UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Fundamentos de JavaScript

QXD0020 - Desenvolvimento de Software para Web

Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Agenda

- Introdução
- Visão geral
- Objetos e Funções

Introdução

Introdução

O que é JavaScript?

- Uma linguagem de scripts e interpretada criada em em meados da década de 90 pela Netscape Communications
- Multiparadigma, da suporte a programação funcional e imperativa
- Possui tipagem dinâmica
- Uma das três principais tecnologias da World Wide Web
- Desde 2013 é a linguagem mais popular de acordo com o StackOverflow Survey

Introdução

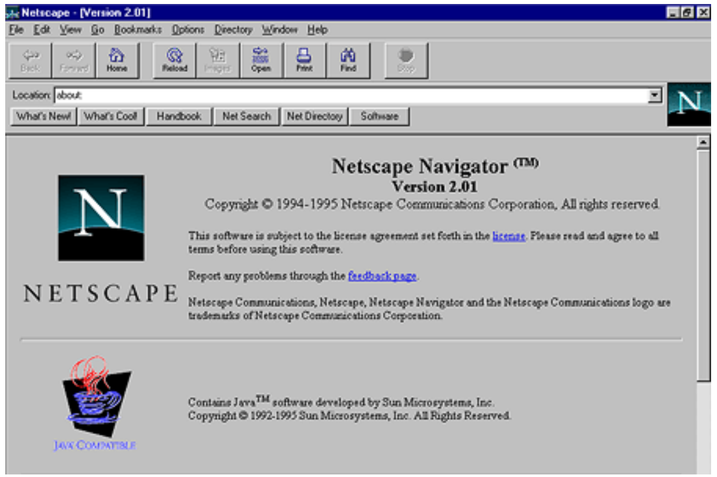
Javascript

- Inicialmente foi criada atender à demanda crescente por sites mais interativos e dinâmicos “front-end”
  - Inserir texto dinamicamente no código HTML
  - Reagir a eventos (ex: carregamento da página, cliques do usuário)
  - Pegar informações sobre o computador do usuário(ex: navegador)
  - Realizar cálculos no computador do usuário (Ex: validação de formulário)
- Atualmente pode ser utilizado no lado do servidor “back-end”
  - Ganhou popularidade em 2019 graças ao NodeJs
- Também pode ser utilizado para o desenvolvimento de aplicativos móveis

