



INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 07 – FUNÇÕES II

O QUE IREMOS APRENDER

- 01** RESUMO DA AULA PASSADA
- 02** ARROW FUNCTIONS
- 03** FUNÇÕES DE CALLBACK
- 04** FUNÇÕES
- 05** PARADIGMA FUNCIONAL
- 06** MÃOS NO CÓDIGO

RESUMO DA AULA PASSADA

Uma função é um bloco de código reutilizável que realiza uma tarefa específica. As funções são usadas para organizar e estruturar o código, tornando-o mais modular e fácil de entender. Elas aceitam entradas (parâmetros), processam essas entradas e retornam um resultado (saída).

Parâmetros: parâmetros são valores que você passa para uma função quando a chama

Retorno: é o resultado que uma função fornece após executar suas instruções internas.

FUNÇÕES

Teste esse código

```
<body>
  <form>
    <label for="nome">Nome: </label>
    <input type="text" id="nome">
    <label for="saudacao">Saudacao: </label>
    <input type="text" id="saudacao">
    <button onclick="somar()">Enviar</button>
  </form>
  <script>
    function somar() {
      let nome = document.getElementById('nome').value
      let saudacao = document.getElementById('saudacao').value
      console.log(saudacao + ' ' + nome)
    }

  </script>
</body>
```

ARROW FUNCTIONS

Arrow functions, são um tipo de sintaxe utilizada para escrever funções de forma mais condensada. A sintaxe de uma arrow function retira a palavra function e adiciona um “=>” (como uma flecha) entre o parênteses e o escopo da função. Ela funciona como uma função anônima, precisando ser associada a alguma outra estrutura de dados.

ARROW FUNCTIONS

EXEMPLO

```
<script>
  // Função tradicional
  function soma(a, b) {
    return a + b;
  }

  // Arrow function equivalente
  const somaArrow = (a, b) => a + b;

  console.log(soma(2, 3)); // Saída: 5
  console.log(somaArrow(2, 3)); // Saída: 5
</script>
```

FUNÇÕES DE CALLBACK

Funções de callback são funções que são passadas como argumentos para outras funções e são executadas após a conclusão de determinada tarefa ou evento assíncrono.

Em outras palavras, uma função de callback é uma função que é "chamada de volta" ou executada mais tarde, geralmente quando uma operação assíncrona é concluída ou quando um evento específico ocorre.

FUNÇÕES DE CALLBACK

EXEMPLO

```
<script>
  function somarNumeros(a, b) {
    return a + b;
  }
  const soma=somarNumeros(10,1)

  function imprimirResultado(resultado) {
    console.log(`O resultado da soma é ${resultado}`)
    resultado(a,b)
  }
  imprimirResultado(soma)
</script>
```


FUNÇÕES DE CALLBACK

EXEMPLO

```
<script>
  /*criamos uma funcao saudacao que possui o parametro nome e
  quando chamamos a funcao aparec um alert*/
  function saudacao(nome) {
    alert(`Olá ${nome}`);
  }
  /*criamos uma funcao saudacao que possui o parametro callback e
  quando chamamos a funcao aparece um prompt pedindo para o usuário
  inserir o nome dele*/
  function chamadaCallback(callback) {
    let nome = prompt('Por favor insira seu nome');
    /*chamamos o parametro da nossa funcao chamadaCallback e estamos colocando o nome que
    o usuário digitou como argumento desse parâmetro*/
    callback(nome);
  }
  /* Aqui, a função chamadaCallback é chamada e passamos a função saudacao como um argumento para ela.
  Isso significa que a função saudacao será usada como a função de callback dentro da função chamadaCallback.*/
  chamadaCallback(saudacao);//Ex: se eu coloquei no prompt José a saída do código será : Olá José
</script>
```

FUNÇÕES

JavaScript é uma linguagem de programação bastante flexível e versátil, e ela possui muitas funções nativas prontas que você pode usar para realizar diversas tarefas. Aqui estão alguns exemplos de funções nativas do JavaScript:

Manipulação de Datas:

`Date()`: Cria um objeto de data.

```
<script>
  const dataAtual = new Date();
  console.log(dataAtual);           // Saída: Fri Aug 18 2023 15:46:59 GMT-0300 (Horário Padrão de Brasília)
  const dataNascimento = new Date('2004-02-12');
  console.log(dataNascimento.getFullYear()); // Saída: 2004
</script>
```

FUNÇÕES

Manipulação de Strings:

- `String.length`: Retorna o comprimento de uma string.
- `String.toLowerCase()`: Converte uma string para minúsculas.
- `String.toUpperCase()`: Converte uma string para maiúsculas.

```
<script>
  const texto = "Olá, Mundo!";
  console.log(texto.length);           // Saída: 11
  console.log(texto.toLowerCase());    // Saída: "olá, mundo!"
  console.log(texto.toUpperCase());    // Saída: "OLÁ, MUNDO!"
</script>
```

FUNÇÕES

Manipulação de Números:

- `parseInt()`: Converte uma string em um número inteiro.
- `parseFloat()`: Converte uma string em um número de ponto flutuante.
- `Number.toFixed()`: Formata um número com um número específico de casas decimais.

FUNÇÕES

Manipulação de Números:

```
<script>
  const numero = parseInt("42");
  console.log(numero);           // Saída: 42
  const decimal = parseFloat("3.14");
  console.log(decimal);         // Saída: 3.14
  const pi = 3.141592653589793;
  console.log(pi.toFixed(2));    // Saída: "3.14"
</script>
```

ATIVIDADE PRÁTICA

Atividade 01

Crie uma arrow function chamada multiplicar que recebe dois números como argumentos e retorna o resultado da multiplicação desses números.

