

Introdução à Linguagem JavaScript

Paradigmas de Linguagens de Programação

Gabriel Marques de Amaral Gravina

Ausberto S. Castro Vera

29 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



HTML

JS

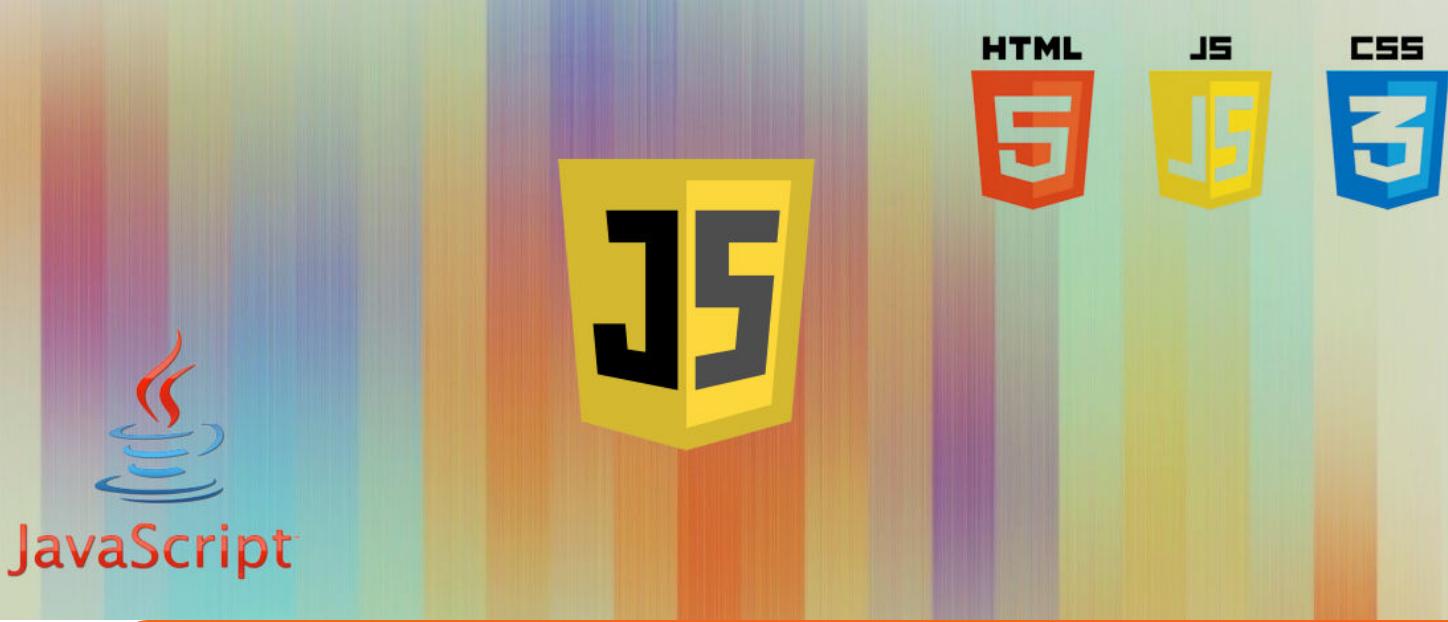
CSS



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	28
4.3.1	Prints Calculadora	29
4.4	Implementação do QuickSort	30
4.4.1	Prints QuickSort	31
4.5	Conversor de Temperatura	32
4.5.1	Conversor de Temperatura Imagens	32
5	Ferramentas existentes e utilizadas	35
5.1	Node JS	35
5.1.1	NPM	36
5.1.2	NVM	36
5.2	IDEs	37
5.2.1	Visual Studio Code	37
5.2.2	Webstorm	38
5.2.3	Atom	39
5.2.4	JS Fiddle	39
5.3	Interpretador UVW	40
5.4	Ambientes de Programação IDE MNP	40
6	Conclusões	41
	Bibliografia	43
	Index	45



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 multiplas linhas, que ignorarão tudo que está dentro dos caracteres. Observe o exemplo abaixo:

