JS

Javascript

Fundamentos

Javascript

- Interpretada
- Baseada em objetos com funções de primeira classe
- Linguagem padrão de scripts para páginas web
- Extensivamente usada também em ambientes sem browser
- Baseada em protótipos, multiparadigma e dinâmica
- Orientação a objetos, declarativa, imperativa, funcional...

ECMAScript

```
1   ECMAScript 1 (1997) First Edition.
2   ECMAScript 2 (1998) Editorial changes only.
3   ECMAScript 3 (1999) Added Regular Expressions.
Added try/catch.

4   ECMAScript 4  Never released.

5   ECMAScript 5 (2009)
Added "strict mode".
Added JSON support.
Added String.trim().
Added Array.isArray().
Added Array Iteration Methods.
```

ECMAScript

```
5.1 ECMAScript 5.1 (2011)  Editorial changes.

6  ECMAScript 2015
Added let and const.
Added default parameter values.
Added Array.find().
Added Array.findIndex().

7  ECMAScript 2016  Added exponential operator (**).
Added Array.prototype.includes.
```

ECMAScript

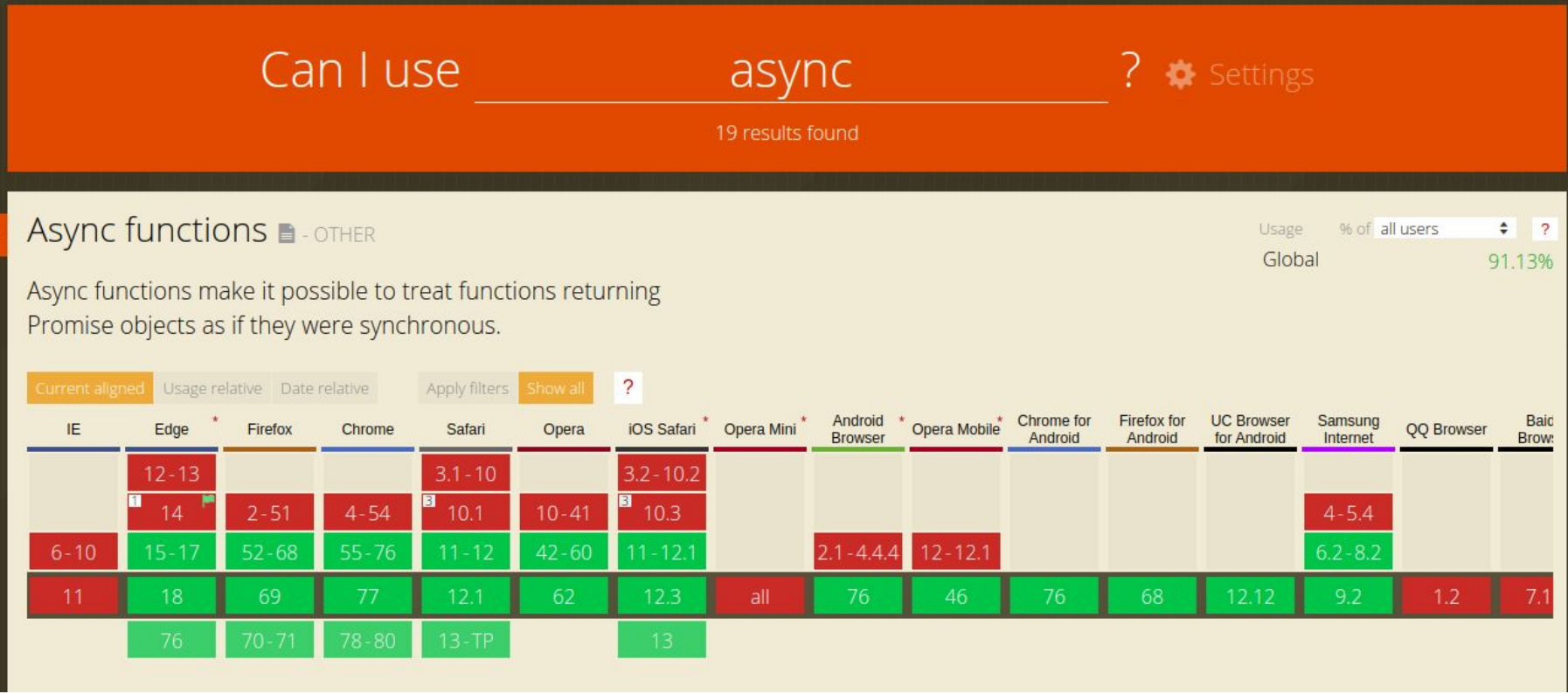
8 ECMAScript 2017 Added string padding.
Added new Object properties.
Added Async functions.
Added Shared Memory.