Introdução

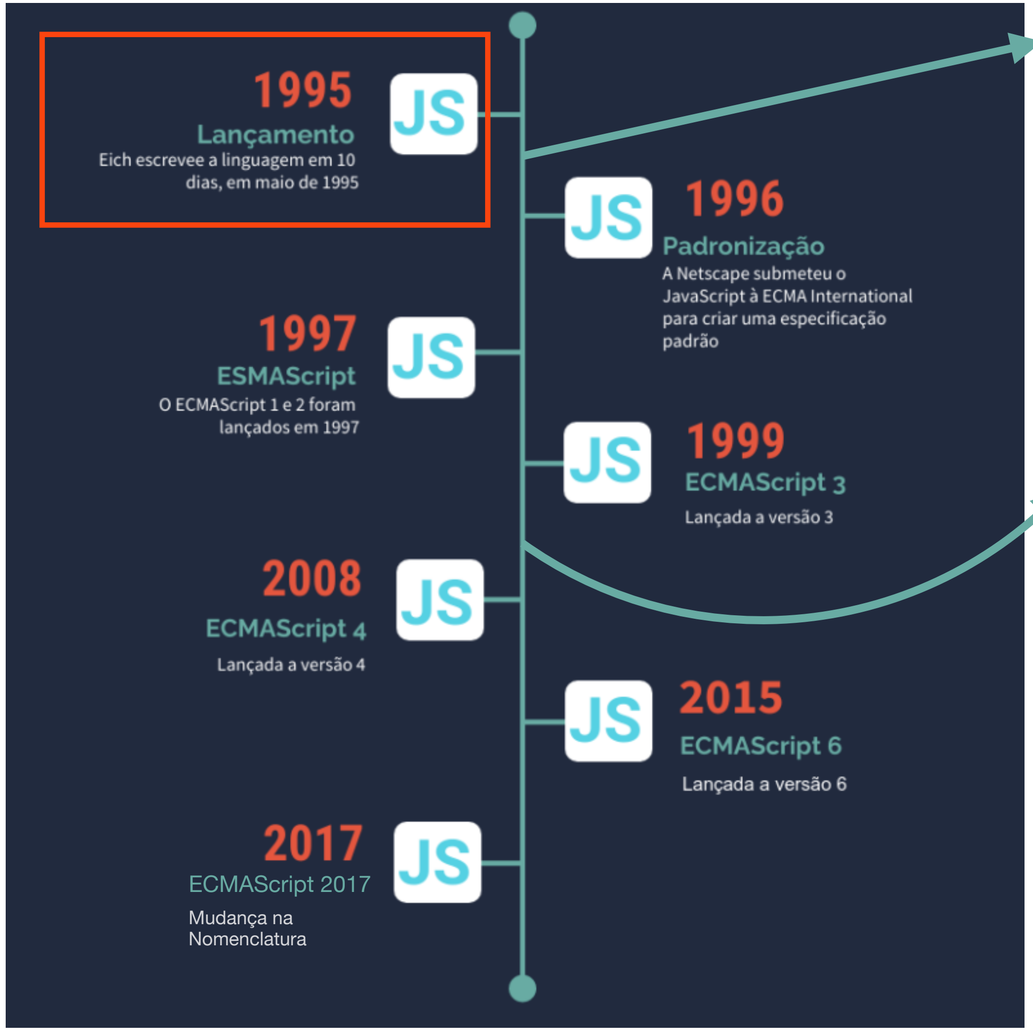
Origem

- Criada por Brendan Eich em 1995 para a Netscape
  - Originalmente chamado LiveScript
  - Mudou de nome por decisão de *marketing* apoiada na popularidade da linguagem Java
  - Foi lançada no Netscape Navigator 2.0 beta 3
- Originalmente implementada como parte dos navegadores
  - Permitir a execução de scripts do lado do cliente
  - Interação com o usuário sem a intervenção do servidor



Introdução

Evolução



- Implementado como JScript pela Microsoft por meio de eng. reversa
  - Internet Explorer 3
  - Tornou difícil o funcionamento de um site em diferentes navegadores
- ECMA Script 4 começa a ser a desenvolvido em 2000
  - Apesar de implementar parte da especificação, a Microsoft não tinha intenção de cooperar
  - Lançado em 2008 após o trabalho conjunto entre a Mozilla, Macromedia (ActionScript) e Brendan Eich
- ECMA Script 5 lançado em 2009 selou a paz entre as empresas
- Passou a contar com uma versão anual

Visão geral

Visão geral

Tipos de dados

- Number
- Null
- Undefined
- String
- Boolean
- Object
- BigInt
- Symbol

Visão geral

O tipo number

- Não há separação entre números inteiros e números reais
  - Ambos são do mesmo tipo, number
- Operadores aritméticos:
  - `+`, `-`, `*`, `/`, `%`, `**`, `++`, `--`, `+=`, `-=`, `*=`, `/=`, `%=`, `**=`
  - Mesma precedência do Java
  - Cuidado: Muitos operadores realização conversão automática de tipos

Visão geral

Declarando variáveis

- Variáveis podem ser declaradas usando três palavras chaves
  - var, let (ECMA6) e const
- O tipo da variável não é especificado
- As função typeof retorna o tipo de dados de objeto

Visão geral

var vs let vs const

Declarações	Escopo	Pode ser alterada
var	Global (hoisting) / Função	Sim
let	Bloco	Sim
const	Bloco	Não

