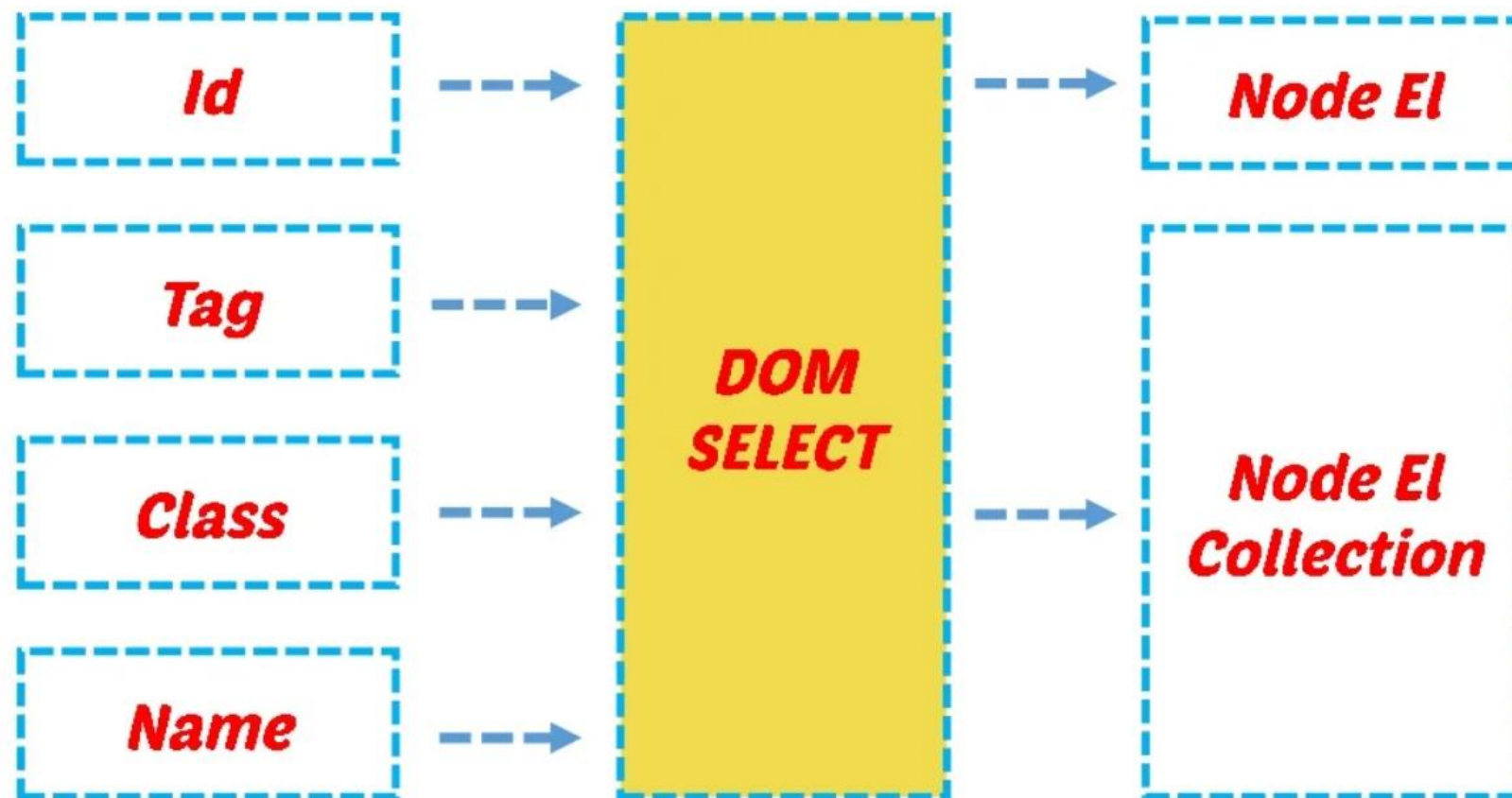
```
document.body.appendChild('new_el');
```

Selecionar Elementos

- Podemos selecionar elementos de várias maneiras:

- ID
- TAG
- Classe
- Nome

Dependendo do caso, cada forma de seleção retorna um nó, ou vários nós em forma de arrays(Node element Collection)



Selecionar por ID

Single Nodel El



```
var el = document.getElementById('el');
```

A seleção por ID retorna um único elemento, que tenha o id passado por parâmetro.

