BRUNA VIEIRA

Desenvolvedora

Experiência com Front-End e Back-end.
Principais linguagens: JavaScript, Angular, React, Java.

VARIÁVEIS

tipagem dinâmica escopo local (var) ou global (window)

exemplos

```
var str = "globo.com";
variavelGlobal = "global";
var indice = 122;
var times = ["Flamengo", "Vasco"];
var time = {nome: "Flamengo",
            titulos: 6,
            estado: "RJ"};
```

tipos

```
var num = 22;
var num2 = "22";

num + num2;
// 2222
```

```
num + parseInt(num2, 10);
num.toString()
```

```
num === num2
// false
typeof num2
// “string”
```

undefined

variável que ainda não recebeu nenhum valor

```
var nome;
typeof nome;
// “undefined”

nome === undefined
// true
```

declarando

```
var str = "globo.com",
    indice = 122,
    times = ["Flamengo", "Vasco"],
    time = {nome: "Flamengo",
            titulos: 6,
            estado: "RJ"};
```

legibilidade + otimização

funções () {}

declarando uma função

```
function log (msg) {
  console.log(msg);
}
```

declaração simples

```
var log = function (msg) {
  console.log(msg);
};
```

expressão

```
var App = {
  log: function (msg) {
    console.log(msg);
  }
};
```

método

escopo

```
var a = 0,
    b = 1;

function soma () {
  var a = 2,
      b = 3;
  return a + b;
}

soma();
// 5
```

todo mundo é **OBJETO**

uma **string** é um objeto
um **número** é um objeto
uma **função** é um objeto
um **array** é um objeto
um **boolean** é um objeto

exemplos

```
var str = new String ("globo.net");
str.replace(".net", ".com");

var num = new Number (1);
num.toString();

var opaChefe = new Function ("alert('Opa')");
opaChefe();
opaChefe.prototype;
// Object
```

estrutura de um objeto

```
var objeto =  {
    descricao: "uma coleção de
propriedades com chave e valor",
    "Tipos de valor": "qualquer coisa,
exceto undefined"
};
```

estrutura de um objeto

```
var objeto =  {
    nome: "globo.com",
    url: "www.globo.com",
    keywords: ["TV", "Python"]
};
```

estrutura de um objeto

```
var objeto =  {
    nome: "globo.com",
    funcao: function () {
        alert(this.nome);
    }
};
```

condições

else if

```
function polaridade(x) {

if (condição 1){
return valor
}

else if (condição 2){
return valor
}

else {
return valor
   }
}
```

switch

```
function condicao(x) {
   switch(x){
       case 1:
      resultado
      break;
       case 2:
      resultado
      break;
       case 3:
      resultado
      break;
       default:
       resultado
  }
}
```

loops

for

```
var numeros = 0

for(var i = 0; i < 10; i++){
  console.log(numeros);
  numeros++
}
console.log("Saiu do 'for'")
```

while

```
var numeros = 0

while (numeros < 20){
  console.log(numeros);
  numeros++
}
console.log("Acabou o 'while'!")
```

