

Introdução à Linguagem JavaScript

Paradigmas de Linguagens de Programação

Gabriel Marques de Amaral Gravina

Ausberto S. Castro Vera

28 de novembro de 2021



Copyright © 2021 Gabriel Marques de Amaral Gravina e Ausberto S. Castro Vera

UENF - UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE DARCY RIBEIRO

CCT - CENTRO DE CIÊNCIA E TECNOLOGIA
LCMAT - LABORATÓRIO DE MATEMÁTICAS
CC - CURSO DE CIÊNCIA DA COMPUTAÇÃO

Primeira edição, Maio 2019

JavaScript



Sumário

1	Introdução	5
1.1	Aspectos históricos da linguagem JavaScript	5
1.2	Áreas de Aplicação da Linguagem	5
1.2.1	NodeJS	6
1.2.2	Orientação a objetos	6
1.2.3	Programação Funcional	6
2	Conceitos básicos da Linguagem JavaScript	7
2.1	Estrutura Léxica	7
2.2	Operadores	8
2.3	Variáveis e Constantes	8
2.3.1	Tipos Primitivos	9
2.3.2	Tipos de Objeto	10
2.4	Estrutura de Controle e Funções	10
2.4.1	O comando IF	11
2.4.2	Laços de Repetição	11
2.4.3	For	12
2.4.4	Do ... While	12
3	Programação Orientada a Objetos com JavaScript	15
3.1	Módulos	15
3.2	Classes e Objetos	15
3.2.1	Listas	16
3.2.2	Propriedades	16
3.2.3	Criando Objetos	16

3.2.4	Lendo e adicionando propriedades	17
3.2.5	Deletando Propriedades	17
3.2.6	Prototype	18
3.3	Herança	18
3.4	Encapsulamento	18
4	Aplicações da Linguagem JavaScript	19
4.1	Pilha Implementação	19
4.1.1	Prints Pilha	19
4.2	Árvore de Busca Binária	19
4.2.1	BST Prints	25
4.3	Calculadora	25
4.3.1	Prints Calculadora	26
4.4	Implementação do QuickSort	26
4.4.1	Prints QuickSort	27
4.5	Conversor de Temperatura	27
4.5.1	Conversor de Temperatura Imagens	28
5	Ferramentas existentes e utilizadas	35
5.1	Node JS	35
5.1.1	NVM	36
5.2	Visual Studio Code	36
5.3	Interpretador UVW	36
5.4	Ambientes de Programação IDE MNP	36
6	Conclusões	37
	Bibliografia	39
	Index	41



JavaScript

1. Introdução

A linguagem de programação JavaScript é a “linguagem da web”. Seu uso é dominante na internet e praticamente quase todos os sites a utilizam. Além disso, smartphones, tablets e vários outros dispositivos têm interpretadores de JavaScript embutidos. Isso a torna uma das linguagens mais utilizadas dos dias atuais e uma das linguagens mais usadas por desenvolvedores de software. É importante dizer que, embora o nome sugira, JavaScript é uma linguagem completamente diferente e independente da linguagem Java. Mesmo assim, suas sintaxes tem traços de semelhança, mas nada além disso.

Por ser uma linguagem fácil de ser aprendida e fortemente tolerante, permitiu que usuários pudessem ter suas necessidades atendidas de forma cômoda e eficiente. A linguagem é de alto-nível, dinâmica e interpretada. Além disso, é adequada para orientação a objeto e programação funcional. É uma linguagem não tipada – ou seja, suas variáveis não tem um tipo específico e seus tipos não são importantes para a linguagem. Baseado no livro [Fla20].

1.1 Aspectos históricos da linguagem JavaScript

A linguagem foi criada na NETSCAPE por Brendan Eich. Tecnicamente, JavaScript é uma marca registrada da Sun Microsystems (atualmente Oracle) usada para descrever a implementação da língua pela Netscape (atualmente Mozilla). Na época, a Netscape enviou a linguagem para a padronização da ECMA – European Computer Manufacturer’s Association, e sua versão padronizada ficou conhecida como “ECMAScript”. Na prática, todos chamam a linguagem apenas de JavaScript. De acordo com [Fla20].

1.2 Áreas de Aplicação da Linguagem

A linguagem JavaScript é completamente versátil e tem aplicações nos mais variados ambientes, seja no client-side ou no server-side. Nesta seção falarei de algumas aplicações e paradigmas da programação que podem ser implementados em JavaScript.

1.2.1 NodeJS

A linguagem foi criada para ser utilizada em navegadores da web, e esse segue sendo seu ambiente mais comum de execução até hoje. Enfim, o ambiente do navegador permite a linguagem obter a entrada de usuários e fazer requests HTTP. Porém, em 2010 outro ambiente foi criado para executar código em JavaScript. O NodeJS, popularmente conhecido como Node, tinha a ideia de invés de manter a linguagem presa a um navegador, permitir que a linguagem tivesse acesso ao sistema operacional. Isso proporcionou a utilização da linguagem no lado do servidor, invés de se limitar apenas ao navegador. Atualmente, o Node tem grande popularidade na implementação de servidores web. Baseado no livro [Fla20].

1.2.2 Orientação a objetos

A linguagem é orientada a objeto, porém apresenta algumas diferenças que valem ser mencionadas. Na linguagem, as classes são baseadas no mecanismo de herança de protótipos. Se dois objetos herdam do mesmo objeto protótipo, então diz-se que são instâncias de uma mesma classe. Membros, ou instâncias da classe, tem suas propriedades para manter e também métodos que definem seu comportamento. Este comportamento é definido pela classe e compartilhado para todas as instâncias. Retirado do artigo da documentação da linguagem, em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>

1.2.3 Programação Funcional

Basicamente, a programação funcional é um paradigma da programação que visa produzir software através de funções puras, evitando compartilhamento de estados, dados mutáveis e efeitos colaterais. Embora JavaScript não seja uma linguagem de programação funcional como Haskell ou Lisp, o fato da linguagem poder manipular funções como objeto significa que técnicas de programação funcional podem ser implementadas na linguagem. Os métodos de array do ECMAScript 5, como map() e reduce() satisfazem bem o estilo de programação funcional. Retirado do livro [Pow15].



2. Conceitos básicos da Linguagem JavaScript

Neste capítulo serão apresentados os principais conceitos da linguagem JavaScript, sua estrutura léxica, operadores, laços de repetição entre outros tópicos. Os livros básicos e recomendados o estudo da Linguagem JavaScript são: [Fla20], [Pow15] entre outros.

