UNIVERSIDADE FEDERAL DO CARIRI
CENTRO DE CIÊNCIAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA EM REDE NACIONAL

JOSÉ JAILTON AZEVEDO DO NASCIMENTO

EXPLORANDO CADERNOS DIGITAIS NO ENSINO DE MATEMÁTICA BÁSICA COM JULIA

JUAZEIRO DO NORTE
2025

JOSÉ JAILTON AZEVEDO DO NASCIMENTO

EXPLORANDO CADERNOS DIGITAIS NO ENSINO DE MATEMÁTICA BÁSICA COM JULIA

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Matemática em Rede Nacional do Centro de Ciências e Tecnologia da Universidade Federal do Cariri, como parte dos requisitos necessários à obtenção do título de Mestre em Matemática. Área de concentração: Matemática na Educação Básica.

Orientador: Prof. Dr. Vicente Helano Feitosa Batista Sobrinho

JUAZEIRO DO NORTE
2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Cariri
Sistema de Bibliotecas

N322e Nascimento, José Jailton Azevedo do.
Explorando Cadernos Digitais no Ensino de Matemática Básica com Julia / José Jailton Azevedo do Nascimento. – 2025.
108 p. (Inclui bibliografia, p. 105–108).

Dissertação (Mestrado) – Universidade Federal do Cariri, Programa de Pós-graduação em Matemática em Rede Nacional(PROFMAT), Juazeiro do Norte, 2025.

Orientador: Prof. Dr. Vicente Helano Feitosa Batista Sobrinho

1. Matemática básica. 2. Jupyter Notebook. 3. Julia. I. Batista Sobrinho, Vicente Helano Feitosa – orientador. II. Título.

CDD 510

Bibliotecário: João Bosco Dumont do Nascimento (CRB 3/1355)

JOSÉ JAILTON AZEVEDO DO NASCIMENTO

EXPLORANDO CADERNOS DIGITAIS NO ENSINO DE MATEMÁTICA BÁSICA COM JULIA

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Matemática em Rede Nacional do Centro de Ciências e Tecnologia da Universidade Federal do Cariri, como parte dos requisitos necessários à obtenção do título de Mestre em Matemática. Área de concentração: Matemática na Educação Básica.

Aprovada em: 26 de agosto de 2025.

BANCA EXAMINADORA

gov.br Documento assinado digitalmente
VICENTE HELANO FEITOSA BATISTA SOBRINHO
Data: 08/09/2025 10:28:06-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Vicente Helano Feitosa Batista Sobrinho
UFCA

gov.br Documento assinado digitalmente
VALDINES LEITE DE SOUSA JUNIOR
Data: 08/09/2025 11:04:35-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Valdinês Leite de Sousa Junior
UFCA

gov.br Documento assinado digitalmente
RAFAEL PERAZZO BARBOSA MOTA
Data: 08/09/2025 10:56:39-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Rafael Perazzo Barbosa Mota
UFRPE

Dedico este trabalho ao meu eu criança e ao meu eu jovem. À criança que acreditava ser possível aprender tudo sobre matemática, e ao jovem que, ao descobrir a vastidão infinita dessa ciência, ainda assim não deixou de sonhar com os caminhos que ela poderia revelar. Sei que ambos se sentiriam orgulhosos ao ver até onde conseguimos chegar.

Agradecimentos

Agradeço, com profunda gratidão, à minha família, em especial à minha esposa, Camila Araújo, pelo apoio constante, paciência e suporte incondicional ao longo de toda esta jornada, e à minha mãe, Valdênia Azevedo, pelo incentivo permanente e por sempre acreditar em meu potencial.

Manifesto meus agradecimentos aos colegas de trabalho, em especial à diretora e amiga Maria Alves, por seu apoio dentro das possibilidades e pela colaboração contínua durante o desenvolvimento deste trabalho.

Agradeço também aos colegas de curso Ana Beatriz, André Marcos, Fábio, Gerlane, Gevanilson, Hermano, João Cipriano, Valéria, Valter, Wilia, Wesley, Ronivon e ao amigo Railo Calvance, por contribuírem para um ambiente de troca, companheirismo e incentivo mútuo. Destaco, em particular, as valiosas horas de estudo compartilhadas com Railo.

Aos professores e demais membros do Departamento de Matemática da Universidade Federal do Cariri, expresso minha admiração e respeito pela excelência acadêmica e pela postura ética e humanizada com que conduzem sua missão docente.

Agradeço, de maneira especial, ao meu orientador, Professor Vicente Helano, pela orientação criteriosa, pelas contribuições valiosas e pelo apoio intelectual prestado durante todo o desenvolvimento deste trabalho.

Este trabalho foi realizado com apoio financeiro da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), no âmbito do Programa de Mestrado Profissional em Matemática em Rede Nacional (Profmat) da Universidade Federal do Cariri. Agradeço à CAPES pelo suporte concedido, essencial para a continuidade dos meus estudos; sem esse auxílio, teria sido significativamente mais difícil manter minha dedicação ao curso e concluir esta etapa acadêmica.

RESUMO

Este trabalho tem como objetivo consolidar objetos de conhecimento da Base Nacional Comum Curricular alinhando-os às competências, habilidades e descritores da matriz de referência do Sistema Permanente de Avaliação da Educação Básica do Ceará, com foco nas turmas do 9º ano do Ensino Fundamental. Como estratégia metodológica, será adotada a resolução de problemas em conjunto com o uso de recursos computacionais. Para isso, serão utilizados cadernos digitais desenvolvidos no ambiente Jupyter Notebook, programados com a linguagem Julia. A proposta contempla o desenvolvimento de sequências didáticas que auxiliam no aprendizado de conteúdos matemáticos fundamentais, explorando conceitos como álgebra, números e geometria. A integração entre linguagens de programação e fundamentos matemáticos possibilita a criação de ambientes interativos e exploratórios, nos quais os alunos podem testar conjecturas, visualizar conceitos abstratos e desenvolver o pensamento lógico-matemático de forma autônoma e efetiva.

Palavras-chave: Matemática básica. Jupyter Notebook. Julia.

ABSTRACT

This study aims to consolidate the mathematical knowledge outlined in the Brazilian National Common Curricular Base, integrating it with the competencies, skills, and descriptors from the reference matrix of the Permanent System for the Evaluation of Basic Education of Ceará, with a focus on 9th-grade Elementary School classes. As a methodological approach, problem-solving will be combined with the use of technological resources. Specifically, digital notebooks developed in the Jupyter Notebook environment and programmed in Julia will be employed. The proposal encompasses the design of instructional sequences that facilitate the learning of core mathematical concepts in algebra, numbers, and geometry. The integration of programming and mathematics fosters interactive and exploratory learning environments in which students can test conjectures, visualize abstract concepts, and develop logical-mathematical reasoning in an autonomous and effective manner.

Keywords: Basic Mathematics. Jupyter Notebook. Julia.

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas

BNCC	Base Nacional Comum Curricular, p. 3
CMD	Prompt de Comando, p. 14
IDE	Integrated Development Environment, p. 13
PCNs	Parâmetros Curriculares Nacionais, p. 1
SAEB	Sistema de Avaliação da Educação Básica, p. 3
SEDUC-CE	Secretaria da Educação do Estado do Ceará, p. 2
SPAECE	Sistema Permanente de Avaliação da Educação Básica do Ceará, p. 2

Sumário

Capítulo 1

Introdução

A matemática, frequentemente, é percebida pelos estudantes como uma disciplina difícil, distante da realidade e repleta de desafios abstratos. Essa resistência tende a aumentar ao longo da trajetória escolar, à medida que novos objetos de conhecimento exigem maior nível de abstração. Como resultado, muitos alunos passam a demonstrar desinteresse, insegurança e, por vezes, aversão, sentimento sintetizado na recorrente pergunta: "Onde vou usar matemática na minha vida?".

Essa indagação revela mais do que uma lacuna de conhecimento técnico; ela expressa uma dificuldade em reconhecer o papel fundamental da matemática como linguagem para compreender o mundo, resolver problemas e modelar situações diversas. Na contramão dessa percepção limitada, os Parâmetros Curriculares Nacionais (PCNs) ressaltam a onipresença da matemática nas mais diversas áreas da vida contemporânea:

> Possivelmente, não existe nenhuma atividade da vida contemporânea, da música à informática, do comércio à meteorologia, da medicina à cartografia, das engenharias às comunicações, em que a matemática não compareça de maneira insubstituível para codificar, ordenar, quantificar e interpretar compassos, taxas, dosagens, coordenadas, tensões, frequências e quantas outras variáveis houver (MEC, 2000, p. 9).

A matemática é uma disciplina que pode despertar curiosidade e interesse quando apresentada de forma prazerosa. No entanto, seu processo de ensino-aprendizagem é complexo, devido a diversos fatores. É inegável que ensinar conceitos, fórmulas e regras matemáticas ainda hoje é uma tarefa desafiadora, muitas vezes vista como entediante pelos alunos, o que introduz uma dificuldade extra e pode provocar o desinteresse pelo estudo. Além disso, a aprendizagem da matemática é um processo contínuo e cumulativo, no qual os conhecimentos prévios dos estudantes são essenciais para o desenvolvimento de novos saberes. Qualquer falha ou interrupção neste processo pode comprometer o desenvolvimento dos alunos.

Analisando os resultados mais recentes do Sistema Permanente de Avaliação da Educação Básica do Ceará (SPAECE), constata-se que o desempenho dos estudantes têm se mantido abaixo dos parâmetros considerados desejáveis. Esse cenário tem gerado preocupações quanto à eficácia das práticas pedagógicas adotadas e suscitado a necessidade de alternativas que contribuam para a melhoria da aprendizagem. Diante disso, é essencial buscar metodologias que possam despertar o gosto e o prazer pelo aprender em matemática.

Novas metodologias pedagógicas podem ser construídas a partir da integração dos conteúdos matemáticos com situações reais, promovendo uma dinâmica que gera resultados significativos e estimula a atitude investigativa do aluno, permitindo-lhe compreender a matemática em contextos cotidianos, explorando o raciocínio lógico, a criatividade e outras habilidades cognitivas. É importante pensar em abordagens que não apenas integrem diferentes áreas do conhecimento, como também considerem o aspecto lúdico e o protagonismo do aluno no processo de aprendizagem. Nesse contexto, o uso da tecnologia no ensino de matemática surge como uma alternativa para promover uma visão diferenciada da disciplina.

1.1 Avaliação da educação básica no Ceará

O SPAECE é um programa desenvolvido pelo Governo do Estado do Ceará, por meio da Secretaria da Educação (SEDUC-CE), desde 1992. De acordo com a página da SEDUC-CE:

> Na vertente Avaliação de Desempenho Acadêmico, o SPAECE caracteriza-se como uma avaliação externa em larga escala que avalia as competências e habilidades dos alunos do Ensino Fundamental e do Ensino Médio, nas áreas de Língua Portuguesa e Matemática. As informações coletadas em cada edição da avaliação permitem identificar o nível de proficiência e a evolução do desempenho dos estudantes ao longo do tempo (SEDUC–CE, 2025).

As provas do SPAECE são aplicadas ao final do 2º, 5º e 9º anos do ensino fundamental e no 3º do ensino médio. Seus resultados subsidiam o planejamento pedagógico das escolas e o monitoramento das políticas educacionais, fornecendo indicadores que permitem identificar fragilidades no processo de ensino-aprendizagem e nortear intervenções mais eficazes.

Ainda segundo a SEDUC–CE (2025), o objetivo do programa é "fornecer subsídios para a formulação, reformulação e monitoramento das políticas educacionais, além de possibilitar aos professores, dirigentes escolares e gestores um panorama da situação da Educação Básica da Rede Pública de Ensino."

1.1.1 Matrizes de referência

Um aspecto importante a ser comentado sobre o SPAECE diz-se respeito às *matrizes de referência*, que norteiam os conteúdos efetivamente avaliados, com o objetivo de aferir competências e habilidades dos alunos. Vale ressaltar que essas matrizes não devem ser confundidas com as matrizes curriculares, pois possuem propósitos distintos.

> Matriz de Referência não se confunde, em absoluto, com Matriz Curricular (currículo). Elas são documentos relacionados, mas possuem objetos e objetivos distintos. A Matriz de Referência é dotada de um âmbito de atuação mais estreito e delimitado do que a Matriz Curricular. A primeira diz respeito ao contexto das avaliações em larga escala, ao passo que a segunda se relaciona com aspectos que, embora envolvam, extrapolam o âmbito da avaliação. (EDUCAÇÃO, 2014, p. 24 apud SAMPAIO, 2018)

Com isso, percebe-se que a matriz de referência tem como foco principal o reforço das competências e habilidades previstas para cada etapa da escolarização. Já a matriz curricular tem como objetivo a seleção e organização dos conteúdos, estruturando o que deve ser ensinado no contexto escolar.

Em 2024, ocorreu um importante alinhamento entre as matrizes de referência do SPAECE e aquelas do Sistema de Avaliação da Educação Básica (SAEB), promovendo maior coerência entre as avaliações em larga escala de âmbito estadual e nacional. Essa iniciativa visou fortalecer o compromisso com a Base Nacional Comum Curricular (BNCC) e assegurar que as competências e habilidades avaliadas estejam em consonância com os objetivos de aprendizagem previstos para cada etapa da educação básica.

O SPAECE organiza suas matrizes de referência em descritores, os quais correspondem a habilidades específicas que dependem do nível em consideração. Por exemplo, alguns dos descritores de matemática presentes no SPAECE 2024 para o 9º ano do Ensino Fundamental estão apresentados na Tabela 1.1. Nela, foi selecionado um descritor representativo para cada uma das cinco unidades temáticas da BNCC. Os descritores D03, D13, D18, D30 e D37 estão relacionados, respectivamente, às unidades temáticas de Geometria, Grandezas e Medidas, Números, Álgebra, e Estatística e Probabilidade.

Tabela 1.1: Alguns descritores extraídos da matriz de referência de matemática do SPAECE 2024 para alunos do 9º ano do ensino fundamental.

Descritor	Descrição da Habilidade
D03	Identificar propriedades de triângulos por meio da comparação de medidas de lados e ângulos.
D13	Resolver problema envolvendo o cálculo da área de figuras planas.
D18	Efetuar cálculos com números inteiros, envolvendo as operações de adição, subtração, multiplicação, divisão e potenciação.
D30	Calcular o valor numérico de uma expressão algébrica.
D37	Associar informações apresentadas em listas e/ou tabelas simples aos gráficos que as representam e vice-versa.

Fonte: Elaborado pelo autor.

Uma análise do desempenho dos alunos baseada nesses descritores possibilita identificar com precisão os conteúdos da matemática sobre os quais eles possuem maior ou menor dificuldade. Essa sistematização fornece subsídios valiosos para que gestores e educadores possam identificar diferentes perfis de desempenho, categorizar os alunos em grupos com necessidades específicas e, assim, planejar intervenções pedagógicas específicas, mais eficazes.

1.1.2 Proficiência e padrões de desempenho

O SPAECE representa a *proficiência média* dos alunos como sendo um número real variando entre 0 e 500, com precisão de uma casa decimal. São calculados valores de proficiência nas escalas estadual, regional, municipal e local (por escola). Com base nesses valores, os alunos são classificados em *padrões de desempenho*. Os intervalos de proficiência correspondentes a cada padrão de desempenho variam de acordo com o componente avaliado.

Para os alunos do 2º ano do ensino fundamental, de acordo com SEDUC–CE/CAEd Digital (2025), os padrões de desempenho são: *não alfabetizado*, *alfabetização incompleta*, *intermediário*, *suficiente* e *desejável*. Já nos casos do 5º ano em diante, o desempenho é dividido em: *adequado*, *intermediário*, *crítico* e *muito crítico*. Na Figura 1.1, há um resumo das características dos padrões na área de matemática para o 9º ano do ensino fundamental. O nível *muito crítico*, correspondente à proficiência de até 225 pontos, caracteriza-se por estudantes que apresentam sérias dificuldades em habilidades matemáticas básicas, exigindo intervenção pedagógica imediata. O nível *crítico*, com proficiência entre 226 e 275 pontos, é atribuído a alunos com domínio limitado de procedimentos e conceitos fundamentais, demandando

reforço contínuo. No nível *intermediário*, compreendido entre 276 e 325 pontos, os estudantes demonstram domínio parcial das habilidades esperadas e encontram-se em processo de consolidação dos conteúdos. Por fim, o nível *adequado*, a partir de 326 pontos, corresponde àqueles que atingem o padrão esperado de desempenho, apresentando domínio consistente dos conteúdos e habilidades matemáticas previstas para a etapa de escolarização.

Figura 1.1: Padrões de desempenho do SPAECE em matemática para o 9º ano do ensino fundamental.

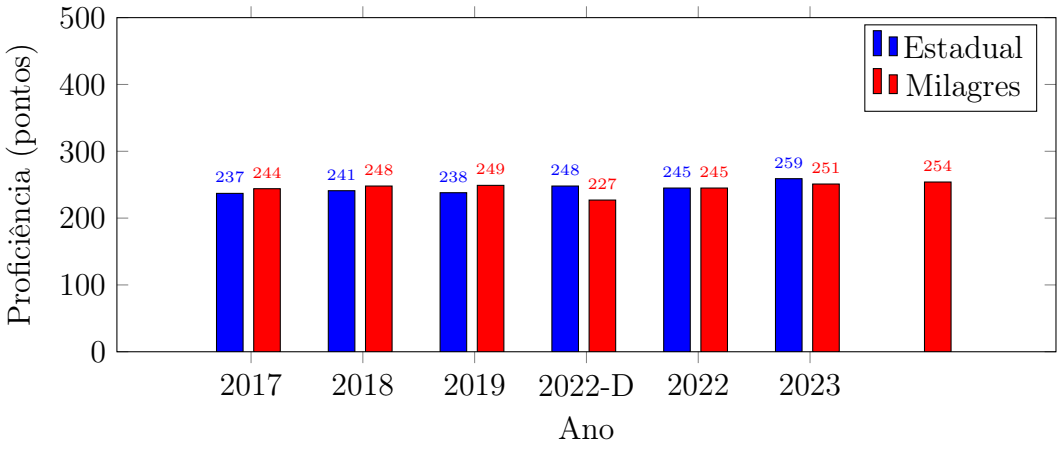
Fonte: SEDUC–CE/CAEd Digital (2025).

1.1.3 Um estudo de caso

Na Figura 1.2 são apresentados resultados referentes à proficiência média em matemática no SPAECE nos últimos seis anos para o 9º ano do ensino fundamental, nos âmbitos estadual e municipal, especificamente para o município de Milagres, região de atuação deste autor. Nota-se uma leve tendência de crescimento ao longo dos últimos anos, tanto em âmbito estadual quanto municipal, com resultados bastante semelhantes. No entanto, os estudantes ainda permanecem majoritariamente no padrão crítico, estando os índices consideravelmente distantes do padrão seguinte, o intermediário.

Mesmo diante dos desafios impostos pela pandemia, em 2022 foram realizadas duas edições do SPAECE. A primeira, denominada SPAECE Diagnóstico, indicada na Figura 1.2 com o rótulo "2022-D", avaliou estudantes do 2º ano do ensino fundamental ao 3º ano do ensino médio, com aplicação de testes nas áreas de língua portuguesa e matemática. O principal objetivo dessa etapa foi identificar lacunas no processo de aprendizagem decorrentes do período de ensino remoto. A segunda aplicação, realizada no mesmo ano, seguiu o modelo tradicional da avaliação externa, contemplando toda a educação básica, com exceção da educação infantil, retomando os padrões de desempenho e os critérios estabelecidos nas matrizes de referência adotadas pelo SPAECE.

Figura 1.2: Valores médios arredondados de proficiência em matemática para os estudantes do 9º ano do ensino fundamental no estado do Ceará e no município de Milagres conforme divulgados por SEDUC–CE/CAEd Digital (2025).

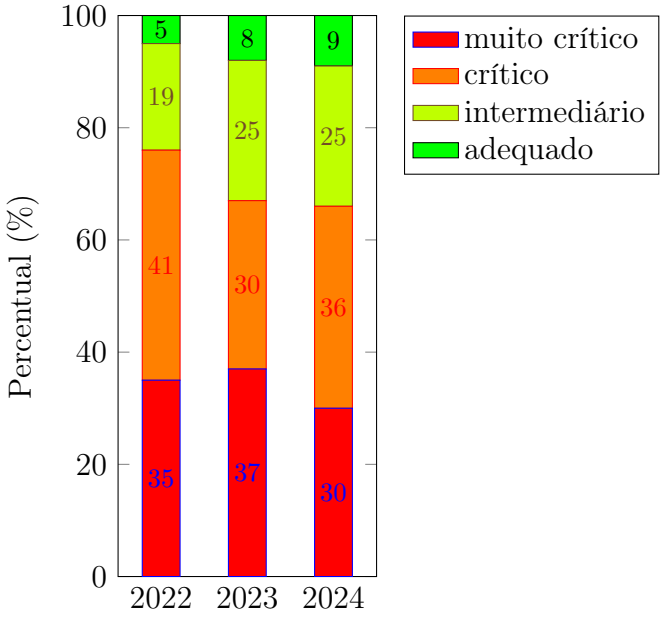


Fonte: Elaborado pelo autor.

Analisando as distribuições dos padrões de desempenho dos estudantes do 9º ano do ensino fundamental do município de Milagres, Ceará, conforme apresentado na Figura 1.3, percebe-se, inicialmente, um dado bastante preocupante e comum aos três anos analisados: mais de 60% dos estudantes permanecem nos padrões de

desempenho muito crítico e crítico, com apenas uma melhora tímida nesses índices ao longo do tempo. No entanto, o gráfico também evidencia uma leve migração desses perfis para níveis mais altos – os percentuais dos padrões muito crítico e crítico apresentaram queda progressiva entre 2022 e 2024. Por outro lado, os padrões intermediário e adequado registraram um crescimento considerável. Ainda assim, apenas 34% dos estudantes encontram-se nesses dois níveis, o que está distante do ideal, especialmente ao se observar o baixo percentual de alunos no padrão adequado, que permanece abaixo de 10%.

Figura 1.3: Padrões de desempenho dos estudantes do 9º ano do ensino fundamental do município de Milagres, Ceará, no SPAECE de 2022, 2023 e 2024.



Fonte: Elaborado pelo autor.

1.2 Programação no ensino de matemática

A análise de séries históricas do SPAECE evidencia a necessidade de repensar as práticas pedagógicas, buscando alternativas que promovam maior engajamento dos estudantes e uma aprendizagem mais efetiva em matemática. Diante desse cenário, surge uma inquietação por novas abordagens metodológicas capazes de despertar o interesse dos alunos.

O uso de tecnologias digitais para o ensino de matemática está fundamentado na BNCC, documento norteador da educação brasileira. Dentre as competências específicas de matemática para o ensino fundamental, destaca-se a competência Nº 5, que propõe:

> Utilizar processos e ferramentas matemáticas, inclusive tecnologias digitais disponíveis, para modelar e resolver problemas cotidianos,

> sociais e de outras áreas de conhecimento, validando estratégias e resultados. (MEC, 2018, p. 267)

Essa competência evidencia a importância de integrar tecnologias digitais ao ensino da matemática, promovendo a resolução de problemas, a comunicação de ideias matemáticas e o desenvolvimento do pensamento computacional.

Considerando que as novas gerações estão completamente imersas e inseridas no mundo digital, torna-se inviável desenvolver atividades pedagógicas que estejam desconectadas dessa realidade. A tecnologia deve ser compreendida como uma grande aliada das práticas docentes, contribuindo para tornar o processo de ensino-aprendizagem mais significativo, interativo e contextualizado. Por isso, é fundamental evidenciar aos estudantes a conexão estreita entre a matemática e os recursos digitais, reforçando o papel dessa disciplina na compreensão e na atuação crítica frente aos desafios do mundo contemporâneo.

Segundo Silva (2014), trazer as tecnologias para o contexto educacional é uma forma eficaz de aproximar o ensino da realidade vivida pelos estudantes, tornando-o mais atraente e relevante:

> Pode-se entender que, para o tempo atual, o interesse da juventude está ligado a diversas coisas e ela consegue se interligar a tudo isso praticamente ao mesmo tempo. Isso significa que trazer as tecnologias para o ambiente educativo pode tornar o processo de ensino e aprendizagem mais prazeroso, mais chamativo e significativo para aquele que aprende e mais dinâmico para aquele que educa. (SILVA; CORREA, 2014, p. 5).

Independente da metodologia adotada, é fundamental destacar o papel do professor, o qual deve proporcionar um ambiente em que o aluno possa expressar suas ideias, esclarecer dúvidas e desenvolver autonomia. É essencial que o docente compreenda as ações dos estudantes para realizar análises e intervenções pedagógicas adequadas, favorecendo a aprendizagem tanto de forma individual quanto coletiva. Trabalhos colaborativos e atividades em grupo são estratégias valiosas, pois promovem a socialização, a cooperação e a construção conjunta do conhecimento. Tais práticas contribuem para que os estudantes compreendam os objetivos de cada proposta, ao mesmo tempo em que desenvolvem suas habilidades de comunicação e argumentação.

Nesse contexto, a associação entre os conteúdos matemáticos e a programação computacional se revela como uma estratégia promissora. Como destacam Azevedo e Maltempi (2020, p. 3), o "[...] pensamento computacional, quando aliado a metodologias ativas de aprendizagem criativa, no contexto escolar, em especial na aprendizagem em matemática, indica, por não ser neutro, possível caminho para se compreender a formação do aluno." A integração entre essas duas áreas possibilita

o desenvolvimento de habilidades lógicas, algorítmicas e de resolução de problemas, promovendo uma aprendizagem ativa, colaborativa e contextualizada. Isto permite que os alunos explorem conceitos matemáticos de forma dinâmica, interativa e significativa, aproximando o conteúdo curricular da realidade tecnológica vivenciada por eles.

É importante ressaltar que lidar com conceitos de programação requer muito cuidado, assim como na matemática. Linguagens de programação de baixo nível, como C, C++ e Fortran, exigem um esforço mental elevado quando comparado a linguagens como Python, Lua ou Julia. Esta última, particularmente, tem se destacado por permitir o uso de símbolos matemáticos e apresentar desempenho equivalente ao C ou Fortran. Ela se torna ainda mais atrativa quando combinada com cadernos digitais, como o Jupyter Notebook, Google Colab ou o Pluto, pois estes possibilitam a construção de hipertextos enriquecidos.

1.2.1 Trabalhos relacionados

O uso da programação computacional associado ao ensino de matemática básica não é um tema novo. Mesmo limitando-se a trabalhos que usam linguagens interpretadas como o Julia, Python, Haskell, Lua, etc., há uma vasta variedade de trabalhos já desenvolvidos. No entanto, ainda são poucos os professores de matemática atuando no ensino fundamental ou médio explorando ambientes de programação ricos, como os cadernos digitais.

