



# Programação Novas Tecnologias

Prof : Pedro Albino



# Tópicos.

- JavaScript Variáveis.
- JavaScript Estruturas de Decisão, Repetição, Array.
- JavaScript Funções.
- JavaScript DOM.
- JavaScript Eventos.

# Tópicos da aula de hoje.

## JavaScript

Nas aulas anteriores, vimos que as linguagens HTML e a CSS são fundamentais para a criação de páginas web. O foco do **HTML é o conteúdo** enquanto o **foco do CSS é a formatação dessas páginas**.

# JavaScript

As linguagens **HTML** e **CSS** não são linguagens de programação. Para **resolver determinados problemas**, é necessário utilizar uma linguagem de **programação**. Por isso, a linguagem de programação JavaScript.

Com a linguagem **JavaScript** podemos **construir páginas extremamente dinâmicas e interativas**. O foco do **JavaScript** é **implementar o comportamento ou a inteligência das páginas web**.

# JavaScript

Como aplicar o JavaScript no HTML.

Existem duas formas de associar código JavaScript e documentos HTML.

# JavaScript

## JavaScript interno

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      alert("Escola ETEPD");
    </script>
  </head>
  <body>

  </body>
</html>
```

O código JavaScript pode ser definido dentro de um documento HTML no corpo do elemento **script**.

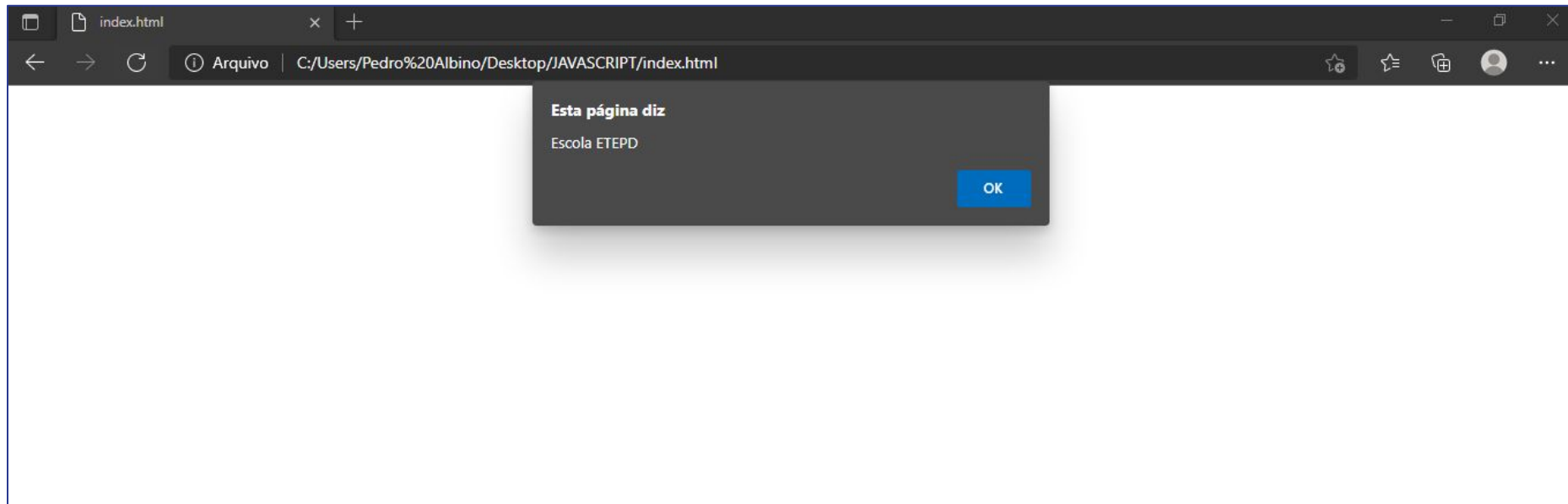
**<script>**

    alert("Escola ETEPD");

**</script>**

Nessa abordagem, o código JavaScript fica “amarrado” a um único documento HTML.

# JavaScript



# JavaScript

## JavaScript externo

```
<!DOCTYPE html>
<html>
  <head>
    <script src="js/script.js"></script>
  </head>
  <body>

