UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I - CAMPINA GRANDE
PRÓ REITORIA DE PÓS-GRADUAÇÃO E PESQUISA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA
MESTRADO PROFISSIONAL EM MATEMÁTICA EM REDE NACIONAL

FLÁVIO RÉGIS ALVES JÚNIOR

O USO DO PYTHON NO ENSINO, APRENDIZAGEM E COMO FERRAMENTAL DO DOCENTE DE MATEMÁTICA

CAMPINA GRANDE
2026

FLÁVIO RÉGIS ALVES JÚNIOR

O USO DO PYTHON NO ENSINO, APRENDIZAGEM E COMO FERRAMENTAL DO DOCENTE DE MATEMÁTICA

Dissertação apresentada à Coordenação do Curso de Mestrado Profissional em Matemática em Rede Nacional da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Mestre em Matemática em Rede Nacional - PROFMAT.

Área de concentração: Ensino Básico Matemática

Orientador: Prof. Dr. Israel Buriti Galvão

CAMPINA GRANDE
2026

A474u Alves Júnior, Flávio Régis.
O uso do Python no ensino, aprendizagem e como ferramental do docente de matemática [manuscrito] / Flávio Régis Alves Júnior. - 2026.
93 f. : il. color.

Digitado.

Dissertação (Mestrado Profissional em Matemática em Rede Nacional) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia, 2026.

"Orientação : Prof. Dr. Israel Burití Galvão, Departamento de Matemática - CCT".

1. Python. 2. Ensino de matemática. 3. Algoritmos. I. Título

21. ed. CDD 327.7

Elaborada por Bruno Rafael Freitas de Lima - CRB - BC 15/1021

FLÁVIO RÉGIS ALVES JÚNIOR

O USO DO PYTHON NO ENSINO, APRENDIZAGEM E COMO FERRAMENTAL DO DOCENTE DE MATEMÁTICA

Dissertação apresentada à Coordenação do Curso de Mestrado Profissional em Matemática em Rede Nacional da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Mestre em Matemática em Rede Nacional - PROFMAT

Linha de Pesquisa: Ensino Básico Matemática.

Aprovada em: 20/02/2026.

BANCA EXAMINADORA

Documento assinado eletronicamente por:

- **Israel Burití Galvão** (***.241.144-**), em **02/03/2026 10:10:10** com chave **2498989a163911f18df8bef80df2d37b**.
- **Weiller Felipe Chaves Barboza** (***.145.684-**), em **02/03/2026 13:48:07** com chave **972d29e8165711f197072684aadcce22**.
- **Aldo Trajano Louredo** (***.317.454-**), em **02/03/2026 12:21:34** com chave **7fd3be1c164b11f1918ebef80df2d37b**.

Documento emitido pelo SUAP. Para comprovar sua autenticidade, faça a leitura do QrCode ao lado ou acesse https://suap.uepb.edu.br/comum/autenticar_documento/ e informe os dados a seguir.
Tipo de Documento: Folha de Aprovação do Projeto Final
Data da Emissão: 02/03/2026
Código de Autenticação: 51fbe6

Dedico este trabalho a Deus, criador da vida e doador da Sabedoria, familiares e amigos e em especial a Isabela Silva Régis, filha amada que motiva meu avançar.

AGRADECIMENTOS

Sou grato a Deus pelo dom da vida, pelo ar que respiro e pelo conhecimento necessário para compreender, ainda que em parte, a matemática. Agradeço aos meus pais por cuidarem de mim e por incentivarem, desde cedo, os meus estudos. Estendo minha gratidão aos meus familiares e amigos e, em especial, à minha filha amada, Isabela Silva Régis, por impulsionar minhas ações voltadas ao ensino e à aprendizagem da matemática. Sua dedicação, habilidades e determinação me inspiram profundamente. Também sou grato a todos os professores do programa PROFMAT da UEPB, em especial ao meu Orientador, Israel Buriti.

“O que sabemos é uma gota, o que ignoramos é um oceano.”(Isaac Newton, matemático e físico)

RESUMO

O Programa de Mestrado em Matemática (PROFMAT) tem desempenhado papel fundamental no aperfeiçoamento da formação matemática. Ao longo das atividades acadêmicas, foram aprofundados os estudos e consolidada uma base consistente de conhecimentos, com vistas a contribuir para o ensino da Matemática. Nesse contexto, o presente trabalho tem como objetivo apresentar como a Matemática, associada à linguagem de programação Python, pode atuar como ferramenta de apoio ao processo de ensino e aprendizagem, auxiliando docentes e discentes. Esta obra também apresenta conceitos relacionados aos algoritmos de programação e à análise de sua eficiência, considerando-se a relação entre o tamanho da entrada e o tempo de execução. Evidencia-se que as funções matemáticas fornecem ferramentas, por meio das notações assintóticas, que permitem a classificação da eficiência dos algoritmos, com destaque para a teoria do Big O. Defende-se que essa abordagem configura-se como uma metodologia ativa no ensino de conteúdos matemáticos, especialmente no que se refere à análise e interpretação de gráficos de funções. Ainda relacionado à eficiência de algoritmos, e considerando que estes constituem um tipo de tecnologia, abordamos alguns algoritmos de ordenação: Insertion Sort, Merge Sort e Quick Sort. Definimos os procedimentos de execução de cada um deles na ordenação de elementos de uma sequência e comparamos seus respectivos tempos de execução. Esses algoritmos também envolvem conceitos relacionados a arranjos matemáticos, sequências e fatorial. Além de contribuir para o ensino da matemática, essa abordagem tecnológica pode despertar o interesse dos estudantes por uma área profissional em expansão. Por fim, apresenta-se uma proposta de ensino de Matemática mediada pela linguagem de programação Python, direcionada a estudantes do ensino básico. A proposta está organizada em uma sequência didática composta por 12 aulas, estruturadas de forma contextualizada, com o objetivo de promover a construção dos conceitos matemáticos por meio da utilização de recursos tecnológicos, tais como notebooks, aplicativos educacionais e acesso à internet.

Palavras-chave: python; ensino de matemática; algoritmos.

ABSTRACT

The Professional Master's Program in Mathematics (PROFMAT) has played a fundamental role in improving mathematical education. Throughout the academic activities, studies were deepened and a consistent knowledge base was consolidated, with the aim of contributing to the teaching of Mathematics. In this context, the present work aims to present how Mathematics, combined with the Python programming language, can act as a supporting tool in the teaching and learning process, assisting both teachers and students. This work also addresses concepts related to programming algorithms and the analysis of their efficiency, considering the relationship between input size and execution time. It is emphasized that mathematical functions provide theoretical support, through asymptotic notations, for classifying the efficiency of algorithms, with particular emphasis on Big O notation. It is argued that this approach constitutes an active methodology in the teaching of mathematical content, especially with regard to the analysis and interpretation of function graphs.Still concerning algorithm efficiency, and considering algorithms as technological resources applicable to teaching, some sorting algorithms are addressed, such as Insertion Sort, Merge Sort, and Quick Sort. The execution procedures of each of these algorithms in sorting the elements of a sequence are defined, and their respective execution times are compared. These algorithms also make it possible to explore mathematical concepts related to arrangements, sequences, and factorials. In addition to contributing to the teaching of Mathematics, this technological approach may stimulate students' interest in a professional field that is in constant expansion.Finally, a proposal for teaching Mathematics mediated by the Python programming language is presented, aimed at basic education students. The proposal is organized into a didactic sequence composed of 12 lessons, structured in a contextualized manner, with the objective of promoting the construction of mathematical concepts through the use of technological resources, such as notebooks, educational applications, and internet access.

Keywords: python; mathematics teaching; algorithms.

SUMÁRIO

1 INTRODUÇÃO

O ensino da Matemática sempre foi desafiador, pois muitos alunos acabam construindo uma percepção equivocada sobre a disciplina. Despertar o interesse e desconstruir a ideia de que a Matemática é difícil e de que grande parte do que se aprende não tem aplicação prática na vida são barreiras que precisam ser superadas no processo de ensino.

É fato que, desde a década de 1990, houve um crescimento acelerado no uso das tecnologias. O advento da internet, o acesso a computadores, softwares, smartphones, aplicativos, jogos virtuais, redes sociais e ferramentas de comunicação instantânea transformaram a maneira de viver e de realizar as tarefas do dia a dia.

Vivemos na era dos chamados nativos digitais, indivíduos que, desde cedo, mantêm contato constante com tecnologias digitais. O educador e pesquisador Marc Prensky (2001) utiliza esse termo para classificar a geração de jovens que nasceu em um contexto marcado pela ampla disponibilidade de informações rápidas e facilmente acessíveis por meio da internet. Essa transformação na estrutura da vida cotidiana é também responsável por uma nova forma de perceber o mundo e de aprender.

Nesse contexto, Kenski (2010) destaca que as tecnologias digitais têm provocado mudanças significativas nos processos de ensino e aprendizagem, exigindo da escola e dos educadores novas metodologias e práticas pedagógicas alinhadas à realidade dos estudantes. Corroborando essa perspectiva, a Lei de Diretrizes e Bases da Educação Nacional (LDB), instituída pela Lei nº 9.394, de 20 de dezembro de 1996, estabelece, em seu artigo 2º, que a educação tem como finalidade "o pleno desenvolvimento do educando, seu preparo para o exercício da cidadania e sua qualificação para o trabalho" (BRASIL, 1996).

Diante da necessidade de um ensino contextualizado de Matemática, em que o acesso às tecnologias digitais ocorre cada vez mais cedo, torna-se pertinente considerar o uso de linguagens de programação no ensino básico.