2.1 Estrutura Léxica

A linguagem JavaScript é feita utilizando o set de caracteres Unicode, que dá suporte a praticamente todas as linguagens utilizadas atualmente no mundo. Essa é uma linguagem case sensitive, ou seja, os nomes de variáveis, funções e outros identificadores devem ser sempre utilizados de maneira consistente, ao contrário do que acontece no html, por exemplo. Além disso, o JavaScript ignora os espaços e as quebras de linha, com algumas exceções. Isso permite que os programas sejam identados de maneira que façam o código ser legível e fácil de entender. Falando em tornar o código legível, os comentários em JavaScript podem ser feitos de duas formas: uma delas são os comentários de uma só linha, que utilizam `"/"` e a outra são os comentários de múltiplas linhas, que ignorarão tudo que está dentro dos caracteres. Observe o exemplo abaixo:

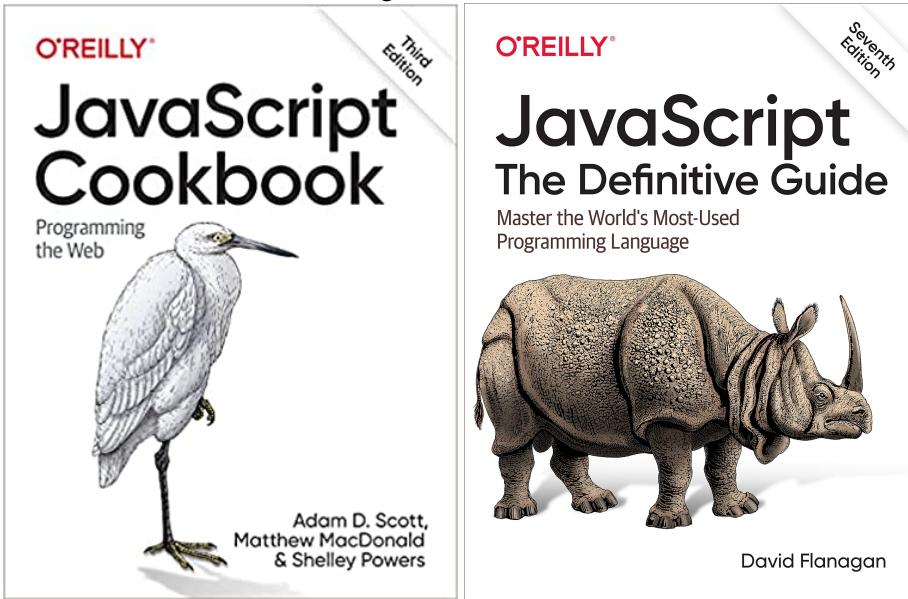
```

1 /* Explicacao do codigo
2   O codigo abaixo realiza... */
3 var helloWorld = function(){
4   console.log('Hello World!');
5 }
6 helloWorld();

```

*No JavaScript o uso de vírgulas é opcional.

Figura 2.1: Melhores Livros



Fonte: O autor

2.2 Operadores

A linguagem JavaScript tem operadores

```

1   ++
2   --
3   !
4   ==
5   !=
6   !==
7   //testa por igualdade estrita, ou seja, o tipo tambem tem que ser o
8   mesmo
9   ===
10  ||
11  &&
12  =
13  *=, /=, %=, +=...
14  <, >, <=, >=
15

```

2.3 Variáveis e Constantes

Com base no livro[Fla20], uma variável é, de forma resumida, um nome simbólico para um valor armazenado no computador. Quando chamamos uma variável, estamos acessando o valor guardado por ela.

Na linguagem JavaScript, existem dois tipos de variáveis: as primitivas e as de objeto. Para se declarar uma no JavaScript é necessário utilizar a palavra reservada "var" ou "let" seguida de seu nome. A linguagem automaticamente detectará o tipo, sem ser necessário especificá-lo com antecedência, o que é o caso em linguagens como C e Java. Abaixo encontram-se exemplos da

declaração de variáveis no JavaScript:

```

1 //E possivel declarar uma variavel vazia
2 var a;
3 var b = 100;
4 var name = "Lucas";
5
6 //Tambem e possivel declarar multiplas variaveis numa so linha
7 var A = 0, B = 1, C = 2;
8
9 //Variaveis tambem podem ser criadas dentro de lacos de repeticao
10 (for var i = 0; i<10; i++){
11     console.log(i)
12 }
13

```

2.3.1 Tipos Primitivos

Os tipos primitivos do JS incluem números, strings de textos e valores booleanos (true e false). Além disso, existem também os tipos especiais "null" e "undefined", que são valores primitivos, porém não são números, strings ou booleanos. Nesse sentido, cada um é considerado membro de um tipo especial.

2.3.1.1 Números

Uma fator da linguagem JavaScript que é incomum em outras línguas é que não há distinção entre inteiros e floats, sendo todos os números representados como floats. A linguagem armazena os números utilizando o formato de floats de 64 bits, podendo armazenar números grandes com precisão considerável.

2.3.1.2 Strings

De acordo com [Fla20], uma string é uma sequência imutável de valores de 16 bits, onde cada um representa geralmente um caractere Unicode. O tamanho da string dependerá de quantos desses valores ela contém. Para incluir uma string num programa, basta colocar aspas (simples ou duplas). Por exemplo:

```

1 //E uma string vazia
2 ''
3
4 "10.24"
5 //Utilizacao da combinacao de aspas simples e duplas
6 'O numero "8" e par'
7
8 mensagemOla = "Ola, seja bem vindo"
9 //Printa o conteudo da variavel no console do navegador
10 console.log(mensagemOla)
11
12 /*compara o valor da variavel e retorna true ou false.
13 No caso, retornara true*/
14 a = "Ola"
15 a == "Ola"
16
17
18

```

2.3.1.3 Booleanos

Conforme [Pow15], um valor booleano é um valor que representa verdade ou falsidade. Deste modo, só há dois possíveis valores para um booleano. No JavaScript, as palavras reservadas para os booleanos são "true" e "false", e são geralmente o resultado de uma comparação. Observe o exemplo:

```

1  a = 10
2  b = 3
3  a == b
4  false
5
6  a == 10
7  true
8

```

2.3.1.4 Tipos Especiais

Consoante a [Fla20], "null" é uma palavra reservada que geralmente indica ausência de um valor. Se utilizarmos o comando "typeof" no "null", veremos que será retornado "object", o que significa que "null" é algo que indica a ausência de objeto. Resumindo, pode ser utilizado para indicar que não há valor em uma variável, string ou objeto.

Por isso, "null" e "undefined" costumam ser definidos como um único objeto do seu tipo.

2.3.2 Tipos de Objeto

Segundo [Fla20], qualquer valor que não seja um número, string, objeto ou null e undefined é um objeto, ou seja, é uma coleção de propriedades onde cada uma tem um nome e valor.

2.3.2.1 Globais

Os objetos globais são aqueles que podem ser usados em todo programa escrito em JavaScript. Quando um interpretador da linguagem inicia, ele cria novos objetos globais e os dá as propriedades que o definem. Algumas das propriedades globais existentes são: undefined, Infinity, NaN. Além de propriedades e funções globais, no JavaScript existem também constructor functions, como: Date(), Object() e objetos globais, como o Math e JSON.

São exemplos de funções globais:

```

1  isNaN()          //Retorna se um valor é um numero ou não.
2  parseInt()       //Recebe o conteúdo de uma string e converte para.
      inteiro
3  parseFloat()    //Recebe uma string e a converte para float.
4  eval()           //Avalia código representado por uma string.
5  isFinite()        //Verifica se um numero é finito.
6
7

```

2.4 Estrutura de Controle e Funções

De acordo com [Fla20], uma estrutura de controle dita a ordem em que instruções serão executadas. Estruturas muito conhecidas em outras linguagens estão presentes também no JavaScript.