Selecionar por Classe

Select All Nodel Elements



```
var el = document.getElementsByClassName('el');
```

```
el[0] // get the first element
```

A seleção por classe retorna todos os elementos de uma determinada classe. Retorna um array onde é possível pegar cada elemento pelo índice.

Selecionar por Tag

Select All Nodel Elements



```
var el = document.getElementsByTagName('el');
```

```
el[0] // get the first element
```

A seleção por Tag retorna todos os elementos de uma determinada Tag. Retorna um array onde é possível pegar cada elemento pelo índice.

Selecionar por Name

Select All Nodel Elements



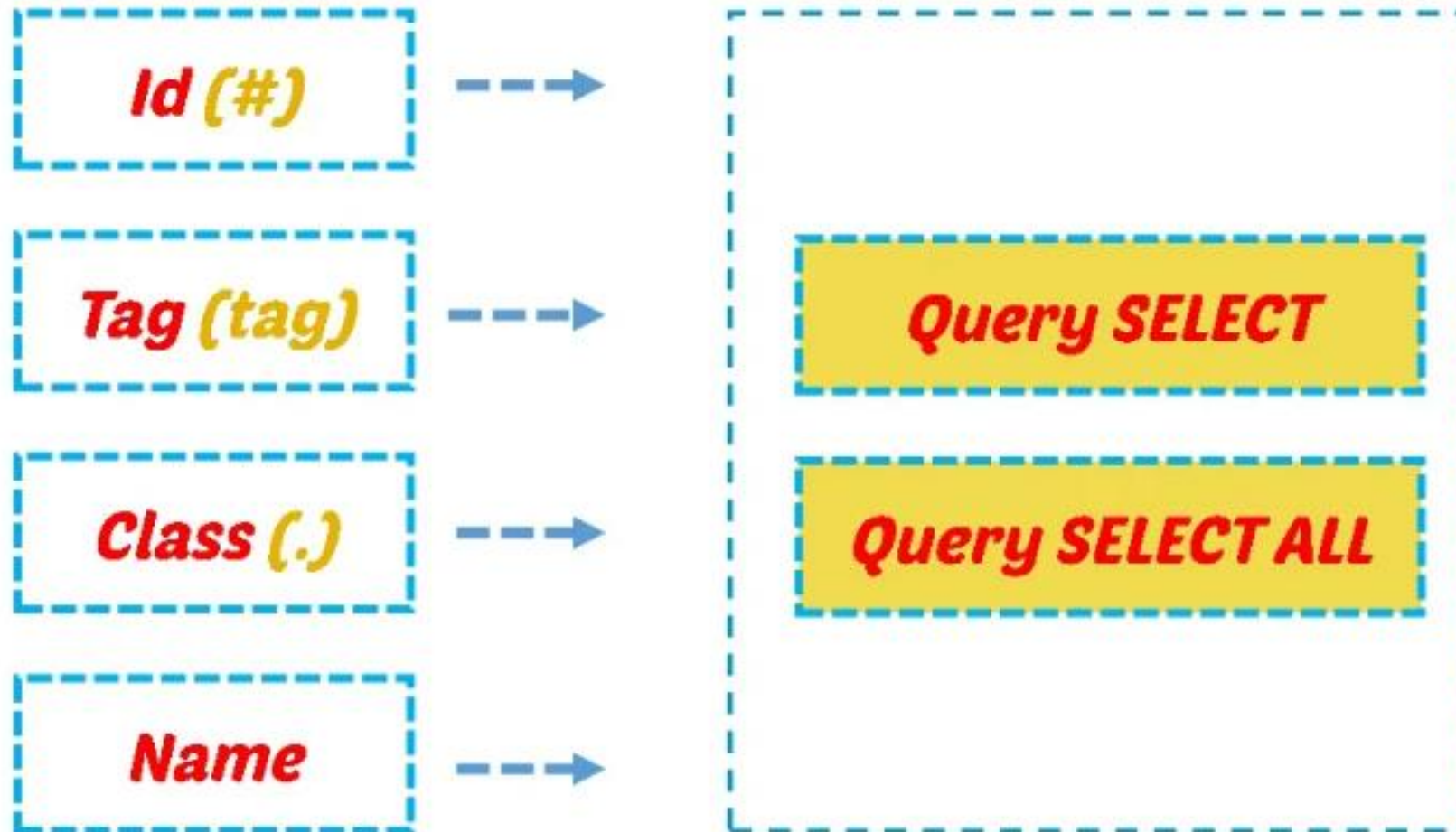
```
var el = document.getElementsByName('el');
```

```
el[0] // get the first element
```