Os conteúdos matemáticos desenvolvidos nessa etapa da educação básica têm como objetivo principal o desenvolvimento de diversas competências, entre as quais se destaca a capacidade de "compreender e utilizar, com flexibilidade e precisão, diferentes registros de representação matemática (algébrico, geométrico, estatístico, computacional etc.), na busca de soluções e na comunicação de resultados de problemas" (BRASIL, 2018).

Diante das circunstâncias elencadas, este trabalho propõe que o ensino de conteúdos matemáticos seja realizado com o auxílio de ferramentas tecnológicas, como a linguagem de programação Python, devido à sua simplicidade e versatilidade, além do estudo da eficiência de algoritmos, como a Teoria do Big O e os algoritmos de ordenação. Isso, além de proporcionar um aprendizado de Matemática de maneira aplicada, coloca o aluno na condição de protagonista, deixando de ser apenas um receptor de conhecimentos e passando a atuar como construtor de saberes. Além disso, essa nova postura promove o

engajamento e a autocrítica (ASHOKA; ALANA, 2017).

Com o objetivo de demonstrar como a linguagem de programação Python pode desempenhar um papel relevante no ensino da Matemática, faz-se necessário compreender o processo histórico de evolução da computação e o caráter inovador dessa ferramenta tecnológica. Nesse sentido, no início desta obra apresenta-se uma breve abordagem histórica da linguagem Python, destacando-se a perseverança de seu criador, Guido van Rossum, bem como as contribuições de outros programadores no processo de desenvolvimento e consolidação de seus principais conceitos. Também são abordados de maneira detalhada os principais conceitos relacionados à linguagem de programação Python, por meio de um tutorial cujo objetivo é possibilitar que alunos do ensino básico programem algoritmos baseados em fórmulas matemáticas. Concluo que se trata de uma ferramenta que pode auxiliar no ensino de conteúdos matemáticos para alunos do ensino básico.

2 LINGUAGEM DE PROGRAMAÇÃO

Os primeiros computadores foram o britânico Colossus (1943) e os norte-americanos Mark I (1944) e ENIAC (1946). Eles eram muito diferentes dos que conhecemos atualmente, a começar pelo tamanho: eram enormes, chegando a ocupar uma sala inteira, e possuíam operações bastante limitadas. Os programas eram escritos em linguagem de máquina, composta por zeros e uns, e apenas um número restrito de pessoas dominava essa tecnologia.

Na década de 1950, Grace Hopper criou uma linguagem de programação, FLOW-MATIC, próxima da linguagem natural, o que impulsionou o desenvolvimento de programas, pois facilitou a comunicação com a máquina. Com o tempo, foram criadas linguagens de programação cada vez mais próximas da linguagem natural, como a linguagem BASIC, criada em 1964 por John G. Kemeny e Thomas E. Kurtz.

Ao apresentar a definição de uma linguagem de programação no contexto educacional, podemos afirmar que pode ser compreendida como um recurso pedagógico que possibilita ao estudante desenvolver o pensamento lógico, a capacidade de resolução de problemas e a compreensão de conceitos abstratos, ao traduzir ideias e algoritmos em instruções que podem ser executadas por um computador (VALENTE, 2016). Nesse contexto, encaixa-se a linguagem de programação Python, que apresenta simplicidade ao programador, é de fácil aprendizado e muito próxima da linguagem comum (ROJAS; KOSTIN, 2018).

Programas Tradutores e Compiladores

Para facilitar a criação de programas computacionais foram criados os programas tradutores, que tem a finalidade de transformar os códigos escritos pelos programadores (código-fonte), em um código que o computador entenda, um código de máquina (código executável).

- Interpretadores: leem e verificam um código-fonte e convertem para um programa executável instrução a instrução.
- Compiladores: verificam e convertem o programa-fonte em um código executável todo de uma vez.
- Sistemas híbridos: combinam características dos Interpretadores e dos Compiladores. O Python é um bom exemplo de um sistema híbrido (ROJAS; KOSTIN, 2018).

2.1 Uma brevíssima introdução à Linguagem Python

Python é uma linguagem de programação amplamente reconhecida por sua versatilidade e simplicidade. Ela foi criada pelo programador holandês Guido Van Rossum, que inicialmente tentou desenvolver a linguagem ABC no início dos anos 1980, sem, contudo,

obter sucesso. Persistente, Guido utilizou as lições aprendidas com a ABC e começou a desenvolver Python em 1989, com a primeira implementação ocorrendo em janeiro de 1991 (ROJAS; KOSTIN, 2018).

A versão inicial do Python, a 0.9.0, foi lançada em 1991 e já incluía recursos avançados, como tipos de dados e funções para tratamento de erros. O Python 1.0 foi introduzido em 1994, trazendo novas funcionalidades que facilitavam o processamento de listas de dados. As versões subsequentes, Python 2.0, lançada em 16 de outubro de 2000, e Python 3.0, lançada em 3 de dezembro de 2008, introduziram uma série de novos recursos voltados para aprimorar a experiência do programador.

Embora o nome "Python" também seja associado a uma espécie de serpente, a nomenclatura da linguagem é, na verdade, uma homenagem ao grupo de comédia britânico Monty Python, no entanto, ao longo do tempo, a linguagem passou a ser frequentemente simbolizada pela imagem de uma serpente Python (em português, píton).

2.1.1 Paradigmas de Linguagem de Programação

Os paradigmas de linguagem de programação são modelos ou estilos de programação que definem a forma como os programas são escritos e organizados. Cada paradigma segue um conjunto de princípios que influenciam a estrutura do código, a forma de resolver problemas e a organização dos dados e funções.

Python suporta diversos paradigmas de programação, permitindo ao desenvolvedor escolher a abordagem mais adequada para cada problema. Entre os principais paradigmas suportados pelo Python, destacam-se a Programação Procedural, a Programação Orientada a Objetos (POO), a Programação Funcional e a Programação Orientada a Eventos.

1. No paradigma Procedural, o desenvolvimento é realizado por meio da escrita de procedimentos, que consistem em sequências de instruções a serem executadas na ordem em que são definidas. Este paradigma é particularmente eficaz para a solução de problemas simples e lineares, onde a execução sequencial de operações é adequada.

2. A Programação Orientada a Objetos (POO), por sua vez, organiza o código em torno de objetos, que são instâncias de classes. No ambiente Python, tudo é tratado como um objeto, desde números até strings`(str)` . A POO permite a criação de classes, que servem como moldes para gerar objetos, facilitando a modularização e a reutilização do código.

3. Já a Programação funcional foca na aplicação de funções puras, que são funções sem efeitos colaterais e que sempre retornam o mesmo resultado para as mesmas

entradas. Este paradigma é especialmente útil para a resolução de problemas complexos, especialmente aqueles que envolvem grandes volumes de dados, devido à sua capacidade de simplificar e abstrair operações matemáticas.

4. Por último, o paradigma orientado a eventos é amplamente utilizado no desenvolvimento de interfaces gráficas de usuário (GUIs) e jogos. Esse paradigma baseia-se na definição de respostas específicas a eventos, como cliques de mouse ou teclas pressionadas, permitindo a criação de interfaces dinâmicas e responsivas, onde as ações do usuário desencadeiam a execução de determinadas funções.

A versatilidade de Python reside em sua capacidade de integrar esses diversos paradigmas em uma única linguagem, oferecendo ao programador uma vasta gama de ferramentas e abordagens para a solução de problemas. Sua sintaxe simples e intuitiva, aliada à natureza interpretada da linguagem, contribui para um desenvolvimento mais rápido e acessível, tornando Python uma escolha ideal tanto para iniciantes quanto para programadores experientes, por isso o uso da linguagem para alunos do ensino básico não terá muitas dificuldades, com um pouco de conhecimento de informática eles serão capazes de compreender e aprender Python.

2.1.2 Python no Ensino Básico

O ensino da linguagem de programação Python, reconhecida por sua simplicidade na sintaxe e versatilidade de uso, surge como uma excelente alternativa para introduzir os estudantes do ensino básico no universo tecnológico.

Python, com seu suporte a diversos algoritmos matemáticos, conceitos de lógica e fórmulas como o algoritmo da divisão de Euclides, além de sua aplicação em temas contemporâneos, como robótica, inteligência artificial (IA), desenvolvimento de jogos e aplicativos, configura-se como uma ferramenta poderosa para despertar talentos e atender à crescente demanda por profissionais qualificados no mercado de programação.

Portanto, a inclusão do ensino de Python pode contribuir significativamente para tornar as aulas mais interessantes e imersivas. Essa prática educativa, de caráter protagonista, pode permitir que os alunos vivenciem experiências práticas capazes de conectar o aprendizado à realidade do mundo tecnológico, fortalecendo, assim, o vínculo entre a teoria e a prática, preparando os estudantes para os desafios do futuro (SANTOS, 2020).

A proposta deste trabalho é o ensino de conteúdos matemáticos com Python, a conexão de fórmulas matemáticas e algoritmos, o que pode desenvolver a tendência de ensino de matemática e informática. Saber usar recursos tecnológicos, hoje, é uma necessidade, e saber Python para aprender matemática pode contribuir para uma aula mais dinâmica. A construção de códigos para as fórmulas matemáticas e a automação de cálculos, através do fornecimento de valores ajuda na exploração da criatividade e engajamento fazendo do aprendizado algo divertido, inovador.

2.1.3 Onde encontrar e como manusear Python

Python é uma linguagem de programação multiplataforma, o que significa que pode ser executada em todos os principais sistemas operacionais, como Windows, Linux, macOS, iOS e Android. Isso permite utilizar o Python tanto em computadores pessoais quanto em smartphones. Qualquer programa escrito em Python deve funcionar em qualquer computador moderno que tenha o interpretador da linguagem instalado (MATHEUS, 2016).

