INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação

ENSINO MÉDIO INTEGRADO: INFORMÁTICA
Disciplina de Desenvolvimento Web
Aula 08: DOM / Eventos / JavaScript Fundamentos

Gil Eduardo de Andrade

FUNDAMENTOS - LINGUAGEM JAVASCRIPT

(https://www.w3schools.com/js/default.asp)
(https://developer.mozilla.org/pt-BR/docs/Web/JavaScript)

A Linguagem JavaScript

JavaScript é uma linguagem de programação de alto nível, interpretada e orientada a objetos que, desde sua criação em 1995 por Brendan Eich na Netscape, revolucionou o desenvolvimento web. Originalmente concebida para adicionar interatividade às páginas web estáticas, o JavaScript evoluiu para se tornar uma das linguagens mais populares e versáteis do mundo da programação.

Ao longo dos anos, o JavaScript transcendeu seu propósito inicial de ser apenas uma linguagem de script para navegadores. Hoje, com o advento de ambientes como o Node.js, é possível utilizar JavaScript tanto no frontend quanto no backend, permitindo o desenvolvimento de aplicações completas utilizando uma única linguagem. Esta característica, conhecida como "JavaScript Everywhere", contribuiu significativamente para a popularidade da linguagem entre desenvolvedores.

O JavaScript é uma linguagem dinâmica, com tipagem fraca e funções de primeira classe, o que significa que as funções são tratadas como qualquer outro valor e podem ser passadas como argumentos, retornadas de outras funções e atribuídas a variáveis. Essas características tornam o JavaScript extremamente flexível e poderoso, permitindo a implementação de diversos paradigmas de programação, como programação funcional, orientada a objetos e procedural.

A evolução do JavaScript é gerenciada pela ECMA International através da especificação ECMAScript. A versão ES6 (ECMAScript 2015) trouxe mudanças significativas para a linguagem, introduzindo recursos como classes, módulos, arrow functions, promises, e muitos outros que melhoraram a legibilidade e a manutenção do código. Desde então, novas versões são lançadas anualmente, cada uma trazendo melhorias e novos recursos.

Organização do Código

Diferentemente da organização utilizada por outras linguagens, tais Java ou PHP, a linguagem JavaScript é organizada em sentença de código, que são linhas de código, e em blocos de código, que são agrupamentos de sentenças. Os blocos de código são especificados, seu início e fim, respectivamente, através da utilização de chaves “{“ - “}”.

Exemplo de Sentença

```JavaScript
console.log("Sentença de Código"); // ';' não é necessário
```

Exemplo de Bloco

```JavaScript
function getHour() {
      let hora = new Date();
      let h = hora.getHours();
      let m = hora.getMinutes();
      let s = hora.getSeconds();
      console.log(`${h}:${m}:${s}`);
}

getHour();
```

Ferramentas de Execução - JS

Os navegadores podem ser utilizados para a execução da linguagem (console do browser - F12), além de algumas plataformas online, que permitem o compartilhamento de código, tais como:

https://codepen.io/pen/
https://jsfiddle.net/

Além das alternativas anteriores também é possível executar JavaScript via terminal de comandos, na sua máquina. Para tal basta ter instalado o Node JS instalado.

$node code.js

OBS.: Plugin Code Runner, VSCode, tecla de atalho: “CTRL + ALT + n”.

Variáveis e Constantes - var | let | const

Variáveis e constantes, em JavaScript, são utilizadas, assim como nas outras linguagens de programação, para armazenamento de dados da aplicação. Existem três formas principais de declarar variáveis em JavaScript: **var**, **let** e **const**.

Utilizando “var”

Permite declarar variáveis que possuem apenas dois escopos possíveis, que podemos chamar de “global” (disponível ao longo de todo o código) e “local” (disponível apenas dentro da função na qual foi declarada). Veja os exemplos:

JavaScript
```
{
      {
            {
                  var v = 10;
            }
      }
}
console.log("Valor = " + v);
```

```
"Valor = 10"
```

JavaScript
```
function getValor() {
      var val = 10;
      return val;
}
console.log("Valor = " + val);
```

```
Uncaught ReferenceError: val is not defined
```

JavaScript
```
var v = 1;
```

```
{
  var v = 2;
  console.log("dentro: " + v);
}

console.log("fora: " + v);
```

```
"dentro: 2"
"fora: 2"
```

OBS.: Evite, sempre, utilizar o escopo "global" com "var", pois ele tende a gerar erros (sobre escrita indesejada de dados) ao longo da implementação.

Utilizando "let" e "const"

O "let" permite declarar variáveis que possuem apenas um escopo possível, que podemos chamar de "local" (disponível apenas dentro do bloco na qual foi declarada). A mesma dinâmica de escopo ocorre com o "const", porém neste caso estamos falando da declaração de constantes, e não variáveis. Veja os exemplos:

JavaScript
```
let v = 1;
{
  let v = 2;
  console.log("dentro: " + v);
}

console.log("fora: " + v);
```

```
"dentro: 2"
"fora: 1"
```

JavaScript
```
const v = 1;
{
```

```
  const v = 2;
  console.log("dentro: " + v);
}

console.log("fora: " + v);
```

```
"dentro: 2"
"fora: 1"
```

Utilizando “var” e “let” em Laço de Repetição

Exemplo “var”

JavaScript
```
for(var i=0; i<10; i++) {
  console.log(i);
}
console.log(i);
```

```
9
10
```

Exemplo “let”

JavaScript
```
for(let i=0; i<10; i++) {
  console.log(i);
}
console.log(i);
```

```
Uncaught ReferenceError: i is not defined
```

Fracamente Tipada

A linguagem JavaScript possui tipagem dinâmica, ou seja, não é necessário especificar o tipo de uma variável quando a mesma é criada. Tal característica permite uma maior flexibilidade durante o desenvolvimento, mas dificulta a identificação e correção de determinados erros (um dos motivos pela utilização, por parte de frameworks e desenvolvedores, do TypeScript).
Importante salientar que, embora a linguagem seja fracamente tipada, isso não significa que ela não tenha tipo, mas sim que não há a necessidade de especificá-lo no momento da criação das variáveis e que esse tipo pode ser alterado ao longo da codificação.

Visualizando os Tipos

JavaScript

```javascript
let val = "Gil Eduardo de Andrade";
console.log(typeof val);

val = 3.1415;
console.log(typeof val);

val = Array(1, 2, 3);
console.log(typeof val);

val = function f() {
    return "Maria";
}
console.log(typeof val);
```

```
string
number
object
function
```

Alguns Tipos em JavaScript
Todos os tipos primitivos no JavaScript, com exceção do “*null*” e do “*undefined*”, são tratados como objetos - eles podem receber propriedades e possuem todas as características de objetos.

Tipo Number

JavaScript

```
const v1 = 1.0;
const v2 = Number('2.5'); // Construtor
const valor = 9.875;

console.log(v1, v2);
console.log(Number.isInteger(v1));
console.log(Number.isInteger(v2));

const total = (valor * v1) / (v1 + v2);
// Variáveis e constantes possuem métodos vinculados
console.log(total.toFixed(2));
console.log(total.toString(2)); // Binário
console.log(typeof total);
console.log(typeof Number);
```

```
1 2.5
true
false
2.82
10.110100100100100100100100100100100100100100100101
number
function
```

Tipo Number - Curiosidades

JavaScript

```
console.log(10 / 0)
console.log("12" / 2)
console.log('12' + 2) // string ganha!
console.log('12' - 2)
console.log("Show!" * 2)
console.log(0.23 + 0.74)
console.log((3.1415).toFixed(2))
```

```
Infinity
6
122
10
NaN
0.97
3.14
```

Objeto Math

```javascript
const r = 10.5;
const area_circulo = Math.PI * Math.pow(r, 2);

console.log(area_circulo.toFixed(3));
console.log(typeof Math);
```

```
346.361
object
```

Tipo String

```javascript
const turma = "INFO23";

console.log(turma.charAt(5));
console.log(turma.charAt(6));
console.log(turma.charCodeAt(2));
console.log(turma.indexOf('3'));

console.log(turma.substring(2));
console.log(turma.substring(0, 2));

console.log('Turma '.concat(turma).concat("!"));
console.log('Turma ' + turma + "!");
console.log(turma.replace('I', 1));

console.log('Gil,Eduardo,Andrade'.split(','));
```

```
3

70
5
FO23
IN
Turma INFO23!
Turma INFO23!
1NFO23
[ 'Gil', 'Eduardo', 'Andrade' ]
```