  </body>
</html>
```

O código JavaScript **pode ser definido em arquivos separados e depois associados aos documentos HTML** através do elemento **script**.

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

Nessa outra abordagem, podemos reutilizar o mesmo código JavaScript em vários documentos

HTML.



# JavaScript

O elemento **script** permite associar código JavaScript aos documentos HTML. Esse elemento pode ser adicionado dentro do corpo dos elementos **head** e **body**. A **localização do elemento script afeta o momento no qual o código JavaScript será carregado pelos navegadores.**

O exemplo a seguir, o elemento **script** foi colocado dentro do corpo do **head**. Dessa forma, o código JavaScript será carregado antes do **body** ser processado. Consequentemente, a página só será exibida depois do carregamento do JavaScript.

# JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <script src="js/script.js"></script>
  </head>
  <body>

  </body>
</html>
```

# JavaScript

O próximo exemplo, o elemento script foi colocado no final do body. Consequentemente, o JavaScript só será carregado depois de todos os outros elementos do body.

# JavaScript

```
<!DOCTYPE html>
<html>
  <head>

  </head>
  <body>

    <script src="js/script.js"></script>
  </body>
</html>
```

# JavaScript

A **segunda abordagem é mais recomendada pois as páginas web são exibidas mais rapidamente aos usuários.** Mas a casos que é necessário utilizar a abordagem do código JavaScript primeiro.

# JavaScript

## Exercício de Fixação

Cria um projeto chamado **aula01** e no projeto crie um arquivo chamado **javascript.html** e nesse projeto crie um script **javascript** com a função de **alert** com a seguinte mensagem Olá Mundo.

# JavaScript - Variáveis

Assim como qualquer linguagem de programação, **JavaScript permite a criação de variáveis através da palavra chave var**. Toda variável deve ter um nome (identificador).

```
var idadeDoJonas = 30;  
var precoDoProduto = 28.75;  
var nomeDoInstructor = "Marcelo Martins";  
var acessoLiberado = true;
```

# JavaScript - Variáveis

O **exemplo anterior**, **todas as variáveis foram inicializadas**. Antes da inicialização, as variáveis possuem o valor especial **undefined**. No próximo exemplo, a variável **altura** não é inicializada. Portanto, ela possuíra o valor **undefined**.

```
var altura;
```



# JavaScript - Operadores

Manipulam os valores das variáveis de um programa, devemos utilizar os operadores oferecidos pela linguagem de programação adotada. A linguagem JavaScript possui diversos operadores e os principais são categorizados da seguinte forma:

- Aritmético: + - \* / %
- Atribuição: = += -= \*= /= %= ++ --
- Relacional: == != < <= > >=
- Lógico: && ||

# JavaScript - Operadores Aritméticos

Funcionam de forma muito semelhante aos operadores da matemática.

- Adição +
- Subtração -
- Multiplicação \*
- Divisão /
- Módulo %

# JavaScript - Operadores Aritméticos

```
1 var umMaisUm = 1 + 1;
2 // umMaisUm = 2
3
4 var tresVezesDois = 3 * 2;
5 // tresVezesDois = 6
6
7 var quatroDivididoPorDois = 4 / 2;
8 // quatroDivididoPor2 = 2
9
10 var seisModuloCinco = 6 % 5;
11 // seisModuloCinco = 1
12
13 var x = 7;
14
15 x = x + 1 * 2;
16 // x = 9
17
18 x = x - 3;
19 // x = 6
20
21 x = x / (6 - 2 - (3 * 5) / (16 - 1));
22 // x = 2
```

# JavaScript - Concatenação de Strings

Vimos anteriormente, o operador + é utilizado para realizar soma aritmética. Mas, ele também pode ser utilizado para concatenar strings.

```
1 var s1 = "Marcelo";  
2 var s2 = " ";  
3 var s3 = "Martins";  
4  
5 // "Marcelo Martins"  
6 var s4 = s1 + s2 + s3;
```

# JavaScript - Concatenação de Strings

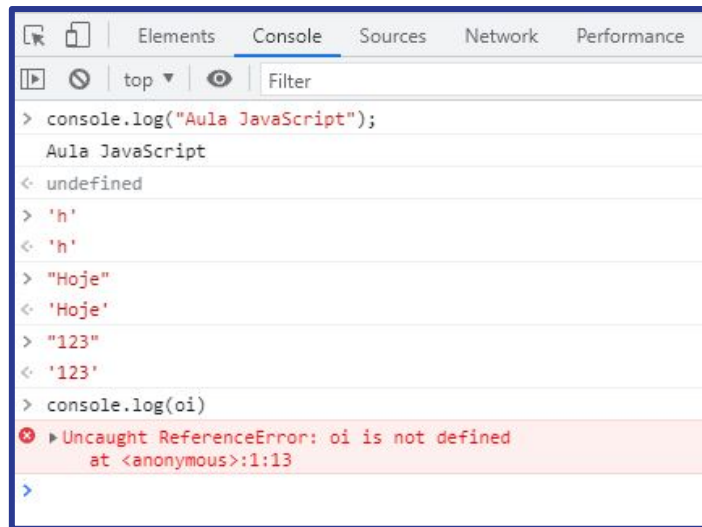
Exemplo a seguir.

```
1 var s1 = "Idade: ";  
2 var idade = 30;  
3  
4 // "Idade: 30"  
5 var s2 = s1 + idade;
```