Exercícios

<h1>

DOM - Document Object Model Com JavaScript

</h1>

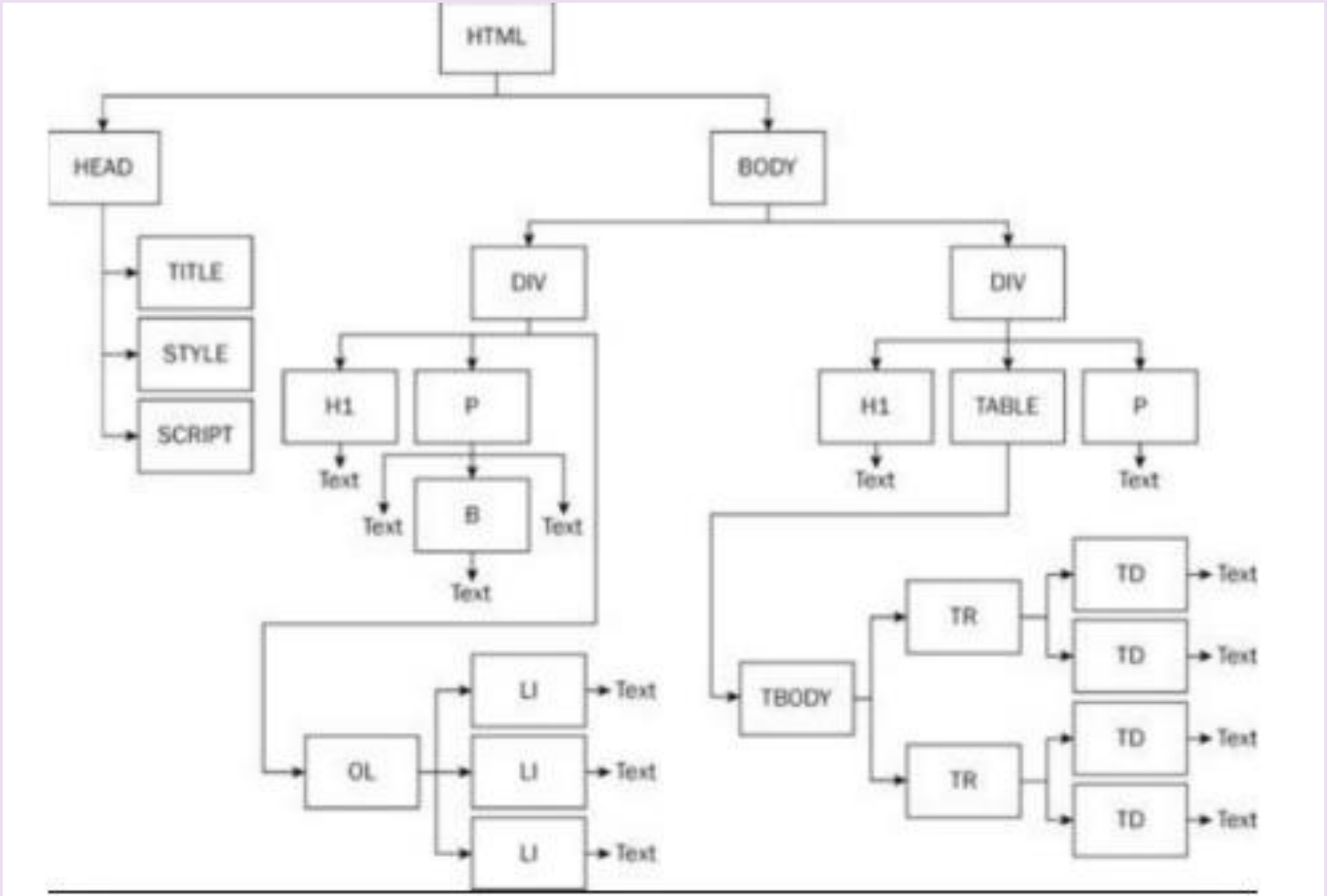
<html>

- DOM – Document Object Model ou Modelo de Objeto de Documento, é uma interface que representa o documento HTML da página web.

- E uma interface de programação para manipulação de documentos HTML, XML e SVG e está disponível para várias linguagens.

- O DOM fornece essa interface através de nós e objetos para que possamos manipular toda nossa página através de métodos e propriedades.

