

# JavaScript: Funções, CSS Dinâmico e Array

---

Prof. Eduardo Lino  
eduardo.lino@pucpr.br

# Mudança dinâmica do HTML

- Existem alguns comandos em JavaScript para a criação/modificação dinâmica da página HTML, por exemplo:

```
document.write("Olá");
```

- O **document.write("string")**, escreve um novo HTML na página.

# Mudança dinâmica do HTML

- Existem alguns comandos em JavaScript para a criação/modificação dinâmica da página HTML, por exemplo:

```
document.getElementById("id").innerHTML = "teste";
```

- O **innerHTML** recebe um novo conteúdo HTML que sobrescreverá o conteúdo HTML anterior do elemento que possui o id em questão.

# Mudança dinâmica do HTML

- Este evento possui duas opções:
  - Obter o conteúdo HTML do elemento.

```
<div id="divProduto">  
  <h2> Tabela de Produtos </h2>  
</div>
```

```
var conteudo = document.getElementById("divProduto").innerHTML;  
  
// conteudo == "<h2> Tabela de Produtos </h2>"
```

# Mudança dinâmica do HTML

- Este evento possui duas opções:
  - Inserir um novo conteúdo HTML no elemento.

```
<div id="divProduto">  
    <h2> Tabela de Produtos </h2>  
</div>
```

# Mudança dinâmica do HTML

- Este evento possui duas opções:
  - Inserir um novo conteúdo HTML no elemento.

```
<div id="divProduto">  
    <h2> Tabela de Produtos </h2>  
</div>
```

```
document.getElementById("divProduto").innerHTML = "<h1> Tabela de Marcas </h1>";
```

# Mudança dinâmica do HTML

- Este evento possui duas opções:
  - Inserir um novo conteúdo HTML no elemento.

```
<div id="divProduto">  
    <h2> Tabela de Produtos </h2>  
</div>
```

```
document.getElementById("divProduto").innerHTML = "<h1> Tabela de Marcas </h1>";
```

```
<div id="divProduto">  
    <h1> Tabela de Marcas </h1>  
</div>
```

# Mudança dinâmica do HTML

---

- O **innerHTML** também insere um novo conteúdo HTML como o último “filho”, logo no final do HTML do elemento.



# Mudança dinâmica do HTML

- O **innerHTML** também insere um novo conteúdo HTML como o último "filho", logo no final do HTML do elemento.

```
<div id="divMarcas">  
  <div> Fiat </div>  
</div>
```

# Mudança dinâmica do HTML

- O **innerHTML** também insere um novo conteúdo HTML como o último "filho", logo no final do HTML do elemento.

```
<div id="divMarcas">  
  <div> Fiat </div>  
</div>
```

```
document.getElementById("divMarcas").innerHTML += "<div> Ford </div>";
```

# Mudança dinâmica do HTML

- O **innerHTML** também insere um novo conteúdo HTML como o último "filho", logo no final do HTML do elemento.

```
<div id="divMarcas">  
  <div> Fiat </div>  
</div>
```

```
<div id="divMarcas">  
  <div> Fiat </div>  
  <div> Ford </div>  
</div>
```

```
document.getElementById("divMarcas").innerHTML += "<div> Ford </div>";
```

# Mudança dinâmica do HTML

- O **innerHTML** também insere um novo conteúdo HTML como o último "filho", logo no final do HTML do elemento.

```
<div id="divMarcas">  
  <div> Fiat </div>  
</div>
```

```
document.getElementById("divMarcas").innerHTML += "<div> Ford </div>";
```

```
<div id="divMarcas">  
  <div> Fiat </div>  
  <div> Ford </div>  
</div>
```

```
document.getElementById("divMarcas").innerHTML += "<div> Renault </div>";
```

