

# JS PRO

Bem vindos ao curso de Javascript Pro

Turma JS PRO 1605873

# Quem sou eu

- 🇧🇷 Leonardo Redin
- 👤 25 anos
- 💻 Desenvolvedor full stack JS
- 👔 Atualmente trabalhando para a Renner

## Hobbies e curiosidades sobre mim

- Tenho 13 🐯
- Tenho 04 🐕
- Toco 🎸
- Sou goleiro 🏆
- Amo cozinhar 🍲

# O que vamos aprender:

- Declaração de variáveis [3-6]
- Strings [7-15]
- Operadores [16-24]
- Objects [25-32]
- Arrays [33-46]
- Functions [47-57]
- Loops & Condicionais [58-70]

# Variáveis em JS

0 que NÃO pode

[] Iniciar com *números*

[] Conter espaços

[] Utilizar palavras reservadas

# Variáveis em JS

```
var 1number = 25  
var my variable = "Leonardo"  
var var = "Redin"
```

# Variáveis em JS

Boas práticas e o que é permitido

[X] Utilizar *sempre* camelCase

[X] Iniciar com letras maiúsculas ou minúsculas

[X] Começar com o sinal de \$

# Variáveis em JS

```
var age = 25  
var name = 'Leonardo'  
var surName = 'Redin'  
var $pecial = 'Special'
```

# Strings

0 que são strings?



# Strings

## Tipos de dados primitivos

- `string`
- `number`
- `boolean`
- `null`
- `undefined`
- `symbol`

# Strings

```
var name = 'Leonardo'  
var name = 'Leonardo'
```

## Boas práticas

- escolher uma das abordagens e seguir durante todo o projeto
- respeitar o styleguide do time

## Erro

```
var name = 'Leonardo'
```

# Strings

Vamos entender 4 tópicos sobre strings

- Citações dentro de strings
- Caracteres especiais
- Propriedade chamada *length*
- Alguns exemplos de *métodos*

# Strings

## Citação

 Erro

```
var phrase = 'João disse: 'Eu estou doente!''
```

 Correto

```
var phrase = 'João disse: "Eu estou doente!"'  
var phrase = "João disse: 'Eu estou doente!'"
```

# Strings

Caso especial: \

Façam esse teste no console:

```
var phrase = "João disse: \'Eu estou doente!\'"
```

O caractere \ torna o próximo caractere em string

# Strings

Alguns usos:

```
var nullCharacter = '\0'  
var backslash = '\\'  
var newLine = '\n'  
var carriageReturn = '\r'  
var verticalTab = '\t'  
var tab = '\t'  
var backspace = '\b'  
var formFeed = '\f'
```

# Strings

## Propriedade length

Retorna o tamanho de determinada string. Todas as strings possuem esse método.

```
var name = 'Leonardo'  
var nameSize = name.length  
nameSize // 8
```

# Strings

Método: `toUpperCase()`

```
var greeting = 'Olá pessoal'  
var allUpper = greeting.toUpperCase()  
allUpper // OLÁ PESSOAL
```



# Operadores

- +
- -
- \*
- /
- %
- =
- !

# Operadores

Em JS temos 3 tipos principais de operadores:

- Unário
- Binário
- Ternário

# Operadores

## Binário

Necessitam de 2 operandos

```
2 + 3  
x * y  
10 % 2
```

# Operadores

## Unário

Necessita de apenas 1 operando

```
y++  
++z
```

# Operadores

## Atitméticos

```
var count = 10 + 5 - 4 // 11  
var mult = 9 * 3 // 27  
var div = 12 / 4 // 3  
var rest = 21 % 4 // 1
```

# Operadores

## Assignment operators

Já utilizamos esse bastante `x = 17`

```
var count = 5  
count = count + 12  
count += 12  
count -= 7
```

# Operadores

## Operadores de comparação

Retornam um valor booleano ao comparar valores

Valores podem ser *numéricos*, *string*, *logico* ou *objetos*

```
// Operador de comparação  
var x = 7  
var y = '7'  
x == y // true  
x === y // false
```

# Operadores

Incremento(++) e Decremento(--)

```
var x = 7  
// Qual a diferença entre :  
x++  
++x
```



# Operadores

`x++` retorna o valor *antes* de ser incrementado

`++x` retorna o valor *após* ser incrementado

# Objects

## O que são objetos?

Um tipo que consiste em pares de chave/valor. Tudo que não for um dado primitivo(string, number, boolean, null, undefined, symbol) é um objeto