</html>

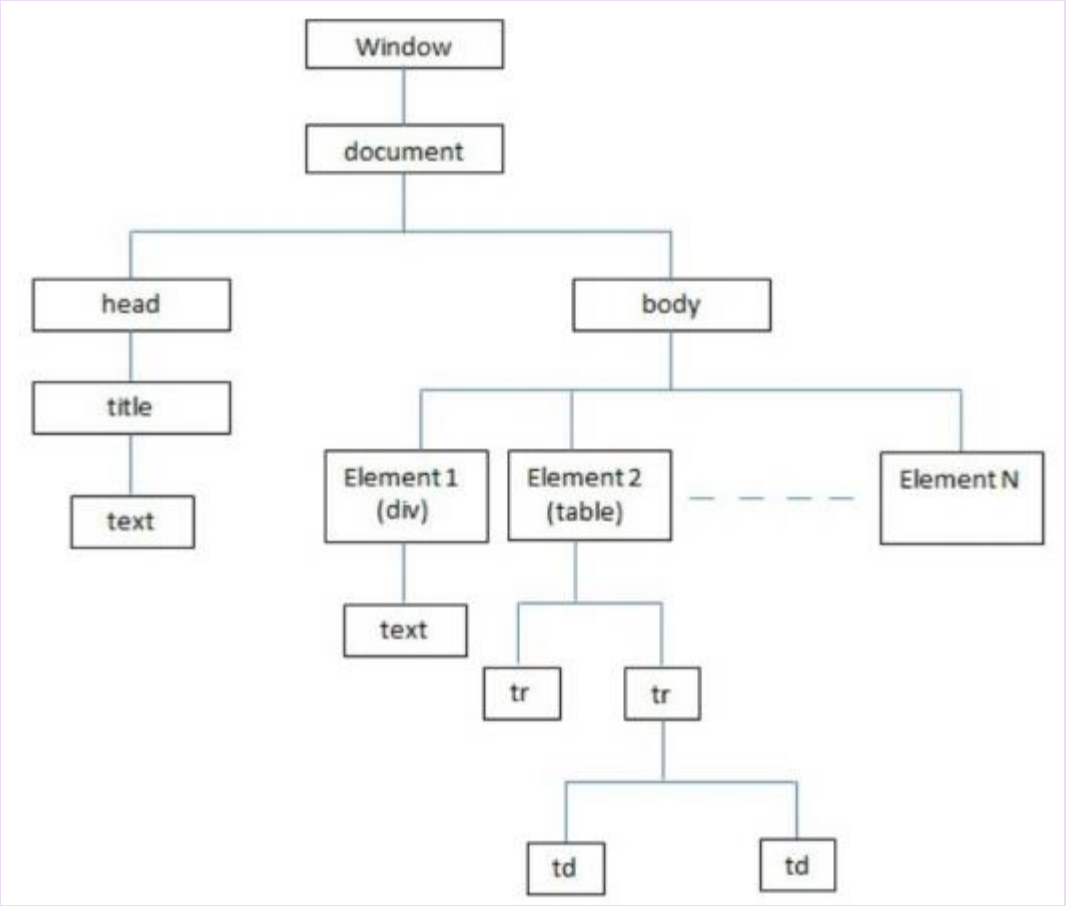

HTML
HEAD
BODY
TITLE
STYLE
SCRIPT
DIV
DIV
H1
P
H1
TABLE
P
Text
Text
B
Text
Text
Text
Text
LI
Text
OL
LI
Text
LI
Text
TBODY
TR
TD
Text
TD
Text
TR
TD
Text
TD
Text

- A hierarquia começa pelo objeto window:

- window.document.body

  OU APENAS

- document.body



Os principais métodos para o entendimento na criação e manipulação de elementos HTML através do DOM:

- getElementById()
- getElementsByClassName()
- getElementsByTagName()
- querySelector()
- querySelectorAll()
- createElement()
- appendChild()
- removeChild()
- parentNode()

- Lembrando que JavaScript é case sensitive isso quer dizer que os métodos do DOM são escritos dessa forma: get**E**lement**B**y**I**d e não getelementbyid.

- Os nomes desses métodos foram assinados com o padrão Camel Case a partir da segunda palavra.

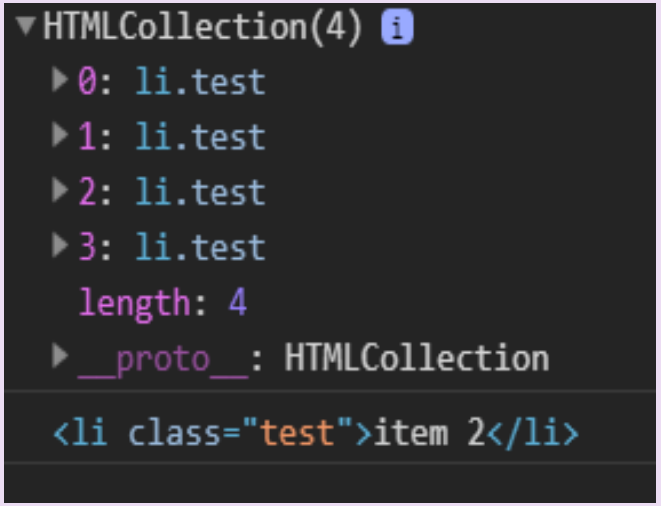
- **getElementById():** pega um elemento através do id e retorna o seu objeto.

```
    <form class="form">
        <label for="name">Nome</label>
        <input type="text" id="nome" value="Bruna">
        <label for="email">Nome</label>
        <input type="text" id="email" value="bruna.vieira.t@hotmail.com">
    </form>

</body>

<script>
    var elemento = document.getElementById('email');
    console.log(elemento);
    console.log(elemento.value);
</script>
```

```
<input type="text" id="email" value="bruna.vieira.t@hotmail.com">
bruna.vieira.t@hotmail.com
```

