JavaScript

Um guia de introdução à linguagem

Rondonópolis - 2018

Alexandre Thebaldi
@ahlechandre

- **GitHub:**
  - **piano-x** *(https://github.com/ahlechandre/piano-x) ~ 2015*
  - **terere-theme** *(https://github.com/ahlechandre/terere-theme) ~ 2015*
  - **mdl-stepper** *(https://github.com/ahlechandre/mdl-stepper) ~ 2016*
  - **chart-handler** *(https://github.com/ahlechandre/chart-handler) ~ 2016*
  - **consl** *(https://github.com/ahlechandre/fp-loop) ~ 2016*
- **Medium:**
  - **Lambda Calculus com JavaScript:** Abordagem introdutória às raízes da programação funcional ~ 2017
  - **Os Méritos da Programação Funcional:** Uma análise sobre os paradigmas ~ 2017
- **StackOverflow:** *~117k people reached* (1,850)

Sumário

Capítulo I

Introdução

O que é JavaScript?

- **Interpretada;**
- **Orientada a objetos;**
- ***Prototype-based;***
- ***First-class functions;***
- **Dinâmica;**
- **Multiparadigma (orientada a objetos, imperativa e funcional);**
- ***Script* para Páginas Web e outros ambientes.**

JavaScript *versus* ECMAScript

JavaScript é padronizado internacionalmente

Principais ECMAScript *engines*:

- **SpiderMonkey -** A primeira. Por Brendan Eich para Netscape Navigator e utilizada no Firefox;
- **V8 -** *Open-source*. Por Google para Google Chrome;
- **JavaScriptCore (ou Nitro) -** *Open-source*. Por Apple para Safari;
- **Chakra (JScript9) -** Por Microsoft para Internet Explorer;
- **Chakra (JavaScript) -** Por Microsoft para Microsoft Edge.

Começando com JavaScript

Ferramentas necessárias

- **Editor de código**
- **Navegador Web || Node.js**
-
-
-
-
-
-
-

Qual a diferença entre o ambiente Node e o Browser?

Browser *versus* Node ~ Variáveis globais

```
// Browser
typeof window // object
typeof global // undefined

// Node
typeof window // undefined
typeof global // object
```

Browser *versus* Node ~ Documento

```
// Browser
typeof document // object

// Node
typeof document // undefined
```

Browser *versus* Node ~ Módulos

```
// Browser
typeof require // undefined

// Node
typeof require // function
```

Browser *versus* Node ~ Papéis

```
// Browser
typeof require('http').createServer //
ReferenceError...

// Node
typeof require('http').createServer //
function
```

Output

```
// Browser
alert('My message here')

// Browser + Node
console.log('My message here')
```

Exercício 1

- Crie um programa que imprima “*Hello World, this app is running on {Node || Browser}*”.

Capítulo II

Gramática e tipos

Considerações Iniciais

```
caseSensitive = 10;

console.log(CaseSensitive); // Uncaught
ReferenceError: CaseSensitive is not
defined
```

Considerações Iniciais

```
x = 10
y = 20
z = 30
xyz = x + y + z // 60
```

Considerações Iniciais

```
x = 10 y = 20 z = 30 xyz = x + y + z
// Uncaught SyntaxError: Unexpected
identifier
```

Considerações Iniciais

```
x = 10; y = 20; z = 30; xyz = x + y + z;
// 60
```

Comentários

```
// a one line comment
/* this is a longer,
 * multi-line comment
 */
/* You can't, however, /* nest comments
*/ SyntaxError */
```

Declarações ~ Tipos

```
var a = 1 // variável, escopo da função

let b = 2 // variável, escopo do bloco

const c = 3 // apenas leitura, escopo do
bloco
```

Declarações ~ Avaliando

```
var a
a // undefined
let b
b // undefined
const c // SyntaxError: Missing
initializer in const declaration
```

Declarações ~ Avaliando

```
let a

a === undefined // true
```

Exercício 2

- Inicialize três variáveis (*var*, *let* e *const*) condicionalmente.
- Em seguida, tente imprimir o valor de todas em contexto global.