2.4.1 O comando IF

O comando IF funciona para fazer com que o JavaScript execute expressões condicionalmente. Isso significa que o computador somente executará uma determinada instrução caso a condição seja verdadeira. Caso a seja falsa, o programa executará outro do bloco de código. O comando IF na linguagem toma a seguinte forma:

```

1 //Sintaxe
2 if(condicao){
3     //realiza instrucao A
4 }else{
5     //realiza outra instrucao
6 }
7
8 //-----Exemplo-----
9 var nome = "Marcos"
10
11 if(nome == Marcos){
12     console.log("Bem vindo, Marcos!")
13 }else{
14     console.log("Apenas Marcos pode ler esta mensagem!")
15 }
16

```

É possível também utilizar IFs dentro de outros IFs, como no exemplo abaixo:

```

1
2 if(animal == cachorro){
3     console.log("Um cachorro e um animal")
4     if(cachorro == panda){
5         console.log("Um panda e um cachorro")
6     }else{
7         console.log("Um panda nao e um cachorro")
8     }
9 }
10
11

```

2.4.2 Laços de Repetição

Laços de repetição executam uma instrução até que uma determinada condição seja verdadeira. Na linguagem JavaScript, os laços de repetição são o 'while', 'do ... while' e o 'for'.

2.4.2.1 While

Os laços de repetição "while" tem a seguinte sintaxe no JavaScript:

```

1 while(condicao == true){
2     execute...
3 }
4
5 //O programa abaixo conta de 0 a 999

```

```

6     vai i = 0;
7
8     while(i < 1000){
9         console.log(i)
10        i++
11    }
12
13

```

É importante que o laço "while" atinja em algum momento uma condição de saída. Caso contrário, o programa continuará executando indefinidamente. No exemplo abaixo, temos um programa sem condição de saída.

```

1     while(Bolsonaro == Horroroso){
2         console.log("O presidente é horroroso")
3     }
4 //O programa printara a mensagem acima indefinidamente
5

```

2.4.3 For

Geralmente, os laços que utilizam o "for" são mais simples de serem lido. Isso devido ao fato de poderem executar uma variável inicial, testá-la e incrementá-la em uma única linha. Na linguagem, o laço for funciona com a seguinte sintaxe:

```

1     for(inicia variavel; testa condicao; incrementa){
2         realiza instrucao
3     }
4
5 //-----Exemplo-----
6 //O programa abaixo calcula fatoriais
7 var fatorial = 10;
8 var resultado = fatorial;
9 var multiplicadorInicial = fatorial - 1
10
11 for(var i = multiplicadorInicial; i > 1; i--){
12     resultado = resultado * i;
13 }
14
15 console.log(resultado)
16
17

```

2.4.4 Do ... While

Ao contrário do "while" e do for, o "dowhile" verifica a condição apenas no final da função. A sintaxe do "dowhile" no JavaScript é a listada abaixo:

```

1     do
2         instrucao

```

```
3  while(condicao == true)
4
5  //-----EXEMPLO-----
6  //O programa abaixo conta de 1 a 100
7  var i = 0;
8  do
9    i++
10   console.log(contador)
11  while(i<100)
12
13
```

Caso o código acima fosse executado com o "while", o programa contaria apenas de 1 a 99, já que a checagem no início impediria o programa de fazer mais uma iteração.



3. Programação Orientada a Objetos com JavaScript

3.1 Módulos

Para tornar o código extensível, reutilizável e acessível, é interessante organizá-lo em classes. Porém, no JavaScript as classes não são o único tipo de código modular. Geralmente, um módulo é um único arquivo de JavaScript, e qualquer pedaço escrito na linguagem pode ser um módulo. Para acessar um módulo primeiro temos que exportá-lo, e isso é feito com a palavra "export". Abaixo, temos um exemplo de um método.

```

1  export const nome = 'triangulo'
2
3  export function desenha(forma ,tamanho , x , y , cor)
4      forma.fillStyle = cor;
5      forma.fillTriangulo(x , y , tamanho)
6
7  return {
8      tamanho: tamanho ,
9      x: x ,
10     y: y ,
11     cor:cor
12  };
13 }
14 }
```

3.2 Classes e Objetos

De acordo com [Fla20], objetos são o tipo de dados fundamentais do JavaScript. Qualquer valor que não seja um tipo "true", "false", "null"ou "undefined", é um objeto. Isso nada mais é do que um valor composto que é constituído de múltiplos valores. Sendo assim, um objeto permite seu

armazenamento e sua busca pelo nome. Na linguagem, os objetos são dinâmicos, o que significa que propriedades podem ser adicionadas ou removidas.

3.2.1 Listas

Listas são um conjunto de dados e características armazenados dentro de uma variável. Os conteúdos de uma lista podem ser acessados através do index dos elementos. Abaixo estão alguns exemplos de como as listas funcionam no JavaScript:

```

1 //Cria uma lista com esses elementos
2 let Alimentos = ['Banana', 'Laranja', 'Melancia', 'Mexirica']
3 //imprime o segundo elemento da lista de alimentos (Laranja)
4 console.log(Alimentos[1])

```

Ambos os objetos e as listas são tipos de dados que podem ser alterados e utilizados para armazenar vários valores. Os objetos servem para representar algo que pode ser definido juntamente às suas características. Por exemplo: um ser humano pode ter seu nome e idade, e, além disso, seus comportamentos herdados de seus pais. A abstração do que é um ser humano é feita utilizando uma classe, que funcionará como um molde para o objeto criado.

Diferente de um objeto, uma lista serve apenas como um meio de armazenar dados em uma única variável. Nesse sentido, os conceitos da orientação a objeto, como herança e polimorfismo, não seriam possíveis de serem implementados dentro de uma lista.

3.2.2 Propriedades

Uma propriedade tem nome e valor, e seu nome pode ser uma string porém não pode existir um objeto que tenha mais de uma propriedade com mesmo nome. Os valores das propriedades podem ser quaisquer que existam dentro da linguagem.

Além de nome e valor, cada propriedade tem valores associados que chamados de atributos de propriedades.

3.2.2.1 Atributos

Segundo [Fla20], o JavaScript apresenta os seguintes atributos de propriedades: "writable" diz se o valor da propriedade pode ser atribuído. Caso seja falso, o valor da propriedade não pode ser alterado. "enumerable" diz se o nome da propriedade é retornado por um for/in loop. Se verdadeiro, a propriedade aparece durante a enumeração das propriedades do objeto correspondente. "configurable" especifica se a propriedade pode ser deletada ou se seus atributos podem ser alterados.

3.2.3 Criando Objetos

A linguagem JavaScript apresenta várias formas de criar um objeto, e uma forma simples e fácil de criá-los é inserindo um literal de objeto. Essa é uma forma extremamente prática e intuitiva, o que torna a programação orientada a objetos na linguagem muito mais simples.