9 ECMAScript 2018 Added rest / spread properties.
Added Asynchronous iteration.
Added Promise.finally().
Additions to RegExp

ECMAScript

https://tc39.es/ecma262/

tip



https://caniuse.com/

can i use?

Desenvolvemos uma aplicação web que utiliza, entre outros:

- **async/await**
- **Promises**
- **for...in**
- **fetch**

Essa aplicação precisa ter uma cobertura de 90% dos navegadores. Estamos seguros pra este lançamento?

declaração de variáveis

var, let e const

```
var nome = 'Lucas'
let segundoNome = 'Daniel'
const sobrenome = 'Azambuja'

console.log(nome) // Lucas
console.log(segundoNome) // Daniel
console.log(sobrenome) // Azambuja
```

var, let e const

```
var exibeMensagem = function() {
    var mensagemForaDoIf = 'Lucas';
    if(true) {
        var mensagemDentroDoIf = 'Daniel';
        console.log(mensagemDentroDoIf)
    }
    console.log(mensagemForaDoIf);
    console.log(mensagemDentroDoIf);
}
```

var, let e const

Em JavaScript, blocos if e loops vazam variáveis do escopo.

```
var exibeMensagem = function() {
    var mensagemForaDoIf = 'Lucas';
    if(true) {
        var mensagemDentroDoIf = 'Daniel';
        console.log(mensagemDentroDoIf) // Daniel
    }
    console.log(mensagemForaDoIf); // Lucas
    console.log(mensagemDentroDoIf); // Daniel
}
```

var, let e const

```
var exibeMensagem = function() {
    mensagem = 'Lucas';
    console.log(mensagem);
    var mensagem;
}
```

var, let e const

Em JavaScript, toda variável é “elevada/içada” (hoisting) até o topo do seu contexto de execução.

```
// Utilização antes da declaração
var exibeMensagem = function() {
    mensagem = 'Lucas';
    console.log(mensagem);
    var mensagem;
}
```

var, let e const

```
var exibeMensagem = function() {
     if(true) {
          var escopoFuncao = 'Lucas';
          let escopoBloco = 'Daniel';

          console.log(escopoBloco); // Daniel
     }
     console.log(escopoFuncao); // Lucas
     console.log(escopoBloco);
}
```

var, let e const

```
void function(){
    const mensagem = 'Lucas';
    console.log(mensagem); // Lucas
    mensagem = Azambuja';
}();
```

var, let e const

keyword	const	let	var
global scope	NO	NO	YES
function scope	YES	YES	YES
block scope	YES	YES	NO
can be reassigned	NO	YES	YES

http://www.constletvar.com/

tipos de dados

tipos de dados

```
// tipagem dinâmica
let message = "hello";
message = 123456;
```

tipos de dados: number

```
// Number serve tanto pra números inteiros
// quanto para números reais
let n = 123;
n = 12.345;

// Tipos numéricos especiais: Infinity, -Infinity e NaN
alert( 1 / 0 ); // Infinity
alert( Infinity ); // Infinity

alert( "not a number" / 2 ); // NaN
// Qualquer operação que envolva NaN, resulta em NaN
alert( "not a number" / 2 + 5 );
// Operações matemáticas em JS são seguras. “Não dão erros”.
```

tipos de dados: number

No console, execute essas atribuições e cheque os valores:

```
let numero1 = 0888
let numero2 = 0777
```

O que tá acontecendo?

tipos de dados: string

```
let str = "Eita";
let str2 = 'Aspas simples funcionam também';
let phrase = `Acento grave (template strings) permite fazer algumas coisas legais. ${str}, que legal!`;

alert(`O resultado é ${1 + 2}`); // O resultado é 3
alert("O resultado é ${1 + 2}"); // O resultado é ${1 + 2} (aspas duplas)

// Não existe o tipo Char.
```

tipos de dados: boolean

```
let nameFieldChecked = true;
let ageFieldChecked = false;

// Também pode ser o resultado de uma operação
let isGreater = 4 > 1;
alert( isGreater );
```

tipos de dados: boolean

```
var feriado = new Boolean(true)
if (feriado) {
  console.log(“Opa, hoje é feriado!”)
}
```

