UNIVERSIDADE ESTADUAL DO PIAUÍ
CAMPUS TORQUATO NETO
CENTRO DE CIÊNCIAS DA NATUREZA
CURSO DE LICENCIATURA EM MATEMÁTICA

Gabriel Guimarães Cruz

# **Matemática e JavaScript**: Programação Para Ensino Básico e Técnico

Teresina
2024

Gabriel Guimarães Cruz

# **Matemática e JavaScript**: Programação Para Ensino Básico e Técnico

Trabalho de Conclusão de Curso do Curso de Licenciatura em Matemática do Campus Torquato Neto da Universidade Estadual do Piauí para a obtenção do título de Licenciatura em Matemática.
Orientador: Prof. Dr. José Danuso Rocha de Oliveira

Teresina
2024

C955m Cruz, Gabriel Guimarães.
Matemática e JavaScript - programação para ensino básico e técnico / Gabriel Guimaraes Cruz. - 2024.
70 f.: il.

Monografia (graduação) - Universidade Estadual do Piauí-UESPI, Licenciatura em Matemática, Campus Poeta Torquato Neto, Teresina-PI, 2024.
"Orientador: Prof. Dr. José Danuso Rocha de Oliveira".

1. Matemática básica. 2. Programação. 3. JavaScript. I. Oliveira, José Danuso Rocha de . II. Título.

CDD 510.07

Ficha elaborada pelo Serviço de Catalogação da Biblioteca da UESPI
Nayla Kedma de Carvalho Santos (Bibliotecário) CRB-3ª/1188

Documento assinado digitalmente
GABRIEL GUIMARAES CRUZ
Data: 19/08/2025 22:15:43-0300
Verifique em https://validar.iti.gov.br

Gabriel Guimarães Cruz

**Matemática e JavaScript**: Programação Para Ensino Básico e Técnico

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Licenciatura em Matemática" e aprovado em sua forma final pelo Curso de Licenciatura em Matemática.

Teresina, 19 de junho de 2024.

Documento assinado digitalmente
ALESSANDRO WILK SILVA ALMEIDA
Data: 19/08/2025 21:39:25-0300
Verifique em https://validar.iti.gov.br

---

Prof. Dr. Alessandro Wilk Silva Almeida
Coordenador do Curso

**Banca Examinadora:**

Documento assinado digitalmente
JOSE DANUSO ROCHA DE OLIVEIRA
Data: 13/08/2025 17:57:39-0300
Verifique em https://validar.iti.gov.br

---

Prof. Dr. José Danuso Rocha de Oliveira
Orientador

Documento assinado digitalmente
ALESSANDRO WILK SILVA ALMEIDA
Data: 19/08/2025 21:42:29-0300
Verifique em https://validar.iti.gov.br

---

Prof. Dr. Alessandro Wilk Silva Almeida
Avaliador
Universidade Estadual do Piauí

Documento assinado digitalmente
RAFAEL ROCHA DE FARIAS
Data: 14/08/2025 15:12:24-0300
Verifique em https://validar.iti.gov.br

---

Prof. Dr. Rafael Rocha Farias
Avaliador
Universidade Federal do Ceará

Este trabalho é dedicado à minha família e colegas, pilares da minha jornada, e a todos que encontram na matemática caminhos para o saber.

## AGRADECIMENTOS

Expresso minha sincera gratidão a todos que me apoiaram e contribuíram para a execução deste trabalho. Agradeço especialmente aos meus professores pela orientação precisa, em particular ao orientador Prof. Dr. José Danuso Rocha de Oliveira, que pacientemente me ajudou e enriqueceu o projeto. Meus sinceros agradecimentos aos meus colegas de turma pela troca de conhecimentos, e à minha família e amigos pelo suporte constante, especialmente ao meu pai, José Manoel Pereira da Cruz, cuja ajuda foi fundamental para que eu chegasse até aqui.

Um agradecimento especial é direcionado aos profissionais que dedicaram seu tempo e expertise para enriquecer este estudo com suas valiosas contribuições.

Agradeço também a Deus, pela fé e força que me guiaram ao longo desta jornada acadêmica.

*"A matemática*
*é a rainha das ciências."*
*(Friedrich Gauss, século XIX)*

## RESUMO

Este Trabalho de Conclusão de Curso explora a interseção entre a matemática e a programação com JavaScript, visando fornecer uma base sólida para o ensino básico e técnico. O objetivo é demonstrar como o uso do JavaScript pode facilitar a compreensão de conceitos matemáticos, tornando o aprendizado mais interativo e acessível. A pesquisa adota uma abordagem prática, com a elaboração de exemplos e exercícios que aplicam a teoria matemática em programas de JavaScript. As discussões destacam a importância da programação no ensino contemporâneo e como ela pode ser integrada ao currículo escolar para melhorar o desempenho dos alunos em matemática. Para alcançar esses objetivos, o estudo desenvolve uma série de aplicações práticas que ilustram conceitos fundamentais da matemática, como funções, álgebra, trigonometria e estatística, através da implementação em JavaScript. São apresentados também gráficos e visualizações interativas que ajudam a solidificar a compreensão teórica dos estudantes, permitindo uma visualização concreta dos problemas matemáticos. Além disso, a pesquisa analisa o impacto da utilização de ferramentas digitais no engajamento dos alunos e como estas podem contribuir para um aprendizado mais dinâmico e envolvente. A metodologia inclui a criação de um ambiente de programação acessível, com instruções detalhadas sobre a configuração e uso de ferramentas como Visual Studio Code e Node.js, facilitando a integração de JavaScript nas práticas pedagógicas. O estudo também considera as diretrizes da Base Nacional Comum Curricular (BNCC) para garantir a relevância e aplicabilidade dos conteúdos abordados no contexto educacional brasileiro. O uso do JavaScript como ferramenta pedagógica enriquece o processo de ensino-aprendizagem, promovendo uma maior interação dos estudantes com os conteúdos matemáticos e também prepara os alunos para o mercado de trabalho, onde habilidades em programação são cada vez mais valorizadas. Este trabalho destaca a importância de inovar nas estratégias de ensino, utilizando a tecnologia para tornar o aprendizado mais eficaz e atrativo.



**Palavras-chave**: Matemática. Programação. JavaScript. Ensino Básico. Ensino Técnico.

## ABSTRACT

This Thesis explores the intersection between mathematics and programming with JavaScript, aiming to provide a solid foundation for basic and technical education. The objective is to demonstrate how the use of JavaScript can facilitate the understanding of mathematical concepts, making learning more interactive and accessible. The research adopts a practical approach, with the development of examples and exercises that apply mathematical theory in JavaScript programs. The discussions highlight the importance of programming in contemporary education and how it can be integrated into the school curriculum to improve students' performance in mathematics. To achieve these objectives, the study develops a series of practical applications that illustrate fundamental mathematical concepts, such as functions, algebra, trigonometry, and statistics, through JavaScript implementation. Interactive graphs and visualizations are also presented to help solidify students' theoretical understanding, allowing for a concrete visualization of mathematical problems. Additionally, the research analyzes the impact of using digital tools on student engagement and how these can contribute to more dynamic and engaging learning. The methodology includes the creation of an accessible programming environment, with detailed instructions on configuring and using tools like Visual Studio Code and Node.js, facilitating the integration of JavaScript into pedagogical practices. The study also considers the guidelines of the Brazilian National Common Core (Base Nacional Comum Curricular - BNCC) to ensure the relevance and applicability of the content addressed in the Brazilian educational context. The use of JavaScript as a pedagogical tool enriches the teaching-learning process, promoting greater student interaction with mathematical content and also preparing students for the job market, where programming skills are increasingly valued. This work highlights the importance of innovating teaching strategies, using technology to make learning more effective and attractive.



**Keywords**: Mathematics. Programming. JavaScript. Basic Education. Technical Education.

## LISTA DE FIGURAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | *Application Programming Interface* (Interface de Programação de Aplicações) |
| BNCC | Base Nacional Comum Curricular |
| CSS | *Cascading Style Sheets* (Folha de Estilo em Cascata) |
| ECMA | *European Computer Manufacturers Association* (Associação Europeia de Fabricantes de Computadores) |
| HTML | *HyperText Markup Language* (Linguagem de Marcação de Hipertexto) |
| MDC | Máximo Divisor Comum |
| MMC | Mínimo Múltiplo Comum |
| TI | Tecnologia da informação |

## SUMÁRIO

# 1 INTRODUÇÃO

A integração entre a matemática e a programação tem se mostrado uma ferramenta poderosa para o ensino e a aprendizagem, especialmente em um contexto onde a tecnologia se torna cada vez mais presente nas salas de aula. Este Trabalho de Conclusão de Curso tem como foco principal explorar essa interseção utilizando a linguagem de programação JavaScript. A intenção é discutir teoricamente como a programação pode facilitar a compreensão de conceitos matemáticos, tornando o aprendizado mais interativo e acessível.

A escolha do JavaScript como objeto de estudo não é arbitrária. Essa linguagem é amplamente utilizada na web e possui uma curva de aprendizado que a torna acessível para iniciantes. Além disso, o JavaScript permite a criação de aplicações interativas que podem ser executadas diretamente no navegador, eliminando a necessidade de configuração complexa de ambiente. Com isso, professores e alunos podem focar mais na aplicação dos conceitos matemáticos e menos nas dificuldades técnicas que outras linguagens de programação poderiam apresentar.

A metodologia adotada neste trabalho envolve a elaboração de exemplos práticos e exercícios que aplicam teorias matemáticas em programas escritos em JavaScript. Essa abordagem prática visa não apenas ilustrar os conceitos, mas também permitir que os alunos experimentem e aprendam fazendo. A prática com programação pode ajudar a solidificar o entendimento de conceitos matemáticos abstratos, ao ver como eles são aplicados em soluções computacionais reais.

As discussões presentes neste trabalho destacam a importância da programação no ensino contemporâneo e como ela pode ser integrada ao currículo escolar para melhorar o desempenho dos alunos em matemática. A introdução de conceitos de programação desde cedo no ambiente escolar pode não apenas aumentar o interesse dos alunos pela matemática, mas também prepará-los para um mundo onde a alfabetização digital é essencial. O desenvolvimento do pensamento lógico e a capacidade de resolver problemas complexos são habilidades que os alunos podem levar para além da sala de aula.

A ideia deste trabalho é mostrar que o uso do JavaScript como ferramenta pedagógica pode enriquecer significativamente o processo de ensino-aprendizagem. A maior interação dos estudantes com os conteúdos matemáticos, proporcionada pela programação, pode levar a um aprendizado mais profundo e duradouro. A proposta é que, ao final deste estudo, os educadores tenham um material prático para incorporar a programação no ensino da matemática, transformando a maneira como esses conceitos são ensinados e aprendidos.

# 2 ALGORITMOS E LINGUAGEM DE PROGRAMAÇÃO

## 2.1 CONCEITOS BÁSICOS DE ALGORITMOS

Em geral, um algoritmo é uma série de comandos ordenados que visam cumprir um objetivo determinado. Assim como em tarefas do dia a dia, na computação, um algoritmo é um conjunto de instruções claras e exatas seguidas meticulosamente pelo computador para executar uma função específica. (Cormen, 2014).

## 2.2 O QUE É LÓGICA DE PROGRAMAÇÃO

A lógica de programação é um conjunto de técnicas e práticas utilizadas para resolver problemas por meio de algoritmos e estruturar programas de computador de maneira lógica e eficiente. Ela é fundamental para o desenvolvimento de sistemas e aplicativos, pois permite que os programadores organizem suas ideias e traduzam-nas em instruções que o computador possa executar.

Lembrando que a programação começa com a compreensão dos conceitos básicos, como variáveis, entrada, processamento e saída de dados. É essencial entender como as estruturas de um programa devem estar organizadas e como usar a lógica para montar essas estruturas corretamente (Iepsen, 2022).

## 2.3 INTRODUÇÃO BÁSICA AO JAVASCRIPT

A Netscape Communications Corporation, em colaboração com a Sun Microsystems, desenvolveu o **JavaScript**, que foi introduzido ao mundo em 1995. Durante esse período, o Netscape Navigator era o navegador mais popular. No entanto, surgiu uma época de disputas entre diferentes navegadores, onde o JavaScript tinha comportamentos inconsistentes entre eles. Para resolver essa questão, em 1996, a Netscape transferiu a gestão do JavaScript para a *European Computer Manufacturers Association* (Associação Europeia de Fabricantes de Computadores) (ECMA), que é focada na padronização tecnológica. Em 1997, a ECMA publicou a primeira versão padronizada da linguagem. É por isso que o JavaScript é também conhecido como ECMAScript, e suas atualizações são identificadas com esse termo, como ECMAScript 6, ECMAScript 7, ECMAScript 2022, entre outras. No *site* `https://ecma-international.org/` é possível verificar notícias e atualizações da linguagem (Iepsen, 2022).

A linguagem JavaScript desempenha um papel crucial na criação de sites, trabalhando em conjunto com *HyperText Markup Language* (Linguagem de Marcação de Hipertexto) (HTML) e *Cascading Style Sheets* (Folha de Estilo em Cascata) (CSS). Enquanto o HTML é responsável por estruturar e dar significado ao conteúdo de um site, o CSS é empregado para estilizar e formatar a aparência visual da página, como cores,

margens e espaçamento. Essa formatação visual é frequentemente a responsabilidade do designer web. Por outro lado, o JavaScript é essencial para adicionar interatividade aos elementos da página, permitindo que os desenvolvedores escrevam scripts que são executados pelos navegadores dos usuários.

O JavaScript permite uma ampla gama de funcionalidades interativas em sites. Por meio dele, é possível criar uma interação dinâmica com os usuários através de formulários, alterar conteúdos e estilos da página, armazenar dados no navegador do usuário, desenvolver layouts complexos, apresentar opções de produtos personalizadas e até mesmo desenvolver jogos que funcionam diretamente no navegador, independentemente do sistema operacional ou dispositivo utilizado pelo usuário.

A lógica de programação é o alicerce para aprender JavaScript, envolvendo a compreensão de instruções, dedução na construção de programas, enumeração de etapas, análise de soluções alternativas, e a atenção aos detalhes. O domínio desses conceitos é crucial para ensinar ao computador como resolver problemas e executar tarefas de maneira eficiente. Além disso, o entendimento de estruturas básicas como entrada, processamento e saída é fundamental para qualquer programador.

Para escrever e testar códigos JavaScript, são utilizados editores de código especializados que oferecem recursos como realce de sintaxe, sugestões de código e formatação automática. Essas ferramentas facilitam o desenvolvimento e a depuração de programas, permitindo que os desenvolvedores se concentrem na lógica e na funcionalidade do código.