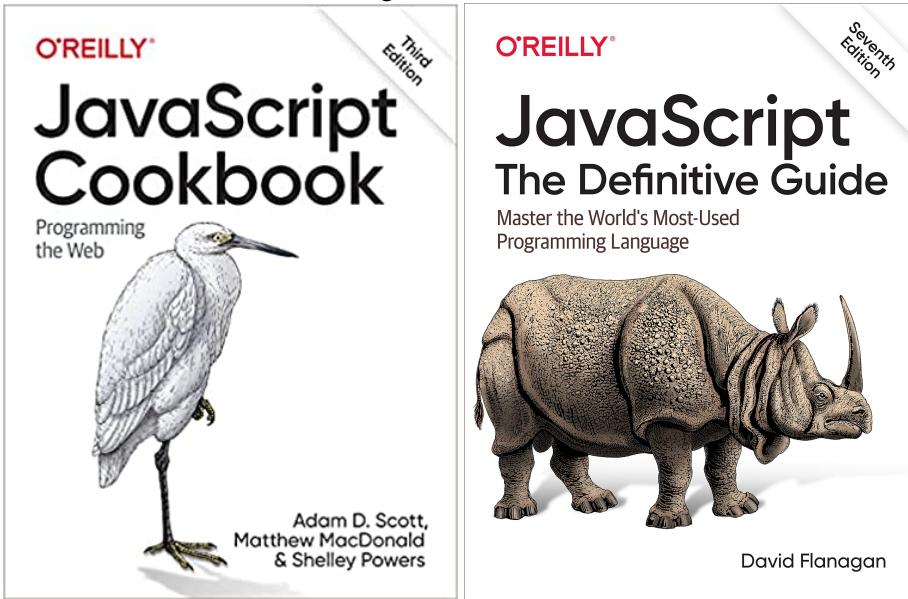
```

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, mas não pode existir um objeto que tenha mais de uma propriedade com o 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 são 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);
```

```

305
306     console.log("postorder traversal");
307     BST.postorder(root);
308     console.log("preorder traversal");
309     BST.preorder(root);
310
311

```

4.2.1 BST Prints

Abaixo se encontram imagens do código e do resultado, respectivamente.

```

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         // 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
47         // if the data is less than the node
48         // data move left of the tree
49         if(newNode.data < node.data)
50
51             // if left is null insert node here
52             if(node.left === null)
53                 node.left = newNode;
54             else
55
56                 // if left is not null recur until
57                 // null is found
58                 this.insertNode(node.left, newNode);
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 tree 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

```

Figura 4.1:

The screenshot shows a developer's environment with several browser tabs and code editors open. The tabs include 'JS BST.js', 'JS Bubble.js', 'JS Index.js', 'JS ConverterUtilities', 'JS Quick.js', and 'JS BST.js' (active). The code editor on the left contains the implementation of a removeNode function for a binary search tree. The code on the right contains the implementation of an insert function for a binary search tree. Both snippets are annotated with line numbers and comments explaining the logic. The background shows other tabs like 'Calculator.js', 'Index.js', 'Quick.js', and 'Utilities.js', and a sidebar with file navigation.

```
JS BST.js > ...
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 // if the root is null then tree is
195 // empty
196 if(node === null)
197 return null;
198
199 // Deleting node with two children
200 // minimum node of the right subtree
201 // is stored in aux
202 var aux = this.findMinNode(node.right);
203 node.data = aux.data;
204
205 node.right = this.removeNode(node.right, aux.data);
206
207 // search for a node with given data
208
209 // Performs inorder traversal of a tree
210 inorder(node)
211 {
212 if(node !== null)
213 {
214 this.inorder(node.left);
215 console.log(node.data);
216 this.inorder(node.right);
217 }
218
219 // Helper function
220 // getMinInode()
221 // getRootNode()
222 // inorder(node)
223 // preorder(node)
224 // postorder(node)
225 // search(node, data)
226
227 // create an object for the BinarySearchTree
228 var BST = new BinarySearchTree();
229
230 // Inserting nodes to the BinarySearchTree
231 BST.insert(15);
232 BST.insert(15);
233 BST.insert(25);
234 BST.insert(10);
235 BST.insert(7);
236 BST.insert(22);
237
238 // 15
239 // 10 25
240 // 7 13 22 27
241
242 // 15
243 // 10 25
244 // 7 13 22 27
245 // 9 17
246
247 var root = BST.getRootNode();
248
249 // prints 5 7 9 10 13 15 17 22 25 27
250 BST.inorder(root);
251
252 // Removing node with no children
253 BST.remove(5);
254
255
256 // Removing node with one child
257 BST.remove(7);
258
259 // 15
260 // 10 25
261 // 7 13 22 27
262
263 // 9 17
264
265
266
267
268 var root = BST.getRootNode();
269
270 // prints 7 9 10 13 15 17 22 25 27
271 BST.inorder(root);
272
273 // Removing node with one child
274 BST.remove(7);
275
276 // 15
277 // 10 25
278 // 7 13 22 27
279
280 // 9 17
```