A seleção pelo atributo name retorna todos os elementos que tem um determinado valor name. Retorna um array onde é possível pegar cada elemento pelo índice.

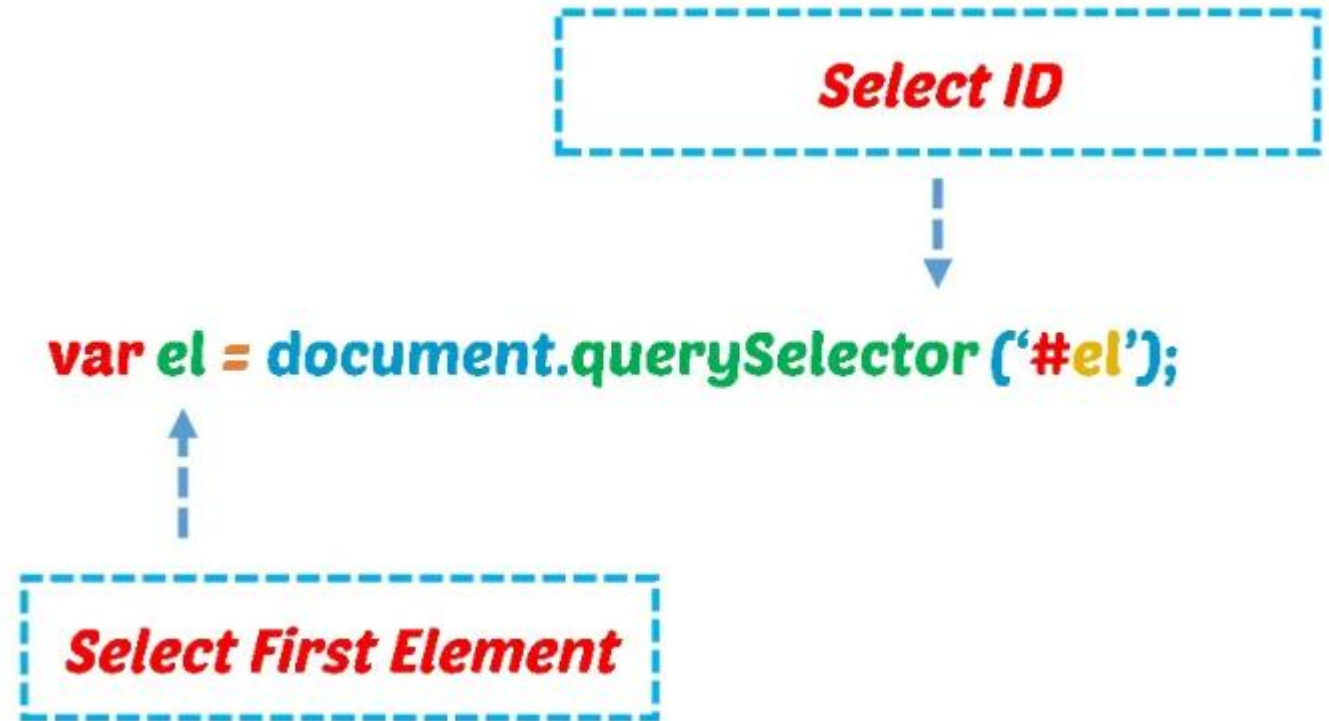
Selecionando por query

Também é possível selecionar elementos por query's CSS.

