



Introdução à Linguagem JavaScript

Paradigmas de Linguagens de Programação

Gabriel Marques de Amaral Gravina

Ausberto S. Castro Vera

4 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



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	10
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	20
4.2	Árvore de Busca Binária	20
4.2.1	BST Prints	26
4.3	Calculadora	26
4.3.1	Prints Calculadora	30
4.4	Implementação do QuickSort	33
4.4.1	Prints QuickSort	35
4.5	BubbleSort	35
4.5.1	BubbleSort Prints	36
	Bibliografia	37
	Index	39



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 de 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, 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 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. De acordo com [Fla20].

1.2.1 NodeJS

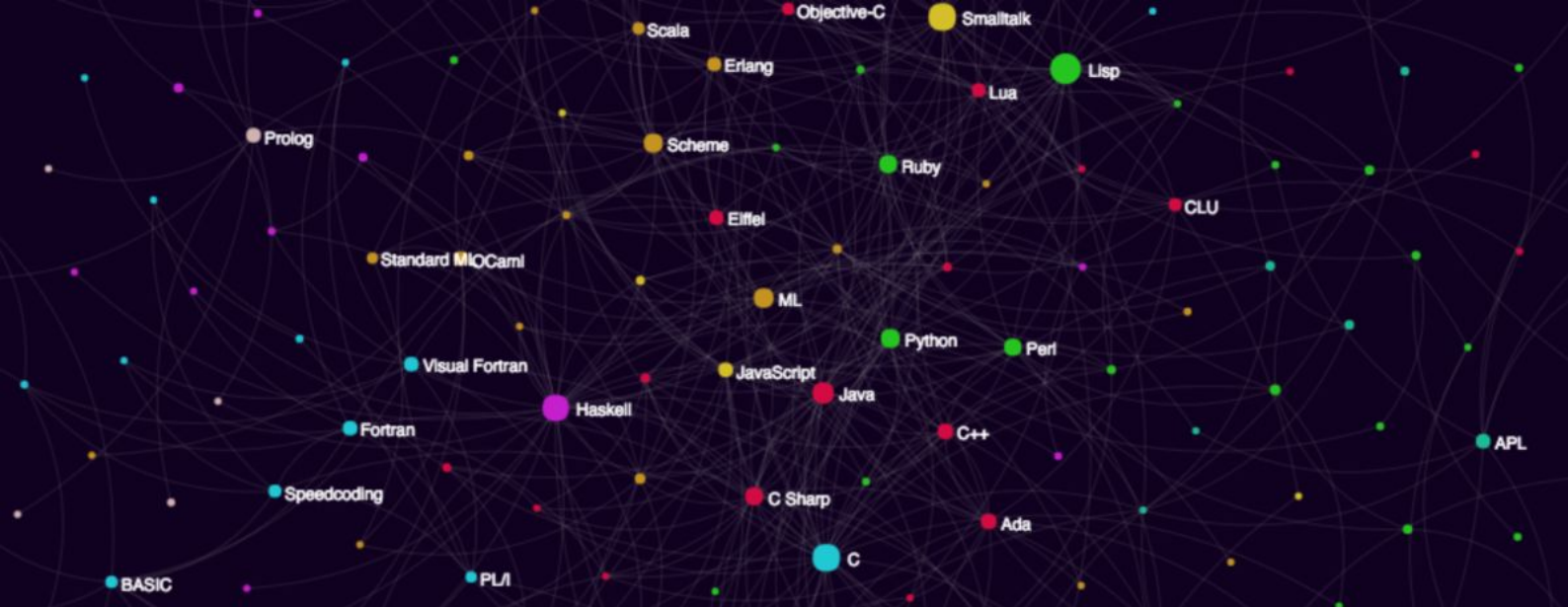
A linguagem foi criada para ser utilizada em navegadores da web, e esse segue sendo seu ambiente mais comum de execução até hoje. Enfim, o ambiente do navegador permite a linguagem obter a entrada de usuários e fazer requests HTTP. Porém, em 2010 outro ambiente foi criado para executar código em JavaScript. 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 instancias. 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 é 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.