tip: falsy

```
// Um valor falsy é um valor que se traduz em falso quando
avaliado em um contexto Boolean.
if (false)
if (null)
if (undefined)
if (0)
if (NaN)
if ('')
```

tipos de dados: Date (objeto, não tipo)

```
var dateObjectName = new Date([parameters]);

// Os parâmetros do código acima podem ser qualquer um a seguir:
// Nada: cria a data e hora de hoje. Por exemplo, today = new Date();.
// Uma string representando uma data da seguinte forma: "Mês dia, ano,
horas:minutos:segundos". Por exemplo, Xmas95 = new Date("25 de dezembro de 1995,
13:30:00"). Se você omitir as horas, minutos ou segundos, o valor será setado
para zero.
// Um conjunto de valores inteiros para ano, mês e dia. Por exemplo, var Xmas95
= new Date(1995, 11, 25).
// Um conjunto de valores inteiros par ano, mês, dia, hora, minuto e segundos.
Por exemplo, var Xmas95 = new Date(1995, 11, 25, 9, 30, 0);.
```

tipos de dados: Date

```
var hoje = new Date();
var fimAno = new Date(2019, 11, 31, 23, 59, 59, 999); // Seta dia e mês
fimAno.setFullYear(hoje.getFullYear());  // Seta o ano para esse ano
var msPorDia = 24 * 60 * 60 * 1000; // Quantidade de milisegundos por dia
var diasRestantes = (fimAno.getTime() - hoje.getTime()) / msPorDia;
var diasRestantes = Math.round(diasRestantes);  //retorna os dias restantes no
ano

/*
Segundos e minutos: de 0 a 59
Horas: de 0 a 23
Dia: 0 (Domingo) a 6 (Sábado)
Data: 1 a 31 (dia do mês)
Meses: 0 (Janeiro) a 11 (Dezembro)
Ano: anos desde 1900
```

tipos de dados: null

```
let age = null;
// Não necessariamente um “null pointer”, é só um valor vazio
ou não existente. Precisa ser assinalado.
```

tipos de dados: undefined

```
let age;
alert( age );

// Similar ao null, mas tipicamente quer dizer que uma variável foi declarada mas não inicializada/definida.
```

tipos de dados: Object

```
let user = new Object(); // "object constructor" syntax
let user = {};  // "object literal" syntax

let user = {
  name: "John",
  age: 30,
  "likes birds": true
};

user["likes birds"] = false;

delete user.age;
```

operadores

operadores: atribuição

```
let x = y; // Atribuição
let x += y // Atribuição de adição: let x = x + y
let x -= y // Atribuição de subtração: let x = x - y
let x *= y // Atribuição de multiplicação: let x = x * y
let x /= y // Atribuição de divisão: let x = x / y
let x %= y // Atribuição de resto
```

operadores: comparação

```
// Igual, onde os operandos são iguais
3 == 3 // true
"3" == 3 // true
3 == '3' // true

// Estritamente igual, onde os operandos e tipos são iguais
3 === 3 // true
"3" === 3 // false
3 === '3' // false
```

operadores: comparação

```
// Não igual, onde os operandos são diferentes
3 != 3 // false
"3" != 3 // false
3 != '3' // false

// Estritamente não igual, onde os operandos e/ou tipos são diferentes
3 !== 3 // false
"3" !== 3 // true
3 !== '3' // true
```

operadores: comparação

```
// Maior que, maior ou igual
12 > 10 // true
"15" > 12 // true
3 >= 4 // false

// Menor que, menor ou igual
12 < 10 // false
"15" < 12 // false
3 <= 4 // true
```

operadores: aritméticos

```
2 + 3 // 4
2 - 3 // -1
2 * 3 // 6
2 / 3 // 0.6666666666666666
```

operadores: lógicos

```
// E (AND)
var a1 = true && true;      // t && t retorna true
var a2 = true && false;     // t && f retorna false
var a3 = false && true;      // f && t retorna false
var a4 = false && (3 == 4); // f && f retorna false
var a5 = "Gato" && "Cão";   // t && t retorna Cão
var a6 = false && "Gato";   // f && t retorna false
var a7 = "Gato" && false;   // t && f retorna false
```

operadores: lógicos

```
// OU (OR)
var o1 =  true || true;      // t || t retorna true
var o2 = false || true;      // f || t retorna true
var o3 =  true || false;     // t || f retorna true
var o4 = false || (3 == 4); // f || f retorna false
var o5 = "Gato" || "Cão";    // t || t retorna Gato
var o6 = false || "Gato";    // f || t retorna Gato
var o7 = "Gato" || false;    // t || f retorna Gato
```