Template String

```javascript
const nome = 'Gil Eduardo'
const concatenacao = 'Olá ' + nome + '!'
const template = `Olá ${nome}!`
console.log(concatenacao, template)

// expressoes...
console.log(`10 + 20 = ${10 + 20}`)

const up = (texto) => texto.toUpperCase()
console.log(`${up('danger')}!`)
```

```
Olá Gil Eduardo! Olá Gil Eduardo!
10 + 20 = 30
DANGER!
```

TIpo Boolean

```javascript
let op = false;
console.log(op);
op = true;
console.log(op);
op = 1;
console.log(!!op);

console.log('TRUE:')
console.log(!!3)
console.log(!!-1)
console.log(!!' ')
console.log(!!'texto')
console.log(!![])
console.log(!!{})
console.log(!!Infinity)
console.log(!!(isAtivo = true))

console.log('FALSE:')
console.log(!!0)
```

```
console.log(!!'')
console.log(!!null)
console.log(!!NaN)
console.log(!!undefined)
console.log(!!(op = false))

console.log('OUTRO:')
console.log(!!('' || null || 0 || ' '))
```

```
false
true
true
TRUE:
true
true
true
true
true
true
true
true
FALSE
false
false
false
false
false
false
OUTRO
true
```

Tipo Array

JavaScript

```
const valores = [7.7, 8.9, 6.3, 9.2];
console.log(valores[0], valores[3]);
console.log(valores[4]);

valores[4] = 10;
console.log(valores);
console.log(valores.length);
```

```
valores.push({id: 3}, false, null, 'teste');
console.log(valores);

console.log(valores.pop());
delete valores[0];
console.log(valores);

console.log(typeof valores);
```

```
7.7 9.2
undefined
[ 7.7, 8.9, 6.3, 9.2, 10 ]
5
[
  7.7,     8.9,
  6.3,     9.2,
  10,      { id: 3 },
  false,   null,
  'teste'
]
teste
[ <1 empty item>, 8.9, 6.3, 9.2, 10, { id: 3 }, false, null ]
object
```

Tipo Objeto

JavaScript

```
// OBJETO PESSOA
const pessoa = {}
pessoa.nome = 'Maria Eduarda'
pessoa.altura= 1.70
// Indice de Massa Corporal
pessoa['imc'] = pessoa.altura / Math.pow(pessoa.altura, 2);

console.log(pessoa)

// OBJETO CIDADE
const cidade = {
    nome: 'Paranaguá',
    habitantes: 183.450
}
```

```
console.log(cidade)
```

```
{ nome: 'Maria Eduarda', altura: 1.7, imc: 0.5882352941176471 }
{ nome: 'Paranaguá', habitantes: 183.45 }
```

OBS.: Importante observar que JSON (*JavaScript Object Notation*) e Objeto JavaScript são conceitos diferentes. O JSON é um padrão de formatação utilizado para estruturar dados no formato texto e transmiti-los entre diferentes sistemas. Já os Objetos JavaScript são variáveis capazes de armazenar uma "coleção de valores", referenciados por nome (chave + valor).

Undefined | NULL

JavaScript

```javascript
let valor; // não inicializada
console.log(valor);

valor = null; // ausência de valor
console.log(valor);

const produto = {};
console.log(produto.preco);
console.log(produto);

produto.preco = 3.50;
console.log(produto);

produto.preco = undefined; // evite atribuir undefined
console.log(!!produto.preco);
// delete produto.preco
console.log(produto);

produto.preco = null; // sem preço
console.log(!!produto.preco);
console.log(produto);
```

```
undefined
null
undefined
{}
{ preco: 3.5 }
false
{ preco: undefined }
false
{ preco: null }
```

Funções

Quase Tudo é Função

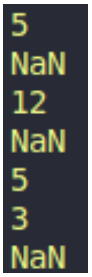
JavaScript
```
console.log(typeof Object);

class Pessoa {};
console.log(typeof Pessoa)
```

```
function
function
```

Função - Com e Sem Retorno

JavaScript
```
// Sem Retorno
function printAdd(a, b) {
    console.log(a + b);
}

printAdd(2, 3);
printAdd(2); // NaN - Not a Number
printAdd(2, 10, 4, 5, 6, 7, 8);
printAdd();  // NaN - Not a Number

// Com Retorno
function add(a, b = 1) {
    return a + b;
}
```

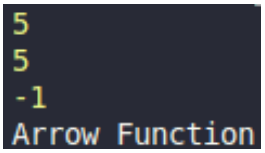
```
console.log(add(2, 3));
console.log(add(2));
console.log(add());
```



Função - Anônima e Seta

JavaScript

```
// Anonima (Anonimous)
const printAdd = function (a, b) {
    console.log(a + b);
}
printAdd(2, 3);

// Seta (Arrow) - Retorno Explícito
const add = (a, b) => {
    return a + b;
}

console.log(add(2, 3));

// Seta (Arrow) - Retorno Implícito
const sub = (a, b) => a - b;
console.log(sub(2, 3));

const print = a => console.log(a);
print('Arrow Function');
```



Par - Chave : Valor

Contexto/Escopo Léxico

```javascript
// chave / valor
const cumprimento = 'Oi'; // contexto/escopo léxico 1

function exec() {
    const cumprimento = 'Olá'; // contexto/escopo léxico 2
    return cumprimento;
}

// Objetos: grupos de chave / valor
const pessoa = {
    nome: 'Mariana',
    idade: 41,
    peso: 65,
    endereco: {
        logradouro: 'Rua Santa Catarina',
        numero: 427
    }
}

console.log(cumprimento);
console.log(exec());
console.log(pessoa);
```

```
Oi
Olá
{
  nome: 'Mariana',
  idade: 41,
  peso: 65,
  endereco: { logradouro: 'Rua Santa Catarina', numero: 427 }
}
```

OBS.: o contexto ou escopo léxico de uma variável é o local onde ela foi criada. Ele é determinado pela estrutura textual (léxica) do programa, podendo, as variáveis, serem declaradas em um escopo específico, sendo acessíveis somente dentro deste local.

Destructuring

Desestruturação: Objeto

JavaScript

```javascript
const pessoa = {
    nome: 'Carlos',
    idade: 12,
    endereco: {
        logradouro: 'Rua Santos Dummont',
        numero: 984
    }
}

const { nome, idade } = pessoa;
console.log(nome, idade);

const { nome: n, idade: i } = pessoa;
console.log(n, i);

const { sobrenome, feliz = true } = pessoa;
console.log(sobrenome, feliz);

const { endereco: { logradouro, numero, cep } } = pessoa;
console.log(logradouro, numero, cep);
```

```
Carlos 12
Carlos 12
undefined true
Rua Santos Dummont 984 undefined
```

Desestruturação: Array

JavaScript

```
const [x] = [12];
console.log(x);

const [n1, , n3, , n5, n6 = 0] = [6.5, 7.5, 4.2, 9.7];
console.log(n1, n3, n5, n6);

const [, [, nota]] = [[, 6.7, 8.3], [9.2, 3.7, 8.6]];
console.log(nota);
```

```
12
6.5 4.2 undefined 0
3.7
```

Desestruturação: Função + Objeto

JavaScript

```
function aleatorio({ min = 1, max = 60 }) {
    const valor = (Math.random() * (max - min)) + min
    return Math.floor(valor)
}

const obj = { max: 50, min: 40 }
console.log(aleatorio(obj))
console.log(aleatorio({ min: 55 }))
console.log(aleatorio({}))
console.log(aleatorio())
```

```
48
55
21
/media/g1l3du4rd0/GEA/IFPR/2025/DW/Front/javascript/Fudamentos/
function aleatorio({ min = 1, max = 60 }) {
                     ^

TypeError: Cannot read properties of undefined (reading 'min')
```

OBS.: A função ***Math.random()*** retorna números pseudo aleatórios entre 0 e 1. Já a função ***Math.floor()*** arredonda um número para o inteiro mais próximo, menor ou igual ao valor original.