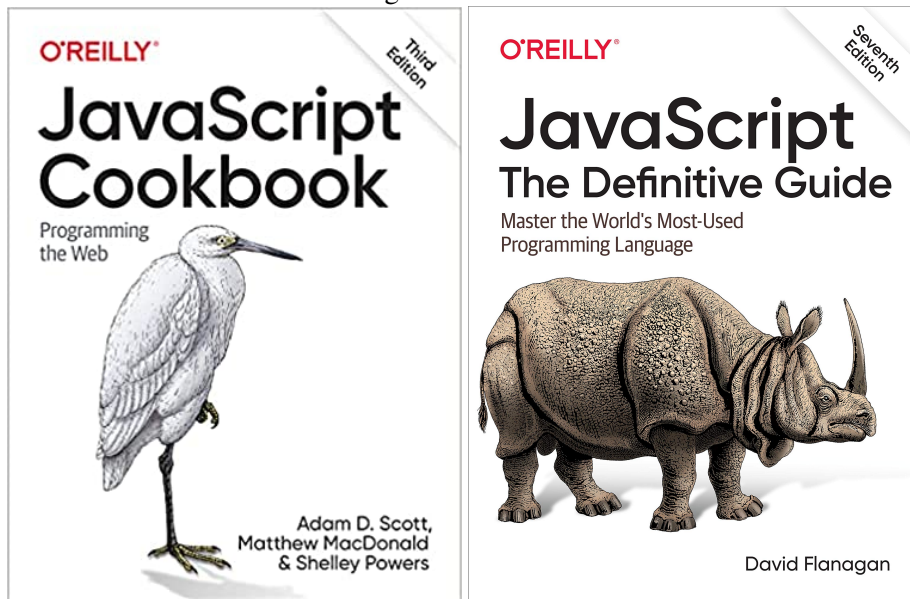
2.1 Estrutura Léxica

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    0 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

```

1      ++                //incremento
2      --                //decremento
3      !                 //inverte valores em booleano
4      ==                //testa igualdade
5      !=                //testa desigualdade
6      //testa por igualdade estrita, ou seja, o tipo tambem tem que ser o
mesmo
7      ===
8      ||                //OR
9      &&                //AND
10     =                  //Atribui um valor a uma variavel
11     *=, /=, %=, +=...  //Faz uma atribuicao e um calculo
12     //Menor que, maior que, menor ou igual que, maior ou igual que
13     <, >, <=, >=
14

```

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" seguida de seu nome. 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     '0 numero "8" e par'
7
8     mensagem01a = "Ola, seja bem vindo"
9     //Printa o conteudo da variavel no console do navegador
10    console.log(mensagem01a)
11
12    /*compara o valor da variavel e retorna true ou false.
13    No caso, retornara true*/
14    a = "01a"
15    a == "01a"
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 e um numero ou nao.
2   parseInt()        //Recebe o conteudo de uma string e converte para.
   inteiro
3   parseFloat()      //Recebe uma string e a converte para float.
4   eval()            //Avalia codigo representado por uma string.
5   isFinite()        //Verifica se um numero e 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 progrma 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" atinga 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 == Horrroso){  
2     console.log("0 presidente e horroroso")  
3 }  
4 //0 programa printara a mensagem acima indefinidamente  
5
```

2.4.3 For

Geralmente, os laços que utilizam o "for" são mais simples de serem lidos. 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     //0 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 //0 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 JavaS

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 junto à 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, porem, como essa propriedade foi
    deletada, o console ira 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.

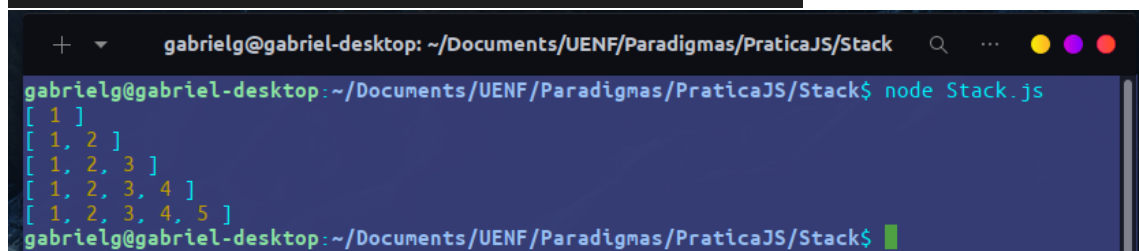
4.1 Pilha Implementação