Observe que o operador + foi aplicado a um valor numérico e a um texto. Nesses casos, o valor numérico é, automaticamente, transformado em texto e a concatenação é realizada.

# JavaScript - Concatenação de Strings

DICA: É correto usar aspas duplas " ou simples ' com strings, desde que você seja consistente.



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following sequence of commands and outputs:

```
> console.log("Aula JavaScript");
Aula JavaScript
< undefined
> 'h'
< 'h'
> "Hoje"
< 'Hoje'
> "123"
< '123'
> console.log(oi)
Uncaught ReferenceError: oi is not defined
at <anonymous>:1:13
```

The error message is highlighted in red, indicating a runtime error due to an undefined variable.

# JavaScript - Concatenação de Strings

**Strings** são uma coleção de caracteres colocados entre aspas duplas ou simples. Você pode usar strings para representar dados como frases, nomes, endereços e muito mais. **Você sabia que pode até adicionar strings?** Em JavaScript, isso é chamado de concatenação.

“Olá,” + “ Recife”

Retorna: “Olá, Recife”

# JavaScript - Concatenação de Strings

## Pare para pensar...

As expressões são avaliadas da esquerda para a direita. Dessa forma, considere o seguinte exemplo:

```
1 alert(1 + 2 + 3 + " testando");  
2 alert("testando" + 1 + 2 + 3);
```

**O que seria exibido nesse exemplo?**



# JavaScript - Concatenação de Strings

Qual é o resultado com "Olá" + "Mundo"?

a) "Olá      Mundo"

b) "OláMundo"

# JavaScript - Concatenação de Strings

O que você acha que acontecerá quando digitar "Olá + 5\*10" no console JavaScript?

- a) "Olá5\*10"
- b) "Olá50"
- c) "Olá+5\*10"
- d) Um erro!

# JavaScript - Concatenação de Strings

O que você acha que acontecerá quando você digitar "Olá" + 5\*10 no console?

- a) "Olá5\*10"
- b) "Olá50"
- c) "Olá+5\*10"
- d) Um erro!

# JavaScript - Atribuição

**Os operadores de atribuição são:**

- Simples =
- Incremental +=
- Decremental -=
- Multiplicativa \*=
- Divisória /=
- Modular %=
- Incremento ++
- Decremento --

# JavaScript - Relacionais

**Os operadores relacionais são:**

- Igualdade ==
- Diferença !=
- Menor <
- Menor ou igual <=
- Maior >
- Maior ou igual >=

```
1 var valor = 2;  
2 var t = false;  
3 t = (valor == 2);  
4 t = (valor != 2);  
5 t = (valor < 2);  
6 t = (valor <= 2);  
7 t = (valor > 1);  
8 t = (valor >= 1);
```

# JavaScript - Lógicos

Permite verificar duas ou mais condições através de operadores lógicos.

# JavaScript - If e else

O comportamento de uma aplicação pode ser influenciado por valores definidos pelos usuários.

Por exemplo, considere um sistema de cadastro de produtos. **Se um usuário tenta adicionar um produto com preço negativo, a aplicação não deve cadastrar esse produto.** Caso contrário, se o preço não for negativo, o cadastro pode ser realizado normalmente.

Para verificar uma determinada condição e decidir qual bloco de instruções deve ser executado, devemos aplicar o comando if.

# JavaScript - If e else

```
<script>
  var preco = -1;
  if (preco <= 0){
    alert('O preço do produto não pode ser negativo');
  }else{
    alert('Produto cadastrado com sucesso');
  }
</script>
```



# JavaScript - If e else

Em JavaScript, você pode representar essa verificação secundária usando uma instrução if extra **chamada de instrução else if**.

```
<script>
  var weather = "Sol";

  if(weather == "Neve") {
    console.log("Traga um casaco.");
  } else if (weather == "Chuva") {
    console.log("Traga uma capa de chuva.");
  } else {
    console.log("Vista o que você tem.");
  }
</script>
```

# JavaScript - If e else

Escreva um código em JavaScript que tenha o preço do produto, o dinheiro que foi pago pela pessoa que comprou e verificar se a pessoa pagou, se a pessoa pagou a mais a mensagem "Você pagou a mais, aqui está o seu troco", se o valor pago for igual ao valor do produto "Você pagou o valor exato, tenha um bom dia!", se o valor pago for abaixo do produto "Isso não é suficiente, você ainda me deve dinheiro."

