**Aulas javascript**

É uma linguagem orientada a objetos baseada em protótipos, não possui classes.

Interpretativa (apenas necessário um browser).

**String** 🡪 tanto aspas simples como dupla.

**New** 🡪 função construtora.

**Funções são objetos.**

**Funções** 🡪 Só podem ser acessadas por elas mesmo ex: variáveis que podem se tornar funções.

**Function Declaration** 🡪 function soma(..){...}, cria uma variável como se fosse soma, é carregada antes da interpretação.

**Function expression** 🡪 var soma = function(...){...} aqui é atribuído a uma variável (função anonima), não é carregada antes da interpretação, para isso não ocorrer seguir ordem correta, pró questão de legibilidade.

**Nomed function expression** 🡪 var soma = function soma(...){...}

**Arrow function** 🡪 Para não sobrescrever e mudar o contexto no this (cada function cria um contexto), arrow function não tem o this próprio.

Const func = (parametro) => {

}

Const func valor => valor \*2;

**LAMBDA** 🡪 Passar uma função como argumento (passar funções como parâmetros).

**Objeto** 🡪 coleção dinâmica de chaves e valores, mapa de chaves e valores

**Simular métodos**

var teste = {

valor1 : 1,

valor2 : 2,

soma : function(){ //objeto soma

return this.valor1 + this.valor2

}

}

teste.soma(); //3

**this** Se refere ao escopo do qual estamos invocando.

**Call 🡪** Passa o objeto e o argumento do parâmetro

**Apply 🡪** Passa o objeto e o argumento do parâmetro como array [1]

**Função construtora vs Função fabrica**

**Função construtora 🡪** Começa com maiúsculo.

var Pessoa = function (idade, nome, funcao){

this.nome = nome;

this.idade = idade;

this.funcao = funcao;

};

console.log(new Pessoa(25, "Henrique", "garoto"));

**Função fabrica 🡪**

var pessoa = function (idade, nome, funcao){

return {

nome: nome,

idade: idade,

funcao: funcao

}

};

console.log(pessoa(25, "Henrique", "garoto"));

**Função** 🡪 Bloco de código executável.

**Closure**

Um closure (fechamento) é uma função que se "lembra" do ambiente — ou escopo léxico — em que ela foi criada.

function init() {

var name = "Mozilla";

function displayName() {

alert(name);

}

displayName();

}

init();

**IIFE 🡪** Expressão invocada automaticamente. }();

**Arrays 🡪 São objetos especiais**

carro = [];

carro = new array()

carro = [“”, “, “”] ;

New array (10), porem, não tem tamanho fixo, apenas tamanho inicial.

**Push** 🡪 adiciona último elemento na última posição, **pop** 🡪 remove último elemento.

Splice(pos, 1, “teste”);

Splice(pos, 0, “teste”); // não remove

**Filter**

carros = [];

carros[0] = {marca: "Ford", modelo: "Ka"};

carros[1] = {marca: "Chevrolet", modelo: "Corsa"};

carros[2] = {marca: "Fiat", modelo: "Palio"};

var carrosFord = carros.filter(function(elemento){

return elemento.marca === "Ford";

});

**Every e Some**

carros = [];

carros[0] = {marca: "Ford", modelo: "Ka"};

carros[1] = {marca: "Chevrolet", modelo: "Corsa"};

carros[2] = {marca: "Fiat", modelo: "Palio"};

var carrosFord = carros.every(function(elemento){

return elemento.marca === "Ford";

});

//false

var carrosFord = carros.some(function(elemento){

return elemento.marca === "Ford";

});

//true

**Map**

carros = [];

carros[0] = {marca: "Ford", modelo: "Ka"};

carros[1] = {marca: "Chevrolet", modelo: "Corsa"};

carros[2] = {marca: "Fiat", modelo: "Palio"};

var carrosFord = carros.map(function(elemento){

return elemento.marca;

});

**Reduce**

carros = [];

carros[0] = {marca: "Ford", modelo: "Ka", preco: 3000};

carros[1] = {marca: "Chevrolet", modelo: "Corsa", preco: 2000};

carros[2] = {marca: "Fiat", modelo: "Palio", preco: 1000};

var carrosValorTotal = carros.reduce(function(prev, cur){

return prev + cur.preco;