## 2.4 AMBIENTES DE PROGRAMAÇÃO

Um ambiente de programação é um conjunto de ferramentas e processos que ajudam na criação de software. Geralmente, inclui um editor de texto para escrever o código, um programa que transforma esse código em algo que o computador possa entender, e uma ferramenta para testar e corrigir erros no código.

Foram utilizados o editor de texto Visual Studio Code e o ambiente de execução Node.js, que será abordado e recomendado durante todo o projeto. No entanto, também é possível usar até mesmo navegadores web e outras alternativas que serão mencionadas a seguir.

## 2.5 APRESENTAÇÃO DO VISUAL STUDIO CODE E NODE.JS

a) **Visual Studio Code**: é um editor de código-fonte gratuito e de código aberto, desenvolvido pela Microsoft. Ele é projetado para ser um editor leve e poderoso que funciona em todas as principais plataformas, como Windows, macOS e Linux. É otimizado para o desenvolvimento de aplicações modernas para a web e para a nuvem, e oferece suporte a uma variedade de linguagens de programação e frameworks (estrutura de suporte definida na qual outro projeto de software pode ser organizado

e desenvolvido, ou seja, fornece uma fundação sobre a qual é possível construir programas para uma plataforma específica).

As extensões no Visual Studio Code são ferramentas poderosas que permitem adicionar novas linguagens, depuradores e outras funcionalidades para apoiar o seu fluxo de trabalho de desenvolvimento. O editor suporta muitos recursos para desenvolvimento em JavaScript e Node.js. Além dos recursos principais que vêm com o produto baixado, como depuração e navegação de código, é possível a instalação de uma grande quantidade de extensões de qualidade para adicionar funcionalidades ao programa para desenvolvimento em JavaScript.

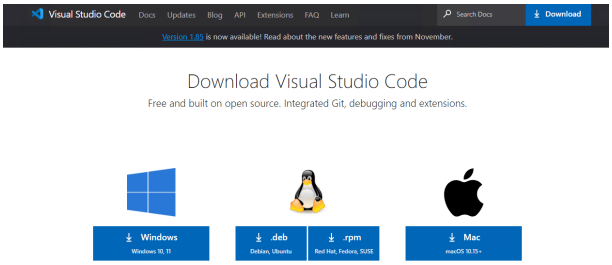
b) **Node.js**: é uma plataforma gratuita que permite usar JavaScript, uma linguagem de programação, fora do navegador. Isso significa que com Node.js, é possível criar servidores, aplicativos da web, ferramentas de linha de comando e *scripts* para automatizar tarefas. Node.js é baseado no motor de JavaScript do Chrome, o que o torna rápido e eficiente para construir aplicativos que podem crescer conforme necessário. Além disso, o Node.js inclui o 'npm', um gerenciador de pacotes que facilita a instalação de bibliotecas e ferramentas extras para ajudar no desenvolvimento de aplicações.

### 2.5.1 Instalação e configuração

O processo de configuração dos programas será relatado em passos:

1. Acesse `https://code.visualstudio.com/docs/setup/setup-overview` para seguir o guia de instalação e instalar a versão relativa ao sistema operacional

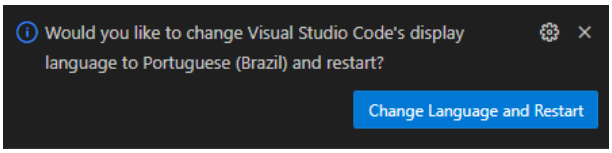
Figura 2.5.1 – Página para baixar o Visual Studio Code.




Fonte: Criada pelo autor, 2024.

2. Após a instalação, execute o aplicativo e verifique se existe uma notificação que recomenda a mudança de idioma para português brasileiro e selecione. Vai reiniciar o aplicativo.

Figura 2.5.2 – Notificação para modificar a interface para português brasileiro.




Fonte: Criada pelo autor, 2024.

Caso não apareça a notificação para modificar o idioma, vá em '*Extensions*', pesquise por "*Portuguese (Brazil) Language Pack*"e instale a extensão.

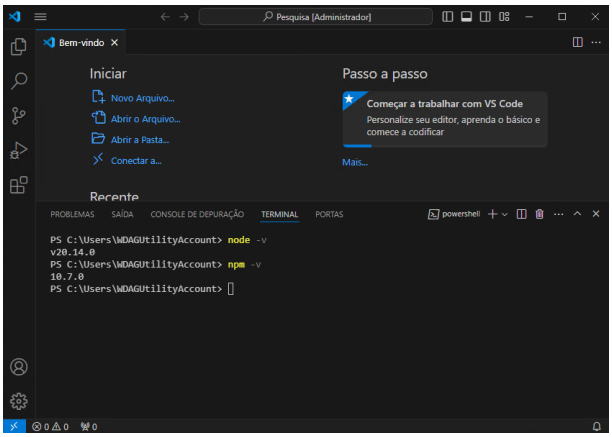
Figura 2.5.3 – Seção de extensões do pacote de idiomas em português brasileiro.


Fonte: Criada pelo autor, 2024.

3. Após as configurações iniciais, baixe e instale o Node.js nesta página `https://nodejs.org/en/download`, de acordo com o sistema operacional adequado.

4. Após a instalação, verifique se o Node.js e o 'npm' está funcionando corretamente, digitando no terminal do Visual Studio Code ou do sistema operacional: `node -v` e `npm -v`. Se retornar as versões do Node.js e 'npm' respectivamente, significa que está funcionando corretamente.

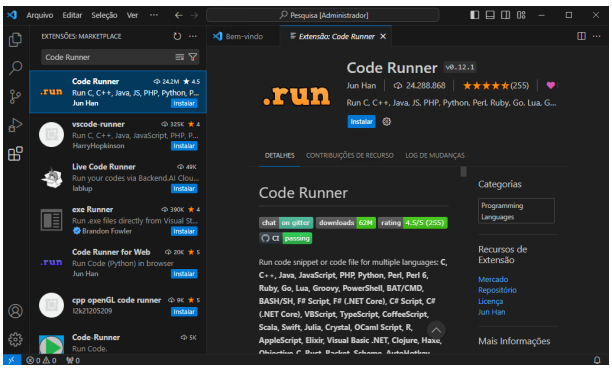
Figura 2.5.4 – Teste do Node.js via terminal do Visual Studio Code.




Fonte: Criada pelo autor, 2024.

5. Para executar os códigos diretamente no Visual Studio Code, usaremos a extensão '*Code Runner*' que está disponível na seção de extensões do programa (ou em `https://marketplace.visualstudio.com/items?itemName=formulahendry.code-runner`).

Figura 2.5.5 – Extensão 'Code Runner' do Visual Studio Code.




Fonte: Criada pelo autor, 2024.

6. Para executar um código em JavaScript, pode-se criar um arquivo com o nome `teste` que termina em '`.js`' tanto pela opção 'Novo Arquivo...' ou pelo terminal ao digitar `code teste.js`, já que esta é a extensão dos arquivos de código em JavaScript e a que será adotada neste projeto.

```
1    console.log("Hello World!");
```

Código 2.1 – Exemplo de código de teste

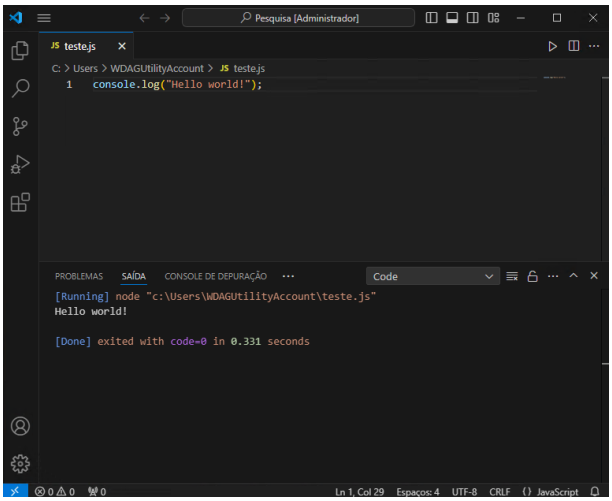
Ao digitar o código, e selecionar '*Run Code*' para executar, será retornado o que foi inserido no código na saída do código.

```
1    [Running] node "c:\Users\WDAGUtilityAccount\teste.js"
2    Hello world!
3
4    [Done] exited with code=0 in 0.331 seconds
```

Código 2.2 – Exemplo do retorno do console de acordo com o código anterior

Neste caso, será importante para esse projeto apenas a mensagem entre '`[Running]`' e '`[Done]`', tendo em vista que é resultado do código executado.

Figura 2.5.6 – Teste do código do arquivo 'teste.js'.




Fonte: Criada pelo autor, 2024.

## 2.6 ALTERNATIVAS

Uma das alternativas ao Visual Studio Code e Node.js são o **myCompiler** (`https://www.mycompiler.io/pt`) e o **OneCompiler** (`https://onecompiler.com/`), pois permitem executar códigos JavaScript utilizando o Node.js de maneira rápida e prática, sem a necessidade de instalações complexas, com a vantagem do OneCompiler executar também códigos HTML e CSS. É acessível em uma ampla gama de dispositivos, pois requer apenas um navegador web. Contudo, é importante ressaltar que o myCompiler e o OneCompiler dependem de uma conexão com a internet para funcionar. Outra possibilidade é utilizar o console de ferramentas para desenvolvedores, que está presente nos principais navegadores web disponíveis no mercado.

Caso haja interesse em utilizar um editor de código, um ambiente de desenvolvimento integrado ou até mesmo uma linguagem de programação distinta para executar as

operações matemáticas deste projeto, isso é perfeitamente viável. No entanto, será necessário realizar as adaptações e modificações pertinentes para garantir a compatibilidade e o funcionamento adequado.

# 3 CONCEITOS E FUNÇÕES BÁSICAS DE JAVASCRIPT

O JavaScript é uma linguagem de programação dinâmica e de tipagem fraca, conhecida principalmente por seu papel em páginas web. No entanto, seu uso vai além, estendendo-se a ambientes que não usam navegadores de internet como Node.js e Adobe Acrobat. Sua natureza baseada em protótipos permite estilos de programação orientada a objetos, imperativos e declarativos, como a programação funcional.

No caso deste projeto, serão abordados apenas as principais operações do JavaScript que serão usadas no capítulo a seguir.

Caso deseje um maior aprofundamento, o MDN Web Docs (`https://developer.mozilla.org/pt-BR/docs/Web`) é uma fonte rica em informações e tutoriais sobre JavaScript, oferecendo uma visão abrangente sobre a linguagem e suas capacidades, seja iniciante ou não.

## 3.1 IMPRIMIR TEXTO

Para imprimir um texto no console, utilizamos o método '`console.log()`'. É uma função muito utilizada no JavaScript para exibir mensagens na console do navegador. É parte da *Application Programming Interface* (Interface de Programação de Aplicações) (API) do console, que fornece várias funcionalidades para ajudar os desenvolvedores a realizar tarefas de depuração durante o desenvolvimento de um site ou aplicação web.

Podemos escrever usando o símbolo (' ') com o texto entre eles, que será mostrado no console.

```
1    console.log('Teste...');
```

Código 3.1 – Exemplo de uso do 'console.log' com (' ')

É possível também utilizar o (" ") para textos.

```
1    console.log("Teste...");
```

Código 3.2 – Exemplo de uso do 'console.log' com (" ")

Podemos adicionar variáveis e funções dentro do '`console.log`'.

```
1    console.log("O valor da constante e: ", Math.E);
```

Código 3.3 – Exemplo de uso do 'console.log' com variáveis

Para adicionar variáveis dentro do texto, usamos o símbolo (` `), e para declarar a variável, usamos '`${variavel}`', com o nome da variável dentro entre os símbolos das chaves.

```
1    console.log(‘Temos que ${Math.PI}... sendo o valor do pi.‘);
```

Código 3.4 – Exemplo de uso do ’console.log’ com variáveis dentro do texto

## 3.2 COMENTÁRIOS

Os comentários são usados para inserir descrições, explicações ou qualquer outra anotação que o desenvolvedor deseja incluir no código. Esses comentários são ignorados pelo motor de execução do JavaScript e não têm efeito sobre o comportamento do código. Existem dois tipos de comentários, para linha única e múltiplas linhas.

Comentários de linha única são iniciados com ’//’ e se estendem até o final da linha atual.

```
1    //Exemplo de comentario em uma unica linha.
```

Código 3.5 – Exemplo de comentário de linha única

Comentários de múltiplas linhas são iniciados com ’/*’ e terminam com ’*/’. Eles podem abranger várias linhas e são úteis para comentários mais extensos.

```
1    /* Exemplo de comentario
2    que suporta
3    multiplas linhas. */
```

Código 3.6 – Exemplo de comentário de múltiplas linhas

## 3.3 DECLARAÇÃO DE VARIÁVEIS

As declarações de variáveis no JavaScript são essenciais para o controle e manipulação de dados dentro de um programa. Uma variável é um local nomeado no código que armazena um valor e esse valor pode ser acessado e modificado usando o nome da variável. O valor da variável vem após o símbolo de igualdade (=) que é usado para atribuir valores a variáveis, e não deve confundir com o operador de igualdade matemático;

As variáveis declaradas com `var` têm escopo de função ou global se declaradas fora de uma função.

```
1    var Variavel = "Teste";
```

Código 3.7 – Exemplo de variável ’var’ com nome ’Variavel’ e de valor "Teste"

Uma variável de escopo de bloco, declarada com `let`, é opcionalmente inicializada com um valor. Podem ter seu valor substituído.

```
1     let x = 2599;
2     let x = 2600; //'x' agora tem o valor 2600.
```

Código 3.8 – Exemplo de variável 'let' com substituição de valor

A declaração `const` serve para declarar uma constante de escopo de bloco, que é somente para leitura após a inicialização. Constantes não podem ter seu valor substituído.

```
1     const aproximadoPi = 3.14;
```

Código 3.9 – Exemplo de variável 'const'

**Observação:** Variáveis declaradas com `let` e constantes declaradas com `const` têm escopo de bloco, o que significa que elas só existem dentro do bloco onde foram declaradas.

## 3.4 TIPOS DE DADOS

Os tipos de dados são muito importantes para entender como manipular informações dentro de um programa.