```
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]  
17
```

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

4.1.1 Prints Pilha

```
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]
```



```
+ ▾ gabrielg@gabriel-desktop: ~/Documents/UENF/Paradigmas/PraticaJS/Stack  
gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/Stack$ node Stack.js  
[ 1 ]  
[ 1, 2 ]  
[ 1, 2, 3 ]  
[ 1, 2, 3, 4 ]  
[ 1, 2, 3, 4, 5 ]  
gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/Stack$
```

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.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
260     //      15
261     //     / \
262     //    10 25
263     //   / \ / \

```

```
264 //    7 13 22 27
265 //    \ /
266 //    9 17
267
268
269 var root = BST.getRootNode();
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.getRootNode();
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.getRootNode();
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

```

gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS$ node BST.js
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$

```

4.3 Calculadora

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

```

1  class Calculator {
2  constructor(previousOperandTextElement, currentOperandTextElement) {
3  this.previousOperandTextElement = previousOperandTextElement
4  this.currentOperandTextElement = currentOperandTextElement
5  this.clear()
6  }

```



```
7
8   clear() {
9     this.currentOperand = ''
10    this.previousOperand = ''
11    this.operation = undefined
12  }
13
14  delete() {
15    this.currentOperand = this.currentOperand.toString().slice(0, -1)
16  }
17
18  appendNumber(number) {
19    if (number === '.' && this.currentOperand.includes('.')) return
20    this.currentOperand = this.currentOperand.toString() + number.toString()
21  }
22
23  chooseOperation(operation) {
24    if (this.currentOperand === '') return
25    if (this.previousOperand !== '') {
26      this.compute()
27    }
28    this.operation = operation
29    this.previousOperand = this.currentOperand
30    this.currentOperand = ''
31  }
32
33  compute() {
34    let computation
35    const prev = parseFloat(this.previousOperand)
36    const current = parseFloat(this.currentOperand)
37    if (isNaN(prev) || isNaN(current)) return
38    switch (this.operation) {
39      case '+':
40        computation = prev + current
41        break
42      case '-':
43        computation = prev - current
44        break
45      case '*':
46        computation = prev * current
47        break
48      case '/':
49        computation = prev / current
50        break
51      default:
52        return
53    }
54    this.currentOperand = computation
55    this.operation = undefined
56    this.previousOperand = ''
57  }
58
59  getDisplayNumber(number) {
60    const stringNumber = number.toString()
61    const integerDigits = parseFloat(stringNumber.split('.')[0])
62    const decimalDigits = stringNumber.split('.')[1]
63    let integerDisplay
64    if (isNaN(integerDigits)) {
65      integerDisplay = ''
66    } else {
```

```

67     integerDisplay = integerDigits.toLocaleString('en', {
68         maximumFractionDigits: 0 })
69     }
70     if (decimalDigits !== null) {
71         return `${integerDisplay}.${decimalDigits}`
72     } else {
73         return integerDisplay
74     }
75
76     updateDisplay() {
77         this.currentOperandTextElement.innerText =
78         this.getDisplayNumber(this.currentOperand)
79         if (this.operation !== null) {
80             this.previousOperandTextElement.innerText =
81             `${this.getDisplayNumber(this.previousOperand)} ${this.operation}`
82         } else {
83             this.previousOperandTextElement.innerText = ''
84         }
85     }
86
87
88
89     const numberButtons = document.querySelectorAll('[data-number]')
90     const operationButtons = document.querySelectorAll('[data-operation]')
91     const equalsButton = document.querySelector('[data-equals]')
92     const deleteButton = document.querySelector('[data-delete]')
93     const allClearButton = document.querySelector('[data-all-clear]')
94     const previousOperandTextElement = document.querySelector('[data-
95     previous-operand]')
96     const currentOperandTextElement = document.querySelector('[data-current
97     -operand]')
98
99     const calculator = new Calculator(previousOperandTextElement,
100     currentOperandTextElement)
101
102     numberButtons.forEach(button => {
103         button.addEventListener('click', () => {
104             calculator.appendNumber(button.innerText)
105             calculator.updateDisplay()
106         })
107     })
108
109     operationButtons.forEach(button => {
110         button.addEventListener('click', () => {
111             calculator.chooseOperation(button.innerText)
112             calculator.updateDisplay()
113         })
114     })
115
116     equalsButton.addEventListener('click', button => {
117         calculator.compute()
118         calculator.updateDisplay()
119     })
120
121     allClearButton.addEventListener('click', button => {
122         calculator.clear()
123         calculator.updateDisplay()
124     })
125
126     deleteButton.addEventListener('click', button => {