operadores: lógicos

```
// !negação lógica
var n1 = !true;    // !t retorna false
var n2 = !false;   // !f retorna true
var n3 = !"Gato"; // !t retorna false
```

operadores: strings

```
console.log("minha " + "string");
// exibe a string "minha string".

var minhaString = "alfa";
minhaString += "beto";
// É avaliada como "alfabeto" e atribui este valor a
minhastring.
```

operadores: ternários

```
condicao ? valor1 : valor2
```

```
var status = (idade >= 18) ? "adulto" : "menor de idade";
```

operadores: typeof

```
typeof operando
typeof (operando)

var meuLazer = new Function("5 + 2");
var forma = "redondo";
var tamanho = 1;
var hoje = new Date();

typeof meuLazer;  // retorna "function"
typeof forma;     // retorna "string"
typeof tamanho;   // retorna "number"
typeof hoje;      // retorna "object"
typeof naoExiste; // retorna "undefined"
```

operadores: typeof

```
typeof true; // retorna "boolean"
typeof null; // retorna "object"

typeof 62;           // retorna "number"
typeof 'Olá mundo'; // retorna "string"

typeof document.lastModified; // retorna "string"
typeof window.length;         // retorna "number"
typeof Math.LN2;              // retorna "number"
```

operadores: spread

```
var partes = ['ombro', 'joelhos'];
var musica = ['cabeca', ...partes, 'e', 'pés'];
```

tip: falsy

```
// Um valor falsy é um valor que se traduz em falso quando
avaliado em um contexto Boolean.
if (false)
if (null)
if (undefined)
if (0)
if (NaN)
if ('')
```

convertendo números

```
// Convertendo uma string para inteiro
var text = '42px';
var integer = parseInt(text, 10);
// retorna 42
```

convertendo números

```
// Convertendo uma string para Float
var text = '3.14someRandomStuff';
var pointNum = parseFloat(text);
// retorna 3.14
```

Pergunte pro usuário 2 números, e mostre em seguida a soma deles.

Utilize os métodos prompt e alert pra ler e mostrar os dados.

```
var nome = prompt('Digite seu nome')
alert('seu nome é ' + nome)
```

Controle de fluxo

controle de fluxo: blocos

```
// Uma declaração em bloco é utilizada para agrupar
declarações. O bloco é delimitado por um par de chaves

{
   declaracao_1;
   declaracao_2;
   .
   .
   .
   declaracao_n;
}
```

controle de fluxo: blocos

```
while (x < 10) {
  x++;
}

// Aqui, { x++; } é a declaração de bloco.
```

controle de fluxo: condicionais

```
// Use a declaração if para executar alguma declaração caso a condição lógica for verdadeira.

// Use a cláusula opcional else para executar alguma declaração caso a condição lógica for falsa.

if (condicao) {
  declaracao_1;
} else {
  declaracao_2;
}
```

controle de fluxo: condicionais

```
// Você pode também combinar declarações utilizando else if
para obter várias condições testadas em sequência, como o
seguinte:

if (condicao) {
  declaracao_1;
} else if (condicao_2) {
  declaracao_2;
} else if (condicao_n) {
  declaracao_n;
} else {
  declaracao_final;
}
```

controle de fluxo: condicionais

```
// Uma declaração switch permite que um programa avalie uma expressão e tente
associar o valor da expressão ao rótulo de um case. Se uma correspondência é
encontrada, o programa executa a declaração associada.

switch (expressao) {
   case rotulo_1:
      declaracoes_1
      [break;]
   case rotulo_2:
      declaracoes_2
      [break;]
   ...
   default:
      declaracoes_padrao
      [break;]
}
```

controle de fluxo: condicionais

```
switch (tipofruta) {
    case "Laranja":
        console.log("O quilo da laranja está R$0,59.<br>");
        break;
    case "Maçã":
        console.log("O quilo da maçã está R$0,32.<br>");
        break;
    default:
        console.log("Desculpe, não temos " + tipofruta +
".<br>");
}
console.log("Gostaria de mais alguma coisa?<br>");
```

controle de fluxo: repetições for

```
// Um laço for é repetido até que a condição especificada seja falsa. O laço for no JavaScript é similar ao Java e C. Uma declaração for é feita da seguinte maneira:

for ([expressaoInicial]; [condicao]; [incremento])
  declaracao
```