Um literal de objeto contém a propriedade e o seu valor, seguido de vírgulas. Abaixo, um exemplo de um literal de objeto:

```

1 var objeto = {
2     primeiraPropriedade: "Caracteristica 1",

```

```

3     segundaPropriedade: 101,
4     terceiraPropriedade: false ,
5     data: {
6       dia: 12,
7       ano: 2003
8     }
9
10    }
11

```

Além disso, há também o operador "new", que cria e inicializa um objeto. Para isso, a palavra reservada "new" vem seguida de uma chamada de função. Essa função é chamada de função construtora e tem como objetivo a inicialização do novo objeto.

```

1 var objeto = new Object() //Cria um objeto vazio {}
2

```

3.2.4 Lendo e adicionando propriedades

Para obter valores de objetos utilizamos o ponto (.) ou colchetes ([]). O exemplo abaixo adiciona propriedades a um objeto chamado pessoa, lê e as coloca em variáveis.

```

1
2 var pessoa = new Object() //Cria um objeto pessoa vazio
3 pessoa.nome = "Marcelo"
4 pessoa.idade = 8
5 pessoa.sexo = "M"
6
7 console.log(pessoa.nome) //Mostra o valor da propriedade nome de pessoa
8 console.log(pessoa.idade) //Mostra o valor da propriedade idade de pessoa
9 console.log(pessoa.sexo) //Mostra o valor da propriedade sexo de pessoa
10
11 console.log(pessoa["nome"]) //E o mesmo que o código da linha 7
12

```

3.2.5 Deletando Propriedades

O operador "delete" remove uma propriedade de um objeto. Isso significa que se um objeto tem uma propriedade, o seu conteúdo não será deletado, mas sim a propriedade em si. O exemplo abaixo ilustra o que aconteceria ao apagar uma propriedade de um objeto existente:

```

1 //Cria um novo objeto chamado pessoa
2 var pessoa = new Object()
3
4 //define a propriedade "nome" como sendo "Lucas"
5 pessoa.nome = "Lucas"
6 //define a propriedade "idade" valendo 18
7 pessoa.idade = 18
8 //define a profissao

```

```

9  pessoa.profissao = "Engenheiro"
10 //define o cpf
11 pessoa.cpf = "123.456.789-00"
12
13 //printa a propriedade cpf do objeto pessoa
14 console.log(pessoa.cpf)
15 //deleta a propriedade cpf do objeto pessoa
16 delete pessoa.cpf
17 //printa a propriedade "cpf" de pessoa, porém, como essa propriedade foi
   deletada, o console irá retornar "undefined"
18 console.log(pessoa.cpf)
19

```

3.2.6 Prototype

Como abordado por [Fla20], uma classe é um conjunto de objetos que herdam propriedades do mesmo objeto prototype. O objeto prototype é herdado por todo objeto criado, e todas as classes herdam dele.

3.3 Herança

Um dos conceitos mais importantes da programação orientada a objetos é a Herança. Ela serve para que um objeto consiga herdar características de um objeto mãe. Isso permite que o código não necessite de ser reescrito. Na linguagem, cada objeto tem um conjunto de propriedades próprias, e elas também herdam propriedades de seu objeto prototype. No exemplo abaixo, temos o exemplo de uma classe "Carro" que herda da classe "Veiculo":

```

1 class Carro extends Veiculo {
2     rodas = 4;
3     cor = "Vermelho";
4 }
5

```

3.4 Encapsulamento

Por definição, o encapsulamento é o processo de esconder dados. Isso acontece porque nem sempre é seguro ou interessante permitir que determinados dados sejam acessados por qualquer um dentro do programa, e por isso costumamos separar a implementação através de uma interface. Basicamente, o processo de encapsulamento traz uma camada de segurança e confiabilidade ao código. No JavaScript é permitido utilizar variáveis privadas para permitir o encapsulamento. A linguagem permite a utilização de getters e setters que não podem ser deletados.



4. Aplicações da Linguagem JavaScript

O capítulo abaixo irá demonstrar implementações em JavaScript de aplicações ou estruturas de dados conhecidas. O código será explicado nos comentários e além disso, em alguns casos o código será feito em html, CSS e JavaScript.

4.1 Pilha Implementação

```
1 let stack = [];  
2  
3     stack.push(1);  
4     console.log(stack); // [1]  
5  
6     stack.push(2);  
7     console.log(stack); // [1,2]  
8  
9     stack.push(3);  
10    console.log(stack); // [1,2,3]  
11  
12    stack.push(4);  
13    console.log(stack); // [1,2,3,4]  
14  
15    stack.push(5);  
16    console.log(stack); // [1,2,3,4,5]
```

O conteúdo foi retirado de: <https://www.javascripttutorial.net/javascript-stack/>