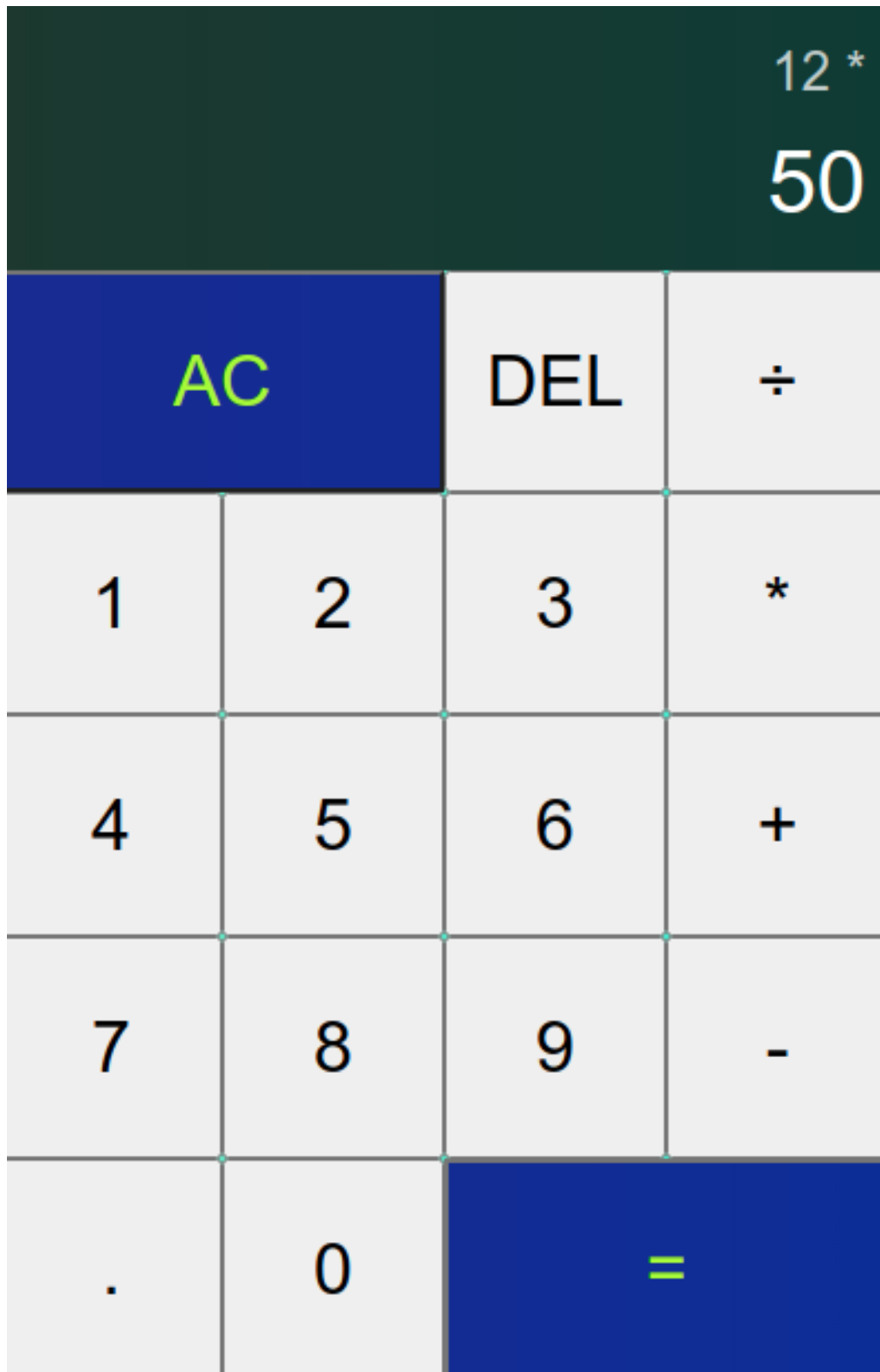
```

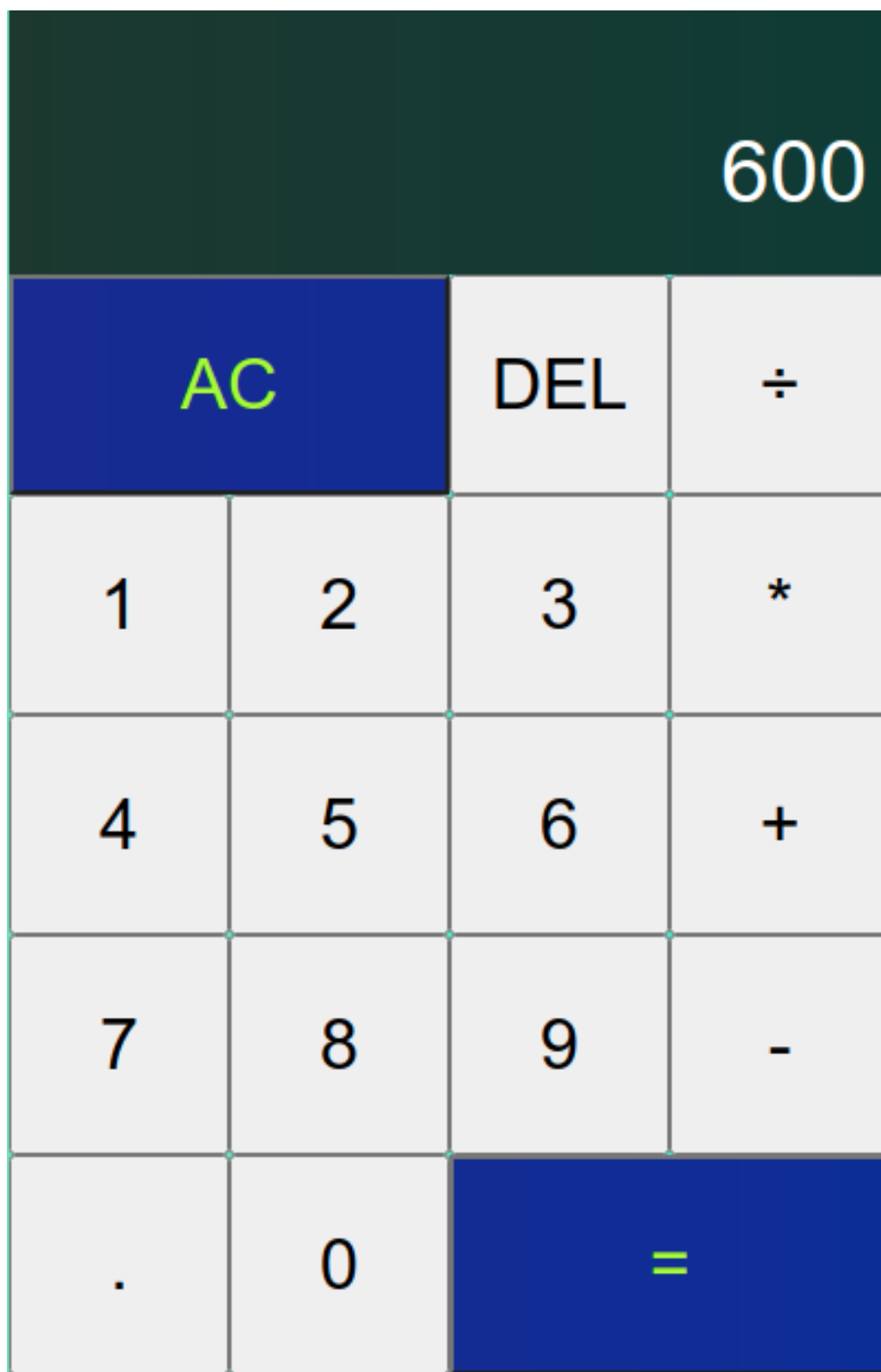
```
124     calculator.delete()  
125     calculator.updateDisplay()  
126     })  
127  
128
```

4.3.1 Prints Calculadora