Grave (2021) viu na programação, associada ao pensamento computacional e ao método de Polya, uma alternativa para melhorar a aprendizagem de matemática, tendo como público-alvo estudantes do 7º ano do ensino fundamental de uma escola privada da cidade de Cruz Alta, no Rio Grande do Sul. O autor relata que o estudo foi aplicado em três etapas, com a resolução remota de atividades utilizando a linguagem Python em conjunto com o Google Colab. Os conteúdos abordados foram área e probabilidade. Os resultados mostram que os alunos desenvolveram maior autonomia e passaram a traçar suas próprias estratégias na resolução dos problemas, favorecidos pela interação com a máquina. Nesse mesmo ano, Santos (2021) apresentou uma proposta metodológica para o ensino das funções trigonométricas e hiperbólicas, voltada a alunos do ensino médio e da graduação. Como recursos tecnológicos, foram utilizados o GeoGebra e o Symbolab para a plotagem de gráficos e a análise das características geométricas dessas funções. Já a linguagem de programação Python, executada no ambiente Google Colab, foi aplicada para calcular a imagem das funções em ângulos específicos. O trabalho destaca-se por integrar diferentes tecnologias com abordagens visuais, numéricas e algébricas, promovendo um ensino mais dinâmico e exploratório desses conteúdos.

Aliando o Jupyter Notebook à linguagem Python, Fernandes (2023) apresentou uma proposta de ensino voltada à introdução da lógica de programação por meio da utilização de cadernos digitais. Ela teve como foco alunos do 1º ano do curso técnico em Redes de Computadores, integrado ao ensino médio, na disciplina de Programação. Segundo a autora, os resultados obtidos indicaram um elevado nível de aceitação dos cadernos digitais pelos alunos e demonstraram que eles podem desempenhar um papel importante na promoção de um aprendizado eficaz.

Casarin (2022), não se limitando apenas ao ensino de Matemática, desenvolveu uma pesquisa voltada ao ensino de Física, com foco na elaboração de uma sequência didática baseada na teoria da aprendizagem significativa de David Ausubel. Utilizando o Jupyter Notebook associado à linguagem Python, o autor desenvolveu um software simples e acessível para simulação de experimentos de difração e interferência para o estudo da óptica ondulatória. A pesquisa foi aplicada a turmas do 2º ano do ensino médio, e os resultados evidenciaram o potencial da integração entre programação e o ensino de física na compreensão de conceitos abstratos por meio de recursos computacionais interativos.

Mais recentemente, o estudo de Sousa (2023) fomentou discussões acerca do ensino de estatística, educação financeira e linguagem de programação, tendo como público-alvo estudantes do ensino médio. Nele, é realizada uma análise dos retornos diários obtidos em carteiras de investimento. O tratamento e a análise dos dados foram realizados empregando a linguagem Python, juntamente com a biblioteca pandas e o ambiente Google Colab.

Embora exista uma grande quantidade de trabalhos sobre esse tema, majoritariamente esses estudos desenvolvem aplicações voltadas para cursos de graduação e pós-graduação, como os de Khatib (2024), Topsakal (2023), Biehler et al. (2020), Zastre (2019), Yavuz Temel, Barenthien e Padubrin (2025), Al-Gahmi, Zhang e Valle (2022), Cobo et al. (2022), Cisneros Guevara (2024) e Vial e Negoita (2018), no âmbito do ensino superior, em programas das áreas de Ciência, Tecnologia, Engenharia e Matemática (Science, Technology, Engineering, and Mathematics – STEM, em inglês), ou têm como foco o aprimoramento de professores para o uso de tecnologias digitais em sala de aula.

Apesar de o uso de linguagens como Python em Jupyter Notebooks já ser consolidado em áreas como ciência de dados, física e computação no Ensino Médio, ainda há poucas aplicações voltadas especificamente ao ensino de matemática básica. Em Thompson, Wu e Mills (2020), o Jupyter foi usado para atividades de Estatística Avançada com dados reais e visualizações gráficas, enquanto Schönbrodt, Wohak e Frank (2022) propõe um modelo online de modelagem matemática com Jupyter e videoconferência. Já outros estudos (Fleischer, Biehler e Schulte (2022); Mañà Marín (2019); Sutrini, Passaro, Pallotta et al. (2022); Angara et al. (2021)) exploram

notebooks principalmente no ensino de programação, física ou computação quântica, destacando o potencial da ferramenta, mas ainda com foco distante da matemática escolar elementar.

Nas pesquisas supracitadas, ainda que aplicadas em componentes curriculares distintos – matemática e física – e em diferentes níveis de ensino — fundamental, médio e superior –, observa-se a preocupação dos autores em propor a programação como uma alternativa metodológica para potencializar o processo de ensino e aprendizagem. Cada estudo, a seu modo, evidencia como a integração entre recursos computacionais (como Python, Jupyter Notebook e Google Colab) e estratégias pedagógicas pode favorecer o desenvolvimento do pensamento lógico, da autonomia dos estudantes e da resolução de problemas.

Percebe-se também que a linguagem de programação Python é utilizada de forma majoritária em comparação com outras linguagens. Embora essa escolha se justifique por sua sintaxe simples, legibilidade e ampla compatibilidade com ambientes interativos como o Google Colab e o Jupyter Notebook, essa predominância acaba por limitar a exploração de outras possibilidades que poderiam ser igualmente promissoras no contexto educacional, como o uso de linguagens visuais (e.g., Scratch) ou de outras linguagens dinâmicas, tais como Julia, JavaScript, Ruby, Lua e R.

Portanto, ao reunir as contribuições de diferentes autores, este referencial teórico evidenciou que, embora já existam estudos sobre o uso de tecnologias digitais no ensino de Matemática, poucos exploram de forma integrada a linguagem Julia e o Jupyter Notebook como recursos para o desenvolvimento de sequências didáticas alinhadas à BNCC. Nesse sentido, este trabalho se diferencia ao propor a construção de cadernos digitais interativos que articulam programação e conteúdos matemáticos do ensino fundamental, contribuindo tanto para a inovação pedagógica quanto para o fortalecimento das competências previstas no currículo nacional.

1.3 Objetivos

O presente trabalho tem como objetivo mais amplo o de promover a articulação entre o pensamento computacional e o ensino de matemática, mediante a aplicação de práticas pedagógicas inovadoras fundamentadas nas diretrizes da BNCC e nas habilidades previstas pelo SPAECE para a educação básica. Em especial, pretende-se elaborar sequências didáticas utilizando a linguagem Julia e cadernos digitais Jupyter direcionadas a estudantes do 9º ano do ensino fundamental, considerando suas especificidades cognitivas e saberes prévios. Ainda que de forma indireta, este trabalho também se propõe a fomentar o uso dessas duas ferramentas como recurso didático para o ensino de matemática.

1.4 Organização do trabalho

Esta dissertação está organizada em três capítulos principais, além desta introdução. O Capítulo 3 contém uma introdução à linguagem de programação Julia, destacando suas principais características, sintaxe, bibliotecas e aplicabilidade no ensino de matemática. No Capítulo 2, é apresentado o ambiente Jupyter Notebook, explicando seu funcionamento e justificando sua escolha como ferramenta pedagógica. Por fim, o Capítulo 4 é dedicado à apresentação de sequências didáticas que envolvem diferentes tipos de atividades – como jogos, calculadoras e algoritmos – com o objetivo de promover o desenvolvimento das habilidades matemáticas previstas na BNCC e avaliadas pelo SPAECE. Todo o material produzido neste último capítulo integram o produto educacional deste trabalho, os quais estão compilados em um e-book que pode ser acessado no endereço:

`https://onmat.github.io/Matematica.jl`

Capítulo 2

Jupyter Notebook

Neste capítulo, exploraremos o *Jupyter Notebook*, suas principais ferramentas, comandos, linguagens de programação suportadas e discutiremos possibilidades de uso no ensino de matemática básica nos anos finais do ensino fundamental. Ele é um *software* que compõe o projeto Jupyter, dedicado ao desenvolvimento de um conjunto de programas de código aberto para computação interativa e exploratória que permite criar narrativas computacionais. Também conhecido como *caderno digital* Jupyter, ele é descrito na página oficial do projeto como "o aplicativo web original para criar e compartilhar documentos computacionais. Ele oferece uma experiência simples, otimizada e centrada em documentos" (PROJECT JUPYTER COMMUNITY, 2024a). Conforme consta em sua documentação oficial,

> Um Jupyter Notebook é um documento que suporta a integração de código executável, equações, visualizações e texto narrativo. Especificamente, os Jupyter Notebooks permitem que o usuário combine dados, código e texto para contar uma história interativa e computacional. Seja ao analisar um corpus de literatura americana, criar música e arte, ou ilustrar os conceitos de engenharia por trás do processamento digital de sinais, os notebooks podem combinar explicações tradicionalmente encontradas em livros didáticos com a interatividade de um aplicativo (PROJECT JUPYTER COMMUNITY, 2024b).

Por estes motivos, o Jupyter Notebook foi adotado neste trabalho como ambiente de desenvolvimento integrado (IDE, do inglês, *Integrated Development Environment*). Além de oferecer suporte à execução de códigos computacionais em suas células, o Jupyter Notebook permite a inclusão de texto formatado em Markdown com suporte à simbologia matemática do LaTeX.

Embora haja alternativas *online* como o Google Colab, Kaggle e o CoCalc, o Jupyter Notebook não requer conexão à internet, o que é uma vantagem significativa, tendo em vista que nem sempre há disponibilidade de conexão de qualidade. Outro ponto positivo do Jupyter Notebook é a possibilidade de transformar um caderno

digital em *slides*, utilizando a extensão RISE (do inglês, *Reveal.js - Jupyter/IPython Slideshow Extension*). Assim, durante uma apresentação é possível compilar e editar códigos e textos.

2.1 Instalação do Jupyter Notebook

Antes de utilizar, de fato, o Jupyter Notebook, é necessário instalá-lo no computador. Para isso, existem duas opções principais. A primeira é instalar a plataforma Anaconda, que é usada para gerenciar ambientes virtuais de desenvolvimento e inclui ferramentas como o Jupyter Notebook, JupyterLab, VS Code e Spyder. Para baixá-la, basta acessar seu site oficial (https://anaconda.org/anaconda/python) e escolher a opção desejada na seção de *downloads*.

A segunda forma de instalação é por meio do `pip` (do inglês, *Package Installer for Python*), o gerenciador padrão de pacotes da linguagem Python. Com o Python previamente instalado em sua máquina, você pode instalar o Jupyter diretamente pelo Prompt de Comando (CMD), utilizando o seguinte comando:

```
C:\> pip install jupyter nbclassic
```

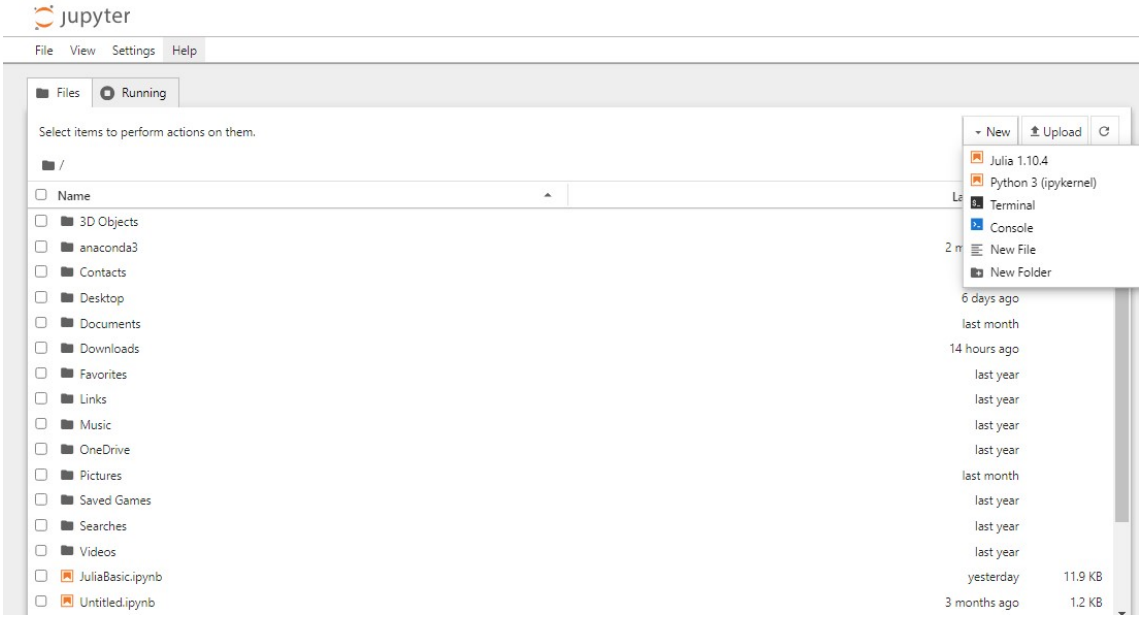
Caso ainda não possua o Python instalado, é possível obter a versão mais recente diretamente no site oficial: https://www.python.org/.

Concluída a instalação do Jupyter Notebook, você pode iniciá-lo executando no CMD o comando:

```
C:\> jupyter nbclassic
```

Será aberta, em seu navegador padrão, a interface do Jupyter Notebook em sua versão clássica. A partir dessa tela, você poderá criar e gerenciar seus cadernos digitais, como mostra a Figura 2.1.

Figura 2.1: Tela inicial do Jupyter Notebook.

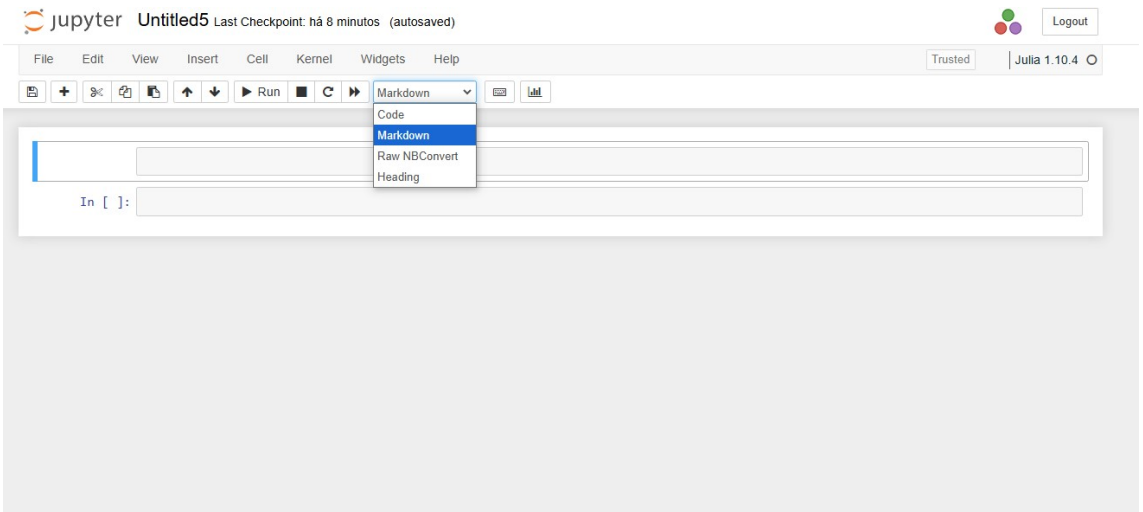


Fonte: Elaborada pelo autor.

2.2 Criação de um Caderno Digital

Com o Jupyter Notebook aberto, conforme mostrado na Figura 2.1, localize o botão `New` no canto superior direito. Ao clicar nele, será exibido um menu suspenso com as versões dos núcleos (em inglês, *kernels*) previamente instalados. Basta selecionar o núcleo desejado e o notebook será criado com a configuração escolhida. Para o propósito deste trabalho, sempre selecionaremos o núcleo `Julia`. Após esta ação, um caderno digital em branco será criado e você verá a tela mostrada na Figura 2.2.

Figura 2.2: Tela do Jupyter Notebook Clássico após a criação de um novo caderno digital.



Fonte: Elaborada pelo autor.

Após criar seu caderno digital, o usuário encontrará no canto superior esquerdo uma *barra de menus* e uma *barra de ferramentas* para acessar as principais funcionalidades do Jupyter Notebook. A barra de menus consiste das seguintes opções:

File: permite criar um novo arquivo, abrir um já existente, salvar ou exportar um notebook;

Edit: permite copiar, cortar, colar e deletar células;

View: possibilita configurar a aparência do caderno digital;

Insert: permite inserir novas células acima ou abaixo da célula ativa;

Cell: para modificar e executar células de código, Markdown, Raw ou Headings;

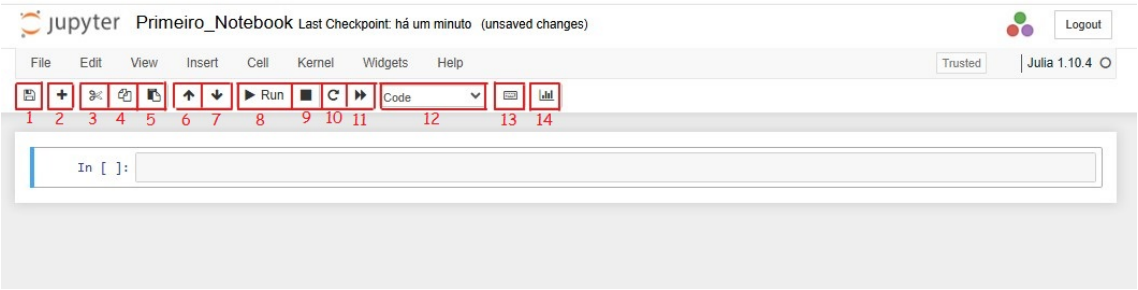
Kernel: possibilita gerenciar o núcleo em execução;

Widgets: para trabalhar com elementos interativos em cadernos digitais;

Help: permite acessar a documentação do Jupyter Notebook.

Já na barra de ferramentas, encontramos as funcionalidades da barra de menus que são mais utilizadas. Ela oferece uma maneira mais rápida de interagir com o bloco de notas.

Figura 2.3: Barra de ferramentas do Jupyter Notebook clássico.



Fonte: Elaborada pelo autor.

Na Figura 2.3, da esquerda para a direita, temos os botões:

1. Salvar;
2. Adicionar célula abaixo;
3. Cortar células selecionadas;
4. Copiar células selecionadas;
5. Colar células selecionadas;

6. Mover célula para cima;
7. Mover célula para baixo;
8. Executar células;
9. Interromper o núcleo;
10. Reiniciar o núcleo;
11. Reinicia o núcleo e executa novamente todo o caderno digital;
12. Menu suspenso para alterar o tipo de célula;
13. Paleta de comandos;
14. Entrar/sair no modo de apresentação RISE.

Na barra de menu na opção `Help`, o usuário pode personalizar a barra de ferramentas adicionando ou removendo botões para tornar sua rotina de desenvolvimento mais eficiente.

2.3 Atalhos de Comandos

Existem dois possíveis modos durante a execução de um bloco de notas Jupyter: o *modo de edição* e o *modo de comando*. O modo de edição ocorre quando o cursor está piscando no interior de uma célula. Caso contrário, o Jupyter Notebook estará em modo de comando.

Para agilizar a interação com o Jupyter Notebook, há atalhos para inserir, excluir e executar células, e muitos outros (GUIDONI, 2024). O comportamento desses atalhos depende do modo que estiver ativado. Na Tabela 2.1, há alguns atalhos que podem ser usados no modo de comando.

Tabela 2.1: Atalhos de comandos do Jupyter Notebook no modo de exibição.

Atalho	Descrição
Enter	Entra no modo de edição
Ctrl + Enter	Executa a célula selecionada
Shift + Enter	Executa a célula selecionada e seleciona a próxima célula
a	Cria uma célula antes da célula selecionada
b	Cria uma célula depois da célula selecionada
d + d	Deleta a célula selecionada
y	Transforma a célula em uma célula de código
m	Transforma a célula em uma célula do tipo Markdown
C	Copia a célula selecionada
X	Recorta a célula selecionada
V	Cola a célula copiada/recortada abaixo da célula selecionada

Fonte: Elaborada pelo autor.

Já para o modo de edição, temos os atalhos da Tabela 2.2 (GUIDONI, 2024).

Tabela 2.2: Atalhos de comandos do Jupyter Notebook no modo de edição.

Atalho	Descrição
Tab	Completa código
Ctrl +]	Indenta código
Ctrl + [	Desfaz indentação do código
Ctrl + A	Seleciona todo o texto da célula
Ctrl + Z	Desfaz alterações do texto da célula
Ctrl + /	Comenta código
Ctrl + Shift + Z	Refaz a última alteração do texto da célula

Fonte: Elaborada pelo autor.

2.4 Tipos de Células

Todo o conteúdo de um caderno digital do Jupyter Notebook é organizado em *células*, as quais podem ser de quatro tipos distintos: *Texto*, *Código*, *Bruta* e *Cabeçalho*. Por simplicidade, discorreremos apenas sobre os dois primeiros, visto que são os únicos que possuem seus correspondentes no Google Colab.

2.4.1 Células de Texto

Células de texto, como o próprio nome evidencia, são usadas para conter texto escrito na linguagem *Markdown*. Com ela, é possível construir textos estruturados, com seções, sub-seções, etc. Para criar o título de um novo caderno digital, inserimos uma célula de texto iniciando com um *hashtag*. As seções, subseções, sub-subseções, etc, são criadas concatenando mais *hashtags*.

De maneira similar ao que ocorre em editores de texto, pode-se dar ênfase a palavras ou a frases formatando-as em negrito, itálico, sublinhado, tachado ou no estilo de código. Para tanto, no Jupyter Notebook, usamos os comandos exibidos na Tabela 2.3.

Tabela 2.3: Configurações de estilo do texto.

Símbolo	Sintaxe	Descrição da saída
asteriscos duplos	**negrito**	**negrito**
asterisco simples	*itálico*	*itálico*
til	~~tachado~~	~~tachado~~
crase	`código`	`código`

Fonte: Elaborada pelo autor.

São as células de texto que nos permitem ciar cadernos digitais enriquecidos com imagens, listas, tabelas, expressões matemáticas, links, áudios e vídeos.

Expressões Matemáticas

Para utilizarmos simbologia matemática nas células de texto – como operadores aritméticos e relacionais – podemos empregar a mesma sintaxe do LaTeX. Ao escrevermos a expressão entre cifrões simples ($...$), ela aparece ao longo do texto. Se forem utilizados cifrões duplos ($$...$$), a expressão é exibida centralizada, em um parágrafo próprio. Por exemplo, o resultado de `$ax^2+bx+c=0$` é a equação $ax^2 + bx + c = 0$. A Tabela 2.4 contém alguns símbolos matemáticos frequentes e seus respectivos códigos em LaTeX.

Tabela 2.4: Símbolos matemáticos frequentes e seus respectivos códigos em LaTeX.

Operadores relacionais		
Descrição	Código LaTeX	Resultado
Igualdade	`$a = b$`	$a = b$
Diferente	`$a \neq b$`	$a \neq b$
Menor que	`$a < b$`	$a < b$
Maior que	`$a > b$`	$a > b$
Menor ou igual a	`$a \leq b$`	$a \leq b$
Maior ou igual a	`$a \geq b$`	$a \geq b$
Aproximadamente igual	`$a \approx b$`	$a \approx b$
Equivalente	`$a \equiv b$`	$a \equiv b$
Semelhante	`$a \sim b$`	$a \sim b$
Aproximado	`$a \simeq b$`	$a \simeq b$
Congruente	`$a \equiv b \pmod{n}$`	$a \equiv b \pmod{n}$
Tende a	`$x \to 0$`	$x \to 0$
Símbolos de conjuntos		
Conjunto vazio	`$ \varnothing $`	$\varnothing$
Pertence	`$\in $`	$\in$
Não pertence	`$ \notin $`	$\notin$
União	`$\bigcup$`	$\bigcup$
Interseção	`$\bigcap$`	$\bigcap$
Subconjunto	`$ \subseteq $`	$\subseteq$
Não é subconjunto	`$ \nsubseteq $`	$\nsubseteq$
Para todo	`$ \forall $`	$\forall$
Existe	`$ \exists $`	$\exists$
Conjunto numéricos		
Naturais	`$\mathbb{N}$`	$\mathbb{N}$
Inteiros	`$\mathbb{Z}$`	$\mathbb{Z}$
Racionais	`$\mathbb{Q}$`	$\mathbb{Q}$
Reais	`$\mathbb{R}$`	$\mathbb{R}$
Símbolos de lógica proposicional		
OU: Conjunção	`$\lor$`	$\lor$
E: Disjunção	`$\land$`	$\land$
NÃO: Negação	`$\neg$`	$\neg$
ENTÃO: Condicional	`$\Rightarrow$`	$\Rightarrow$
SE SÓ SE: Bicondicional	`$\Leftrightarrow$`	$\Leftrightarrow$

Fonte: Elaborada pelo autor.

Na escrita de textos sobre geometria, é bem comum usarmos símbolos para relacionar ou nos referirmos a certos objetos. Por exemplo, para dizer que duas retas r e s são paralelas, escrevemos `$r \parallel s$`, cujo resultado é $r \parallel s$. A Tabela 2.5 contém outros símbolos matemáticos úteis para a área de geometria.

Tabela 2.5: Símbolos matemáticos da área de geometria.

Descrição	Código LaTeX	Resultado
Paralelo	`$\parallel$`	$\parallel$
Perpendicular	`$perp$`	$\perp$
Segmento AB	`$\overline{A B}$`	$\overline{AB}$
Segmento orientado AB	`$\overrightarrow{A B}$`	$\overrightarrow{AB}$
Ângulo	`$\angle$`	$\angle$
Arco AB	`$\overset{\frown} {AB}$`	$\overset{\frown}{AB}$
Vetor u	`$\vec{u}$`	$\vec{u}$
Triângulo	`$\triangle$`	$\triangle$
Quadrado	`$\square$`	$\square$

Fonte: Elaborada pelo autor.

No caso da álgebra, são essenciais os símbolos apresentados na Tabela 2.6.

Tabela 2.6: Símbolos matemáticos utilizados em expressões algébricas.