4.1.1 Prints Pilha

4.2 Árvore de Busca Binária

Código retirado de: <https://www.geeksforgeeks.org/implementation-binary-search-tree-javascript/>

```
1 // Node class
2 class Node
3 {
4     constructor(data)
5     {
6         this.data = data;
7         this.left = null;
8         this.right = null;
9     }
10 }
11
12 // Binary Search tree class
13 class BinarySearchTree
14 {
15     constructor()
16     {
17         // root of a binary search tree
18         this.root = null;
19     }
20
21     // function to be implemented
22     // insert(data)
23     // remove(data)
24     insert(data)
25     {
26         // Creating a node and initialising
27         // with data
28         var newNode = new Node(data);
29
30         // root is null then node will
31         // be added to the tree and made root.
32         if(this.root === null)
33             this.root = newNode;
34         else
35
36             // find the correct position in the
37             // tree and add the node
38             this.insertNode(this.root, newNode);
39     }
40
41     // Method to insert a node in a tree
42     // it moves over the tree to find the location
43     // to insert a node with a given data
44     insertNode(node, newNode)
45     {
46         // if the data is less than the node
47         // data move left of the tree
48         if(newNode.data < node.data)
49         {
50             // if left is null insert node here
51             if(node.left === null)
52                 node.left = newNode;
53             else
54
55                 // if left is not null recur until
56                 // null is found
57                 this.insertNode(node.left, newNode);
58         }
59
60         // if the data is more than the node
```

```
61     // data move right of the tree
62     else
63     {
64         // if right is null insert node here
65         if(node.right === null)
66             node.right = newNode;
67         else
68
69             // if right is not null recur until
70             // null is found
71             this.insertNode(node.right,newNode);
72         }
73     }
74     search(node, data)
75     {
76         // if trees is empty return null
77         if(node === null)
78             return null;
79
80         // if data is less than node's data
81         // move left
82         else if(data < node.data)
83             return this.search(node.left, data);
84
85         // if data is less than node's data
86         // move left
87         else if(data > node.data)
88             return this.search(node.right, data);
89
90         // if data is equal to the node data
91         // return node
92         else
93             return node;
94     }
95
96     // returns root of the tree
97     getRootNode()
98     {
99         return this.root;
100    }
101    // finds the minimum node in tree
102    // searching starts from given node
103    findMinNode(node)
104    {
105        // if left of a node is null
106        // then it must be minimum node
107        if(node.left === null)
108            return node;
109        else
110            return this.findMinNode(node.left);
111    }
112    // Performs postorder traversal of a tree
113    postorder(node)
114    {
115        if(node !== null)
116        {
117            this.postorder(node.left);
118            this.postorder(node.right);
119            console.log(node.data);
120        }
121    }
```

```
122 // Performs preorder traversal of a tree
123 preorder(node)
124 {
125     if(node !== null)
126     {
127         console.log(node.data);
128         this.preorder(node.left);
129         this.preorder(node.right);
130     }
131 }
132
133 // helper method that calls the
134 // removeNode with a given data
135 remove(data)
136 {
137     // root is re-initialized with
138     // root of a modified tree.
139     this.root = this.removeNode(this.root, data);
140 }
141
142 // Method to remove node with a
143 // given data
144 // it recur over the tree to find the
145 // data and removes it
146 removeNode(node, key)
147 {
148
149     // if the root is null then tree is
150     // empty
151     if(node === null)
152         return null;
153
154     // if data to be delete is less than
155     // roots data then move to left subtree
156     else if(key < node.data)
157     {
158         node.left = this.removeNode(node.left, key);
159         return node;
160     }
161
162     // if data to be delete is greater than
163     // roots data then move to right subtree
164     else if(key > node.data)
165     {
166         node.right = this.removeNode(node.right, key);
167         return node;
168     }
169
170     // if data is similar to the root's data
171     // then delete this node
172     else
173     {
174         // deleting node with no children
175         if(node.left === null && node.right === null)
176     {
177             node = null;
178             return node;
179         }
180
181         // deleting node with one children
182         if(node.left === null)
```

```
183  {
184    node = node.right;
185    return node;
186  }
187
188  else if(node.right === null)
189  {
190    node = node.left;
191    return node;
192  }
193
194 // Deleting node with two children
195 // minimum node of the right subtree
196 // is stored in aux
197 var aux = this.findMinNode(node.right);
198 node.data = aux.data;
199
200 node.right = this.removeNode(node.right, aux.data);
201 return node;
202 }
203
204 // search for a node with given data
205
206 }
207 // Performs inorder traversal of a tree
208 inorder(node)
209 {
210   if(node !== null)
211   {
212     this.inorder(node.left);
213     console.log(node.data);
214     this.inorder(node.right);
215   }
216 }
217
218
219 // Helper function
220 // findMinNode()
221 // getRootNode()
222 // inorder(node)
223 // preorder(node)
224 // postorder(node)
225 // search(node, data)
226 }
227
228 // create an object for the BinarySearchTree
229 var BST = new BinarySearchTree();
230
231 // Inserting nodes to the BinarySearchTree
232 BST.insert(15);
233 BST.insert(25);
234 BST.insert(10);
235 BST.insert(7);
236 BST.insert(22);
237 BST.insert(17);
238 BST.insert(13);
239 BST.insert(5);
240 BST.insert(9);
241 BST.insert(27);
242
243 //      15
```

```
244 //      / \
245 //    10  25
246 //    / \ / \
247 //    7  13  22  27
248 //    / \ /
249 //  5  9  17
250
251 var root = BST.getNode();
252
253 // prints 5 7 9 10 13 15 17 22 25 27
254 BST.inorder(root);
255
256 // Removing node with no children
257 BST.remove(5);
258
259
260 //      15
261 //      / \
262 //    10  25
263 //    / \ / \
264 //    7  13  22  27
265 //    \ /
266 //     9  17
267
268
269 var root = BST.getNode();
270
271 // prints 7 9 10 13 15 17 22 25 27
272 BST.inorder(root);
273
274 // Removing node with one child
275 BST.remove(7);
276
277 //      15
278 //      / \
279 //    10  25
280 //    / \ / \
281 //    9  13  22  27
282 //    /
283 //     17
284
285
286 var root = BST.getNode();
287
288 // prints 9 10 13 15 17 22 25 27
289 BST.inorder(root);
290
291 // Removing node with two children
292 BST.remove(15);
293
294 //      17
295 //      / \
296 //    10  25
297 //    / \ / \
298 //    9  13  22  27
299
300 var root = BST.getNode();
301 console.log("inorder traversal");
302
303 // prints 9 10 13 17 22 25 27
304 BST.inorder(root);
```

```

JS BST.js > ...
1 // Node class
2 class Node
3 {
4     constructor(data)
5     {
6         this.data = data;
7         this.left = null;
8         this.right = null;
9     }
10}
11
12 // Binary Search tree class
13 class BinarySearchTree
14 {
15     constructor()
16     {
17         // root of a binary search tree
18         this.root = null;
19     }
20
21     // function to be implemented
22     // insert(data)
23     // remove(data)
24     // insert(data)
25     {
26         // Creating a node and initialising
27         // with data
28         var newNode = new Node(data);
29
30         // if root is null then node will
31         // be added to the tree and made root.
32         if(this.root === null)
33             this.root = newNode;
34         else
35             // find the correct position in the
36             // tree and add the node
37             this.insertNode(this.root, newNode);
38     }
39
40     // Method to insert a node in a tree
41     // it moves over the tree to find the location
42     // to insert a node with a given data
43     insertNode(node, newNode)
44     {
45         // if the data is less than the node
46         // data move left of the tree
47         if(newNode.data < node.data)
48             ...
49
50         // if left is null insert node here
51         if(node.left === null)
52             node.left = newNode;
53         else
54             // if left is not null recur until
55             // null is found
56             this.insertNode(node.left, newNode);
57
58         ...
59
60         // if the data is more than the node
61         // data move right of the tree
62         else
63         {
64             // if right is null insert node here
65             if(node.right === null)
66                 node.right = newNode;
67             else
68                 // if right is not null recur until
69                 // null is found
70                 this.insertNode(node.right, newNode);
71
72         }
73
74         search(node, data)
75     {
76         // If trees is empty return null
77         if(node === null)
78             return null;
79
80         // If data is less than node's data
81         if(data < node.data)
82             move left
83             else if(data < node.data)
84                 return this.search(node.left, data);
85
86         // If data is less than node's data
87         // move left
88         else if(data > node.data)
89             return this.search(node.right, data);
90
91         // If data is equal to the node data
92         else
93             return node;
94
95     }
96
97     // returns root of the tree
98     getRootNode()
99     {
100        ...
101        // finds the minimum node in tree
102        // searching starts from given node
103        findMinNode(node)
104        {
105            // if left of a node is null
106            // then it must be minimum node
107            if(node.left === null)
108                return node;
109            else
110                return this.findMinNode(node.left);
111
112        // Performs postorder traversal of a tree
113        postorder(node)
114        {
115            if(node !== null)
116            {
117                this.postorder(node.left);
118                this.postorder(node.right);
119                console.log(node.data);
120            }
121
122        // Performs preorder traversal of a tree
123        preorder(node)
124        {
125            if(node !== null)
126            {
127                console.log(node.data);
128                this.preorder(node.left);
129                this.preorder(node.right);
130            }
131        }
132
133        // helper method that calls the
134        // removeNode with a given data
135        remove(data)
136        {
137            // root is re-initialized with
138            // root of a modified tree.
139            this.root = this.removeNode(this.root, data);
140        }
141
142        // Method to remove node with a
143        // given data
144    }
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
745
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
154
```

```

143 // given data
144 // it recur over the tree to find the
145 // data and removes it
146 removeNode(node, key)
147 {
148     // if the root is null then the tree is
149     // empty
150     if(node === null)
151         return null;
152
153     // if data to be delete is less than
154     // root's data then move to left subtree
155     else if(key < node.data)
156     {
157         node.left = this.removeNode(node.left, key);
158         return node;
159     }
160
161     // if data to be delete is greater than
162     // root's data then move to right subtree
163     else if(key > node.data)
164     {
165         node.right = this.removeNode(node.right, key);
166         return node;
167     }
168
169     // if data is similar to the root's data
170     // then delete this node
171     else
172     {
173         // deleting node with no children
174         if(node.left === null && node.right === null)
175         {
176             node = null;
177             return node;
178         }
179
180         // deleting node with one children
181         if(node.left === null)
182         {
183             node = node.right;
184             return node;
185         }
186
187         else if(node.right === null)
188         {
189             node = node.left;
190             return node;
191         }
192
193         // if the node has two children
194         // Deleting node with two children
195         // node becomes the right subtree
196         // is stored in aux
197         var aux = this.findMinNode(node.right);
198         node.data = aux.data;
199
200         node.right = this.removeNode(node.right, aux.data);
201
202         return node;
203     }
204
205     // search for a node with given data
206
207     // Performs inorder traversal of a tree
208     inorder(node)
209     {
210         if(node !== null)
211         {
212             this.inorder(node.left);
213             console.log(node.data);
214             this.inorder(node.right);
215         }
216     }
217
218
219     // Helper function
220     findMinNode()
221     {
222         // getRootNode()
223         // inorder(node)
224         // postorder(node)
225         // searchInode, data
226     }
227
228     // Create an object for the BinarySearchTree
229     var BST = new BinarySearchTree();
230
231     // Inserting nodes to the BinarySearchTree
232     BST.insert(15);
233     BST.insert(25);
234     BST.insert(10);
235     BST.insert(7);
236     BST.insert(22);
237     BST.insert(13);
238     BST.insert(5);
239     BST.insert(9);
240     BST.insert(27);
241     BST.insert(21);
242
243     //      15
244     //    /   \
245     //   10  25
246     //  / \   \
247     // 7  13 22 27
248     //   \   \
249     //    5  9 17
250
251     var root = BST.getRootNode();
252
253     // prints 5 7 9 10 13 15 17 22 25 27
254     BST.inorder(root);
255
256     // Removing node with no children
257     BST.remove(5);
258
259     //      15
260     //    /   \
261     //   10  25
262     //  / \   \
263     // 7  13 22 27
264     //   \   \
265     //    9  17
266
267     // Removing node with one child
268     BST.remove(7);
269
270     // prints 7 9 10 13 15 17 22 25 27
271     BST.inorder(root);
272
273     //      15
274     //    /   \
275     //   10  25
276     //  / \   \
277     // 7  13 22 27
278     //   \   \
279     //    9 13 22 27
280     //      /
281     //   9 13 22 27
282     //      /
283     //    17

