

Roteiro Didático de Aula  
Aula 01 – JavaScript  
22/09/2020

Assuntos que serão abordados:

Configuração de Ambiente

Variáveis

Operadores

Funções

## Configuração de Ambiente

A configuração do ambiente para trabalhar com javascript é muito simples e intuitiva, é necessário apenas ter um editor de texto e um pouco de força de vontade. Aqui vão alguns exemplos de editores de texto que podem ser usados na manipulação do javascript:

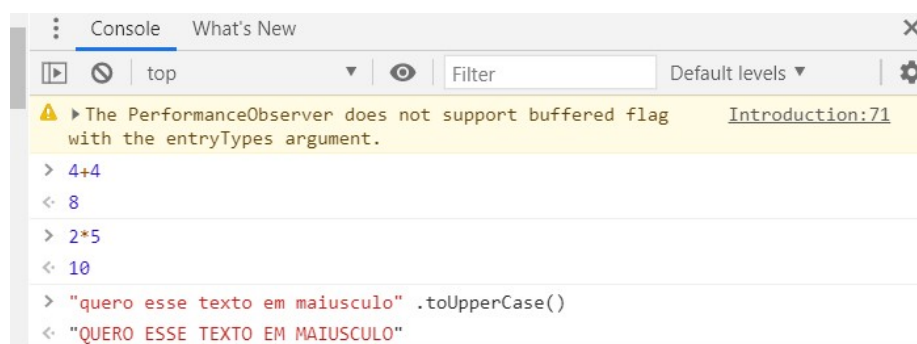
Vs code

Notepad++

Sublime

## Sintaxe

Podemos utilizar comandos javascript direto no console, entrando no modo desenvolvedor no nosso navegador.



O console nos permite testar códigos diretamente no navegador. Porém, não podemos pedir aos usuários do site que sempre abram o console, copiem um código e cole para ele ser executado. Para um código JavaScript ser executado na abertura de uma página, é necessário utilizar a tag <script>

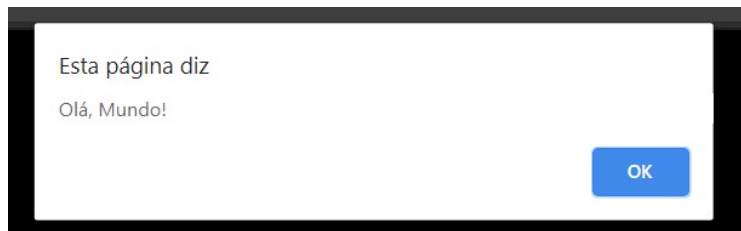
Sendo assim, vamos criar uma estrutura html básica para suportar o nosso JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Aula de JavaScript</title>
  </head>
  <body>
    <h1>JavaScript</h1>
  </body>
</html>
```

Vamos fazer nosso primeiro olá mundo

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Aula de JavaScript</title>
    <script>
      alert("Olá, Mundo!");
    </script>
  </head>
  <body>
    <h1>JavaScript</h1>
  </body>
</html>
```

Como resultado teremos:



Os **comentários de uma única linha** começam com //. Qualquer texto entre // e o final da linha será ignorado pelo JavaScript (não será executado).

```
<script>
  // alert("Olá, Mundo!");
</script>
```

Os **comentários de várias linhas** começam com /\*e terminam com \*/. Qualquer texto entre /\* e \*/ será ignorado pelo JavaScript.

```
<script>
  /*
    alert("Olá, Mundo!");
  */
</script>
```