Visão geral

var vs let vs const

```
console.log(mensagem); // undefined
var mensagem = "declaracao";
var mensagem = “redeclaracao”;
console.log(mensagem); // redeclaracao

mensagem2 = 'Msg';
console.log(mensagem); // Msg
var mensagem2;

var saudacao = "oi";
var n = 4;
if (n > 3) {
  var saudacao = "ola;
}
console.log(saudacao); // "oka"
```

Usando var é possível redeclarar uma variável

Hoisting / Içamento

Escopo global
Hoisting + Redeclaraçao

Visão geral

var vs let vs const

```
function exibirMensagem() {
  var msmForaDoIf = 'Msg 1';
  if(true) {
    var msgDentroDoIf = 'Msg 2';
    console.log(msgDentroDoIf);// Msg 2
  }
  console.log(msgForaDoIf); // Msg 1
  console.log(msgDentroDoIf); // Ms2
}
```

```
function exibirMensagem() {
  var msmForaDoIf = 'Msg 1';
  if(true) {
    var msgDentroDoIf = 'Msg 2';
    let letMsg = 'let';
    console.log(msgDentroDoIf);// Msg 2
  }
  console.log(msgForaDoIf); // Msg 1
  console.log(letMsg); // Erro
}
```

Visão geral

null e Undefined

- null
  - Existe, mas foi atribuído com vazio
  - Deliberadamente sem valor
- undefined
  - Variáveis declaras, mas não inicializadas
  - Membros objeto/Array que não existem

Visão geral

O tipo String

- Principais métodos
  - charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring toLowerCase, toUpperCase
- + é o operador de concatenação

Visão geral

O tipo String e os seus principais métodos

Método / Propriedade	Descrição
length	Propriedade que contém o tamanho da string
concat( )	Concatena um ou mais strigs
indexOf( )	Retorna a primeira ocorrência de um caractere na string
lastIndexOf( )	Retorna a última ocorrência de um caractere na string
match( )	Verifica a ocorrência de uma expressão regular na string
replace( )	Substitui alguns caracteres na string
slice( )	Extrai em uma nova string, parte da string original
split( )	Quebra a string em um array de strings
toLowerCase( )	Mostra a string em letras minúsculas
toUpperCase( )	Mostra a string em letras maiúsculas

Visão geral

Conversão entre String e Number

- Convertendo String em números

```
let count = 10;
let s1 = "" + count; // "10"
let s2 = count + " bananas, ah ah!"; // "10 bananas, ah ah!"
let n1 = parseInt("42 is the answer"); // 42
let n2 = parseFloat("booyah"); // NaN
```

- Acessando caracteres

```
let firstLetter = s[0]; // fails in IE
let firstLetter = s.charAt(0);// does work in IE
let lastLetter = s.charAt(s.length - 1);
```

Visão geral

Comentários

- Idêntico ao do C

```
// comentário de uma linha
/* comentário de
   Múltiplas
   linhas       */
```

Visão geral

Estruturas de controle

- if / else
  - Idêntico ao java
  - Praticamente qualquer coisa pode ser usada como condição
- Operador ternário

Visão geral

O tipo booleano

- Qualquer valor pode ser usado como Boolean
- Valores considerados false:
    - 0, 0.0, NaN, "", null, e undefined
- Valores para verdadeiro
    - Todos o resto
- Convertendo um valor para boolean explicitamente

Visão geral

O tipo booleano

```
let boolValue = Boolean(outroValor); // Convertendo para Boolean
let iLike190M = true;
let ieIsGood = "IE6" > 0;
if ("web dev is great") { /* true */
  if (0) { /* false */ }
}
```

Visão geral

Operadores relacionais

Operadores	Descrição
>	Maior que
>=	Maior que ou igual a
<	Menor que
<=	Menor que ou igual a
==	Igualdade
!=	Diferente
===	Igualdade sem coerção
!==	Igualdade com coerção

A maioria dos operados convertem os tipos automaticamente

Visão geral

Operadores relacionais

- A maioria dos operadores convertem os tipos automaticamente
- === e ! == não realizam a conversão de tipo

```
5 < “7” // true
42 == 42.0  // true
”5.0”== 5 // true
”5.0”=== 5 // false
```