Atividade 02

Crie uma arrow function chamada contarVogais que recebe uma string como argumento e retorna a quantidade de vogais na string.

ATIVIDADE PRÁTICA

Atividade 03

Crie uma função chamada `converterTemperatura` que recebe uma temperatura em graus Celsius e uma função de callback para conversão como argumentos. A função deve aplicar a função de callback à temperatura e retornar o resultado.

Atividade 04

Implemente uma função (callback) que receba um número como parâmetro e informe o cubo desse número

PARADIGMA FUNCIONAL

O **paradigma funcional** em JavaScript refere-se à aplicação dos princípios e conceitos do paradigma funcional no desenvolvimento de programas em JavaScript. Isso significa usar funções como a principal unidade de construção, enfatizar a imutabilidade dos dados, evitar efeitos colaterais e adotar a composição de funções para resolver problemas.

Vantagens do Paradigma Funcional em JavaScript:

- | | |
|-----------------|----------------------------|
| 1. Legibilidade | 2. Previsibilidade |
| 3. Reutilização | 4. Manutenção Simplificada |

PARADIGMA FUNCIONAL

JavaScript, embora uma linguagem multiparadigma, suporta bem a programação funcional com isso permite que os desenvolvedores adotem abordagens funcionais em seus projetos, beneficiando-se das vantagens desse paradigma.



PARADIGMA FUNCIONAL

Em JavaScript, as funções são cidadãs de **primeira classe**, o que significa que podem ser atribuídas a variáveis, passadas como argumentos para outras funções e retornadas como valores de outras funções.

```
const dobrar = x => x * 2;  
const triplicar = x => x * 3;  
  
const aplicarOperacao = (funcao, valor) => funcao(valor);  
  
console.log(aplicarOperacao(dobrar, 5)); // Saída: 10  
console.log(aplicarOperacao(triplicar, 5)); // Saída: 15
```

PARADIGMA FUNCIONAL

O paradigma funcional incentiva o uso de dados **imutáveis**, ou seja, dados que não podem ser alterados após serem criados.

```
// Imutabilidade usando concat (em vez de push)
const lista1 = [1, 2, 3];
const lista2 = lista1.concat(4);

console.log(lista1); // Saída: [1, 2, 3]
console.log(lista2); // Saída: [1, 2, 3, 4]
```

PARADIGMA FUNCIONAL

Funções puras não têm efeitos colaterais e retornam o mesmo resultado para os mesmos argumentos.

```
// Função impura (dependendo da variável externa)
let x = 10;
const adicionarX = y => y + x;

console.log(adicionarX(5)); // Saída: 15

x = 20;
console.log(adicionarX(5)); // Saída: 25

// Função pura
const somar = (a, b) => a + b;

console.log(somar(3, 4)); // Saída: 7
```

PARADIGMA FUNCIONAL

No paradigma funcional, você se concentra em descrever o que você quer alcançar, em vez de detalhar como fazer isso.

```
// Programação imperativa
const numeros = [1, 2, 3];
let quadrados = [];

for (let i = 0; i < numeros.length; i++) {
  quadrados.push(numeros[i] * numeros[i]);
}

console.log(quadrados); // Saída: [1, 4, 9]

// Programação declarativa (usando map)
//OBS: O map será abordado nas aulas posteriores

const numeros = [1, 2, 3];
const quadrados = numeros.map(x => x * x);
console.log(quadrados); // Saída: [1, 4, 9]
```

ATIVIDADE PRÁTICA

Atividade 05

Crie uma função que aceita um número e retorna uma função que, por sua vez, aceita outro número e retorna a soma dos dois.

Atividade 06

Crie uma função chamada `operacoesMatematicas` que recebe dois números e uma função de callback como argumentos. A função deve aplicar a função de callback aos dois números e retornar o resultado.

DESAFIO PRÁTICO

Você está desenvolvendo um sistema de autenticação para um aplicativo. Sua tarefa é implementar uma função chamada `autenticarUsuario` que aceita um nome de usuário, uma senha e duas funções de callback como argumentos. A função `autenticarUsuario` deve simular um processo de autenticação simples, onde o nome de usuário e a senha são verificados.

DESAFIO PRÁTICO

Se as informações forem válidas, a primeira função de callback deve ser chamada e mostrar uma frase de login com o nome de usuário em letras minúsculas. Caso contrário, a segunda função de callback deve ser chamada para indicar a falha na autenticação..

SE LIGA NO CONTEÚDO DA PRÓXIMA AULA!

AULA 08 DE JAVASCRIPT.
EVENTOS DOM



IN

INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

EVENTOS DOM

DOM (Document Object Model) é uma representação da nossa árvore de elementos do HTML. Ela nos traz todo o modelo do nosso documento HTML, nos permitindo manipulá-lo de diversas formas.

Ela se assemelha muito a uma árvore genealógica, onde cada elemento tem seus respectivos pai e filhos. No entanto, cada elemento pode ter apenas um elemento pai, mas diversos elementos filhos.



EVENTOS DOM

Exemplos da **DOM** em JavaScript:

```
const meuElemento = document.getElementById('meuElemento')  
const elementoFilho1 = document.createElement('p')  
const elementoFilho2 = document.createElement('p')  
  
meuElemento.append(elementoFilho1, elementoFilho2)
```



INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 07 – FUNÇÕES II