controle de fluxo: repetições for

```
// Quando um for é executado, ocorre o seguinte:
// 1 - A expressão expressao Inicial é inicializada e, caso possível, é executada. Normalmente essa expressão inicializa um ou mais contadores, mas a sintaxe permite expressões de qualquer grau de complexidade. Podendo conter também declaração de variáveis.
// 2 - A expressão condicao é avaliada. caso o resultado de condicao seja verdadeiro, o laço é executado. Se o valor de condicao é falso, então o laço terminará. Se a expressão condicao é omitida, a condicao é assumida como verdadeira.
// 3 - A instrução é executada. Para executar múltiplas declarações, use uma declaração em bloco ({ ... }) para agrupá-las.
// A atualização da expressão incremento, se houver, executa, e retorna o controle para o passo 2.
```

controle de fluxo: repetições for

```
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

controle de fluxo: repetições do..while

```
// A instrução do...while repetirá até que a condição
especificada seja falsa.

do
  declaracao
while (condicao);
```

Pergunte pro usuário uma quantidade de notas escolares, em seguida, pergunte o valor de todas as notas e mostre pra eles a média simples das notas.

Utilize os métodos prompt e alert pra ler e mostrar os dados.

controle de fluxo: repetições while

```
// Uma declaração while executa suas instruções, desde que uma condição especificada seja avaliada como verdadeira. Segue uma declaração while:

while (condicao)
  declaracao

// Exemplo
n = 0;
x = 0;
while (n < 3) {
  n++;
  x += n;
}
```

controle de fluxo: repetições, labels

// Um label provê um identificador que permite que este seja referenciado em outro lugar no seu programa. Por exemplo, você pode usar uma label para identificar um laço, e então usar **break** ou **continue** para indicar quando o programa deverá interromper o laço ou continuar sua execução.

```
markLoop: while (theMark == true) {
   facaAlgo();
}
```

controle de fluxo: repetições, break

```
// Use break para terminar laços, switch, ou um conjunto que
utiliza label.

for (i = 0; i < a.length; i++) {
  if (a[i] == theValue) {
    break;
  }
}
```

controle de fluxo: repetições, break label

```
var x = 0;
var z = 0
labelCancelaLaco: while (true) {
  console.log("Laço exterior: " + x);
  x += 1;
  z = 1;
  while (true) {
    console.log("Laço interior: " + z);
    z += 1;
    if (z === 10 && x === 10) {
      break labelCancelaLaco;
    } else if (z === 10) {
      break;
    }
  }
}
```

controle de fluxo: repetições, continue

// Quando você utiliza **continue** sem uma label, ele encerrará a iteração atual mais interna de uma instrução while, do-while, ou for e continuará a execução do laço a partir da próxima iteração. Ao contrário da instrução break, continue não encerra a execução completa do laço. Em um laço while, ele voltará para a condição. Em um laço for, ele pulará para a expressão de incrementação.

```
i = 0;
n = 0;
while (i < 5) {
  i++;
  if (i == 3) {
    continue;
  }
  n += i;
}
```

controle de fluxo: repetições for..in

```
// A declaração for...in executa iterações a partir de uma variável específica, percorrendo todas as propriedades de um objeto. Para cada propriedade distinta, o JavaScript executará uma iteração.

for (variavel in objeto) {
  declaracoes
}
```

controle de fluxo: repetições for..in

Abrir o site da Unoeste e mostrar os tipos e nomes das propriedades do objeto window. Ex:

function -> postMessage
function -> blur
function -> focus
object -> parent
number -> length

controle de fluxo: repetições for..of

```
// A declaração for...of cria uma laço com objetos interativos
(incluindo, Array, Map, Set, assim por diante), executando uma
iteração para o valor de cada propriedade distinta.

for (variavel of objeto) {
  declaracoes
}
```

controle de fluxo: repetições for..in vs for..of

```
let arr = [3, 5, 7];
arr.foo = "hello";

for (let i in arr) {
   console.log(i); // logs "0", "1", "2", "foo"
}

for (let i of arr) {
   console.log(i); // logs "3", "5", "7"
}
```

Arrays

Arrays: declaração

```
// Um array é um tipo especial de variável que pode ter mais de um valor

var cars = new Array(3);

var cars = new Array("Saab", "Volvo", "BMW");

var cars = ["Saab", "Volvo", "BMW"];
```

Arrays: elementos podem ser objetos

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
myArray[3] = 4;
myArray[4] = function () { return "eita" };
```