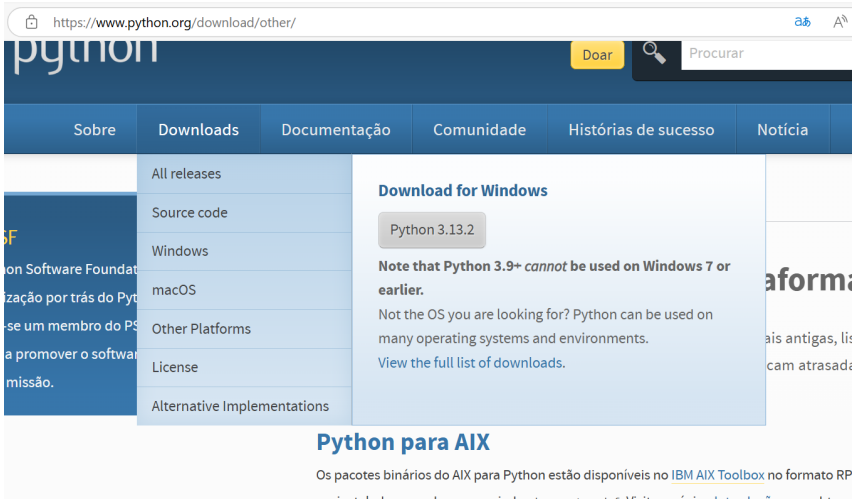
Aqui apresento a instalação, os conceitos e as funcionalidades relacionadas ao Python. Também ensino a construção dos códigos de programação mais usuais e, assim, pretendo fornecer um manual básico de Python voltado para alunos do ensino básico, cujo propósito é habilitá-los a usar algoritmos para construir fórmulas matemáticas.

Instalação

Como já foi abordado, o Python é compatível com vários sistemas operacionais tais como WINDOWS, LINUX, ANDROID, iOS. A instalção não é algo que apresente muitas dificuldades, com um pouco de conhecimento de informática, o download do arquivo de instalação do Python pode ser efetuado.

Antes de iniciar o processo de instalação é preciso verificar qual o sistema operacional, depois no site `http://www.Python.org/download/` é possível fazer o download do programa Python 3.14.2, versão mais recente. Nesse site existem várias opções de download compatível com várias opções de sistemas operacionais.

Figura 1 – Tela de download do Python

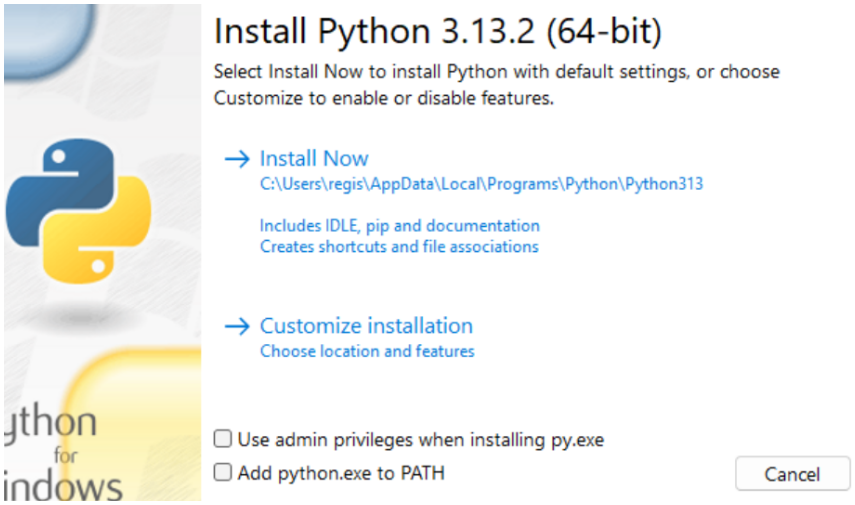


Fonte: Disponível em: `https://www.python.org/downloads/`.

Com o arquivo baixado, o processo de instalação ocorre de forma automatizada, sendo necessário marcar as caixas 'Use administrative privileges when installing py.exe' e 'Add Python.exe to PATH', o que facilita a configuração correta do programa.

Ao escolher a opção ‘Customize installation’, surge uma nova tela com configurações padrão. Ao clicar em ‘Next’, inicia-se o processo de instalação, que ocorre rapidamente. Por fim, a instalação é concluída ao clicar no botão ‘Finish’.

Figura 2 – Caixa de diálogo do processo de instalação



Fonte: Elaborado pelo autor, 2025.

Após a instalação, no módulo terminal é possível escrever códigos e aperfeiçoá-los. Já no módulo interativo IDLE, um ambiente mais atraente, podemos treinar algoritmos. Também é possível abrir um novo documento e testar sua funcionalidade na opção ‘RUN’, após salvar o arquivo. Como já mencionado, não é difícil instalar o Python no computador; feito isso, o programa estará operacional para uso e apto para o desenvolvimento criativo de algoritmos de programação.

Variáveis

A noção de variável é um conceito antigo inicialmente concebido para a máquina de Von Neumann. Nela, uma variável (identificador) ocupa uma posição na memória e seu valor em outra, basicamente a associação entre um nome e um valor. Existem dois tipos de variáveis, imutáveis e mutáveis. As Imutáveis que se atribui um valor a uma variável é estabelecido um endereço na memória para o nome da variável e outro para o seu valor. O ponteiro faz ligação entre os dois. Mutáveis o ponteiro aponta sempre para o mesmo endereço, mesmo quando se altera o endereço da variável.

Em Python usamos o símbolo de igualdade (=) para atribuir variáveis; neste caso, o símbolo não denota a igualdade no sentido matemático, mas significa um comando de atribuição.

Na linguagem de programação, é muito importante a habilidade de atribuir variáveis na construção dos algoritmos, e são recorrentes erros relacionados com atribuições inadequadas. Esquecer uma aspa em dados do tipo string (`str`) ou deixar de caracterizar

um dado numérico como inteiro (`int`) ou real (`float`) gera erros de sintaxe. Por isso, ser atencioso e detalhista ajuda na confecção correta dos códigos de programa.

Existem diversas possibilidades para a atribuição de variáveis, tais como: um nome, um dado numérico, uma fórmula matemática, uma mensagem, uma grandeza ou uma medida. Esse recurso é bastante amplo e pode ser utilizado na construção de diversas fórmulas interativas, funcionando como um reforço didático eficaz.

Exemplo 2.1. Segue a atribuição de valores às variáveis a e b e a operação soma entre elas.

```
a = 5          # atribui valor 5 à variável a;
b = 7          # atribui valor 7 à variável b;
soma = a + b   # atribui à variável "soma" resultado da soma
               # de a mais b;
```

Linhas de programação

Um programa desenvolvido na linguagem Python é estruturado em linhas lógicas, as quais podem abranger uma ou múltiplas linhas físicas de código. Cada linha lógica corresponde a uma instrução específica, podendo envolver a atribuição de valores a variáveis, a solicitação de entrada de dados pelo usuário, a execução de funções pré-definidas ou personalizadas, bem como a realização de operações aritméticas ou lógicas. Essa organização permite a construção de algoritmos claros, modulares e de fácil manutenção, favorecendo a compreensão e o desenvolvimento de soluções computacionais eficientes.

Cada linha de código representa uma unidade fundamental na construção do algoritmo que orienta a execução de um programa computacional. A elaboração precisa e coerente de cada instrução é essencial, uma vez que falhas, mesmo que localizadas, podem comprometer a integridade do algoritmo, resultando na interrupção de sua execução ou na produção de resultados incorretos. Dessa forma, a atenção aos detalhes na escrita do código é um aspecto crucial no desenvolvimento de soluções computacionais robustas e funcionais.

As instruções de um programa são dispostas em ordem sequencial crescente, sendo cada uma delas associada a uma posição identificada por um número natural. Tal ordenação não ocorre de maneira arbitrária, mas reflete uma característica fundamental da construção algorítmica: a necessidade de uma sequência lógica e encadeada de comandos, em que cada etapa subsequente depende da execução correta da anterior. Nesse sentido, as linhas de código desempenham um papel estruturante na programação, funcionando como um ambiente organizado de execução, análogo ao plano cartesiano na matemática, ao oferecer um referencial espacial e lógico para o desenvolvimento de algoritmos coerentes e funcionais.

Exemplo 2.2. Segue o exemplo da aplicação de linhas de execução na construção de algoritmos:

```
# Solicita o nome do usuário
nome = input(‘Qual é o seu nome? ’)
print(f‘Seja bem-vindo, {nome}!’)

# Solicita a idade do usuário
idade = input(‘Quantos anos você tem? ’)
print(‘Estamos felizes por você compartilhar essa informação!’)

# Finaliza o programa
exit()
```

Dados Primitivos

Python adota um sistema de tipagem dinâmica e forte, em que os tipos são intrínsecos aos objetos manipulados, determinando sua natureza e as operações aplicáveis. Essa característica é essencial para a correta execução de algoritmos e a modelagem eficiente de soluções computacionais. Para desenvolver programas, é fundamental selecionar adequadamente os dados a serem processados, e Python simplifica esse processo ao inferir automaticamente os tipos das variáveis durante a execução, sem exigir declarações explícitas.

Os tipos de dados em Python classificam-se em primitivos (simples) e estruturados (compostos). Os primitivos são reconhecidos nativamente pela linguagem, dispensando construções adicionais. Incluem:

- Dados numéricos: inteiros (`int`), reais (`float`) e complexos (`complex`);
- Textos: strings (`str`);
- Valores lógicos: booleanos (`bool`), com True ou False;
- Valor nulo: `None`, representando ausência de valor.