```

Figura 4.2:

```

15     return value1 * value2;
16 } else if
17 (operator == "/") {
18     return value1 / value2;
19 } else {
20     throw new Error('Operacao invalida');
21 }
22 }

23
24
25     console.log('O resultado e: ', calcular(operator, valor1, valor2))
26

```

4.3.1 Prints Calculadora

Imagens do código e do resultado, respectivamente:

4.4 Implementação do QuickSort

O algoritmo QuickSort é feito da seguinte forma no JavaScript:

```

1
2 // basic implementation, where pivot is the first element
3 function quickSortBasic(array) {
4     if(array.length < 2) {
5         return array;
6     }
7
8     var pivot = array[0];
9     var lesserArray = [];
10    var greaterArray = [];
11

```

```

12 for (var i = 1; i < array.length; i++) {
13   if (array[i] > pivot) {
14     greaterArray.push(array[i]);
15   } else {
16     lesserArray.push(array[i]);
17   }
18 }
19
20 return quickSortBasic(lesserArray).concat(pivot, quickSortBasic(
21   greaterArray));
22
23 //***** Testing Quick sort algorithm *****/
24
25 // Returns a random integer between min (inclusive) and max (inclusive).
// Using Math.round() will give a non-uniform distribution, which we dont
// want in this case.
26
27 function getRandomInt(min, max) {
28   return Math.floor(Math.random() * (max - min + 1)) + min;
29 // By adding 1, I am making the maximum inclusive (the minimum is
// inclusive anyway). Because, the Math.random() function returns a
// floating-point, pseudo-random number in the range from 0 inclusive up
// to but not including 1
30 }
31
32 var arr = [];
33
34 for (var i = 0; i < 10; i++) { //initialize a random integer unsorted array
35   arr.push(getRandomInt(1, 100));
36 }
37
38 console.log("Unsorted array: ");
39 console.log(arr); //printing unsorted array
40
41 arr = quickSortBasic(arr, 0, arr.length - 1);
42 console.log("Sorted array: ");
43 console.log(arr);

```

O código foi retirado de: <https://javascript.plainenglish.io/quick-sort-algorithm-in-javascript-5cf5ab7d251b>

4.4.1 Prints QuickSort

Código Fonte e imagem do resultado, respectivamente:

4.5 Conversor de Temperatura

Retirado de: <https://www.delftstack.com/howto/javascript/javascript-bubble-sort/>

```

1 // TC/5 = (TF-32)/9 = (TK-273)/5
2
3 const prompt = require('prompt-sync')();
4
5 const temperatura = Number(prompt('Qual temperatura? '));
6
7

```

```
8 const escala = prompt('Qual a escala da temperatura inserida? (1: Celsius;
9   2: Fahrenheit; 3: Kelvin)');
10 switch(escala) {
11   case '1':
12     temperatura_fahrenheit = (temperatura/5)*9+32;
13     temperatura_kelvin = (temperatura + 273.15);
14     console.log("A temperatura: ", temperatura_kelvin, "K e ",
15       temperatura_fahrenheit, "graus F");
16     break;
17   case '2':
18     temperatura_celsius = ((temperatura-32)/9)*5;
19     temperatura_kelvin = ((temperatura-32)/9)*5+273.15;
20     console.log("A temperatura: ", temperatura_celsius, "graus C e ",
21       temperatura_kelvin, "K");
22     break;
23   case '3':
24     temperatura_fahrenheit = ((temperatura-273.15)/5)*9+32;
25     temperatura_celsius = (temperatura)-273.15;
26     console.log("A temperatura: ", temperatura_fahrenheit, "graus F e ",
27       temperatura_celsius, "graus C");
28     break;
29   default:
30     console.log('Insira uma escala valida de 1 a 3.');
31 }
```

4.5.1 Conversor de Temperatura Imagens

Código fonte e imagem do resultado:

```
263 //    / \ / \
264 //    7 13 22 27
265 //    \
266 //    9 17
267
268
269 var root = BST.getNode();
270
271 // prints 7 9 10 13 15 17 22 25 27
272 BST.inorder(root);
273
274 // Removing node with one child
275 BST.remove(7);
276
277 //      15
278 //      / \
279 //      10 25
280 //      / \ / \
281 //      9 13 22 27
282 //      /
283 //      17
284
285
286 var root = BST.getNode();
287
288 // prints 9 10 13 15 17 22 25 27
289 BST.inorder(root);
290
291 // Removing node with two children
292 BST.remove(15);
293
294 //      17
295 //      / \
296 //      10 25
297 //      / \ / \
298 //      9 13 22 27
299
300 var root = BST.getNode();
301 console.log("inorder traversal");
302
303 // prints 9 10 13 17 22 25 27
304 BST.inorder(root);
305
306 console.log("postorder traversal");
307 BST.postorder(root);
308 console.log("preorder traversal");
309 BST.preorder(root);
310
```

```
gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS$ node BST
5
7
9
10
13
15
17
22
25
27
7
9
10
13
15
17
22
25
27
9
10
13
15
17
22
25
27
inorder traversal
9
10
13
17
22
25
27
postorder traversal
9
13
10
22
27
25
17
preorder traversal
17
10
9
13
25
22
27
gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS$ █
```