# JavaScript - If e else

O que será impresso no console se o código a seguir for executado?

```
<script>
  var dinheiro = 100.50;
  var preco = 100.50;

  if (dinheiro > preco) {
    console.log("Você pagou a mais, aqui está o seu troco");
  } else if (dinheiro == preco) {
    console.log("Você pagou o valor exato, tenha um bom dia!");
  } else {
    console.log("Isso não é suficiente, você ainda me deve dinheiro.");
  }
</script>
```

- a) Você pagou a mais, aqui está o seu troco
- b) Você pagou o valor exato, tenha um bom dia!
- c) Isso não é suficiente, você ainda me deve dinheiro.

# JavaScript - While

Existem muitos tipos diferentes de loops, mas todos eles fazem essencialmente a mesma coisa: eles repetem uma ação algumas vezes.

As três informações principais que qualquer loop deve ter são:

**Quando começar:** O código que configura o loop - definindo o valor inicial de uma variável, por exemplo.

**Quando parar:** a condição lógica para testar se o loop deve continuar.

**Como ir para o próximo item:** A etapa de incremento ou decremento - por exemplo,  $x = x * 3$  ou  $x = x - 1$

Aqui está um exemplo básico de loop while que inclui todas as três partes.

# JavaScript - While

Existem muitos tipos diferentes de loops, mas todos eles fazem essencialmente a mesma coisa: eles repetem uma ação algumas vezes.

As três informações principais que qualquer loop deve ter são:

**Quando começar:** O código que configura o loop - definindo o valor inicial de uma variável, por exemplo.

**Quando parar:** a condição lógica para testar se o loop deve continuar.

**Como ir para o próximo item:** A etapa de incremento ou decremento - por exemplo,  $x = x * 3$  ou  $x = x - 1$

Aqui está um exemplo básico de loop while que inclui todas as três partes.

# JavaScript - While

```
<script>
  var start = 0; //quando começar
  while (start < 10) { // quando parar
    console.log(start);
    start = start + 2; // quando ir para o próximo item
  }
</script>
```

*Impressões:*

0

2

4

6

8

# JavaScript - While

Quantas vezes o loop while será executado?

```
var x = 10;  
while (x <= 25) {  
  console.log('Imprimindo x = ' + x);  
  x = x + 2;  
}
```

- a) 15
- b) 16
- c) 8
- d) 9

# JavaScript - While

Em alguns casos, é necessário repetir um determinado trecho de código várias vezes. Por exemplo, suponha que seja necessário exibir 10 vezes a mensagem: “Bom Dia”. Podemos resolver essa tarefa com o seguinte código JavaScript.



# JavaScript - While

```
<!DOCTYPE html>
<html>
  <head>

  </head>
  <body>

    <script>
      var texto = " Boa noite";
      var i = 0;
      while (i < 10) {

        console.log(i++ + texto);
      }

    </script>
  </body>
</html>
```

# JavaScript - FOR

O comando for é análogo ao while. A principal diferença entre esses dois comandos é que o for recebe três argumentos.

# JavaScript - FOR

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <script>
      for(var contador = 0; contador < 5; contador++){
        console.log("Boa noite");
      }
    </script>
  </body>
</html>
```

# Funções JavaScript

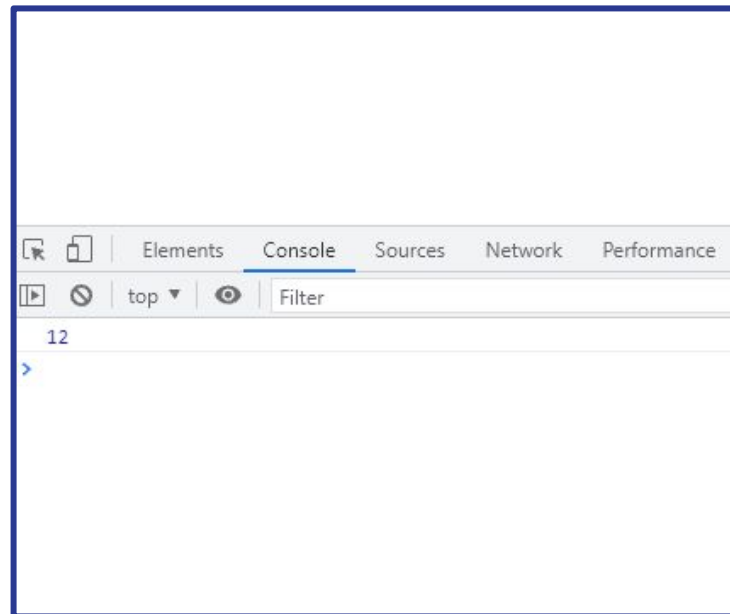
Uma função JavaScript é um bloco de código projetado para executar uma tarefa específica.

Uma função JavaScript é executada quando "algo" a invoca.

# Funções JavaScript