Descrição	Código LaTeX	Resultado
Adição	+	+
Subtração	-	−
Multiplicação	`$\cdot$`	$\cdot$
Divisão	`$\div$`	$\div$
Multiplicação (alternativa)	`$\times$`	$\times$
Mais ou Menos	`$\pm$`	$\pm$
Fração	`$\frac{a}{b}$`	$\frac{a}{b}$
Reticência	`$\cdots$`	$\cdots$
Expoente	`$a^{n}$`	a^n
Índice	`$a_{n}$`	a_n
Raiz quadrada de b	`$\sqrt{b}$`	$\sqrt{b}$
Raiz enésima de b	`$\sqrt[n]{b}$`	$\sqrt[n]{b}$

Fonte: Elaborada pelo autor.

Também é bastante comum utilizarmos o alfabeto grego. Na geometria, designamos planos usando o pi e o phi. Em álgebra, temos por exemplo o delta da fórmula resolutiva de equações do segundo grau. Para obtê-los, basta usar a barra invertida

seguida do nome da letra grega desejada, em inglês. A Tabela 2.7 traz algumas letras gregas minúsculas.

Tabela 2.7: Letras gregas minúsculas.

Descrição	Código LaTeX	Resultado
Alpha	`\alpha`	α
Beta	`\beta`	β
Gamma	`\gamma`	γ
Delta	`\delta`	δ
Pi	`\pi`	π
Phi	`\phi`	ϕ
Ômega	`\omega`	ω

Fonte: Elaborada pelo autor.

Caso o usuário necessite de uma letra grega maiúscula, basta escrever a primeira letra do código em maiúsculo. Por exemplo, escrevemos a expressão algébrica `$\Delta = b^2 -4ac$` para representar o discriminante *delta* da fórmula de Bháskara. Após compilá-la, o resultado será: $\Delta = b^2 - 4ac$.

Para expressões envolvendo somatórios, produtórios, limites, integrais, logaritmos e funções trigonométricas, o LaTeX disponibiliza os comandos da Tabela 2.8. Observe que, diferentemente do cosseno, o código padrão para o seno é `\sin`, que produz sin, em inglês.

Tabela 2.8: Somatórios, produtórios, limites, integrais, logaritmos e funções trigonométricas em LaTeX.

Descrição	Código LaTeX	Resultado
Logaritmo	`$\log_b{a}$` ou `$\log{a}$`	$\log_b a$ ou $\log a$
Logaritmo natural	`$\ln{a}$`	$\ln a$
Seno	`$\operatorname{sen}{x}$`	$\operatorname{sen} x$
Cosseno	`$\cos{x}$`	$\cos x$
Tangente	`$\tan{x}$`	$\tan x$
Limite	`$ \lim_{x \to \infty} f(x) $`	$\lim_{x\to\infty} f(x)$
Integral	`$ \int_{a}^{b} x^2 \, dx $`	$\int_a^b x^2 \, dx$
Somatório	`$ \sum_{k=1}^{n} \frac{1}{k} $`	$\sum_{k=1}^{n} \frac{1}{k}$
Produtório	`$ \prod_{k=1}^{n} \frac{1}{k} $`	$\prod_{k=1}^{n} \frac{1}{k}$

Fonte: Elaborada pelo autor.

Para expressões complexas, envolvendo equações longas, matrizes e funções definidas por partes, usamos ambientes matemáticos especiais, como ilustrado a seguir.

Ambiente `align`.

```
\begin{align*}
    \sum_{k=1}^{n}k^3 &= 1^3 + 2^3 + \cdots + n^3\\
                      &= \left(\frac{n(n+1)}{2}\right)^2
\end{align*}
```

$$\begin{aligned}\sum_{k=1}^{n} k^3 &= 1^3 + 2^3 + \cdots + n^3 \\ &= \left(\frac{n(n+1)}{2}\right)^2\end{aligned}$$

Ambiente `cases`.

```
$$
|x| = \begin{cases}
         x, & \text{se $x \ge 0$}\\
        -x, &\text{se $x < 0$}
      \end{cases}
$$
```

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

Ambiente `bmatrix`.

```
$$
\begin{bmatrix}
    1 & 2 & 3 \\
    4 & 5 & 6 \\
    7 & 8 & 9
\end{bmatrix}
$$
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Além desses, há muitos outros ambientes matemáticos disponíveis no LaTeX, como o `multline`, `split`, `gather`, `matrix`, `vmatrix` e o `pmatrix` (AMS, 2020).

Imagens

Para inserir imagens, usamos o comando:

```
![descrição do link](endereço do link)
```

Dentro dos colchetes, fornecemos uma descrição da imagem, enquanto que entre parênteses, é necessário informar o endereço eletrônico (URL, do inglês, *Uniform Resource Locator*) da imagem. Se a imagem estiver armazenada localmente, fornecemos o caminho do arquivo, incluindo sua extensão. As extensões compatíveis incluem JPG, GIF, PNG, SVG, PSD, WEBP, RAW, TIFF, BMP ou PDF.

Por exemplo, o código:

```
![Logotipo do CTAN](https://ctan.org/assets/lion/lion.svg)
```

insere uma imagem a partir de seu URL, enquanto que:

```
![Logotipo do CTAN](C:\Users\Usuario\Desktop\lion.svg)
```

terá o mesmo efeito, desde que o arquivo `lion.svg` esteja armazenado localmente, na área de trabalho.

É importante ressaltar que a inserção de imagens por esse método não permite ajustar o tamanho da figura.

Links

Muitas vezes, precisamos criar atalhos no texto que nos direcionam para uma determinada página na internet. A inserção de *links* em células de texto possui sintaxe semelhante a usada para inserir imagens. A única diferença é que, no caso dos links, não há a necessidade do sinal de exclamação. Entre colchetes, inserimos palavras que descrevem o link e entre parênteses, o URL da página, como a seguir:

```
[Ir para UFCA](https://www.ufca.edu.br/)
```

Listas

As listas são elementos textuais que nos ajudam a organizar informações de um modo linear. As células de texto permitem a criação de listas ordenadas e não ordenadas. Para construir listas não ordenadas, colocamos os caracteres * (asterisco), - (hífen) ou + (mais), precedendo cada um de seus itens. No caso das listas ordenadas, devemos usar letras ou números antes de cada item, seguidos do sinal de ponto final. Assim, o código:

```
- Ciências Humanas e suas Tecnologias.
- Linguagens, Códigos e suas Tecnologias.
- Ciências da Natureza e suas Tecnologias.
- Matemática e suas Tecnologias.
- Redação.
```

produzirá uma lista não ordenada, enquanto que:

```
1. Grandezas proporcionais
1. Estatística, gráficos e tabelas
3. Aritmética
4. Geometria plana e espacial
1. Funções
6. Probabilidade
```

produz uma lista ordenada. A numeração dos itens de uma lista ordenada é calculada automaticamente, a partir do número de seu primeiro elemento, não importando o número especificado para os demais.

Tabelas

Para criar uma tabela, usamos barras verticais (|) e hifens para especificar seu formato. A barra vertical serve para definir colunas e o hífen para inserir uma linha que separa a primeira linha das demais. O código a seguir produzirá uma tabela com três linhas e quatro colunas.

```
|título_1|título_2|título_3|
|--------|--------|--------|
|célula_1|célula_2|célula_3|
|célula_4|célula_5|célula_6|
|célula_7|célula_8|célula_9|
```

Com isso, uma tabela verdade poderia ser construída como a seguir:

```
|$p$   |$q$  | p$ \lor$ q |p $\land$ q|p $ \Rightarrow$ q|
|----|---|:----------:|:---------:|:---------------:|
|V   |V  |     V      |     V     |     V           |
|V   |F  |     V      |     F     |     F           |
|F   |V  |     V      |     F     |     V           |
|F   |F  |     F      |     F     |     V           |
```

Observe que, na segunda linha da tabela, nas colunas três, quatro e cinco, utilizamos dois pontos antes e após os hifens. Isso modifica o alinhamento do conteúdo

dessas colunas, centralizando-os horizontalmente. Para alinhar à esquerda, colocamos os dois pontos antes dos hifens e para alinhar à direita colocamos os dois pontos após os hifens.

Apesar de sua simplicidade, esse modo de se construir tabelas não oferece outros layouts nem possibilita controlar a aparência das células. Para isso, seria necessário utilizar a linguagem HTML.

Formatação de texto com HTML

A produção de hipertextos visualmente sofisticados no Jupyter Notebook requer a utilização de HTML.

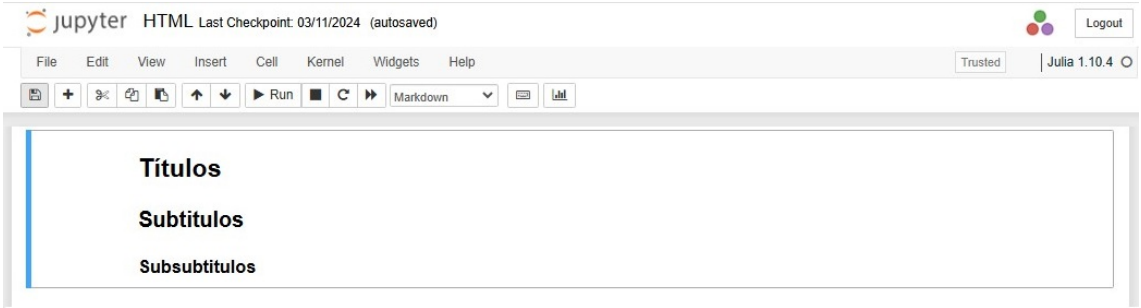
> HTML (HyperText Markup Language) é a linguagem fundamental da web, desempenhando um papel crucial na criação e estruturação de páginas na internet. Esta linguagem de marcação é essencial para qualquer pessoa interessada em desenvolver conteúdo para a web (ACADEMY, 2024).

As *tags* em HTML são elementos de marcação que definem a estrutura e o significado do conteúdo em uma página. Elas são envolvidas por sinais de "menor que" e "maior que" que definem a abertura e o fechamento de uma tag e podem ter atributos que fornecem informações adicionais. Por exemplo, as tags `<h1>`, `<h2>`, ..., `<h6>` servem para definir seções, subseções, sub-subseções, etc. O número após o `h` indica a hierarquia dos títulos de uma seção, sendo <h1> o nível mais alto e <h6> o nível mais baixo. Em uma célula de texto, ao escrever:

```
1    <h1>Título</h1>
2    <h2>Subtítulo</h2>
3    <h3>Sub-subtítulo</h3>
```

o resultado é o que está apresentado na Figura 2.4.

Figura 2.4: Definindo seções usando HTML.



Fonte: Elaborada pelo autor.

A Tabela 2.9 contém algumas tags comuns em HTML que serão exploradas mais a frente neste trabalho.

Tabela 2.9: Algumas tags usadas na formatação de textos em HTML.

Tag	Descrição
<a>	Cria um hyperlink (link em documentos da web internos/externos)
<audio>	Incorpora conteúdo de áudio
<b>	Especifica o texto em negrito
 	Insere uma quebra de linha
<i>	Representa um texto em formato itálico
<iframe>	Define um quadro embutido que incorporou conteúdo externo
<img>	Insere imagem em um documento da web
	Define uma lista não ordenada de itens
	Define uma lista ordenada de itens
	Define um item de uma lista, ordenada , ou não ordenada
<s>	Exibe texto no estilo tachado
	Agrupa elementos em linha
<small>	Deixa o texto com uma fonte menor em relação ao texto circundante
<source>	Especifica vários recursos de mídia, como vídeo e áudio
	Representa forte ênfase em uma parte importante do texto
<style>	Adiciona estilo CSS a um documento HTML
<sub>	Representa o texto subscrito embutido
<sup>	Representa texto sobrescrito embutido
<table>	Define uma tabela em um documento HTML
<td>	Cria células de dados em uma tabela HTML
<tfoot>	Adiciona conteúdo de rodapé em uma tabela
<th>	Cria cabeçalho de um grupo de células na tabela HTML
<time>	Representa a data e/ou hora em um documento HTML
<title>	Representa o título de um documento HTML
<tr>	Define uma linha de células em uma tabela
<u>	Representa o texto sublinhado
<video>	Incorpora conteúdo de vídeo

Fonte: Elaborada pelo autor.

Por exemplo, para reproduzirmos a frase "Uso de tags para formatar em **negrito**, *itálico* e sublinhado", usamos o código:

```
1 <p>Uso de tags para formatar em <b>negrito</b>, <i>itálico</i>
2 e <u>sublinhado</u>.</p>
```

Aqui, usamos a tag `<p>` para indicar que a frase é um parágrafo e as tags `<b>`, `<i>` e `<u>` para alterar o estilo do texto para negrito, itálico e sublinhado, respectivamente.

Podemos ainda alterar a cor de porções de texto usando o código:

```
<span style="color:cor">Texto</span>
```

onde a `cor` pode ser especificada através de seu nome em inglês ou seu código RGB, HEX, HSL, RGBA ou HSLA.

A tag <hr> permite criar uma linha horizontal, muitas vezes usada para separar seções. É possível controlar sua espessura, cor e alinhamento. Para criar uma linha com espessura de 4 pixels, na cor rosa e ocupando todo o comprimento de uma célula de texto, usamos:

```
<hr style="width:100%; height:4px; background-color:pink;
border: none; margin: auto;">
```

Para inserirmos uma imagem, a forma geral do código é:

```
<img src="arquivo" alt="descrição" width="largura"  height="altura">
```

onde:

- `src`: é usado para especificar o caminho onde arquivo está armazenado (localmente ou online);
- `alt`: contém uma descrição da imagem;
- `width`: define a largura da imagem em pixels;
- `height`: define a altura da imagem em pixels.

Caso o atributo `height` não seja especificado, a altura da imagem é calculada proporcionalmente à largura especificada pelo atributo `width`. Diferentemente de outras tags, a `<img>` não precisa de fechamento.

Um link é criado com a tag `<a>`, utilizada para abertura do bloco, acompanhada ao menos do atributo `href`, usado para especificar a URL apontada. Para o fechamento do bloco, usamos a tag `</a>`. Entre as tags de abertura e fechamento, colocamos aquilo que será apontado à URL, podendo ser um texto ou imagem, por exemplo. O código a seguir ilustra como criar um link que nos direcione ao site oficial da UFCA:

```
<a href="https://www.ufca.edu.br/">Ir para a UFCA</a>
```

Para criar listas ordenadas e não ordenadas, respectivamente, usamos as tags `<ol>` e `<ul>`. As listas apresentadas anteriormente com Markdown ficariam assim em HTML:

```
<ul>
  <li>Ciências Humanas e suas Tecnologias.</li>
  <li>Linguagens, Códigos e suas Tecnologias.</li>
  <li>Ciências da Natureza e suas Tecnologias.</li>
  <li>Matemática e suas Tecnologias.</li>
  <li>Redação.</li>
</ul>
```

e

```
<ol>
  <li>Grandezas proporcionais</li>
  <li>Estatística, gráficos e tabelas</li>
  <li>Aritmética</li>
  <li>Geometria plana e espacial</li>
  <li>Funções</li>
  <li>Probabilidade</li>
</ol>
```

Observe que a única diferença entre elas é a utilização da tag `<ul>` ou `<ol>`. Assim como em Markdown, listas de diferentes tipos podem ser aninhadas para criar estruturas mais complexas.

Por fim, mostraremos como se construir tabelas em HTML. Para isso, usamos a tag `<table>` para definir o corpo da tabela e as tags `<tr>` para criar linhas e `<td>` para criar colunas. O cabeçalho da tabela é definido com tags `<th>`, geralmente posicionadas na primeira linha. Vale lembrar que todas essas tags precisam de fechamento. A tabela verdade vista anteriormente em Markdown ficaria assim em HTML:

```
<table width="30%" border="1" style="text-align: center;">
    <tr>
        <th>p</th>
        <th>q</th>
        <th>p $\lor$ q</th>
        <th>p $\land$ q</th>
        <th>p $\Rightarrow$ q</th>
    </tr>
    <tr>
        <td> <span style="color:green"> V </span></td>
        <td> <span style="color:green"> V </span></td>
```

```
12          <td> <span style="color:green"> V </span></td>
13          <td> <span style="color:green"> V </span></td>
14          <td> <span style="color:green"> V </span></td>
15      </tr>
16      <tr>
17          <td> <span style="color:green"> V </span></td>
18          <td> <span style="color:red"> F </span> </td>
19          <td> <span style="color:green"> V </span></td>
20          <td> <span style="color:red"> F </span> </td>
21          <td> <span style="color:red"> F </span> </td>
22      </tr>
23      <tr>
24          <td> <span style="color:red"> F </span> </td>
25          <td> <span style="color:green"> V </span></td>
26          <td> <span style="color:green"> V </span></td>
27          <td> <span style="color:red"> F </span> </td>
28          <td> <span style="color:green"> V </span></td>
29      </tr>
30      <tr>
31          <td> <span style="color:red"> F </span> </td>
32          <td> <span style="color:red"> F </span> </td>
33          <td> <span style="color:red"> F </span> </td>
34          <td> <span style="color:red"> F </span> </td>
35          <td> <span style="color:green"> V </span></td>
36      </tr>
37  </table>
```

Para melhorar a apresentação da tabela, foram acrescentados os atributos `width`, `border` e `style`, responsáveis, respectivamente, por: ajustar a largura da tabela, definir a espessura da borda e alinhar horizontalmente o conteúdo dentro das células. Além disso, a tag `<span>` juntamente com o atributo `style` foi usada para destacar os valores lógicos, exibindo valores verdadeiros em verde e valores falsos em vermelho.

2.4.2 Células de Código

As células de código permitem editar, com realce de sintaxe, e executar de modo interativo trechos de códigos escritos em diversas linguagens de programação. A linguagem utilizada ao longo de todo o documento fica determinada ao se criar o caderno digital, definindo o seu *núcleo*. O Jupyter Notebook, por padrão, vem com o núcleo de Python. Porém, podem ser adicionados outros núcleos a qualquer momento, tais como: Julia, R, Scala, JavaScript, Ruby, Haskell, Go, C e C++.

Neste trabalho, será utilizada a linguagem Julia. Por isso, o Capítulo 3 foi dedicado para introduzi-la, explorar suas particularidades e mostrar como sua utilização pode ser uma poderosa ferramenta para o ensino de matemática básica. Com o núcleo de Julia instalado, em uma célula de código, podemos inserir o texto:

Código 2.1: Exemplo de código em Julia para o cálculo das raízes reais de uma equação do segundo grau usando a fórmula de Bháskara.

```
print("Digite o coeficiente a: ")
a = parse(Int, readline())
print("Digite o coeficiente b: ")
b = parse(Int, readline())
print("Digite o coeficiente c: ")
c = parse(Int, readline())
Δ = b^2 -4*a*c
println("Δ = $Δ")
if Δ>0
    x_1=(-b+sqrt(Δ))/(2a)
    x_2=(-b-sqrt(Δ))/(2a)
    println("A equação possui duas raízes reais. \nS={$x_1, $x_2} ")
elseif Δ==0
    x_1=-b/(2a)
    println("A equação possui uma raiz real. \nS={$x_1}")
else
    println("A equação não possui raízes reais. \nS = \u2205")
end
```

Fonte: Elaborado pelo autor.

Esse código calcula as raízes reais da equação $ax^2 + bx + c = 0$ (caso existam) a partir dos valores dos coeficientes a, b e c, que são informados pelo usuário de forma interativa. Ao usá-lo para resolver a equação $x^2 - 5x + 6 = 0$, o resultado será:

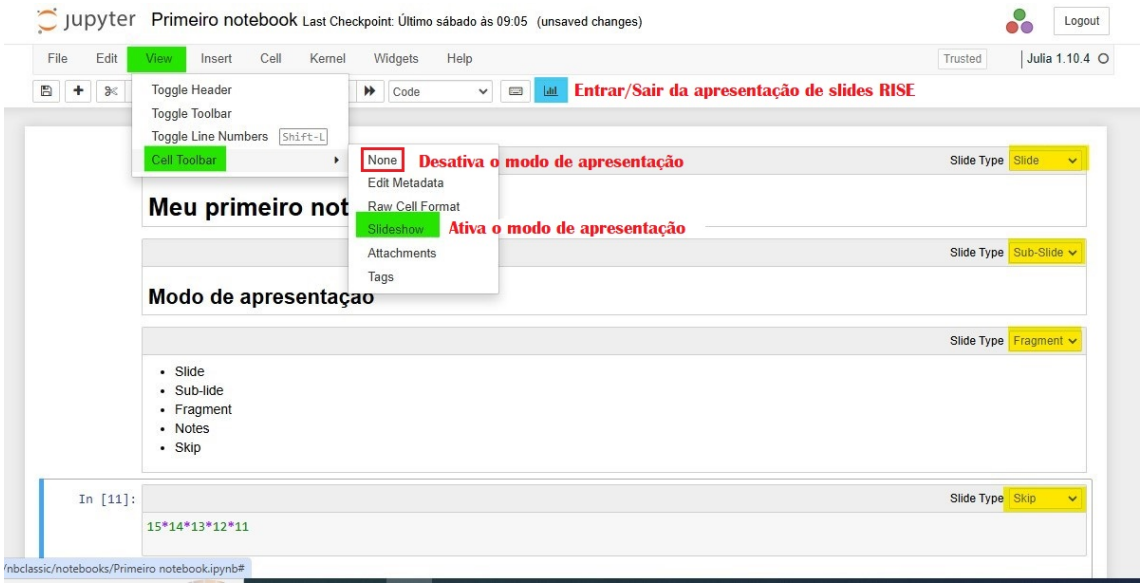
```
Digite o coeficiente a: 1
Digite o coeficiente b: -5
Digite o coeficiente c: 6
Δ = 1
A equação possui duas raízes reais.
S={3.0, 2.0}
```

2.5 Modo de Apresentação

Uma funcionalidade importante do Jupyter Notebook é o fato de um caderno digital poder ser usado como uma apresentação. Isto é realizado por intermédio da extensão RISE, do inglês, *Reveal.js – Jupyter/IPython Slideshow Extension* (AVILA, 2025), que pode ser instalada através do comando '*pip install RISE*'. Essa ferramenta permite transformar o caderno digital em uma apresentação interativa, na qual o usuário pode editar e executar células de texto ou código durante a apresentação, visualizando o resultado em tempo real. Isto é particularmente importante para aulas de matemática, tendo em vista que podemos combinar explicações teóricas com experimentação prática.

Com o RISE instalado, na barra de menu da tela inicial (Figura 2.5), ao clicar em View > Cell Toolbar > Slideshow, aparecerá à frente de cada célula a lista suspensa Slide Type. Ela deve ser usada para definir como uma célula deve se comportar.

Figura 2.5: Tela: Modo de apresentação



Fonte: Elaborada pelo autor.

Há cinco opções de comportamento para cada célula:

- `slide`: a célula inteira é usada para definir um slide;
- `sub-slide`: a célula inteira é usada para definir um sub-slide, vinculado ao slide anterior mais próximo;
- `fragment`: o conteúdo da célula é parte do slide ou sub-slide anterior mais próximo e deve ser adicionado de modo incremental;
- `notes`: adiciona anotações que serão visíveis apenas na tela do dispositivo usado para abrir a apresentação;
- `skip`: ignora a célula durante a apresentação.

Após configurar cada célula de um caderno digital, a apresentação é iniciada ao clicar no botão “Enter/Exit Rise show”. Para navegar pelos slides, deve-se usar as setas do teclado ou a tecla de espaço.

Capítulo 3

Programando em Julia

É pertinente começarmos este capítulo com a seguinte indagação: o que é uma linguagem de programação?

> Uma linguagem de programação é um sistema de comunicação entre o ser humano e o computador, que permite escrever instruções que o computador pode executar. Essas instruções são dadas por meio de comandos, regras e sintaxes específicas que são compreendidas pelo processador de um computador (SOMMERVILLE, 2016).

Atualmente, há uma infinidade de linguagens de programação disponíveis. Contudo, uma classe particular delas, as *linguagens interpretadas*, tem recebido cada vez mais atenção da comunidade em geral. Destas, merece destaque a Julia, uma linguagem simples de programar como o Python, com a promessa de ser tão rápida quanto a C ou o Fortran. Ela vem sendo desenvolvida desde 2009 por um grupo de pesquisadores do Instituto de Tecnologia de Massachusetts que conta com a participação de Alan Edelman, Stefan Karpinski, Jeff Bezanson e Viral Shah. O intuito deles era facilitar a análise de dados numéricos, buscando atender inicialmente à comunidade de computação científica e alto desempenho, mas com potencial para a programação de cunho geral.

Como a maioria das linguagens interpretadas modernas, a Julia é uma linguagem de tipagem dinâmica. Dessa forma, ela determina o tipo das variáveis de acordo com o dado atribuído.

> Existe uma série de propriedades que são desejáveis em uma linguagem de programação, sendo determinantes na escolha da mesma para o desenvolvimento de algum software. São elas: legibilidade, regibilidade, confiabilidade, eficiência, facilidade de aprendizado, ortogonalidade, modificabilidade, reusabilidade, portabilidade (SOUZA MAGALHAES; AVALOS; SANTOS, 2017).

Nesse contexto, pode-se dizer que a Julia reúne grande parte dessas características, oferecendo uma sintaxe clara e amigável. Além disso, seu desempenho ímpar

e semelhança com linguagens bastante conhecidas como Python e Matlab facilita a migração de usuários já experientes, tornando-a uma alternativa interessante para projetos científicos e educacionais. Uma característica peculiar da Julia que ainda vale ressaltar é sua capacidade de lidar com uma simbologia semelhante a que usamos cotidianamente nas aulas de matemática. Com isso, é possível utilizar operadores relacionais, letras gregas, conectivos lógicos, entre outros.

3.1 Instalação

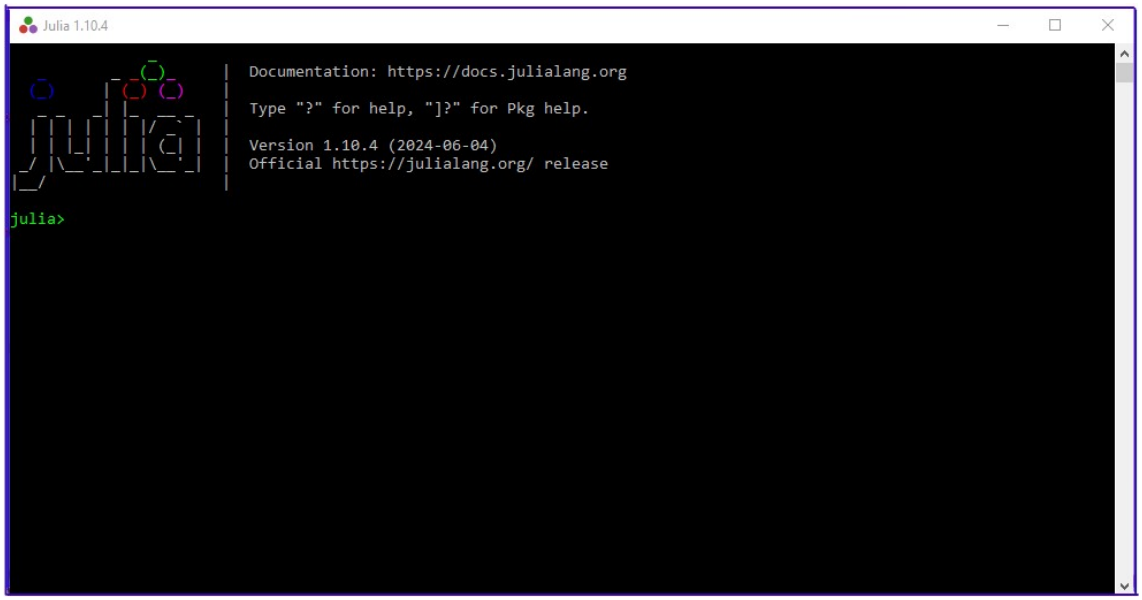
Neste trabalho, a Julia é utilizada dentro de cadernos digitais do Jupyter Notebook. Para isso, é preciso ter instalado tanto o interpretador `julia` quanto o núcleo `IJulia`. A seguir, será detalhado todo o processo desde o download do arquivo de instalação até o passo a passo para sua execução.

Para instalar o `julia`, deve-se acessar o site julialang.org e entrar na guia *Install*. Lá, são indicadas opções de instalação para diferentes sistemas operacionais. No Windows, é possível instalá-lo usando a Microsoft Store ou de modo manual, sendo a primeira maneira a mais recomendada. Neste caso, no CMD, basta executar o comando:

```
C:\> winget install --name Julia --id 9NJNWW8PVKMN -e -s msstore
```

Após reabrir o CMD, o `julia` estará pronto para ser utilizado. Agora, ao digitar `julia` e pressionar a tecla *Enter*, o resultado será a tela apresentada na Figura 3.1.

Figura 3.1: Tela inicial do console do `julia` no Windows.



Fonte: Elaborada pelo autor.

Supondo o Jupyter Notebook instalado, no console do `julia`, deve-se executar o comando:

```
julia> using Pkg
```

e em seguida:

```
julia> Pkg.add("IJulia")
```

A partir de agora, ao se criar um novo caderno digital no Jupyter Notebook, o núcleo IJulia estará disponível. Uma vez escolhido, todas as células de código terão a Julia como linguagem padrão. Todos os resultados de execução de códigos apresentados neste trabalho foram produzidos utilizando-se o Jupyter Notebook.

3.2 Variáveis

Toda e qualquer informação que é manipulada em um programa está associada a um *rótulo* e um *objeto*. Um objeto por ser visto como uma região da memória que contém dados (o conteúdo) e informações adicionais sobre estes dados. Estas informações podem ser, por exemplo, o tipo do conteúdo armazenado e o endereço na memória aonde o objeto está localizado. Os objetos podem ser desde números inteiros até funções.

Os rótulos são identificadores compostos por um ou mais caracteres concatenados, sempre iniciando com uma letra (A–Z ou a–z), sublinhado (_) ou um subconjunto de códigos Unicode maiores do que U+00A0. Os demais caracteres podem incluir números, letras, mais sublinhados ou outros códigos Unicode. É importante saber também que a Julia faz a diferenciação entre letras minúsculas e minúsculas.

Na prática, é comum nos referirmos a um dado representado por um rótulo e seu objeto usando simplesmente o termo *variável*. Assim, uma variável é um nome associado (ou vinculado) a um valor, ou seja, um local na memória com um nome específico, utilizado para armazenar valores que podem ser resgatados, alterados ou manipulados a qualquer momento.

Ao criar um nome para uma variável, recomenda-se:

1. não utilizar palavras reservadas, como: `if`, `while`, `function`, etc;
2. evitar caracteres especiais ou espaços em branco;
3. utilizar o estilo *snake_ case* (palavras separadas por sublinhados). Por exemplo, ao invés de usar `alturaDoTriangulo`, optar por `altura_do_triangulo`.

3.3 Tipos de Dados

A Julia é uma linguagem de tipagem dinâmica, isso significa que o usuário não precisa declarar explicitamente o tipo de uma variável. O tipo é inferido de acordo

com o valor atribuído à variável. Todavia, a Julia permite também que o tipo seja definido, caso necessário.

Muitos são os tipos de dados disponíveis na Julia. Os que serão explorados neste trabalho e, por isso, merecem destaque são:

- `Int`: usado para armazenar números inteiros;
- `Float`: usado para armazenar números reais;
- `String`: usado para armazenar cadeias de caracteres, como as que ocorrem em textos;
- `BigInt` e `BigFloat`: usados para armazenar números inteiros e reais grandes, que necessitam de mais do que os 64-bits ocupados pelos tipos `Int` e `Float`.

O Código 3.1 é um exemplo de como podemos usar o comando `typeof` para exibir o tipo de variável determinado pelo `julia`. Além disso, foram usadas a função `factorial`, cujo resultado é o fatorial de um número inteiro, e a função `big`, que converte um inteiro em um `BigInt`.

Código 3.1: Visualização dos tipos de variáveis atribuídos pelo `julia`.

```
1 x = 10
2 y = 2.5
3 nome = "Olá, Mundo"
4 fatorial = factorial(big(30))
5 typeof(x),typeof(y),typeof(nome),typeof(fatorial)
```

Fonte: Elaborado pelo autor.

Ao executar o Código 3.1, o resultado será:

```
(Int64, Float64, String, BigInt)
```

Pelos valores exibidos na saída, podemos ver que `x` é uma variável do tipo `Int64`, `y` é um `Float64`, `nome` é do tipo `String` e `fatorial` é um `BigInt`.

3.4 Comentários

Durante a elaboração de um programa, é recomendado adicionar explicações ou descrições referentes ao funcionamento de trechos do código-fonte, conhecidas como *comentários*. Durante a execução do código, os comentários são ignorados pelo interpretador/compilador, ou seja, não interferem no fluxo do código. O uso adequado de comentários é indispensável, pois não só facilita a compreensão do que está sendo desenvolvido, mas também contribui para verificações e manutenções futuras. Além

disso, eles permitem que outra pessoa, ao acessar o código, consiga compreender mais rapidamente a lógica implementada.

Em Julia, utilizamos `#` e `#= =#` para comentários curtos e longos, respectivamente. No caso de comentários curtos, ocupando somente uma linha, procedemos como ilustrado no Código 3.2.

Código 3.2: Exemplo de comentário de linha única.

```
x = 3.14  # Uma aproximação para o valor de pi
```

Fonte: Elaborado pelo autor.

Para comentários de múltiplas linhas, utilizamos `#=` e `=#` para marcar o início e o fim do comentário, respectivamente. Isto é útil também para desativar porções de código, como ilustrado no Código 3.3.

Código 3.3: Exemplo de comentário de múltiplas linhas.

```
#=
Este é um comentário de múltiplas linhas.
Ele é útil para incluir explicações detalhadas
ou desativar partes do código durante o desenvolvimento.
=#
lado = 2.0  # Esta parte do código será executada
#=
perimetro = 4*lado  # Esta parte do código não será executada
=#
area = lado*lado  # Esta parte do código será executada
```

Fonte: Elaborado pelo autor.

Vale ressaltar que é uma boa prática evitar comentários óbvios ou redundantes. O ideal é inserir comentários para explicar o que não é evidente no código, fornecendo esclarecimento ao que não é intuitivo. Comentários desnecessários não agregam valor algum à compreensão do código.

3.5 Indentação

A indentação de um código refere-se ao espaçamento ou tabulação usados no início das linhas de código. Sua finalidade é indicar a hierarquia entre blocos de código e melhorar a organização e legibilidade. Diferentemente do que acontece em Python, em Julia, o uso de indentação não é obrigatório, pois o término de um bloco de

código será indicado pela palavra-chave `end`. No entanto, é altamente recomendado, pois facilita a compreensão do programa e torna sua estrutura mais clara.

3.6 Entrada e saída

Em praticamente todos os códigos apresentados no restante deste trabalho, são empregadas as funções básicas de entrada e saída da Julia: `print`, `println`, `parse` e `readline`. Essas funções são essenciais para a criação de aplicações minimamente interativas.

`print`

Essa função tem como sintaxe: `print(x)`. Ela é usada para imprimir/exibir o valor de `x` na saída padrão, que geralmente é o terminal.

O Código 3.4, tem como resultado:

```
Olá, Mundo! Essa mensagem foi escrita em linguagem Julia.
```

Código 3.4: Impressão de uma sentença usando várias chamadas à função `print`.

```
1 print("Olá, ")
2 print("Mundo! ")
3 print("Essa mensagem foi escrita em linguagem ")
4 print("Julia.")
```

Fonte: Elaborado pelo autor.

O mesmo resultado poderia ter sido obtido usando apenas uma linha, como no Código 3.5. No entanto, quisemos enfatizar o fato de que a função `print` não faz quebra de linhas automaticamente.

Código 3.5: Impressão de uma sentença usando uma única chamada à função `print`.

```
1 print("Olá, Mundo! Essa mensagem foi escrita em linguagem Julia."),
```

Fonte: Elaborado pelo autor.

Poderíamos ainda, ter atribuído a mensagem a uma variável e usar a função `print` para mostrar o conteúdo dessa variável (Código 3.6).

Código 3.6: Impressão de uma sentença usando a função `print` com uma variável auxiliar.

```
1 mensagem = "Olá, Mundo! Essa mensagem foi escrita em linguagem Julia"
2 print(mensagem)
```

Fonte: Elaborado pelo autor.

Vejamos como a `print` pode ser útil em um exemplo de código que calcula a área e o perímetro de um retângulo de base 7,5 m e altura 10 m (Código 3.7).

Código 3.7: Exibindo a área e o perímetro de um retângulo de dimensões fixas usando a `print`.

```
1 b = 7.5  # base
2 h = 10   # altura
3 perimetro = 2 * (b + h)
4 area = b * h
5 print("O retângulo possui ")
6 print(perimetro)
7 print(" m de perímetro e ")
8 print(area)
9 print(" m² de área.")
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.7 será:

O retângulo possui 35.0 m de perímetro e 75.0 m² de área.

Observe que foi usado o comando `print` repetidas vezes. Temos a possibilidade de reescrever o Código 3.7 substituindo as linhas 5–9 por:

```
print("O retângulo possui $perimetro m de perímetro e $area m² de área.")
```

O cifrão `$` que escrevemos precedendo as variáveis `perimetro` e `area` serve para mostrar os valores atribuídos a elas. Outra maneira de se obter o mesmo resultado é escrevermos diretamente as expressões que definem o perímetro e a área entre parênteses, também precedidas de cifrão, reduzindo assim a quantidade de variáveis:

```
print("O retângulo possui $(2 * (b + h)) m de perímetro e "*
      "$(b * h) m² de área.")
```

Aqui, para quebrar a linha de código em duas linhas, evitando extrapolar os limites das margens deste texto, separamos a sentença em duas cadeias de caracteres, concatenadas pelo operador `*`.

`println`

A função `println` possui as mesmas funcionalidades da `print`. O que as difere é que a função `println` realiza uma quebra de linha automática ao final da saída, isto é, exibe as informações organizadas em linhas separadas.

Retomando o problema de se calcular o perímetro e a área de um retângulo, da seção anterior, poderíamos usar o Código 3.8.

Código 3.8: Exibindo a área e o perímetro de um retângulo de dimensões fixas usando a `println`.

```
1 b = 7.5 # base
2 h = 10  # altura
3 println("Perímetro do retângulo: $(2 * (b + h)) m")
4 println("Área do retângulo: $(b * h) m²")
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.8 será:

```
Perímetro do retângulo: 35.0 m
Área do retângulo: 75.0 m²
```

`readline`

A `readline` é uma função de entrada que captura dados informados pelo usuário através do teclado. Essa função pausa o programa até o usuário informar algo e pressionar a tecla *Enter*. A informação passada é armazenada em uma variável do tipo `String`, desconsiderando a quebra de linha. Sua sintaxe é:

```
variável = readline()
```

Caso os dados de entrada sejam valores numéricos, a `readline` precisa ser usada em combinação com a `parse`, descrita a seguir.

`parse`

A função parse converte uma variável do tipo `String` para um outro tipo desejado, como `Int` ou `Float`, por exemplo. Isso é útil quando deseja-se processar valores numéricos informados pelo usuário.

Para exemplificar sua utilidade, criaremos um código que constrói uma tabuada de multiplicação de modo interativo. Nele, é solicitado um valor inteiro ao usuário e exibido na tela a tabuada correspondente (Código 3.9).

Código 3.9: Tabuada de multiplicação interativa usando a `readline` e a `parse`.

```
1  println("Você deseja calcular a tabuada de qual número?: ")
2  a = parse(Int, readline())
3  println("* TABUADA DE MULTIPLICAÇÃO DO $a *" )
4  println("1 x $a = $(1*a)" )
5  println("2 x $a = $(2*a)" )
6  println("3 x $a = $(3*a)" )
7  println("4 x $a = $(4*a)" )
8  println("5 x $a = $(5*a)" )
9  println("6 x $a = $(6*a)" )
10 println("7 x $a = $(7*a)" )
11 println("8 x $a = $(8*a)" )
12 println("9 x $a = $(9*a)" )
13 println("10 x $a = $(10*a)")
```

Na linha 2, o comando `readline` armazenará o valor informado pelo usuário em uma variável temporária e o comando `parse` converterá esse valor para o tipo `Int`, cujo resultado será atribuído à variável `a`. As linhas seguintes são responsáveis por imprimir o cabeçalho e o corpo da tabuada.

Ao executar o Código 3.9 e informar na entrada o valor 7, o resultado será:

```
Você deseja calcular a tabuada de qual número?:
stdin> 7
* TABUADA DE MULTIPLICAÇÃO DO 7 *
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
```

É importante alertar que, no caso particular do Jupyter Notebook, cada chamada à `readline` faz surgir o texto `stdin>` na saída, o que não acontece quando o mesmo código é executado no CMD ou no VSCode, por exemplo.

3.7 Operadores Aritméticos

As operações aritméticas na Julia funcionam de forma simples e intuitiva. Elas possuem sintaxe similar a outras linguagens de programação. A Tabela 3.1 contém uma lista de todas as operações aritméticas binárias disponíveis.

Tabela 3.1: Operadores aritméticos suportados por todos os tipos numéricos primitivos da Julia.

Operação	Símbolo	Descrição
Adição	$+$	Soma dois números
Subtração	$-$	Subtrai um número de outro
Multiplicação	$*$	Multiplica dois números
Divisão	$/$	Divide um número pelo outro
Divisão inversa	$\backslash$	Retorna o inverso da divisão com barra (/)
Divisão inteira (`\div` + *Tab*)	$\div$	Retorna o quociente da divisão inteira
Resto da divisão	`%`	Retorna o resto da divisão
Potência	`^`	Eleva um número a uma potência

Fonte: Elaborada pelo autor.

A seguir, ilustremos como usar os operadores aritméticos em Julia a partir de problemas matemáticos simples.

Exemplo 3.1. Considere o problema de desenvolver um programa que solicite ao usuário um número inteiro. O programa deve calcular e exibir o antecessor e o sucessor desse número de forma clara e organizada.

Demonstração. Uma solução para esse problema está apresentada no Código 3.10. Inicialmente, nas linhas 1–3, criamos uma caixa retangular contornada com asteriscos, com um título que descreve o código em seu interior. Isto é feito com o auxílio do comando `repeat`, que repete o valor de seu primeiro argumento na quantidade de vezes indicada pelo segundo argumento.

Após ler um valor inteiro `num` na linha 6, o antecessor `ant` e o sucessor `suc` são calculados nas linhas 8 e 9 usando os operadores `-` e `+`. O código termina exibindo os valores calculados.

Código 3.10: Cálculo do antecessor e do sucessor de um número inteiro.

```
println(repeat("*", 25))
println("* ANTECESSOR E SUCESSOR *")
println(repeat("*", 25))
# Solicita um número inteiro ao usuário
println("Digite um número inteiro: ")
num = parse(Int,readline())
# Calcula o antecessor e o sucessor
ant = num - 1
suc = num + 1
# Exibe o resultado de forma clara
println("O número digitado foi $num.")
println("Seu antecessor é $ant.")
println("Seu sucessor é $suc.")
```

Fonte: Elaborado pelo autor

O resultado da execução do Código 3.10 para `num` igual a 100 é:

```
*************************
* ANTECESSOR E SUCESSOR *
*************************
Digite um número:
stdin> 100
O número digitado foi 100.
Seu antecessor é 99.
Seu sucessor é 101.
```

□

□

Exemplo 3.2. Ilustraremos agora a diferença entre a divisão inteira e a real. Para isso, elaboraremos um programa que lê a partir do teclado dois números inteiros e, com base nesses valores, calcula e exibe o seguinte:

- O resultado da divisão considerando os operandos reais;
- O quociente da divisão inteira, com seu resto.

Vimos na Tabela 3.1 que há comandos específicos para realizar a operação de divisão de acordo com o resultado desejado. Para a divisão real, usamos o operador `/`; para a divisão inteira, devemos usar o `\div`. O resto é determinado pelo operador `%`. Estes cálculos são realizados nas linhas 10–12. Observe que, na divisão inteira,

o símbolo que surge é o ÷. Isto acontece quando se digita o comando `\div` e em seguida pressionamos a tecla *Tab*. A substituição é feita automaticamente.

Código 3.11: Cálculo da divisão inteira e real de dois números informados pelo usuário.

```
1  println(repeat("*", 26))
2  println("* DIVISÃO INTEIRA E REAL *")
3  println(repeat("*", 26))
4  # Bloco de interação com o usuário
5  println("Digite um número para o dividendo")
6  num = parse(Int,readline())
7  println("Digite um número para o divisor")
8  den = parse(Int,readline())
9  # Bloco de cálculos
10 quoc     = num / den
11 resto    = num % den
12 quoc_int = num ÷ den
13 # Bloco de exibição formatada dos resultados
14 println("A divisão a ser realizada é: $num ÷ $den")
15 println("O resultado da divisão real é: $quoc")
16 println("O resultado da divisão inteira é: $quoc_int")
17 println("O resto da divisão inteira é: $resto")
```

Fonte: Elaborado pelo autor.

Ao executar o Código 3.11 e informar na entrada os valores 2025 e 2, o resultado será:

```
**************************
* DIVISÃO INTEIRA E REAL *
**************************
Digite um número para o dividendo
stdin> 2025
Digite um número para o divisor
stdin> 2
A divisão a ser realizada é: 2025 ÷ 2
O resultado da divisão real é: 1012.5
O resultado da divisão inteira é: 1012
O resto da divisão inteira é: 1
```

□

Agora iremos ilustrar uma funcionalidade peculiar da Julia, a capacidade intrínseca de operar com frações.

Exemplo 3.3. Dados dois números racionais, desejamos calcular e exibir os resultados da soma, diferença, produto e quociente entre eles.

Na Julia, realizar operações com frações usando é uma tarefa simples e intuitiva. Para isso, precisamos utilizar operandos do tipo `Rational`. Podemos construir um `Rational` lendo e informando seu numerador e denominador como a seguir:

```
num = parse(Int, readline())
den = parse(Int, readline())
x = Rational(num, den)
```

ou como está sendo feito no Código 3.12, usando o operador `//`. Nele, usamos o comando `parse` para converter o texto digitado para o tipo `Rational{Int}`, que representa números racionais com numerador e denominador inteiros.

Código 3.12: Operações aritméticas com números racionais.

```
println(repeat("*",25))
println("* OPERAÇÕES COM FRAÇÕES *")
println(repeat("*",25))
# Dados de entrada
println("Digite uma fração no formato a//b:")
f1 = parse(Rational{Int}, readline())
println("Digite o valor da segunda fração:")
f2 = parse(Rational{Int}, readline())
# Operações com as frações
soma = f1 + f2
dif  = f1 - f2
prod = f1 * f2
quoc = f1 / f2
# Bloco de exibição
println("A soma: $f1 + $f2 = " , soma)
println("A diferença: $f1 - $f2  = ", dif)
println("O produto: $f1 × $f2 = ", prod)
println("O quociente: $f1 ÷ $f2 = ", quoc)
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.12 será:

```
*************************
* OPERAÇÕES COM FRAÇÕES *
*************************
Digite uma fração no formato a//b:
stdin> 2//4
```

```
Digite o valor da segunda fração:
stdin> 5//6
A soma: 1//2 + 5//6 = 4//3
A diferença: 1//2 - 5//6  = -1//3
O produto: 1//2 × 5//6 = 5//12
O quociente: 1//2 ÷ 5//6 = 3//5
```

Observe que a fração 2/4 foi automaticamente transformada em 1/2, sua forma irredutível. Os resultados das operações também são simplificados, sempre que possível. □

Na resolução de expressões numéricas, a Julia segue a mesma ordem de precedência estabelecida na matemática:

1º) Parênteses, colchetes e chaves {[( )]}. Vale ressaltar que, em linguagens de programação, geralmente substituímos as chaves e colchetes por parênteses, e a resolução ocorre dos mais internos para os mais externos;
2º) Expoentes (potência e raiz);
3º) Multiplicação, divisão e módulo (da esquerda para a direita);
4º) Adição e subtração (da esquerda para a direita).

3.8 Operadores Boolianos e Relacionais

Os operadores boolianos servem para mimetizar os conectivos e (∧), ou (∨) e não (¬) da lógica proposicional. Eles estão descritos na Tabela 3.2.

Tabela 3.2: Operadores boolianos em Julia.

Operação	Símbolo	Descrição
Negação (NOT)	`!x`	Nega o valor lógico
E (AND)	`x && y`	Verdadeiro, se ambas as condições forem verdadeiras
OU (OR)	`x || y`	Verdadeiro, se pelo menos uma condição for verdadeira

Fonte: Elaborado pelo autor.

Para comparações numéricas, temos os operadores relacionais da Tabela 3.3.

Tabela 3.3: Operadores relacionais em Julia.

Operação	Símbolo	Descrição
Igualdade	`==`	Verifica se dois valores são iguais
Diferente	`!=`	Verifica se dois valores são diferentes
Maior que	`>`	Verifica se um valor é maior que outro
Maior ou igual a	`>=`	Verifica se um valor é maior ou igual a outro
Menor que	`<`	Verifica se um valor é menor que outro
Menor ou igual a	`<=`	Verifica se um valor é menor ou igual a outro

Fonte: Elaborado pelo autor.

Exemplo 3.4. A proposição composta:

$$(3 + 2 = 6) \wedge (3 \neq 10)$$

expressa em Julia, possui a seguinte forma:

```
3 + 2 == 6 && 3 != 10
```

e tem como resultado:

```
false
```

O mesmo resultado poderia ser obtido usando o comando `\neq`:

```
3 + 2 == 6 || 3 ≠ 10
```

e isso deixaria o código visualmente mais próximo da sua notação matemática. □

3.9 Funções Matemáticas Elementares

Além de operadores, a Julia conta com uma enorme variedade de funções nativas bastante relevantes para o desenvolvimento de atividades matemáticas.

> Julia fornece uma coleção abrangente de funções e operadores matemáticos. Essas operações matemáticas são definidas em uma classe tão ampla de valores numéricos quanto permitem definições sensatas, incluindo inteiros, números de ponto flutuante, racionais e complexos, onde quer que tais definições façam sentido (JULIA DOCUMENTATION, s.d.).

Essas funções englobam funções exponenciais, logarítmicas e trigonométricas, cálculo de fatorial, determinação da hipotenusa de triângulos retângulos, arredondamentos, valor absoluto, sorteios pseudo-aleatórios, entre outras. Algumas delas que são exploradas neste trabalho estão apresentadas na Tabela 3.4.

No caso das funções trigonométricas, por padrão, a Julia vem configurada para realizar os cálculos em radianos. Caso seja necessário usar ângulos em graus, basta acrescentar o sufixo `d` ao nome da função desejada. Por exemplo, `sind(x)` calcula o seno de x, onde x é especificado em graus. De modo similar, acrescentamos o prefixo `a` para acessar as funções trigonométricas inversas. Com isso, o comando:

```
asind(sqrt(2)/2)
```

retornará:

```
45.00000000000001
```

Tabela 3.4: Algumas funções matemáticas elementares da Julia.

Comando	Descrição
`sqrt(x)`	Cálculo de raiz quadrada
`cbrt(x)`	Cálculo de raiz cúbica
`hypot(x,y)`	Cálculo da hipotenusa
`lcm(x,y)`	Cálculo de MMC entre x e y
`gcd(x,y)`	Cálculo de MDC entre x e y
`factorial(n)`	Cálculo do fatorial de n
`union(A, B)`	O operador de união, $A \cup B$
`intersect(A, B)`	O operador de interseção, $A \cap B$
`exp(x)`	Função exponencial natural em x
`log(x)`	Logaritmo natural de x
`log(b, x)`	Logaritmo de base b de x
`log2(x)`	Logaritmo de base 2 de x
`log10(x)`	Logaritmo de base 10 de x
`abs(x)`	Calcula o valor absoluto de x
`round(x, digitos=b)`	Arredonda x com b casas decimais
`trunc(x)`	Remove a parte decimal sem arredondar
`floor(x)`	Arredonda para baixo (menor inteiro)
`ceil(x)`	Arredonda para cima (maior inteiro)
`sin(x)`	Cálculo do seno do ângulo x
`cos(x)`	Cálculo do cosseno do ângulo x
`tan(x)`	Cálculo da tangente do ângulo x
`cot(x)`	Cálculo da cotangente do ângulo x
`sec(x)`	Cálculo da secante do ângulo x
`csc(x)`	Cálculo da cossecante do ângulo x

Fonte: Elaborado pelo autor.

A seguir, usaremos algumas dessas funções para resolver exercícios.

Exemplo 3.5 (Exercícios Brasil Escola). Sobre os ângulos notáveis, julgue as afirmativas a seguir:

I. O valor do seno de 30° é igual ao valor do cosseno de 60°.
II. O valor da tangente de 45° é igual ao valor do cosseno de 45°.
III. O valor do seno de 60° é igual ao valor do cosseno de 60°.

Marque a alternativa correta:

A) Somente a afirmativa I é verdadeira.
B) Somente a afirmativa II é verdadeira.
C) Somente a afirmativa III é verdadeira.
D) Nenhuma das afirmativas é verdadeira.

Para resolver este problema, podemos testar cada uma das alternativas fazendo, como no Código 3.13.

Código 3.13: Resolvendo o Exemplo 3.5 usando as funções trigonométricas `sind`, `cosd` e `tand`.

```
1 println("A afirmativa I: ", sind(30)==cosd(60))
2 println("A afirmativa II: ", tand(45)==cosd(45))
3 println("A afirmativa III: ", sind(60)==cosd(60))
```

Fonte: Elaborado pelo autor.