Figura 4.2:

```
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
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:

4.3 Calculadora

O exemplo abaixo foi feito em JavaScript com HTML e CSS e interpretado pelo navegador.

```
1 | var valor1, valor2, operador;
2 | var readlineSync = require('readline-sync');
```

```

3  operador = readlineSync.question("Qual operacao deseja efetuar (+) (-)
4  (*) (/)? : \n");
5  valor1 = parseFloat(readlineSync.question("Insira o primeiro numero: \n
6  "));
7  valor2 = parseFloat(readlineSync.question("Insira o segundo numero: \n"
7));
8
9  function calcular(operator, value1, value2) {
10    if (operator == "+") {
11      return value1 + value2;
12    } else if
13      (operator == "-") {
14        return value1 - value2;
15    } else if
16      (operator == "*") {
17        return value1 * value2;
18    } else if
19      (operator == "/") {
20        return value1 / value2;
21    } else {
22      throw new Error('Operacao invalida');
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:

```

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: 8.444444444444445
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:

4.4 Implementação do QuickSort

O algoritmo QuickSort é feito da seguinte forma no JavaScript:

```

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(
20        greaterArray));
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, 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:

```

QuickSort > js Quick.js > ...
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(greaterArray));
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:

```

gabrielg@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:

4.5 Conversor de Temperatura

O código abaixo foi feito utilizando o módulo 'prompt-sync'.

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

4.5.1 Conversor de Temperatura Imagens

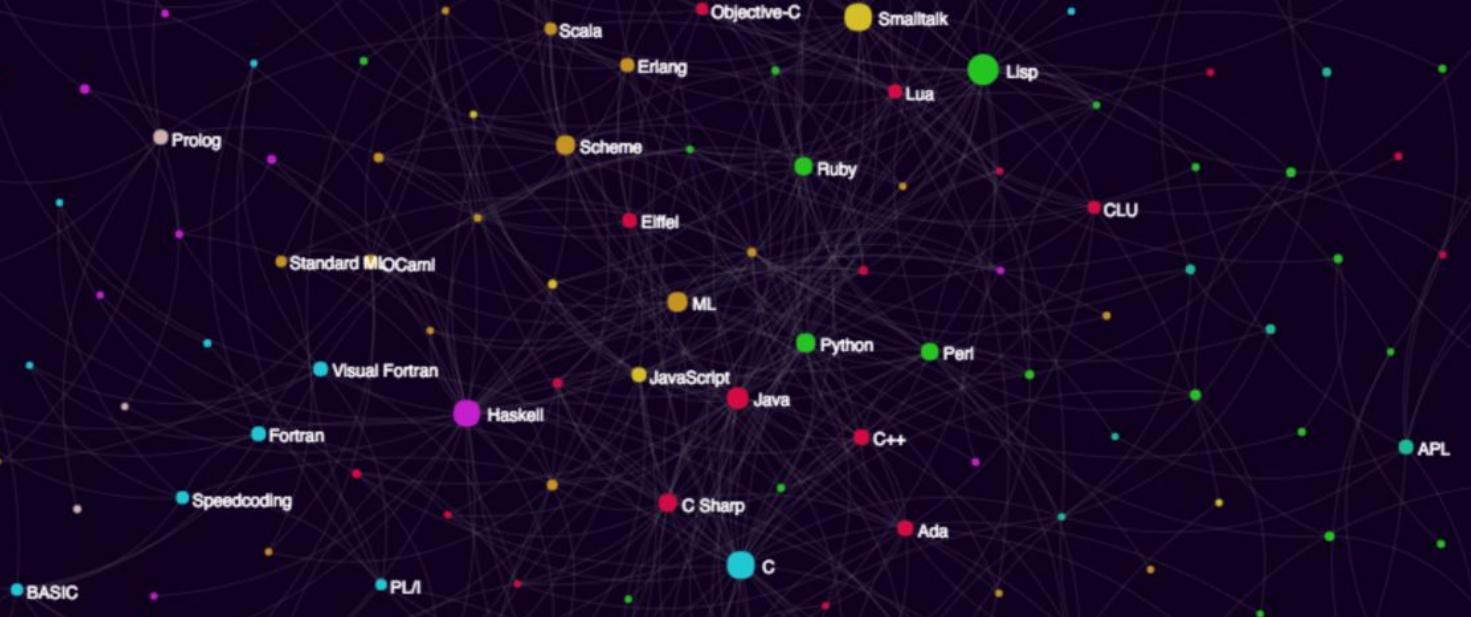
Código fonte e imagem do resultado:

```
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 -35.55555555555556 °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 NPM



Figura 5.2:

Um gerenciador de pacotes é uma ferramenta fundamental para o desenvolvimento de aplicações. Por ele é possível instalar diversas dependências necessárias para o funcionamento da aplicação e gerenciar os pacotes e suas versões que o programa irá utilizar. O NPM (Node Package Manager) é o gerenciador de pacotes mais utilizado para o JavaScript. Na imagem abaixo, utilizei o NPM para baixar um pacote chamado "express".

