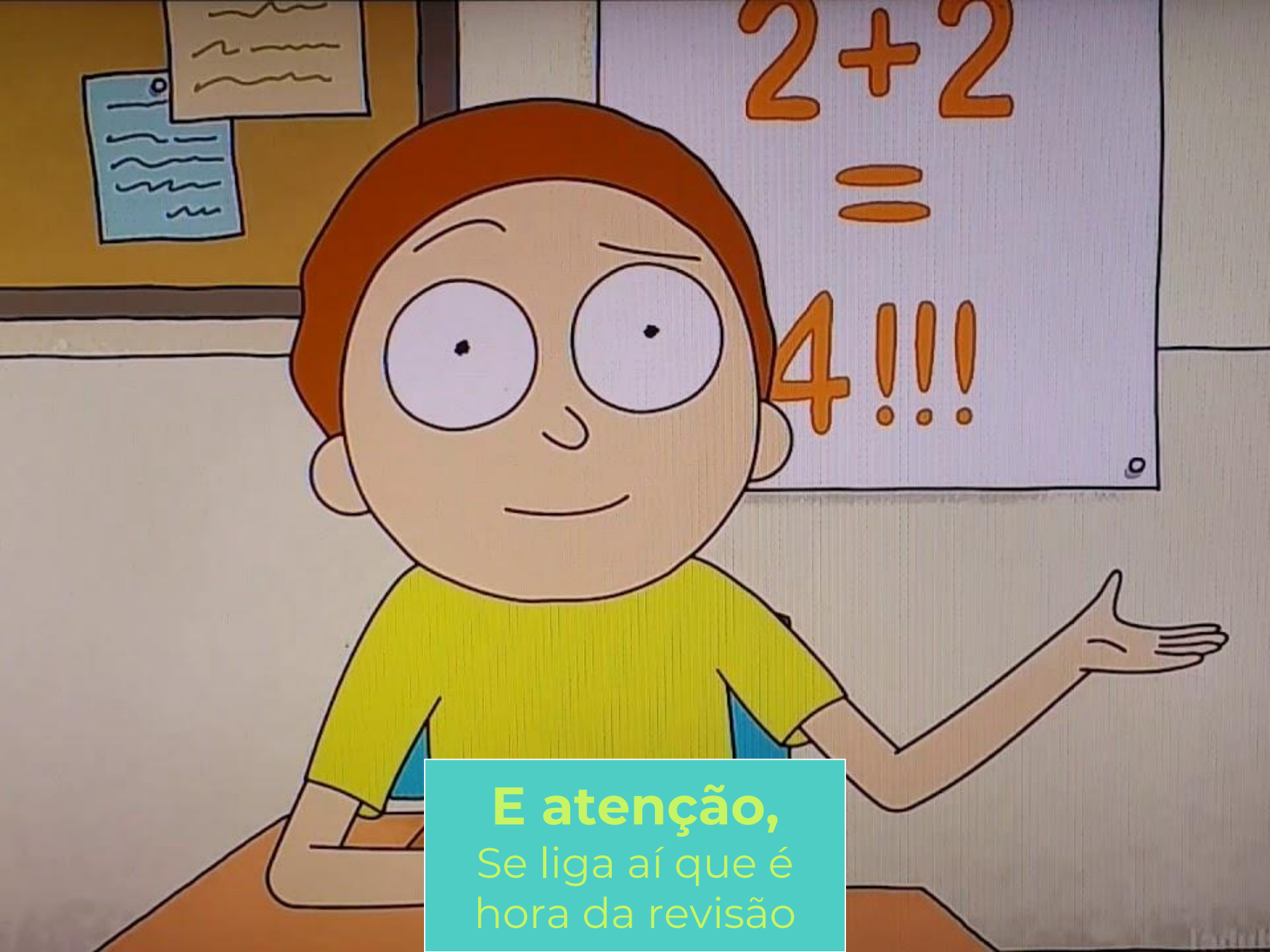
Olá!


2+2
=
4!!!
E atenção,
Se liga aí que é
hora da revisão

Lembrando...

1. JSON
2. JavaScript API
    a. String, number, date

JavaScript
Aula 11

1.

Map, reduce, filter

Vamos aprender três funções essenciais quando falamos de arrays.

Arrays - Funções

Arrays são amplamente utilizados em projetos que contém JavaScript, pela versatilidade que apresenta e suas diversas formas de uso. Para facilitar o desenvolvimento, algumas funções primordiais são definidas por padrão.

Map

A função map aplica uma função a todos os elementos do array, resultando um array modificado mas com a mesma quantidade de elementos do original.

```
var fahrenheit = [ 0, 32, 45, 50, 75, 80, 99, 120 ];

var celcius = fahrenheit.map( function( elem ) {
    return Math.round( ( elem - 32 ) * 5 / 9 );
} );
```

Filter

A função filter aplica um filtro sobre o array original, fazendo uma comparação a cada elemento, retornando um array de quantidade menor ou igual ao array original.

```
var numbers = [65, 44, 12, 4];

numbers.filter(function (number) {
    return number >= 15;
};
```

Reduce

A função reduce aplica uma operação sobre um array, resultando em apenas um único valor.

```
var numbers = [1, 2, 3, 4, 5];

numbers.reduce(function(previous, next){
    return previous + next;
}
```

Exercício 1

1. Você irá fazer uma festa, mas a mesma possuirá regras muito rígidas para que apenas um grupo seleto de pessoas possam ir nela. Acabaram de lhe passar uma lista de pessoas interessadas, e você deverá:
   a. Para cada pessoa da lista, que possui um nome e um ano de nascimento, calcular seu aniversário e adicionar como propriedade do objeto.
   b. Depois, filtre apenas pessoas que tenham mais de 18 anos
   c. Por fim, calcule a média de idade de todas as pessoas da festa.

```
var guests = [
  {name: "Maíra", birthYear: "1994"},
  {name: "José", birthYear: "2008"},
  {name: "Antônio", birthYear: "2000"},
  {name: "Sérgio", birthYear: "1990"}
];
```

2.