Assim, ao executá-lo, teremos:

```
A afirmativa I: true
A afirmativa II: false
A afirmativa III: false
```

Portanto, concluímos que apenas a afirmativa I é verdadeira. Isso torna a alternativa A correta. □

Exemplo 3.6 (IFG – 2020). O desmatamento tem sido uma problemática crescente no Brasil. Supondo que, ao efetuar o desmatamento de uma determinada área, um madeireiro se depara com uma árvore que já se encontra quebrada; parte do tronco da árvore que se manteve fixa ao solo mede 3 m e forma com este um ângulo de 90°; a ponta da parte quebrada que toca o solo encontra-se a 4 m de distância da base da árvore. Qual era a altura da árvore antes de se quebrar?

A) 5 m B) 7 m C) 8 m D) 9 m

Este problema pode ser resolvido com o uso do Teorema de Pitágoras, já que a árvore quebrada forma um triângulo retângulo com o solo. Dessa forma, temos que

os catetos são representados pela parte do tronco da árvore que permanece fixa e mede 3 m, e pela distância que há entre a parte quebrada que toca o solo e a base da árvore, que mede 4 m. A parte da árvore que se encontra quebrada representa a hipotenusa. O intuito desta questão é encontrar a medida da hipotenusa e somar com a medida do tronco que permaneceu fixa.

No Código 3.14 utilizamos a função `hypo` para obter a hipotenusa e acrescentar os 3 m referentes ao tronco fixo, obtendo assim o comprimento total da árvore.

Código 3.14: Resolvendo o Exemplo 3.6 usando a função `hypot`.

```
hip = hypot(3, 4)
println("A hipotenusa mede $hip metros.")
println("A altura da árvore é de $(hip + 3) metros.")
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.14 será:

```
A hipotenusa mede 5.0 metros.
A altura da árvore é de 8.0 metros.
```

Portanto, a árvore media 8 m, tornando o item C a alternativa correta. □

Exemplo 3.7 (FAMEMA – 2020). Sílvia e Márcio moram em cidades diferentes no interior. Sílvia vai à capital uma vez a cada 10 dias, e Márcio vai à capital uma vez a cada 12 dias. A última vez em que eles se encontraram na capital foi um sábado. O próximo encontro dos dois na capital ocorrerá em:

a) uma terça-feira.
b) uma quarta-feira.
c) um domingo.
d) um sábado.

Para sabermos a quantidade de dias que é necessária para que Sílvia e Márcio se encontrem novamente na capital, precisamos calcular o mínimo múltiplo comum (MMC) entre 10 e 12, que é o intervalo de dias das viagens deles. De acordo com a Tabela 3.4, `lcm` é a função que calcula MMC. Agora resta saber em que dia ocorrerá esse encontro. Como uma semana possui sete dias, o resto da divisão do MMC por 7 nos dará essa resposta. Esta lógica está implementada no Código 3.15.

Código 3.15: Resolvendo o Exemplo 3.7 usando a função `lcm`.

```
mmc = lcm(10, 12)
println("O MMC(10, 12) vale ", mmc)
println("O resto de $mmc dividido por 7 é $(mmc % 7).")
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.15 será:

```
O MMC(10, 12) vale 60
O resto de 60 dividido por 7 é 4.
```

A partir do sábado, que foi o último dia do encontro deles, contaremos mais quatro dias e chegaremos a uma quarta-feira. Tornado o item b) a alternativa correta. □

Exemplo 3.8 (IBADE – 2024). Considerando os conjuntos:

$$A = \{1, 3, 4, 6\}, \quad B = \{4, 5, 8, 9\}, \quad C = \{2, 3, 5, 8\}$$

assinale a alternativa que representa corretamente o conjunto $(A \cup B) \cap C$.

a) $\{5, 8, 9\}$
b) $\{1, 3, 4, 8, 9\}$
c) $\{4, 5, 8, 9\}$
d) $\{3, 5, 8\}$
e) $\{2, 3, 5, 8\}$

Na linguagem de programação Julia, embora haja o tipo `Set` dedicado a conjuntos não ordenados, podemos representar conjuntos cujos elementos são todos de um mesmo tipo através de arranjos unidimensionais, usando o tipo `Vector`. Com isso, podemos usar as operações em conjuntos disponíveis, por exemplo, `union`, `intersect`, `complement` e `setdiff`. O Código 3.16 explora isto.

Código 3.16: Resolvendo o Exemplo 3.8 usando as funções `union` e `intersect`.

```
A = [1, 3, 4, 6]
B = [4, 5, 8, 9]
C = [2, 3, 5, 8]
println("(A ∪ B) ∩ C = ", intersect(union(A, B), C))
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.16 será:

```
(A ∪ B) ∩ C = [3, 5, 8]
```

correspondendo à alternativa D. □

Exemplo 3.9 (Quadrix – 2021). O resultado da divisão do número de anagramas da palavra PRIMAVERA pelo número de anagramas da palavra INVERNO é um quadrado perfeito. Classifique esta afirmação em verdadeiro ou falso.

Aqui, usaremos a função `factorial` e o fato de que o número de anagramas de uma palavra de n letras, onde algumas letras se repetem a vezes, b vezes, c vezes $\cdots$, é dado por:

$$\frac{n!}{a! \times b! \times c! \times \cdots}$$

A palavra PRIMAVERA possui 9 letras, sendo que as letras "R" e "A" aparecem duas vezes cada. Já a palavra INVERNO, possui 7 letras, sendo que a letra "N" aparece duas vezes. Assim, sendo `prim` e `inv` as variáveis que armazenam a quantidade de anagramas formados pelas palavras PRIMAVERA e INVERNO, respectivamente, a solução deste problema será calculada pelo Código 3.17.

Código 3.17: Resolvendo o Exemplo 3.9 usando a função `factorial`.

```
prim = factorial(9) / (factorial(2) * factorial(2))
inv = factorial(7) / factorial(2)

println("A quantidade de anagramas de PRIMAVERA é: $prim")
println("A quantidade de anagramas de INVERNO é: $inv")
println("A divisão de $prim por $inv é: $(prim / inv)")
println("A raiz quadrada é dada por √$(prim / inv) = $(√(prim / inv))")
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.17 será:

```
A quantidade de anagramas de PRIMAVERA é: 90720.0
A quantidade de anagramas de INVERNO é: 2520.0
A divisão de 90720.0 por 2520.0 é: 36.0
A raiz quadrada é dada por √36.0 = 6.0
```

Logo, a afirmação inicial é verdadeira, pois 36 é um quadrado perfeito. □

Exemplo 3.10 (CONSULPAM – 2024). Sabendo que $A = \log_{10}^{1000}$, $B = \log_{7}^{49}$ e $C = \log_{3}^{\frac{1}{81}}$, calcule $A + B - C$.

A) 1 B) -1 C) 5 D) 9

Vimos na Tabela 3.4 que a sintaxe das funções que calculam logaritmos depende da base. Para bases comuns, como 2, e e 10, temos `log2`, `log` e `log10`, respectivamente. Estas funções recebem um único parâmetro de entrada. No caso de uma base arbitrária, a sintaxe é `log(base, argumento)`. Por conveniência, usaremos apenas esta última na solução do problema posto inicialmente (Código 3.18).

Código 3.18: Resolvendo o Exemplo 3.10 usando a função `log`.

```
1 println("A = ", log(10, 1000))
2 println("B = ", log(7, 49))
3 println("C = ", log(3, 1/81))
4 println("A + B - C = ", log(10, 1000) + log(7, 49) - log(3, 1/81))
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.18 será:

```
A = 3.0
B = 2.0
C = -4.0
A + B - C = 9.0
```

Portanto, $A + B - C = 9$, tornando correta a alternativa D. □

3.10 Funções Definidas pelo Usuário

Por fim, apresentaremos um comando importantíssimo em linguagens de programação: o `function`. Esse comando serve para fazer o encapsulamento de códigos, prática que possibilita o uso posterior de um código salvo como função.

A sintaxe básica é a seguinte:

Código 3.19: Sintaxe básica para definição de uma função em Julia.

```
1 function nome_da_funcao(parametro1, parametro2, ...)
2     # corpo da função
3     return resultado
4 end
```

Fonte: Elaborado pelo autor.

Caso a função não tenha um valor de retorno, a linha 3 do Código 3.19 pode ser omitida.

Exemplo 3.11. Para ilustrar o comando `function`, definimos uma função que realiza a soma de dois números no Código 3.20. Na linha 1, temos o nome atribuído à função e a lista de seus parâmetros entre parênteses. Na linha 2, o comando `return` fará o retorno do resultado da expressão `a + b`. A definição da função é encerrada com a palavra-chave `end`. Ao executar o código no Jupyter Notebook, a mensagem

```
soma (generic function with 1 method)
```

será apresentada, indicando que a função foi definida corretamente e está pronta para uso.

Código 3.20: Exemplo de função definida pelo usuário em Julia.

```
1 function soma(a,b)
2     return a + b
3 end
```

Fonte: Elaborado pelo autor.

Agora, para somar os números 2 e 3, basta executar:

```
soma(2,3)
```

cujo resultado será:

```
5
```

□

3.11 Unicode e LaTeX

O Unicode é um padrão para a representação de caracteres. Ele foi desenvolvido pelo Unicode Consortium entre 1988 e 1991. Ao usar dois bytes (16 bits) para representar cada caractere, o Unicode permite representar um total de $2^{16} = 65.536$ símbolos. Desta forma, quase todas as línguas escritas do mundo são suportadas, contornando limitações das tabelas de caracteres tradicionais (FETTER; LIMA; LIMA, s.d.).

Cada caractere em Unicode é representado por um número específico em base hexadecimal no formato U+XXXX, onde X representa um dígito hexadecimal (0-9, A-F).

De modo geral, os símbolos Unicode estão organizados assim:

U+0000 a U+007F: ASCII Básico, que inclui caracteres alfabéticos (A–Z, a–z), dígitos decimais (0–9), e sinais de pontuação comuns.

U+0080 a U+07FF: Extensões de caracteres para idiomas que usam alfabetos além do latino, como grego, cirílico, árabe, etc.

U+0800 a U+FFFF: Símbolos e emojis, incluindo uma vasta gama de símbolos matemáticos, caracteres de outras línguas, e também emojis e pictogramas.

U+10000 a U+10FFFF: Caracteres suplementares, que incluem emojis avançados, caracteres históricos e simbólicos, e outras adições que não se encaixam nas categorias anteriores.

Devido à sua capacidade de representar uma ampla variedade de símbolos matemáticos, o uso de caracteres Unicode na linguagem de programação Julia permite-nos desenvolver códigos com saídas mais amigáveis, fazendo com que o conteúdo seja exibido de forma precisa e compreensível para os alunos.

Para Santos Lima e Neto (2024), outro aspecto interessante da linguagem Julia, especialmente útil para as ciências matemáticas, é o suporte a elementos Unicode integrados ao código, que permite a utilização de símbolos matemáticos de forma completamente natural. Podemos utilizar, por exemplo, os símbolos "∪" (união) e "∩" (interseção) para realizar operações entre conjuntos. Isso torna a sintaxe da Julia mais próxima da notação matemática. Com isso, é possível reescrever a solução do Exemplo 3.8 como no Código 3.21.

Código 3.21: Resolvendo o Exemplo 3.8 usando as símbolos Unicode.

```
1 A = [1, 3, 4, 6]
2 B = [4, 5, 8, 9]
3 C = [2, 3, 5, 8]
4 println("(A ∪ B) ∩ C = ", (A ∪ B) ∩ C)
```

Fonte: Elaborado pelo autor.

Outro fato interessante sobre os símbolos Unicode no Julia, é que a maioria deles pode ser acessada usando comandos semelhantes aos do LaTeX. Por exemplo, ao digitar `\pi` e pressionar a tecla *Tab*, o resultado será o símbolo π. A Tabela 3.5 traz alguns dos caracteres mais comuns quando se trabalha com matemática e mostra como inseri-los tanto utilizando um código Unicode quanto através de comandos similares aos do LaTeX (THE JULIA LANGUAGE, 2024).

Tabela 3.5: Símbolos matemáticos em Unicode e seus respectivos comandos no estilo do LaTeX disponíveis na Julia.

Descrição	Código U+xxxx	Código + tab	Saída
Símbolos Relacionais e de Conjunto			
Diferente	\u2260	\neq	$\neq$
Menor ou igual	\u2264	\leq	$\leq$
Maior ou igual	\u2265	\geq	$\geq$
Pertence a	\u2208	\in	$\in$
Não pertence a	\u2209	\notin	$\notin$
Conjunto vazio	\u2205	\varnothing	$\varnothing$
União	\u222A	\cup	$\cup$
Interseção	\u2229	\cap	$\cap$
Subconjunto	\u2286	\subseteq	$\subseteq$
Não é subconjunto	\u2288	\nsubseteq	$\nsubseteq$
Conjuntos Numéricos			
Naturais	\u2115	\bbN	$\mathbb{N}$
Inteiros	\u2124	\bbZ	$\mathbb{Z}$
Racionais	\u211A	\bbQ	$\mathbb{Q}$
Reais	\u211D	\bbR	$\mathbb{R}$
Símbolos de Lógica Proposicional			
Conjunção	\u2227	\wedge	$\wedge$
Disjunção	\u2228	\vee	$\vee$
Negação	\u00AC	\neg	$\neg$
Condicional	\u21D2	\Rightarrow	$\Rightarrow$
Bicondicional	\u21D4	\Leftrightarrow	$\Leftrightarrow$
Para todo	\u2200	\forall	$\forall$
Existe	\u2203	\exists	$\exists$
Operadores Matemáticos			
Multiplicação	\u22C5	\cdot	$\cdot$
Multiplicação (alternativa)	\u00D7	\times	$\times$
Divisão	\u00F7	\div	$\div$
Reticência	\u22EF	\cdots	$\cdots$
Expoente 2	\u00B2	\^2	^{2}
Expoente 3	\u00B3	\^3	^{3}
Expoente 4	\u2074	\^4	^{4}
Índice 1	\u2081	_1	$_1$
Índice 2	\u2082	_2	$_2$
Índice 3	\u2083	_3	$_3$
Raiz quadrada	\u221A	\sqrt	$\sqrt{}$
Raiz cúbica de	\u221B	\cbrt	$\sqrt[3]{}$

Fonte: Elaborada pelo autor.

3.12 Controle de Fluxo

Todos os comandos ou funções estudados até o momento funcionam de forma linear, ou seja, sem a necessidade de analisar nenhuma condição prévia à sua execução. No entanto, em alguns casos, é necessário avaliar uma determinada condição e alterar o fluxo de execução do algoritmo. Para essa tarefa, em praticamente todas as linguagens de programação, existe o comando `if`. Sua sintaxe em Julia está detalhada no Código 3.22.

Código 3.22: Sintaxe da estrutura condicional `if`.

```
1  if condição_1
2      # Executado apenas se a condição_1 for verdadeira.
3  elseif condição_2
4      # Executado se apenas a condição_2 for verdadeira.
5  ...
6  elseif condição_n
7      # Executado se apenas a condição_n for verdadeira.
8  else
9      # Executado apenas se condição_1, ..., condição_n forem falsas.
10 end
```

Fonte: Elaborado pelo autor.

Iniciando na linha 1, a `condição_1` será avaliada. Se ela for verdadeira, então o bloco correspondente é executado; caso contrário, a `condição_2` é avaliada. Sendo verdadeira, seu bloco de código é executado; e assim, sucessivamente. O bloco de comandos do `else` será executado apenas quando nenhuma das condições forem verdadeiras.

Caso necessário, os comandos `elseif` e `else` podem ser omitidos. No entanto, sempre que se utiliza o `elseif`, recomenda-se encerrar com o `else`.

Exemplo 3.12. Considere o programa apresentado no Código 3.23. Ele recebe um número inteiro do usuário e determina se ele é par ou ímpar.

Para verificar a paridade de um número inteiro, na linha 6, utiliza-se o operador de resto `%`. Com base no resultado da condição, temos as seguintes situações:

- Se o resto for 0, significa que o número é par, e a mensagem correspondente será impressa pela linha 7.
- Caso contrário (resto igual a 1), o número é ímpar, e outra mensagem será exibida pela linha 9.

Código 3.23: Verificação de paridade de um número inteiro.

```
1  println(repeat("*", 21))
2  println("* Teste de paridade *")
3  println(repeat("*", 21))
4  println("Informe um número.")
5  num = parse(Int, readline())
6  if num % 2 == 0
7      println("$num é um número PAR!")
8  else
9      println("$num é um número ÍMPAR!")
10 end
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.23 para a entrada 2025 será:

```
**********
* Teste de paridade *
**********
Informe um número.
stdin> 2025
2025 é um número ÍMPAR!
```

□

Exemplo 3.13. Agora, considere um programa que solicita ao usuário o comprimento de três segmentos de reta e determina se eles podem formar um triângulo, utilizando a desigualdade triangular.

Inicialmente, para criar esse código, é necessário relembrar o que estabelece a desigualdade triangular. Essa propriedade afirma que, em qualquer triângulo, o comprimento de cada lado deve ser sempre inferior à soma dos comprimentos dos outros dois.

No Código 3.24, após a exibição do cabeçalho (linhas 1 a 3), os comprimentos dos segmentos são solicitados nas linhas 4 a 9, sendo armazenados nas variáveis `a`, `b` e `c` como números reais do tipo `Float64`. A verificação da existência do triângulo é feita na linha 11. Caso as três condições verificadas nessa linha sejam verdadeiras, o programa confirma a validade do triângulo na linha 12; caso contrário, exibe que os valores não satisfazem a condição necessária (linha 14).

Código 3.24: Verificação da existência de um triângulo usando a desigualdade triangular.

```
println(repeat("*", 30))
println("* Existência de um triângulo *")
println(repeat("*", 30))
println("Informe o primeiro segmento.")
a = parse(Float64, readline())
println("Informe o segundo segmento.")
b = parse(Float64, readline())
println("Informe o terceiro segmento.")
c = parse(Float64, readline())
# Aplicação da desigualdade triangular
if a < b + c && b < a + c && c < a + b
    println("Os números $a, $b e $c definem um triângulo.")
else
    println("Os números $a, $b e $c não definem um triângulo.")
end
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.24 para as entradas 1, 2 e 3 será:

```
******************************
* Existência de um triângulo *
******************************
Informe o primeiro segmento.
stdin> 1
Informe o segundo segmento.
stdin> 2
Informe o terceiro segmento.
stdin> 3
Os números 1.0, 2.0 e 3.0 não definem um triângulo.
```

Já para as entradas 3, 4 e 5, teremos:

```
******************************
* Existência de um triângulo *
******************************
Informe o primeiro segmento.
stdin> 3
Informe o segundo segmento.
stdin> 4
Informe o terceiro segmento.
```

```
stdin> 5
Os números 3.0, 4.0 e 5.0 definem um triângulo.
```

□

O comando `if` pode também ser usado dentro dos blocos de execução de um `if`, `elseif` ou `else`. Esta situação é conhecida como aninhamento. Isto é utilizado quando temos mais de uma condição a ser verificada, como no exemplo a seguir.

Exemplo 3.14. Agora, além de verificar a existência do triângulo, desejamos classificá-lo em: equilátero, isósceles ou escaleno.

No Código 3.25, a classificação ocorre entre as linhas 15 a 21. Primeiramente, verificamos se os três lados são iguais (`a == b == c`). Se essa condição for verdadeira, o triângulo é equilátero, e a linha 16 é executada. Caso contrário, o fluxo do programa segue para a linha 17, onde é avaliado se pelo menos dois lados são iguais – ou seja, se `a == b`, `a == c` ou `b == c`. Se alguma dessas expressões for verdadeira, o triângulo é isósceles, e a mensagem correspondente é exibida na linha 18. Por fim, se nenhuma das condições anteriores for satisfeita, o programa conclui que os três lados são diferentes entre si, classificando o triângulo como escaleno.

Código 3.25: Classificação de um triângulo em equilátero, isósceles ou escaleno usando `if`s aninhados.

```
println(repeat("*", 30))
println("* Classificação do triângulo *")
println(repeat("*", 30))

# Os valores de a, b e c são números reais.
println("Informe o primeiro segmento.")
a = parse(Float64, readline())
println("Informe o segundo segmento.")
b = parse(Float64, readline())
println("Informe o terceiro segmento.")
c = parse(Float64, readline())
# Aplicação da desigualdade triangular
if a < b + c && b < a + c && c < a + b
    # Verifica os tipos de triângulo.
    if a == b == c
        println("Os números $a, $b, $c definem um triângulo EQUILÁTERO.")
    elseif a == b || a == c || b == c
        println("Os números $a, $b, $c definem um triângulo ISÓSCELES.")
    else
        println("Os números $a, $b, $c definem um triângulo ESCALENO.")
    end
else
    println("Os números $a, $b e $c não definem um triângulo.")
end
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.25 para as entradas 5, 5 e 5 será:

```
******************************
* Classificação do triângulo *
******************************
Informe o primeiro segmento.
stdin> 5
Informe o segundo segmento.
stdin> 5
Informe o terceiro segmento.
stdin> 5
Os números 5.0, 5.0, 5.0 definem um triângulo EQUILÁTERO.
```

□

3.13 Estruturas de Repetição

Iniciamos essa seção com o desafio simples: criar um código que imprime na tela dez vezes a frase "Olá, mundo!". Naturalmente, com base apenas no conteúdo apresentado até este momento, a solução ingênua para esse problema seria a apresentada no Código 3.26.

Código 3.26: Solução ingênua para o problema de se exibir na tela 10 vezes a mensagem "Olá, mundo!".

```
println("Olá, Mundo!")
println("Olá, Mundo!")
println("Olá, Mundo!")
println("Olá, Mundo!")
println("Olá, Mundo!")
println("Olá, Mundo!")
println("Olá, Mundo!")
println("Olá, Mundo!")
println("Olá, Mundo!")
println("Olá, Mundo!")
```

Fonte: Elaborado pelo autor.

Suponhamos agora que queiramos imprimir a mesma frase, mas agora 1000 vezes. Obviamente, será trabalhoso usar o método empregado no Código 3.26. Para lidar com esse tipo de situação, existem as estruturas de repetição `for` e `while`.

`for`

O uso do `for` é indicado para situações nas quais conhecemos previamente a quantidade de vezes que um bloco de comandos será repetido. Sua sintaxe básica é:

```
for i in valores # Define a variável de controle e seus valores
    # Bloco de comandos a serem executados a cada repetição
end
```

onde a palavra-chave `in` pode ser substituída por `=` ou $\in$.

Agora, a solução do desafio proposto pode ser reduzido a três linhas (Código 3.27).

Código 3.27: Solução para o problema de se exibir na tela 10 vezes a mensagem "Olá, mundo!" usando um `for`.

```
for i in 1:10
    println("Olá, Mundo!")
end
```

Fonte: Elaborado pelo autor.

A seguir, exploramos o `for` para construir novamente a tabuada de multiplicação.

Exemplo 3.15. No Código 3.9, foram necessárias dez linhas para construir o corpo de uma tabuada de multiplicação. Usando o `for`, conseguimos reduzir para três linhas (Código 3.28). Isto torna a tarefa de programar mais rápida e o código conciso.

Código 3.28: Construção da tabuada de multiplicação usando um `for`.

```
println("Você deseja calcular a tabuada de qual número? ")
a = parse(Int, readline())
println(repeat("*", 33))
println("* TABUADA DE MULTIPLICAÇÃO DO $a *")
println(repeat("*", 33))
for i in 1:10
    println("$i x $a = $(i*a)")
end
```

Fonte: Elaborado pelo autor.

□

As estruturas de repetição são extremamente úteis para a resolução de problemas envolvendo estruturas lineares como sequências e séries. A seguir, mostraremos como construir a sequência de Fibonacci.

Exemplo 3.16. A sequência de Fibonacci é uma recorrência de segunda ordem, na qual os dois primeiros termos são conhecidos, e os termos seguintes são obtidos pela soma dos dois termos anteriores. Assim, seus oito primeiros termos são: 1, 1, 2, 3, 5, 8, 13, 21. Matematicamente, ela pode ser expressa por:

$$F_n = \begin{cases} 1, & \text{se } n = 1 \\ 1, & \text{se } n = 2 \\ F_{n-1} + F_{n-2}, & \text{se } n > 2 \end{cases}$$

Há diversos algoritmos possíveis para calculá-la (DASDAN, 2018). O Código 3.29 contém uma implementação talvez do algoritmo mais simples, baseado em um único

laço de repetição. Nele, nas linhas 7 e 8, as variáveis `a` e `b` representam os termos F_1 e F_2, respectivamente. Na linha 10, a variável `i` varia somente de 1 a $n-2$, pois devemos desconsiderar os termos F_1 e F_2, que foram contados e impressos anteriormente. Na linha 11, a variável `c` recebe a soma dos dois termos anteriores, armazenados em `a` e `b`. Os valores de `a` e `b` são atualizados nas linhas 13 e 14, e esse ciclo é repetido até completar as $n-2$ iterações.

Código 3.29: Impressão dos n primeiros termos da sequência de Fibonacci usando um `for`.

```
1  println(repeat("*", 26))
2  println("* SEQUÊNCIA DE FIBONACCI *")
3  println(repeat("*", 26))
4  println("Informe um número.")
5  n = parse(Int, readline())
6  a = 1
7  b = 1
8  println("Estes são os $n primeiros termos de Fibonacci:")
9  print("$a, $b, ")
10 for i in 1:(n-2)
11     c = a + b
12     print("$c, ")
13     a = b
14     b = c
15 end
```

Fonte: Elaborado pelo autor.

Usando este código para exibir os 20 primeiros termos da sequência de Fibonacci, teremos o seguinte resultado:

```
**************************
* SEQUÊNCIA DE FIBONACCI *
**************************
Informe um número.
stdin> 20
Estes são os 20 primeiros termos de Fibonacci:
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
2584, 4181, 6765,
```

□

`while`

O `while` tem a mesma função que o `for`, porém seu uso é indicado quando não sabemos a quantidade de vezes que o bloco de comandos deve ser repetido, ou seja, quando a quantidade de repetições fica restrita a uma condição. Sua sintaxe é:

```
while condição
    # Bloco de comandos a serem executados
end
```

Em suma, ela funciona assim: enquanto a `condição` preestabelecida for verificada como verdadeira, o bloco de comandos será repetido.

Exemplo 3.17. Um problema interessante da teoria de números é determinar se um número natural n é um quadrado perfeito ou não. Para isso, temos que verificar se a raiz quadrada de n também é um número natural. Isto é uma tarefa perfeita para a aplicação do `while`.