# Objects

## Object Literal Notation {}

```
// pensem num pokemon  
var pokemon = {}  
// o que um pokemon tem ?
```

# Objects

## Charmander

[X] Nome

[X] Peso

[X] Altura

[X] Tipo

# Objects

```
var pokemon = {  
  name: 'Charmander',  
  weight: 8.5,  
  height: '61cm',  
  type: 'Fire',  
}
```

# Objects

## Dot Notation

```
var pokemon = {}  
  
// object.property = value  
  
pokemon.name = 'Charmander'  
pokemon.weight = 8.5  
pokemon.height = '61cm'  
pokemon.type = 'Fire'  
  
console.log(pokemon)
```

# Objects

## Bracket Notation

```
var pokemon = {}  
  
// object['property'] = value  
  
pokemon['name'] = 'Charmander'  
pokemon['weight'] = 8.5  
pokemon['height'] = '61cm'  
pokemon['type'] = 'Fire'  
  
console.log(pokemon)
```

# Objects

## Object Constructor

```
//e se fôssemos criar vários pokemons ?  
var pokemon1 = {  
  name: 'Charmander',  
  weight: 8.5,  
  height: '61cm',  
  type: 'Fire',  
}  
  
var pokemon2 = {  
  name: 'Bulbasaur',  
  weight: 6.9,  
  height: '71.1cm',  
  type: 'Grass & Poison',  
}
```



# Objects

```
function Pokemon(name, weight, height, type) {  
  this.name = name  
  this.weight = weight  
  this.height = height  
  this.type = type  
}  
  
var poke1 = Pokemon('Charmander', 8.5, '61cm', 'Fire')
```

# Arrays

Arrays são objetos com comportamentos diferentes

```
var pokemons = []  
  
pokemons = ['Charmander', 'Bulbasaur', 'Squirtle']  
  
console.log(pokemons)
```

# Arrays

```
// 'Charmander' é um elemento do array  
// Arrays possuem informações em ordem  
// Cada array possui um índice de acesso a cada elemento  
// Esses índices começam com a contagem a partir de 0
```

# Arrays

Arrays usam apenas *Dot Notation*

```
// para acessar algum elemento no array
// ARRAY[INDEX]
pokemons = ['Charmander', 'Bulbasaur', 'Squirtle']

console.log(pokemons[0])

// console.log(pokemons.0)
```

# Arrays

Arrays não são tipados

```
var randomArray = ['string', 7, true]
```

# Arrays

## `[] . MÉTODOS()`

- `POP()`
- `PUSH()`
- `SHIFT()`
- `UNSHIFT()`
- `CONCAT()`
- `REVERSE()`
- `SORT()`
- `SLICE()`

# Arrays

## POP()

```
var pokemons = ['Charmander', 'Bulbasaur', 'Squirtle']  
var lastPokemon = pokemons.pop()  
// último elemento removido  
pokemons // ['Charmander', 'Bulbasaur']  
  
console.log(pokemons)  
console.log(lastPokemon)
```

# Arrays

## PUSH()

```
var pokemons = ['Charmander', 'Bulbasaur']  
pokemons.push('Squirtle')  
// adiciona um elemento no final de um array  
  
console.log(pokemons)
```



# Arrays

## SHIFT()

```
var pokemons = ['Charmander', 'Bulbasaur']  
pokemons.shift()  
  
console.log(pokemons)
```

# Arrays

## UNSHIFT()

```
var pokemons = ['Bulbasaur']  
pokemons.unshift('Charmander', 'Squirtle')  
console.log(pokemons)
```

# Arrays

## CONCAT()

```
var pokemons = ['Charmander', 'Bulbasaur']  
var waterPokes = ['Squirtle']  
var allPokes = pokemons.concat(waterPokes)  
  
console.log(allPokes)
```

# Arrays

## REVERSE()

```
var pokemons = ['Charmander', 'Bulbasaur', 'Squirtle']  
var reversed = pokemons.reverse()  
  
console.log(reversed)
```

# Arrays

## `sort()`

```
var pokemons = ['Charmander', 'Bulbasaur', 'Squirtle']  
var alphabetic = pokemons.sort()  
  
console.log(alphabetic)
```

# Arrays

## SLICE()

```
// slice recebe 2 argumentos  
// 1 é o início  
// 2 é o final  
// se os argumentos são negativos estamos indo na ordem inversa  
var pokemons = ['Charmander', 'Bulbasaur', 'Squirtle']  
var sliced = pokemons.slice(0, 2)  
  
console.log(sliced)
```