Visão geral

Operadores lógicos

Operadores	Descrição
&&	E
||	Ou
!	Negação

Visão geral

Avaliação Curto Circuito Lógico

- && e || só executam o segundo operando, dependendo do resultado do primeiro
- Útil para checagem de objetos antes de acessar seus atributos
- Atribuição de valor default

```
let name = o && o.getName();
let name = otherName || "default”;
```

Visão geral

Estruturas de controle

- Switch / case
  - Comparação usa o operador ===

Visão geral

Vetores

- Existem duas maneira de inicializar um vetor
- Vetores funcionam como estrutura de dados e aumentam de acordo com a necessidade
  - Pilhas
    - push e pop, adicionam e removem respectivamente
  - Fila
    - unshift e shift, adicionam e e removem respectivamente
- Principais métodos:
  - concat, pop, push, reverse, shift, unshift, slice, sort, splice, toString

Visão geral

Vetores

- Existem duas maneira de inicializar um vetor
- O tamanho do vetor aumenta de acordo com a a necessidade

```
let name = new Array();
let name = []; //empty array
let name = [value, value, ..., value]; //pre filled array
name[index] = value; //stored element

let ducks = ["Huey", "Dewey", "Louie"];

let stooges = []; // stooges.length e 0
stooges[0] = "Larry"; // stooges.length e 1
stooges[1] = "Moe"; // stooges.length e 2
stooges[4] = "Curly"; // stooges.length e 5
stooges[4] = "Shemp"; //stooges.length e 5
```

Visão geral

Vetores

- Principais métodos:
  - concat, pop, push, reverse, shift, unshift, slice, sort, splice, toString
- Vetores funcionam como estrutura de dados
  - Pilhas: push e pop, adicionam e removem respectivamente
  - Fila: unshift e shift, adicionam e e removem respectivamente

```
let a = ["Stef", "Jason"]; //Stef, Jason
a.push("Brian"); //Stef, Jason, Brian
a.unshift("Kelly"); //Kelly, Stef, Jason, Brian
a.pop(); //Kelly, Stef, Jason
a.shift(); //Stef, Jason
a.sort(); // Jason, Stef
```

Visão geral

Vetores

- split
  - Quebra a string em partes utilizando um delimitador
  - Pode ser utilizado com expressões regulares
- join

  Transforma um vetor em uma string, utilizando um delimitador entre os elementos

```
let s = "the quick brown fox";
let a = s.split(" "); // ["the", "quick", "brown", "fox"]
a.reverse(); // ["fox", "brown", "quick", "the"]
s = a.join("!"); // "fox!brown!quick!the"
```

Visão geral

Estruturas de Repetição

- while
- do / while
- for
- for/in e for/of (ECMA 2015)
- break / continue

Visão geral

Estruturas de Repetição - while

- A condição é testada antes de iniciar a execução do bloco

```
while (condition) {
    code block to be executed
}

while (i < 10) {
    text += "The number is " + i;
    i++;
}
```

Visão geral

Estruturas de Repetição – do/while

- A condição é testada após a execução do bloco
  - O laço é executado pelo menos uma vez

```
do {
    code block to be executed
}
while (condition);

do {
    text += "The number is " + i;
    i++;
}
while (i < 10);
```

Visão geral

Estruturas de Repetição – for

- Instrução 1 - Executada antes de iniciar o bloco
- Instrução 2 - Executadas antes de cada iteração do laço
- Instrução 3 - Executadas após a iteração do laço

```
for (instrução 1; instrução 2; instrução 3) {
    code block to be executed
}

for (i = 0; i < 10; i++) {
    if (i === 3) { break }
    text += "The number is " + i + "<br>";
}
```