# Mudança dinâmica do HTML

- O **innerHTML** também insere um novo conteúdo HTML como o último "filho", logo no final do HTML do elemento.

```
<div id="divMarcas">  
  <div> Fiat </div>  
</div>
```

```
document.getElementById("divMarcas").innerHTML += "<div> Ford </div>";
```

```
<div id="divMarcas">  
  <div> Fiat </div>  
  <div> Ford </div>  
</div>
```

```
document.getElementById("divMarcas").innerHTML += "<div> Renault </div>";
```

```
<div id="divMarcas">  
  <div> Fiat </div>  
  <div> Ford </div>  
  <div> Renault </div>  
</div>
```

# Função

---

- Significado no dia-a-dia?
  - Obrigação a cumprir, papel a desempenhar, uso a que se destina algo, etc.

# Função

---

- Significado no dia-a-dia?
  - Obrigação a cumprir, papel a desempenhar, uso a que se destina algo, etc.
- Significado em uma Linguagem de programação.
  - Conjunto de instruções que executa uma tarefa ou calcula um valor. Para usar uma função, você deve defini-la em algum lugar no escopo do qual você quiser chamá-la.

# Declaração

---

- A declaração da função consiste no uso da palavra chave **function**, seguida por:
  - Nome da Função.
  - Lista de argumentos para a função, entre parênteses e separados por vírgulas.
  - Declarações JavaScript que definem a função, entre chaves {}.



# Declaração (Java x JavaScript)

- Declaração de um método em Java.

```
int quadrado(int numero)
{
    return numero * numero;
}
```

# Declaração (Java x JavaScript)

- Declaração de um método em Java.

```
int quadrado(int numero)
{
    return numero * numero;
}
```

- Declaração de uma função em JavaScript.

```
function quadrado(numero)
{
    return numero * numero;
}
```

# Declaração de Função em JavaScript

---

Sempre será **function** no início

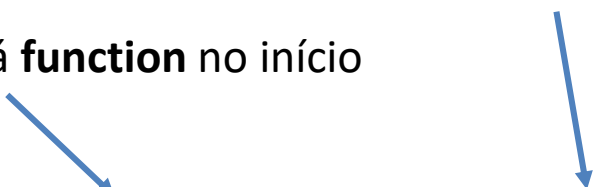


```
function quadrado(numero)
{
    return numero * numero;
}
```

# Declaração de Função em JavaScript

Sempre será **function** no início

Nome da função



```
function quadrado(numero)
{
    return numero * numero;
}
```