Os booleanos (`boolean`) representam uma entidade lógica que pode ter dois valores: `true` (verdadeiro) ou `false` (falso). São usados para tomar decisões no código, como em estruturas condicionais e laços (*loops*).

```
1     const variavelVerdade = true;
2     const variavelFalso = false;
```

Código 3.10 – Exemplo de variáveis de valor booleano

O tipo nulo tem exatamente um valor: `null`. Representa a ausência intencional de qualquer valor de objeto e é usado quando se deseja indicar que uma variável não deve apontar para nenhum objeto.

```
1     const variavelNula = null;
```

Código 3.11 – Exemplo de variável de valor nulo

Uma variável que não teve um valor atribuído possui o valor `undefined`. Indica que uma variável foi declarada, mas ainda não foi definida com um valor.

```
1     let Indefinido = undefined; // ou
2     let Indefinido;
```

Código 3.12 – Exemplo de variável de valor não definido

Uma variável com valor de números incluem tanto inteiros quanto números de ponto flutuante (`float`). Um número de ponto flutuante é um número que possui uma parte inteira e uma parte fracionária, separadas por um ponto (*dot*), que é conhecido como ponto decimal. No JavaScript, os números de ponto flutuante seguem o padrão IEEE 754 para aritmética de ponto flutuante, que é um formato binário de 64 bits de precisão dupla.

```
1    const numeroInteiro = 1997;
2    const numeroDecimal = 1.41327489;
```

Código 3.13 – Exemplo de variáveis de valores númericos

Uma variável de `string` são sequências de caracteres usadas para representar texto. Podem ser criadas usando aspas simples, duplas ou crases (como citado em imprimir no console).

```
1    const exemploString = "exemplo de texto";
```

Código 3.14 – Exemplo de variável com 'string'

Uma variável de objeto (`Object`) são coleções de propriedades, onde cada propriedade é uma associação entre um nome (ou chave) e um valor. O valor de uma propriedade pode ser uma função, nesse caso, a propriedade é conhecida como um método.

```
1    const exemploObjeto = Object;
```

Código 3.15 – Exemplo de variável 'Object'

## 3.5 AJUSTE DE CASAS DECIMAIS

O método '`toFixed()`' é utilizado para formatar um número usando notação de ponto fixo, o que é especialmente útil para controlar o número de casas decimais exibidas após o ponto decimal.

Temos '`numero.toFixed([digitos])`' onde `digitos` é o número de dígitos que aparecem depois do ponto decimal; este pode ser um valor entre 0 e 20, e algumas implementações podem suportar uma variação de números maiores. Se este argumento for omitido, será tratado como 0. O método '`toFixed()`' retorna uma `string` representando o número que não usa notação exponencial e tem exatamente `digitos` dígitos depois da casa decimal.

O número será arredondado se necessário, e será adicionado zeros à parte após a vírgula para que este tenha o tamanho que foi especificado. Se o número for maior que $1 \times 10^{21}$ , então será invocado o método '`Number.prototype.toString()`' e será retornado uma string em notação exponencial.

```
1     let inteiro = 21;
2     let racional = 9.33333333;
3     let DizimaPeriodica = 1/3;
4     let raizQuadrada = Math.SQRT2;
5     let notCientifica = 10 ** -21; //Significa 10 elevado a -21
6     console.log(inteiro.toFixed(3)); //Retorna 21.000
7     console.log(racional.toFixed()); //Retorna 9
8     console.log(DizimaPeriodica.toFixed(5)); //Retorna 0.33333
9     console.log(raizQuadrada.toFixed(10)); //Retorna 1.4142135624
10    console.log(notCientifica); //Retorna 1.0000000000000001e-21
```

Código 3.16 – Exemplos de uso do método 'toFixed'

## 3.6 CONVERSÃO DE STRING PARA NÚMERO

A conversão de **strings** para números é uma operação comum no JavaScript.

A função '**parseInt()**' analisa uma **string** e retorna um inteiro na base especificada, e a sintaxe é '**parseInt(string[, radix])**', onde o parâmetro **radix** é opcional e representa a base do sistema numérico que está sendo usado, e caso o **radix** não seja especificado, o JavaScript determinará a base numérica considerando o conteúdo da **string**. Retorna **NaN** se a **string** não puder ser convertida em um número.

```
1     var numeroEmString = "3.1415";
2     console.log("Valor inteiro: " + parseInt(numeroEmString)); //
         converte a string em numero inteiro
```

Código 3.17 – Exemplo de uso do 'parse.Int'

A função '**parseFloat()**' analisa uma **string** e retorna um número de ponto flutuante. A sintaxe é '**parseFloat(string)**', onde ignora os espaços em branco iniciais da **string** e analisa até que encontre um caractere que não seja um número válido. Retorna **NaN** se o primeiro caractere não puder ser convertido em um número.

```
1     var numeroEmString = "3.1415";
2     console.log("Valor: " + parseFloat(numeroEmString)); //converte em
         numero de ponto flutuante
```

Código 3.18 – Exemplo de uso do 'parse.Float'

## 3.7 DECLARAÇÃO EM BLOCO

É uma estrutura fundamental que permite agrupar zero ou mais declarações. Essa estrutura é delimitada por um par de chaves (**{}**) e pode conter uma lista de declarações e instruções. A declaração de bloco é frequentemente chamada de declaração composta em outras linguagens e é comumente usada em associação com instruções de controle de fluxo, como **if...else**, **for**, etc.

```
1    {
2        valor1 = 0.5,
3        valor2 = "0.5",
4        valor3 = .5
5    }
```

Código 3.19 – Exemplo de declaração de um bloco

## 3.8 FUNÇÃO

Funções são blocos fundamentais de construção de código que permitem a execução de tarefas ou o cálculo de valores. Uma função pode ser vista como um procedimento que recebe alguns dados de entrada (parâmetros), processa esses dados e retorna um resultado de saída.

Uma função é declarada usando a palavra-chave `function`, seguida por: nome da função; uma lista de parâmetros entre parênteses, separados por vírgulas; e um bloco de declarações entre chaves que definem o corpo da função.

```
1    const nomePessoa = "Fulano";
2    function bemVindo(nome) { //'bemVindo' o nome da funcao e 'nome'
         sendo parametro.
3        return "Seja bem-vindo, "+nome+"!"; //Usa-se 'return' para citar
             o valor que deve ser retornado quando a funcao e chamada.
4    }
5    console.log(bemVindo(nomePessoa)); // '\bemVindo()': chama a funcao
         com o valor de 'nomePessoa'.
```

Código 3.20 – Declaração de uma função

Além das declarações de função, as funções também podem ser criadas por meio de expressões de função. Uma expressão de função pode ser anônima (sem nome) ou nomeada, e pode ser armazenada em uma variável.

```
1    const saudacao = function bemVindo(nome) {
2    // ...
3    }
```

Código 3.21 – Declaração de uma função como uma variável

As funções `arrow` são uma alternativa compacta às expressões de função tradicionais. Elas são úteis para funções que requerem uma sintaxe curta e não são métodos.

```
1    const saudacao = nome => "Seja bem-vindo, "+nome+"!"; //Nome da
         variavel 'saudacao', com 'nome' sendo o parametro, (=>) declara a
          funcao arrow, e apos ele sera o valor retornado
2    console.log(saudacao("Fulano"));
```

Código 3.22 – Exemplo de declaração de uma função 'arrow'

## 3.9 ARRAY

`Arrays` são estruturas de dados que permitem armazenar múltiplos valores em uma única variável. Eles são objetos que representam uma lista ordenada de valores, onde cada valor é identificado por um índice, e temos que o índice se inicia em 0.

```
1   var materiaisEscolares = ["Lapis", "Caneta", "Caderno"];
```

Código 3.23 – Exemplo de declaração de 'array'

Os `arrays` possuem métodos e propriedades:

a) `forEach`: é uma função de iteração disponível em `arrays` para executar uma função em cada um dos elementos do `array`.

```
1   var materiaisEscolares = ["Lapis", "Caneta", "Caderno", "Borracha"];
2   materiaisEscolares.forEach(function(MaterialEscolar) { // Imprime
        cada elemento do array por linha
3       console.log("->", MaterialEscolar);
4   });
```

Código 3.24 – Exemplo de uso do 'forEach'

b) `reduce`: o método executa uma função "*reducer*" para cada elemento do `array`, resultando em um único valor de retorno, ou seja, pode ser usada para transformar um `array` em um único valor. A ideia é "reduzir" um `array` a um valor que pode ser um número, uma `string`, um objeto, etc. A função "*reducer*" recebe os seguintes argumentos: acumulador, que que se acumula após cada operação e carrega o resultado da operação anterior e é passado para a próxima iteração; valor atual, o elemento atual do array que está sendo processado; e o valor inicial, que é o elemento atual do `array` que está sendo processado e caso não for fornecido, o primeiro elemento do `array` será usado como acumulador inicial e o `reduce` começará do segundo elemento.

```
1   var arrayNum = [7, 12, 15, 9];
2   function somar(acumulador, valorAtual) { //Define uma funcao que
        sera usada para somar os valores do array
3       return acumulador + valorAtual; //Retorna a soma do acumulador
            com o valor atual
4   }
5   //Usa o metodo 'reduce' com a funcao 'somar' para processar o array
6   var somaValoresArray = arrayNum.reduce(somar, 0); //O segundo
        argumento de 'reduce' (0) e o valor inicial do acumulador
7   console.log(somaValoresArray); //Retorna 43
```

Código 3.25 – Exemplo de uso do 'reduce'

c) `join`: junta todos os elementos de um `array` em uma `string`. Pode-se especificar um separador para ser usado entre os elementos. Se nenhum separador for especificado, uma vírgula será usada por padrão. É útil para converter `arrays` em `strings` para exibição ou armazenamento.

```
1  var arrayNumeros = [7, 12, 15, 9];
2  var unirValoresEmArray = console.log(`${arrayNumeros.join('o, ')}o.`
       ); //'o' significa o simbolo de numero ordinal
3  //Adiciona 'o' em cada elemento de 'arrayNumeros'
```

Código 3.26 – Exemplo de uso do 'join'

d) `length`: retorna ou define o número de elementos em um `array`. É frequentemente usada em laços para determinar o número de vezes que o laço deve executar. A propriedade `length` é atualizada automaticamente à medida que novos elementos são adicionados ou removidos do `array`.

```
1  var arrayNumeros = [7, 12, 15, 9];
2  var quantidadeValoresArray = arrayNumeros.length; //Retorna o valor
       4
```

Código 3.27 – Exemplo de uso do 'length'

e) `sort`: ordena os elementos de um `array` *in-place* (`array` que é modificado diretamente na sua estrutura original) e retorna o `array`. A ordenação não é necessariamente estável ou numérica por padrão, e é feita convertendo os elementos para `strings` e comparando suas sequências de valores UTF-16. Pode-se fornecer sua própria função de comparação para determinar a ordem dos elementos.

```
1  var arrayNumeros = [7, 12, 15, 9];
2  var ordenarValoresdoArray = arrayNumeros.sort((a, b) =>
3      a - b //Neste caso, estamos ordenando em ordem crescente.
4  ); //Retorna [ 7, 9, 12, 15 ]
```

Código 3.28 – Exemplo de uso do 'sort'

f) `push`: é utilizado para adicionar um ou mais elementos ao final de um `array`. Ao usar esse método, ele retorna o novo comprimento do `array` após os elementos terem sido adicionados.

```
1  var arrayPares = [2, 4, 6, 8];
2  arrayPares.push(10, 12); //Adicionou os valores 10 e 12
3  ); //Agora 'arrayPares' retorna [ 2, 4, 6, 8, 10, 12 ]
```

Código 3.29 – Exemplo de uso do 'push'

g) `map`: chama a função `callback` para cada elemento do `array` original, em ordem, e constrói um novo `array` com base nos retornos de cada chamada. Importante notar que o '`map()`' não modifica o `array` original, mas sim cria um novo com os resultados da função `callback`.

```
1  const numeros = [2, 4, 6];
2  const dobro = numeros.map(num => num * 2);
3  console.log(numeros, dobro)
4  //'dobro' reorna [2, 4, 6], 'numeros' ainda retorna [4, 8, 12]
```

Código 3.30 – Exemplo de uso do 'map'

## 3.10 SISTEMAS NUMÉRICOS

No JavaScript, os sistemas numéricos permitem aos desenvolvedores trabalhar com diferentes bases numéricas para representar e manipular números.

**Observação**: ao trabalhar com valores não-decimais em JavaScript, como binários, octais ou hexadecimais, e os imprime no console, o JavaScript converte automaticamente esses valores para o sistema numérico decimal.

a) **Binários**: os números binários tem base 2 e usam apenas os dígitos 0 e 1. Eles são prefixados com `0b` ou `0B`.

```
1  const binario = 0b101001; //Valor decimal: 41.
```

Código 3.31 – Exemplo de valor binário

b) **Octal**: os números octais tem base 8 e usam uma sequência de dígitos octais (0-7). Usamos o prefixo `0o` ou `0O` para representar octais.

```
1  const octal = 0o35; //Valor decimal: 29
```

Código 3.32 – Exemplo de valor octal

c) **Hexadecimal**: números hexadecimais que tem base 16 e usam dígitos de 0 a 9 e letras de A a F (ou minúsculos). Eles são prefixados com `0x` ou `0X`.

```
1  const hexadecimal = 0xA2d; //Valor decimal: 2605
```

Código 3.33 – Exemplo de valor hexadecimal

## 3.11 PROPRIEDADE `LENGHT`

A propriedade `length` em `strings` retorna o número de unidades de código UTF-16 presentes na `strings`. Isso é particularmente importante porque, embora o JavaScript

utilize a codificação UTF-16, onde a maioria dos caracteres comuns são representados por uma única unidade de código, caracteres menos comuns podem necessitar de duas unidades de código.

```
1    const palavra = "BrasilCampeao"; //A string possui 13 elementos
2    console.log("'Brasil Campeao' tem",palavra.length,"letras!");
```

Código 3.34 – Exemplo de uso do 'length' em uma variável 'string'

## 3.12 OPERADORES E INCREMENTADORES

Os operadores são símbolos especiais que realizam operações em operandos (valores e variáveis). Operadores e incrementadores são úteis para a lógica de programação, permitindo realizar comparações, atribuições, lógicas e operações matemáticas, que é de grande importância para este projeto.

a) **Operadores de comparação**

i) **Maior que (>)**: se o operando à esquerda for maior que o operando à direita, retorna `true`;

ii) **Menor que (<)**: se o operando à esquerda for menor que o operando à direita, retorna `true`;

iii) **Maior ou igual a (>=)**: se o operando à esquerda for maior ou igual que o operando à direita, retorna `true`;

iv) **Menor ou igual que (<=)**: se o operando à esquerda for menor ou igual que o operando à direita, retorna `true`.

```
1    let valor = 2;
2    const comparacao1 = valor > 1; //Retorna 'true'
3    const comparacao2 = valor < 1; //Retorna 'false'
4    const comparacao3 = valor <= 2; //Retorna 'true'
5    const comparacao4 = valor >= 1; //Retorna 'true'
```