Já os tipos estruturados consistem em agrupamentos de dados primitivos, organizados de forma ordenada ou não, utilizando delimitadores como colchetes (`[ ]`), chaves (`{ }`) ou parênteses (`( )`). Entre eles, destacam-se: listas, tuplas, conjuntos e dicionários, cada um com particularidades de acesso e manipulação. Esta categorização sistemática visa facilitar o entendimento e a aplicação prática dos tipos de dados em Python.

Numéricos

Os dados numéricos em Python são classificados em três categorias principais: inteiros (`int`), reais (`float`) e complexos (`complex`). Esses tipos numéricos são empregados para representar quantidades e realizar operações matemáticas.

Diferentemente das cadeias de caracteres (*strings*), os números não exigem delimitação por aspas ou qualquer outro caractere especial. Basta inserir o valor numérico, e ele será automaticamente interpretado pelo interpretador da linguagem, sem a necessidade de conversões explícitas.

Para distinguir números inteiros de números reais em Python, utiliza-se o ponto (.) como separador decimal, ao contrário da vírgula empregada em algumas convenções numéricas. Além disso, não há um limite pré-definido para o tamanho dos números armazenáveis, sendo restritos apenas pela capacidade da memória do computador.

No caso dos números complexos, sua estrutura é composta por duas partes: a parte real e a parte imaginária. Em Python, essa representação segue o formato '`real + imagj`', em que a unidade imaginária i ($\sqrt{-1}$) é denotada pela letra j, conforme a convenção adotada na linguagem.

Exemplo 2.3. A seguir, apresentamos alguns exemplos de como diferentes tipos de números podem ser representados na linguagem Python:

```
# Soma de números inteiros
3 + 6
9 # Resultado

# Representação de um número real (ponto flutuante)
3.0  # Número real, indicado pelo uso do ponto decimal

# Número real com parte decimal
6.899999999995  # Representação de um número real com maior precisão

# Número complexo (composto por parte real e imaginária)
a = 2.0 + 3.0j  # O sufixo 'j' indica a parte imaginária
```

Lógicos ou Booleanos

Um dos tipos primitivos de dados simples mais recorrentes na linguagem de programação Python é o tipo lógico, comumente denominado booleano (bool). A designação "booleano" tem origem na lógica matemática desenvolvida por George Boole (1815–1864),

matemático britânico do século XIX, cuja teoria fundamenta-se em operações lógicas binárias. Essa lógica opera com apenas dois valores possíveis, verdadeiro (True) e falso (False), que, no contexto computacional, são representados pelos dígitos binários 1 e 0, respectivamente. Devido à natureza binária dos sistemas digitais, a lógica booleana (`bool`) tornou-se essencial na ciência da computação, sendo amplamente empregada na construção de estruturas de controle de fluxo, expressões condicionais e validação de dados em algoritmos e programas computacionais (ROJAS; KOSTIN, 2018).

O tipo de dado booleano (`bool`) é frequentemente utilizado na implementação de estruturas de controle, notadamente nas instruções condicionais e nos laços de repetição. Em razão de sua natureza lógica, esse tipo de dado permite a formulação de sentenças condicionais cuja avaliação determina o prosseguimento ou a interrupção de blocos específicos de código. Assim, é possível elaborar algoritmos de automação nos quais a execução de determinadas ações está diretamente vinculada ao valor lógico de uma condição previamente estabelecida. Essa característica torna os valores booleanos (`bool`) indispensáveis à construção de sistemas computacionais que demandam tomada de decisão e adaptação dinâmica a diferentes estados de execução.

Exemplo 2.4. segue aqui algumas representações do uso de dados booleanos:

```
# Avaliação booleana para comparadores matemáticos simples.
5 > 3       # Entrada de desigualdade;
True        # Saída booleana para a entrada acima;

14 + 4 == 5 # Entrada de igualdade;
False       # Saída booleana para a entrada acima;
```

Strings

Uma string é uma sequência de caracteres. Tudo o que estiver entre aspas é considerado uma string em Python. Basicamente, são textos na linguagem de programação, utilizados na construção de mensagens. Devem ser escritas entre aspas simples ou duplas, no início e no final (MATHEUS, 2016).

O tipo de dado strings (`str`) é amplamente utilizado na programação em Python devido à sua versatilidade e facilidade de manipulação. Ele é empregado em diversas situações, como exibição de textos, armazenamento de informações alfanuméricas, atribuição a variáveis e até mesmo na combinação de valores provenientes de diferentes variáveis. Embora seja um tipo de dado intuitivo, é de extrema importância que o programador adote boas práticas de codificação para evitar erros de sintaxe, ou seja, situações nas quais o Python não reconhece a strings (`str`) como válida devido a problemas no formato ou estrutura da declaração.

No contexto da programação em Python, é imperativo observar a correta manipulação de dados do tipo string (str) quando combinados com outros tipos de dados primitivos, como números reais. O tratamento inadequado de números como strings, ou seja, ao inseri-los entre aspas, resulta em uma falha de interpretação no código. Quando isso ocorre, o número é tratado como uma sequência de caracteres, o que impede que ele seja utilizado em operações aritméticas. Dessa forma, é essencial garantir que os números não sejam envolvidos por aspas, a fim de preservar seu valor numérico e permitir a realização de cálculos de forma correta e eficiente.

A integração de strings nos algoritmos permite a criação de mensagens informativas, o que facilita a comunicação entre a máquina e o usuário. Dessa forma, é possível fornecer feedbacks claros sobre os cálculos realizados, além de tornar as fórmulas matemáticas mais acessíveis.

Por exemplo, ao calcular o valor de uma expressão matemática e apresentá-lo dentro de um contexto textual, Python facilita a compreensão do estudante ao contextualizar o resultado. A utilização de strings permite que o código exiba resultados de maneira intuitiva e compreensível, o que é especialmente útil no processo de ensino de conceitos matemáticos.

Exemplo 2.5. Apresentação mensagem combinando dado numérico e dado *string* (`str`):

```
# Imprime o número 2 e uma mensagem informando que é primo
print(2, 'é um número primo') # Entrada.
2 é um número primo           # Saída.
```

Dados estruturados

No contexto da ciência da computação, os tipos de dados estruturados referem-se àqueles que representam um agrupamento de dados primitivos, permitindo a organização, manipulação e acesso eficiente às informações. Em linguagens de programação como Python, esses tipos estruturados são fundamentais para o desenvolvimento de algoritmos e sistemas computacionais mais complexos. Os principais tipos de dados estruturados incluem: listas, tuplas, conjuntos e dicionários (ROJAS; KOSTIN, 2018).

Fazendo uma analogia, os dados estruturados podem ser comparados a uma parede construída por tijolos, em que cada tijolo representa um dado do tipo simples. Juntos, esses elementos formam uma unidade sólida e funcional, facilitando o tratamento mais sistemático e eficiente das informações.

O ordenamento de dados, a implementação de algoritmos voltados para a realização de buscas, bem como o tratamento de estruturas como listas, vetores e matrizes, constituem algumas das principais formas de utilização dos dados estruturados nas linguagens de programação.

Listas

É um conjunto ordenado de valores, cada valor possui um índice, ou seja uma localização na lista. Esses valores são chamados de elementos. A estrutura da lista é composta por chaves e vírgulas.

Para caracterizar o tipo de dado lista na linguagem de programação Python, é possível identificar algumas propriedades fundamentais. As listas são estruturas de dados mutáveis, ou seja, seus elementos podem ser modificados após a criação da estrutura.

Além disso, as listas permitem a duplicação de elementos, não havendo restrições quanto à repetição de valores. Essa característica as diferencia de outras estruturas, como os conjuntos (sets), que não admitem elementos duplicados. Dessa forma, as listas se mostram uma ferramenta versátil e amplamente utilizada no desenvolvimento de aplicações em Python.

Exemplo 2.6. Segue exemplo de lista com dados primitivos simples numéricos e strings (`str`):

```
a = [2,3,4,2,5,‘Bárbara’]   # Atribuição de variável;
a                           # Entrada.
[2, 3, 4, 2, 5, ‘Barbára’]  # Saída.
```

No exemplo apresentado, a lista possui seis elementos, sendo cinco valores numéricos e um do tipo string (str). Observa-se também que o número 2 aparece repetido, o que reforça a possibilidade de duplicação de elementos nessa estrutura.

Cada um desses elementos está associado a um índice, ou seja, a uma posição específica dentro da lista. A indexação em Python é baseada em zero, de modo que o primeiro elemento está na posição 0, o segundo na posição 1, e assim sucessivamente até o último elemento.

De maneira análoga a uma sequência matemática, cada termo de uma lista possui uma posição determinada por um número natural ou pelo zero. Em Python, existem diversos recursos que possibilitam a manipulação de listas de forma criativa e eficiente. Ainda em relação aos índices, os valores armazenados em uma lista podem ser acessados por meio de colchetes (`[]`), nos quais se insere o índice correspondente ao elemento desejado.

Exemplo 2.7. Segue lista contendo seis elementos tipo numérico e maneira de acessá-los por meio dos respectivos índices:

```
lista = [3,2,5,6,7,1] # Atribuição de variável lista;
lista[1]              # Entrada.
2                     # Saída.
```

```
lista[0]              # Entrada.
3                     # Saída.
lista[5]              # Entrada.
1                     # Saída.
```

Além de exibir os valores de uma lista, Python oferece diversas funções e comandos bastante úteis para sua manipulação. Um exemplo é o método append, que permite adicionar um elemento ao final da lista. Essa função modifica a lista original ao inserir um novo item, tornando-a dinâmica e expansível.

Outro método importante é o count, que informa quantas vezes um determinado objeto aparece na lista. Em termos de conteúdo matemático, trata-se de uma ferramenta útil em algoritmos que visam identificar a frequência absoluta de uma variável em uma amostra.