# Declaração de Função em JavaScript

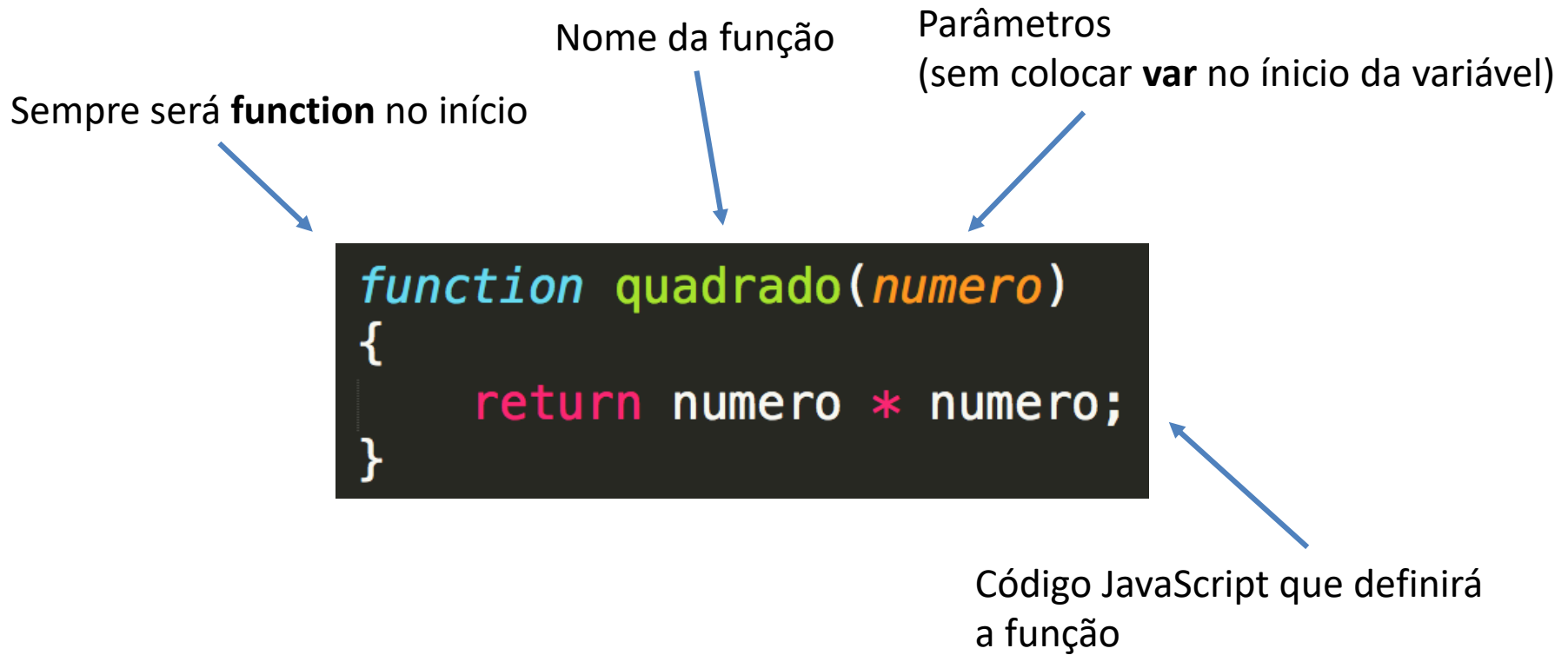
Sempre será **function** no início

Nome da função

Parâmetros  
(sem colocar **var** no início da variável)

```
function quadrado(numero)
{
    return numero * numero;
}
```

# Declaração de Função em JavaScript



# Declaração de Função em JavaScript

---

```
function quadrado(numero)
{
    return numero * numero;
}

var x = quadrado(4);

alert(x); // mostrará o valor 16
```

# Escopo da função

- As variáveis definidas no interior de uma função não podem ser acessadas de nenhum lugar fora da função, porque a variável está definida apenas no escopo da função.
- No entanto, uma função pode acessar todas variáveis e funções definida fora do escopo onde ela está definida.
- Em outras palavras, a função definida no escopo global pode acessar todas as variáveis definidas no escopo global.



# Escopo da função

```
var num1 = 10;
var num2 = 2;

function multiplica()
{
    var resultado = num1 * num2;

    alert(resultado);
}
```

# Escopo da função

```
var num1 = 10;
var num2 = 2;

function multiplica()
{
    var resultado = num1 * num2;

    alert(resultado); // 20
}
```

# Escopo da função

```
var num1 = 10;
var num2 = 2;

function multiplica()
{
    var num1 = 5;
    var resultado = num1 * num2;

    alert(resultado);
}
```

# Escopo da função

```
var num1 = 10;
var num2 = 2;

function multiplica()
{
    var num1 = 5;
    var resultado = num1 * num2;

    alert(resultado); // 10
}
```

# Exercício em Sala

- Desenvolver uma aplicação web utilizando as linguagens HTML e JavaScript.
- A aplicação deverá possuir um campo na tela e dois botões.
  - Botão 1: Este botão deverá mostrar somente os números pares.
  - Botão 2: Este botão deverá mostrar somente os números ímpares.
  - Condição de parada: Os números deverão ser mostrados até um número máximo, que será o número digitado no campo input.
- Deverá utilizar funções no exercício, uma função que calcula os números pares e uma outra função que calcula os números ímpares

# CSS Dinâmico

- Ao mesmo tempo em que se pode adicionar classes em CSS diretamente por atributo `class="nome-classe"` nos elementos do HTML, também é possível pelo JavaScript.
- A mudança dinâmica dos estilos de sites está presente na maior parte da web.