Arrays: métodos e propriedades

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length; // the length of fruits is 4
fruits.sort(); // ["Apple", "Banana", "Mango", "Orange"]
```

Arrays: acessando os valores

```
var cars = ["Saab", "Volvo", "BMW"];
var first = cars[0]; // Saab
var last = cars[cars.length - 1]; // BMW
```

Arrays: alterando um elemento

```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
```

Arrays: iterando

```
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

for (i = 0; i < fLen; i++) {
  console.log(`Fruta ${i + 1}: ${fruits[i]}`);
}
```

Arrays: iterando com forEach

```
var fruits;
fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.forEach(writeFruit)

function writeFruit(fruit, index) {
    console.log(`Fruta: ${index + 1}: ${fruit}`)
}
```

Arrays: adicionando elementos

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

// Utilizando o método push
fruits.push("Lemon");

// Utilizando o índice
fruits[fruits.length] = "Strawberry";
```

Arrays: removendo elementos do final

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

var last = fruits.pop();

// fruits: ["Banana", "Orange", "Apple"]
// last: "Mango"
```

Arrays: removendo elementos do início

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

var first = fruits.shift();

// fruits: ["Orange", "Apple", "Mango"]
// first: "Banana"
```

Arrays: adicionando elementos no início

```
var fruits = ["Orange", "Apple", "Mango"];

fruits.unshift("Banana");

// fruits: ["Banana", "Orange", "Apple", "Mango"]
```

Arrays: procurando o índice de um item

```
var fruits = ["Orange", "Apple", "Mango"];

var appleIndex = fruits.indexOf("Apple");

// appleIndex: 1
```

Arrays: removendo itens pela posição do index

```
var fruits = ["Orange", "Apple", "Mango"];

var appleIndex = fruits.indexOf("Apple");

fruits.splice(appleIndex, 1);
```

Pergunte pro usuário o valor 5 notas escolares, em seguida mostre pra eles a média simples das 3 maiores notas.

Utilize os métodos prompt e alert pra ler e mostrar os dados.

Funções e OO

Funções

```
// Funções são blocos de códigos designados a executar uma
tarefa específica, e só são executadas quando algo chama ela.

function nome(parametro1, parametro2) {
    // código que será executado
}

function soma (n1, n2) {
    return n1 + n2
}
```

Funções: retorno

```
// Quando o javascript atinge um comando return, ele para a execução da função. Se a função foi chamada em uma declaração, o javascript vai executar a função e então retornar o resultado.

let area = calculaArea(base, altura)

function calculaArea(base, altura) {
   return base * altura
}
```