```
1 class Calculator {
2   constructor(previousOperandTextElement, currentOperandTextElement) {
3     this.previousOperandTextElement = previousOperandTextElement
4     this.currentOperandTextElement = currentOperandTextElement
5     this.clear()
6   }
7
8   clear() {
9     this.currentOperand = ''
10    this.previousOperand = ''
11    this.operation = undefined
12  }
13
14  delete() {
15    this.currentOperand = this.currentOperand.toString().slice(0, -1)
16  }
17
18  appendNumber(number) {
19    if (number === '.' && this.currentOperand.includes('.')) return
20    this.currentOperand = this.currentOperand.toString() + number.toString()
21  }
22
23  chooseOperation(operation) {
24    if (this.currentOperand === '') return
25    if (this.previousOperand !== '') {
26      this.compute()
27    }
28    this.operation = operation
29    this.previousOperand = this.currentOperand
30    this.currentOperand = ''
31  }
32
33  compute() {
34    let computation
35    const prev = parseFloat(this.previousOperand)
36    const current = parseFloat(this.currentOperand)
37    if (isNaN(prev) || isNaN(current)) return
38    switch (this.operation) {
39      case '+':
40        computation = prev + current
41        break
42      case '-':
43        computation = prev - current
44        break
45      case '*':
46        computation = prev * current
47        break
48      case '/':
49        computation = prev / current
50        break
51      default:
52        return
53    }
54    this.currentOperand = computation
55    this.operation = undefined
56    this.previousOperand = ''
57  }
58
59  getDisplayNumber(number) {
60    const stringNumber = number.toString()
61    const integerDigits = parseFloat(stringNumber.split('.')[0])
62    const decimalDigits = stringNumber.split('.')[1]
63    let integerDisplay
64    if (isNaN(integerDigits)) {
65      integerDisplay = ''
66    } else {
67      integerDisplay = integerDigits.toLocaleString('en', { maximumFractionDigits: 0 })
68    }
69    if (decimalDigits != null) {
70      return `${integerDisplay}.${decimalDigits}`
71    } else {
72      return integerDisplay
73    }
74  }
75 }
```