# CSS Dinâmico

- Em uma representação comum apenas entre HTML e CSS, temos:

```
.botao{  
    background-color: red;  
}
```

```
<button class="botao"> Acessar </button>
```

# CSS Dinâmico

- Porém, é possível que esta associação seja realizada também, através do JavaScript. O exemplo anterior também pode ser realizado desta forma:

```
.botao{  
    background-color: red;  
}
```

```
<button id="bAcessar"> Acessar </button>
```



# CSS Dinâmico

- Porém, é possível que esta associação seja realizada também, através do JavaScript. O exemplo anterior também pode ser realizado desta forma:

```
.botao{  
    background-color: red;  
}
```

```
<button id="bAcessar"> Acessar </button>
```

```
var elemento = document.getElementById("bAcessar");  
elemento.classList.add("botao");
```

# CSS Dinâmico

- No JavaScript, o método ***add*** adiciona uma nova classe CSS para um elemento em específico, mas também é possível remover uma classe de um elemento, com o método ***remove***.

```
var elemento = document.getElementById("bAcessar");  
elemento.classList.add("botao");
```

```
var elemento = document.getElementById("bAcessar");  
elemento.classList.remove("botao");
```

# Exercício em Sala 2

---

- Criar um formulário de cadastro EXATAMENTE igual ao formulário da próxima página (poderá utilizar o mesmo da aula passada).
- Ao clicar no botão "salvar", deverá verificar se algum campo está em branco (não preenchido), caso algum esteja, este campo deverá ter seu estilo modificado para o estilo do exemplo com a borda vermelha.

# Exercício em Sala 2

The image displays two browser window mockups side-by-side, both showing a registration form. The browser address bar for both is `file:///Users/josenuneslino/Desktop`.

**Left Mockup (Form Structure):**

- Nome:** Input field with a height of 10px and a width of 20px.
- Sobrenome:** Input field.
- E-mail:** Input field with a red border and a width of 1px.
- Matricula:** Input field.
- Usuário:** Input field.
- Senha:** Input field.
- Confirmar Senha:** Input field.
- Salvar:** Green button with a width of 100px and a height of 20px. The text 'Salvar' is in white. The background color is #088A68.

**Right Mockup (Form Data):**

- Nome:** Eduardo
- Sobrenome:** Lino
- E-mail:** (Red border, #FA5858)
- Matricula:** 1018912
- Usuário:** eduardo.lino
- Senha:** (Masked with dots)
- Confirmar Senha:** (Masked with dots)
- Salvar:** Green button with a width of 100px and a height of 20px. The text 'Salvar' is in white. The background color is #04B486. A hand cursor is pointing at the button.

# Array em JavaScript

## ■ Inserir elemento:

- `.push();`
  - Insere no final do array.
- `.unshift();`
  - Insere no início do array.
- `.splice();`
  - Insere em posição específica, pode inserir vários ao mesmo tempo.

## ■ Remover elemento:

- `.pop();`
  - Remove do final.
- `.shift();`
  - Remove do início do array.
- `.splice();`
  - Remove de posição específica, pode remover vários ao mesmo tempo.

# Array em JavaScript

## ■ Inserir elemento:

- `.push();`
  - Insere no final do array.
- `.unshift();`
  - Insere no início do array.
- `.splice();`
  - Insere em posição específica, pode inserir vários ao mesmo tempo.

## ■ Remover elemento:

- `.pop();`
  - Remove do final.
- `.shift();`
  - Remove do início do array.
- `.splice();`
  - Remove de posição específica, pode remover vários ao mesmo tempo.