```
gabrielg@gabriel-desktop:~$ npm install express
npm WARN old lockfile
npm WARN old lockfile The package-lock.json file was created with an old version of npm,
npm WARN old lockfile so supplemental metadata must be fetched from the registry.
npm WARN old lockfile
npm WARN old lockfile This is a one-time fix-up, please be patient...
npm WARN old lockfile
npm WARN deprecated jQuery@1.7.4: This is deprecated. Please use 'jquery' (all lowercase).
added 1 package, removed 1 package, and audited 115 packages in 6s
11 packages are looking for funding
  run 'npm fund' for details
3 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run 'npm audit' for details.
gabrielg@gabriel-desktop:~$
```

Figura 5.3:

O NPM gera um arquivo chamado package.json. Este arquivo contém informações importantes do programa, como o nome, versão, arquivo inicial e suas dependências. Também é possível especificar certas condições nestes arquivos.

```
package.json
{
  "name": "t",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "William Bruno <wbrunom@gmail.com> (http://wbruno.com.br)",
  "license": "ISC"
}
```

Figura 5.4:

5.1.2 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. Abaixo, utilizei o comando nvm ls, que lista todas as versões do node instaladas no computador.

```
gabrielg@gabriel-desktop:~$ nvm ls
->      v12.17.0
        v16.13.0
          system
default -> 12.17 (> v12.17.0)
iojs -> N/A (default)
unstable -> N/A (default)
node -> stable (> v16.13.0) (default)
stable -> 16.13 (> v16.13.0) (default)
lts/* -> lts/gallium (> v16.13.0)
lts/argon -> v4.9.1 (> N/A)
lts/boron -> v6.17.1 (> N/A)
lts/carbon -> v8.17.0 (> N/A)
lts/dubnium -> v10.24.1 (> N/A)
lts/erbium -> v12.22.7 (> N/A)
lts/fermium -> v14.18.1 (> N/A)
lts/gallium -> v16.13.0
gabrielg@gabriel-desktop:~$
```

Figura 5.5:

5.2 IDEs

As Interfaces de Desenvolvimento Integrado, IDEs, são ferramentas importantes que auxiliam o programador a desenvolver aplicações fornecendo recursos, praticidade e dinamicidade. Imagine ter que escrever códigos enormes, acima de 200 linhas, num bloco de notas. Ou ter que navegar em arquivos e editá-los dentro de dezenas de diretórios e sub-diretórios sem nenhuma forma de visualizá-los. Seria uma tarefa extremamente árdua, e por isso, programar numa IDE poderosa permite a otimização do tempo de trabalho e fornece ganhos enormes de produtividade. Diversos recursos, como complemento de código e auto-importação de arquivos necessários, são apenas exemplos de formas que uma interface de desenvolvimento pode ajudar um programador.