Desestruturação: Função + Array

JavaScript
```
function aleatorio([ min = 1, max = 60]) {
    const valor = (Math.random() * (max - min)) + min;
    return Math.floor(valor);
}

const arr = [10, 20];
console.log(aleatorio(arr));
console.log(aleatorio([55]));
console.log(aleatorio([, 5]));
console.log(aleatorio([]));
```

```
15
58
3
29
```

Operadores Relacionais

Comparação entre Valores / Tipos

JavaScript
```
console.log('01)', '12' == 12);
console.log('02)', '12' === 12);
console.log('03)', '12' != 12);
console.log('04)', '12' !== 12);

console.log('05)', 6 < 3);
console.log('06)', 6 > 3);
console.log('07)', 6 <= 3);
console.log('08)', 6 >= 3);
```

```
console.log('09)', undefined == null);
console.log('10)', undefined === null);
```

```
01) true
02) false
03) false
04) true
05) false
06) true
07) false
08) true
09) true
10) false
```

Operadores Ternários

Condicional Se - Senão

JavaScript

```
const media = 5.6;
media >= 7 ? console.log('Aprovado') :
console.log('Reprovado');

// Arrow Function
const resultado = (media) => media >= 7 ? 'Aprovado' :
'Reprovado';
console.log(resultado(7.1))
console.log(resultado(6.7))
```

```
Reprovado
Aprovado
Reprovado
```

Tratamento de Erros

try - catch - finally

JavaScript

```
function handleError(e) {
    // throw 10;
    // throw true;
    // throw 'mensagem';
    throw {
        nome: e.name,
        msg: e.message,
        date: new Date
    }
}

function printName(obj) {
    try {
        console.log(obj.name.toUpperCase() + '!!!');
    } catch (e) {
        // console.log(e.message);
        handleError(e);
    } finally {
        console.log('Terminou!!');
    }
}

const pessoa = { nome: 'Carlos Eduardo' };
printName(pessoa);
```

```
Terminou!!

/media/g1l3du4rd0/GEA/IFPR/2025/DW/Front/javascript/Fudamentos/try-catch.js:5
    throw {
    ^
{
  nome: 'TypeError',
  msg: "Cannot read properties of undefined (reading 'toUpperCase')",
  date: 2025-01-14T20:39:56.403Z
}
```

Laços de Repetição - forEach e for(in)

Laço forEach() - Array

```javascript
const arr = [1, 2, 3, 4, 5];
arr.forEach((item) => console.log(item));

const parImpar = function(item) {
    if(item%2 == 0) console.log("PAR");
    else console.log("ÍMPAR");
}
arr.forEach(parImpar);
```

```
1
2
3
4
5
ÍMPAR
PAR
ÍMPAR
PAR
ÍMPAR
```

OBS.: O método *"forEach()"* executa uma função para cada elemento do array em questão.

Laço for(in) - Objeto

```javascript
// Define um Objeto Aluno
const aluno = { nome: "Carlos", curso: "INFO", turma: 2023 };

for (prop in aluno) {
  console.log("aluno." + prop + " = " + aluno[prop]);
}
```

```
aluno.nome = Carlos
aluno.curso = INFO
aluno.turma = 2023
```

OBS.: o laço *"for(in)"* interage sobre propriedades enumeradas de um objeto, na ordem original de inserção, podendo ser executado para cada propriedade distinta do objeto.

Async/Await e Promises

A programação assíncrona é uma parte fundamental do JavaScript moderno, especialmente para operações que podem levar tempo, como requisições de rede, leitura de arquivos ou consultas a bancos de dados. Existem dois mecanismos essenciais para lidar com código assíncrono em JavaScript: Promises e a sintaxe Async/Await

Promises

As *Promises* (promessas) foram introduzidas no ***ES6*** (ECMAScript 2015) e são objetos que representam o eventual término (com sucesso ou falha) de uma operação assíncrona. Elas oferecem uma maneira mais elegante de lidar com operações que levam tempo para serem concluídas, tais como requisições HTTP ou operações de I/O. Uma promessa pode conter os seguintes estados:

- *Pending*: A operação ainda está em andamento.
- *Fulfilled*: A operação foi concluída com sucesso.
- *Rejected*: A operação falhou.

Promises: Exemplo Básico

JavaScript

```javascript
const myPromise = new Promise((resolve, reject) => {
  // Simulando uma operação assíncrona
  setTimeout(() => {
    resolve('Operação concluída com sucesso!');
  }, 2000);
});

// Consumindo a Promisse
myPromise.then((resultado) => {
  console.log("Sucesso:", resultado);
}).catch((erro) => {
  console.log("Erro:", erro);
});

console.log('Código Continua...')
```

```
Código Continua...
Sucesso: Operação concluída com sucesso!
```

Async/Await

Async/Await são palavras reservadas, da linguagem JavaScript, que tornam o código assíncrono mais parecido com código síncrono, facilitando a sua leitura e escrita.

async

Introduzido no ES2017 (***ES8***), Async/Await é uma sintaxe que facilita o trabalho com Promises, tornando o código assíncrono mais parecido com código síncrono tradicional. A utilização do *async* permite definir que um trecho de código (função) deve ser executado de maneira assíncrona. Normalmente a palavra reservada *await* é utilizada dentro do trecho de código assíncrono, que sempre retorna como resultado uma *Promise*.

Async / Promise - Comparação

JavaScript

```javascript
async function minhaFuncaoAsync() {
      return "Olá, mundo!";
}

// Equivalente a:
function minhaFuncaoPromise() {
      return Promise.resolve("Olá, mundo!");
}
```

await

A utilização do *await* permite pausar a execução de um trecho de código (função) assíncrona até que o resultado (*Promise*) seja resolvido.

Async / Await: Exemplo Básico

JavaScript

```javascript
async function fetchData() {
  const response = await fetch('https://api.example.com/data');
  const data = await response.json();
  return data;
```

```
}

async function fetchDataUser() {
  const response = await fetch('https://randomuser.me/api/?results=1000');
  const data = await response.json();
  console.log(data)
}

// fetchData()
fetchDataUser()
console.log('Código Continua...')
```

Funcionamento *async* + *await*

- *async*: marca uma função como assíncrona, permitindo o uso de await.
- *await*: pausa a execução da função até que uma *Promise* seja resolvida, permitindo tratar o resultado como se fosse um valor síncrono.

Benefícios *async* + *await*

- Código mais legível: se assemelha ao código síncrono, facilitando a compreensão.
- Gerenciamento de erros: cláusula *try{}catch()* pode ser utilizada para tratar erros de forma mais clara.

Resumindo

- *Promises*: representam o eventual resultado de uma operação assíncrona.
- *Async/Await*: simplificam a utilização de *Promises*, tornando o código mais legível e fácil de entender.
- *Benefícios*: código mais limpo, tratamento de erros mais fácil e melhor organização.

Funções JavaScript

Como comentado anteriormente, uma função JavaScript é um bloco de código criado para executar uma tarefa específica, sendo executada no momento em que é invocada. Uma função é definida através do uso da palavra-chave ***"function"***, seguida por um nome e posteriormente por parênteses ( ). Os parênteses podem incluir nomes de parâmetros separados por vírgulas: (parâmetro1, parâmetro2, ...).

Os parâmetros da função são listados dentro dos parênteses () na definição da função, já os argumentos são os valores recebidos quando a função é invocada. Dentro da função, os argumentos (os parâmetros) se comportam como variáveis locais.