Código 3.35 – Exemplo de operadores de comparação

b) **Operadores de igualdade**

i) **Operador de atribuição** (=): é usado para atribuir valores a variáveis. Não confundir com o operador de igualdade matemático;

ii) **Operador de igualdade** (==): compara dois valores para verificar se são iguais, mas antes de fazer a comparação, ele converte os valores para um tipo comum se eles forem de tipos diferentes;

iii) **Operador de igualdade restrita** (`===`): compara tanto o valor quanto o tipo dos dois operandos, sem converter os tipos. Se os tipos forem diferentes, a comparação resultará em `false`;

iv) **Operador de desigualdade estrita** (`!==`): retorna `true` se os operandos não são iguais ou não são do mesmo tipo.

```
1     let valor1 = 2;
2     let valor2 = 4;
3     const valor3 = "2";
4     const comparacao1 = valor1 == valor2; //Retorna 'false'.
5     const comparacao2 = valor1 === valor3; //Retorna 'false', mas caso
          '==', retornaria 'true'.
6     const comparacao3 = valor1 !== valor2; //Retorna 'true'
```

Código 3.36 – Exemplo de uso dos operadores de igualdade

c) **Operadores condicional** (`?`): é o único operador JavaScript que possui três operandos e é frequentemente usado como um atalho para a instrução `if`.

```
1     let valor = 2;
2     const comparacao1 = (valor > 1) ? "Verdadeiro" : "Falso";
3     const comparacao2 = (valor < 1) ? "Verdadeiro" : "Falso";
4      // Realiza uma verificacao na condicao, e atribui um valor com base
           nesta condicao. Caso for 'true', retorna "Verdadeiro". Senao,
          retorna "Falso".
```

Código 3.37 – Exemplo de uso de operador condicional

d) **Operadores de atribuição**

i) **Atribuição de adição** (`+=`): adiciona o valor do operando direito ao operando esquerdo e atribui o resultado ao operando esquerdo;

ii) **Atribuição de subtração** (`-=`): subtrai o valor do operando direito do operando esquerdo e atribui o resultado ao operando esquerdo;

iii) **Atribuição de multiplicação** (`*=`): multiplica o operando esquerdo pelo operando direito e atribui o resultado ao operando esquerdo;

iv) **Atribuição de divisão** (`/=`): divide o operando esquerdo pelo operando direito e atribui o resultado ao operando esquerdo.

```
1     let valor1 = 2;
2     let valor2 = 4;
3     valor1 += valor2; //Agora valor1 retorna 6
4     valor2 -= valor1; //Agora valor2 retorna -2
5     valor1 *= valor2; //Agora valor1 retorna -12
6     valor2 /= valor1; //Agora valor2 retorna 0.16666666666666666
```

Código 3.38 – Exemplos de uso dos operadores de atribuição

e) **Operadores lógicos**

i) **E lógico** (`&&`): retorna `true` se ambos os operandos forem verdadeiros;

ii) **OU lógico** (`||`): retorna `true` se pelo menos um dos operandos for verdadeiro.

```
1    let valor = 2;
2    const eLogico = (valor > 1) && (valor < 1); //Retorna 'false'
3    const ouLogico = (valor > 1) || (valor < 1); //Retorna 'true'
```

Código 3.39 – Exemplo de uso dos operadores lógicos

f) **Operadores de incremento e decremento**

i) **Decremento** (`--`): diminui o valor do operando em um;

ii) **Incremento** (`++`): aumenta o valor do operando em um.

**Observação**: ao adicionar o incremento ou decremento depois da variável, os operadores primeiro retornam o valor da variável e depois realizam a operação de incremento ou decremento. Caso adicione o incremento ou decremento depois da variável, realizam a operação antes de retornar o valor, e é útil dependendo de quando é necessário do valor atualizado.

```
1    let valor1 = 2;
2    let valor2 = 4;
3    valor1++; //Agora valor1 retorna 3
4    valor2--; //Agora valor2 retorna 3
```

Código 3.40 – Exemplo de pré incremento e pré decremento

## 3.13 DECLARAÇÃO `IF...ELSE`

A declaração `if...else` é uma estrutura condicional que executa diferentes blocos de código com base em uma condição booleana. Temos que a sintaxe é '`if (condicao){...}`', em que `condicao` é uma expressão que é avaliada como verdadeira (`true`) ou falsa (`false`). Se a condição for verdadeira, o primeiro bloco de código dentro das chaves (`{}`) após o `if` será executado. Se for falsa, o código dentro do bloco `else` será executado. Também é possível aninhar declarações `if...else` para verificar múltiplas condições (usando `else if`).

```
1     var numero = 15;
2     if (numero > 0) {
3         console.log(numero,": positivo.");
4     } else if (numero < 0) {
5         console.log(numero,": negativo.")
6     } else {
7         console.log(numero,": zero.")
8     }
```

Código 3.41 – Exemplo de uso do 'if...else'

## 3.14 DECLARAÇÃO `SWITCH`

É uma estrutura de controle de fluxo que permite a execução de diferentes blocos de código com base em diferentes casos. Temos a estrutura:

1. A expressão dentro do `switch` é avaliada uma vez;
2. O valor da expressão é comparado com os valores de cada `case`;
3. Se houver uma correspondência, o bloco de código associado é executado;
4. A palavra-chave `break` é usada para sair de um case e continuar a execução do código após o `switch`;
5. Se nenhum case corresponder, o código no `default` é executado, se houver.

```
1     var laterna = "Acesa";
2     switch (laterna) {
3         case "Acesa":
4             console.log("Laterna acesa!");
5         break;
6         case "Apagada":
7             console.log("Laterna apagada!");
8         break;
9         default:
10            console.log("Entrada invalida...");
11    }
```

Código 3.42 – Exemplo de uso do 'if...else'

## 3.15 DECLARAÇÃO `FOR`

É uma das estruturas de *loop* (laço) mais utilizadas, permitindo executar um bloco de código várias vezes com diferentes valores. A sintaxe básica da declaração for é '`for (inicializacao; condicao; expressaoAtualizacao)`', onde: `inicializacao` é usada para definir o contador, e as variáveis declaradas aqui são visíveis apenas dentro do *loop*; `condicao` é avaliada antes de cada iteração do *loop*, se verdadeira o *loop* continua e se falsa o *loop* termina; `expressaoAtualizacao` é executada no final de cada iteração, geralmente

usada para atualizar o contador. E dentro do bloco, o código será executado em cada iteração do *loop*, desde que a condição seja verdadeira.

```
1     for (let valor = 1; valor <= 10; valor++) {
2         console.log("Num: ", valor);
3     } //Imprime no console os numeros de 1 a 10.
```

Código 3.43 – Exemplo de uso do 'loop' com 'for'

## 3.16 DECLARAÇÃO `WHILE`

É usada para criar um *loop* que executa uma instrução especificada enquanto a condição de teste for avaliada como verdadeira. A condição é avaliada antes da execução da instrução. A sintaxe é '`while (condicao)`', com `condicao` sendo uma expressão avaliada antes de cada passagem pelo *loop*. Se a condição for avaliada como verdadeira, a instrução é executada. Quando a condição for avaliada como falsa, a execução continua com a instrução após o laço `while`.

```
1     let valor = 1;
2     while (valor <= 11) {
3       console.log("Num: ", valor);
4       valor++;
5     } //Enquanto a condicao for valida, vai imprimir no console ate o
        numero 11
```

Código 3.44 – Exemplo de uso do 'while'

## 3.17 OBJETO `MATH`

O objeto `Math` em JavaScript é um objeto embutido que possui propriedades e métodos estáticos para realizar tarefas matemáticas. Este objeto não é um construtor e todas as suas propriedades e métodos são estáticos, o que significa que eles pertencem ao próprio objeto `Math` e não a instâncias de `Math`. Existem uma variedade de propriedades e métodos, e inclui várias outras funções matemáticas que podem ser úteis dependendo do problema (disponíveis em `https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Math`). Exemplos de algumas das propriedades e métodos estáticos:

a) **Propriedades estáticas**

i) `Math.E`: a base dos logaritmos naturais;

ii) `Math.LN10`: o logaritmo natural de 10;

iii) `Math.PI`: O valor de $\pi$ (pi), que é a relação entre a circunferência de um círculo e o seu diâmetro;

iv) `Math.SQRT2`: A raiz quadrada de 2.

b) **Métodos estáticos**

i) `Math.abs(x)`: retorna o valor absoluto (módulo) de um número;

ii) `Math.floor(x)`: arrendonda um número, e retorna o maior inteiro menor ou igual a esse número;

iii) `Math.random()`: retorna um número pseudo-aleatório entre 0 e 1;

iv) `Math.round(x)`: retorna o valor de um número arredondado para o inteiro mais próximo;

v) `Math.sqrt(x)`: retorna a raiz quadrada de um número (`sqrt` lembra *square root*);

vi) `Math.sin(x)`: retorna o seno de um ângulo em radianos;

vii) `Math.cos(x)`: retorna o cosseno de um ângulo em radianos;

viii) `Math.tan(x)`: retorna a tangente de um ângulo em radianos.

# 4 MATEMÁTICA BÁSICA EM JAVASCRIPT

Neste capítulo, serão exploradas definições e teoremas fundamentais da matemática, acompanhados de suas respectivas demonstrações. Para ilustrar a aplicabilidade prática desses conceitos, será apresentado um exemplo de código em JavaScript, que realiza uma operação matemática pertinente ao tema discutido. O professor pode utilizar estes exemplos a seguir em sala de aula para motivar os alunos a aprender JavaScript com os conceitos matemáticos ou vice-versa.

## 4.1 OPERAÇÕES BÁSICAS

### 4.1.1 Operadores básicos

Temos as quatro principais operações básicas, também disponíveis no JavaScript como operadores: adição, subtração, multiplicação e divisão.

a) Para adição:

```
1    const valor1 = 10;
2    const valor2 = 5;
3    let soma = valor1 + valor2;
4    console.log("Resultado:",soma);
```

Código 4.1 – Exemplo de adição

b) Para subtração:

```
1    const valor1 = 10;
2    const valor2 = 5;
3    let subtracao = valor1 - valor2;
4    console.log("Resultado:",subtracao);
```

Código 4.2 – Exemplo de subtração

c) Para multiplicação:

```
1    const valor1 = 10;
2    const valor2 = 5;
3    let multipilcacao = valor1 * valor2;
4    console.log("Resultado:",multiplicacao);
```

Código 4.3 – Exemplo de multiplicação

d) Para divisão:

```
1    const valor1 = 10;
2    const valor2 = 5;
3    let divisao = valor1 / valor2;
4    console.log("Resultado:",divisao);
```

Código 4.4 – Exemplo de divisão

### 4.1.2 Operador módulo

O operador módulo (%) é um operador de distribuição que retorna o resto de uma divisão entre dois valores inteiros.

```
1        const valor1 = 15;
2        const valor2 = 4;
3        const resto = valor1 % valor2;
4        console.log(`O resto da divisao de ${valor1} por ${valor2}: ${
           resto}.`);
```

Código 4.5 – Exemplo que retorna o resto de uma divisão

### 4.1.3 Potência e raízes

Ao realizar operações de potenciação, temos duas alternativas;

a) Potência com `Math.pow`. Neste caso, `valor1` é a base e `valor2` é o exponte da potência:

```
1        const valor1 = 4;
2        const valor2 = 3;
3        const potencia = Math.pow(valor1, valor2); //'potencia' recebe o
             resultado de 4 elevado a 3
4        console.log("Potencia:",potencia);
```

Código 4.6 – Exemplo de cálculo de potência usando o 'Math.pow'

b) Potência com o operador matemático '**', com uso similar ao anterior:

```
1        const valor1 = 4;
2        const valor2 = 3;
3        const potencia = valor1 ** valor2;
4        console.log("Potencia:",potencia);
```

Código 4.7 – Exemplo de cálculo de potência usando um operador matemático

Para realizar cálculo de raízes, temos;

c) Calcular uma raiz quadrada, neste exemplo seria raiz quadrada de 4, também usando o objeto `Math`:

```
1         let valor = 9;
2         const raizQuadrada = Math.sqrt(valor);
3         console.log("Raiz quadrada:",raizQuadrada);
```

Código 4.8 – Exemplo de cálculo de raíz quadrada usando 'Math.sqrt'

d) Calcular uma raiz enésima qualquer, usa-se da potência de expoente racional (Iezzi; Dolce; Murakami, 2013):

Dado $a \in \mathbb{R}_+^*$ e $\frac{p}{q} \in \mathbb{Q}$, com $p \in \mathbb{Z}$ e $q \in \mathbb{N}^*$, tem-se

$$a^{\frac{p}{q}} = \sqrt[q]{a^p}.$$

```
1         const valor1 = 4;
2         const valor2 = 3;
3         const raizN = Math.pow(valor1, 1/valor2);
4         console.log("Raiz enesima:",raizN);
```

Código 4.9 – Exemplo de cálculo de raiz usando 'Math.pow' com a definição de potência de expoente racional

### 4.1.4 Porcentagem

Consideremos o cálculo de porcentagem de um valor, cujo a razão é cem, onde $p$ é porcentagem, $C$ capital e $i$ a taxa (Bosquilha; Amaral; Miranda, 2010):

$$p = \frac{C \cdot i}{100}.$$

```
1     const i = 30;
2     const capital = 500;
3     const p = (i / 100) * capital;
4     console.log(`${i}% de ${capital}: ${p}.`);
```

Código 4.10 – Exemplo de cálculo de porcentagem

## 4.2 MÁXIMO DIVISOR COMUM

O Máximo Divisor Comum (MDC) refere-se ao maior número que divide dois ou mais números inteiros não-nulos sem deixar resto, ou seja, é o maior número presente no conjunto de divisores comuns a esses números. Este conjunto é limitado e finito, pois não pode exceder o menor dos números analisados. Sendo assim, MDC é o elemento máximo desse conjunto de divisores comuns (Bosquilha; Amaral; Miranda, 2010).

Usaremos o Lema de Euclides para descobrir o MDC entre dois números (Moreira, 2012).

**Lema.** *Se temos dois números inteiros $a$ e $b$, onde $a = bq + r$, com $r \in \mathbb{Z}$ e $0 \leq r < b$, então o máximo divisor comum* mdc *de $a$ e $b$ é igual ao* mdc *de $b$ e $r$, ou seja,* $\text{mdc}(a, b) = \text{mdc}(b, r)$.