5.2.1 Visual Studio Code

Nos dias atuais existem diversas versões de IDEs para as mais variadas linguagens de programação. Os exemplos vão desde o Netbeans e Eclipse para o Java ao 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 e o VS Code. Porém, atualmente este fato já não é mais realidade. Com o Visual Studio Code, que chamarei de VSC ou VS Code, programar em várias linguagens é fácil e dinâmico. 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. Abaixo estão as extensões de JavaScript mais utilizadas:

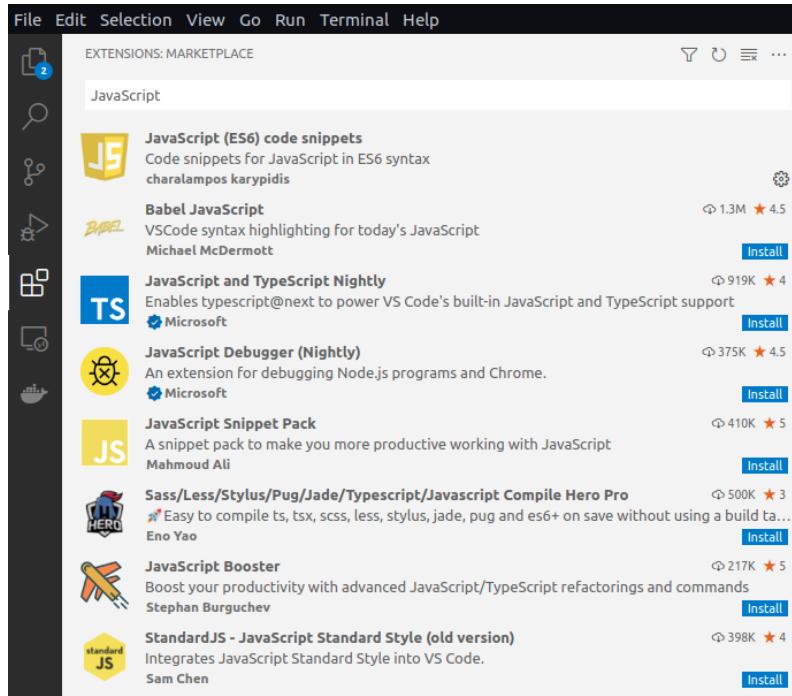


Figura 5.6:

Portanto, para programar em JavaScript, o VSCode é uma das melhores IDEs disponíveis. Além de permitir programar para web utilizando JS, HTML e CSS, também suporta outras linguagens e tem suporte ao GitHub.

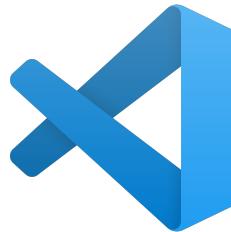


Figura 5.7:

5.2.2 Webstorm

Embora seja pago, o Webstorm é uma excelente escolha na hora de desenvolver em JavaScript e TypeScript. O programa é desenvolvido pela JetBrains, uma empresa bem estabelecida no mercado de IDEs. PyCharm, IntelliJ entre outros são programas extremamente confiáveis e práticos desenvolvidos pela JetBrains.



Figura 5.8:

5.2.3 Atom

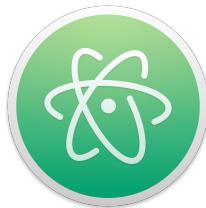


Figura 5.9:

O Atom é um programa leve, gratuito e prático de utilizar, assim como o VS Code. Tem integração com o GitHub e é ótimo para desenvolver em colaboração com outros desenvolvedores. É importante dizer que além disso, o Atom é open-source, assim como o VS Code.

```
File Edit View Selection Find Packages Help scale.fix.js CART.js Bengali_New.indx

1 //fixScale = function(doc) {
2
3     var addEvent = 'addEventListener',
4         type = 'gesturestart',
5         qsa = 'querySelectorAll',
6         scales = [1, 1],
7         meta = qsa in doc ? doc[qsa]('meta[name=viewport}') : [];
8
9     function fix() {
10        meta.content = 'width=device-width,minimum-scale=' + scales[0] + ',max-scale=' + scales[1];
11        doc.removeEventListener(type, fix, true);
12    }
13
14    if ((meta = meta[meta.length - 1]) && addEvent in doc) {
15        fix();
16        scales = [.25, 1.6];
17        doc[addEvent](type, fix, true);
18    }
19
20};