Parâmetros e Retorno Opcionais

JavaScript

```
function getStatus(imc) {

    if(isNaN(imc)) return "INDEFINIDO";
    else if(imc < 18.5)
        return "MAGREZA";
    else if(imc < 25)
        return "NORMAL";
    else if(imc < 30)
        return "SOBREPESO";
    else if(imc < 40)
        return "OBESIDADE";
    else
        return "OBESIDADE GRAVE";
}

function getIMC(altura, peso) {

    const imc = peso / (altura*altura);
    let ret = {
        imc: imc.toFixed(2),
        msg: getStatus(imc)
    };

    return ret;
}

console.log(getIMC(1.70, 61.5));
console.log(getIMC(1.70));
console.log(getIMC());
console.log(getIMC(1.80, 135.7, 0, 10, 19));
console.log(getIMC(1.65, 95.4));
console.log(getIMC(1.80, 45.4));
```

```
{ imc: '21.28', msg: 'NORMAL' }
{ imc: 'NaN', msg: 'INDEFINIDO' }
{ imc: 'NaN', msg: 'INDEFINIDO' }
{ imc: '41.88', msg: 'OBESIDADE GRAVE' }
{ imc: '35.04', msg: 'OBESIDADE' }
{ imc: '14.01', msg: 'MAGREZA' }
```

Número de Parâmetros Variável

JavaScript

```
function soma() {
    let soma = 0;
    for(i in arguments) {
        soma += arguments[i];
    }
    return soma;
}

console.log(soma());
console.log(soma(1));
console.log(soma(1.1, 2.2, 3.3));
console.log(soma(1.1, 2.2, "Teste"));
console.log(soma('a', 'b', 'c'));
```

```
0
5
6.283
3.3000000000000003Teste
0abc
```

Número de Parâmetros Variável - rest

JavaScript

```
function soma(...pars) {
    let soma = 0;
    pars.forEach( (par) => soma += par);
    return soma;
}

console.log(soma());
console.log(soma(5));
```

```
console.log(soma(3.1415, 0, 3.1415));
console.log(soma(1.1, 2.2, "Teste"));
console.log(soma('a', 'b', 'c'));
```

```
0
5
6.283
3.3000000000000003Teste
0abc
```

Valor Padrão com Número de Parâmetros Variável

JavaScript

```
function somaC(a = 1, b = 1, c = 1) {
    return a + b + c;
}

console.log(somaC(), somaC(3), somaC(1, 2, 3), somaC(0, 0, 0));
```

```
3 5 6 0
```

Arrow Function - Básico

JavaScript

```
let area = function (raio) {
    return Math.PI * Math.pow(raio, 2);
}
console.log(area(4.25));

dobro = (raio) => {
    return Math.PI * Math.pow(raio, 2);
}
console.log(area(4.25));

dobro = raio => Math.PI * Math.pow(raio, 2); // return implícito
console.log(area(4.25));
```

```
let mensagem = function () {
    return 'Olá Mundo!'
}
console.log(mensagem());

mensagem = () => 'Olá Mundo!'
console.log(mensagem());
```

Arrow Function - Como Parâmetro

```
JavaScript
function Carro() {
    this.velocidade = 0;
    setInterval(() => {
        this.velocidade+=10;
        console.log(this.velocidade);
    }, 1000);
}

new Carro;
```

OBS.: a função `setInterval()`, em JavaScript, permite que uma função seja executada repetidamente, em intervalos de tempo específicos. Ela se comporta como um cronômetro, que dispara a função especificada a cada intervalo de tempo definido.

```
setInterval(function, tempo);
```

- `function:` função que deseja executar repetidamente;
- `tempo:` intervalo de tempo (milissegundos) entre cada execução da função;

Função Anônima

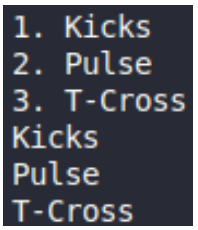
```
JavaScript
const parImpar = function (x) {
```

```
    if(x%2 == 0)
        return "PAR";

    return "ÍMPAR";
}
const fatorial = function (x) {
    let fat = 1;
    for(let i=2; i<=x; i++) fat *= i;
    return fat;
}
const print = function (a, operacao = parImpar) {
    console.log(operacao(a));
}
print(4);
print(7);
print(3, fatorial);
print(5, fatorial);
print(8, function(a) { return a * a; } );
print(10, (a) => a + a);

const atleta = {
    correr: function() { console.log('Running...') } ,
    parar: () => console.log('Stoping...')
}
atleta.correr();
atleta.parar();
```

```
PAR
ÍMPAR
6
120
64
20
Running...
Stoping...
```

Função de Callback

```javascript
const carros = ["Kicks", "Pulse", "T-Cross"];

function imprimir(nome, indice) {
    console.log(`${indice + 1}. ${nome}`);
}

carros.forEach(imprimir);
carros.forEach(nome => console.log(nome));
```



OBS.: uma função de callback, em JavaScript, é uma função que é passada como argumento para outra função, sendo utilizadas, tanto do lado do cliente, quanto do servidor. Elas são fundamentais para operações assíncronas, facilitando a manipulação de dados e a interação com o usuário.

Função Construtora

```javascript
function Carro(velMax = 200, delta = 5) {
    // Atributo privado
    let vel = 0;

    // Método publico (this)
    this.acelerar = function () {
        if (vel + delta <= velMax)
            vel += delta;
        else
            vel = velMax;
    }

    // Metodo publico (this)
    this.getVel = function () {
        return vel;
```

```
    }
}

const gol = new Carro;
gol.acelerar();
console.log(gol.getVel() + " km/h");

const abarth = new Carro(350, 40);
abarth.acelerar();
abarth.acelerar();
abarth.acelerar();
console.log(abarth.getVel() + " km/h");

console.log(typeof Carro);
console.log(typeof Abarth);
```

```
5 km/h
120 km/h
function
object
```

OBS.: as funções construtoras em JavaScript são como classes em outras linguagens (Java), se diferenciando apenas pela sintaxe. No que diz respeito ao funcionamento, tanto as funções construtoras, no JavaScript, quanto as Classes, no Java, possuem a mesma utilidade: servir como modelo para a criação de objetos.

Factory - Básico

JavaScript

```
function criarAluno() {
    return {
        nome: 'Evelyn',
        turma: 'MEC',
        ano: 2024,
    }
}

console.log(criarAluno());
```

OBS.: basicamente, uma factory, em JavaScript, é uma função que retorna (fabrica) um objeto, permitindo a uma nova instância seja criada toda vez em que é invocada. Esse procedimento evita que vários procedimentos de copiar e colar sejam efetuados para se obter o mesmo resultado.

OBS.: Factory Method é um padrão de projeto (Design Patterns) utilizado em outras linguagens de programação.

*__Design Patterns (Padrões de Projetos)__: são utilizados para encapsular as melhores práticas e soluções testadas para problemas comuns no mundo do desenvolvimento de software, fornecendo um método organizado para escrita de código confiável e sustentável. A seguir são apresentadas as principais vantagens do uso de Padrões de Projetos:

- Reutilização de código: promovem a reutilização de soluções bem-sucedidas, reduzindo a necessidade de reinvenção das mesmas e acelerando o processo de desenvolvimento;
- Escalabilidade: fornecem uma base para código escalável e extensível, possibilitando que as aplicações cresçam e se adaptem às mudanças de requisitos;
- Manutenibilidade: o código se torna mais modular, fácil de compreender, depurar e manter ao longo do tempo.

Factory - Personalizando Parâmetros

JavaScript

```
function criarAluno(nome, turma, ano) {
    return {
        nome,
        turma,
        ano
    }
}

console.log(criarAluno('Thomas', 'MAMB', 2022));
console.log(criarAluno('Ilana', 'PROD', 2025));
```

```
{ nome: 'Thomas', turma: 'MAMB', ano: 2022 }
{ nome: 'Ilana', turma: 'PROD', ano: 2025 }
```

Objetos JavaScript

Na vida real, objetos são coisas como: casas, carros, pessoas, animais ou quaisquer outros assuntos. Aqui está um exemplo de objeto carro:

PROPRIEDADES	MÉTODOS
nome	ligar
modelo	acelerar
cor	freiar

As variáveis JavaScript são contêineres que podem conter um valor. Os objetos também são variáveis, contudo podem conter muitos valores.