Declarações ~ Escopo (var)

```
if (true) {
  var x = 5
}

x  // 5
```

Declarações ~ Escopo (let)

```
if (true) {
  let y = 5
}

y  // ReferenceError: y is not defined
```

Declarações ~ *Hoisting* (var)

```
x === undefined // ??

var x = 'wtf'

x // ??
```

Declarações ~ *Hoisting* (var)

```
x === undefined // true

var x = 'wtf'

x // "wtf"
```

Declarações ~ *Hoisting* (var)

```
var x // undefined

x === undefined // true

x = 'wtf'

x // "wtf"
```

Declarações ~ *Hoisting* (let)

```
x === undefined // ReferenceError: x is
not defined

let x = 'now it makes sense'
```

Declarações ~ Global

```
window // {...}
window.x // undefined

if (true) {
 x = 'this goes to global object'
}
window.x // "this goes to global object"
```

Tipos de dados

```
// 1. Boolean
// 2. Null
// 3. Undefined
// 4. Number
// 5. String
// 6. Symbol
// 7. Object
```

Tipos de dados ~ Conversão

```
let x = 1

x = 'now it is a string'
```

Tipos de dados ~ Conversão

```
let x = 'the answer is ' + 100 // "the
answer is 100"

let y = '100' + 10 // "10010"

let z = '100' - 10 // 90
```

Literais

```
// 1. Array literals
// 2. Boolean literals
// 3. Floating-point literals
// 4. Integers
// 5. Object literals
// 6. RegExp literals
// 7. String literals
```

Literais ~ Arrays

```
var list = [1, 2, 3, 4]

let list2 = ['a', 2, 'c', 4]

const list3 = ['a', 'b', 'c', 'd']
```

Literais ~ Objects

```
let myObject = {
  a: 10,
  b: 'something',
  c: 1.2
}
```

Literais ~ Objects

```
let rectangle = {
  length: 10,
  width: 20,
  area: function () {
    return this.length * this.width
  }
}
```

Literais ~ Strings

```
let first = 'one' // "one"

let second = first + ", two" // "one,
two"

let third = second + ', three' // "one,
two, three"
```

Literais ~ Strings (template)

```
// `string text`

// `string text line 1
//  string text line 2`

// `string text ${expression} string
text`
```

Exercício 3

- Inicialize duas variáveis apenas de leitura com valores inteiros.
- Em seguida, inicialize uma variável (string) com o template: “*{value} + {value2} = {result}*”

Capítulo III

Fluxo de controle e manipulação de erros

Block statement

```
// {
//    statement_1;
//    statement_2;
//    ...
//    statement_n;
// }
```

Block statement ~ Exemplo

```
const x = 10
{
  let y = 'abc'
  console.log(y, x)
}
```

Statements condicionais

```
// if (condition) {
//    statement_runs_if_is_true;
// } else {
//    statement_runs_if_is_false;
// }
```

Statements condicionais

```
const condition = true

if (condition) {
  console.log(`It's true`)
} else {
  console.log(`It's false`)
}
```

Statements condicionais

```
// switch (expression) {
//   case label_1:
//     statements_1
//     ...
//   default:
//     statements_def
// }
```

Statements condicionais

```
switch (color) {
  case 'red':
    console.log(`It's red!`)
    break
  default:
    console.log(`It's not red!`)
    break
}
```

Valores “falsos”

```
// false
// undefined
// null
// 0
// NaN
// ""
```

Manipulação de erros

```
// throw statement
// try...catch statement
```

Manipulação de erros ~ Exemplo

```
try {
  throw new Error('My custom message')
} catch (e) {
  console.log(e.message) // "My custom
message"
}
```

Exercício 4

- Lance um erro condicionalmente e capture-o para exibir sua mensagem.

Capítulo IV

Laços e iteração

for statement

```
// for ([initialExpression]; [condition];
[incrementExpression])
//   statement
```

for statement

```
for (let i = 0; i < 10; i++) {
  console.log(i) // 0, 1, 2, ..., 9
}
```

do...while statement

```
// do
//    statement
// while (condition);
```

do...while statement

```
let x = 0

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

while statement

```
// while (condition)
//   statement
```

while statement

```
let x = 0

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

for...in statement

```
// for (variable in object) {
//   statements
// }
```

for...in statement

```
const myObject = {x: 10, y: 20, z: 30}

for (let key in myObject) {
  console.log(key) // "x", "y", "z"
}
```

Exercício 5

- Inicialize uma variável contendo um objeto.
- Ande pelas propriedades do objeto, concatene suas *keys* em uma string separando-as por “-” e imprima o resultado.

for...of statement

```
// for (variable of array) {
//   statements
// }
```

for...of statement

```
const myArray = [10, 20, 30]

for (let value of myArray) {
  console.log(value) // 10, 20, 30
}
```

Exercício 6

- Inicialize uma lista com valores inteiros.
- Ande pelo array, some todos os valores e imprima o resultado.