Para resolver este problema, utilizaremos o método clássico de extração de raiz quadrada, o qual se fundamenta na busca por dois fatores iguais cujo produto seja igual a n. Esse procedimento é realizado pelas linhas 7–11 do Código 3.30. Nele, além do comando `while`, utilizamos um `if` ao final (linha 13) para verificar se o número fornecido é ou não um quadrado perfeito.

Código 3.30: Verificando se um número é um quadrado perfeito por inspeção.

```
1  println(repeat("*", 31))
2  println("* NÚMEROS QUADRADOS PERFEITOS *")
3  println(repeat("*", 31))
4  println("Informe um número.")
5  num = parse(Int, readline())
6  # Inicializa a variável 'raiz' com zero
7  raiz = 0
8  # Loop para encontrar a raiz quadrada
9  while raiz * raiz < num
10     raiz += 1
11 end
12 # Verifica se o número fornecido é um quadrado perfeito
13 if raiz * raiz == num
14     println("$num é um quadrado perfeito, pois √$num = $raiz.")
15 else
16     println("$num não é um quadrado perfeito, pois √$num não é exata.")
17 end
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.30 para o número 200 será:

```
*******************************
* NÚMEROS QUADRADOS PERFEITOS *
*******************************
Informe um número.
stdin> 200
200 não é um quadrado perfeito, pois √200 não é exata.
```

□

3.14 Bibliotecas

Embora o intuito desse trabalho não seja fazer uso de bibliotecas, dependendo do objeto de conhecimento a ser desenvolvido na aula, podemos recorrer a elas.

> Uma *biblioteca* é uma coleção de funções e rotinas pré-programadas que podem ser utilizadas por desenvolvedores para evitar a repetição de código. As bibliotecas fornecem ferramentas para tarefas comuns, como manipulação de arquivos, cálculos matemáticos e interação com sistemas, permitindo que o programador se concentre em aspectos mais complexos do seu projeto (LARMAN, 2013).

Na Julia, antes de utilizar uma biblioteca pela primeira vez, é necessário instalá-la. Para isso, usamos o gerenciador de pacotes padrão, o `Pkg`, como a seguir:

```
import Pkg
Pkg.add("NomeDaBiblioteca")
```

Uma vez concluída a instalação, a biblioteca pode ser carregada no ambiente com o comando `using`, seguido do nome da biblioteca. A partir daí, estará disponível para uso sempre que necessário.

3.14.1 Gráficos com a Plots.jl

Há várias bibliotecas em Julia capazes de desenhar gráficos, como por exemplo: Plots.jl, Makie, AlgebraOfGraphics.jl e Gadfly. Dentre essas, a mais utilizada e bem documentada é a biblioteca Plots.jl. Ela é a principal ferramenta da Julia para construção de gráficos, oferecendo suporte a vários *backends*, como o GR (padrão do Plots), PyPlot, Plotly, PGFPlotsX, etc, que são os responsáveis pelo visual final do gráfico.

Para instalar a **Plots**, digitamos no `julia` as linhas:

```
import Pkg
Pkg.add("Plots")
```

De agora em diante, sempre que desejarmos ter acesso às funcionalidades da Plots, incluímos em nosso código os comandos:

```
using Plots
gr()
```

A Tabela 3.6 contém uma lista de funções da `Plots` que permitem traçar gráficos de barras, pizza, dispersão, XY e de superfícies.

Tabela 3.6: Funções elementares da biblioteca Plots.jl usadas para traçar gráficos.

Função	Descrição
`bar`	Cria um gráfico de barras
`pie`	Cria um gráfico de setores circulares (pizza)
`scatter`	Plota apenas pontos no plano cartesiano
`plot`	Plota o gráfico de uma função ou conjunto de dados
`surface`	Plota uma superfície no espaço tridimensional

Fonte: Elaborada pelo autor.

Exemplo 3.18 (Castrucci e Júnior (2018)). Em um determinado jardim, a área de plantio de flores obedece à seguinte tabela:

Tipo	Cravo	Lírio	Rosa	Tulipa
Área (em m^2)	4	6	4	12

Construa um gráfico para auxiliar a análise dos dados dessa tabela.

Provavelmente, o gráfico mais natural para esta tarefa seja o gráfico de barras. Para construí-lo, necessitamos basicamente de dois vetores `x` e `y` para armazenar, respectivamente, os nomes das flores e a área de plantio de cada uma delas. Em seguida, fazemos uso da função `bar`, como no Código 3.31.

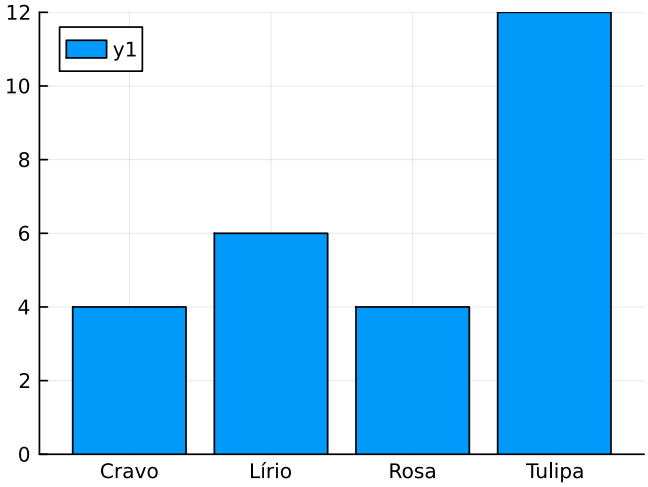
Código 3.31: Construção de um gráfico de barras com a biblioteca Plots.

```
1 using Plots
2 x = ["Cravo", "Lírio", "Rosa", "Tulipa"]
3 y = [4, 6, 4, 12]
4 bar(x,y)
```

Fonte: Elaborado pelo autor.

O resultado do Código 3.31 está apresentado na Figura 3.2.

Figura 3.2: Gráfico de barras gerado pelo Código 3.31.



Fonte: Elaborado pelo autor.

□

Podemos melhorar o gráfico do Exemplo 3.18 adicionando um título para o gráfico, inserindo rótulos para os eixos coordenados e usando cores distintas para cada tipo de flor. Esses ajustes não apenas embelezarão o gráfico, como também facilitarão a leitura e a interpretação dos dados representados. O Código 3.31 implementa esses e outros ajustes, como a utilização de linhas de grade e transparência. A utilidade de cada parâmetro adicionado à chamada da função `bar` foi inserida como comentário no próprio código.

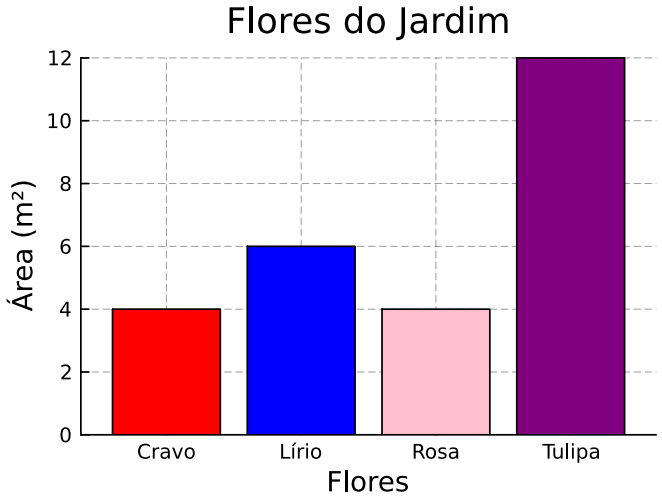
Código 3.32: Versão melhorada do Código 3.31.

```
using Plots
gr(size=(400,300))                          # Define o tamanho do gráfico
x = ["Cravo", "Lírio", "Rosa", "Tulipa"]
y = [4, 6, 4, 12]
    bar(x,y,
    label = nothing,                        # Remove a legenda
    title = "Flores do Jardim",
    xlabel = "Flores",                      # Nome do eixo X
    ylabel = "Área (m²)",                   # Nome do eixo Y
    color = [:red, :blue, :pink, :purple],  # Cores das barras
    grid = :on,                             # Ativa a grade
    gridalpha = 0.4,                        # Transparência da grade
    gridstyle = :dash                       # Estilo tracejado da grade
    )
```

Fonte: Elaborado pelo autor.

O resultado da execução do Código 3.32 será:

Figura 3.3: Versão melhorada do gráfico de barras do Exemplo 3.18.



Fonte: Elaborado pelo autor.

A `Plots` pode ser útil para resolver problemas por inspeção visual, como veremos a seguir.

Exemplo 3.19. Considere as funções $f(x) = \operatorname{sen} x$ e $g(x) = \cos x$. O número de raízes da equação $f(x) = g(x)$ no intervalo $[-2\pi, 2\pi]$ é:

A) 3 B) 4 C) 5 D) 6 E) 7

Aqui, podemos adotar duas abordagens distintas. A primeira consiste em resolver a equação $f(x) = g(x)$ de forma algébrica. Já a segunda seria esboçar os gráficos das funções $\operatorname{sen} x$ e $\cos x$ e contar a quantidade de pontos de interseção entre eles, uma vez que cada ponto de interseção representa uma raiz da equação. Com o auxílio da `Plots`, podemos explorar esta última.

No Código 3.33, utilizamos a função `plot` para construir um gráfico do tipo XY. O domínio das funções f e g está definido na linha 4, enquanto que as funções propriamente ditas são definidas de modo compacto nas linhas 6 e 7. Este tipo de definição de função, em uma linha, é uma característica peculiar da Julia, que pode ser usada apenas para funções que envolvem uma única expressão. O ponto que aparece entre o `sin`/`cos` e a lista de argumentos indica que `x` pode ser um vetor e, nesse caso, essas funções serão aplicadas a cada elemento `x` automaticamente, sem a necessidade de um laço explícito. Esta funcionalidade é chamada de *broadcasting*.

Por fim, o comando `plot` é executado uma vez para cada função. Observe que, na segunda chamada, é usado o símbolo `!` entre `plot` e sua lista de parâmetros. Isto é usado para indicar que o gráfico deve ser desenhado sobre o último gráfico gerado anteriormente, fazendo com que os dois gráficos fiquem sobrepostos.

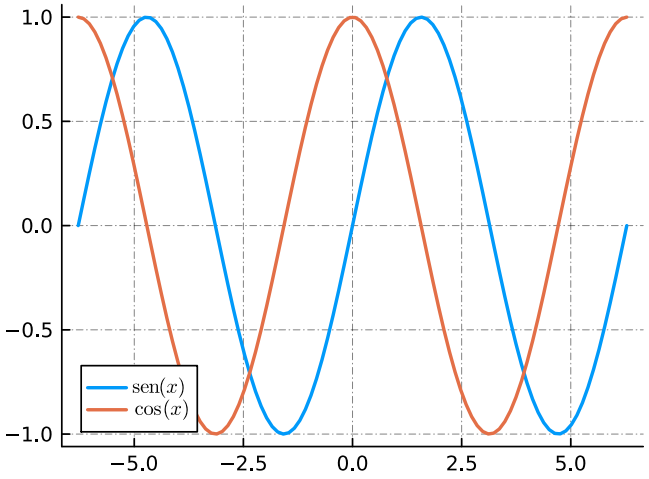
Código 3.33: Representação gráfica das funções trigonométricas sen(x) e cos(x).

```
using Plots
gr(size=(600,500))
# Definição do domínio das funções f e g
x = range(-2pi, 2pi, length=100)
# Regras das funções f e g
f(x) = sin.(x)
g(x) = cos.(x)
# Plota a função f(x) = sen(x)
plot(x, f,
    linewidth = 2,
    label = "\$\\operatorname{sen}(x)\$",
    grid = :on, gridalpha = 0.5, gridstyle = :dashdot)
# Plota a função g(x) = cos(x)
plot!(x, g,
    linewidth = 2,
    label = "\$\\cos(x)\$")
```

Fonte: Elaborado pelo autor.

O gráfico que será gerado pelo Código 3.33 está mostrado na Figura 3.4. Nela, percebemos que há quatro pontos de interseção entre as duas curvas. Assim, concluímos que a alternativa B é a correta.

Figura 3.4: Representação gráfica das funções trigonométricas.



Fonte: Elaborado pelo autor.

□

Capítulo 4

Sequências didáticas integradoras

Neste capítulo, propõe-se sequências didáticas que contemplem objetos de conhecimento de cada uma das cinco unidades temáticas da BNCC, com foco na recomposição e consolidação de competências e habilidades que integram a matriz de referência do SPAECE para o 9º ano do ensino fundamental, dando ênfase especial naqueles que figuram entre os que apresentam maiores índices de erro por parte dos estudantes.

De modo geral, a metodologia adotada fundamenta-se na resolução de problemas, por meio da apresentação de uma sequência de situações-problema contextualizadas. Os estudantes, organizados em grupos, discutirão estratégias de resolução e, com o apoio do professor, decidirão quais funcionalidades da linguagem de programação utilizar para resolver cada situação proposta. Mais especificamente, as sequências didáticas podem ser organizadas em três abordagens distintas:

Calculadoras ou conversores. Abordagem voltada à automação de procedimentos matemáticos, permitindo a exploração de conceitos por meio de simulações interativas e interpretações dinâmicas de dados e fórmulas.

Jogos. Estratégia centrada no engajamento dos estudantes, favorecendo a aprendizagem por meio da resolução de desafios, da interação lúdica, da experimentação criativa e do fornecimento de *feedback* automático.

Resolução de problemas. Resolução de situações-problema por meio da aplicação de comandos e bibliotecas da linguagem de programação Julia, promovendo o raciocínio lógico, a experimentação computacional e a iniciação à modelagem matemática.

As sequências propostas, acompanhadas de suas soluções, foram organizadas em um conjunto de cadernos digitais Jupyter disponíveis em:

`https://onmat.github.io/Matematica.jl/`

Ao longo de cada sequência, os problemas estão organizados por nível de complexidade, seguindo o modelo de avaliações externas. Essa abordagem busca promover o desenvolvimento gradual das habilidades previstas nos descritores, ao mesmo tempo em que valoriza a experimentação, o trabalho colaborativo e o uso consciente da tecnologia no processo de aprendizagem.

Embora não haja uma ordem cronológica obrigatória para a realização das sequências a seguir, recomenda-se iniciar com um momento para introduzir a linguagem Julia e ambientar os alunos aos cadernos digitais Jupyter (Seção 4.1), e seguir para a sequência sobre números (Seção 4.3). Daí em diante, o docente pode selecionar a sequência desejada em ordem arbitrária.

4.1 Familiarização com a Julia e o Jupyter

Antes de iniciar efetivamente as sequências didáticas, é necessário apresentar aos estudantes as metodologias e os recursos que serão utilizados ao longo das aulas. Essa etapa inicial tem como objetivo familiarizá-los com o ambiente de programação (neste caso, o Jupyter Notebook) e com a linguagem Julia. Além disso, serão discutidas as diferentes formas de interação com os cadernos digitais, promovendo um uso consciente e funcional das ferramentas computacionais.

Conteúdos abordados

- Noções básicas sobre o *Jupyter Notebook*:
  - Interface do ambiente: atalhos principais.
  - Tipos de células: texto (*Markdown*) e código;
  - Organização e execução das células.
- Fundamentos da linguagem Julia:
  - Operadores básicos: adição, subtração, multiplicação, divisão, potência e módulo;
  - Tipos, declaração e manipulação de variáveis;
  - Comandos de entrada e saída: `println`, `parse` e `readline`;
  - Controle de fluxo: `if`, `else`, `elseif`;
  - Estrutura de repetição: `for`.

Carga horária

Quatro aulas de 50 minutos.

Desenvolvimento

Durante esta aula, será adotada uma abordagem expositiva e demonstrativa, com intervenções práticas que permitam aos alunos experimentarem diretamente os conceitos apresentados. A familiarização com o ambiente Jupyter Notebook é essencial para o desenvolvimento autônomo dos desafios computacionais que serão propostos nas etapas seguintes.

Neste primeiro momento, é recomendável limitar-se às operações básicas e à declaração de variáveis, de modo a não sobrecarregar os estudantes com estruturas mais complexas. Tópicos como estruturas condicionais e comandos de repetição serão aprofundados posteriormente, à medida que surgirem demandas específicas.

Para cumprir com este momento, recomenda-se que os alunos realizem os três problemas a seguir.

Exercício 4.1.1. Desenvolva um programa que solicite ao usuário um número inteiro. O programa deve calcular e exibir o antecessor e o sucessor desse número de forma clara e organizada. *Dica:* Utilize como referência o Código 3.10.

Exercício 4.1.2. Crie uma tabuada de multiplicação interativa. O programa deve solicitar um número inteiro ao usuário e imprimir a tabuada completa desse número como saída. *Dica:* utilize como referência o Código 3.9.

Exercício 4.1.3. Refaça o Exercício 4.1.2 usando o comando `for`. *Dica:* utilize como referência o Código 3.28.

4.2 Lidando com Expressões Algébricas

Esta sequência tem por objetivo introduzir o conceito de valor numérico de uma expressão algébrica, explorando situações em que o aluno precisa substituir variáveis por valores reais e realizar os cálculos corretamente. Ela será utilizada ainda para analisar padrões numéricos, permitindo que os alunos identifiquem regularidades e formulem generalizações com base na substituição e observação dos resultados. Com isso, busca-se tanto desenvolver a habilidade de compreender a linguagem algébrica, quanto familiarizá-lo com a manipulação de variáveis em um ambiente computacional.

Descritores relacionados

- D30: Calcular o valor numérico de uma expressão algébrica;
- D32: Identificar a expressão algébrica que expressa uma regularidade observada em sequências de números ou figuras (padrões).

Carga horária

Quatro aulas de 50 minutos.

Desenvolvimento

Foram selecionados cinco exercícios de múltipla escolha provenientes de concursos e vestibulares, relacionados aos objetos de conhecimento que visam desenvolver as habilidades associadas aos descritores D30 e D32. Nessas atividades, o estudante utilizará a linguagem de programação para resolver as questões, promovendo a integração entre o raciocínio matemático e a prática computacional.

Esses exercícios são apresentados aos estudantes em células de texto Markdown. Em seguida, os alunos inserem células de código nas quais atribuem valores às variáveis e reescrevem a expressão algébrica. Ao executar o código, obtêm o resultado da expressão avaliada. Essa dinâmica facilita a aplicação das atividades e a conferência dos resultados pelos próprios estudantes. O formato interativo do caderno permite que os alunos experimentem diferentes valores para as variáveis e observem imediatamente o valor numérico das expressões, favorecendo a aprendizagem ativa e a fixação dos conceitos matemáticos.

Exercício 4.2.1 (Projeto pro(seguir))**.** O valor numérico de $2x + y$ para $x = 1$ e $y = 2$ é igual a:

A) 3 B) 4 C) 5 D) 23

Este exercício pode ser resolvido rapidamente com os comandos:

```
x = 1
y = 2
println("A solução para 2x + y é: ", 2x + y)
```

cujo resultado da execução será:

```
A solução para 2x + y é: 4
```

Exercício 4.2.2 (SAEP 2013)**.** Na escola de Marcos Antônio são realizadas duas avaliações por bimestre, sendo que a primeira tem peso 1 e a segunda tem peso 2, usando a seguinte fórmula:

$$M = \frac{A_1 + 2A_2}{3}$$

onde A_1 é o valor da primeira avaliação e A_2 é o valor da segunda avaliação. Se Marcos Antônio tirou 5,6 na primeira avaliação e 9,2 na segunda avaliação, a sua média do bimestre em questão foi:

A) 8,0 B) 7,4 C) 6,8 D) 5,0

Exercício 4.2.3 (Corpo de Bombeiros – RJ)**.** Quais os valores numéricos da expressão $2x^2 - 16x + 17$ para $x = 2$ e para $x = -2$, respectivamente:

A) 5 e 40 B) -7 e 57 C) -11 e -7 D) -15 e -23

Código 4.1: Cálculo da expressão $2x^2 - 16x + 17$

```
1 x = 2
2 println("Quando x = $x, o valor de 2x² - 16x + 17 é: ", 2x^2 - 16x + 17)
```

Fonte: Elaborado pelo autor.

O resultado para a execução do Código 4.1 será:

```
Quando x = 2, o valor de 2x² - 16x + 17 é: -7
```

Para $x = -2$, substituímos esse valor na linha 1 do Código 4.1 e, após executar o código, com o valor atualizado de `x`, obtemos como saída:

```
Quando x = -2, o valor de 2x² - 16x + 17 é: 57
```

Exercício 4.2.4 (PAAE – 2013)**.** Uma das formas de se estimar a área aproximada (A) da superfície corporal de um adulto, em m², tem como base sua massa (M), em quilogramas, e sua altura (h), em metros. A fórmula utilizada para esse cálculo é:

$$A = \frac{1}{6}\sqrt{M \times h}$$

De acordo com essa fórmula, a área, em m², da superfície do corpo de um adulto com 1,80 m de altura e 80 kg de massa é próxima de:

A) 1,8 B) 2,0 C) 2,2 D) 2,4

Exercício 4.2.5 (Prova Brasil)**.** Dada a expressão:

$$x = \frac{-b + \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Sendo $a = 1$, $b = -7$ e $c = 10$, o valor numérico de x é:

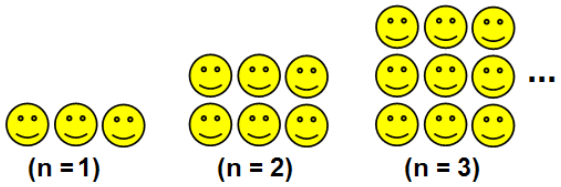
A) -5 B) -2 C) 2 D) 5

Exercício 4.2.6 (Supletivo 2014)**.** Na festa junina da escola, a professora de Artes resolveu fazer as bandeirinhas de São João conforme a sequência periódica de quatro cores. A cor da primeira bandeira é verde, a segunda é vermelha, a terceira é azul e a quarta é cinza. A partir daí, segue a sequência: verde, vermelho, azul, cinza, verde, vermelho, azul, cinza, e assim sucessivamente.

Qual é a cor da 55ª bandeira?

A) Verde B) Vermelho C) Azul D) Cinza

Exercício 4.2.7 (Supletivo 2010)**.** Observe a quantidade de figuras em cada coluna no quadro abaixo:



Mantendo esse mesmo padrão, a expressão algébrica que representa o número de figuras (F) na ordem n ($n = 1, 2, 3, \cdots$) é:

A) $F(n) = 3n+1$ B) $F(n) = 3n$ C) $F(n) = 2n+1$ D) $F(n) = 4n-1$

Exercício 4.2.8 (Prova Brasil)**.** As variáveis n e P assumem valores conforme mostra o quadro abaixo:

n	5	6	7	8	9	10
P	8	10	12	14	16	18

A relação entre P e n é dada pela expressão:

A) $P = n + 1$ B) $P = n + 2$ C) $P = 2n - 2$ D) $P = n - 2$

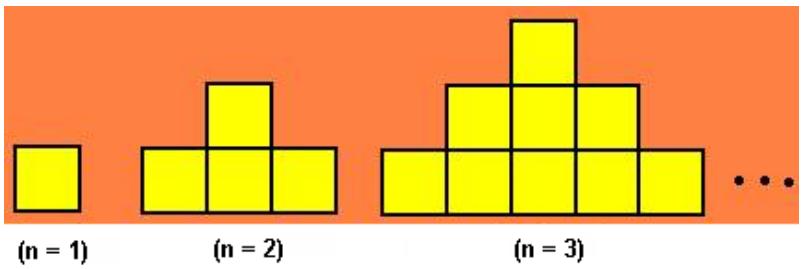
Exercício 4.2.9 (SAEPE)**.** Observe a sequência numérica abaixo:

n	1	2	3	4	5	$\cdots$
a_n	0	3	8	15	24	$\cdots$

A expressão algébrica que permite calcular o enésimo termo dessa sequência (a_n) em função da posição n é:

A) $2n - 1$ B) $2n + 2$ C) $n^2 - 1$ D) $n^2 + 2$

Exercício 4.2.10 (P. Wr)**.** As figuras mostradas abaixo estão organizadas dentro de um padrão que se repete:



Mantendo essa disposição, a expressão algébrica que representa o número de quadradinhos Q em função da ordem n ($n = 1, 2, 3, \dots$) é:

A) $Q = n$ B) $Q = n^2$ C) $Q = n^2 + 1$ D) $Q = n^2 + 2$

4.3 Calculadora de Frações

Para explorar as propriedades e operações numéricas, optou-se por trabalhar as quatro operações aritméticas com números racionais por intermédio de uma calculadora. Essa abordagem permite aos alunos manipular diferentes representações e operar com frações de maneira dinâmica e visual, favorecendo a construção de significados matemáticos.

Descritores relacionados

- D22: Identificar fração como representação que pode estar associada a diferentes significados;
- D23: Identificar frações equivalentes;
- D25: Efetuar cálculos que envolvam operações com números racionais (adição, subtração, multiplicação, divisão, potenciação).

Carga horária

Dez aulas de 50 minutos.

Metodologia

A construção da calculadora será dividida em cinco etapas, organizadas sob a forma de exercícios. Cada etapa proporá uma situação a ser resolvida pelos estudantes, estimulando o raciocínio lógico e a aplicação dos conhecimentos matemáticos e computacionais. Ao final de cada exercício, os alunos deverão implementar uma parte do código, de forma cumulativa, construindo progressivamente a calculadora completa.

Antes de iniciar o desenvolvimento do código, será necessário aplicar um diagnóstico com os estudantes. Essa etapa é essencial para identificar possíveis lacunas conceituais e adaptar a abordagem pedagógica às necessidades da turma, garantindo uma transição gradual entre os conceitos matemáticos e sua representação computacional.

Dessa forma, propõe-se, no exercício a seguir, a primeira etapa:

Exercício 4.3.1. Solicite aos estudantes que, de forma colaborativa, listem os objetos de conhecimento matemático envolvidos na construção da calculadora, assim

como os comandos e estruturas básicas da linguagem de programação necessários para o desenvolvimento do código.

O Exercício 4.3.1 favorecerá a apropriação dos conteúdos e estimulará o protagonismo dos alunos no processo de aprendizagem.

Exercício 4.3.2. Desenvolva um código que exiba o resultado das operações de adição, subtração, multiplicação e divisão, para as frações $\frac{2}{3}$ e $\frac{1}{2}$.

No Exercício 4.3.2, o código deve ser construído para trabalhar com duas frações definidas por seus numeradores e denominadores, armazenados em (quatro) variáveis isoladas. A nomeação clara e sistemática dessas variáveis facilitará sua manipulação e entendimento ao longo do desenvolvimento do código. Após a realização de cada operação, os estudantes deverão utilizar o comando `println` para exibi-los no console. Para as operações de adição e subtração, o cálculo do MMC entre os denominadores deverá ser realizado utilizando a função `gcd`. Ao final, espera-se uma implementação semelhante ao Código 4.2.

Código 4.2: Uma calculadora de frações.