Também é possível manipular a ordem dos elementos presentes na lista. Para isso, Python disponibiliza o método sort, que organiza os valores em ordem crescente. Esse recurso é particularmente útil em análises que exigem a ordenação de dados para facilitar sua interpretação ou posterior processamento.

A versatilidade desse tipo de dado estruturado faz das listas uma excelente opção para a construção de algoritmos interativos. Explorar suas funcionalidades pode favorecer o ensino de conceitos matemáticos, uma vez que elas permitem representar sequências, manipular dados numéricos e implementar operações fundamentais de forma prática e acessível.

Conjuntos ou Set

A definição de conjunto em Python não difere significativamente daquela apresentada na matemática. De modo geral, um conjunto pode ser entendido como um grupo de elementos que compartilham uma determinada propriedade P.

Entre as principais características desse tipo de dado estruturado, destacam-se o fato de que seus elementos não são armazenados em uma ordem específica e não podem ser repetidos. Diferentemente das listas, os conjuntos eliminam automaticamente valores duplicados.

Embora apresentem certa semelhança visual com as listas, os conjuntos em Python são definidos entre chaves (`{}`), com os elementos separados por vírgulas, ao passo que as listas utilizam colchetes (`[]`).

Exemplo 2.8. Segue exemplo de dado estruturado do tipo conjunto em Python:

```
c = {1,1,3,4,5,6,6} # Atribuição de variável conjunto;
c                   # Entrada.
{1, 3, 4, 5, 6}     # Saída.
```

Em matemática, dados os conjuntos A e B, a operação de união de conjuntos, representada por $A \cup B$, consiste em reunir os elementos de A e de B, sem repetir os elementos comuns, formando um novo conjunto.

Esse procedimento também pode ser realizado em Python, por meio do método union. Vejamos um exemplo da operação de união entre conjuntos na linguagem:

Exemplo 2.9. Sejam as listas a e b, aqui mostramos como transformá-las em conjuntos e depois usamos o método de união de conjuntos.

```
a = [1,1,1,2,2,3,3]   # 1ª lista com 7 elementos;
a = set(a)            # Função transformar em conjunto;
a                     # Entrada de a.
{1, 2, 3}             # Saída.
b = [2,2,2,3,3,4,4,4] # 2ª lista de elementos;
b = set(b)            # função transformar em conjunto;
b                     # Entrada b.
{2, 3, 4}             # Saída.
c = a.union(b)        # método união de conjuntos;
c                     # Entrada c;
{1, 2, 3, 4}          # Saída conjunto união;
```

Analogamente, também é possível realizar as operações de interseção e diferença de conjuntos em Python. O método intersection() retorna um conjunto correspondente à interseção de dois conjuntos fornecidos, ou seja, contém apenas os elementos comuns a ambos. Já o método difference() remove todos os elementos compartilhados entre os conjuntos, retornando apenas aqueles que pertencem ao primeiro conjunto e não estão presentes no segundo.

Essas funcionalidades evidenciam como a linguagem de programação Python disponibiliza recursos eficazes para o desenvolvimento de algoritmos que envolvem conceitos matemáticos, contribuindo para uma abordagem mais prática e interativa no ensino da matemática.

Tuplas

A tupla é uma estrutura de dados presente na linguagem de programação Python que se assemelha à lista, diferenciando-se desta por ser um objeto imutável, ou seja, seus elementos não podem ser alterados após a sua criação. Os valores que compõem uma tupla são separados por vírgulas e podem ser definidos com ou sem o uso de parênteses,

sendo esta última forma a mais convencional e recomendada para fins de legibilidade e clareza sintática (ROJAS; KOSTIN, 2018).

Por se tratar de uma estrutura imutável, a tupla representa uma excelente alternativa para o armazenamento de dados que não devem ser alterados, como registros estáticos. Nesse sentido, uma variável do tipo tupla pode ser utilizada para armazenar informações cadastrais de uma pessoa, funcionando como uma lista de somente leitura e contribuindo para a integridade e segurança dos dados.

Exemplo 2.10. Segue exemplo de dado primitivo estrutura do tipo tupla contendo registros:

```
# Atribuição da variável Diogo a uma tupla contendo registros;
Diogo = ‘Mecânico’,‘Brasileiro’,‘pardo’, ‘formação superior’
# Entrada.
Diogo
# Saída.
(‘Mecânico’, ‘Brasileiro’, ‘pardo’, ‘formação superior’)
```

Como discutido anteriormente, esse tipo de estrutura se mostra particularmente útil em implementações voltadas à coleta de dados e à geração automatizada de documentos. Trata-se de uma funcionalidade cada vez mais requisitada em sistemas informatizados, sobretudo no contexto de soluções desenvolvidas para órgãos governamentais, onde a integridade e a consistência das informações são requisitos essenciais.

Dicionários

O dicionário é uma estrutura de dados amplamente utilizada na linguagem de programação Python, caracterizada pela organização de seus elementos em pares chave-valor. Esses pares são delimitados por chaves ({}) e separados entre si por vírgulas, sendo que cada chave é associada a um valor por meio do uso de dois pontos (:). Essa estrutura permite a recuperação eficiente de informações com base exclusivamente em suas chaves, o que a torna especialmente útil em aplicações que requerem acesso direto e rápido a dados de forma não sequencial.

Assim como em um dicionário tradicional da língua, no qual cada termo possui um ou mais significados associados, a estrutura de dicionário na linguagem Python organiza seus dados em pares denominados chave-valor. Nesse modelo, a chave funciona como o identificador único que possibilita o acesso ao dado, enquanto o valor representa a informação vinculada à respectiva chave. Por exemplo, em um sistema de cadastro, a chave pode corresponder ao nome de uma pessoa, e o valor, à sua nacionalidade. Essa estrutura possibilita a associação direta entre identificadores e seus conteúdos, promovendo eficiência na manipulação e recuperação de dados.

Exemplo 2.11. Representação de um dicionário na linguagem Python contendo as chaves Nome, País e profissão e os respectivos valores:

```
# Atribuição de um dado estrutura tipo dicionário a variável d;
d = {'Nome': 'João da Silva',
    'País': 'Brasil',
    'Profissão': 'Professor,
    }
# Entrada.
d
# Saída.
{ 'Nome': 'João', 'País': 'Brasil', 'Profissão': 'Professor'}
```

Sendo uma estrutura de dados altamente organizada, o dicionário em Python apresenta diversas funcionalidades que o tornam uma ferramenta robusta e versátil. É possível inicializá-lo tanto de forma vazia quanto previamente preenchido com dados, de acordo com as necessidades da aplicação. Além disso, a linguagem oferece um conjunto abrangente de métodos que possibilitam a interação com seus elementos, a realização de cópias, a verificação de chaves, entre outras operações. Essa combinação de flexibilidade e facilidade de uso consolida o dicionário como uma das estruturas mais eficazes para a manipulação e o armazenamento de dados em ambientes de programação.

2.2 Elementos de Programação

2.2.1 Input e Print

Os comandos de entrada e saída de dados constituem elementos essenciais para a interação entre o usuário e um programa computacional, permitindo a recepção de informações externas e a exibição de resultados processados. No contexto da linguagem de programação Python, tais operações são frequentemente realizadas por meio das funções `input()` e `print()`, amplamente utilizadas devido à sua simplicidade e eficácia. A função `input()` possibilita a inserção de dados pelo usuário durante a execução do programa, promovendo uma maior interatividade entre o sistema e o agente humano. Essa característica é particularmente relevante em aplicações que demandam respostas personalizadas, configurando-se como uma ferramenta fundamental para o desenvolvimento de sistemas interativos e responsivos.

Exemplo 2.12. Segue exemplo da aplicação do comando de entrada "input" na linguagem de programação Python. O algorítimo executa um comando de apresentação de perguntas para que o usuário insira os dados:

```
nome = input(‘Qual o seu nome: ’)          # Entrada.
Qual o seu nome: João                      # Saída.
idade = float(input(‘Qual a sua idade: ’))# Entrada.
Qual a sua idade: 35                       # Saída.
```

Conforme exemplificado, para realizar a entrada de um dado do tipo numérico em Python, é necessário especificar previamente se o valor será tratado como um número inteiro ou real. Essa definição é feita por meio do uso das funções `int()` ou `float()` associadas ao comando `input()`, respeitando a sintaxe adequada, incluindo os parênteses.

Os comandos de entrada também podem ser empregados como mecanismos de pausa na execução do programa, aguardando uma interação do usuário tipo pressionar qualquer tecla ou uma tecla específica antes de prosseguir.

Essa característica da linguagem Python, igualmente presente em diversas outras linguagens de programação, evidencia sua capacidade de estabelecer uma interface mais intuitiva entre o ser humano e a máquina. Tal abordagem favorece a clareza na comunicação e contribui para o desenvolvimento de sistemas mais acessíveis e eficientes.

Além dos comandos de entrada, é igualmente essencial, na construção de algoritmos e na execução de ações por parte do computador, o uso de comandos de saída, como o print. Tal comando tem como finalidade exibir informações na tela, permitindo a visualização dos dados processados pelo programa.

O print, assim como o input, constitui um dos elementos fundamentais da linguagem de programação Python e de outras linguagens. O domínio desse recurso é muito importante para iniciantes e profissionais da área pois possibilita a criação de programas simples ou desenvolvimento de aplicações mais complexas. A compreensão e aplicação adequada dos comandos de saída representam uma competência básica para a lógica de programação.

Podemos verificar logo abaixo uma das maneiras da aplicação dos comandos de entrada, input, e saída, print, na construção de códigos de programação.