Capítulo V

Funções

Declaração de função

```
function square(number) {
  return number * number
}
```

First-class functions

“The language supports passing functions as arguments to other functions, returning them as the values from other functions, and assigning them to variables or storing them in data structures” **~ Wikipedia**

Expressão de função

```
const square = function (number) {
  return number * number
}
```

Aplicando uma função

```
let result = square(10)
```

Exercício 7

- Crie uma expressão de função que imprima “*Hello World*” e aplique-a antes da sua linha de definição.
- Substitua a expressão de função por uma declaração de função.

Hoisting em funções

```
hoist() // "this is hoisted"

function hoist() {
  return 'this is hoisted'
}
```

Hoisting em funções

```
hoist() // TypeError: hoist is not a
function

var hoist = function () {
  return 'this is not hoisted'
}
```

Hoisting em funções

```
hoist() // ??

let hoist = function () {
  return 'this is not hoisted also'
}
```

Hoisting em funções

```
hoist() // ReferenceError: hoist is not
defined

let hoist = function () {
  return 'this is not hoisted also'
}
```

IIFE (*Immediately Invoked Function Expression*)

```
const invokeMe = function () {
  return 'Invoked!'
}

invokeMe // function
invokeMe() // "Invoked!"
```

IIFE (*Immediately Invoked Function Expression*)

```
const invokeMe = function () {
  return 'Invoked!'
}()

invokeMe // "Invoked"
invokeMe() // TypeError...
```

Exercício 8

- Crie uma expressão de função imediatamente invocada (IIFE) que retorne o ano de nascimento, dado uma idade.

Exercício 8

```
const birth = (function (age) {
  return 2018 - age
})(22)

birth // 1996
```

Escopo de função

```
const number = 10 // Escopo global.

let square = function () {
  // Acessa variáveis fora do seu escopo.
  return number * number
}
```

Exercício 9

- Crie uma função que acessa e altera o valor de uma variável externa ao seu escopo.

Closures

“A closure is an expression that can have free variables together with an environment that binds those variables” **~ MDN**

Closures

```
let square = function (number) {
  let result = function () {
    return number * number
  }

  return result()
}
```

Exercício 10

- Crie uma *closure* que retorna um argumento inteiro dobrado vindo da função parente.

Currying

“Is the technique of translating the evaluation of a function that takes multiple arguments into evaluating a sequence of functions, each with a single argument” **~ Wikipedia**

Currying

```
const sum = function (x, y, z) {
  return x + y + z
}

sum(1, 2, 3) // 6
```

Currying

```
const sum = function (x) {
  return function (y) {
    return function (z) {
      return x + y + z
    }
  }
}
```

Currying ~ Aplicação parcial

```
sum(1)(2)(3) // 6
// Ou
const first = sum(1) // function
const second = first(2) // function
const third = second(3) // 6
```

Exercício 11

- Crie uma função para calcular a área de um retângulo (*base * altura*) e aplique-a parcialmente.

Recursão

“A function that calls itself is called a recursive function. In some ways, recursion is analogous to a loop. Both execute the same code multiple times, and both require a condition” **~ MDN**

Loop (sem recursão)

```
let x = 0

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

Recursão

```
let loop = function (x) {

  if (x >= 10) return

  return loop(x + 1)
}
loop(0)
```

Recursão

```
let loop = function (x) {

  if (x >= 10) return

  return arguments.callee(x + 1)
}
loop(0)
```

Recursão

```
let loop = function internalName(x) {

  if (x >= 10) return

  return internalName(x + 1)
}
loop(0)
```

Exercício 12

- Crie uma função recursiva que decrementa um número inteiro até *-100*.

Parâmetros padrão em funções

```
const multiply = function (a, b) {
  b = b === undefined ? 1 : b

  return a * b
}
multiply(10) // 10
multiply(10, 2) // 20
```

Parâmetros padrão em funções