```
72     return integerDisplay
73   }
74 }
75
76 updateDisplay() {
77   this.currentOperandTextElement.innerText =
78     this.getDisplayNumber(this.currentOperand)
79   if (this.operation !== null) {
80     this.previousOperandTextElement.innerText =
81       `${this.getDisplayNumber(this.previousOperand)} ${this.operation}`
82   } else {
83     this.previousOperandTextElement.innerText = ''
84   }
85 }
86 }
87
88
89 const numberButtons = document.querySelectorAll('[data-number]')
90 const operationButtons = document.querySelectorAll('[data-operation]')
91 const equalsButton = document.querySelector('[data-equals]')
92 const deleteButton = document.querySelector('[data-delete]')
93 const allClearButton = document.querySelector('[data-all-clear]')
94 const previousOperandTextElement = document.querySelector('[data-previous-operand]')
95 const currentOperandTextElement = document.querySelector('[data-current-operand]')
96
97 const calculator = new Calculator(previousOperandTextElement, currentOperandTextElement)
98
99 numberButtons.forEach(button => {
100   button.addEventListener('click', () => {
101     calculator.appendNumber(button.innerText)
102     calculator.updateDisplay()
103   })
104 })
105
106 operationButtons.forEach(button => {
107   button.addEventListener('click', () => {
108     calculator.chooseOperation(button.innerText)
109     calculator.updateDisplay()
110   })
111 })
112
113 equalsButton.addEventListener('click', button => {
114   calculator.compute()
115   calculator.updateDisplay()
116 })
117
118 allClearButton.addEventListener('click', button => {
119   calculator.clear()
120   calculator.updateDisplay()
121 })
122
123 deleteButton.addEventListener('click', button => {
124   calculator.delete()
125   calculator.updateDisplay()
126 })
127 }
```





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(
    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);
44
45 /* Output -
46 Unsorted array:
47 [ 63, 95, 63, 26, 76, 19, 65, 8, 63, 26 ]
48 Sorted array:
49 [ 8, 19, 26, 26, 63, 63, 65, 76, 95 ]
50 [Finished in 0.1s]
51 */
```