*Demonstração.* Para provar que $\text{mdc}(a, b) = \text{mdc}(b, r)$, mostramos que os conjuntos de divisores $D_a \cap D_b$ e $D_b \cap D_r$ são iguais. Se $d$ é um divisor comum de $a$ e $b$, então $d$ também divide $a - bq$, que é igual a $r$. Portanto, $d$ está em $D_b \cap D_r$. O mesmo raciocínio se aplica se $d$ é um divisor comum de $b$ e $r$, mostrando que $d$ também divide $a$. □

No código a seguir, `i` é um valor temporário que admite os valores de `n1` e `n2` de forma que continuará iterando enquanto `n2` não for 0. Cada vez que itera, o valor `n1` ficará igual ao de `n2`. Sendo assim, quando `n2` for 0, o MDC será o valor final de `n1`.

```
1   let num1 = 108;
2   let num2 = 90;
3   function buscarMDC(n1, n2) {
4       while (n2 !== 0) {
5           let i = n2;
6           n2 = n1 % n2;
7           n1 = i;
8       }
9       return n1;
10  }
11  const MDC = buscarMDC(num1, num2);
12  console.log(`O MDC de ${num1} e ${num2}: ${MDC}.`);
```

Código 4.11 – Exemplo de descoberta do MDC de dois números

## 4.3 MÍNIMO MÚLTIPLO COMUM

O Mínimo Múltiplo Comum (MMC) é o menor número positivo que é múltiplo comum de no mínimo dois números positivos. Significa que é o menor número que pode ser dividido por cada um dos números originais sem deixar resto. Ele é encontrado na interseção do conjunto de múltiplos de cada um dos números envolvidos (Bosquilha; Amaral; Miranda, 2010).

No código a seguir, usaremos a relação do MDC e MMC (Moreira, 2012), onde:

$$\text{mdc}(a, b) \cdot \text{mmc}(a, b) = a \cdot b \iff \text{mmc}(a, b) = \frac{a \cdot b}{\text{mdc}(a, b)}.$$

Sobre o código, usamos a mesma função `buscarMDC` do código anterior para descobrir o MDC de dois números. Daí, usa-se uma função simples, `buscarMMC`, para calcular o MMC e `Math.abs` retorna o valor absoluto, onde não considera o sinal e o valor sempre será positivo.

```
1     let num1 = 80;
2     let num2 = 15;
3     function buscarMDC(n1, n2) {
4         while (n2 !== 0) {
5             let i = n2;
6             n2 = n1 % n2;
7             n1 = i;
8         }
9         return n1;
10    }
11    function buscarMMC(n1, n2) {
12        return Math.abs(n1 * n2) / buscarMDC(n1, n2);
13    }
14    const MMC = buscarMMC(num1, num2);
15    console.log(`MMC de ${num1} e ${num2}: ${MMC}.`);
```

Código 4.12 – Exemplo de descoberta do MMC usando a fórmula da relação entre MDC e MMC

## 4.4 FUNÇÕES

Considere dois conjuntos $A$ e $B$, ambos não vazios. Uma **função** de $A$ para $B$ é uma relação especial que atribui a cada elemento $x$ pertencente a $A$ exatamente um elemento correspondente $y$ em $B$ (Iezzi et al., 2016a).

**Definição.** *Se $f$ é um conjunto de pares ordenados $(x, y)$ que define a função de $A$ em $B$, dizemos*

$$f : A \to B$$

*e se em $f$, $y \in B$ é imagem de $x \in A$, onde*

$$y = f(x).$$

*Temos que $A$ é o domínio de $f$ e $B$ é o contradomínio de $f$.*

### 4.4.1 Construção de gráficos de funções usando HTML

Como estamos trabalhando via console, torna-se uma tarefa difícil gerar gráficos. Mas o JavaScript, por ser uma linguagem web, possui ótima integração com o HTML, que é uma linguagem de marcação. Com isso, permite trabalhar visualmente e, para auxiliar, usaremos o **Plotly.js**, que é uma biblioteca gráfica poderosa e versátil para criar gráficos interativos e visualizações de dados diretamente no navegador.

O Plotly.js é uma biblioteca de gráficos para JavaScript de alto nível e é gratuito e de código aberto. Para mais informações, o site oficial `https://plotly.com/javascript/` explica com mais detalhes o de uso e configuração.

Antes de usarmos esta biblioteca, será feita uma breve introdução de HTML que será usada como base para o Plotly.js, e também devemos salvar os arquivos com códigos HTML na extensão '`.html`' caso deseje usar no Visual Studio Code ou em outro editor.

A visualização do gráfico pode ser feita via navegador web, em algum site de compilação ou usando uma extensão do Visual Studio Code. Temos a seguinte estrutura de código, com comentários:

```
1  <!-- Comentario em HTML -->
2  <!DOCTYPE html> <!-- Informa ao navegador que esta e uma pagina
       HTML5 -->
3  <html> <!-- Elemento raiz de uma pagina HTML, que contem todo o
       conteudo da pagina -->
4      <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
           <!-- Estamos incluindo a biblioteca Plotly.js na pagina
           atraves de um CDN (Content Delivery Network) -->
5      <body> <!-- Corpo da pagina, onde todo o conteudo visivel e
           colocado. -->
6          <div id="myPlot" style="width:100%;max-width:700px"></div>
               <!-- Atua como o conteiner para o grafico, e o id 'myPlot
               ' usa-se para identificar este elemento, e o estilo (
               style) CSS define a largura do conteiner do grafico. -->
7      <script> <!-- Dentro deste elemento, insere o codigo JavaScript
           necessario para criar e exibir o grafico usando Plotly.js. O
           codigo JavaScript vai interagir com a 'div' -->
8          // Codigo JavaScript...
9      </script>
10     </body>
11 </html>
12 <!-- Toda tag que e aberta '<exemplo>' deve ser fechada com </
       exemplo> para indicar o fim daquele elemento especifico -->
```

Código 4.13 – Exemplo de código HTML com JavaScript usando Plotly.js

A partir do segundo código de criação de gráfico em diante, serão exibidos apenas exemplos de blocos de códigos JavaScript, sem a inclusão do código HTML, já que serão as mesmas linhas de códigos em todos os casos.

### 4.4.2 Função afim

**Definição.** *A função afim (ou polinomial do 1º grau) é uma função* $f : \mathbb{R} \to \mathbb{R}$ *dado por*

$$f(x) = ax + b$$

*em que* $a$, $b \in \mathbb{R}$ *e* $a \neq 0$.

#### 4.4.2.1 Gráfico

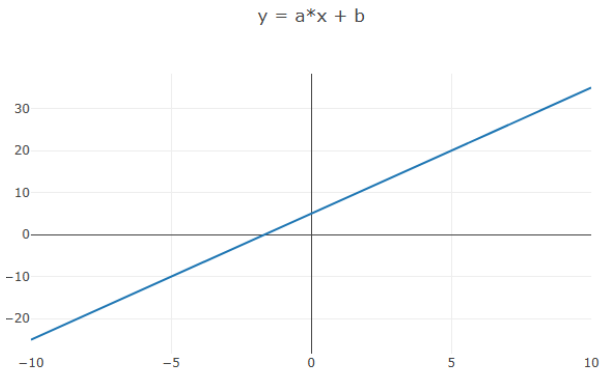
O código serve para calcular os valores da $f(x)$ em um intervalo fechado $[m, n]$ e a variação sendo determinadas no laço `for`.

```
<!DOCTYPE html>
<html>
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
<body>
<div id="myPlot" style="width:100%;max-width:700px"></div>
<script>
    // Funcao afim: f(x) = a^x + b
    const a = 3;
    const b = 5
    // Gerar valores
    const xValor = [];
    const yValor = [];
    for (let x = -10; x <= 10; x += 0.1) { //Intervalo [-10, 10],
        variacao 0.1
      xValor.push(x); //Adiciona um valor x ao array
      yValor.push(a*x + b); //Adiciona um valor f(x) ao array
    }
    //Comando serve para mostrar o grafico usando o Plotly
    const dados = [{x:xValor, y:yValor, mode:"lines"}];
    const layout = {title: "y = a*x + b"}; //Titulo acima do grafico
    Plotly.newPlot("myPlot", dados, layout);
</script>
</body>
</html>
```

Código 4.14 – Exemplo de gráfico para uma função afim em um intervalo

O gráfico gerado pelo exemplo acima será:

Figura 4.4.1 – Gráfico da função afim gerado pelo código 4.14




Fonte: Criada pelo autor, 2024.

#### 4.4.2.2 Raiz

A raiz de uma função afim é dada por $f(x) = ax + b$ e $a \neq 0$, com $x \in \mathbb{R}$ tal que $f(x) = 0$, temos:

$$f(x) = 0 \implies ax + b = 0 \implies x = -\frac{b}{a}.$$

Na função (`calculoRaizesAfim`) do código temos: Verifica se `a` é diferente de zero; o '`===`' é similar ao '`==`', mas ele exige que os valores e tipos sejam idênticos; e caso `a` $= 0$, `b` $\neq 0$, não tem raízes.

```
1     let cA = 3;
2     let cB = 10;
3     function calculoRaizesAfim(a, b) {
4         if (a !== 0) {
5             let raiz = -b / a;
6             return raiz;
7         } else if (b === 0) {
8             return 'A f(x) tem infinitas raizes.';
9         } else {
10            return 'A f(x) nao tem raizes.';
11        }
12    }
13    let raizAfim = calculoRaizesAfim(cA, cB);
14    console.log(`f(x) = (${cA})*x + (${cB}). Raiz: ${raizAfim}`)
```

Código 4.15 – Exemplo de cálculo da raiz uma função afim

### 4.4.3 Função Quadrática

A função quadrática (ou polinomial do 2º grau) é uma função $f : \mathbb{R} \to \mathbb{R}$ dado por

$$f(x) = ax^2 + bx + c$$

em que $a$, $b$, $c \in \mathbb{R}$ e $a \neq 0$.

#### 4.4.3.1 Gráfico

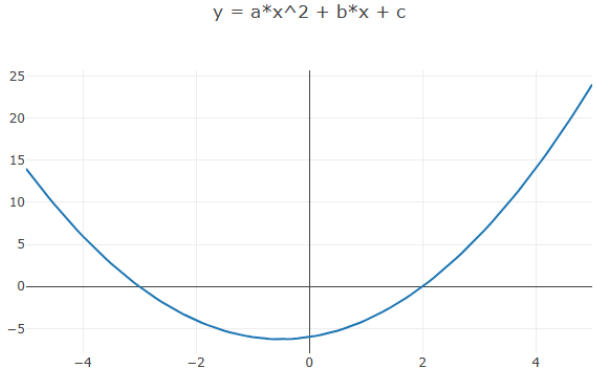
Análogo ao código anterior:

```
1     // Funcao quadratica: f(x) = a*x^2 + b*x + c
2     const a = 1;
3     const b = 1;
4     const c = -6;
5     const xValor = [];
6     const yValor = [];
7     for (let x = -5; x <= 5; x += 0.1) { //Intervalo [-5, 5], variacao
          0.1
8       xValor.push(x);
9       yValor.push(a*(x**2) + b*x + c);
10    }
11    const dados = [{x:xValor, y:yValor, mode:"lines"}];
12    const layout = {title: "y = a*x^2 + b*x + c"};
13    Plotly.newPlot("myPlot", dados, layout);
```

Código 4.16 – Exemplo de gráfico para uma função quadrática em um intervalo

O gráfico gerado pelo exemplo acima será:

Figura 4.4.2 – Gráfico da função quadrática gerado pelo código 4.16




Fonte: Criada pelo autor, 2024.

#### 4.4.3.2 Raiz

A raiz de uma função quadrática é dada por $f(x) = ax^2 + bx + c$ e $a \neq 0$, com $x \in \mathbb{R}$ tal que $f(x) = 0$, temos que as raízes (caso exista) de $y = ax^2 + bx + c = 0$:

$$
\begin{aligned}
f(x) &= 0 \\
ax^2 + bx + c &= 0 \\
a\left(x^2 + \frac{b}{a}x + \frac{c}{a}\right) &= 0 \\
x^2 + \frac{b}{a}x + \frac{c}{a} &= 0 \\
x^2 + \frac{b}{a}x &= -\frac{c}{a} \\
x^2 + \frac{b}{a}x + \frac{b^2}{4a^2} &= \frac{b^2}{4a^2} - \frac{c}{a} \\
\left(x + \frac{b}{2a}\right)^2 &= \frac{b^2 - 4ac}{4a^2} \\
x + \frac{b}{2a} &= \pm\frac{\sqrt{b^2 - 4ac}}{2a} \\
x &= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\end{aligned}
$$

e defina $\Delta = b^2 - 4ac$.

Em `calcularRaizesQuadraticas`, verificamos se a função possui duas raízes distintas, somente uma ou nenhuma.

```
1     let cA = 1;
2     let cB = -5;
3     let cC = 4;
4     function calcularRaizesQuadraticas(a, b, c) {
5         let delta = b ** 2 - 4 * a * c;
6         if (delta > 0) { //Verifica se tem duas raizes reais distintas.
7             let raiz1 = (-b + Math.sqrt(delta)) / (2 * a);
8             let raiz2 = (-b - Math.sqrt(delta)) / (2 * a);
9             return [raiz1, raiz2];
10        } else if (delta === 0) {
11            let raiz = -b / (2 * a); //Quando tem uma raiz real.
12            return raiz;
13        } else {
14            return "Nao-reais";
15        }
16    }
17    let raizQuadratica = calcularRaizesQuadraticas(cA, cB, cC);
18    console.log(`f(x) = (${cA})*x^2 + (${cB})x + (${cC}). Raizes: (${
          raizQuadratica}).`);
```

Código 4.17 – Exemplo de cálculo de raízes de uma função quadrática

### 4.4.4 Função Exponencial

A função exponencial é uma função $f : \mathbb{R} \to \mathbb{R}_+^*$ dado por

$$f(x) = a^x$$

em que $a \in \mathbb{R}$, $a > 0$ e $a \neq 1$.

#### 4.4.4.1 Gráfico

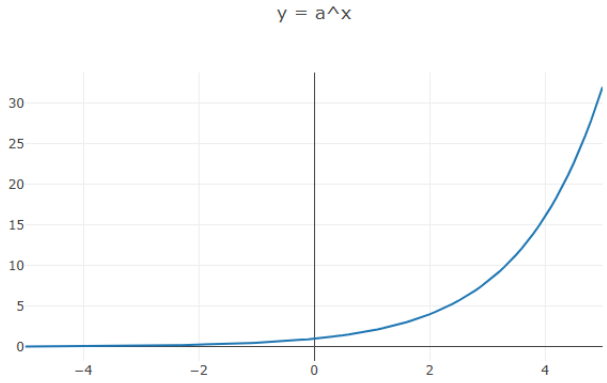
Analogamente:

```
1     //Funcao exponencial: f(x) = a^x
2     const a = 2;
3     const xValor = [];
4     const yValor = [];
5     for (let x = -5; x <= 5; x += 0.1) { //Intervalo [-5, 5], variacao
          0.1
6       xValor.push(x);
7       yValor.push(a**x);
8     }
9     const dados = [{x:xValor, y:yValor, mode:"lines"}];
10    const layout = {title: "y = a^x"};
11    Plotly.newPlot("myPlot", dados, layout);
```

Código 4.18 – Exemplo de gráfico para uma função exponencial em um intervalo

O gráfico gerado pelo exemplo acima será:

Figura 4.4.3 – Gráfico da função exponencial gerado pelo código 4.18




Fonte: Criada pelo autor, 2024.