}, 0);

**Concat**

var carros = ["Ford", "Corsa", "Palio"];

var motos = ["Honda", "Yamaha"];

var veiculos = carros.concat(motos);

veiculos.toString();

**Slice**

**Fatiando um Array**

carros.slice(0,2); //[‘Ka’, ‘Corsa’]

**Reverse**

Inverte a ordem original do array

**Sort**

**Ordenando por preço**

carros = [];

carros[0] = {marca: "Ford", modelo: "Ka", preco: 1500};

carros[1] = {marca: "Chevrolet", modelo: "Corsa", preco: 3000};

carros[2] = {marca: "Fiat", modelo: "Palio", preco: 1350};

carros.sort(function(a, b){

return a.preco - b.preco;

});

**Join**

**Gera um string**

Carros.join(“;”); //‘Ka;Corsa;Palio;Gol’

**Expressões Regulares RefExp**

Sequência de caracteres que especificam um padrão formal.

Var regExp = /abc/;

Var regExp = new RegExp(“abc”);

**exec 🡪** //detalhes

**test 🡪/**/true ou false

\\ Escapa de caracteres especiais 🡪escape

var regExp = /\(48\) 9999-9999/;

var telefone = "O telefone é (48) 9999-9999, tratar com João"

console.log(regExp.test(telefone)); //**true**

**^** **circunflexo**🡪 Inicia com um determinado caractere.

**$** 🡪 Finaliza com um determinado caractere.

var regExp = /^\(48\) 9999-9999/;

var telefone = "48) 9999-9999"

console.log(regExp.test(telefone)); //**False**

var regExp = /^\(48\) 9999-9999$/; **ou** var regExp = /^\(48\) 9999-9999/;

var telefone = "(48) 9999-9999"

console.log(regExp.test(telefone)); //**True**

**^** **circunflexo 🡪 Quando está dentro de algum grupo é negação: [^abc], não caracteres dentro desse grupo.**

**Conjuntos**

var regExp = /^\([0-9][0-9]\) [0-9][0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$/;

var telefone = "(51) 99157-0963"

console.log(regExp.test(telefone)); //true

**Evitar repetição usar quantificadores**

**Quantificadores**

**{n} – Quantifica um número específico.**

**{n,} - Quantifica um número mínimo.**

**{n,m} - Quantifica um número mínimo e máximo.**

var regExp = /^\([0-9]{2}\) [0-9]{4,5}-[0-9]{4}$/;

var telefone = "(51) 99157-0963"

console.log(regExp.test(telefone)); //true

var regExp = /^\([0-9]{2}\) [0-9]{4,5}-[0-9]{4}$/;

var telefone1 = "(51) 99157-0963"

var telefone2 = "(51) 9157-0963"

console.log(regExp.test(telefone)); //**true**

console.log(regExp.test(telefone2)); //**true**

**? – Zero ou um**

**\* – Zero ou mais**

**+ – Um ou mais**

var regExp = /^\([0-9]{2}\) [0-9]{4,5}-?[0-9]{4}$/;

var telefone1 = "(51) 991570963"

var telefone2 = "(51) 99157-0963"

console.log(regExp.test(telefone1)); //**false**

console.log(regExp.test(telefone2)); //**true**

var regExp = /<table><tr>(<td>\([0-9]{2}\) [0-9]{4,5}-?[0-9]{4}<\/td>)+<\/tr><\/table>/;

var telefone = "<table><tr><td>(80) 999778899</td><td>(90) 99897-8877</td><td>(70) 98767-9999</td></tr></table>";

console.log(regExp.test(telefone));

**Metacharacters 🡪 Expressões, pesquisar google tabela.**

**\d = [0-9]**

**\s = espaço**

var regExp = /<table><tr>(<td>\(\d{2}\)\s\d{4,5}-?\d{4}<\/td>)+<\/tr><\/table>/;

var telefone = "<table><tr><td>(80) 999778899</td><td>(90) 99897-8877</td><td>(70) 98767-9999</td></tr></table>";

console.log(regExp.test(telefone));

var regExp = /\(\d{2}\)\s\d{4,5}-?\d{4}/;

var telefone = "<table><tr><td>(80) 999778899</td><td>(90) 99897-8877</td><td>(70) 98767-9999</td></tr></table>";

console.log(telefone.match(regExp));