```
const multiply = function (a, b = 1) {
  return a * b
}

multiply(10) // 10
multiply(10, 2) // 20
```

Parâmetros REST em funções

```
const myFunc = function (...theArgs) {
  return theArgs
}

myFunc(1, 2, 3, 4) // [1, 2, 3, 4]
```

Exercício 13

- Crie uma função que receba um parâmetro padrão com valor “false“ e um número indefinido de argumentos como array e imprima todos os parâmetros.

Arrow functions

“Arrow functions are a more concise syntax for writing function expressions” ***~ SitePoint***

Arrow functions ~ Sintaxe

```
// (param1, ..., paramN) => { statements }

// (param1, ..., paramN) => expression
```

Function expression *versus* Arrow function

```
// Function expression
function (x) {
  return x
}

// Arrow function
x => x
```

Arrow functions ~ Sem parâmetros

```
const func = () => expression
```

Arrow functions ~ Um parâmetro

```
const func = x => x

const sameAsFunc = (x) => x
```

Arrow functions ~ Múltiplos parâmetros

```
const func = (x, y) => x + y
```

Arrow functions ~ *Concise body*

```
const func = x => x * 2

const sameAsfunc = x => (x * 2)
```

Arrow Functions ~ *Block body*

```
const func = (x, y) => {
  if (y > 0) {
    return x * z
  }
  return x
}
```

Exercício 14

- Escreva a seguinte expressão de função da forma mais concisa possível (prêmio para quem utilizar menos caracteres):

Exercício 14

```
function (x, y, z) {
  if (x === 0) {
    return y + z
  }
  return x + y + z
}
```

Exercício 15

- Escreva uma *arrow function* de corpo conciso que retorne um objeto.

Arrow functions ~ Retornando objetos

```
const func = () => {
  a: 10,
  b: 20,
  c: 30
}

func() // SyntaxError...
```

Arrow functions ~ Retornando objetos

```
const func = () => ({
  a: 10,
  b: 20,
  c: 30
})

func() // { a: 10, ...}
```

Higher-order functions

- Recebe uma ou mais funções como argumento
- Retorna uma função como resultado

Higher-order function ~ Exemplo

```
const applyMe = function (x) {
  return x()
}
applyMe(function () {
  return 'Higher order function!'
}) // 'Higher order function!'
```

Higher-order function ~ Exemplo

```
const applyMe = x => x()

applyMe(() => 'Higher order function!')
// 'Higher order function!'
```

Higher-order function ~ *setTimeout(callback, delay)*

```
setTimeout(function () {
  console.log('After 2 secs...')
}, 2000)
```

Exercício 16

- Escreva uma *arrow function* que imprime “*Hello World*” e agende-a para ser aplicada após 5 segundos.

Higher-order function ~ *setInterval(callback, delay)*

```
setInterval(function () {
  console.log('After every sec...')
}, 1000)
```

Exercício 17

- Escreva uma *arrow function* para ser aplicada a cada 2 segundos.
- A cada aplicação, a função deve mostrar a quantidade atual de aplicações.
- Após a quinta aplicação, a função deve parar de ser aplicada.

Capítulo VI

Arrays

Arrays

“Arrays are collections of data which are ordered by an index value” **~ MDN**

Criando um array ~ *Constructor*

```
let list = new Array(1, 2, 3, 4, 5)

list // [1, 2, 3, 4, 5]
```

Criando um array ~ Função

```
let list = Array(1, 2, 3, 4, 5)

list // [1, 2, 3, 4, 5]
```

Criando um array ~ Literal

```
let list = [1, 2, 3, 4, 5]

list // [1, 2, 3, 4, 5]
```

Tamanho do array

```
let list = [10, 20, 30]

list.length // 3
```

Populando um array

```
let list = [10, 20, 30]
// Ou
let list = []
list[0] = 10
list[1] = 20
list[2] = 30
```

Iterando sobre array

```
let colors = ['red', 'blue', 'green']

for (let i = 0; i < colors.length; i++) {
  console.log(colors[i])
}
```

Exercício 18

- Escreva uma função que receba uma lista de valores inteiros e retorne uma nova lista com os valores dobrados.

Exercício 19

- Escreva uma função que receba uma lista de valores inteiros e retorne uma nova lista com os valores pares.

Exercício 20

- Escreva uma função que receba uma lista de valores inteiros e retorne a somatória de todos eles.

Métodos de array ~ *concat()*