```
# Frações fixas para teste:
num_1 = 2
den_1 = 3
num_2 = 1
den_2 = 2

# Produto
prod_num = num_1 * num_2
prod_den = den_1 * den_2
println("Produto: $(prod_num)/$(prod_den)")

# Quociente
div_num = num_1 * den_2
div_den = den_1 * num_2
println("Quociente: $(div_num)/$(div_den)")

# Soma
m = lcm(den_1, den_2)
soma_num = m ÷ den_1 * num_1 + m ÷ den_2 * num_2
println("Soma: $(soma_num)/$(m)")

# Diferença
sub_num = m ÷ den_1 * num_1 - m ÷ den_2 * num_2
println("Diferença: $(sub_num)/$(m)")
```

Fonte: Elaborado pelo autor.

O docente deve instruir os alunos a modificarem os valores das variáveis `num_X` e `den_X` para testar o funcionamento do código. Além disso, incentivá-los a desafiar as demais equipes, propondo casos diferentes e comparando os resultados obtidos.

Com essa atividade, conclui-se a segunda etapa da construção da calculadora, na qual os estudantes consolidam os conceitos matemáticos fundamentais e iniciam a familiarização com os comandos básicos da linguagem Julia.

Exercício 4.3.3. Implemente o Código 4.2 para que ele simplifique os resultados obtidos nas operações com frações.

Para simplificar os resultados obtidos em cada uma das operações, utilizaremos o cálculo do MDC entre o numerador e o denominador, armazenando esse valor em uma variável. A divisão de ambos os termos pelo MDC resultará na fração irredutível. Isto pode ser ilustrado pelo seguinte trecho de código:

```
mdc = gcd(num, den)
println("Resultado: $(num ÷ mdc)/$(den ÷ mdc)")
```

A utilização do operador ÷ (divisão inteira) garante que o resultado da simplificação seja apresentado em forma de número inteiro, evitando a exibição de valores decimais. Esse procedimento deve ser repetido para todas as operações (multiplicação, divisão, soma e subtração), assegurando que todas as frações geradas pelo algoritmo sejam apresentadas em sua forma irredutível.

Exercício 4.3.4. Encapsular o código do Exercício 4.3.3 em uma função reutilizável, denominada `calculadora`. A função deve exibir ao usuário as frações fornecidas como entrada antes da realização das operações.

Para encapsular o código da calculadora, insere-se na primeira linha a instrução:

```
function calculadora(num1, den1, num2, den2)
```

e, ao final, o comando `end`. Entre essas duas instruções, deve-se realocar o bloco de comandos previamente construído. Ressalta-se a importância de manter a indentação adequada dentro da função, a fim de garantir a legibilidade, a organização do código e seu correto funcionamento.

Exercício 4.3.5. Atualize o código do 4.3.4 para que ele realize apenas uma operação, previamente escolhida pelo usuário. Caso a operação informada seja inválida, o programa deve apresentar uma mensagem de erro.

Para implementar essa funcionalidade, o aluno deve usar a estrutura de decisão `if`, de modo que o fluxo de execução do programa seja direcionado conforme a operação escolhida pelo usuário. Além disso, será necessário acrescentar uma nova variável, do tipo `String`, destinada a armazenar a informação fornecida pelo usuário quanto à operação desejada. Com isso, o programa executará apenas a operação indicada e, caso a entrada seja inválida, apresentará uma mensagem de erro apropriada.

Para consolidar os conceitos abordados, os estudantes devem ser convidados a resolver o seguinte exercício.

Exercício 4.3.6 (IV/UFG – 2025)**.** Se $a = \frac{1}{2}$ e $b = \frac{1}{4}$, então $a^2 + b^2 + a + b$ vale:

A) $\frac{9}{16}$ B) $\frac{13}{16}$ C) $\frac{17}{16}$ D) $\frac{25}{16}$ E) $\frac{49}{16}$

4.4 Desafios do Triângulo

Considere três pontos de coordenadas inteiras geradas aleatoriamente no plano cartesiano. A depender dos valores das coordenadas, tais pontos podem ou não formar

um triângulo. Quando a figura formada for, de fato, um triângulo, os estudantes deverão enfrentar oito desafios sequenciais, que os conduzirão à investigação das propriedades geométricas envolvidas. Serão abordados temas como: verificação da existência do triângulo, análise das coordenadas dos vértices, medidas dos lados e dos ângulos, classificação quanto aos lados e ângulos, além do cálculo de perímetro e área. Essa sequência visa promover o raciocínio lógico e a integração de conceitos algébricos e geométricos, a partir de uma situação exploratória, investigativa e contextualizada.

Descritores relacionados

- D01: Identificar a localização ou movimentação de objetos em mapas, croquis e outras representações gráficas.
- D03: Identificar propriedades de triângulos por meio da comparação de medidas de lados e ângulos.
- D09: Interpretar informações apresentadas por meio de coordenadas cartesianas.
- D10: Utilizar relações métricas do triângulo retângulo para resolver problemas significativos.
- D12: Resolver problemas envolvendo o cálculo de perímetro de figuras planas.
- D13: Resolver problemas envolvendo o cálculo de área de figuras planas.
- D27: Efetuar cálculos simples com valores aproximados de radicais.

Carga horária

Quatro aulas de 50 minutos.

Desenvolvimento

Ao longo do caderno, serão exibidas algumas definições e conceitos teóricos com o intuito de oferecer suporte aos estudantes durante a realização das atividades. Os alunos interagem diretamente com as células de código do caderno de questões, respondendo às perguntas propostas e recebendo *feedback* automático. Caso não obtenham sucesso em determinada etapa, poderão refazer o processo quantas vezes forem necessárias, o que favorece a consolidação gradual dos conhecimentos e o desenvolvimento da autonomia na aprendizagem. Essa abordagem promove um ambiente de experimentação guiada, no qual o erro é compreendido como parte natural do processo investigativo. Além disso, o uso da programação permite que os estudantes testem hipóteses, verifiquem resultados e visualizem propriedades geométricas com o auxílio de representações gráficas.

Ao iniciar esta sequência didática, devemos executar um conjunto de códigos que precedem os desafios, descritos a seguir. Após concluir todos os desafios, o usuário pode realizar um novo sorteio de pontos e repetir o processo, jogando quantas vezes forem necessárias até consolidar os objetos de conhecimento propostos.

O Código 4.3 é responsável por gerar aleatoriamente três pontos e calcular as distâncias entre eles, utilizando a fórmula da distância entre dois pontos no plano cartesiano. Além disso, calculará o perímetro e a área da figura eventualmente formada.

Código 4.3: Geração dos pontos, cálculo das distâncias entre os vértices, do perímetro e da área da figura

```
1  using Plots
2  gr(size=(500, 500))  # Define o backend gráfico e o tamanho da janela
3  # Declarar fora da função para facilitar o uso depois
4  global x, y, AB, AC, BC, P, S
5  function gerar_figura()
6      global x, y, AB, AC, BC, P, S  # Necessário para modificar as variáveis globais
7      # Geração dos pontos A, B e C
8      a = rand(0:10)
9      b = rand(0:10)
10     x = [a, rand(0:10), rand(0:10), a]  # x[1]=Ax, x[2]=Bx, x[3]=Cx, x[4]=Ax
11     y = [b, rand(0:10), rand(0:10), b]  # y[1]=Ay, y[2]=By, y[3]=Cy, y[4]=Ay
12     # Cálculo das distâncias entre os pontos
13     AB = hypot(x[1] - x[2], y[1] - y[2])
14     AC = hypot(x[1] - x[3], y[1] - y[3])
15     BC = hypot(x[2] - x[3], y[2] - y[3])
16     # Perímetro
17     P = round(AB + AC + BC, digits = 1)
18     # Área (fórmula do determinante)
19     S = abs((x[1]*y[2] + x[2]*y[3] + x[3]*y[1] -
20              y[1]*x[2] - y[2]*x[3] - y[3]*x[1])) / 2
21     return nothing
22 end
```

Fonte: Elaborado pelo autor.

Na linha 4, as variáveis são declaradas como globais para que possam ser utilizadas e modificadas posteriormente em outros trechos do caderno.

Nas linhas 8 a 11, com a função `rand`, são sorteadas coordenadas inteiras para os vértices da figura, e os valores são armazenados nos vetores `x` e `y`.

O cálculo das distâncias entre os pontos A, B e C, nas linhas 13, 14 e 15 do código, é realizado por meio da função `hypot`, que determina a hipotenusa de um

triângulo retângulo. A partir dessas distâncias, o código calcula o perímetro do triângulo, somando as três distâncias, conforme a expressão na linha 17. Esse valor é armazenado na variável `P`, com arredondamento para uma casa decimal.

A área do triângulo formada pelos três pontos é calculada utilizando uma fórmula da geometria analítica baseada em determinantes, na linha 19. O valor é calculado utilizando uma fórmula que corresponde à metade do valor absoluto do determinante de ordem 3 formado pelas coordenadas dos três pontos. Caso o resultado seja igual a zero, os pontos são colineares e, portanto, não formam um triângulo válido.

Por fim, o comando `return nothing` garante que a função não retorne nenhum valor válido, evitando que o resultado da última expressão (no caso, o cálculo da área) seja exibido quando a função é executada. Além disso, o trecho de código correspondente às linhas 6 a 21 está encapsulado na função `gerar_figura`, o que permite executar essa rotina sempre que necessário, apenas chamando-a pelo nome.

O Código 4.4 é responsável por exibir a figura gerada no plano cartesiano, com uma malha quadriculada, restringindo a visualização ao primeiro quadrante, com os eixos limitados ao intervalo de -1 a 11. Nele, das linhas 2 a 12, utiliza-se a função `plot` para traçar uma figura a partir dos vetores `x` e `y` gerados no Código 4.3. Essa função exibe o título da figura, define a largura dos segmentos, desativa a legenda, limita os eixos ao intervalo de $[-1, 11]$, gradua os eixos de 0 a 10, define a cor da área interna do triângulo, ativa a malha quadriculada e aplica um preenchimento com transparência.

Código 4.4: Função para exibir a figura no plano cartesiano.

```
1  function plotar_figura(x,y)
2      plot(x, y,
3          title = "Triângulo",
4          lw = 3,
5          label = false,
6          xlim = (-1, 11),
7          ylim = (-1, 11),
8          aspect_ratio = :equal,
9          grid = :on, gridalpha = 0.7, gridstyle = :dash,
10         fillrange = 0, fillalpha = 0.5, fillcolor = :yellow,
11         xticks = 0:10,
12         yticks = 0:10)
13     scatter!(x[1:3], y[1:3],
14         label = false,
15         annotations = (x[1:3] .+ 0.3, y[1:3] .+ 0.3, ["A", "B", "C"]))
16 end
```

Fonte: Elaborado pelo autor.

A função `scatter!`, entre as linhas 13 e 15, é empregada para exibir os rótulos dos vértices A, B e C, posicionando as anotações ligeiramente deslocadas em relação aos pontos, de modo a facilitar a leitura do gráfico. Por fim, todo o código é encapsulado na função `plotar_figura(x,y)`, permitindo gerar o gráfico do triângulo sempre que necessário, apenas chamando essa função com os vetores de coordenadas.

A partir deste ponto, criaremos mais oito códigos, cada um correspondente a um dos desafios propostos. Esses códigos serão encapsulados por meio do comando `function`, pois serão reutilizados ao longo dos desafios em diferentes partes do caderno.

Cada uma das funções apresentadas tem como finalidade analisar as respostas fornecidas pelos estudantes, fornecendo feedback imediato e promovendo o desenvolvimento da autonomia no processo de aprendizagem.

A partir deste ponto, organizaremos a sequência de desafios. Ainda na mesma página do Jupyter Notebook, certifique-se de que todos os códigos, desde o `gerar_figura()` (Código 4.3) até o último código `area(S)` (Código 4.12), estejam funcionando corretamente e obedecendo à ordem disposta. Insira células de texto para os títulos e instruções e, respectivamente, células de código para chamar as funções. A título de exemplo, iniciaremos a construção dos desafios.

Os demais códigos implementados são referentes a cada um dos oito desafios propostos.

Exercício 4.4.1. Verificar a existência do triângulo a partir dos vértices sorteados, analisando se satisfazem a desigualdade triangular.

Código 4.5: Função que verifica a existência do triângulo pela área.

```
1  function existência_triângulo(S)
2      println("A figura é um triângulo?:")
3      println("\u270D Digite (s/n) para informar sua conclusão:")
4      d1 = readline()
5      # Verifica a existência através da área:
6      if S>0
7          exist = "s"  # Sim, o triângulo existe
8      else
9          exist = "n"  # Não, o triângulo não existe
10     end
11     if exist == d1
12         println("\u2705 Você acertou!")
13     else
14         println("\u274C Você errou!")
15         println("\U0001F501 Retorne à célula anterior e gere novas coordenadas")
16     end
17 end
```

Fonte: Elaborado pelo autor.

A função `existência_triângulo(S)` recebe como parâmetro o valor da área. Nas linhas seguintes, o Código 4.5 interage com o usuário (linhas 2 a 4), perguntando se a figura visualizada representa um triângulo. A resposta do usuário é armazenada na variável `d1`, utilizando o comando `readline()`.

Em seguida, entre as linhas 6 e 10, o código utiliza o valor armazenado na variável `S` para verificar a existência do triângulo. Para isso, emprega-se a estrutura condicional `if`: se a área for maior que zero, conclui-se que os pontos não são colineares e, portanto, formam um triângulo, executando-se a linha 7. Caso contrário, se a área for igual a zero, significa que os pontos estão alinhados (colineares) e não determinam um triângulo válido, de modo que será executada a linha 9.

Por fim, entre as linhas 11 e 16, uma nova estrutura `if` é usada para comparar a resposta fornecida pelo usuário com o resultado obtido pelo código. De acordo com essa comparação, é exibida uma mensagem de acerto ou erro, indicando se o julgamento do usuário foi compatível com a análise feita a partir da área.

Exercício 4.4.2. Caso o triângulo exista, identificar as coordenadas de seus vértices.

Código 4.6: Função que compara as coordenadas informadas pelo estudante com os valores sorteados.

```
1  function verificar_vertices(x, y)
2      println("Informe as coordenadas x e y do ponto A")
3      x_1 = parse(Int, readline())
4      y_1 = parse(Int, readline())
5      println("Informe as coordenadas x e y do ponto B")
6      x_2 = parse(Int, readline())
7      y_2 = parse(Int, readline())
8      println("Informe as coordenadas x e y do ponto C")
9      x_3 = parse(Int, readline())
10     y_3 = parse(Int, readline())
11     println(repeat("*", 39))
12     A = [x[1], y[1]]
13     B = [x[2], y[2]]
14     C = [x[3], y[3]]
15     #Verificação das respostas
16     if A == [x_1, y_1]
17         println("\u2705 Ponto A!")
18     else
19         println("\u274C Ponto A!")
20         println("\U0001F501 Tente novamente!")
21     end
22     if B == [x_2, y_2]
23         println("\u2705 Ponto B!")
24     else
25         println("\u274C Ponto B!")
26         println("\U0001F501 Tente novamente!")
27     end
28     if C == [x_3, y_3]
29         println("\u2705 Ponto C!")
30     else
31         println("\u274C Ponto C!")
32         println("\U0001F501 Tente novamente!")
33     end
34 end
```

Fonte: Elaborado pelo autor.

O Código 4.6, nas linhas 2, 3 e 4, interage com o usuário solicitando que ele informe as coordenadas correspondentes ao vértice A. Esses valores são armazenados nas variáveis `x_1` e `y_1`, utilizando o comando `parse(Int, readline())`. O mesmo

processo é realizado para os vértices B e C, cujas coordenadas são atribuídas às variáveis `x_2`, `y_2`, `x_3` e `y_3`. O comando da linha 11 é opcional e serve apenas para exibir uma sequência de caracteres, com a finalidade de organizar visualmente a saída do código no console.

Na sequência, o bloco de código das linhas 12, 13 e 14 define as coordenadas dos pontos A, B e C. Já o bloco de código entre as linhas 16 e 21 realiza a verificação das coordenadas fornecidas pelo estudante, comparando o vetor `[x_1, y_1]` com o vetor `A = [x[1], y[1]]`, que representa o ponto A sorteado anteriormente. Esse mesmo processo é repetido para os pontos B e C, validando as entradas do usuário para todos os vértices e exibindo mensagens de acertos ou erros conforme o resultado da comparação.

Todo o código está encapsulado na função `verificar_vertices(x,y)`, permitindo sua reutilização ao longo da sequência didática.

Exercício 4.4.3. Calcular as medidas dos lados do triângulo, utilizando a fórmula da distância entre dois pontos, com precisão de uma casa decimal.

Código 4.7: Função `verificar_lados` que compara as medidas fornecidas com as medidas dos lados do triângulo.

```
1  function verificar_lados(AB,AC,BC)
2      println("\u270D Informe a medida do lado AB")
3      AB_1 = parse(Float64, readline())
4      println("\u270D Informe a medida do lado AC")
5      AC_1 = parse(Float64, readline())
6      println("\u270D Informe a medida do lado BC")
7      BC_1 = parse(Float64, readline())
8      if round(AB, digits=1) == AB_1
9          println("O lado AB: \u2705")
10     else
11         println("O lado AB: \u274C")
12         println("\U0001F501 Tente novamente!")
13     end
14     if round(AC, digits=1) == AC_1
15         println("O lado AC: \u2705")
16     else
17         println("O lado AC: \u274C")
18         println("\U0001F501 Tente novamente!")
19     end
20     if round(BC, digits=1) == BC_1
21         println("O lado BC: \u2705")
22     else
23         println("O lado BC: \u274C")
24         println("\U0001F501 Tente novamente!")
25     end
26 end
```

Fonte: Elaborado pelo autor.

A função `verificar_lados(AB,AC,BC)` recebe como parâmetros as medidas reais dos lados do triângulo, previamente calculadas no Código 4.3: `AB`, `AC` e `BC`.

Nas linhas iniciais, a função solicita ao usuário que informe os valores obtidos para cada um dos lados do triângulo. Para isso, utiliza o comando `readline()`, que lê o valor digitado, e o comando `parse(Float64, ...)`, que converte a entrada textual em um número do tipo ponto flutuante. Esse valor é então armazenado em uma variável correspondente a cada lado — por exemplo, AB_1 corresponde ao lado AB.

Na sequência, nas linhas 8 a 13, o código compara, por meio da estrutura condicional `if`, o valor informado e armazenado em `AB_1` com o valor arredondado

da medida de AB, com precisão de uma casa decimal. Caso a resposta esteja correta, é exibida uma mensagem de confirmação; caso contrário, é apresentada uma mensagem de erro, seguida da sugestão de uma nova tentativa.

Esse processo é repetido para os demais lados do triângulo, permitindo ao aluno verificar sua própria resolução e, em caso de erro, revisar os cálculos realizados.

Exercício 4.4.4. Classificar o triângulo quanto aos lados (equilátero, isósceles ou escaleno), com base nas medidas obtidas.

Código 4.8: Função para verificar a classificação do triângulo pelos lados

```
1  function classificacao_lados(AB, AC, BC)
2      println("\u270D Qual classificação do triângulo ABC?")
3      println("1\u20E3 Equilátero")
4      println("2\u20E3 Isósceles")
5      println("3\u20E3 Escaleno")
6      d2 = parse(Int, readline())
7      if AB == AC && AC == BC
8          class = 1   # Equilátero
9      elseif AB == AC || AB == BC || AC == BC
10         class = 2   # Isósceles
11     else
12         class = 3   # Escaleno
13     end
14     if class == d2
15         println("\u2705 Você acertou!")
16     else
17         println("\u274C Você errou!")
18         println("\u27F3 Tente novamente!")
19     end
20 end
```

Fonte: Elaborado pelo autor.

A função `classificacao_lados` tem como objetivo verificar se o estudante é capaz de identificar corretamente a classificação de um triângulo com base nas medidas de seus lados.

Nas linhas iniciais do Código 4.8, o programa interage com o usuário, solicitando que ele escolha uma das opções para classificar o triângulo: `1` para equilátero, `2` para isósceles e `3` para escaleno. A resposta é armazenada na variável `d2`.

Em seguida, o código utiliza estruturas condicionais `if-elseif-else` (linhas 7 a 13) para verificar a classificação correta do triângulo. Os critérios utilizados são:

se a condição da linha 7 for verdadeira, isso implica que os lados AB, AC e BC são congruentes, e assim será atribuído à variável `class` o número 1. Se ocorrer uma das igualdades na linha 9, a variável `class` receberá o valor 2. Caso nenhuma das condições anteriores seja satisfeita, será atribuído à variável `class` o número 3.

Por fim, nas linhas 14 a 19, o programa compara a resposta do usuário com a classificação correta armazenada na variável `class`. Se forem iguais, uma mensagem de acerto é exibida. Caso contrário, o código informa o erro e sugere uma nova tentativa.

Exercício 4.4.5. Determinar o terceiro ângulo interno do triângulo, a partir da informação de dois ângulos, considerando que a soma dos ângulos internos é 180°.

Código 4.9: Função `ângulo_interno` para verificar a medida do terceiro ângulo de um triângulo.

```
1  function ângulo_interno(AB,AC,BC,S)
2      α = round(asind((2 * S) / (AB * AC)))
3      β = round(asind((2 * S) / (AB * BC)))
4      println("As medidas de dois dos ângulos internos do triângulo ABC são:
5      $(α)° e $(β)°.")
6      println("Qual a medida do terceiro ângulo?")
7      γ = parse(Int, readline())
8      if γ == 180 - α - β
9          println("\u2705 Você acertou!")
10     else
11         println("\u274C Você errou!")
12         println("\U0001F501 Refaça seus cálculos e tente novamente")
13     end
14 end
```

Fonte: Elaborado pelo autor.

A função `ângulo_interno(AB,AC,BC,S)` tem como objetivo calcular dois dos ângulos internos de um triângulo com base na área e nas medidas de seus lados. Para isso, utiliza a fórmula:

$$\sin(\theta) = \frac{2A}{ab},$$

onde A representa a área do triângulo e a e b são lados adjacentes ao ângulo θ. A função `asind` retorna o arco seno em graus, permitindo o cálculo direto dos ângulos internos de um triângulo.

Nas linhas 2 e 3, os valores de α e β são calculados a partir do valor da área armazenado em `S` e das medidas dos lados armazenadas nas variáveis `AB`, `AC` e `BC`. Opta-se por não arredondar os valores utilizados nesse cálculo, a fim de evitar erros

no código, uma vez que o argumento da função `asind` deve estar limitado ao intervalo $[-1, 1]$.

O programa exibe ao usuário os dois ângulos calculados e solicita, na linha 5, que ele informe o valor do terceiro ângulo, γ. A verificação ocorre na linha 6, comparando o valor digitado com $180° - \alpha - \beta$, com base no fato de que a soma dos ângulos internos de qualquer triângulo é sempre 180°.

Se a resposta do usuário estiver correta, é exibida a mensagem "Você acertou!". Caso contrário, o programa informa o erro e sugere uma nova tentativa, exibindo "Refaça seus cálculos e tente novamente".

Exercício 4.4.6. Classificar o triângulo quanto aos ângulos (acutângulo, retângulo ou obtusângulo), com base nas relações métricas entre os lados.

Código 4.10: Função `classificacao_ângulos` que verifica a classificação do triângulo quanto aos ângulos internos.

```
1  function classificacao_ângulos(AB, AC, BC)
2      println("\u270D Qual classificação do triângulo ABC?")
3      println("1\u20E3 Acutângulo")
4      println("2\u20E3 Obtusângulo")
5      println("3\u20E3 Retângulo")
6      d3 = parse(Int, readline())  # Armazena a resposta do usuário
7
8      lados = Float64[AB, AC, BC]
9      l = sort(lados)  # Coloca em ordem crescente
10
11     if l[3]^2 < l[1]^2 + l[2]^2
12         class_a = 1  # Acutângulo
13     elseif l[3]^2 > l[1]^2 + l[2]^2
14         class_a = 2  # Obtusângulo
15     else
16         class_a = 3  # Retângulo
17     end
18     if class_a == d3
19         println("\u2705 Você acertou!")
20     else
21         println("\u274C Você errou!")
22         println("\U0001F501 Tente novamente!")
23     end
24 end
```

Fonte: Elaborado pelo autor.

A função `classificacao_ângulos` tem como objetivo classificar o triângulo quanto aos seus ângulos internos. Para isso, ela recebe como parâmetros as medidas dos lados do triângulo: `AB`, `AC` e `BC`.

Nas linhas 2 a 5 do Código 4.10, o programa apresenta ao usuário três opções de classificação, representadas pelos códigos numéricos: 1 (acutângulo), 2 (obtusângulo) e 3 (retângulo). A resposta do usuário é lida por meio da função `readline()` e convertida em número inteiro com `parse(Int, ...)`.

Em seguida, as medidas dos lados são armazenadas em um vetor `lados` (linha 8), que é ordenado em ordem crescente no vetor `l` com o comando `sort()` (linha 9). Dessa forma, a posição `l[3]` corresponde ao maior lado do triângulo, enquanto `l[1]` e `l[2]` representam os menores.

A classificação é realizada com base na relação entre os lados do triângulo, utilizando o Teorema de Pitágoras como critério:

- Se $l_3^2 < l_1^2 + l_2^2$, o triângulo é acutângulo.
- Se $l_3^2 > l_1^2 + l_2^2$, o triângulo é obtusângulo.
- Se $l_3^2 = l_1^2 + l_2^2$, o triângulo é retângulo.

Nas linhas 11 e 13, são verificados, por meio das estruturas `if` e `elseif`, o primeiro ou o segundo critério para a classificação do triângulo. Se a condição do `if` for verdadeira, a variável `class_a` receberá o número 1 ou 2, dependendo do caso. Caso nenhum dos critérios seja atendido, será executado o bloco `else`, e `class_a` receberá o número 3.

Por fim, o programa comparará a resposta do usuário, armazenada na variável `d3`, com o valor obtido pela verificação automática, na variável `class_a`. Caso coincidam, uma mensagem de acerto é exibida; caso contrário, é apresentada uma mensagem de erro, seguida de uma sugestão de nova tentativa.

Exercício 4.4.7. Calcular o perímetro do triângulo por meio da soma das medidas de seus lados.

Código 4.11: Função `perimetro` para verificação da resposta do perímetro do triângulo.