Exemplo 2.13. No algoritmo a seguir o comando input solicita a entrada de dois valores reais fornecidos pelo usuário, atribui a uma terceira variável a razão entre as duas primeiras e por fim, o comando print imprime o valor da razão.

```
# Define variável a com a entrada de valor real;
a = float(input(‘digite uma número: ’))
digite uma número: 3                    # Entrada.
# Define variável b com a entrada de valor real;
b = float(input(‘digite outro número: ’))
digite outro número: 4                  # Entrada.
# Atribui a variável c o valor da razão entre e b;
```

```
c = a/b
# Imprime c.
print(c)
0.75                                    # Saída.
```

2.2.2 Aritmética e Python

Antes de aprofundar o estudo das funcionalidades da linguagem de programação Python, é fundamental compreender que sua lógica operacional está enraizada nos princípios da aritmética, ramo da matemática baseado em um sistema de axiomas e teoremas que fundamentam as operações inicialmente definidas sobre os números naturais e inteiros.

Na linguagem de programação Python, as operações aritméticas são implementadas de maneira acessível e direta, visto que os símbolos que as representam, bem como os princípios matemáticos que as regem, são nativamente interpretados pela linguagem, dispensando declarações prévias por parte do desenvolvedor.

O domínio dos fundamentos da aritmética constitui um elemento crucial para a elaboração precisa de algoritmos que envolvem operações com números inteiros, os quais representam um dos tipos primitivos numéricos mais elementares em linguagens de programação. Dada a expressiva importância desses conceitos matemáticos para a compreensão adequada das operações na linguagem Python, propõe-se, neste trabalho, a apresentação dos principais fundamentos da aritmética como base para o desenvolvimento lógico e computacional.

Aritmética

Segundo Abramo Hefez (2016), a Aritmética, denominação usualmente atribuída à parte elementar da Teoria dos Números, tem como marco inaugural a obra Os Elementos, de Euclides, por volta de 300 a.C. Seu desenvolvimento atingiu notável progresso nos trabalhos de Pierre de Fermat (1601–1665) e Leonhard Euler (1707–1783), consolidando-se, na contemporaneidade, como um dos pilares fundamentais da Matemática.

A teoria dos números aborda algumas propriedades relacionadas ao conjunto dos números inteiros.

Seja $\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$, o conjunto dos números inteiros munido das operações de adição $(a, b) \to a + b$ e de multiplicação $(a, b) \to a \cdot b$. As operações de adição e de multiplicação em $\mathbb{Z}$ possuem propriedades operacionais muito bem definidas, ou seja, dados, $a, b, c, a', b' \in \mathbb{Z}$, valem:

1. A adição e a multiplicação são bem definidas:

$$\text{se } a = a' \text{ e } b = b', \text{ então } a + b = a' + b' \text{ e } a \cdot b = a' \cdot b'.$$

2. A adição e a multiplicação são comutativas:

$$a + b = b + a \text{ e } a \cdot b = b \cdot a.$$

3. A adição e a multiplicação são associativas:

$$(a + b) + c = a + (b + c) \text{ e } (a \cdot b) \cdot c = a \cdot (b \cdot c).$$

4. A adição e multiplicação possuem elementos neutros:

$$a + 0 = a \text{ e } a \cdot 1 = a.$$

5. A adição possui elementos simétricos:

$$\text{existe } b = (-a) \text{ tal que } a + b = 0.$$

6. A multiplicação é distributiva com relação à adição:

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

.

A seguir, apresentamos alguns resultados que demonstram a consistência das operações elementares, as quais constituem a base para as operações implementadas na linguagem Python.

Teorema 2.1. $a \cdot 0 = 0$ *para todo* $a \in \mathbb{Z}$.

Demonstração. Pelas propriedades 4 e 6, temos:

$$a \cdot 0 = a \cdot (0 + 0) = a \cdot 0 + a \cdot 0.$$

Adicionando $-(a \cdot 0)$ a ambos os lados da igualdade e utilizando as propriedades 5, 4, 3 e

2, obtemos:

$$\begin{aligned} 0 &= -(a\cdot 0) + a\cdot 0 = -(a\cdot 0) + (a\cdot 0 + a\cdot 0) \\ &= (-(a\cdot 0) + a\cdot 0) + a\cdot 0 = 0 + a\cdot 0 \\ &= a\cdot 0. \end{aligned}$$

□

Teorema 2.2. *A adição é compatível e cancelativa em relação à igualdade:*

$$\forall\ a, b, c \in \mathbb{Z}, a = b \Leftrightarrow a + c = b + c.$$

Demonstração. A implicação

$$a = b \Rightarrow a + c = b + c,$$

decorre diretamente do fato de a adição ser uma operação bem definida. Suponha que $a + c = b + c$. Somando $(-c)$ a ambos os lados, obtemos o desejado (HEFEZ, 2016). □

Ordenação dos inteiros

A noção de ordem, amplamente fundamentada na teoria dos números, constitui um elemento essencial para a definição e aplicação das desigualdades matemáticas. Por meio dessas relações, torna-se possível estabelecer comparações entre elementos numéricos, identificando, com precisão lógica, quais são maiores, menores ou iguais dentro de um conjunto numérico. No conjunto dos números inteiros, tal ordenação está bem definida e obedece a propriedades rigorosas que sustentam a estrutura algébrica desse sistema. No âmbito da computação, particularmente na linguagem de programação Python, essas operações são incorporadas de forma nativa, permitindo a manipulação e comparação eficiente de dados numéricos.

Para uma melhor compreensão das relações de desigualdade no conjunto dos números inteiros, apresenta-se a seguir um conjunto de princípios e teoremas que fundamentam as operações mencionadas.

Tricotomia

A tricotomia é um princípio fundamental da teoria dos números e da lógica matemática. Esse princípio assegura a natureza bem definida da relação de ordem entre os números, permitindo uma comparação rigorosa e inequívoca de quaisquer dois elementos.

No contexto da construção de algoritmos e da programação de computadores, a propriedade da tricotomia é essencial. Ela fundamenta o desenvolvimento de estruturas de decisão, tais como comandos condicionais `(if-else, switch-case)` e algoritmos de ordenação e busca, que dependem da comparação entre valores para determinar o fluxo de

execução. A certeza de que uma única relação será verdadeira entre dois números permite a formulação de algoritmos corretos e eficientes, além de garantir a robustez lógica dos programas.

Assim, a tricotomia não apenas reforça a compreensão teórica da ordem numérica, mas também desempenha papel central na prática computacional, fornecendo uma base lógica sólida para a construção de algoritmos que envolvem operações de comparação.

Segue a formalização de tal princípio: Dados $a, b \in \mathbb{Z}$, uma, e apenas uma, das seguintes possibilidades é verificada:

- $a = b$;
- $b - a \in \mathbb{N}$;
- $-(b - a) = a - b \in \mathbb{N}$.

Ou seja, apenas uma das seguintes condições é verificada:

- $a = b$;
- $a < b$;
- $b < a$.

A notação $b > a$ significa que b é maior do que a. Como $a - 0 = a$, das definições $a > 0$ se, e somente se, $a \in \mathbb{N}$. Portanto,

$$\mathbb{N} = \{x \in \mathbb{Z}; x > 0\} \quad \text{e} \quad -\mathbb{N} = \{x \in \mathbb{Z}; x < 0\}.$$

Daí decorre que $a > 0$, se, e somente se, $-a < 0$.

Princípio da Boa Ordenação

No conjunto dos números inteiros, o Princípio da Boa Ordenação (PBO) estabelece que todo subconjunto não vazio de números inteiros positivos possui um elemento mínimo, isto é, um elemento que é inferior ou igual a todos os demais elementos do subconjunto. Esse princípio assegura a existência de um limite inferior para subconjuntos de inteiros positivos, constituindo uma ferramenta teórica fundamental para a análise e demonstração de propriedades em matemática e ciência da computação.

Na construção de algoritmos, o Princípio da Boa Ordenação desempenha um papel crucial na definição de propriedades de limite, sendo amplamente utilizado na demonstração da corretude, da terminação de algoritmos e em procedimentos de otimização. Sua aplicação garante que processos iterativos ou recursivos, sob condições adequadas, alcancem um ponto de parada bem definido.

A seguir, apresenta-se a definição formal do Princípio da Boa Ordenação, mas antes precisamos lembrar que um subconjunto S de $\mathbb{Z}$ é dito ser limitado inferiormente, se existir $c \in \mathbb{Z}$ tal que $c \leq x$ para todo $x \in S$. Diz-se que $a \in S$ é um menor elemento de S se $a \leq x$ para todo $x \in S$. Isto posto:

Axioma 2.1 (Princípio da Boa Ordenação)**.** Se S é um subconjunto não vazio de Z e limitado inferiormente, então S possui um menor elemento.

Ainda no contexto de ordenação dos inteiros, a seguir apresento princípio que fornece garantias de contagem e asseguram a uma máquina a ausência de elementos fora do contexto de números inteiros. Por exemplo, entre dois números inteiros consecutivos, é impossível haver outro número inteiro intermediário. Sem perda de generalidade, demonstraremos essa propriedade por meio do seguinte teorema.

Teorema 2.3. *Não existe nenhum número inteiro n tal que $0 < n < 1$.*

Demonstração. Suponha por absurdo que exista n com essa propriedade. Logo o conjunto

$$S = \{x \in \mathbb{Z}; 0 < x < 1\},$$

é não vazio, além de ser limitado inferiormente. Portanto, S possui um menor elemento a, com

$$0 < a < 1.$$

Multiplicando esta última desigualdade por a, obtemos

$$0 < a^2 < a < 1,$$

uma contradição. Portanto, S é vazio (HEFEZ, 2016). □

Aritmética em Python

Em Python, as propriedades da aritmética são implementadas por meio de operadores e funções previamente definidos pela linguagem, os quais possibilitam a execução de operações matemáticas de forma precisa e eficiente. Tais operadores são fundamentais para a construção de algoritmos que envolvem cálculos numéricos, sendo amplamente utilizados tanto em aplicações básicas quanto em contextos científicos e de engenharia (ROJAS; KOSTIN, 2018).