Formas de Criação de Objetos

```JavaScript
// Notação Literal
const obj = {};
console.log(obj);

// Object em JS
const obj1 = new Object;
console.log(obj1);

// Função Construtora
function Carro(modelo, preco, desconto) {
    this.modelo = modelo;
    this.getPreco = () => {
        return preco * (1 - desconto);
    }
}
```

```
const c1 = new Carro('Gol', 85600, 0.15);
const c2 = new Carro('Camaro', 535000, 0.2);
console.log(c1.getPreco(), c2.getPreco());

// Função Factory
function Aluno(nome, notas) {
    return {
        nome,
        notas,
        getMedia() {
            let soma = 0;
            notas.forEach( element => soma+=element );
            return (soma/4).toFixed(1);
        }
    }
}

const a1 = Aluno('Marcelo', [4.5, 6.8, 7.9, 9.1]);
const a2 = Aluno('Verônica', [6.1, 7.2, 6.0, 9.8]);
console.log(a1.getMedia(), a2.getMedia());

// Object.create
/*  O método Object.create() cria um novo objeto,
    utilizando um outro objeto existente como
    protótipo para o novo objeto a ser criado.
*/
const pessoa = Object.create(null);
pessoa.nome = 'Carla';
console.log(pessoa);
const a3 = Object.create(c1);
console.log(a3);
console.log(a3.getPreco());
```

```
{}
{}
72760 428000
7.1 7.3
[Object: null prototype] { nome: 'Carla' }
Carro {}
72760
```

Notação Literal

JavaScript

```
const x = 10;
const y = 20;
const z = 30;

const obj1 = { x: x, y: y, z: z };
const obj2 = { x, y, z };
console.log(obj1, obj2)

const propriedade = 'pi';
const valor = 3.1415;

const obj3 = {};
obj3[propriedade] = valor;
console.log(obj3);

const obj4 = {[propriedade]: valor};
console.log(obj4);

const obj5 = {
    f1: function() {
        // ...
    },
    f2() {
        // ...
    }
}
console.log(obj5);
```

```
{ x: 10, y: 20, z: 30 } { x: 10, y: 20, z: 30 }
{ pi: 3.1415 }
{ pi: 3.1415 }
{ f1: [Function: f1], f2: [Function: f2] }
```

JSON x Object

```javascript
const obj = {
    x: 10,
    y: 12,
    z: 14,
    media() {
        return (x + y + z)/3;
    }
};
// Converter objetos e valores em uma string JSON
console.log(JSON.stringify(obj));
// Convete uma string JSON em um Objeto JavaScript
console.log(JSON.parse('{ "x": 10, "y":12, "z": 14 }'));
console.log(JSON.parse('{ "a": 3.1415, "b": "gil", "c": false,
"d": {}, "e": [] }'));
```

```
{"x":10,"y":12,"z":14}
{ x: 10, y: 12, z: 14 }
{ a: 3.1415, b: 'gil', c: false, d: {}, e: [] }
```

Métodos Arrays JavaScript

Um *array* é uma variável especial que pode conter mais de um valor, costumam ser utilizados quando temos uma lista de itens (uma lista de nomes de carros, por exemplo). Sendo assim, um *array* pode conter muitos valores referenciados por um único nome de variável, que podem ser acessados através de um índice.

Um array pode ser criado de duas formas, literal, utilizando dois colchetes ***[ ]***, ou tradicional, utilizando a expressão ***new Array()***. Veja os exemplos abaixo, eles fazem exatamente a mesma coisa:

Exemplo - Métodos para Criação de Array

```javascript
// Literal
const carros = ["FIAT", "Honda", "Volkswagen"];
```

```
// Tradicional
const carros = new Array("FIAT", "Honda", "Volkswagen");
```

Por uma questão de simplicidade, legibilidade e velocidade de execução, recomenda-se a utilização do método literal de criação de array, na linguagem JavaScript.

Método - map()

JavaScript

```
const numeros = [1, 2, 3, 4, 5];

// Laço com propósito
let resultado = numeros.map(function(item) {
    return item * 2;
})

console.log(resultado);

const soma10 = e => e + 10;
const triplo = e => e * 3;
const convert = e => `R$ ${parseFloat(e).toFixed(2).replace('.',
',')}`

resultado = numeros.map(soma10).map(triplo).map(convert);
console.log(resultado);
```

```
[ 2, 4, 6, 8, 10 ]
[ 'R$ 33,00', 'R$ 36,00', 'R$ 39,00', 'R$ 42,00', 'R$ 45,00' ]
```

OBS.: o método *map()* é usado para aplicar uma função a cada elemento de um array e retornar um novo array com os resultados da aplicação da função. Ele não modifica o array original.

Método Filter

JavaScript

```
const produtos = [
```

```
    { nome: 'Sofá', preco: 3200, fragil: false },
    { nome: 'iPhone', preco: 8999, fragil: true },
    { nome: 'Taça de Cristal', preco: 12.49, fragil: true },
    { nome: 'Garrafa Pet', preco: 3.99, fragil: false }
]

console.log(
    produtos.filter(function(p) {
        return false;
    })
)

const preco = produto => produto.preco >= 500;
const fragil = produto => produto.fragil;

console.log(produtos.filter(preco).filter(fragil));
```

```
[]
[ { nome: 'iPhone', preco: 8999, fragil: true } ]
```

OBS.: o método *filter()* cria um novo array com todos os elementos que passam no teste implementado pela função fornecida. Ele percorre cada elemento do array original e, para cada elemento, chama uma função callback. Se a função callback retornar true, o elemento é adicionado ao novo array; caso contrário, é ignorado. O método não altera o array original.

Método Reduce

JavaScript

```
const alunos = [
    { nome: 'Agatha', nota: 9.3 },
    { nome: 'Daniel', nota: 7.2 },
    { nome: 'Heloisa', nota: 8.8 },
    { nome: 'Nathan', nota: 6.7 }
]

console.log(alunos.map(a => a.nota));
```

```
const resultado = alunos.map(a => a.nota)
    .reduce(
        function(acumulador, atual) {
            console.log(acumulador, atual);
            return acumulador + atual;
        },
        0
    )

console.log(`Total = ${resultado}`);
```

```
[ 9.3, 7.2, 8.8, 6.7 ]
0 9.3
9.3 7.2
16.5 8.8
25.3 6.7
Total = 32
```

OBS.: o método *reduce()* é usado para percorrer um array e acumular um valor único baseado numa função de callback fornecida. Ele, essencialmente, "recompila" o array num único valor, sendo muito útil para operações como somar elementos, encontrar o maior valor ou agrupar dados

---

MANIPULANDO DOM - JAVASCRIPT

(https://www.w3schools.com/jsref/dom_obj_document.asp)

O DOM (Document Object Model) ou Modelo de Objeto de Documento, trata-se de uma representação hierárquica dos elementos HTML que compõem uma página web. Tal abordagem permite que os elementos HTML, e seus atributos, sejam acessados, modificados e atualizados através de linguagens de programação, tais como o JavaScript.