Visão geral
Exceções

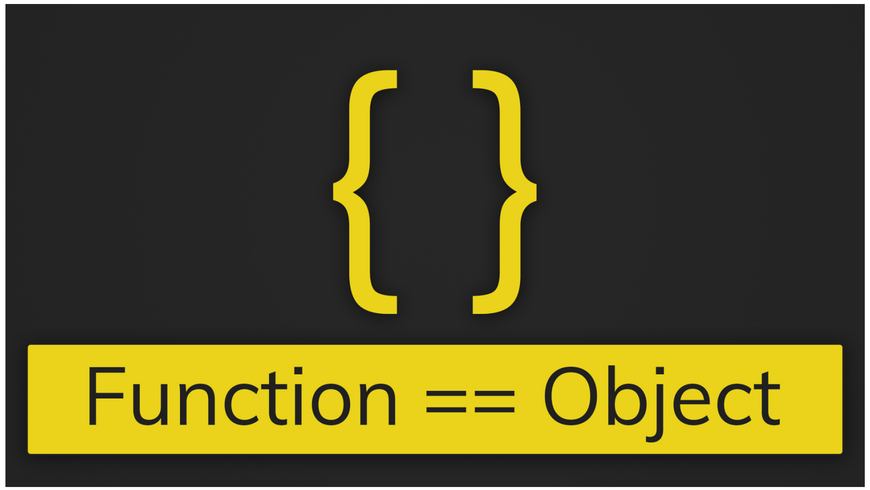
- try
  - Obrigatória . Usada para delimitar o bloco de código que pode gerar a exceção
- catch
  - O bloco interno é executado caso a exceção ocorra
  - A cláusula interrompe a propagação do erro
  - É possível acessar a exceção lançada
- finally
  - É opcional. O seu bloco de código é SEMPRE executado
  - Independentemente da exceção ser lançada
  - Mesmo que o bloco try possua um return

Visão geral

Exceções

```
try {
  //  Block of code to try
}
catch(err) {
  //  Block of code to handle errors
}
finally {
  // Block of code to be executed regardless of the try / catch result
}
```

Objetos e Funções



Objetos e Funções

Funções em JS

- Em JavaScript funções são objetos, logo possuem propriedades e métodos
  - Métodos: apply( ) e call( )
  - Propriedades: length e constructor
- Funções são first-class citizen. Podem ser:
  - usada como um valor qualquer
  - armazenadas em vetores, variáveis e objetos
  - passadas como argumentos para outras funções
  - retornadas por outras funções

Objetos e Funções

Exemplo

```
function exemplo(a, b){
    return a * b;
}

exemplo.length // 2
exemplo.constructor // Function()

const square = function(number) { return number * number }
let x = square(4) // x -> 16
```

Declaração

Notação literal - Function expression
c/ função anônima

Objetos e Funções

Passagem de parâmetros

- É possível passar qualquer quantidade de parâmetro para qualquer função
  - Não resulta em erro
- Os parâmetros são armazenados em uma estrutura similar a um vetor chamada de arguments
- A propriedade length de uma função armazena a quantidade de parâmetros da função

Objetos e Funções

O objeto argumento

```
let x = soma(1, 123, 500, 115, 44, 88);

function soma() {
  let i, soma = 0;
  for (i = 0; i < arguments.length; i++) {
    soma += arguments[i];
  }
  return soma;
}
```

Permite acessar todos os argumentos

Objetos e Funções

Rest parameter - ECMA 6

```
let x = soma(1, 123, 500, 115, 44, 88);

function soma(...args) {
  let soma = 0;
  for (let arg of args) soma += arg;
  return soma;
}

let x = soma(4, 9, 16, 25, 29, 100, 66, 77);
```

Armazenas todos os argumentos em um vetor

Objetos e Funções

Valor default - ECMA 6

```
function soma(x, y = 10) {
  return x + y;
}
soma(5); //15
```

Valor atribuído ao argumento caso nenhum valor seja passado

Objetos e Funções

Auto-Invocação

- São outra aplicação para funções anônimas
- São chamadas imediatamente após a sua declaração
- Pode ser utilizada para realizar alguma tarefa sem criar variáveis globais
- Normalmente utilizada para tarefas de inicialização

```
(function () {
    let x = "Oi e Adeus";        // So e executado uma vez
})();
```