Além dos operadores básicos, o Python oferece a biblioteca padrão `math`, que amplia significativamente as capacidades da linguagem no domínio matemático. Essa biblioteca inclui funções para cálculos de raiz quadrada (`math.sqrt()`), exponenciação (`math.pow()`), fatorial (`math.factorial()`), e funções trigonométricas usuais como seno (`math.sin()`), cosseno (`math.cos()`), entre outras.

A utilização combinada de operadores aritméticos e funções matemáticas permite ao programador a formulação de expressões algébricas complexas e a resolução de problemas computacionais diversos, constituindo, portanto, um pilar fundamental no desenvolvimento de aplicações que demandam raciocínio lógico-matemático.

Segue abaixo tabela contendo os operadores aritméticos nativos da linguagem de programação Python:

Tabela 1 – Operações aritméticas em Python

OPERAÇÃO	OPERANDO	EXEMPLO	RESULTADO
Atribuição	=	$a = 5$	5
Adição	+	$4 + 6$	10
Subtração	–	$7 - 3$	4
Multiplicação	*	$3 * 5$	15
Divisão	/	$8/2$	4
Exponenciação	**	$2 * *3$	8
Divisão inteira	//	$11//5$	2
Resto da divisão	%	$7\%2$	1

Fonte: Elaborado pelo autor, 2025.

Exemplo 2.14. Segue modelo de algoritmo envolvendo operadores aritméticos:

```
a = 12          # Atribui o valor 12 a variável a;
b = 5           # Atribui o valor 5 a variável b;
a + b           # Entrada operação adição de variáveis.
17              # Saída.
2 * 8 - 3**2    # Entrada expressão aritmética.
7               # Saída.
```

2.2.3 Estrutura de decisão

No desenvolvimento de programas computacionais, é recorrente a necessidade de realizar tomadas de decisão com base em determinadas condições previamente estabelecidas. Esse processo é implementado por meio das chamadas estruturas condicionais, que permitem ao algoritmo adotar diferentes caminhos de execução conforme a avaliação de expressões lógicas. Essas expressões usam operadores lógicos relacionados com o valor booleano `(True ou False)` (MATHEUS, 2016).

A linguagem de programação tem sua estrutura profundamente alicerçada no raciocínio lógico-matemático, elemento fundamental para a formulação e desenvolvimento de algoritmos. Assim como os conceitos de aritmética são empregados para a realização de operações numéricas, os princípios da lógica clássica, também conhecida como lógica aristotélica, são amplamente utilizados na construção de instruções baseadas em decisões.

Sendo assim, com o objetivo de proporcionar um melhor aproveitamento das funcionalidades relacionadas às estruturas de decisão, abordam-se, neste trabalho, os conceitos fundamentais da lógica matemática, com a finalidade de oferecer o suporte teórico necessário à compreensão da lógica de programação.

2.2.4 Lógica

O estudo da lógica teve início por volta do século IV a.C., com os trabalhos do filósofo grego Aristóteles, considerado o precursor da lógica formal. A partir de então, diversos matemáticos e filósofos contribuíram para o desenvolvimento e a consolidação dessa área do conhecimento. Entre os principais nomes que se destacaram nesse processo, pode-se citar Gottfried Wilhelm Leibniz, Leonhard Euler, Augustus De Morgan, George Boole, Bertrand Russell, entre outros, cujas contribuições foram fundamentais para a formação da lógica moderna, especialmente no que diz respeito à sua formalização e aplicação no contexto matemático e computacional (MAGALHÃES, 2011).

Nos dias atuais, o pensamento lógico tornou-se uma ferramenta extremamente valiosa no âmbito da programação. O pensamento computacional tem se beneficiado significativamente dessa base teórica fornecida pela lógica matemática, a qual contribui de forma decisiva para o desenvolvimento de algoritmos e para a resolução eficiente de problemas computacionais.

Proposição

No estudo da lógica, torna-se essencial a compreensão do conceito de proposição lógica e de seu papel na estruturação do raciocínio. A lógica, enquanto linguagem formal de caráter matemático, difere da linguagem natural ao exigir precisão e objetividade na formulação das ideias. Enquanto, na língua portuguesa, uma frase corresponde a um conjunto de palavras que exprimem um pensamento, no contexto lógico, uma proposição é definida como qualquer enunciado declarativo ao qual se pode atribuir, de maneira exclusiva, um valor de verdade, verdadeiro ou falso, ainda que, em determinadas circunstâncias, não seja possível verificar empiricamente tal valor.

Por convenção usaremos as letras *p, q, r, s, ...* para representar proposições. Sendo simples quando há apenas uma e composta quando combinamos duas ou mais com auxílio de conectivos (ALENCAR FILHO, 2002).

Exemplo 2.15. “O planeta terra faz parte do sistema solar", essa frase pode assumir o valor lógico verdadeiro ou falso. Já a expressão ‘Qual é o seu nome?’ não pode ser avaliada pelo critério de verdadeira ou falsa, não possui valor lógico, logo não é uma proposição.

Princípios

Conforme já exposto, uma proposição, ou sentença lógica, é definida como uma expressão dotada de valor lógico, isto é, pode ser classificada como verdadeira ou falsa. Essa estrutura lógica fundamenta-se em dois princípios clássicos da lógica formal. O primeiro é o Princípio da Não Contradição, segundo o qual nenhuma proposição pode ser simultaneamente verdadeira e falsa. O segundo é o Princípio do Terceiro Excluído, que estabelece que toda proposição deve necessariamente assumir um único valor de verdade: ou é verdadeira, ou é falsa, não havendo uma terceira alternativa admissível. Esses princípios são pilares essenciais na construção do raciocínio lógico e na formalização de argumentos dedutivos (ALENCAR FILHO, 2002).

Conectivos

Nas situações em que o valor lógico de uma estrutura depende de mais de uma proposição, proposição composta, é necessário o uso de conectivos lógicos, elementos responsáveis por articular proposições simples em uma unidade lógica maior. Esses conectivos permitem a construção de uma sentença composta cujo valor de verdade está logicamente determinado pelos valores de verdade das proposições que a integram. Assim, a análise da veracidade da proposição composta requer a consideração conjunta das sentenças que a compõem e da natureza do conectivo empregado.

Desses conectivos lógicos os principais são: conjunção($\wedge$), disjunção($\vee$), disjunção mutuamente exclusiva($\underline{\vee}$), condicional($\rightarrow$), bicondicional($\leftrightarrow$) e negação($\neg$). Cada um desses operadores possui uma função específica na estrutura lógica, determinando o valor de verdade da proposição composta com base nos valores de verdade das proposições simples que os compõem. A correta utilização desses conectivos é fundamental para a formalização de argumentos e para a análise rigorosa de inferências no campo da lógica matemática.

A fim de promover uma compreensão mais precisa e rigorosa do funcionamento da lógica proposicional, faz-se necessário examinar, de forma detalhada, cada um dos principais conectivos lógicos e suas respectivas propriedades.

Negação

No âmbito da lógica proposicional, quando se pretende atribuir a uma proposição o valor lógico contrário ao originalmente atribuído, recorre-se ao conectivo de negação, simbolizado por ($\neg$).

Trata-se de um operador lógico unário que incide sobre uma única proposição, invertendo seu valor de verdade: se a proposição for verdadeira, sua negação será falsa; se for falsa, sua negação será verdadeira. A negação constitui um dos fundamentos da estrutura lógica formal, sendo amplamente utilizada na construção de enunciados contraditórios, na

análise de inferências e na formulação de provas por contraposição ou redução ao absurdo (MAGALHÃES, 2011).

Exemplo 2.16. Segue aqui exemplo de sentença lógica e aplicação do conectivo de negação ($\neg$):

Seja a proposição p: Eu sei nadar, $\neg p$: Eu não sei nadar.

Observe que as proposições p e $\neg p$ sempre possuem valores lógicos diferentes. Isto é, quando p é verdadeiro, $\neg p$ é falso e vice-versa. Podemos verificar isso aqui na tabela verdade negação:

Tabela 2 – Tabela verdade da negação

p	$\neg p$
V	F
F	V

Fonte: Elaborado pelo autor, 2025.

Conjunção ($\wedge$) e Disjunção ($\vee$)

A teoria dos conjuntos constitui um ramo fundamental da matemática, responsável por fornecer uma linguagem simbólica rigorosa e amplamente utilizada na formulação e na fundamentação de diversos conceitos matemáticos. Desenvolvida inicialmente pelo matemático Georg Cantor no final do século XIX, essa teoria estabeleceu as bases para a estruturação lógica da matemática moderna, sendo essencial para áreas como a lógica, a análise, a álgebra e a topologia. Sua importância reside, sobretudo, na capacidade de representar e manipular coleções de objetos de maneira formal, contribuindo significativamente para a precisão e a consistência do discurso matemático.

A teoria dos conjuntos estabelece conceitos operacionais fundamentais para o tratamento formal de coleções de elementos, destacando-se, entre eles, os conceitos de união e interseção de conjuntos. Seja A e B dois conjuntos quaisquer, define-se a união $A \cup B$ como o conjunto de todos os elementos que pertencem a A, a B, ou a ambos simultaneamente. Em termos formais, tem-se:

$$A \cup B = \{x \in U | x \in A \text{ ou } x \in B\},$$

onde U representa o Universo considerado.