javascript/scale.fix.js 1:1
```

Figura 5.10:

5.2.4 JS Fiddle



Figura 5.11:

É importante mencionar também interpretadores e IDEs onlines que auxiliam no desenvolvimento de aplicações. Uma delas é o JS Fiddle. O JS Fiddle permite desenvolver, compartilhar e executar aplicações de forma similar à uma IDE padrão. Além disso, pela praticidade, pode ser uma alternativa para edição rápida em situações que configurar a interface de desenvolvimento não seja possível.

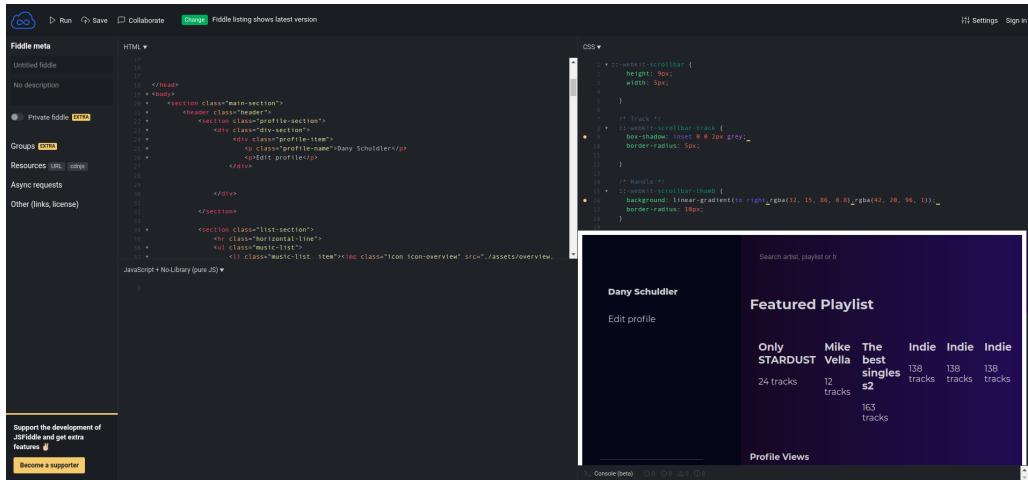


Figura 5.12:

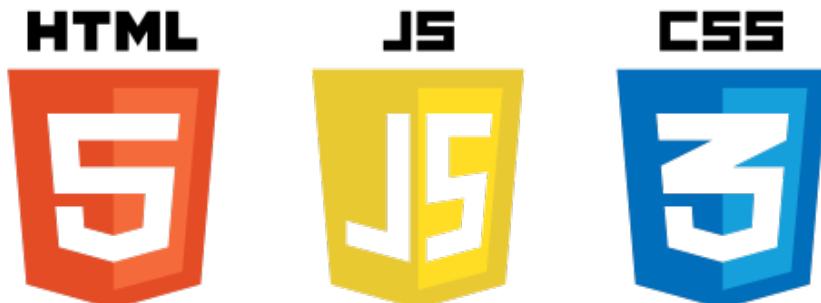


6. Conclusões

Fazer este trabalho ao longo do semestre com certeza não foi uma tarefa fácil. Foram longas noites pensando no que seria escrito, aprendendo a usar o LaTeX e revisando o conteúdo produzido. A pesquisa e elaboração do trabalho foram árduas e, mesmo com certo grau de experiência na linguagem, encontrei muitos conceitos que até então eram desconhecidos. Os livros da linguagem com certeza ajudaram muito, porém nem sempre foram a única fonte de referências utilizada. Como diria Richard Feynman, se você não consegue explicar algo em termos simples, você não entende do assunto. Logo, para explicar algo, primeiro é preciso entender. Nesse sentido, para reforçar meu conhecimento foram assistidos tutoriais em vídeo da linguagem, além da leitura da documentação oficial da linguagem. Ademais, aprender esses conceitos foi a parte mais difícil do trabalho. Contudo, foi a busca incessante de referências e conhecimentos que fizeram a experiência de desenvolver esse trabalho ser tão gratificante. Por isso, explicar algo é sempre a melhor forma de aprender, e sendo assim, escrever este livro foi uma excelente forma de aprendizado.

O trabalho foi desenvolvido com o objetivo de explicar e introduzir a linguagem de forma simples e resumida. Portanto, não foi possível explicar detalhadamente alguns conceitos como os pontos flutuantes binários e os erros de arredondamento.

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