```
<!DOCTYPE html>
<html>
  <head>

  </head>
  <body>
    <script>
      function myFunction(p1, p2) {
        return p1 * p2;
      }
      console.log(myFunction(4, 3));
    </script>
  </body>
</html>
```



# Funções JavaScript

Sintaxe da função JavaScript.

- Uma função JavaScript é definida com a palavra chave **function**, seguida por um **nome** , seguida por parênteses ( ) .
- Os nomes das funções podem conter letras, dígitos, sublinhados e cifrões (mesmas regras das variáveis).
- Os parênteses podem incluir nomes de parâmetros separados por vírgulas:(**parâmetro1, parâmetro2, ...** )
- O código a ser executado, pela função, é colocado entre chaves: {}

# Funções JavaScript

```
<!DOCTYPE html>
<html>
  <head>

  </head>
  <body>
    <script>
      function nome(parametro1, parametro2, ... ){
        //código a ser executado
      }
    </script>
  </body>
</html>
```

# Funções JavaScript

## Invocação de Função

O código dentro da função será executado quando "algo" invocar (chamar) a função:

- Quando ocorre um evento (quando um usuário clica em um botão)
- Quando é invocado (chamado) a partir do código JavaScript
- Automaticamente (auto-invocado)



# Funções JavaScript

## Retorno de Função

Quando o JavaScript atinge uma instrução **return**, a função para de ser executada.

Se a função foi chamada a partir de uma instrução, o JavaScript "retorna" para executar o código após a instrução de chamada.

As funções geralmente calculam um valor de retorno. O valor de retorno é "retornado" de volta ao "chamador":

# Funções JavaScript

Exemplo: Calcule o produto de dois números e retorne o resultado:

```
<!DOCTYPE html>
<html>
  <head>

</head>
  <body>
    <script>
      var x = myFunction(4, 4);

      function myFunction(a, b) {
        return a * b;
      }
      console.log(x);
    </script>
  </body>
</html>
```

# Funções JavaScript

## **Por que funções?**

Você pode reutilizar o código: Define o código uma vez e use várias vezes.

Você pode usar o mesmo código muitas vezes com argumentos diferentes, para produzir resultados diferentes.

# Funções JavaScript

```
<!DOCTYPE html>
<html>
  <head>

  </head>
  <body>
    <script>
      function toCelsius(f) {
        return (5/9) * (f-32);
      }
      console.log(toCelsius(77));
    </script>
  </body>
</html>
```

# Funções JavaScript

## O operador () invoca a função

Usando o exemplo acima, **toCelsius** refere-se ao objeto de função e **toCelsius()** refere-se ao resultado da função.

Acessar uma função sem () retorna o objeto de função em vez do resultado da função.

# Funções JavaScript

```
<!DOCTYPE html>
<html>
  <head>

  </head>
  <body>
    <script>
      function toCelsius(f) {
        return (5/9) * (f-32);
      }
      console.log(toCelsius);
    </script>
  </body>
</html>
```

# Exercícios 1

Percorra todos os números de 1 até 100. Para os números ímpares, exiba no console um “\*”, e para os números pares, dois “\*\*”. Veja o exemplo abaixo:

## Exercícios 2

Percorra todos os números de 1 até 100. Para os números múltiplos de 4, exiba a palavra “PIN”, e para os outros, exiba o próprio número. Veja o exemplo abaixo



## Exercícios 3

Imprimir os números ímpares menores que o valor informado.

# JavaScript

O **SweetAlert2** é uma biblioteca JavaScript que nos auxilia na criação de alertas em nossas aplicações web.

<https://sweetalert2.github.io/>

Vamos vê a documentação.