# Array em JavaScript – Push / Pop

- Inserir elemento:

```
var frutas = [];  
  
frutas.push("morango");  
  
// array == ["morango"]  
  
frutas.push("uva");  
  
// array == ["morango", "uva"]
```

- Remover elemento:

```
frutas.pop();  
  
// array == ["morango"]
```

# Array em JavaScript

## ■ Inserir elemento:

- `.push();`
  - Insere no final do array.
- `.unshift();`
  - Insere no início do array.
- `.splice();`
  - Insere em posição específica, pode inserir vários ao mesmo tempo.

## ■ Remover elemento:

- `.pop();`
  - Remove do final.
- `.shift();`
  - Remove do início do array.
- `.splice();`
  - Remove de posição específica, pode remover vários ao mesmo tempo.



# Array em JavaScript – Unshift / Shift

- Inserir elemento:

```
var frutas = [];  
  
frutas.unshift("morango");  
  
// array == ["morango"]  
  
frutas.unshift("uva");  
  
// array == ["uva", "morango"]
```

- Remover elemento:

```
frutas.shift();  
  
// array == ["morango"]
```

# Array em JavaScript

## ■ Inserir elemento:

- `.push();`
  - Insere no final do array.
- `.unshift();`
  - Insere no início do array.
- **`.splice();`**
  - Insere em posição específica, pode inserir vários ao mesmo tempo.

## ■ Remover elemento:

- `.pop();`
  - Remove do final.
- `.shift();`
  - Remove do início do array.
- **`.splice();`**
  - Remove de posição específica, pode remover vários ao mesmo tempo.

# Splice (Sintaxe)

---

```
array.splice(/*posicao_inicial*/, /*qtde_remove*/, /*item_adicionar*/);
```

# Splice (Sintaxe)

---

```
array.splice(/*posicao_inicial*/, /*qtde_remove*/, /*item_adicionar*/);
```



Opcional, somente quando for adicionar!

# Splice (Sintaxe)

---

```
var frutas = ["maca", "banana", "pera", "uva", "abacaxi"];  
frutas.splice(1,1);  
// frutas == ["maca", "pera", "uva", "abacaxi"];
```

# Splice (Sintaxe)

---

```
var frutas = ["maca", "banana", "pera", "uva", "abacaxi"];  
frutas.splice(1, 0, "caju");  
// frutas == ["maca", "caju", "banana", "pera", "uva", "abacaxi"];
```

# Splice (Sintaxe)

---

```
var frutas = ["maca", "banana", "pera", "uva", "abacaxi"];  
frutas.splice(1, 0, "caju", "morango");  
// frutas == ["maca", "caju", "morango", "banana", "pera", "uva", "abacaxi"];
```