Objetos e Funções

Callback

```
var a = [
  'Hydrogen','Helium','Lithium','Beryllium'
];
var a2 = a.map(function(s) { return s.length; });
console.log(a2); // logs [8, 6, 7, 9]
```

Objetos e Funções

Arrow functions - ECMA 6

```
var a = [
  'Hydrogen','Helium','Lithium','Beryllium'
];
var a3 = a.map(s => s.length);
console.log(a3); // logs [8, 6, 7, 9]

function (a, b){
  return a + b + 100;
}

(a + b) =>  {
  return a + b;
}

let soma = (a + b) => a + b;
```

Arrow function com um único argumento

Arrow function com dois argumentos

Arrow function com uma única instrução

Objetos e Funções

Funções Internas

```
function somaQuadrados(x, y){

  function elevaAoQuadrado(x){
    return x ** 2;
  }

  return elevaAoQuadrado(x) + elevaAoQuadrado(y);
}
```

- Funções internas tem acesso as variáveis do seu escopo e do seus parentes

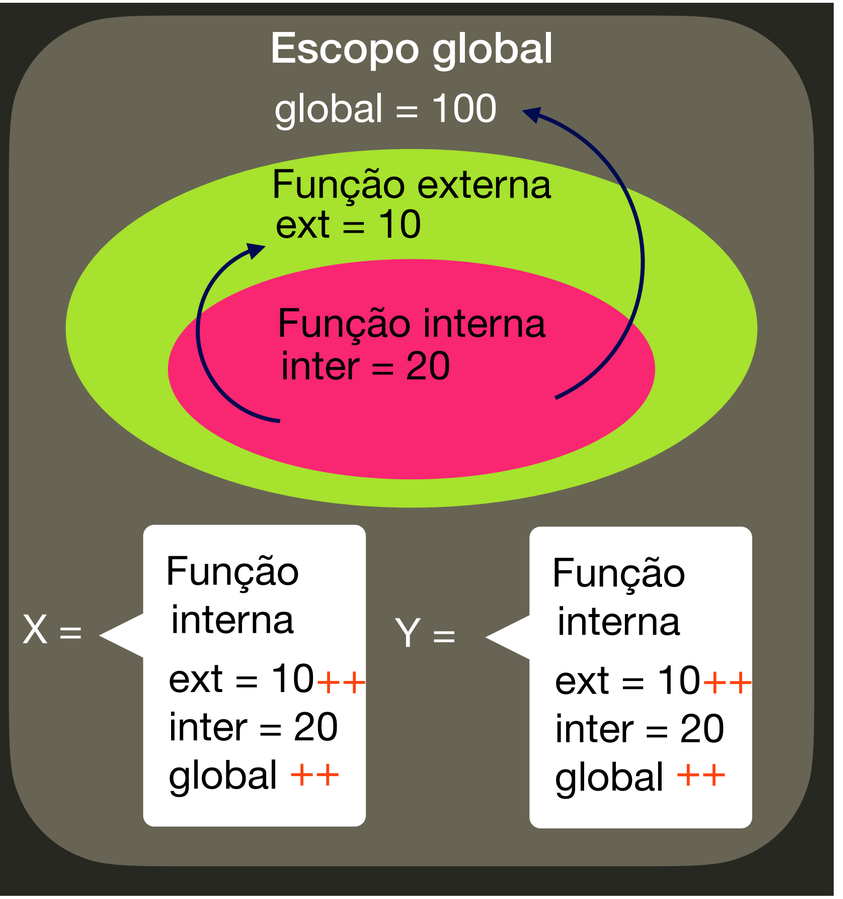
Objetos e Funções

Clousures

```
var global = 100;

function externa() {
  var ext = 10;
  global++;
  function interna() {
     var inter = 20;
     console.log(`inter=${inter}, ext=${ext}, global=${global}`);
     ext++;
     global++;
  }
  return interna;
}
var X = externa();
var Y = externa();

X();  //inter=20, ext=10, global=102
X();  //inter=20, ext=11, global=103
X();  //inter=20, ext=12, global=104
Y();  //inter=20, ext=10, global=105
```