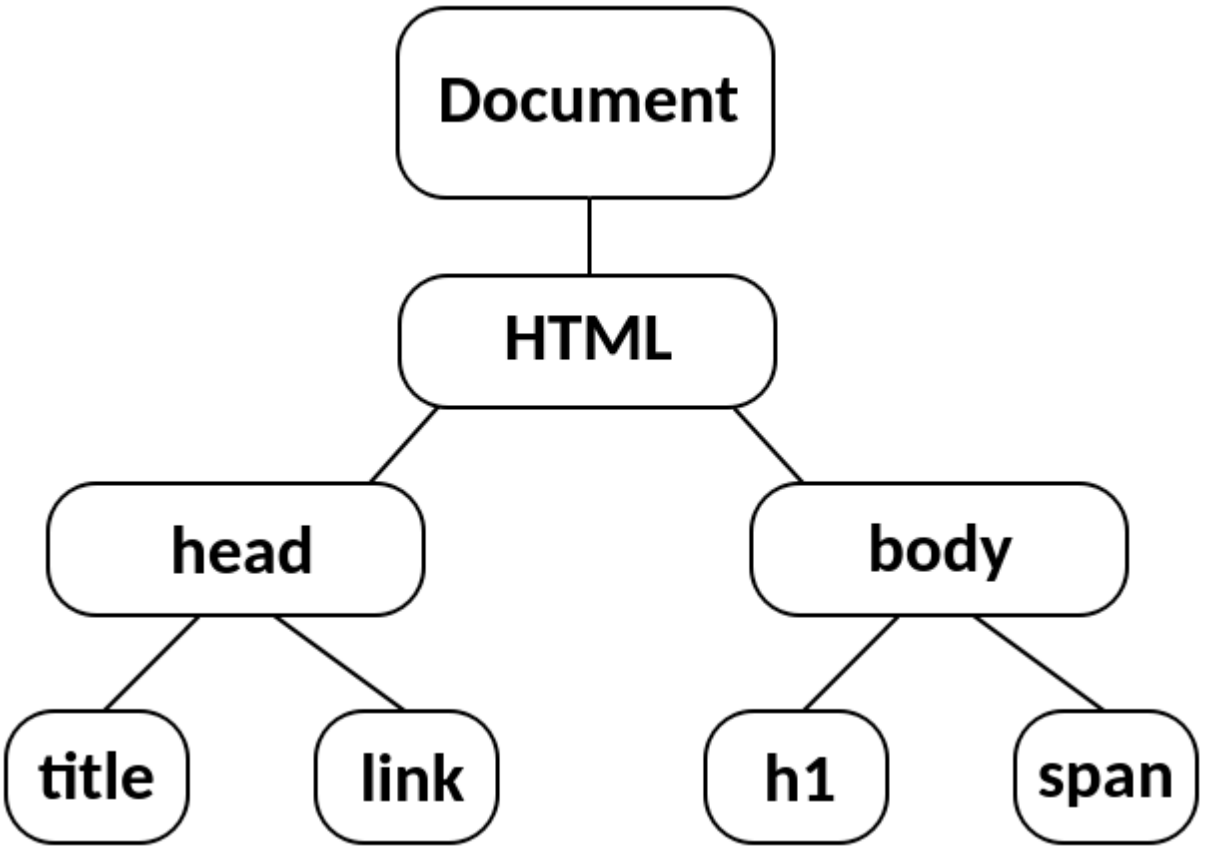


Figura 01: Representação gráfica - DOM

EVENTOS DOM - JAVASCRIPT

A linguagem JavaScript permite, como uma de suas principais funcionalidades, que o DOM (Document Object Model) seja manipulado, possibilitando que a interação com os elementos da página ocorra, tendo como base para tal os eventos.
Através destes eventos, torna-se possível adicionar interatividade e dinamismo às páginas web que estão sendo criadas. Em outras palavras, ao manipular eventos, através da linguagem JavaScript, é possível especificar as ações que deverão ser executadas para cada um dos eventos que estão sendo monitorados (clicar num botão, passar o mouse sobre uma imagem, etc).
A seguir é apresentado uma tabela que mapeia as propriedades CSS, dos elementos HTML, que podem ser acessadas e modificadas, através de linguagem JavaScript, quando um ou mais eventos ocorrem.

Tabela 01: Mapeamento das propriedades CSS no JavaScript.

Propriedade – CSS	Referência – JavaScript (style)
background	background
background-attachment	backgroundAttachment
background-color	backgroundColor
background-image	backgroundImage
background-position	backgroundPosition
background-repeat	backgroundRepeat
border	border
border-bottom	borderBottom
border-bottom-color	borderBottomColor
border-bottom-style	borderBottomStyle
border-bottom-width	borderBottomWidth
border-color	borderColor
border-left	borderLeft
border-left-color	borderLeftColor
border-left-style	borderLeftStyle
border-left-width	borderLeftWidth
border-right	borderRight
border-right-color	borderRightColor
border-right-style	borderRightStyle
border-right-width	borderRightWidth
border-style	borderStyle
border-top	borderTop
border-top-color	borderTopColor
border-top-style	borderTopStyle
border-top-width	borderTopWidth
border-width	borderWidth
clear	clear
clip	clip
color	color
cursor	cursor
display	display
filter	filter
font	font
font-family	fontFamily
font-size	fontSize
font-variant	fontVariant
font-weight	fontWeight
height	height
left	left

letter-spacing	letterSpacing
line-height	lineHeight
list-style	listStyle
list-style-image	listStyleImage
list-style-position	listStylePosition
list-style-type	listStyleType
margin	margin
margin-bottom	marginBottom
margin-left	marginLeft
margin-right	marginRight
margin-top	marginTop
overflow	overflow
padding	padding
padding-bottom	paddingBottom
padding-left	paddingLeft
padding-right	paddingRight
padding-top	paddingTop
page-break-after	pageBreakAfter
page-break-before	pageBreakBefore
position	position
float	styleFloat
text-align	textAlign
text-decoration	textDecoration
text-decoration: blink	textDecorationBlink
text-decoration: line-through	textDecorationLineThrough
text-decoration: none	textDecorationNone
text-decoration: overline	textDecorationOverline
text-decoration: underline	textDecorationUnderline
text-indent	textIndent
text-transform	textTransform
top	top
vertical-align	verticalAlign
visibility	visibility
width	width
z-index	zIndex

A seguir são apresentados alguns exemplos de eventos DOM, manipulados através da linguagem JavaScript, que permitem alterar as propriedades CSS dos elementos HTML, inserindo interatividade e dinamismos nas páginas web.

Arquivo: ./javascript/DOM e Eventos/exemplo_inicial.html

JavaScript

```
<div id="manipula">Eu sou a frase do exemplo. Serei alterada
quando você clicar.</div>

<hr>

<h2>Altere a frase acima clicando nos botões</h2>
<button onclick="mudaCor()">Muda cor da fonte</button>

<input type="button" value="Muda tamanho da fonte"
onclick="mudaTamanhoFonte()"/>

<input type="button" value="Muda background"
onclick="mudaBackground()"/>

<br><br>

<input type="button" value="Deixa em italico"
onclick="mudaItalico()"/>

<input type="button" value="Coloca borda"
onclick="colocaBorder()"/>

<input type="button" value="Muda a cor da borda"
onclick="corBorder()"/>

<br><br>

<input type="button" value="Exclui borda"
onclick="excluiBorder()"/>

<input type="button" value="Esconde a div"
onclick="escondeDiv()"/>

<input type="button" value="Reaparece com a div"
onclick="reapareceDiv()"/>
```

JavaScript

```
#manipula {
      height:50px;
      line-height:50px;
      color: yellow;
      font-size:10pt;
      background-color: red;
      margin-bottom:15px;
}
```

JavaScript

```
function mudaCor() {
      document.getElementById("manipula").style.color =
"#ffffff";
}

function mudaTamanhoFonte() {
      document.getElementById("manipula").style.fontSize =
"20pt";
}

function mudaBackground() {
      document.getElementById("manipula").style.backgroundColor
= "green";
}

function mudaItalico() {
      document.getElementById("manipula").style.fontStyle =
"italic";
}

function escondeDiv() {
```

```
      document.getElementById("manipula").style.display =
"none";
}

function reapareceDiv() {
      document.getElementById("manipula").style.display =
"block";
}

function colocaBorder() {
      document.getElementById("manipula").style.border = "2px
solid black";
}

function corBorder() {
      document.getElementById("manipula").style.border = "2px
solid #0000ff";
}

function excluiBorder() {
      document.getElementById("manipula").style.border = "0px";
}
```

Os conceitos apresentados anteriormente (eventos DOM, propriedades CSS, JavaScript) serão abordados, a partir de agora, de maneira mais detalhada neste documento.

MANIPULAÇÃO DO HTML

Utilizando Método: “document.querySelector()”

Selecionar Elemento

```javascript
document.querySelector("tag");          // Pela <tag> HTML
document.querySelector("#id");          // Pelo ID do elemento HTML
document.querySelector(".class");       // Pela classe CSS
```

OBS.: o comando “***querySelector()***” seleciona apenas o primeiro elemento encontrado para <tag> HTML, identificador (id) do elemento HTML e classe CSS.

OBS.: para selecionar todos os elementos encontrados para <tag> HTML, identificador (id) do elemento HTML e classe CSS utilizamos o comando “***querySelectorAll()***”.

Outros Métodos Similares - “querySelector()”

Selecionar Elemento

```javascript
document.getElementsByTagName("tag");    // Pela <tag> HTML
```

```
document.getElementById("id");           // Pelo ID do elemento HTML
document.getElementsByClassName("class"); // Pela classe CSS
```

OBS.: sempre que possível utilize o comando "***querySelector()***", os comandos acima foram apresentados com intuito de ampliar os conhecimentos sobre o tema, porém, sendo anteriores ao surgimento do "***querySelector()***".

Utilizando Propriedade: "textContent"

Alterar Conteúdo do Elemento

```JavaScript
let titulo = document.querySelector("h1");
titulo.textContent = "Novo Título";   // Altera o Conteúdo
```

Outra Propriedade Similar - "textContent"

Alterar Conteúdo do Elemento

```JavaScript
let titulo = document.querySelector("h1");
titulo.innerHTML = "Novo Título";     // Altera o Conteúdo
```

OBS.: sempre que possível utilize a propriedade "***textContent***", a propriedade acima foi apresentada com intuito de ampliar os conhecimentos sobre o tema, porém, é anterior ao surgimento da "***textContent***".