Funções: Orientação a Objeto

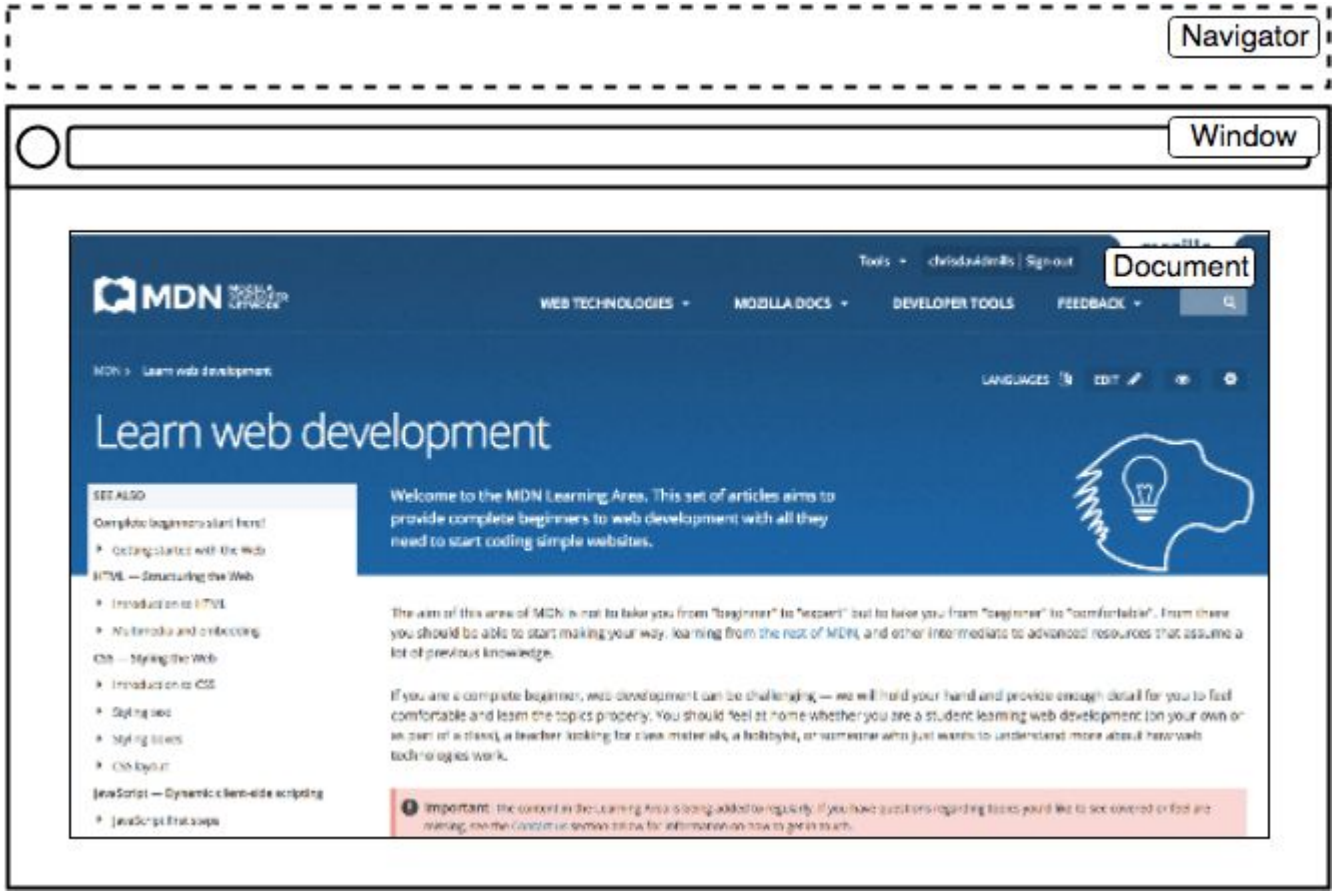
```
// O javascript usa constructor functions para definir e
inicializar objetos.

function Person(name) {
  this.name = name;
  this.greeting = function() {
    alert('Hi! I\'m ' + this.name + '.');
  };
}

var person = new Person('Lucas')
person.name // Lucas
person.greeting()
```

DOM

DOM


Navigator
Window
Document
MDN
WEB TECHNOLOGIES
MOZILLA DOCS
DEVELOPER TOOLS
FEEDBACK
Learn web development
Welcome to the MDN Learning Area. This set of articles aims to provide complete beginners to web development with all they need to start coding simple websites.

DOM: Window

Window é a aba do navegador em que a página WEB foi carregada; isso é representado em JavaScript pelo objeto Window. Usando os métodos disponíveis nesse objeto, você pode fazer coisas como retornar o tamanho da janela (consulte Window.innerWidth e Window.innerHeight), manipular o documento carregado nessa janela, armazenar dados específicos para esse documento no lado do cliente (por exemplo, usando um banco de dados local ou outro mecanismo de armazenamento), anexar um manipulador de eventos à janela...

DOM: Navigator

O Navigator representa o estado e a identidade do navegador (ou seja, o agente do usuário). Em JavaScript, isso é representado pelo objeto Navigator. Você pode usar esse objeto para recuperar coisas como o idioma preferido do usuário, um fluxo de mídia da webcam do usuário etc.

DOM: Document

O documento (representado pelo DOM nos navegadores) é a página real carregada na janela e é representada em JavaScript pelo objeto Document. Você pode usar esse objeto para retornar e manipular informações no HTML e CSS que compõem o documento, por exemplo, obter uma referência a um elemento no DOM, alterar seu conteúdo de texto, aplicar novos estilos a ele, criar novos elementos, criar novos elementos e adicioná-los a o elemento atual como filhos ou até mesmo excluí-lo completamente.

DOM: Document

```
<!DOCTYPE html>
<html>

<head>
      <meta charset="utf-8">
      <title>Simple DOM example</title>
</head>

<body>
      <section>
            <img src="dinosaur.png" alt="A red Tyrannosaurus Rex: A two legged dinosaur standing upright like a human, with small arms, and a large head with lots of sharp teeth." >
            <p>Here we will add a link to the <a href="https://www.mozilla.org/" >Mozilla homepage</a>
            </p>
      </section>
</body>

</html>
```

DOM: Árvore

```
DOCTYPE: html
HTML
  HEAD
    #text:
    META charset="utf-8"
    #text:
    TITLE
      #text: Simple DOM example
    #text:
  #text:
  BODY
    #text:
    SECTION
      #text:
      IMG src="dinosaur.png" alt="A red Tyrannosaurus Rex: A two legged dinosaur standing
      upright like a human, with small arms, and a large head with lots of sharp teeth."
      #text:
      P
        #text: Here we will add a link to the
        A href="https://www.mozilla.org/"
          #text: Mozilla homepage
        #text:
    #text:
```

DOM: Terminologia

Nó do elemento (node): um elemento, como existe no DOM.