### 4.4.5 Função Logarítmica

Com $a > 0$ e $a \neq 1$, chama-se função logarítmica de base $a$ a função $f : \mathbb{R}_+^* \to \mathbb{R}$, onde

$$f(x) = \log_a x.$$

#### 4.4.5.1 Gráfico

Também é calculado os valores da $f(x)$ em um intervalo fechado $[m, n]$ $(m, n \in \mathbb{Z})$. O `Math.log` calcula o logaritmo decimal, ou seja, logaritmo de base 10 ($\log_{10} x = \log x$) do valor declarado. A operação dentro de '`yValor.push`' serve para calcular logaritmos de qualquer base usando a propriedade de mudança de base (Iezzi et al., 2016a, p. 158):

**Proposição.** $\log_{b_1} M = \dfrac{\log_{b_2} M}{\log_{b_2} b_1}$

*Demonstração.* Tome $\begin{cases} m_1 = \log_{b_1} M \iff b_1^{m_1} = M \\ m_2 = \log_{b_2} M \iff b_2^{m_2} = M \\ \alpha = \log_{b_2} b_1 \iff b_2^{\alpha} = b_1 \end{cases}$.

E ainda, tem-se que

$$\begin{aligned} b_1^{m_1} &= M \\ (b_2^{\alpha})^{m_1} &= M = b_2^{m_2} \\ b_2^{\alpha m_1} &= b_2^{m_2} \\ \alpha m_1 &= m_2 \\ m_1 &= \frac{m_2}{\alpha} \end{aligned}$$

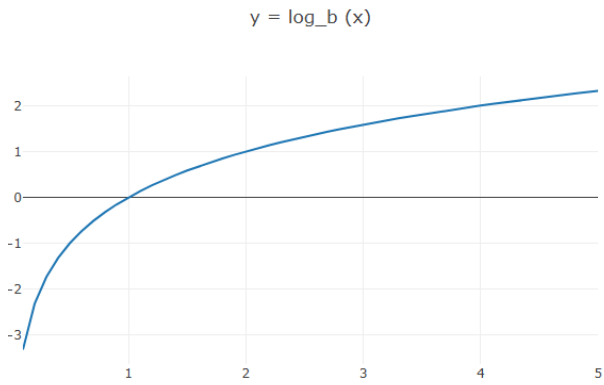
Sabendo os valores de $m_1$, $m_2$ e $\alpha$, logo: $\log_{b_1} M = \dfrac{\log_{b_2} M}{\log_{b_2} b_1}$. □

```
1     // Funcao logaritmica: f(x) = log_b (x)
2     const b = 2;
3     const xValor = [];
4     const yValor = [];
5     for (let x = 0.1; x <= 5; x += 0.1) { //Intervalo [0.1, 5], variacao
          0.1
6       xValor.push(x);
7       yValor.push(Math.log(x) / Math.log(b));
8     }
9     const dados = [{x:xValor, y:yValor, mode:"lines"}];
10    const layout = {title: "y = log_b (x)"};
11    Plotly.newPlot("myPlot", dados, layout);
```

Código 4.19 – Exemplo de gráfico para uma função logarítmica em um intervalo

O gráfico gerado pelo exemplo acima:

Figura 4.4.4 – Gráfico da função logarítmica gerado pelo código 4.19




Fonte: Criada pelo autor, 2024.

### 4.4.6 Função Modular

Uma função modular é uma função $f : \mathbb{R} \to \mathbb{R}$ que associa cada $x \in \mathbb{R}$ ao seu valor absoluto (módulo), ou seja, $f(x) = |x|$. A função modular é definida:

$$f(x) = \begin{cases} x, & \text{se } x \geq 0 \\ -x, & \text{se } x < 0 \end{cases}$$

#### 4.4.6.1 Gráfico

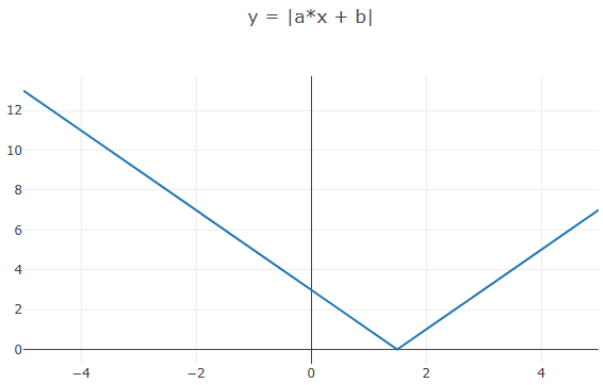
Analogamente aos exemplos de códigos anteriores. O `Math.abs` serve para calcular o módulo (ou valor absoluto):

```
1   //Funcao modular: f(x) = |a*x + b|
2   const a = -2;
3   const b = 3;
4   const xValor = [];
5   const yValor = [];
6   for (let x = -5; x <= 5; x += 0.1) { //Intervalo [-5, 5], variacao
        0.1
7     xValor.push(x);
8     yValor.push(Math.abs(a*x + b));
9   }
10  const dados = [{x:xValor, y:yValor, mode:"lines"}];
11  const layout = {title: "y = |a*x + b|"};
12  Plotly.newPlot("myPlot", dados, layout);
```

Código 4.20 – Exemplo de gráfico para uma função modular em um intervalo

O gráfico gerado pelo exemplo acima:

Figura 4.4.5 – Gráfico da função modular gerado pelo código 4.20




Fonte: Criada pelo autor, 2024.

## 4.5 TRIGONOMETRIA

A trigonometria é um ramo da matemática que estuda as relações entre os ângulos e os lados dos triângulos. No nosso contexto, ela é apresentada como um conjunto de funções matemáticas, como seno, cosseno e tangente, que são definidas em relação a um ângulo (Bosquilha; Amaral; Miranda, 2010).

A trigonometria é aplicada em diversos campos, como engenharia, física e astronomia, para resolver problemas que envolvem medidas de ângulos e distâncias. No ensino fundamental, é introduzida através do estudo das funções trigonométricas no triângulo retângulo.

### 4.5.1 Funções Trigonométricas

As funções trigonométricas são fundamentais para o estudo de triângulos e fenômenos periódicos. Elas incluem o seno, cosseno e tangente, cada uma relacionando ângulos a razões específicas entre lados de um triângulo retângulo.

a) **Seno** (sen): O seno de um ângulo agudo em um triângulo retângulo é a razão entre o comprimento do cateto oposto ao ângulo e o comprimento da hipotenusa.

O `Math.sin` retorna o valor do seno do ângulo em radianos, assim como `Math.PI` retorna um valor aproximado de pi ($\pi$).

```
1    let x = Math.PI / 2;
2    let seno = Math.sin(x);
3    console.log(`Seno de ${x}: ${seno}`);
```

Código 4.21 – Exemplo de cálculo do seno de um ângulo

b) **Cosseno** (cos): O cosseno de um ângulo agudo é a razão entre o comprimento do cateto adjacente ao ângulo e o comprimento da hipotenusa.

O `Math.cos` retorna o valor do cosseno do ângulo em radianos.

```
1      let x = Math.PI / 2;
2      let cosseno = Math.cos(x);
3      console.log(`Cosseno de ${x}: ${cosseno}`);
```

Código 4.22 – Exemplo de cálculo do cosseno de um ângulo

c) **Tangente** (tan): A tangente de um ângulo é a razão entre o seno e o cosseno desse ângulo, ou seja, a razão entre o cateto oposto e o cateto adjacente.

O `Math.tan` retorna o valor da tangente do ângulo em radianos.

```
1      let x = Math.PI / 2;
2      let tangente = Math.tan(x);
3      console.log(`Tangente de ${x}: ${tangente}`);
```

Código 4.23 – Exemplo de cálculo da tangente de um ângulo

### 4.5.2 Conversão para radianos e graus

Usaremos a fórmula

$$(a \text{ rad}) \cdot \frac{180}{\pi} = b^\circ$$

para a conversão de medida de um ângulo de radianos para graus.

Como o JavaScript retorna o valor da função trigonométrica pelo objeto `Math` em radianos, pode-se criar uma função que converta o valor para graus.

```
1  let rad = 3 * (Math.PI / 2);
2  let radianosParaGraus = rad * (180 / Math.PI);
3  console.log("Valor de radianos para graus: ",radianosParaGraus);
```

Código 4.24 – Exemplo de conversão de radianos para graus

## 4.6 ESTATÍSTICA

A estatística, um ramo vital da matemática, envolve-se com a coleta, análise, interpretação e exibição de dados. Sua importância transcende várias disciplinas, fornecendo uma base sólida para decisões informadas através de dados quantitativos. Além disso, serve como uma chave para compreender e manipular conjuntos de dados, visualizá-los de forma gráfica e discernir suas tendências e atributos significativos (Iezzi et al., 2016c).

O processo estatístico inicia-se com a coleta de dados, adquiridos via pesquisas ou experimentações. Posteriormente, esses dados são sistematizados em tabelas de frequência, que evidenciam a recorrência de cada observação.

### 4.6.1 Frequência absoluta

A frequência absoluta é um conceito fundamental na estatística descritiva e refere-se ao número de vezes que um determinado valor ou evento ocorre em um conjunto de dados. É uma contagem direta e não normalizada que indica quantas vezes um resultado específico é observado dentro do conjunto de dados analisado (Iezzi; Hazzan; Degenszajn, 2013).

O código lê um conjuntos de valores em um `array` (no exemplo chamado de `conjuntodeDados`) calcula a frequência que esses valores se repetem usando o laço `for`.

```
1   let conjuntoDeDados = [
2   "Lapis", "Caneta", "Caderno", "Borracha", "Livro", "Livro", "Lapis",
        "Lapis", "Caneta", "Lapis", "Caneta", "Borracha"
3   ];
4   function calcularFrequencia(dados) {
5     let frequencia = {}; // Objeto para armazenar as frequencias
6     for (let i = 0; i < dados.length; i++) { // Iteracao sobre o array
7       let elemento = dados[i]; // Obtem-se o elemento atual
8       if (frequencia[elemento]) { // Se o elemento estiver no objeto
            de frequencias, incrementa a contagem
9         frequencia[elemento]++;
10      } else { // Senao, adiciona-se com contagem 1
11        frequencia[elemento] = 1;
12      }
13    }
14    return frequencia;
15  }
16  let frequencias = calcularFrequencia(conjuntoDeDados);
17  console.log("-> Frequencia dos elementos no conjunto de dados <-");
18  for (let elemento in frequencias) {
19    console.log(`${elemento}: ${frequencias[elemento]} vezes.`);
20  }
```

Código 4.25 – Exemplo de cálculo de frequência absoluta de um 'array'

### 4.6.2 Média aritmética

Sejam $x_1, x_2, \ldots, x_n$ os valores observados de uma variável quantitativa $x$ e $n$ o número total de observações. A média aritmética, denotada por $\bar{x}$, é calculada pela fórmula (Iezzi et al., 2016c):

$$\bar{x} = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

Em `calcularMediaAritmetica`, o '`dados.lenght === 0`' verifica se o `array` de dados não está vazio; caso array for vazio, retornará a média zero; o método `reduce` calcula a soma de todos os elementos do `array`, com `total` sendo o acumulado, `valor` o valor atual do `array` que está sendo processado, e `0` o valor inicial do acumulador.

```
1   let notas = [7.5, 8.0, 8.3, 7.0];
2   function calcularMediaAritmetica(dados) {
3       if (dados.length === 0) {
4         return 0;
5       }
6       let soma = dados.reduce((total, valor) => total + valor, 0);
7       let media = soma / dados.length;
8       return media;
9   }
10  let mediaAritmetica = calcularMediaAritmetica(notas);
11  console.log(`A media das notas: ${mediaAritmetica}`);
```

Código 4.26 – Exemplo de cálculo da média aritmética de quatro notas

### 4.6.3 Média ponderada

Considere um conjunto de valores representados pelos elementos $x_1, x_2, \ldots, x_k$, que possuem frequências absolutas correspondentes de $n_1, n_2, \ldots, n_k$, respectivamente. A média aritmética ponderada desses valores é dada por:

$$\bar{x} = \frac{x_1 \cdot n_1 + x_2 \cdot n_2 + ... + x_k \cdot n_k}{n_1 + n_2 + ... + n_k}$$

Temos na função `calcularMediaPonderada` que `n` é a nota e `p` é o peso; a variável `somaProdutos` soma os produtos das notas pelo peso.

```
1  const notas = [4.5, 8.5, 8.0, 7.8];
2  const pesoNotas = [0.5, 0.3, 0.4, 0.3];
3  function calcularMediaPonderada(n, p) {
4      if (n.length !== p.length || n.length === 0) {
5          console.error("Notas e pesos devem ter a mesma quantidade e
               nao devem ser vazios!");
6          return null;
7      }
8      const somaProdutos = n.reduce((acumulador, n, indice) => {
9          return acumulador + n * p[indice];
10     }, 0); //O acumulador inicia com 0.
11     const somaPesos = p.reduce((acumulador, p) => acumulador + p, 0)
           ;
12     return somaProdutos / somaPesos; //Formula da media ponderada.
13 }
14 const mediaPonderada = calcularMediaPonderada(notas, pesoNotas);
15 console.log("Media ponderada:", mediaPonderada);
```

Código 4.27 – Exemplo de cálculo da média ponderada de quatro notas e seus pesos

### 4.6.4 Mediana

Considere que $x_1 \leq x_2 \leq \cdots \leq x_n$ sejam os $n$ valores ordenados assumidos por uma variável quantitativa $X$, em um conjunto de observações. A mediana ($Me$) é definida:

$$Me = \begin{cases} X\left(\frac{n+1}{2}\right), & \text{se } n \text{ for ímpar.} \\ \frac{X\left(\frac{n}{2}\right) + X\left(\frac{n}{2}+1\right)}{2}, & \text{se } n \text{ for par.} \end{cases}$$

A variável `dadosOrdenados` ordena os dados em ordem decrescente; o laço `if` e `else` calcula a mediana; e o '`...d`' passa cada elemento do `array` '`d`' como argumento separado para as funções.

```
const conjuntoDeDados = [2, 5, 1, 6, 9, 7, 3, 11];
function calcularMediana(d) {
    if (d.length === 0) {
        console.error("O conjunto de dados nao pode estar vazio!");
        return null;
    }
    const dadosOrdenados = [...d].sort((a, b) => a - b);
    const tamanho = dadosOrdenados.length;
    if (tamanho % 2 === 0) {
        const meio1 = dadosOrdenados[tamanho / 2 - 1];
        const meio2 = dadosOrdenados[tamanho / 2];
        return (meio1 + meio2) / 2;
    } else {
        return dadosOrdenados[tamanho / 2];
    }
}
const mediana = calcularMediana(conjuntoDeDados);
console.log("Mediana dos dados apresentados:", mediana);
```

Código 4.28 – Exemplo de cálculo da mediana de um 'array'

### 4.6.5 Moda

A moda ($Mo$) em um conjunto de dados é o número que aparece com maior recorrência, ou seja, é o valor com a maior frequência absoluta no conjunto.