Utilizando o Método: "setAttribute"

Configurar Atributos HTML

```javascript
let imagem = document.querySelector("#logo");
imagem.setAttribute("src", "./img/logo.png");   // Altera o Atributo
imagem.setAttribute("width", "250px");          // Altera o Atributo
```

MANIPULAÇÃO DO CSS

Utilizando Propriedade: “style”

Aplicar Estilo ao Elemento

```javascript
let titulo = document.querySelector("h1");
titulo.style.color = "red";                 // Altera o Estilo
titulo.style.backgroundColor = "#AAA";      // Altera o Estilo
titulo.style.border = "1px solid red";      // Altera o Estilo
titulo.style.borderRadius = "10px";         // Altera o Estilo
```

OBS.: a abordagem apresentada anteriormente, utilizando a propriedade “***style***”, é indicada para situações onde apenas uma ou duas propriedades de estilo serão configuradas. A partir do momento em que é necessário um conjunto maior, torna-se mais adequado utilizar a abordagem apresentada a seguir, utilizando o método “***setAttribute()***” para alterar a propriedade “*class*”.

Utilizando Propriedade: “setAttribute(‘class’, ‘...’)”

Aplicar Estilo ao Elemento

```javascript
let titulo = document.querySelector("h1");
titulo.setAttribute("class", "dark");  // Altera o Atributo
```

```
.dark {
     color: red;
```

```
    background-color: #AAA;
    border: 1px solid red;
    border-radius: 10px;
}
```

OBS.: a abordagem apresentada, se comparada a anterior, permite efetuar a configuração do estilo através de uma única linha, em contrapartida as quatro utilizadas anteriormente. Obviamente, para tal, seria necessário criar a classe **"*dark*"** contendo todo o estilo que deve ser aplicado.

Utilizando Propriedade: "removeAttribute()"

Remover Estilo do Elemento

```JavaScript
let titulo = document.querySelector("h1");
titulo.removeAttribute("class");      // Remove o Atributo
```

OBS.: é importante salientar que o método "removeAttribute()", para propriedade "class", remove todas as classes de estilo que tenham sido aplicadas ao elemento HTML em questão. A seguir serão apresentados métodos que permitem especificar quais classes devem ser removidas ou adicionadas ao estilo de uma determinado elemento HTML.

Utilizando Métodos: "classList.add()" e "classList.remove()"

Remover Estilo do Elemento

Arquivo: ./javascript/DOM e Eventos/setAtributte

Código HMTL

```JavaScript
<button id="btn_main">
  Main Mode
```

```
</button>
<button class="btn_dark" id="btn_dark">
  Dark Mode
</button>
<button class="btn_light" id="btn_light">
  Light Mode
</button>

<h1 class="init" id="titulo">
  GIL EDUARDO DE ANDRADE
</h1>
```

Código CSS

JavaScript

```
.btn_dark {
  color: white;
  background-color: black;
}

.btn_light {
  color: black;
  background-color: white;
}

.init {
  display: flex;
  border-radius: 10px;
  padding: 10px 10px;
  justify-content: center;
}

.dark {
  color: white;
  background-color: black;
}
```

```
.light {
  color: #555;
  background-color: #DDD;
}
```

Código JavaScript

JavaScript

```
let btnM = document.querySelector("#btn_main")
let btnD = document.querySelector("#btn_dark")
let btnL = document.querySelector("#btn_light")
let tit = document.querySelector("#titulo")

btnD.addEventListener("click", darkMode)
btnL.addEventListener("click", lightMode)
btnM.addEventListener("click", mainMode)

function mainMode() {
  tit.classList.remove("dark")
  tit.classList.remove("light")
}

function darkMode() {
  tit.classList.remove("light")
  tit.classList.add("dark")
}

function lightMode() {
  tit.classList.remove("dark")
  tit.classList.add("light")
}
```

OBS.: os conteúdos de funções e eventos JavaScript, utilizados no exemplo acima - ***function()*** | ***addEventListener()*** - serão abordados em detalhes na sequência.

INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação

Main Mode Dark Mode Light Mode

GIL EDUARDO DE ANDRADE

Main Mode Dark Mode Light Mode

GIL EDUARDO DE ANDRADE

Main Mode Dark Mode Light Mode

GIL EDUARDO DE ANDRADE

Resultado - Codificação Anterior

FUNÇÕES JAVASCRIPT - BÁSICO

(https://www.w3schools.com/js/js_functions.asp)

Funções / Funções Anônimas / Arrow Functions

Diferentes formas de declaração

Uma função pode ser definida como um conjunto de instruções, ou bloco de código, criado para realizar uma uma tarefa específica. No JavaScript, uma função é executada quando um determinado trecho de código a invoca.

Funções Anônimas (Anonymous Functions)
Uma função anônima, em JavaScript, é uma função que não possui um nome. Este tipo de função pode ser atribuída a uma variável ou passada como parâmetro para outras funções. As funções anônimas possibilitam que a escrita do código seja feita de forma clara e eficiente, necessitando de uma quantidade menor de linhas.

JavaScript

```
// Não possui um nome definido após a palavra-chave function
// Atribuímos a função a variável para que seja possível invocá-la
const show = function() {
      return "Anonymous Function";

}
```

```
console.log(show());
```

Funções Seta (Arrow Functions)
As *arrow functions* possuem uma sintaxe mais simples (reduzida), clara e moderna para escrita de funções em JavaScript - são assim chamadas devido à seta (=>) usada na sua sintaxe.

JavaScript
```
// Atribuímos a função a variável para que seja possível invocá-la
const show = () => "Arrow Function";
console.log(show());
```

Utilizando: Anonymous e Arrow Functions

Contador Simples

Arquivo: ./javascript/DOM e Eventos/functions

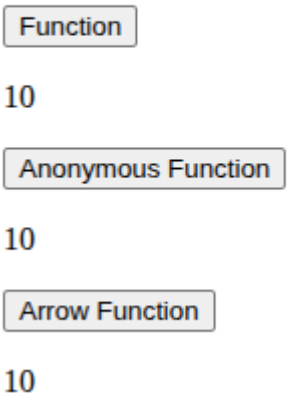
JavaScript
```
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1.0">
        <title>Funções - JavaScript</title>
    </head>
    <body onload="setValues()">
        <h1 class="init" id="titulo">FUNÇÕES JAVASCRIPT</h1>

        <button id="btn_fun"> Function </button>
        <p id="v1"></p>

        <button id="btn_ano"> Anonymous Function </button>
        <p id="v2"></p>

        <button id="btn_arr"> Arrow Function </button>
```

```
        <p id="v3"></p>
    </body>
    <script type="text/javascript">

        let val = 0;
        let fun = document.querySelector('#btn_fun');
        let ano = document.querySelector('#btn_ano');
        let arr = document.querySelector('#btn_arr');

        function setValues(num=0) {
            val+=num;
            document.querySelector("#v1").textContent = val;
            document.querySelector("#v2").textContent = val;
            document.querySelector("#v3").textContent = val;
        }

        // FUNCTION
        function func() { setValues(1); }

        // ANONYMOUS FUNCTION
        const anon = function () { setValues(2); }

        // ARROW FUNCTION
        const arro = () =>  setValues(3);

        // CLICK EVENTS
        fun.addEventListener("click", func);
        ano.addEventListener("click", anon);
        arr.addEventListener("click", arro);
    </script>
</html>
```

INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação

FUNÇÕES JAVASCRIPT



EVENTOS JAVASCRIPT

(https://www.w3schools.com/js/js_events.asp)

Principais Eventos DOM | JavaScript

Manipulando Eventos

Eventos HTML são “ações” que ocorrem com os elementos HTML, que podem ser detectados pela linguagem JavaScript. Um evento HTML pode ser uma ação que o navegador faz, ou uma ação que o usuário faz, como por exemplo:

- Uma página da web HTML terminou de carregar
- Um campo de entrada HTML foi alterado
- Um botão HTML foi clicado

Neste contexto, quando os eventos ocorrem, pode ser necessário reagir a eles, ou seja, através da linguagem JavaScript é possível executar rotinas específicas para cada um deles.
Para tal, o HTML possibilita que atributos do manipulador de eventos, com código JavaScript, sejam adicionados a elementos HTML.

Tabela 02: Principais eventos HTML.