**Modificadores**

**i – case-insesitive matching.**

**g – Global matching.**

**m – Multiline matching.**

var regExp = /\(\d{2}\)\s\d{4,5}-?\d{4}/g;

var telefone = "<table><tr><td>(80) 999778899</td><td>(90) 99897-8877</td><td>(70) 98767-9999</td></tr></table>";

console.log(telefone.match(regExp));

**//retorna apenas os telefones dentro de um array**

**["(80) 999778899", "(90) 99897-8877", "(70) 98767-9999"]**

var regExp = /\(\d{2}\)\s\d{4,5}-?\d{4}/g;

var telefone = "<table><tr><td>(80) 999778899</td><td>(90) 99897-8877</td><td>(70) 98767-9999</td></tr></table>";

console.log(telefone.replace(regExp, "telefone"));

**//Troca todos os números por telefone**

**<table><tr><td>telefone</td><td>telefone</td><td>telefone</td></tr></table>**

**Date**

**Era Unix 01/01/1970**

**UTF**

new Date("2021-02-16")

Mon Feb 15 2021 21:00:00 GMT-0300 (Horário Padrão de Brasília)

Time zone GMT-0300, é onde vivo por isso o -3, se vivesse no greenwich daria GMT-0000

new Date("2021-02-16T21:30:00")

new Date("2021-02-16T21:30:00Z")

Tue Feb 16 2021 18:30:00 GMT-0300 (Horário Padrão de Brasília)

**Operadores**

10 == new Number(10) //true

É como se houvesse um valueOf() interno que não podemos ver.

Se não houver valueOf(), utilizasse toString, mas sempre que houver valueOf(), ele irá utilizar ele.

!!0 🡪 Para acessar toBoolean();

**| 🡪** é outra operação binária, onde se algum dos dígitos binários forem 1, o resultado será um, exemplo:

int a = 60; // Em binário, 60 é 0011 1100

int b = 13; // Em binário, 13 é 0000 1101

int c = a | b; // Quando um dos dígitos forem 1, o resultado será 1 também. "c" dará 0011 1101, que é 61.

**|| 🡪** significa "or", ou "ou", por exemplo:

if (1 == 1 || 6 > 9) {

// Se 1 for igual a 1 OU 6 for maior que 9, isso vai acontecer, no caso sim, pois 1 é igual a 1, e isso já basta para o |.

}

**& 🡪 é uma operação binária**

int a = 60; // Em binário, 60 é 0011 1100

int b = 13; // Em binário, 13 é 0000 1101

int c = a & b; // Complicado, mas quando os dois dígitos forem 1, o resultado será 1, se não será 0. Ou seja, "c" dará 0000 1100, que é 12.

**&& 🡪 significa "and", ou "e", por exemplo:**

if (1 == 1 && 6 > 9) { // Se 1 for igual a 1 E 6 for maior que 9, isso vai acontecer, no caso não, pois 6 não é maior que 9. }

typeof 10; //’number’

new 🡪 operador

in 🡪 procura propriedade no objeto.

**Stataments**

For, if, else, continue, while. Switch, break. Throw, try, catch.

**Continue 🡪 finaliza laço**

**Throw 🡪 Tratamento de erro, if(!text) throw “Invalid text.”;**

**Try 🡪 tentar, Catch 🡪 caso não consiga.**

**Herança**

var homem = {

sexo: “masculino”

}

var Pedro = {

nome: “pedro”.

idade: 20

}

**//Adiciona objeto homem no joao**

Obeject.setPrototypeOf(joao, homem)

**//Cria um novo objeto**

var pedro = Obeject.create (homem)

pedro.nome = “pedro”;

**Shadowing**

**Pedro faz sombra para propriedade homem,**

Obeject.keys(pedro) // **propriedade do objeto**

**Homem.call(Pedro, “Pedro”, 18);**

**Erro manualmente.**

**Try{**

**Num = prompt(“difgite um valor”);**

**If(num > 10){**

**Throw new Error(“Valor invalido”)**

**}**

**document.write(“Valor:” +num);**

**}catch(e){**

**document.write(“Erro:” + e.message) //”e” é a messagem do throw**

**}**

**// Throw new Error, mensagem cai no “e” do catch**