O código verifica qual valor aparece com mais frequência em um `array`; a variável `frequencia` dentro da função `calcularModa` é um objeto que conta a frequência de cada valor.

```
const conjuntoDeDados = [2, 2, 3, 3, 4, 5, 5, 5, 6, 6];
function calcularModa(d) {
    if (d.length === 0) {
        console.error("O conjunto de dados nao pode estar vazio!");
        return null;
    }
    const frequencia = {};
    for (const valor of d) {
        frequencia[valor] = (frequencia[valor] || 0) + 1;
    }
    let moda = null;
    let maxFrequencia = 0;
    for (const valor in frequencia) { //Serve para encontrar valor
       com a maior frequencia
        if (frequencia[valor] > maxFrequencia) {
            moda = valor;
            maxFrequencia = frequencia[valor];
        }
    }
    return moda;
}
const moda = calcularModa(conjuntoDeDados);
console.log("Moda do conjunto de dados:",moda);
```

Código 4.29 – Exemplo que procura a moda de um 'array'

### 4.6.6 Amplitude

A amplitude de um conjunto de valores assumidos por uma variável quantitativa é o número real obtido pela diferença entre o maior e o menor valor registrados, nesta ordem.

Essa medida de dispersão é calculada com a fórmula:

$$\text{Amplitude} = \text{Maior Valor} - \text{Menor Valor}.$$

Temos que `Math.max` e `Math.min` retorna o valor máximo e mínimo, respectivamente.

```
1   const conjuntoDeDados = [5, 10, 15, 20, 25, 21, 22, 18, 26, 16];
2   function calcularAmplitude(d) {
3       if (d.length === 0) {
4         console.error("O conjunto de dados nao pode estar vazio!");
5         return null;
6       }
7       const maiorValor = Math.max(...d);
8       const menorValor = Math.min(...d);
9       return maiorValor - menorValor;
10  }
11  const amplitude = calcularAmplitude(conjuntoDeDados);
12  console.log("Amplitude dos dados apresentados:",amplitude);
```

Código 4.30 – Exemplo que calcula a amplitude de um 'array'

### 4.6.7 Variância

Considere $x_1, x_2, ..., x_n$ como os valores assumidos por uma variável quantitativa X, e $\bar{x}$ como a média aritmética desses valores. Para cada $x_i$ (onde $i = 1, 2, ..., n$), calculamos o quadrado da diferença entre $x_i$ e a média $\bar{x}$, ou seja, $(x_i - \bar{x})^2$, que é denominado desvio quadrático. A variância, representada por $\text{Var}(X)$ ou $\sigma^2$ (sigma ao quadrado), é definida como a média aritmética dos desvios quadráticos. Matematicamente, a variância é expressa por:

$$\sigma^2 = \frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + ... + (x_n - \bar{x})^2}{n}.$$

Calculamos pelo código usando a função `calcularMedia` novamente para a média aritmética; o '`somaDosQuadrados / d.length`' retorna a variância dividindo a soma dos quadrados das diferenças pelo número de elementos.

```
1   const conjuntoDeDados = [2, 5, 1, 6, 9, 7, 3, 11];
2   function calcularMedia(d) {
3       if (d.length === 0) {
4           console.error("O conjunto de dados nao pode estar vazio!");
5           return null;
6       }
7       const soma = d.reduce((acumulador, valor) => acumulador + valor,
            0);
8       return soma / d.length;
9   }
10  function calcularVariancia(d) {
11      if (d.length === 0) {
12          console.error("O conjunto de dados nao pode estar vazio!");
13          return null;
14      }
15      const media = calcularMedia(d);
16      const somaQuadradosDasDiferencas = d.reduce((acumulador, valor)
            => {
17          const diferenca = valor - media;
18          return acumulador + diferenca * diferenca;
19      }, 0);
20      return somaQuadradosDasDiferencas / d.length;
21  }
22  const variancia = calcularVariancia(conjuntoDeDados);
23  console.log("Variancia dos dados:", variancia);
```

Código 4.31 – Exemplo que calcula a variância de um 'array'

## 4.7 MATRIZES

Sejam $m, n \in \mathbb{N}$ e $m, n \neq 0$. Uma tabela de $m \cdot n$ números reais dispostos em $m$ linhas (horizontais) e $n$ colunas (verticais) é uma matriz do tipo (mesma ordem) $m \times n$, ou matriz $m \times n$ (Iezzi et al., 2016b).

### 4.7.1 Declaração e acesso de matrizes

Representa-se uma matriz A quaisquer como:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}_{m \times n}$$

No JavaScript, declarar uma matriz é muito similar a declarar um `array`, é como por exemplo declarar um `arrays` dentro de um outro `array` como veremos no exemplo do código a seguir.

Assim como no `array`, o primeiro elemento começa com o valor 0, logo tome em

uma matriz onde o elemento da primeira linha e primeira coluna é $a_{00}$, ou seja:

$$A = \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0(n-1)} \\ a_{10} & a_{11} & \cdots & a_{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(m-1)0} & a_{(m-1)1} & \cdots & a_{(m-1)(n-1)} \end{pmatrix}_{(m-1)\times(n-1)}$$

No código, cada `array` dentro dos colchetes de `matrizExemplo` é uma linha da matriz. Observe que neste exemplo é uma matriz $3 \times 3$. A matriz completa é impressa no console em uma única linha.

```
1    const matrizExemplo = [
2        [1, 2, 3],
3        [4, 5, 6],
4        [7, 8, 9]
5    ];
6    console.log(matrizExemplo);
```

Código 4.32 – Exemplo de declaração de matriz

O processo de acessar os valores de uma matriz é similar a um `array`, onde no código a variável `acessoMatriz` realiza o acesso com '`matrizExemplo[0][2]`'; sendo a linha e coluna o primeiro valor a posição da linha inserido, respectivamente. Ou seja, está sendo chamado o elemento da primeira linha e terceira coluna ($a_{02}$) de `matrizExemplo`, ou o elemento $a_{13}$ desta matriz de ordem $3 \times 3$, logo retornará o valor 3 no console.

```
1    const matrizExemplo = [
2        [1, 2, 3],
3        [4, 5, 6],
4        [7, 8, 9]
5    ];
6    const acessoMatriz = matrizExemplo[0][2];
7    console.log(acessoMatriz);
```

Código 4.33 – Exemplo de acesso de valores em uma matriz

### 4.7.2 Transposição de matriz

Considerando uma matriz qualquer $A$, com $A = (a_{ij})_{m\times n}$, temos $A^t$ que é **matriz transposta** de A, onde:

$$A^t = (a'_{ji})_{n\times m}$$

tal que $a'_{ji} = a_{ij}$, para todo $i$ e $j$.

No código, o método `map` serve para criar um novo `array`, aplicando uma função a cada elemento do `array`, não modificando o `array` original, mas cria um novo `array` com os resultados da função aplicada a cada elemento. . Para cada índice `i`, criamos um novo

`array '(linha => linha[i])'`, que contém os elementos de cada linha da matriz original na posição `i`. Isso efetivamente cria a matriz transposta.

```
1     const matrizOriginal = [
2         [1, 3, 5],
3         [7, 9, 11]
4     ];
5     const transporMatriz = (matriz) => { // Funcao para transpor a
          matriz
6         //Vai mapear cada elemento da primeira linha da matriz original
7         return matriz[0].map((coluna, i) =>
8             // Para cada coluna, retorna um novo array contendo os
                  elementos correspondentes de cada linha
9             matriz.map(linha => linha[i])
10        );
11    };
12    const matrizTransposta = transporMatriz(matrizOriginal);
13    console.log("Matriz original:", matrizOriginal);
14    console.log("Matriz transposta:", matrizTransposta);
```

Código 4.34 – Exemplo de transposição de uma matriz

### 4.7.3 Adição e subtração de matrizes

a) **Adição**: Dado matrizes $A$ e $B$ de mesma ordem, $A = (a_{ij})_{m\times n}$ e $B = (b_{ij})_{m\times n}$, a soma de $A$ com $B$ $(A + B)$ é a matriz $C$, com $C = (c_{ij})_{m\times n}$, em que $c_{ij} = a_{ij} + b_{ij}$, para $1 \leq i \leq m$ e $1 \leq j \leq n$.

A ideia deste código é receber duas matrizes como parâmetro e usar dois laços para percorrer cada elemento das matrizes. Observe que em `adicaoDeMatrizes`, o `i` é o índice das linhas e `j` índices da coluna.

```
const matrizA = [
    [1, 3],
    [5, 7]
];
const matrizB = [
    [2, 4],
    [6, 8]
];
function adicaoDeMatrizes(m1, m2) {
    const resultado = [];
    for (let i = 0; i < m1.length; i++) {
        const linhaDeResultado = [];
        for (let j = 0; j < m1[i].length; j++) {
            linhaDeResultado.push(m1[i][j] + m2[i][j]);
        }
        resultado.push(linhaDeResultado);
    }
    return resultado;
}
console.log("Resultado da adicao:",adicaoDeMatrizes(matrizA, matrizB
    ));
```

Código 4.35 – Exemplo de adição de matrizes

b) **Subtração**: Dado matrizes $A$ e $B$ de mesma ordem, $A = (a_{ij})_{m\times n}$ e $B = (b_{ij})_{m\times n}$, a diferença entre $A$ com $B$ $(A - B)$ é a matriz soma de $A$ com a oposta de $B$, ou seja, $A - B = A + (-B)$.

Este código é análogo ao da soma, mas resultando na subtração entre os elementos de `matrizA` e `matrizB`.

```
const matrizA = [
    [1, 3],
    [5, 7]
];
const matrizB = [
    [2, 4],
    [6, 8]
];
function subtracaoDeMatrizes(m1, m2) {
    const resultado = [];
    for (let i = 0; i < m1.length; i++) {
      const linhaDeResultado = [];
        for (let j = 0; j < m1[i].length; j++) {
            linhaDeResultado.push(m1[i][j] - m2[i][j]);
        }
        resultado.push(linhaDeResultado);
    }
    return resultado;
}
console.log("Resultado da subtracao:",subtracaoDeMatrizes(matrizA,
    matrizB));
```

Código 4.36 – Exemplo de diferença entre matrizes

### 4.7.4 Multiplicação de matrizes

Dado matrizes $A$ e $B$ de mesma ordem, $A = (a_{ij})_{m \times n}$ e $B = (b_{jk})_{n \times p}$, o produto de A por B $(A \cdot B)$ é a matriz $C$, com $C = (c_{ik})_{m \times p}$, em que

$$c_{ik} = a_{i1} \cdot b_{1k} + a_{i2} \cdot b_{2k} + a_{i3} \cdot b_{3k} + ... + a_{in} \cdot b_{nk};$$

para todo $i \in \{1, 2, ..., m\}$ e todo $k \in \{1, 2, ..., p\}$:

1. Tome os $n$ elementos em ordem da linha $i$ da matriz $A$: $a_{i1}, a_{i2}, ..., a_{in}$; (I)
2. Tome os $n$ elementos em ordem da coluna $i$ da matriz $B$: $b_{1k}, a_{2k}, ..., a_{in}$; (II)
3. Multiplica-se o primeiro elemento de (I) pelo primeiro elemento, o segundo de (I) pelo segundo de (II), e assim sucessivamente até o último elemento de (I) pelo último de (II);
4. Somando os produtos obtidos, temos:

$$c_{ik} = a_{i1} \cdot b_{1k} + a_{i2} \cdot b_{2k} + ... + a_{in} \cdot b_{nk}.$$

Em '`if (m1[0].length !== m2.length)`', será realizado a verificação se matrizes podem ser multiplicadas. O código inclui vários laços `for` desde realizar a inicialização da matriz resultado até realizando a multiplicação de seus elementos.

```
1     const matrizA = [
2         [1, 3],
3         [5, 7],
4         [9, 11]
5     ];
6     const matrizB = [
7         [2, 4, 6],
8         [8, 10, 12]
9     ];
10    function multiplicacaoDeMatrizes(m1, m2) {
11        const resultado = [];
12        if (m1[0].length !== m2.length) {
13            console.error("O numero de colunas de uma matriz e diferente
                  do numero de linhas da outra matriz!");
14            return resultado;
15        }
16        for (let i = 0; i < m1.length; i++) {
17            resultado[i] = [];
18        }
19        for (let i = 0; i < m1.length; i++) {
20            for (let j = 0; j < m2[0].length; j++) {
21                let soma = 0;
22                for (let k = 0; k < m2.length; k++) {
23                    soma += m1[i][k] * m2[k][j];
24                }
25            resultado[i][j] = soma;
26          }
27        }
28        return resultado;
29    }
30    const matrizResultadoMultiplicacao = multiplicacaoDeMatrizes(matrizA
          , matrizB);
31    console.log("Resultado da multiplicacao das matrizes:",
          matrizResultadoMultiplicacao);
```

Código 4.37 – Exemplo de diferença entre matrizes

### 4.7.5 Determinante de uma matriz $2 \times 2$

O número real $ad - bc$ da matriz $2 \times 2$:

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

é a **determinante** (det) da matriz incompleta M dos coeficientes do sistema. Tem-se que

$$\det M = a \cdot d - b \cdot c \text{ ou } \begin{vmatrix} a & b \\ c & d \end{vmatrix}.$$

A funcão `calcularDeterminante` calcula a determinante usando a fórmula '`m[0][0] * m[1][1] - m[0][1] * m[1][0]`', que seleciona a posição dos elementos para a operação. Isso é equivalente a calcular $(ad - bc)$ para uma matriz $2 \times 2$.

```
    const matriz = [
        [2, 4],
        [6, 8]
    ];
    function calcularDeterminante(m) {
        const determinante = m[0][0] * m[1][1] - m[0][1] * m[1][0];
        return determinante;
    }
    console.log("A determinante da matriz",matriz,":",
       calcularDeterminante(matriz));
```

Código 4.38 – Exemplo de cálculo de determinante de uma matriz 2X2

## 4.8 PROBABILIDADE

A probabilidade é um conceito matemático que mede a chance de um evento ocorrer. Ela é expressa numericamente entre 0 e 1, onde 0 indica impossibilidade e 1 indica certeza absoluta. Um experimento aleatório é aquele cujo resultado não pode ser previsto com certeza, apesar de todos os possíveis resultados serem conhecidos (Iezzi et al., 2016b).