Evento	Descrição
onclick	Elemento é clicado
ondblclick	Elemento recebe duplo click
onmouseover	Mouse é posicionado sob um elemento

onmouseenter	Mouse entra na área compreendida por um elemento
onmouseout	Mouse sai da área compreendida por um elemento
onkeypress	Tecla é pressionada
onkeydown	Tecla é pressionada (antes do onkeypress)
onsubmit	Formulário é enviado / submetido
onscroll	Uma página é rolada, via scroll
onfocus	Elemento obtém foco
onblur	Elemento perde foco
onload	Elemento, ou toda a página, é carregado
onresize	Janela do navegador é redimensionada
onchange	Valor de um campo de entrada é alterado

A tabela acima apresenta apenas os principais eventos HTML que podem ser manipulados via JavaScript. Cada um dos eventos apresentados possui diferentes propriedades e comportamentos. Para mais detalhes verifique a documentação oficial.

Eventos: “onload” e “onscroll”

Carregar Título* | *Efetuar Contagem

Arquivo: ./javascript/DOM e Eventos/events/load_scroll.html

JavaScript

```
<html>
      <body onload="setTitle()" onscroll="add()" style="height:
1000px;">
            <h1 id="titulo"></h1>
            <h2 id="subtitulo"></h2>
            <h3 id="contador"></h3>
      </body>

</html>

<script type="text/javascript">
    let val = 0;

    function setTitle() {
```

```
        document.querySelector("#titulo").textContent =
"Events DOM - JavaScript";
        document.querySelector("#subtitulo").textContent =
"onload() | onscroll()";
        document.querySelector("#contador").textContent = val;
    }

    function add() {
        val++;
        document.querySelector("#contador").textContent = val;
    }

</script>
```

Events DOM - JavaScript

onload() | onscroll()

63

Eventos: “mouseover” e “keypress”

Aumentar Imagem | Exibir e Esconder

Arquivo: ./javascript/DOM e Eventos/events/mouseover_keypress.html

JavaScript

```
<html>
      <head>
            <style>
                  .shadow {
                        box-shadow: 10px 10px 7px darkgray;
                        transition: box-shadow 0.5s ease-in-out;
                  }
            </style>
      </head>
```

```
      <body onload="setTitle()">
              <h1 id="titulo"></h1>
              <h2 id="subtitulo"></h2>
              <img
                  id="img_fogo"

src="https://gileduardo.com.br/ifpr/dw/img/fundo-display.png"
                  width="280px"
                  height="120px"
                  onmouseenter="addShadow()"
                  onmouseout="removeShadow()"
              >
              <br>
          </body>
</html>

<script type="text/javascript">

    document.onkeypress = function (e) {

        let img = document.querySelector("#img_fogo")

        // Letra e/E
        if(e.keyCode == 101 || e.keyCode == 69) {
            img.style.visibility = "hidden"
        }
        // Letra m/M
        else if(e.keyCode == 109 || e.keyCode == 77) {
            img.style.visibility = "visible"
        }
    };

    function setTitle() {
        document.querySelector("#titulo").textContent =
"Events DOM - JavaScript";
        document.querySelector("#subtitulo").textContent =
"mouseover() | mouseOut() | keypress()";
```

```
    }

    function addShadow() {
        let img = document.querySelector("#img_fogo")
        img.classList.add("shadow")
    }

    function removeShadow() {
        let img = document.querySelector("#img_fogo")
        img.classList.remove("shadow")
    }

</script>
```

Events DOM - JavaScript

mouseover() | mouseOut() | keypress()

"M - Mostrar" / "E - Esconder" Imagem

Eventos: “click” e “focus”

Visibilidade da Imagem | Exibir e Esconder

Arquivo: ./javascript/DOM e Eventos/events/click_focus.html

JavaScript

```
<html>
      <head>
```

```
            <style>
                  .hide {
                        opacity: 0.2;
                        transition: opacity 2s ease-in-out;
                  }
                  .show {
                        opacity: 1;
                        transition: opacity 2s ease-in-out;
                  }
            </style>
      </head>
      <body onload="setTitle()">
             <h1 id="titulo"></h1>
              <h2 id="subtitulo"></h2>
              <img
                  id="img_fogo"

src="https://gileduardo.com.br/ifpr/dw/img/fundo-display.png"
                  width="280px"
                  height="120px"
                  onclick="setOpacityHide()"
              >
              <p ondblclick="setOpacityShow()">Visibilidade -
Duplo Click</p>
              <input id="texto" type="text" onfocus="imgHide()"
onblur="imgShow()" onkeypress="msgBox()">
      </body>
</html>
<script type="text/javascript">

    function setTitle() {
        document.querySelector("#titulo").textContent =
"Events DOM - JavaScript";
        document.querySelector("#subtitulo").textContent =
"click() | dblclick() | focus() | blur()";
    }
```

```
    function setOpacityHide() {
        let img = document.querySelector("#img_fogo")
        img.classList.remove("show")
        img.classList.add("hide")
    }

    function setOpacityShow() {
        let img = document.querySelector("#img_fogo")
        img.classList.remove("hide")
        img.classList.add("show")
    }

    function imgHide() {
        let img = document.querySelector("#img_fogo")
        img.style.visibility = "hidden"
    }

    function imgShow() {
        let img = document.querySelector("#img_fogo")
        img.style.visibility = "visible"
    }

    function msgBox() {
        let t = document.querySelector("#texto")
        alert(t.value)
    }
</script>
```

ATUALIZAÇÕES PERIÓDICAS

(https://www.w3schools.com/jsref/met_win_setinterval.asp)

setInterval() / clearInterval()

Função

Em JavaScript, a função setInterval() possibilita executar um bloco de código (uma função) repetidamente, com um intervalo de tempo predefinido. Tal funcionalidade mostra-se útil para criação de animações, atualizações de dados em tempo real, ou qualquer outra ação que precisa ser executada periodicamente.

Funcionamento

O método ***setInterval()*** invoca uma função (bloco de código) dentro de um intervalo de tempo pré-definido, em milissegundos. O setInterval() continua invocando a função até que o método ***clearInterval()*** seja chamado ou a janela seja fechada.

- *1 segundo = 1000 milissegundos.*

Arquivo: ./javascript/DOM e Eventos/setinterval/index.html

JavaScript

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

```
    <head>
        <meta charset="UTF-8">
            <meta  name="viewport"  content="width=device-width,
initial-scale=1.0">
        <title>setInterval - JavaScript</title>
    </head>

    <body onload="startCounter()">
        <h1 class="init" id="titulo">setInerval JAVASCRIPT</h1>
        <h3 id="v">0</h3>
        <button id="btn_stop">PARAR</button>
        <button id="btn_continue">CONTINUAR</button>
    </body>

    <script type="text/javascript">
        let val = 0;
        let btn_s = document.querySelector('#btn_stop');
        let btn_c = document.querySelector('#btn_continue');
        let id;

        // FUNCTION
        function add() {
            val++;
            document.querySelector("#v").textContent = val;
        }
        // SET INTERVAL
        function startCounter() {
            id = setInterval(add, 1000)
        }
         // CLEAR INTERVAL
        function stopCounter() {
            clearInterval(id);
        }
        // CLICK EVENTS
        btn_s.addEventListener("click", stopCounter);
        btn_c.addEventListener("click", startCounter);
    </script>
```

```
</html>
```

ARMAZENAMENTO LOCAL

(*localStorage*)

(https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage)

O **localStorage** Javascript é uma API web de armazenamento que permite armazenar dados no formato de pares "*chave-valor*", no navegador do usuário. Diferente dos *cookies*, estes dados não possuem uma data de expiração e persistem mesmo após o navegador ser fechado e/ou o computador ser reiniciado. Possui capacidade máxima de armazenamento de, aproximadamente, ***5MB*** por domínio.

Principais Operações

Método	Descrição	Exemplo de Sintaxe
`setItem()`	Salva um par chave-valor	`localStorage.setItem('user', 'Alice')`
`getItem()`	Recupera um valor vinculado a uma chave	`const user = localStorage.getItem('user');`
`removeItem()`	Remove um item específico	`localStorage.removeItem('user');`
`clear()`	Remove todos dados armazenados	`localStorage.clear();`

Manipulando Objetos e Arrays

O localStorage armazena apenas dados com formato string UTF-16. Sendo assim, para armazenar estruturas de dados complexas, tais como objetos e arrays é necessário utilizar o JSON serialization:

1. Para armazenar: converta o objeto para uma string com ***JSON.stringfy()***

```JavaScript
const profile = { name: "Alice", theme: "dark" };
localStorage.setItem('profile', JSON.stringify(profile));
```

2. Para ler: converta a string novamente para objeto com ***JSON.parse()***

```JavaScript
const storedProfile = JSON.parse(localStorage.getItem('profile'));
```