# FUNCTIONS

## Declaração de funções

```
function favoritePokemon() {  
  return 'Meu Pokemon favorito é o Charmander'  
}  
  
console.log(favoritePokemon())
```

# Functions

Parâmetros(paramOne, paramTwo, ...)

```
function yourFavoritePokemon(pokemonName) {  
  return 'Seu Pokemon favorito é ' + pokemonName  
}  
  
console.log(yourFavoritePokemon('Bulbasaur'))
```



# Functions

- Function Expression
- Funções anônimas
- IIFE

# Functions

## Function Expression e Função Anônima

```
var favoritePokemon = function () {  
    return 'Meu Pokemon favorito é o Charmander'  
}  
  
console.log(favoritePokemon())
```

# Functions

Qual a diferença entre

```
function favoritePokemon() {  
    return 'Meu Pokemon favorito é o Charmander'  
}  
  
var favoritePokemon = function () {  
    return 'Meu Pokemon favorito é o Charmander'  
}
```

# Functions

## Hoist de variáveis

```
console.log(favoritePokemon())

function favoritePokemon() {
  return 'Meu Pokemon favorito é o Charmander'
}

/*
console.log(favoritePokemon())

var favoritePokemon = function () {
  return 'Meu Pokemon favorito é o Charmander'
}
*/
```

# Functions

## IIFE

Immediately Invoked Function Expressions

```
var greetFullName = (function (firstName, lastName) {  
    return 'Olá' + firstName + ' ' + lastName  
})();
```

# Functions

## Escopo

Javascript possui 2 escopos: global e local. Variáveis declaradas fora de uma função são globais. Variáveis declaradas dentro de uma função são locais

# Functions

## Escopo

```
var scope = 'public'

function checkScope() {
  var scope = 'private'
  return scope
}

console.log(scope)
```

# Functions

## Escopo

```
var scope = 'public'
var newScope = 'variável global'

function checkScope() {
  var scope = 'private'
  return [scope, newScope]
}

console.log(newScope)
console.log(checkScope())
```



# Functions

Como JS procura as variáveis por Escopo

- Child
- Parent
- Global

# Functions

## Problemas com escopo e VAR

```
var scope = 'public'

function checkScope() {
  scope = 'private'
  return scope
}

scope
// checkScope()
// scope
```

# Loops e Condicionais

Antes de vermos os laços e condicionais

0 que são valores *TRUE* e *FALSE*

# Loops e Condicionais

FALSE

- `false`
- `0`
- `null`
- `undefined`
- `NaN`

# Loops e Condicionais

TRUE

- Tudo o que não for falso

# Loops e Condicionais

## Operadores lógicos

- (AND) &&
- (OR) ||

# Loops e Condicionais

## Condicional IF

```
var favoritePokemon = 'Charmander'  
if (favoritePokemon === 'Charmander') {  
  console.log('Yay! Charmander é o máximo!')  
} else {  
  console.log('Tudo bem, outros pokemons são ótimos também')  
}
```

# Loops e Condicionais

## Condicionais em cadeia

```
if (score > 90) {  
  console.log('Sua nota final é A')  
} else if (score > 80) {  
  console.log('Sua nota final é B')  
} else if (score > 70) {  
  console.log('Sua nota final é C')  
} else {  
  console.log('Acho melhor estudar um pouco mais :)')  
}
```



# Loops e Condicionais

## Condiciona! SWITCH

```
switch (expression) {  
    case x:  
        // código  
        break  
    case y:  
        // return código  
    default:  
        // código  
}
```

# Loops e Condicionais

WHILE, DO/WHILE, FOR

# Loops e Condicionais

## WHILE

```
while (expression) {  
    // código aqui  
}
```

# Loops e Condicionais

## DO/WHILE

```
// vantagem é que aqui sempre vamos  
// executar o código pelo menos 1 vez  
do {  
    // código aqui  
} while (expression)
```

# Loops e Condicionais

## FOR

```
for (inicialização; condição; atualização) {  
    // código aqui  
}
```

# Loops e Condicionais

## FOR...IN

```
var object = {a: 1, b: 2, c: 3}

for (var property in object) {
  console.log(`${property}: ${object[property]}`)
}
```

# Loops e Condicionais

## FOR...OF

```
var arr = [1, 2, 3]

for (var value of array) {
  console.log(value)
}
```

Fim da Parte 1

Agora vamos para um JS moderno

ES6+