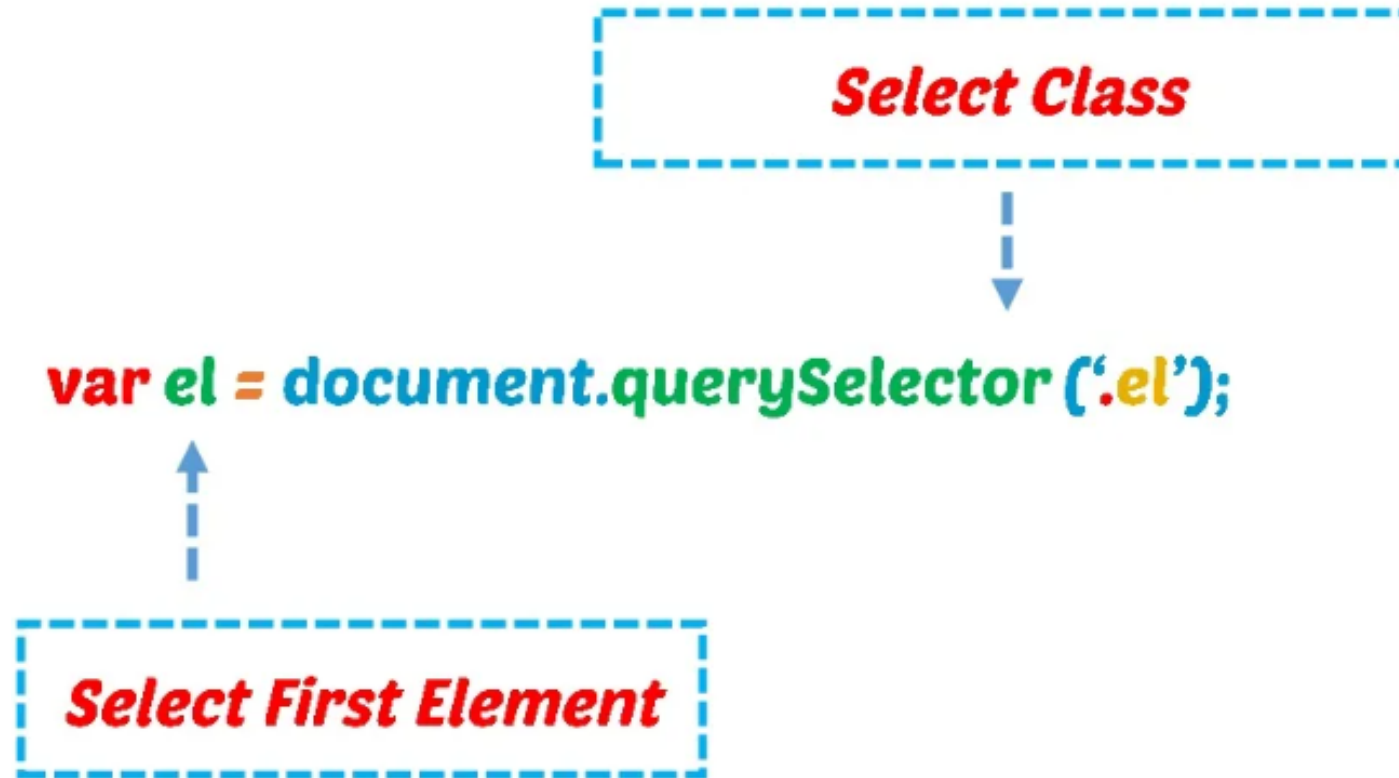


Selecionando por query::id

- O exemplo ao lado seleciona o elemento pelo id. Observe que esse é o mesmo seletor utilizado no CSS.
- A função `querySelector()` sempre retorna o primeiro elemento



Selecionando por query::class



Seleciona o primeiro elemento com determinada classe.