O alert é uma das mais simples caixas de diálogo, com uma aparência simples e intuitiva elas são muito usadas em validações de formulários e/ou bloqueio de ações do browser.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Aula de JavaScript</title>
    <script>
      function funcao1()
      {
        alert("Eu sou um alert!");
      }
    </script>
  </head>
  <body>
    <h1>JavaScript</h1>
    <input type="button" onclick="funcao1()" value="Exibir Alert" />
  </body>
</html>
```

A função de confirmação é um pouco diferente da função alert em JavaScript, dessa vez são exibidos dois botões, um de OK e outro de CANCELAR, separados por valores true (verdadeiro) e false (falso).

A função confirm() é muito utilizada em sistemas que utilizamos estruturas condicionais, como confirmação de alteração/exclusão de algum registro do banco de dados.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Aula de JavaScript</title>
    <script>
      function funcao1()
      {
        var x;
        var r=confirm("Escolha um valor!");
        if (r==true)
        {
          x="você pressionou OK!";
        }
        else
        {
          x="Você pressionou Cancelar!";
        }
        document.getElementById("demo").innerHTML=x;
      }
    </script>
  </head>
  <body>
```

```
<h1>JavaScript</h1>
<input type="button" onclick="funcao1()" value="Exibir Confirm" />
<p id="demo"></p>
</body>
</html>
```

O prompt é um pouco diferente do alert() e do confirm(), pois ele necessita que o usuário insira algum valor, ou seja, precisa de uma interação direta do usuário para que ele funcione.

Para chamarmos a função utilizamos o prompt(), o qual irá receber uma string(mensagem) que será exibida, normalmente em forma de pergunta, ao usuário.

A estrutura básica dessa caixa de diálogo é:

Um campo input

Botão OK

Botão Cancelar

A função sempre irá retornar um valor, tudo que o usuário digitar no campo input será convertido em valor e será exibido na tela.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Aula de JavaScript</title>
  </head>
  <body>
    <p>Clique para exibir.</p>
    <button onclick="myFunction()">Clique aqui</button>
    <p id="demo"></p>
    <script>
      function myFunction()
      {
        var x;

        var idade=prompt("Digite sua idade:");

        if (idade!=null)
        {
          x="Idade: " + idade + " anos.";
          document.getElementById("demo").innerHTML=x;
        }
      }
    </script>
  </body>
</html>
```

## Variáveis

Existem três tipos de declarações em JavaScript.

### **var**

Declara uma variável, opcionalmente, inicializando-a com um valor.

### **let**

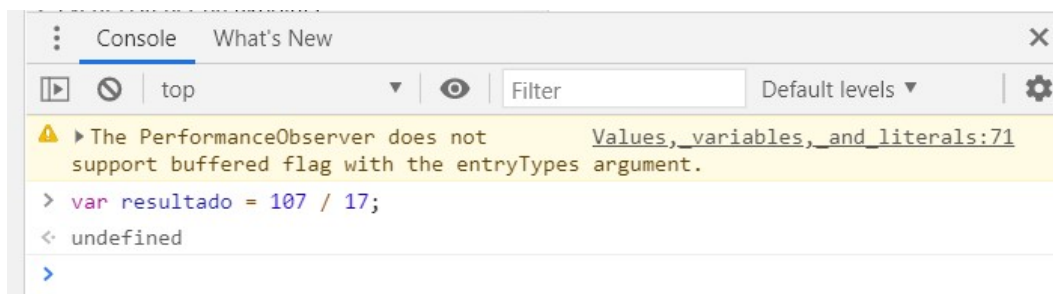
Declara uma variável local de escopo do bloco, opcionalmente, inicializando-a com um valor.

### **const**

Declara uma constante de escopo de bloco, apenas de leitura.

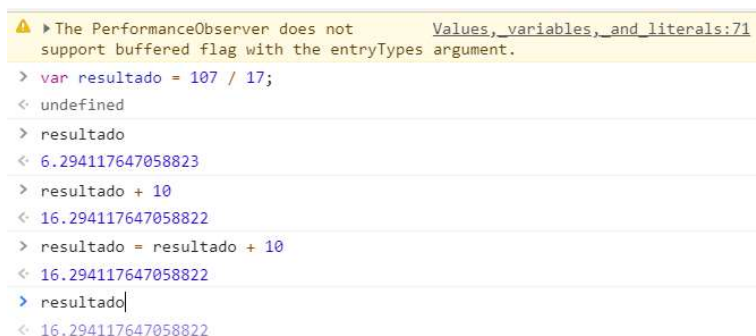
Um identificador JavaScript deve começar com uma letra, underline (\_), ou cifrão (\$); os caracteres subsequentes podem também ser números (0-9). Devido JavaScript ser case-sensitive, letras incluem caracteres de "A" a "Z" (maiúsculos) e caracteres de "a" a "z" (minúsculos).

Para armazenarmos um valor para uso posterior, podemos criar uma variável, então no console do navegador podemos digitar:



```
⋮ Console What's New X
[icon] [icon] top [icon] Filter Default levels [icon]
⚠ ▶ The PerformanceObserver does not support buffered flag with the entryTypes argument. Values, variables, and literals:71
> var resultado = 107 / 17;
< undefined
>
```

No exemplo acima, guardamos o resultado de 102/17 na variável resultado. O comportamento padrão ao criar uma variável no console é receber a mensagem: undefined. Para obter o valor que guardamos nela ou mudar o seu valor, digitamos seu nome no console, por exemplo:



```
⚠ ▶ The PerformanceObserver does not support buffered flag with the entryTypes argument. Values, variables, and literals:71
> var resultado = 107 / 17;
< undefined
> resultado
< 6.294117647058823
> resultado + 10
< 16.294117647058822
> resultado = resultado + 10
< 16.294117647058822
> resultado
< 16.294117647058822
```

O mais recente padrão ECMAScript define sete tipos de dados:

Seis tipos de dados são os chamados primitivos:

**Boolean.** true e false.

**null.** Uma palavra-chave que indica valor nulo. Devido JavaScript ser case-sensitive, null não é o mesmo que Null, NULL, ou ainda outra variação.

**undefined.** Uma propriedade superior cujo valor é indefinido.

**Number.** 42 ou 3.14159.

**String.** "Howdy"

**Symbol** (novo em ECMAScript 6). Um tipo de dado cuja as instâncias são únicas e imutáveis. e Object

Embora esses tipos de dados sejam uma quantidade relativamente pequena, eles permitem realizar funções úteis em suas aplicações. Objetos e funções são outros elementos fundamentais na linguagem. Você pode pensar em objetos como recipientes para os valores, e funções como métodos que suas aplicações podem executar.

## Operadores

Nome	Operador encurtado	Significado
Atribuição	$x = y$	$x = y$
Atribuição de adição	$x += y$	$x = x + y$
Atribuição de subtração	$x -= y$	$x = x - y$
Atribuição de multiplicação	$x *= y$	$x = x * y$
Atribuição de divisão	$x /= y$	$x = x / y$
Atribuição de resto	$x \% = y$	$x = x \% y$
Atribuição exponencial	$x ** = y$	$x = x ** y$
Atribuição bit-a-bit por deslocamento á esquerda	$x \ll = y$	$x = x \ll y$
Atribuição bit-a-bit por deslocamento á direita	$x \gg = y$	$x = x \gg y$
Atribuição de bit-a-bit deslocamento á direita não assinado	$x \ggg = y$	$x = x \ggg y$
Atribuição AND bit-a-bit	$x \& = y$	$x = x \& y$
Atribuição XOR bit-a-bit	$x \wedge = y$	$x = x \wedge y$
Atribuição OR bit-a-bit	$x   = y$	$x = x   y$

## Funções

A definição da função (também chamada de declaração de 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 {}.

Por exemplo, o código a seguir define uma função simples chamada square:

```
<script>
  function square(numero) {
    return numero * numero;
  }
</script>
```

Parâmetros primitivos (como um número) são passados para as funções por valor; o valor é passado para a função, mas se a função altera o valor do parâmetro, esta mudança não reflete globalmente ou na função chamada.

Se você passar um objeto (ou seja, um valor não primitivo, tal como Array ou um objeto definido por você) como um parâmetro e a função alterar as propriedades do objeto, essa mudança é visível fora da função, conforme mostrado no exemplo a seguir:

```
<script>
  function minhaFuncao(objeto) {
    objeto.make = "Toyota";
  }

  var meucarro = {make: "Honda", model: "Accord", year: 1998};
  var x, y;

  x = meucarro.make;    // x recebe o valor "Honda"

  minhaFuncao(meucarro);
  y = meucarro.make;    // y recebe o valor "Toyota"
                      // (a propriedade make foi alterada pela função)
</script>
```

A definição de uma função não a executa. Definir a função é simplesmente nomear a função e especificar o que fazer quando a função é chamada. Chamar a função executa realmente as ações especificadas com os parâmetros indicados. Por exemplo, se você definir a função square, você pode chamá-la do seguinte modo:

```
square(5);
```

A declaração anterior chama a função com o argumento 5. A função executa as instruções e retorna o valor 25.

Uma função pode chamar a si mesma. Por exemplo, a função que calcula os fatoriais recursivamente:

```
function fatorial(n){  
  if ((n == 0) || (n == 1))  
    return 1;  
  else  
    return (n * fatorial(n - 1));  
}
```

Você poderia, então, calcular os fatoriais de um a cinco:

```
var a, b, c, d, e;  
a = fatorial(1); // a recebe o valor 1  
b = fatorial(2); // b recebe o valor 2  
c = fatorial(3); // c recebe o valor 6  
d = fatorial(4); // d recebe o valor 24  
e = fatorial(5); // e recebe o valor 120
```

### 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 as variáveis e funções definidas 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. A função definida dentro de outra função também pode acessar todas as variáveis definidas na função hospedeira e outras variáveis ao qual a função hospedeira tem acesso.

```
// As seguintes variáveis são definidas no escopo global  
var num1 = 20,  
    num2 = 3,  
    nome = "Chamahk";  
  
// Esta função é definida no escopo global  
function multiplica() {  
  return num1 * num2;  
}  
  
multiplica(); // Retorna 60  
  
// Um exemplo de função aninhada  
function getScore () {  
  var num1 = 2,  
      num2 = 3;  
  
  function add() {  
    return nome + " scored " + (num1 + num2);  
  }  
  
  return add();  
}
```



}

getScore(); // Retorna "Chamahk scored 5"

## Funções pré-definidas

### `eval()`

O método `eval()` avalia código JavaScript representado como uma *string*.

### `isFinite()`

A função global `isFinite()` determina se o valor passado é um número finito. Se necessário, o parâmetro é primeiro convertido para um número.

### `parseFloat()`

A função `parseFloat()` analisa um argumento do tipo *string* e retorna um número de ponto flutuante.

### `parseInt()`

A função `parseInt()` analisa um argumento do tipo *string* e retorna um inteiro da base especificada (base do sistema numérico).

### `decodeURI()`

A função `decodeURI()` decodifica uma *Uniform Resource Identifier* (URI) criada anteriormente por `encodeURI` ou por uma rotina similar.

### `decodeURIComponent()`

O método `decodeURIComponent()` decodifica um componente *Uniform Resource Identifier* (URI) criado anteriormente por `encodeURIComponent` ou por uma rotina similar.

### `encodeURIComponent()`

O método `encodeURIComponent()` codifica um componente *Uniform Resource Identifier* (URI), substituindo cada ocorrência de determinados caracteres por um, dois, três, ou quatro sequências de escape que representa a codificação UTF-8 do caractere (só serão quatro sequências de escape para caracteres compostos de dois caracteres "substitutos").

## Referências

MDN. (s.d.). [https://developer.mozilla.org/pt-](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Fun%C3%A7%C3%B5es)

[BR/docs/Web/JavaScript/Guide/Fun%C3%A7%C3%B5es](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Fun%C3%A7%C3%B5es). Fonte: MDN web docs.