```
let list = ['a', 'b', 'c']

let newList = list.concat('d', 'e', 'f')

newList // ['a', 'b', 'c', 'd', 'e', 'f']
```

Métodos de array ~ *join(delimiter = ',')*

```
let list = ['a', 'b', 'c']

let newList = list.join('-')

newList // 'a-b-c'
```

Métodos de array ~ *push()*

```
let list = ['a', 'b', 'c']

const newLength = list.push('d') // 4

list // ['a', 'b', 'c', 'd']
```

Métodos de array ~ *pop()*

```
let list = ['a', 'b', 'c']

const lastItem = list.pop() // 'c'

list // ['a', 'b']
```

Métodos de array ~ *shift()*

```
let list = ['a', 'b', 'c']

const firstItem = list.shift() // 'a'

list // ['b', 'c']
```

Métodos de array ~ *unshift()*

```
let list = ['a', 'b', 'c']

const newLength = list.unshift('w', 'y',
'z') // 6

list // ['w', 'y', 'z', 'a', b', 'c']
```

Métodos de array ~ *reverse()*

```
let list = ['a', 'b', 'c']

list.reverse()

list // ['c', 'b', 'a']
```

Métodos de array ~ *sort()*

```
let list = [5, 3, 6, 1, 4, 2]

list.sort()

list // [1, 2, 3, 4, 5, 6]
```

MDN web docs / Indexed collections

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Indexed_collections

Higher-order function ~ *map(callback)*

```
let list = [1, 2, 3, 4]

let newList = list.map(function (value) {
  return value + 1
})

newList // [2, 3, 4, 5]
```

Higher-order function ~ *map(callback)*

```
let list = [1, 2, 3, 4]

let newList = list.map(value => value +
1)

newList // [2, 3, 4, 5]
```

Higher-order function ~ *map(callback)*

```
const plusOne = value => value + 1

let list = [1, 2, 3, 4]

let newList = list.map(plusOne)

newList // [2, 3, 4, 5]
```

Exercício 21

- Escreva uma função que receba uma lista de valores inteiros e retorne uma nova lista com os valores dobrados usando “map”.

Higher-order function ~ *filter(callback)*

```
const greaterThan2 = value => value > 2

let list = [1, 2, 3, 4]

let newList = list.filter(greaterThan2)

newList // [3, 4]
```

Exercício 22

- Uma função que receba uma lista de valores inteiros e retorne uma nova lista com os valores pares usando “filter”.

Higher-order function ~ *reduce(callback, initialValue)*

```
let list = [1, 3, 4, 2]

let greater = list.reduce((prev, current)
=> {
 return prev > current ? prev : current
}) // 4
```

Exercício 23

- Uma função que receba uma lista de valores inteiros e retorne a somatória de todos eles usando “reduce”.

Capítulo VII

Objetos

Objetos

“An object is a collection of properties, and a property is an association between a name (or key) and a value” **~ MDN**

Acessando propriedades do objeto

```
myObject.myProperty

// Ou

myObject['myProperty']
```

Acessando propriedades não atribuídas do objeto

```
myObject.myProperty // undefined
```

Criando um objeto ~ Literal

```
let myCar = {
  make: 'Ford',
  model: 'Mustang',
  year: 1969
}
```

Criando um objeto ~ Literal

```
let myCar = {}

myCar.make = 'Ford'
myCar.model = 'Mustang'
myCar.year = 1969
```

Criando um objeto ~ Literal

```
let myCar = {}

myCar.make = 'Ford'
myCar['model'] = 'Mustang'
myCar.year = 1969
```

Criando um objeto ~ *Constructor*

```
let myCar = new Object()

myCar.make = 'Ford'
myCar.model = 'Mustang'
myCar.year = 1969
```

Criando um objeto ~ Função construtora

```
function Car(make, model, year) {
  this.make = make
  this.model = model
  this.year = year
}
```

Criando um objeto ~ Função construtora

```
let myFord = new Car('Ford', 'Mustang', 1969)

let myHonda = new Car('Honda', 'Civic', 2005)

let myChevrolet = new Car('Chevrolet', 'Onix',
2018)
```

Exercício 24

- Criar duas funções construtoras de objetos: uma para representar **pessoas** e outra para **carros**.
- Todo **carro** deve ter um atributo “proprietário”, que receberá uma instância de **pessoas**.
- Em **carro**, escreva um método para retornar o nome do proprietário.