Figura 4.4:

```

var valor1;
var valor2;
var operador;
var readlineSync = require('readline-sync');
operador = readlineSync.question("Qual operacao deseja efetuar (+) (-) (*) (/)? : \n");
valor1 = parseFloat(readlineSync.question("Insira o primeiro numero: \n"));
valor2 = parseFloat(readlineSync.question("Insira o segundo numero: \n"));

function calcular(operator, value1, value2) {
    if (operator == "+") {
        return value1 + value2;
    } else if
        (operator == "-") {
        return value1 - value2;
    } else if
        (operator == "*") {
        return value1 * value2;
    } else if
        (operator == "/") {
        return value1 / value2;
    } else {
        throw new Error('Operação inválida');
    }
}

console.log('O resultado é: ', calcular(operador, valor1, valor2))

```

Figura 4.5:

```

gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/SimpleCalculator$ node index
Qual operacao deseja efetuar (+) (-) (*) (/)? :
+
Insira o primeiro numero:
31
Insira o segundo numero:
22
O resultado é: 53
gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/SimpleCalculator$ node index
Qual operacao deseja efetuar (+) (-) (*) (/)? :
/
Insira o primeiro numero:
76
Insira o segundo numero:
9
O resultado é: 8.44444444444445
gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/SimpleCalculator$ node index
Qual operacao deseja efetuar (+) (-) (*) (/)? :
*
Insira o primeiro numero:
-86
Insira o segundo numero:
9
O resultado é: -774

```

Figura 4.6:

```

QuickSort > m: Quickjs > ...
1 // basic implementation, where pivot is the first element
2 function quickSortBasic(array) {
3     if (array.length < 2) {
4         return array;
5     }
6
7     var pivot = array[0];
8     var lesserArray = [];
9     var greaterArray = [];
10
11    for (var i = 1; i < array.length; i++) {
12        if (array[i] > pivot) {
13            greaterArray.push(array[i]);
14        } else {
15            lesserArray.push(array[i]);
16        }
17    }
18
19    return quickSortBasic(lesserArray).concat(pivot, quickSortBasic(greaterArray));
20}
21
22
23 //***** Testing Quick sort algorithm *****/
24
25 // Returns a random integer between min (inclusive) and max (inclusive). Using Math.round() will give a non-uniform distribution, which we dont want in this case.
26
27 function getRandomInt(min, max) {
28     return Math.floor(Math.random() * (max - min + 1)) + min;
29     // By adding 1, I am making the maximum inclusive (the minimum is inclusive anyway). Because, the Math.random() function returns a floating-point.
30     //pseudo-random number in the range from 0 inclusive up to but not including 1
31 }
32
33 var arr = [];
34
35 for (var i = 0; i < 10; i++) { //initialize a random integer unsorted array
36     arr.push(getRandomInt(1, 100));
37 }
38
39 console.log("Unsorted array: ");
40 console.log(arr); //printing unsorted array
41
42 arr = quickSortBasic(arr, 0, arr.length - 1);
43 console.log("Sorted array: ");
44 console.log(arr);
45

```

Figura 4.7:

```

gabriel@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/QuickSort$ node Quick
Unsorted array:
[
  10, 41, 81, 55, 25,
  68, 43, 96, 72, 36
]
Sorted array:
[
  10, 25, 36, 41, 43,
  55, 68, 72, 81, 96
]

```

Figura 4.8:

```

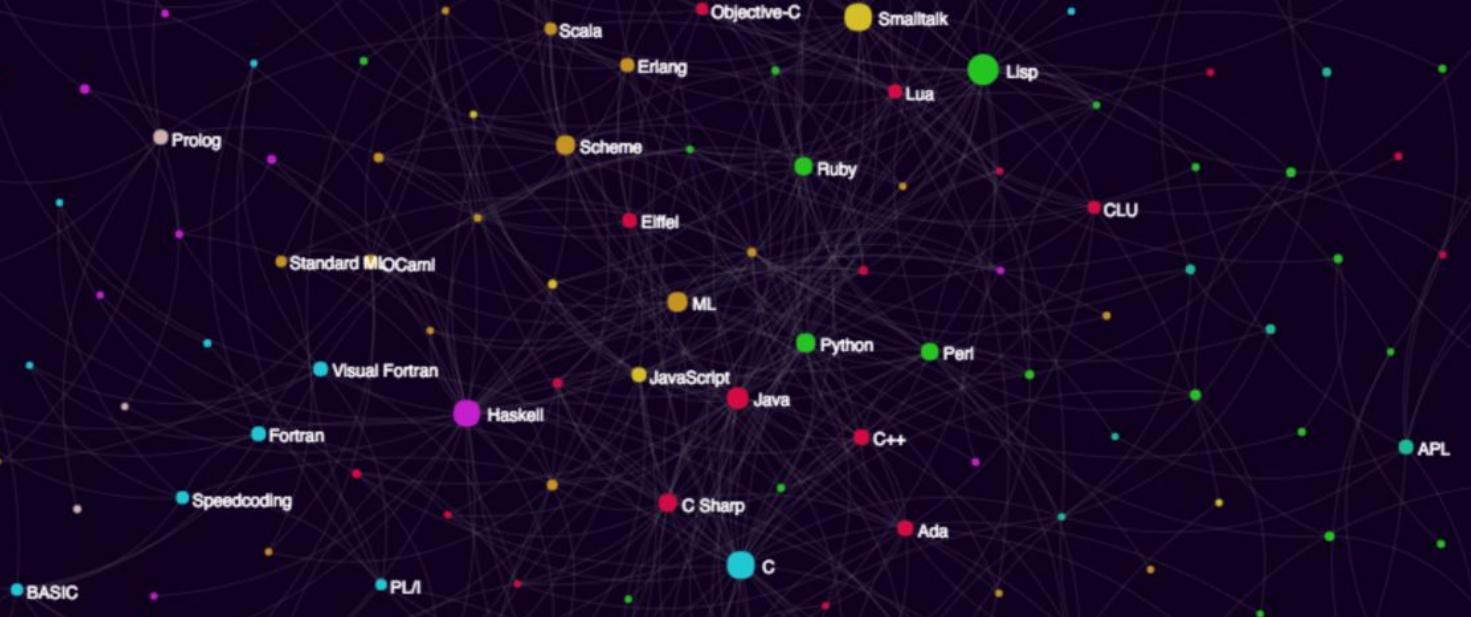
ConversorUnidades > JS index.js > ...
1 const prompt = require('prompt-sync')();
2
3 const temperatura = Number(prompt('Qual temperatura? '));
4 const escala = prompt('Qual a escala da temperatura inserida? (1: Celsius; 2: Fahrenheit; 3: Kelvin)');
5
6 switch(escala) {
7     case '1':
8         temperatura_fahrenheit = (temperatura/5)*9+32;
9         temperatura_kelvin = (temperatura + 273.15);
10        console.log("A temperatura é ", temperatura_kelvin, "K e ", temperatura_fahrenheit, "°F");
11        break;
12    case '2':
13        temperatura_celsius = ((temperatura-32)/9)*5;
14        temperatura_kelvin = ((temperatura-32)/9)*5+273.15;
15        console.log("A temperatura é ", temperatura_celsius, "°C e ", temperatura_kelvin, "K");
16        break;
17
18    case '3':
19        temperatura_fahrenheit = ((temperatura-273.15)/5)*9+32;
20        temperatura_celsius = (temperatura)-273.15;
21        console.log("A temperatura é ", temperatura_fahrenheit, "°F e ", temperatura_celsius, "°C")
22        break;
23
24    default:
25        console.log('Insira uma escala válida de 1 a 3.');
26