Objetos e Funções

- Simples pares nome-valor, como
  - Dicionários em Python
  - Hashes em Perl e Ruby
  - Hash tables em C e C++
  - HashMaps em Java
  - Arrays associativos em PHP
- Muito comuns, estrutura de dados versátil
- Nome é uma string e o valor pode ser qualquer coisa

Objetos e Funções

Sintaxe Literal de Objetos

```
let pikachu = {
  nome: "Pikachu",
  especie: "Pikachu",
  nivel: 1,
  falar: function () {
    return `${this.nome} ${this.nome}`;
  }
};
```

Objetos e Funções

Criação de Objetos

- É possível criar objetos de duas formas
- Após a criação é possível adicionar métodos e atributos
- É possível acessar os atributos usando duas notações
  - Dot notation
  - Bracket notation
- É possível percorrer um objeto
- Podemos ainda usar funções construtoras

Objetos e Funções

Criação de Objetos

- É possível criar objetos de duas formas
- Após a criação é possível adicionar métodos e atributos

```
let pikachu = {
  nome: "Pikachu",
  especie: "Pikachu",
  nivel: 1,
  falar: function () {
    return `${this.nome} ${this.nome}`;
  }
}
```

```
let charmander = new Object();
charmander.nome = "Charmander";
charmander.falar = () => "chaarrr";
```

Objetos e Funções

Acessando atributos

- É possível acessar os atributos usando duas notações
  - Dot notation
  - Bracket notation

```
let pikachu = {
  nome: "Pikachu",
  …
}
let charmander = new Object();
charmander.nome = “Charmander”;

console.log(pikachu.nome);
console.log(charmander['nome']);
```

Objetos e Funções

Percorrendo um objeto

```
let pikachu = {
  nome: "Pikachu",
  nivel: 1,
}

for ( let attr in pikachu) {
  console.log(`${attr} = ${pikachu[attr]}`);
}

for ( let [attr, value] of Object.entries(pikachu)) {
  console.log(`${attr} = ${value}`);
}

for ( let value of Object.values(pikachu)) {
  console.log(`${value}`);
}
```

Adicionados no ECMA 6

Objetos e Funções

Funções construtoras

```
function Pokemon(nome, especie, nivel=1) {
  this.nome = nome;
  this.especie = especie;
  this.nivel = nivel;
  this.falar = () => `${this.nome} ${this.nome}`;
}

let pikachu = new Pokemon("Pikachu", "Pikachu");
let charmander = new Pokemon("Charmander", “Charmander”, nivel=1)
```

Objetos e Funções

Classes - ECMA 6

- Nova sintaxe adicionada no ECMA 6
- JavaScript continua sendo baseada em prototype
- Facilita a implementação de herança
- Syntatic Sugar -> Parecido com outra linguagens

Objetos e Funções

Classes - ECMA 6

```
class Pokemon {
  constructor(nome, especie, nivel=1) {
    this.nome = nome;
    this.especie = especie;
    this.nivel = nivel;
  }
  falar = () => `${this.nome} ${this.nome}` ;

  get nivel() { return this._nivel}
  set nivel(valor) { this._nivel = valor > 0 ? valor : 1}
}
let pikachu = new Pokemon("Pikachu", "Pikachu", -1);
console.log(`${pikachu.falar()} ${pikachu.nivel}`); // pikachu pikachu 1
```

Referências

- https://blog.betrybe.com/javascript/
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript
- https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript
- https://pt.wikipedia.org/wiki/JavaScript
- https://javascriptrefined.io/nan-and-typeof-36cd6e2a4e43
- https://www.toptal.com/javascript/es6-class-chaos-keeps-js-developer-up

Por hoje é só