Selecionando por query::class



Neste caso, `querySelectorAll()`, seleciona TODOS os elementos da classe. Os elementos podem ser acessados pelos índices da coleção.

Atualizar Elemento

Para atualizar um elemento temos as seguintes formas

```
element.innerHTML = "HTML code";
```

Atualiza o conteúdo entre(inner) as tags do elemento. Podemos adicionar tags a um elemento dessa forma, ou com o `appendChild()`

```
element.getAttribute("attribute");
```

```
element.setAttribute("attribute");
```

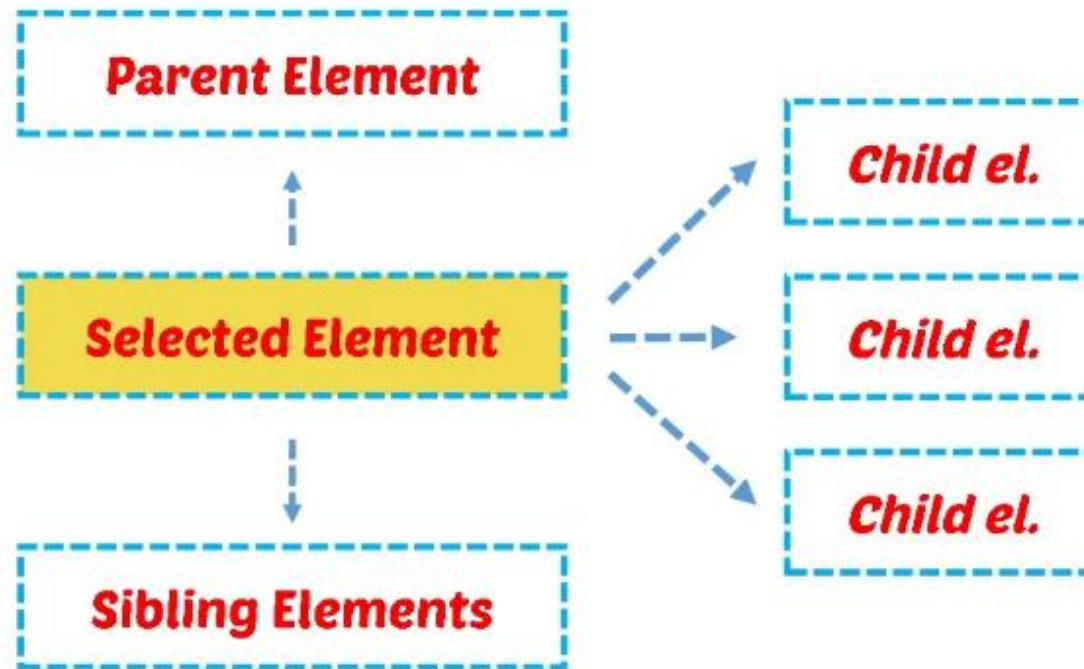
Podemos atualizar um determinado atributo do elemento HTML.

Deletar elemento

A função `removeChild()` serve para remover um elemento filho. Para funcionar, devemos selecionar o pai e o filho



Pegar elementos por hierarquia



A partir de um elemento, podemos selecionar seu:

- Pai (Parent Element)
- Filhos (Child elements)
- Irmãos (Sibling elements)

Seletores por hierarquia

Selecionando o elemento PAI

```
var el = element.parentNode;
```

Selecionando os filhos

```
var el = element.children;
```

Selecionando os irmãos (próximo e anterior)

```
var el = element.nextElementSibling;
```

```
var el = element.previousElementSibling;
```