```
function perimetro(P)
    println("\u270D Informe a medida do perímetro do triângulo ABC:")
    d4 = parse(Float64, readline())

    if d4 == P
        println("\u2705 Sua resposta está correta!")
        println("O perímetro do triângulo ABC mede: $P unidades.")
    else
        println("\u274C Sua resposta está incorreta.")
        println("\u21BA Tente novamente!")
    end
end
```

Fonte: Elaborado pelo autor.

A função `perimetro(P)` recebe como parâmetro o valor do perímetro do triângulo, calculado previamente no Código 4.3.

Primeiramente, o código solicita ao usuário que informe a medida do perímetro do triângulo ABC, através da função `readline()` para capturar a resposta, que é convertida para o tipo `Float64` com o comando `parse` e armazenada em `d4`.

Em seguida, o valor fornecido pelo usuário é comparado com o valor armazenado na variável `P` através de uma estrutura condicional `if`. Se a resposta estiver correta, a função exibe uma mensagem de confirmação, destacando que o valor informado está correto. Caso contrário, apresenta uma mensagem de erro e incentiva o usuário a tentar novamente.

Exercício 4.4.8. Calcular a área do triângulo, com base na análise da figura ou por meio de fórmulas apropriadas.

Código 4.12: Função `area` para validar a resposta do usuário sobre a área do triângulo

```
function area(S)
    println("\u270D Informe a medida da área do triângulo ABC:")
    d5 = parse(Float64, readline()) #Recebe o valor da área
    if d5 == S
        println("\u2705 Sua resposta está correta!")
        println("A área do triângulo ABC mede: $S unidades².")
        println("\u2728 PARABÉNS! \u2728")
        println()
        println("Se você está lendo esta mensagem, significa que venceu todos
        os desafios!")
        println("Excelente trabalho, continue praticando e desafiando
        sua mente!")
        println("Até os próximos desafios!")
    else
        println("\u274C Sua resposta está incorreta.")
        println("\u21BB Tente novamente!")
    end
end
```

Fonte: Elaborado pelo autor.

A função `area(S)` recebe como parâmetro o valor da área do triângulo calculada previamente e solicita ao usuário que informe a medida da área do triângulo ABC.

A entrada fornecida pelo usuário é capturada, convertida e armazenada na variável `d5` com a função `readline()` e convertida para número decimal de precisão dupla por meio do comando `parse(Float64, ...)`. O valor informado é então comparado ao valor armazenado na variável `S` utilizando uma estrutura condicional `if`.

Caso o usuário acerte, a função exibe uma mensagem de confirmação com símbolos visuais, além de uma mensagem motivacional para estimular a continuidade do aprendizado.

Se a resposta estiver incorreta, uma mensagem de erro é exibida, juntamente com uma sugestão para revisar os cálculos, incentivando o usuário a tentar novamente.

A presença de símbolos Unicode em todos os códigos torna o feedback mais amigável e envolvente, promovendo uma experiência interativa e motivadora durante a resolução dos desafios.

4.5 Jogo das Três Medidas

Esta atividade está fundamentada em uma abordagem lúdica, por meio de um jogo, desenvolvido com o intuito de revisar os conceitos relacionados à massa e suas unidades de medida.

Descritores relacionados

- D15: Resolver problema utilizando relações entre diferentes unidades de medida.

Carga horária

Quatro aula de 50 minutos.

Desenvolvimento

Este jogo é indicado para ser utilizado como atividade de revisão, permitindo aos estudantes aplicar os conhecimentos adquiridos de forma prática, dinâmica e interativa, o que favorece a fixação dos conteúdos e o desenvolvimento do raciocínio matemático.

Antes da realização do jogo, deve ser realizada uma breve exposição teórica abordando:

- a definição de massa e sua distinção de peso;
- como a massa pode ser mensurada;
- qual a unidade padrão de medida no Sistema Internacional – o quilograma (kg) – e como realizar a conversão entre as unidades usuais, como miligrama (mg), grama (g), quilograma (kg) e tonelada (t).

Inicialmente, são sorteados aleatoriamente:

- Um valor numérico entre 0 e 1000;
- Uma unidade de medida de massa associada a esse valor (tonelada, quilograma ou grama);
- Uma unidade de destino para a qual esse valor deve ser convertido, também em tonelada, quilograma ou grama.

Isto é feito utilizando a função `rand`. O primeiro sorteio, `a = rand(0:1000)`, gera um número inteiro no intervalo de 0 a 1000, que representa o valor da massa, sendo armazenado na variável `a`. O segundo sorteio, `b = rand(["kg", "g", "T"])`, seleciona uma *String* da lista `[‘‘kg’’, ‘‘g’’, ‘‘T’’]`, correspondente à unidade de origem da massa, e o valor é armazenado na variável `b`. Por fim, o terceiro sorteio,

`c = rand(["quilogramas", "gramas", "toneladas"])`, define a unidade de destino da conversão, escolhida na lista [‘‘quilogramas’’, ‘‘gramas’’, ‘‘toneladas’’], sendo armazenada na variável `c`.

Em seguida, é solicitado ao usuário que informe o valor convertido corretamente. O jogador deve converter o valor sorteado de `b` para `c` e digitar sua resposta, que é armazenada na variável `resp`. A verificação da resposta é realizada por meio de estruturas condicionais do tipo `if`, combinadas com o conectivo lógico OU (||), para comparar a entrada do usuário com o valor esperado, levando em conta as diferentes combinações possíveis entre as unidades. O primeiro caso, linha 8, verificado corresponde à situação em que a unidade de massa sorteada é igual à unidade de destino sorteada. Nesse caso, o jogador não precisa realizar nenhuma conversão, apenas repetir o valor numérico informado. Já as próximas verificações são realizadas de acordo com a conversão que o jogador deve fazer. Por exemplo, o bloco de código correspondente às linhas 15 a 19 realiza a verificação de todos os casos em que o valor informado pelo código deve ser dividido por 1000. Isso ocorre quando a conversão exige a passagem de uma unidade menor para uma unidade maior, como de gramas para quilogramas ou de quilogramas para toneladas. E assim se segue para os demais casos, com verificações específicas de acordo com a relação entre as unidades sorteadas, utilizando operações de multiplicação ou divisão por potências de 10 conforme necessário.

Para cada caso, o sistema fornece um feedback automático: uma mensagem de acerto com o ícone unicode `\u2705` quando a conversão está correta, ou uma mensagem de erro `\u274C`, acompanhada da resposta correta. Esse retorno imediato permite que o jogador aprenda com o próprio erro, promovendo o reforço do conteúdo de forma interativa.

Código 4.13: Trecho principal do código do *Jogo das Três Medidas*.

```
# Sorteios
a = rand(0:1000)
b = rand(["kg", "g", "T"])
c = rand(["quilogramas", "gramas", "toneladas"])
println("Converta $a$b para $c") # Interação com o usuário
resp = parse(Float64, readline())
# Avaliação da resposta informada e feedback
if b == "kg" && c == "quilogramas" || b == "g" && c == "gramas" || ...
    if resp == a
        println("\u2705 VOCÊ ACERTOU!")
    else
        println("\u274C VOCÊ ERROU! A resposta correta é: $a")
    end
elseif b == "g" && c == "quilogramas" || b == "kg" && c == "toneladas"
    if resp == a / 1000
        println("\u2705 VOCÊ ACERTOU!")
    else
        println("\u274C VOCÊ ERROU! A resposta correta é: $(a / 1000)")
    end
elseif b == "T" && c == "quilogramas" || b == "kg" && c == "gramas"
    if resp == a * 1000
        println("\u2705 VOCÊ ACERTOU!")
    else
        println("\u274C VOCÊ ERROU! A resposta correta é: $(a * 1000)")
    end
elseif b == "g" && c == "toneladas"
    if resp == a / 1000000
        println("\u2705 VOCÊ ACERTOU!")
    else
        println("\u274C VOCÊ ERROU! A resposta correta é: $(a / 1000000)")
    end
elseif b == "T" && c == "gramas"
    if resp == a * 1000000
        println("\u2705 VOCÊ ACERTOU!")
    else
        println("\u274C VOCÊ ERROU! A resposta correta é: $(a * 1000000)")
    end
end
```

Fonte: Elaborado pelo autor.

Essa lógica torna o jogo um recurso valioso para fixar conteúdos sobre unidades de medida e suas conversões. Além disso, esse tipo de atividade permite ao estudante exercitar a lógica da conversão entre diferentes ordens de grandeza no sistema métrico decimal, promovendo tanto o desenvolvimento do raciocínio matemático quanto a familiarização com tecnologias digitais como ferramentas de aprendizagem.

Este jogo pode ser adaptado para trabalhar outras grandezas, como unidades de comprimento, tempo ou capacidade. Essa adaptação pode ser proposta como um desafio criativo aos próprios estudantes, que, sob a orientação do professor, poderão modificar o código original, ampliando sua compreensão tanto dos conteúdos matemáticos quanto dos princípios básicos da programação. Neste sentido, são propostos os seguintes exercícios.

Exercício 4.5.1. Construam um código que:

- exiba o título "Conversora de Unidade de Massa";
- solicite um valor numérico e a respectiva unidade de medida de massa (tonelada, quilograma ou grama);
- converta esse valor e exiba o resultado correspondente nas três unidades de medida de massa: tonelada, quilograma e grama.

Exercício 4.5.2. (IDEPB) Fernanda usou dois quilogramas de biscoito em uma receita de doce. Quantos gramas desse biscoito ela usou nessa receita?
A) 2 B) 100 C) 1.000 D) 2.000

Exercício 4.5.3. (AREAL) Observe no caminhão abaixo quantas toneladas de mercadorias ele pode transportar por viagem.

No máximo, quantos quilogramas de mercadorias esse caminhão pode transportar em uma viagem?
A) 38 kg B) 380 kg C) 3.800 kg D) 38.000 kg

Exercício 4.5.4. (SAEMI) Mário comprou 1.800 kg de cal, 2.300 kg de cimento, 2.000 kg de rejunte e 200 kg de argamassa para serem usados em uma obra. Quantas toneladas de material, ao todo, Mário comprou para essa obra?
A) 5,3 B) 6,3 C) 530 D) 630

4.6 Dois M's da Estatística

A construção da calculadora denominada *Calculadora de Tendências Centrais* tem por finalidade trabalhar os conceitos e a aplicação das medidas de tendência central — média e mediana.

Descritores relacionados

- D77: Resolver problema utilizando a média aritmética.

Carga horária

Quatro aula de 50 minutos

Desenvolvimento

Embora o descritor D77 enfatize exclusivamente a média, aproveitaremos essa oportunidade para ampliar o repertório dos estudantes, apresentando também o conceito de mediana a partir de uma sequência numérica. Essa abordagem integrada permite uma compreensão mais ampla das medidas de tendência central e favorece o desenvolvimento do pensamento estatístico. Neste sentido, propomos o seguinte exercício:

Exercício 4.6.1. Desenvolva um código que receba um conjunto de n números reais e exiba:

- os números organizados em ordem crescente;
- a soma dos valores;
- a quantidade de números informados;
- a média aritmética;
- a mediana.

Para desenvolvermos um código que atenda a todos os itens exigidos neste exercício, recomendamos proceder em duas etapas. Primeiramente, escreveremos um código que trate apenas do cálculo da média. Quando essa etapa estiver funcionando corretamente, implementaremos o cálculo da mediana.

O algoritmo deve receber a quantidade de elementos da amostra e, em seguida, solicitar cada um dos valores, armazenando-os em um vetor. A partir dessas informações, o código exibirá os elementos em ordem crescente, a soma total, a quantidade de números informados e, por fim, a média aritmética. Isto está implementado no Código 4.14.

Código 4.14: Cálculo da média de uma sequência de números.

```
1  println(repeat("*", 38))
2  println("* Calculadora de Tendências Centrais *")
3  println(repeat("*", 38))
4
5  # Coleta e armazena os dados no vetor
6  println("Informe a quantidade de números:")
7  n = parse(Int, readline())
8  x = Vector{Float64}(undef, n)  # Cria um vetor com n posições não inicializadas
9
10 for i in 1:n
11     println("Informe o $(i)º número:")
12     x[i] = parse(Float64, readline())
13 end
14
15 # Calcula e exibe os resultados
16 total = sum(x)
17 media = round(total / n, digits=2)
18
19 println("\n--- Resultados ---")
20 println("Quantidade de números informados: ", length(x))
21 println("Soma dos números da sequência: ", total)
22 println("Média da sequência: ", media)
```

Fonte: Elaborado pelo autor.

Na linha 7 do Código4.14, a quantidade de elementos informada pelo usuário é armazenada na variável `n`. Em seguida, na linha 8, declara-se o vetor `x` com `n` posições do tipo `Float64`, inicialmente não inicializadas.

No bloco de código compreendido entre as linhas 10 e 13, o comando `for` realiza `n` iterações, solicitando e armazenando cada valor informado pelo usuário na posição correspondente do vetor `x[i]`, onde `i` indica a posição do elemento no vetor `x` e $i \in [1, n]$.

Na linha 16, a função `sort(x)` ordena os elementos do vetor em ordem crescente. A linha 17 utiliza a função `sum(x)` para calcular a soma de todos os elementos da sequência. Já na linha 18, calcula-se a média aritmética, arredondada para duas casas decimais, por meio da função `round`.

Por fim, nas linhas de 20 a 24, são exibidos os resultados da análise: os números ordenados, a quantidade de elementos, a soma total e a média da sequência.

Para a segunda parte deste exercício, basta acrescentar o seguinte bloco de código:

```
# Ordena o vetor
x_ordenado = sort(x)

# Cálculo da mediana
if n % 2 == 1
    # Para n ímpar: mediana é o elemento do meio
    mediana = x_ordenado[(n + 1) ÷ 2]
else
    # Para n par: mediana é a média dos dois elementos centrais
    mediana = (x_ordenado[n ÷ 2] + x_ordenado[(n ÷ 2) + 1]) / 2
end
```

A ideia implementada é a seguinte:

- Se o número de elementos for par, o código calculará a média aritmética entre os dois elementos centrais do vetor ordenado, `x_ordenado`, ou seja, os elementos das posições $\frac{n}{2}$ e $\frac{n}{2} + 1$.
- Caso seja ímpar, o código exibirá diretamente o termo central desse vetor, localizado na posição $\frac{n+1}{2}$, que corresponde ao índice inteiro do elemento central na sequência.

Capítulo 5

Considerações Finais

É inegável a existência de inúmeros desafios que permeiam a educação atualmente, sobretudo no ensino da matemática. Tais dificuldades não são recentes, estendendo-se por muitos anos. Além disso, é evidente que não podem ser superadas por soluções rápidas ou simplistas. Trata-se de um trabalho árduo, complexo e contínuo, que depende tanto de fatores internos quanto externos ao ambiente escolar, tais como a formação dos docentes, a infraestrutura escolar, a dedicação dos alunos, a participação efetiva das famílias no cotidiano escolar, bem como a gestão dos investimentos destinados à área da educação. Sem dúvida, trata-se de um desafio de grande magnitude, sempre evidenciado pelos resultados nos sistemas de avaliação externos vigentes.

Constantemente, surgem novas práticas e propostas por parte de educadores dispostos a transformar esse cenário e reduzir a gravidade da situação. Nesse contexto, a computação apresenta-se como um recurso didático promissor e positivo, amplamente defendido por diversos estudiosos, sobretudo por seu potencial de contribuir não apenas para o aprendizado matemático, mas também para a formação social e moral dos alunos.

O uso da programação integrada ao ensino da matemática configura-se como uma alternativa promissora, pois enriquece as aulas e desperta o interesse pela aprendizagem. Ao trabalhar em sala de aula de forma articulada com a tecnologia, os alunos tendem a se mostrar mais motivados, buscando compreender, resolver desafios e superar obstáculos. Nesse processo, enquanto pensam e interagem com as atividades propostas, desenvolvem de forma contínua o raciocínio lógico e intuitivo, além de perceberem a aplicabilidade e a importância da matemática em seu cotidiano. Essa perspectiva converge com a BNCC no que se refere ao letramento matemático.

Ao longo deste trabalho, foi possível constatar o elevado potencial que o Jupyter Notebook, associado à linguagem de programação Julia, possui para a construção de sequências didáticas. Essa combinação permitiu a criação de materiais intera-

tivos, muitas vezes com apelo visual, pedagogicamente relevantes, favorecendo a compreensão dos conteúdos matemáticos de forma contextualizada e dinâmica.

Por fim, espera-se que este trabalho contribua de forma significativa para o fortalecimento do pensamento computacional no ensino de matemática, incentivando professores a incorporarem linguagens de programação em suas práticas pedagógicas; tendo em vista que ensinar, enquanto processo de mediação do conhecimento, é uma tarefa desafiadora, na qual o professor também se coloca como aprendiz. Do mesmo modo, almeja-se que essa abordagem motive os estudantes a se engajarem ativamente no processo de aprendizagem, desenvolvendo as competências esperadas e fortalecendo sua autonomia intelectual.

Referências

ACADEMY, Asimov. **Guia HTML**. 2024. Disponível em: <https://hub.asimov.academy/blog/guia-html/>. Acesso em: 19 dez. 2024.

AMS. **User's Guide for the amsmath Package**. [S.l.], 2020. Version 2.1. Disponível em: <https://ctan.org/pkg/amsmath>.

ANGARA, Prashanti Priya et al. Teaching quantum computing to high-school-aged youth: A hands-on approach. **IEEE Transactions on Quantum Engineering**, v. 3, p. 1–15, 2021.

AVILA, Damian. **RISE**. 2025. Disponível em: <https://github.com/damianavila/RISE>.

AZEVEDO, Greiton Toledo de; MALTEMPI, Marcus Vinicius. Processo de Aprendizagem de Matemática à luz das Metodologias Ativas e do Pensamento Computacional. **Ciência & Educação (Bauru)**, v. 26, 2020. DOI: 10.1590/1516-731320200061.

BIEHLER, Rolf et al. Data science education in secondary schools: Teaching and learning decision trees with CODAP and Jupyter Notebooks as an example of integrating machine learning into statistics education. In: PROCEEDINGS of the IASE 2020 Roundtable. [S.l.: s.n.], 2020.

CASARIN, Bruno Martins Siqueira. **Interfaces gráficas para experimentos de interferência e difração via Python no ensino da óptica ondulatória**. 2022. Dissertação de Mestrado – Universidade Federal do Tocantins.

CASTRUCCI, Benedicto; JÚNIOR, José Ruy Giovanni. **A Conquista da Matematica - 9º ano**. 4. ed. São Paulo: FTD, 2018.

CISNEROS GUEVARA, Marco. **Formación del profesorado dentro del Marco del modelo Conocimiento Tecnológico Pedagógico del Contenido (TPACK) como innovación en la Enseñanza de las cónicas nivel medio superior preparatoria UNIVA**. 2024. Dissertação de Mestrado – Universidad de Guadalajara.

COBO, A et al. IMPLEMENTATION OF VIRTUAL LABORATORIES WITH JUPYTER NOTEBOOKS: PRACTICAL EXPERIENCES IN COMPUTER SCIENCE AND MATHEMATICS. In: EDULEARN22 Proceedings. [S.l.: s.n.], 2022. P. 8746–8753.

DASDAN, Ali. Eleven Simple Algorithms to Compute Fibonacci Numbers. **CoRR**, abs/1803.07199, 2018. Disponível em: <http://arxiv.org/abs/1803.07199>.

EDUCAÇÃO, CEARÁ. Secretaria da. **Boletim de Resultados da Escola – SPAECE 2014**. CAEd/UFJF. 2014. Disponível em: <http://www.spaece.caedufjf.net/wp-content/uploads/2016/08/CESPAECE-2014-RP-MT-1EM_2EM_3EM-WEB.pdf>. Acesso em: 30 jan. 2018.

FERNANDES, Mayara Leal Reis. **Utilização dos cadernos digitais para o ensino de lógica de programação**. 2023. Dissertação de Mestrado – PROFMAT/Universidade Federal de Santa Maria, Santa Maria.

FETTER, Sandro; LIMA, Edna Lucia da Cunha; LIMA, Guilherme Cunha. TECNOLOGIA TIPOGRÁFICA E ESCRITA MANUAL: O UNICODE, O OPENTYPE E AS INOVAÇÕES NA PROGRAMAÇÃO DE FONTES DE SIMULAÇÃO DA ESCRITA HUMANA. [S.l.].

FLEISCHER, Yannik; BIEHLER, Rolf; SCHULTE, Carsten. Teaching and learning data-driven machine learning with educationally designed Jupyter notebooks. **Statistics Education Research Journal**, v. 21, n. 2, p. 7–7, 2022.

AL-GAHMI, Abdulmalek; ZHANG, Yong; VALLE, Hugo. Jupyter in the classroom: An experience report. In: PROCEEDINGS of the 53rd ACM Technical Symposium on Computer Science Education. [S.l.: s.n.], 2022. P. 425–431.

GRAVE, Leomir Augusto Severo. **O pensamento computacional na prática: uma experiência usando python em aulas de matemática básica**. 2021. Dissertação de Mestrado – PROFMAT/Universidade Federal de Santa Maria.

GUIDONI, Karl. **Cheat Sheet: Jupyter Notebook**. [S.l.: s.n.], 2024. Disponível em: <https://karloguidoni.com/til/cheatsheets/jupyter-notebook/>. Acesso em: 23 out. 2024.

JULIA DOCUMENTATION. **Funções Matemáticas**. [S.l.: s.n.]. https://julia-cn.readthedocs.io/pt-br/latest/manual/mathematical-operations.html.

KHATIB, E. BRIDGING THEORY AND PRACTICE: REVOLUTIONIZING STEM EDUCATION WITH JUPYTER. In: INTED2024 Proceedings. [S.l.: s.n.], 2024. P. 5984–5994. DOI: 10.21125/inted.2024.1575.

LARMAN, Craig. **Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos**. São Paulo: Bookman, 2013.

MAÑÀ MARÍN, Ferran. **Jupyter notebooks as a development and documentation tool for supporting computer programming learning among adolescents: a case study in a k-18 school**. 2019. Monografia de Graduação – Universitat de Barcelona.

MEC. **Base Nacional Comum Curricular**. 2018. Disponível em: <https://basenacionalcomum.mec.gov.br/>. Acesso em: 1 mai. 2025.

_____. **Parâmetros Curriculares Nacionais para o Ensino Médio: Parte III – Ciências da Natureza, Matemática e suas Tecnologias**. 2000. Disponível em: <http://portal.mec.gov.br/seb/arquivos/pdf/blegais.pdf>. Acesso em: 12 jan. 2025.

PROJECT JUPYTER COMMUNITY. **Project Jupyter**. 2024. Disponível em: <https://jupyter.org/>. Acesso em: 20 dez. 2024.

_____. **Project Jupyter Documentation**. 2024. Disponível em: <https://docs.jupyter.org/en/latest/index.html>.

SAMPAIO, Alan de Souza. **Sistema permanente de avaliação da educação básica do Ceará (SPAECE): detalhamento da matriz de referência de matemática do 3° ano do ensino médio**. 2018. Dissertação de Mestrado – Universidade Estadual do Ceará/PROFMAT, Quixadá.

SANTOS, Diego Rabelo dos. **Estratégias metodológicas para o estudo das funções trigonométricas e funções hiperbólicas**. 2021. Dissertação de Mestrado – PROFMAT/Universidade Federal Rural do Semi-Árido.

SANTOS LIMA, Gabriel dos; NETO, José Aprígio Carneiro. LINGUAGEM DE PROGRAMAÇÃO JULIA: uma linguagem feita para a ciência. **P2P & Inovação**, v. 1, n. 1, 2024. DOI: 10.21728/p2p.2024v11n1e-7060.

SCHÖNBRODT, Sarah; WOHAK, Kirsten; FRANK, Martin. Digital tools to enable collaborative mathematical modeling online. **Modelling in Science Education and Learning**, v. 15, n. 1, p. 151–174, 2022.

SEDUC–CE. **Sistema Permanente de Avaliação da Educação Básica do Ceará (SPAECE)**. 2025. Disponível em: <https://www.seduc.ce.gov.br/spaece/>. Acesso em: 15 jun. 2025.

SEDUC–CE/CAED DIGITAL. **Avaliação e Monitoramento Ceará – Matrizes de Referência do SPAECE**. 2025. Disponível em: <https://avaliacaoemonitoramentoceara.caeddigital.net/>. Acesso em: 15 jun. 2025.

SILVA, Renildo Franco da; CORREA, Emilce Sena. Novas tecnologias e educação: a evolução do processo de ensino e aprendizagem na sociedade contemporânea. **Educação & Linguagem**, v. 1, n. 1, p. 23–35, 2014.

SOMMERVILLE, Ian. **Engenharia de Software**. 10. ed. São Paulo: Pearson, 2016.

SOUSA, Hernandes Macedo de. **Investigações de Carteira com o Python como Proposta no Ensino de Elementos da Estatística Descritiva**. 2023. Dissertação de Mestrado – PROFMAT/Universidade Federal do Pará.

SOUZA MAGALHAES, José Wesley de; AVALOS, Josué Barbosa; SANTOS, Leonardo Júnio Alves dos. Dossiê Linguagem Julia. Universidade Federal de Viçosa. [S.l.], 2017.

SUTRINI, Claudio; PASSARO, Davide; PALLOTTA, Filippo et al. The potential of using jupyter notebook in physics education: Experimentation for high school students. **Il nuovo cimento C**, p. 1–4, 2022.

THE JULIA LANGUAGE. **Unicode Input in Julia**. [S.l.: s.n.], 2024. Disponível em: <https://docs.julialang.org/en/v1/manual/unicode-input/>. Acesso em: 8 fev. 2025.

THOMPSON, JaCoya; WU, Sally PW; MILLS, Jacob. The use of computer programming in a secondary mathematics class. In: 2020 ASEE Virtual Annual Conference Content Access. [S.l.: s.n.], 2020.

TOPSAKAL, Oguzhan. Teaching algorithms design approaches via interactive jupyter notebooks. **European Journal of Technique**, v. 13, n. 1, p. 1–6, 2023.

VIAL, Gregory; NEGOITA, Bogdan. Teaching programming to non-programmers: the case of Python and jupyter notebooks. In: ICIS 2018 Proceedings. [S.l.: s.n.], 2018.

YAVUZ TEMEL, Güler; BARENTHIEN, Julia; PADUBRIN, Thore. Using Jupyter Notebooks as digital assessment tools: An empirical examination of student teachers' attitudes and skills towards digital assessment. **Education and Information Technologies**, Springer, p. 1–30, 2025.

ZASTRE, Michael. Jupyter notebook in CS1: An experience report. In: PROCEEDINGS of the 24th Western Canadian Conference on Computing Education. [S.l.: s.n.], 2019. P. 1–6.