### 4.8.1 Lançamento de moeda

O lançamento de uma moeda é um exemplo comum de um experimento aleatório em probabilidade. Quando lançamos uma moeda honesta, não viciada, temos dois possíveis resultados: cara ou coroa. Cada um desses resultados é chamado de evento elementar.

A função `lanceMoeda` gera um valor aleatório entre 0 e 1 (`Math.random`). Se o resultado for menor que $0,5$, é considerado cara e senão, coroa. No laço `for`, é simulado o lançamento `i` vezes da moeda, onde irá imprimir no console cada um dos resultados desse lançamento.

```
    function lanceMoeda() {
        const resultado = Math.random();
        const lado = resultado < 0.5 ? 'Cara' : 'Coroa';
        return lado;
    }
    for (let i = 1; i <= 5; i++) { // Simulado 5 lancamentos
        console.log(`Lance ${i}: ${lanceMoeda()}`);
    }
```

Código 4.39 – Simulação de lançamentos de uma moeda

### 4.8.2 Espaço amostral de dados

O espaço amostral ($\Omega$) é o conjunto de todos os resultados possíveis de um experimento aleatório. Por exemplo, o lançamento de um dado honesto e não viciado tem seu espaço amostral $\Omega = \{1, 2, 3, 4, 5, 6\}$.

Temos um laço `for`, onde cada par de resultados forma um elemento do espaço amostral.

```
1   const dado = [1, 2, 3, 4, 5, 6];
2   function espacoAmostralDoisDados(d) {
3       const espacoAmostral = [];
4       for (const resultado1 of d) {
5           for (const resultado2 of d) {
6               espacoAmostral.push([resultado1, resultado2]);
7           }
8       }
9       return espacoAmostral;
10  }
11  console.log("Espaco Amostral de dois dados:",espacoAmostralDoisDados
        (dado));
```

Código 4.40 – Exemplo de espaço amostral de lançamento simultâneo de dois dados

### 4.8.3 Frequência relativa

A frequência relativa ($K$) é uma medida que indica a proporção de vezes que um evento ocorre em relação ao número total de tentativas ou observações. É calculada pela razão entre o número de ocorrências de um evento e o número total de lançamentos ou observações.

Temos que '`d.forEach((elemento) => {`' conta a frequência de cada elemento, com `forEach` executando a função `arrow` em cada elemento de um `array`. No laço `for`, a frequência relativa será calculada.

```
1   const conjuntoDeDados = [1, 1, 2, 2, 2, 3, 4, 4, 5, 5, 5, 5, 6];
2   function calcularFrequenciaRelativa(d) {
3       const frequenciaRelativa = {};
4       d.forEach((elemento) => {
5           frequenciaRelativa[elemento] = (frequenciaRelativa[elemento]
                || 0) + 1;
6       });
7       for (const elemento in frequenciaRelativa) {
8           frequenciaRelativa[elemento] /= d.length * 100;
9       }
10      return frequenciaRelativa;
11  }
12  console.log("Resultado: ",calcularFrequenciaRelativa(conjuntoDeDados
        ));
```

Código 4.41 – Exemplo de cálculo de frequência relativa

# 5 JUSTIFICATIVA

Devido à estreita relação entre as áreas de Tecnologia da informação (TI) e as ciências exatas, a programação emerge como um complemento valioso para a matemática. Ela serve não só como uma ferramenta metodológica de ensino, mas também como meio para resolver problemas complexos. É importante considerar a crescente demanda do mercado de TI e o fato de que a programação está sendo cada vez mais implementada em todos os setores da sociedade. Isso exige que os alunos do ensino superior, técnico e, em alguns casos, até do ensino básico, estejam constantemente atualizados nesta área.

A reforma do ensino médio brasileiro, guiada pela Base Nacional Comum Curricular (BNCC), representa um avanço significativo na preparação dos jovens para um mercado de trabalho cada vez mais orientado para a tecnologia. A inclusão da programação como parte dos itinerários formativos não apenas atende às demandas do mercado de TI, mas também enriquece o aprendizado em ciências exatas, oferecendo uma abordagem interdisciplinar e aplicada.

A escolha da linguagem de programação JavaScript deve-se à sua facilidade e acessibilidade, sendo uma linguagem predominantemente voltada para a web. Além disso, o ambiente de programação está disponível nos principais sistemas operacionais de computadores domésticos (Windows, Linux e MacOS), dispositivos móveis (Android e iOS) e até mesmo em navegadores de internet.

# 6 OBJETIVOS

## 6.1 GERAL

Apresentar o conceito de algoritmos e linguagem de programação para os estudantes do ensino médio e técnico e até professores, ao mesmo tempo utilizar fórmulas, operações, conceitos e funções através do JavaScript para compreender tanto a linguagem, quanto estas propriedades da matemática.

## 6.2 ESPECÍFICOS

Apresentar a instalação da IDE (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado). O Visual Studio Code em conjunto com o Node.js será usado como base neste projeto (apesar de ser possível usar outras IDE), configurar o ambiente para o aprendizado através deste projeto. Também é necessário entender as ideias básicas e introdutórias de algoritmos, classes, orientação a objetos, métodos e funções da linguagem JavaScript.

Após a aplicação das ferramentas de JavaScript citadas anteriormente, compreender e testar os códigos ao utilizar operações, funções e equações matemáticas para exercitar a prática de programação dos educandos e auxiliar os docentes.

# 7 REFERENCIAL TEÓRICO

No ponto de vista metodológico, as tecnologias da informação são poderosas ferramentas no processo de aprendizagem do estudante, se o professor souber se aproveitar desses recursos. Utilizando-se da linguagem de programação como auxiliar de aprendizado para a área de modelagem matemática, é visto que, durante a execução da proposta do ambiente de pesquisa, o desenvolvimento de uma simulação realizada através da programação de computadores entregou uma representação nova para a tarefa de modelagem, como o exemplo da simulação de semáforos (Carvalho; Klüber, 2021, p. 321). Observe que o mesmo pode ser aplicado as áreas de lógica, aritmética e equações, tendo em vista que muitos dos recursos de programação exigem algum pré-requisito em matemática para se ter um melhor aprofundamento (Cormen, 2014).

A base da escolha da linguagem JavaScript para a realização deste projeto, é o fato que é uma linguagem principal para trabalhar com aplicações web, como softwares e sites, e soma-se o fato de ter uma enorme comunidade de desenvolvedores que contribuem para deixar a linguagem acessível e atualizada (Souza, 2019).

# 8 MÉTODOS E PROCEDIMENTOS

Primeiramente, podem ser utilizadas tecnologias básicas voltadas a informática, como computador (ou *smartphones*) e internet, sabendo-se que alguns alunos e professores possuem acesso a essas ferramentas. Também algumas escolas do ensino básico e técnico podem fornecem laboratórios de informática para aqueles que não possuem estas tecnologias disponíveis.

No projeto, a abordagem será feita de forma simplificada de tal modo que não será necessário ter experiência prévia em linguagem de programação. É recomendável ter o mínimo de conhecimento em informática básica, como o uso mínimo do sistema operacional por exemplo, porém é possível realizar a proposta caso não tenha este pré-requisito, com o auxílio de um docente ou discente com a devida experiência prévia.

# 9 CONCLUSÃO

A conclusão deste Trabalho de Conclusão de Curso sobre a interseção entre matemática e programação com JavaScript ressalta os benefícios teóricos da integração dessas duas áreas no ensino básico e técnico. Ao longo deste estudo teórico, foi discutido como o uso do JavaScript pode potencialmente facilitar a compreensão de conceitos matemáticos, tornando o aprendizado mais interativo e acessível.

Através da discussão de exemplos e exercícios práticos, podemos analisar que o JavaScript permitir aos estudantes visualizar e interagir com os conceitos matemáticos de forma mais dinâmica. Isso pode melhorar o entendimento e aumentar o interesse dos alunos pela matemática e programação.

Além disso, é importante considerar a integração da programação ao currículo escolar, devido ao papel crescente da tecnologia na sociedade contemporânea. A inclusão de linguagens de programação como JavaScript nas aulas de matemática pode proporcionar aos alunos habilidades essenciais para o mercado de trabalho e para a vida acadêmica futura.

O novo ensino médio influencia positivamente o estudo da programação. A flexibilização do currículo permite que os alunos escolham itinerários formativos que incluam disciplinas de tecnologia, favorecendo uma formação mais completa e alinhada às demandas atuais. Essa reforma possibilita a criação de espaços para que disciplinas como matemática e programação se complementem, promovendo um aprendizado interdisciplinar que melhor prepara os alunos para os desafios futuros.

Para estudos futuros, sugere-se a exploração de outras linguagens de programação e suas possíveis aplicações no ensino de diferentes disciplinas. A expansão do uso de tecnologias educacionais pode abrir novas possibilidades para métodos de ensino mais eficazes e adaptados às necessidades dos estudantes.

Concluindo: este trabalho contribui para a compreensão de como a programação pode ser uma aliada poderosa no ensino de matemática, propondo uma abordagem inovadora e prática que pode ser adotada em diversos contextos educacionais. Acredita-se que essa integração pode transformar o aprendizado, tornando-o mais significativo e atrativo para os alunos.

# REFERÊNCIAS BIBLIOGRÁFICAS


BOSQUILHA, Alessandra; AMARAL, João Tomás do; MIRANDA, Mônica. **Manual compacto de matemática**: ensino fundamental. 1. ed. São Paulo: Rideel, 2010. ISBN 978-85-339-1661-6.

CABRAL, Luiz Cláudio. **Matemática básica explicada passo a passo**. Rio de Janeiro: Elvesier, 2013.

CARVALHO, Felipe José Rezende; KLÜBER, Tiago Emanuel. Modelagem Matemática e Programação de Computadores: uma possibilidade para a Construção de Conhecimento na Educação Básica. **Educação Matemática Pesquisa**, v. 23, n. 1, p. 297–323, 2021. Disponível em: `https://revistas.pucsp.br/index.php/emp/article/view/50880`. Acesso em: 17 jan. 2024.

CORMEN, Thomas H. **Desmistificando algoritmos**. 1. ed. Rio de Janeiro: Elsevier, 2014.

DOLCE, Osvaldo; POMPEO, José Nicolau. **Fundamentos de Matemática Elementar 9**: geometria plana. 9. ed. São Paulo: Atual Editora, 2013. ISBN 978-85-357-1686-3.

FUCUSHIMA, Ana Ayumi; MARQUES, Ana Paula Ambrosio Zanelato; PARRÃO, Juliene Aglio O. Revisão da literatura sobre usabilidade e acessibilidade em ambiente web. **Revista de Ensino, Pesquisa e Extensão**, v. 2, n. 1, p. 1–15, 2020. Disponível em: `http://intertemas.toledoprudente.edu.br/index.php/ETIC/article/view/72171`. Acesso em: 17 jan. 2024.

GITHUB. **GitHub Docs**. [S.l.]. Disponível em: `https://docs.github.com/pt`. Acesso em: 25 maio 2024.

IEPSEN, Edécio Fernando. **Lógica de Programação e Algoritmos com JavaScript**: uma introdução à programação de computadores com exemplos e exercícios para iniciantes. 2. ed. São Paulo: Novatec, 2022. ISBN 978-65-86057-91-1.

IEZZI, Gelson et al. **Matemática**: ciência e aplicações, volume 1. 9. ed. São Paulo: Saraiva, 2016. ISBN 978-85-472-0536-2.

IEZZI, Gelson et al. **Matemática**: ciência e aplicações, volume 2. 9. ed. São Paulo: Saraiva, 2016. ISBN 978-85-472-0538-6.

IEZZI, Gelson et al. **Matemática**: ciência e aplicações, volume 3. 9. ed. São Paulo: Saraiva, 2016. ISBN 978-85-472-0540-9.

IEZZI, Gelson; DOLCE, Osvaldo; MURAKAMI, Carlos. **Fundamentos de Matemática Elementar 2**: logaritmos. 10. ed. São Paulo: Atual Editora, 2013. ISBN 978-85-357-1682-5.

IEZZI, Gelson; HAZZAN, Samuel; DEGENSZAJN, David Mauro. **Fundamentos de Matemática Elementar 11**: matemática comercial, matemática financeira, estatística descritiva. 9. ed. São Paulo: Atual Editora, 2013. ISBN 978-85-357-1760-0.

MANZANO, José Augusto N.G.; OLIVEIRA, Jayr Figueiredo de. **Algoritmos**: lógica para desenvolvimento de programação de computadores. 9. ed. São Paulo: Atual Editora, 2013.

MICROSOFT. **Documentation for Visual Studio Code**. [S.l.]. Disponível em: `https://code.visualstudio.com/Docs`. Acesso em: 17 jan. 2024.

MOREIRA, Carlos Gustavo. MDC, MMC, Algoritmo de Euclides e o Teorema de Bachet-Bézout. **Polos Olímpicos de Treinamento Intensivo**, 2012. Disponível em: `https://poti.impa.br/uploads/material_teorico/4qdovup0ktgk4.pdf`. Acesso em: 25 jan. 2024.

MOZILLA. **JavaScript**: Aprendendo desenvolvimento web. [S.l.]. Disponível em: `https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript`. Acesso em: 17 jan. 2024.

NODE.JS. **Index | Node.js v22.2.0 Documentation**. [S.l.]. Disponível em: `https://nodejs.org/docs/latest/api/`. Acesso em: 2 jun. 2024.

PLOTLY. **Plotly javascript graphing library in JavaScript**. [S.l.]. Disponível em: `https://plotly.com/javascript/`. Acesso em: 13 jun. 2024.

POLLOCK, John. **JavaScript**: A Beginner's Guide. [S.l.]: McGraw Hill Professional, 2009.

SARTIM, Ademir. **Matemática básica**: volume 1. Vitória: EDUFES. (Série Didáticos, 3). ISBN 978-65-88077-48-1. Disponível em: `http://repositorio.ufes.br/handle/10/774`. Acesso em: 17 jan. 2024.

SARTIM, Ademir. **Matemática básica**: volume 2. Vitória: EDUFES. (Série Didáticos, 4). ISBN 978-65-88077-44-3. Disponível em: `http://repositorio.ufes.br/handle/10/774`. Acesso em: 17 jan. 2024.

SOUZA, Ivan de. **JavaScript**: o que é, como funciona e por que usá-lo no seu site. [S.l.], 2019. Disponível em: `https://rockcontent.com/br/blog/javascript/`. Acesso em: 18 jan. 2024.