ECMAScript 6

Vamos aprender o novo padrão do JavaScript.

ECMA 6

ECMA 6 nada mais é do que a versão mais atual do JavaScript, que veio para corrigir alguns problemas da versão anterior, trazer melhorias de sintaxe e novas funcionalidades para facilitar o desenvolvimento de programas complexos.

O que há de novo?

1. Constantes
2. Variáveis com escopo
3. Arrow functions
4. Funcionalidades extendidas para parâmetros
5. Template String
6. Funções como propriedades
7. Desestruturação de objetos
8. Exportar/Importar componentes JS
9. Classes

Constantes

Quando queremos criar valores imutáveis, agora podemos usar constantes para tal finalidade.

```
const b = 1
b = 2 // Uncaught SyntaxError "b" is read-only
```

Mas se criarmos um objeto constante com uma propriedade, e tentar alterar o valor de tal propriedade, o que acontece?

Variáveis de escopo

A nova versão do JavaScript trouxe o conceito de variáveis com escopo de utilização. Para isso, usamos a palavra reservada let, para declarar nossas variáveis.

```
// escopo de função com var
function doSomething() {
  var a = 1;
  if (true) {
    var b = 2;
  }
  var c = a + b;
}
```

```
//escopo de bloco com let
function doSomethingElse() {
  let a = 1
  if (true) {
    let b = 2
  }
  let c = a + b
}
```

Arrow Functions

As funções agora podem ser escritas de maneira mais simples e direta, trazendo uma certa familiaridade estética com programação funcional.

O nome "arrow function" vem da sintaxe da notação, onde uma seta "=>" é usada para separar parâmetros de lógica e resultado da função.

Arrow Functions

ECMA 5

```
odds  = evens.map(function (v) { return v + 1; });
pairs = evens.map(function (v) { return { even: v, odd: v + 1 }; });
nums  = evens.map(function (v, i) { return v + i; });
```

ECMA 6

```
odds  = evens.map(v => v + 1)
pairs = evens.map(v => ({ even: v, odd: v + 1 }))
nums  = evens.map((v, i) => v + i)
```

Arrow Functions

Lembrem-se:

1. Se a função possuir mais de um parâmetro, deverá estar separado por vírgula, entre parênteses.
2. Se o retorno da função não possuir mais de uma linha, deverá vir entre chaves
3. A palavra reservada return só é necessária caso o processamento da função tenha mais de uma linha.

Arrow Functions

```
let newWay = (name, nickname) => {
  return 'My name is ' + nickname + ', ' + name;
};

console.log( newWay('James Bond', 'Bond') );
// My name is Bond, James Bond
```

Funcionalidades extendidas para parâmetros

Na nova versão do JavaScript, parâmetros de funções podem receber valores default e operador de espalhamento.

```
function f (x, y = 7, z = 42) {
    return x + y + z
}
f(1) === 50
```

```
var params = [ "hello", true, 7 ]
var other = [ 1, 2, ...params ] // [ 1, 2, "hello", true, 7 ]
```

Template string

Agora é possível construir strings de forma mais fluida e direta utilizando template string.

```
var customer = { name: "Foo" }
var card = { amount: 7, product: "Bar", unitprice: 42 }
var message = `Hello ${customer.name},
want to buy ${card.amount} ${card.product} for
a total of ${card.amount * card.unitprice} bucks?`
```

${} indica que você processará um valor JavaScript

Template String começa termina com crase

Funções como propriedade

Funções podem ser aplicadas diretamente como parâmetro, sem a necessidade de declaração de uma propridade para atribuição.

```
obj = {
    foo: function (a, b) {
        ...
    },
    bar: function (x, y) {
        ...
    }

    ...
};
```

```
obj = {
    foo (a, b) {
        ...
    },
    bar (x, y) {
        ...
    }

}
```

Desestruturação de objetos

A partir de agora é possível desestruturar objetos utilizando apenas o nome das propriedades.

```
let person = {name: "João", age: 23};
let {name, age} = person;
console.log(name, age);
```

Exportar / importar componentes JS

A partir de agora é possível desestruturar objetos utilizando apenas o nome das propriedades.

```
//  lib/math.js
export function sum (x, y) { return x + y }
export var pi = 3.141593

//  someApp.js
import * as math from "lib/math"
console.log("2π = " + math.sum(math.pi, math.pi))

//  otherApp.js
import { sum, pi } from "lib/math"
console.log("2π = " + sum(pi, pi))
```

Classes

Classes nos dão uma sintaxe amigável que definem o estado e o comportamento de objetos que representam as abstrações que usamos diariamente.

Classes

```
class Animal {
  constructor(name) {
    this._name = name
  }

  getName() {
    return this._name
  }

  setName(name) {
    this._name = name
  }
}

const animal = new Animal('dog')
animal.getName() // dog
animal.setName('cat')
animal.getName() // cat
```

Exercício 2

1. Adapte a resposta do exercício 1 para passar a utilizar JavaScript 6

Exercício 3

1. Utilizando JavaScript 6:
    a. Crie uma função que recebe como parâmetro duas Strings
        i. Essa função deverá concatenar os parâmetros, separando eles por " --- ".
    b. Crie um array de Strings que deverá ser varrido e todas as strings, ao final, deverão estar concatenadas e separadas por vírgula (usando a função criada anteriormente).

Dúvidas?

github.com/drcabral

twitter.com/DrCabrales

diogo.cabral.dev@gmail.com