Nó raiz (root node): o nó superior da árvore, que no caso do HTML é sempre o nó HTML (outros vocabulários de marcação como SVG e XML personalizado terão diferentes elementos raiz).

Nó filho (child node): um nó diretamente dentro de outro nó. Por exemplo, IMG é filho de SECTION no exemplo acima.

DOM: Terminologia

Nó descendente (descendant node): um nó em qualquer lugar dentro de outro nó. Por exemplo, IMG é filho de SECTION no exemplo anterior e também é descendente. O IMG não é filho do BODY, pois está dois níveis abaixo dele na árvore, mas é um descendente do BODY.

Nó pai (parent node): um nó que possui outro nó dentro dele. Por exemplo, BODY é o nó pai de SECTION no exemplo anterior.

Nós irmãos (sibling nodes): nós que ficam no mesmo nível na árvore DOM. Por exemplo, IMG e P são irmãos no exemplo anterior.

Nó de texto (text node): um nó que contém uma sequência de texto.

DOM: Manipulando

```
// Para manipular elementos dentro do DOM, precisamos
referenciar ou guardar em uma variável:

var link = document.querySelector('a');

link.textContent = 'Mudei o texto do link';

link.href = 'https://www.google.com'

// document.querySelector(‘<CSS SELECTOR>’)
// document.querySelectorAll(‘<CSS SELECTOR>’)
```

DOM: Manipulando

```
// selecionando uma seção
var sect = document.querySelector('section');

// Criando um parágrafo através do método createElement
var para = document.createElement('p');
para.textContent = 'We hope you enjoyed the ride.';

// Adicionando a seção ao parágrafo através do método
appendChild
sect.appendChild(para);
```

DOM: Manipulando

```
// Criando um nó de texto
var text = document.createTextNode(' — the premier source for
web development knowledge.');

// Adicionando dentro de um parágrafo
var linkPara = document.querySelector('p');
linkPara.appendChild(text);
```

DOM: Manipulando

```
// Adicionando
sect.appendChild(linkPara);

// Removendo
sect.removeChild(linkPara);

// Removendo baseado somente no elemento em si
linkPara.parentNode.removeChild(linkPara);
```

DOM: Manipulando

```
var para = document.querySelector('p');

para.style.color = 'white';
para.style.backgroundColor = 'black';
para.style.padding = '10px';
para.style.width = '250px';
para.style.textAlign = 'center';

para.setAttribute('class', 'highlight');
```

Local Storage

Local Storage

```
// A interface de Armazenamento da Web Storage API fornece
acesso ao armazenamento de sessão ou armazenamento local para
um domínio específico, permitindo que você, por exemplo,
adicione, modifique ou exclua itens de dados armazenados.

// Gravando um item
localStorage.setItem('Nome', 'Lucas');

// Buscando um item
var meuItem = localStorage.getItem('Nome');
```

Fetch

Fetch

```
// A API Fetch fornece uma interface para acessar e manipular partes do pipeline HTTP, tais como os pedidos e respostas. Ela também fornece o método global fetch() que fornece uma maneira fácil e lógica para buscar recursos de forma assíncrona através da rede.

fetch('https://swapi.co/api/people/')
.then(function(response) {
  return response.json();
})
.then(function(personagens) {
  console.log(personagens)
});
```

Fetch

```
// Enviando POST JSON

fetch('http://localhost:3000/api', { method: 'POST', headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    }, body: JSON.stringify({ nome: 'Lucas Azambuja' }), mode:
'cors' })
.then(function(response) {
  return response.json();
})
.then(function(resposta) {
  console.log(resposta)
});
```

Promises

Promises

```
// Uma Promise é um objeto que representa a eventual conclusão
ou falha de uma operação assíncrona.

new Promise(function(funcaoResolve, funcaoRejeita) {
  if (deucerto) {
    funcaoResolve(objeto)
  } else {
    funcaoRejeita(objeto)
  }
})
```