```

Figura 4.9:

```
gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/ConversorUnidades$ node index.js
Qual temperatura? 100
Qual a escala da temperatura inserida? (1: Celsius; 2: Fahrenheit; 3: Kelvin)1
A temperatura é 373.15 K e 212 °F
gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/ConversorUnidades$ node index.js
Qual temperatura? 0
Qual a escala da temperatura inserida? (1: Celsius; 2: Fahrenheit; 3: Kelvin)3
A temperatura é -459.6699999999996 °F e -273.15 °C
gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/ConversorUnidades$ node index.js
Qual temperatura? -32
Qual a escala da temperatura inserida? (1: Celsius; 2: Fahrenheit; 3: Kelvin)2
A temperatura é -35.5555555555556 °C e 237.59444444444443 K
gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/ConversorUnidades$ █
```

Figura 4.10:



5. Ferramentas existentes e utilizadas

Neste capítulo devem ser apresentadas pelo menos DUAS (e no máximo 5) ferramentas consultadas e utilizadas para realizar o trabalho, e usar nas aplicações. Considere em cada caso:

- Nome da ferramenta (compilador-interpretador)
- Endereço na Internet
- Versão atual e utilizada
- Descrição simples (máx 2 parágrafos)
- Telas capturadas da ferramenta
- Outras informações

5.1 Node JS

A linguagem JavaScript não está mais atrelada somente ao navegador. Por isso, para utilizar a linguagem sem precisar recorrer a um Browser, utiliza-se o nodeJS. O NodeJS, conhecido apenas como Node, nada mais é do que o V8 (Engine do JavaScript no navegador Google Chrome) fora do Chrome. Sendo assim, para instalar o Node basta entrar em <https://nodejs.org> e baixar a versão desejada.



Figura 5.1:

5.1.1 NVM

Devido a existência de várias versões do Node, o NVM - Node Version Manager - facilita o trabalho com versões diferentes. Por exemplo, se num projeto antigo decide-se pela versão 11.5, é possível com o NVM utilizar o node 11.5 sempre naquele projeto, mesmo após atualizar o node para versões mais recentes.

5.2 Visual Studio Code

Hoje em dia existem diversas versões de IDEs para as mais variadas linguagens de programação. Netbeans e Eclipse para o Java, Pycharm para o Python entre outras. IDEs feitas pensando especificamente em uma linguagem sempre tiveram vantagem em relação às IDEs multi-uso, como o sublime-text. Porém, com o Visual Studio Code, que chamarei de VSC, programar em várias linguagens é fácil e sem dor de cabeça. O VSC é uma IDE fácil de usar, relativamente leve e permite a instalação de inúmeras extensões para tornar o desenvolvimento o mais produtivo possível. Portanto, para programar em JavaScript, o VSCode é uma das melhores IDEs disponíveis. É fácil programar para web utilizando JS, HTML e CSS, e além disso, vários recursos são disponibilizados pelo programa.

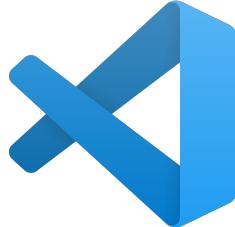


Figura 5.2:

5.3 Interpretador UVW**5.4 Ambientes de Programação IDE MNP**



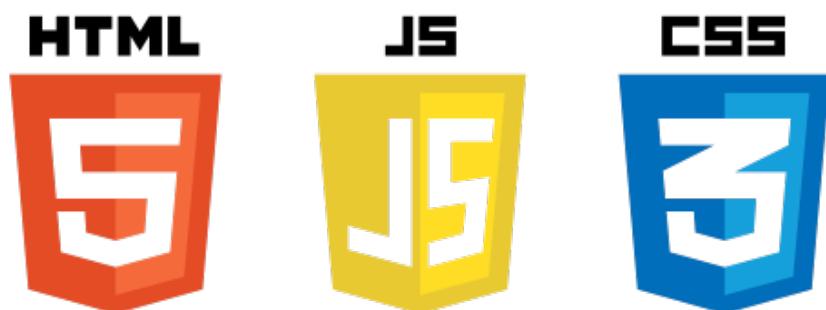
6. Conclusões

Os problemas enfrentados neste trabalho ...

O trabalho que foi desenvolvido em forma resumida ...

Aspectos não considerados que poderiam ser estudados ou úteis para ...

Figura 6.1: Linguagens de programação modernas



Fonte: O autor



Referências Bibliográficas

- [Fla20] David Flanagan. *JavaScript : the definitive guide : master the world's most-used programming language*. O'Reilly Media, Sebastopol, CA, 2020. Citado 9 vezes nas páginas [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [15](#), [16](#) e [18](#).
- [Pow15] Shelley Powers. *JavaScript cookbook : [programming the web]*. O'Reilly Media, Sebastopol, CA, 2015. Citado 3 vezes nas páginas [6](#), [7](#) e [10](#).

Disciplina: *Paradigmas de Linguagens de Programação 2021*

Linguagem: *Linguagem JavaScript*

Aluno: *Gabriel Marques de Amaral Gravina*

Ficha de avaliação:

Aspectos de avaliação (requisitos mínimos)	Pontos
Elementos básicos da linguagem (Máximo: 01 pontos) • Sintaxe (variáveis, constantes, comandos, operações, etc.) • Usos e áreas de Aplicação da Linguagem	
Cada elemento da linguagem (definição) com exemplos (Máximo: 02 pontos) • Exemplos com fonte diferenciada (Courier , 10 pts, azul)	
Mínimo 5 exemplos completos - Aplicações (Máximo : 2 pontos) • Uso de rotinas-funções-procedimentos, E/S formatadas • Menu de operações, programas gráficos, matrizes, aplicações	
Ferramentas (compiladores, interpretadores, etc.) (Máximo : 2 pontos) • Ferramentas utilizadas nos exemplos: pelo menos DUAS • Descrição de Ferramentas existentes: máximo 5 • Mostrar as telas dos exemplos junto ao compilador-interpretador • Mostrar as telas dos resultados obtidos nas ferramentas • Descrição das ferramentas (autor, versão, homepage, tipo, etc.)	
Organização do trabalho (Máximo: 01 ponto) • Conteúdo, Historia, Seções, gráficos, exemplos, conclusões, bibliografia	
Uso de Bibliografia (Máximo: 01 ponto) • Livros: pelo menos 3 • Artigos científicos: pelo menos 3 (IEEE Xplore, ACM Library) • Todas as Referências dentro do texto, tipo [ABC 04] • Evite Referências da Internet	
Conceito do Professor (Opcional: 01 ponto)	
Nota Final do trabalho:	

Observação: Requisitos mínimos significa a *metade* dos pontos