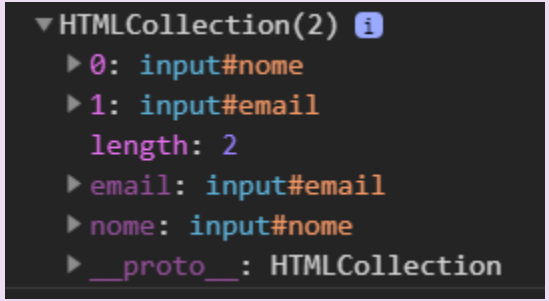
- **getElementsByClassName()**: Seleciona uma coleção de elementos através do nome da classe. Perceba o plural "**Elements**"

```html
<body>
    <ul>
        <li class="test">item 1</li>
        <li class="test">item 2</li>
        <li class="test">item 3</li>
        <li class="test">item 4</li>
    </ul>
</body>

<script>
    var itens = document.getElementsByClassName('test');
    console.log(itens);
    console.log(itens[1]);
</script>
```



- **getElementsByTagName():** retorna os elementos, ou seja, uma coleção, através do nome de uma tag:

```
    <form class="form">
        <label for="name">Nome</label>
        <input type="text" id="nome" value="Bruna">
        <label for="email">Nome</label>
        <input type="text" id="email" value="bruna.vieira.t@hotmail.com">
    </form>
</body>

<script>
    var elemento = document.getElementsByTagName('input');
    console.log(elemento);
</script>
```



```
var elementos = document.getElementsByTagName('input');

for(var pos = 0; pos < elementos.length; pos++){
    console.log("valor do input " + elementos[pos].id.toUpperCase());
    console.log(elementos[pos].value);
}
```

```
valor do input NOME
Bruna
valor do input EMAIL
bruna.vieira.t@hotmail.com
```

- **querySelector():** retorna um elemento, da mesma forma que o getElementById() porém você você utiliza um seletor CSS ao invés do nome do ID.

```
var inputLastChild = document.querySelector('input:last-child');
console.log('last child '+ inputLastChild);
```

```
<input class="form__input" type="text" id="email" value="bruna.vieira.t@hotmail.com">
```

```
var formInput = document.querySelector('.form__input');
console.log(formInput);
```

```
<input class="form__input" type="text" id="nome" value="Bruna">
```

```
var formInputLastChild = document.querySelector('.form__input:last-child');
console.log(formInputLastChild);
```

```
<input class="form__input" type="text" id="email" value="bruna.vieira.t@hotmail.com">
```

- **querySelectorAll():** funciona da mesma forma que o querySelector porém retorna um Array NodeList.

```
<script>
    var elementos = document.querySelectorAll('.form__input');
    console.log(elementos);
</script>
```

```
NodeList(2) [input#nome.form__input, input#email.form__input]
  0: input#nome.form__input
  1: input#email.form__input
  length: 2
  __proto__: NodeList
```

createElement e appendChild

- Esses dois métodos da interface DOM servem para criar elementos e adicioná-los ao DOM.
- É como se fossemos utilizar o innerHTML, porém aqui a ideia é criar cada elemento como um objeto separadamente.
- Imagine que você precisa criar um elemento div qualquer e adicioná-lo em sua página. Com innerHTML fizemos algo como na imagem abaxo:

```
var div = '<div></div>';
document.body.innerHTML = div;
```

- Com create element você cria um fragmento do elemento em memória e logo em seguida utiliza o método appendChild para inserir o elemento de fato no HTML. Exemplo:

```
var div = document.createElement('div');
document.body.appendChild(div);
```


Vamos aos exercícios!


<the end>
Thanks \o/
</the end>