O código foi retirado de: <https://javascript.plainenglish.io/>

quick-sort-algorithm-in-javascript-5cf5ab7d251b

4.4.1 Prints QuickSort

```

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 // ===== Testing Quick sort algorithm =====
23 // Returns a random integer between min (inclusive) and max (inclusive). Using Math.round() will give a non-uniform distribution, which we don't want in this case.
24 function getRandomInt(min, max) {
25   return Math.floor(Math.random() * (max - min + 1)) + min;
26 }
27 // By setting 1, we make the maximum inclusive (the minimum is inclusive anyway). Because, the Math.random() function returns a floating point,
28 // pseudo-random number in the range from 0 inclusive up to but not including 1
29 var arr = [];
30 for (var i = 0; i < 10; i++) { //Initialize a random integer unsorted array
31   arr.push(getRandomInt(1, 100));
32 }
33 console.log("Unsorted array: ");
34 console.log(arr); //printing unsorted array
35
36 arr = quickSortBasic(arr, 0, arr.length - 1);
37 console.log("Sorted array: ");
38 console.log(arr);
39
40 /* Output -
41 Unsorted array:
42 [ 63, 95, 63, 26, 76, 19, 65, 8, 63, 26 ]
43 Sorted array:
44 [ 8, 19, 26, 26, 63, 63, 63, 65, 76, 95 ]
45 [Finished in 0.1s]
46 */

```

Código Fonte:

```

gabrielg@gabriel-desktop: ~/Documents/UENF/Paradigmas/PraticaJS/QuickSort$ node Quick.js
Unsorted array:
[
  99, 91, 19, 56, 100,
  78, 56, 53, 79, 45
]
Sorted array:
[
  19, 45, 53, 56, 56,
  78, 79, 91, 99, 100
]
gabrielg@gabriel-desktop: ~/Documents/UENF/Paradigmas/PraticaJS/QuickSort$

```

Resultado:

4.5 BubbleSort

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

```

1 function bubbleSort(items) {
2   var length = items.length;
3   for (var i = 0; i < length; i++) {
4     for (var j = 0; j < (length - i - 1); j++) {
5       if (items[j] > items[j+1]) {
6         var tmp = items[j];
7         items[j] = items[j+1];
8         items[j+1] = tmp;
9       }
10    }
11  }
12 }
13
14 var arr = [5, 4, 3, 2, 1];
15 bubbleSort(arr);
16
17 console.log(arr);
18

```

4.5.1 BubbleSort Prints

```
1  function bubbleSort(items) {
2      var length = items.length;
3      for (var i = 0; i < length; i++) {
4          for (var j = 0; j < (length - i - 1); j++) {
5              if(items[j] > items[j+1]) {
6                  var tmp = items[j];
7                  items[j] = items[j+1];
8                  items[j+1] = tmp;
9              }
10         }
11     }
12 }
13
14 var arr = [5, 4, 3, 2, 1];
15 bubbleSort(arr);
16
17 console.log(arr);
```

gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/BubbleSort\$ node Bubble.js
[1, 2, 3, 4, 5]
gabrielg@gabriel-desktop:~/Documents/UENF/Paradigmas/PraticaJS/BubbleSort\$



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 9.

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) <ul style="list-style-type: none"> • Sintaxe (variáveis, constantes, comandos, operações, etc.) • Usos e áreas de Aplicação da Linguagem 	
Cada elemento da linguagem (definição) com exemplos (Máximo: 02 pontos) <ul style="list-style-type: none"> • Exemplos com fonte diferenciada (Courier , 10 pts, azul) 	
Mínimo 5 exemplos completos - Aplicações (Máximo : 2 pontos) <ul style="list-style-type: none"> • Uso de rotinas-funções-procedimentos, E/S formatadas • Menu de operações, programas gráficos, matrizes, aplicações 	
Ferramentas (compiladores, interpretadores, etc.) (Máximo : 2 pontos) <ul style="list-style-type: none"> • 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) <ul style="list-style-type: none"> • Conteúdo, Historia, Seções, gráficos, exemplos, conclusões, bibliografia 	
Uso de Bibliografia (Máximo: 01 ponto) <ul style="list-style-type: none"> • 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