# Splice (Sintaxe)

---

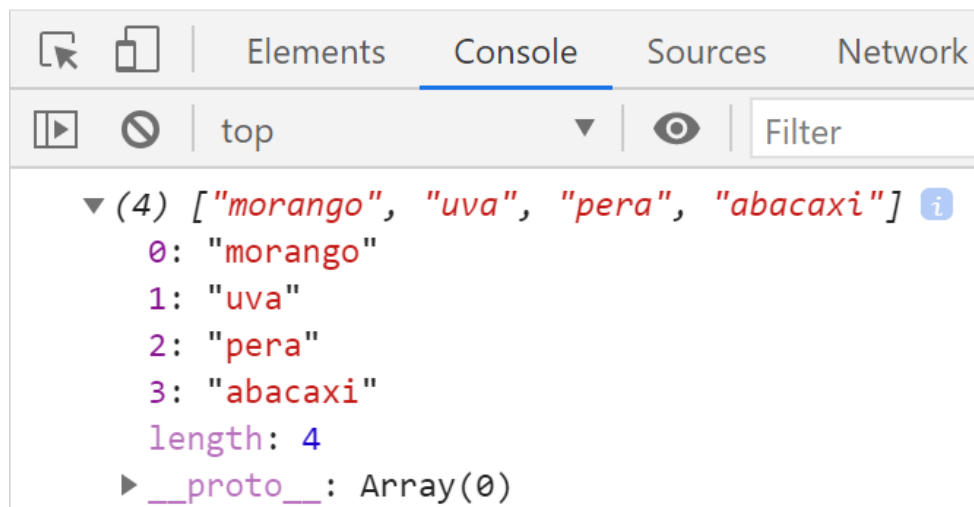
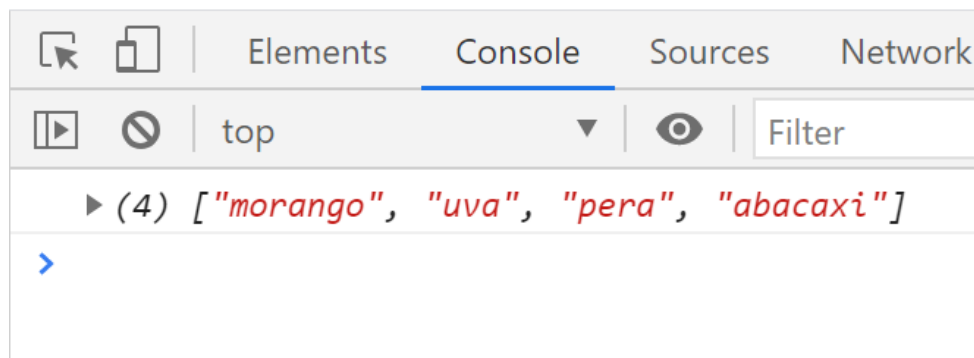
```
var frutas = ["maca", "banana", "pera", "uva"];  
frutas.splice(0, 2, "caju", "morango");  
// frutas == ["caju", "morango", "pera", "uva"];
```



# Print em Console

- Console.log:

```
var frutas = [];  
  
frutas.push("morango");  
frutas.push("uva");  
frutas.push("pera");  
frutas.push("abacaxi");  
  
console.log(frutas);
```



# JavaScript - Laço de Repetição

- Laços oferecem um jeito fácil e rápido de executar uma ação repetidas vezes, com o uso do **for**, podemos expressar um laço desta forma:

```
var frutas = [];  
  
frutas.push("morango");  
frutas.push("uva");  
frutas.push("pera");  
frutas.push("abacaxi");  
  
for(var contador = 0; contador < frutas.length; contador++)  
{  
    console.log(frutas[contador]);  
}
```

# JavaScript - Laço de Repetição

- Laços oferecem um jeito fácil e rápido de executar uma ação repetidas vezes, com o uso do **for**, podemos expressar um laço desta forma:

```
var frutas = [];  
  
frutas.push("morango");  
frutas.push("uva");  
frutas.push("pera");  
frutas.push("abacaxi");  
  
for(var contador = 0; contador < frutas.length; contador++)  
{  
    console.log(frutas[contador]);  
}
```

**Expressão inicial: código que será executado quando começar o laço (apenas uma vez)**

# JavaScript - Laço de Repetição

- Laços oferecem um jeito fácil e rápido de executar uma ação repetidas vezes, com o uso do **for**, podemos expressar um laço desta forma:

```
var frutas = [];  
  
frutas.push("morango");  
frutas.push("uva");  
frutas.push("pera");  
frutas.push("abacaxi");  
  
for(var contador = 0; contador < frutas.length; contador++)  
{  
    console.log(frutas[contador]);  
}
```

Condição de parada: código que será executado toda vez que finalizar um laço, caso a condição seja verdadeira

# JavaScript - Laço de Repetição

- Laços oferecem um jeito fácil e rápido de executar uma ação repetidas vezes, com o uso do **for**, podemos expressar um laço desta forma:

```
var frutas = [];  
  
frutas.push("morango");  
frutas.push("uva");  
frutas.push("pera");  
frutas.push("abacaxi");  
  
for(var contador = 0; contador < frutas.length; contador++)  
{  
    console.log(frutas[contador]);  
}
```

**Incremento:** este código será executado toda vez que finalizar uma iteração do laço, geralmente será um incremento da variável contadora