Analogamente, a interseção entre os conjuntos A e B, denotada por $A \cap B$, é definida como o conjunto de todos os elementos que pertencem simultaneamente a A e a B, ou seja:

$$A \cap B = \{x \in U | x \in A \text{ e } x \in B\}.$$

No contexto da lógica proposicional, as proposições compostas apresentam uma correspondência conceitual direta com determinadas operações da teoria dos conjuntos. Especificamente os conceitos de união e interseção de conjuntos estão, respectivamente, associados aos conectivos lógicos de disjunção e conjunção.

A disjunção lógica, representada pelo símbolo ($\vee$), estabelece que uma proposição composta $p \vee q$ é verdadeira se ao menos uma das proposições p ou q for verdadeira. Tal definição guarda analogia com a união de conjuntos $A \cup B$. Por sua vez, a conjunção lógica, indicada por ($\wedge$), define a proposição composta $p \wedge q$ como verdadeira somente quando ambas as proposições são simultaneamente verdadeiras, o que se assemelha à interseção de conjuntos $A \cap B$.

Essa analogia evidencia a profunda inter-relação entre a lógica formal e a teoria dos conjuntos, permitindo a formulação de estruturas matemáticas consistentes e sistemáticas. Além disso, tal relação contribui para a compreensão e aplicação de conceitos fundamentais em diversas áreas do conhecimento, como a matemática, a computação, a filosofia e a linguística formal.

Exemplo 2.17. Nesse escopo, evidencia-se a utilização dos conectivos lógicos de disjunção e conjunção na formação de proposições compostas. A partir da aplicação da tabela-verdade, é possível determinar sistematicamente os valores lógicos resultantes dessas proposições, com base nas diferentes combinações de verdade e falsidade atribuídas às proposições atômicas que as constituem.

Sejam as proposições p e q tais que:

p : Hoje é domingo.

q : O idioma oficial do Brasil é o Português.

Tabela 3 – Tabela verdade da conjunção e disjunção

p	q	$p \wedge q$	$p \vee q$
V	V	V	V
V	F	F	V
F	V	F	V
F	F	F	F

Fonte: Elaborado pelo autor, 2025.

Além disso, é importante ressaltar que o uso desses conectivos lógicos possibilita a realização das operações do cálculo proposicional (ou cálculo sentencial), ramo da lógica formal que se ocupa, entre outras coisas, da análise de proposições compostas resultantes de operações lógicas, bem como da determinação de seus valores lógicos.

Disjunção mutuamente exclusiva ($\underline{\vee}$)

A disjunção exclusiva, no âmbito da lógica proposicional, representa uma operação lógica que expressa a veracidade de exatamente uma entre duas proposições. Distingue-se da disjunção inclusiva (o "ou"convencional), pois, enquanto esta admite a possibilidade de ambas as proposições serem simultaneamente verdadeiras, a disjunção exclusiva restringe-se ao caso em que apenas uma das proposições possui valor lógico verdadeiro. Assim, caso ambas as proposições sejam verdadeiras ou ambas sejam falsas, o valor lógico da proposição composta será falso.

Do ponto de vista da Teoria dos Conjuntos, essa operação pode ser relacionada à noção de diferença simétrica, que consiste no conjunto dos elementos que pertencem a um ou a outro conjunto, mas não simultaneamente a ambos. Nesse contexto, a disjunção exclusiva reflete uma condição em que não há interseção entre os elementos dos conjuntos considerados, enfatizando a exclusividade da pertença a apenas um dos conjuntos (MAGALHÃES, 2011).

A operacionalização do conectivo disjunção exclusiva pode ser ilustrada por meio de um exemplo específico, cuja estrutura lógica é formalizada e analisada com base no cálculo proposicional. Nesse sentido, a tabela-verdade constitui um instrumento fundamental para a visualização dos possíveis valores lógicos assumidos pelas proposições simples e pela proposição composta que resulta da aplicação do referido conectivo.

Exemplo 2.18. Considere, para fins de exemplificação, as seguintes proposições:

p : A matemática é uma ciência exata.

q : Python é uma linguagem de máquina.

$p \underline{\vee} q$: Ou a matemática é uma ciência exata ou Python é uma linguagem de máquina.

Tabela 4 – Tabela verdade da disjunção exclusiva

p	q	$p \underline{\vee} q$
V	V	F
V	F	V
F	V	V
F	F	F

Fonte: Elaborado pelo autor, 2025.

Conectivo condicional ($\rightarrow$)

Na lógica formal, uma das formas fundamentais de composição de sentenças é por meio da condicional, também denominada implicação lógica. Tal estrutura estabelece uma relação de dependência entre duas proposições, em que o valor lógico de uma delas determina ou influencia o valor lógico da outra. A condicional é expressa na forma “Se P, então Q”, simbolicamente representada por $(P \rightarrow Q)$, onde:

P é denominada antecedente ou hipótese;

Q é denominada consequente ou conclusão.

Sejam p e q proposições quaisquer, com o uso do conectivo condicional, p implica q, ou seja, se p então q (MAGALHÃES, 2011).

Do ponto de vista formal, essa proposição composta assume valor falso unicamente quando a proposição antecedente (p) é verdadeira e a consequente (q) é falsa; em todos os demais casos, a implicação é considerada verdadeira. Importa ressaltar que, na lógica formal, tal conectivo não implica necessariamente uma relação de causalidade, mas sim uma relação condicional baseada exclusivamente nos valores de verdade atribuídos às proposições envolvidas.

Esse princípio lógico encontra aplicação direta no campo da computação, particularmente nas linguagens de programação, por meio das chamadas estruturas de decisão. Tais estruturas permitem que algoritmos avaliem condições lógicas e executem determinados blocos de instruções com base no resultado dessa avaliação, conferindo ao programa a capacidade de tomar decisões dinâmicas em tempo de execução.

Exemplo 2.19. Para a verificação da construção de proposições compostas a partir de duas proposições simples, interligadas por meio do conectivo condicional, a seguir, apresenta-se a tabela-verdade correspondente ao condicional material:

p : 'Você vai embora'.

q : 'Eu vou chorar'.

$p \rightarrow q$: 'Se você for embora então eu vou chorar'.

Tabela 5 – Tabela verdade do condicional

p	q	$p \rightarrow q$
V	V	V
V	F	F
F	V	V
F	F	V

Fonte: Elaborado pelo autor, 2025.

Ainda no contexto do conectivo condicional, representado pela proposição $(p \rightarrow q)$, identificam-se três formas notáveis de reconfiguração lógica: a contrapositiva, a inversa e a recíproca. Tais variações decorrem da manipulação das proposições simples envolvidas no condicional e são de fundamental importância para a análise de equivalência lógica e para a condução de demonstrações dedutivas rigorosas.

A contrapositiva de uma proposição condicional consiste na negação do consequente e do antecedente, com a inversão de suas posições, resultando na proposição $(\neg q \rightarrow \neg p)$, lida como "se não q, então não p". Do ponto de vista lógico, a contrapositiva é logicamente equivalente à proposição condicional original, ou seja, ambas possuem os mesmos valores de verdade em todas as combinações possíveis dos valores das proposições envolvidas.

No âmbito da lógica proposicional, a recíproca de uma proposição condicional é obtida por meio da inversão da ordem entre a hipótese e a consequência, isto é, da proposição do tipo 'se P, então Q' (P→Q), obtém-se 'se Q, então P' $(Q \to P)$. Ressalta-se, contudo, que a proposição recíproca não é logicamente equivalente à condicional original, uma vez que seus valores de verdade podem divergir em determinadas situações.

Já a inversa de uma proposição condicional é construída por meio da negação simultânea tanto da hipótese quanto da consequência da proposição original. Assim, a inversa da condicional 'se P, então Q' $(P \to Q)$ é expressa como 'se não P, então não Q' $(\neg P \to \neg Q)$. Tal como a recíproca, a inversa também não é logicamente equivalente à proposição condicional inicial, pois pode assumir valores de verdade distintos.

Exemplo 2.20. considere a proposição condicional: 'Se está chovendo, abro o guarda-chuva'. A contrapositiva pode ser escrita como 'Se não abro o guarda-chuva, então não esta chovendo' , a recíproca pode ser escrita como 'Se abro o guarda-chuva, então está chovendo' e a inversa como 'Se não está chovendo, não abro o guarda-chuva'.

Condições necessárias e condições suficientes

Na análise de proposições compostas que envolvem o conectivo condicional, é essencial a compreensão precisa dos conceitos de condição suficiente e condição necessária. Dada uma proposição condicional da forma 'se P, então Q' $(P \to Q)$, afirma-se que P constitui uma condição suficiente para Q, uma vez que a ocorrência de P implica logicamente a ocorrência de Q. Por sua vez, Q é tida como uma condição necessária para P, na medida em que a validade de P depende da veracidade de Q. A distinção entre esses conceitos desempenha um papel crucial na interpretação lógica de enunciados, na construção de argumentos válidos e na fundamentação de inferências dedutivas.

Conectivo Bicondicional $(p \leftrightarrow q)$

Em determinados contextos, entretanto, pode-se estabelecer uma relação de equivalência lógica entre duas proposições, de modo que ambas implicam uma à outra. Quando isso ocorre, utiliza-se o conectivo bicondicional, expresso por duas proposicões p e q, a proposição $p \leftrightarrow q$ indica qual é a relacão de equivalência entre p e q no seguinte sentido: $p \leftrightarrow q$ será verdadeira se p e q forem ambas verdadeiras ou ambas falsas; e será falsa caso contrário.

Na lógica formal, a condição de equivalência expressa por meio do conectivo bicondicional estabelece uma relação simétrica entre duas proposições, de modo que a validade da proposição composta $(P \leftrightarrow Q)$ requer que tanto a implicação $(P \to Q)$ quanto sua recíproca $Q \to P$ sejam logicamente verdadeiras. Essa estrutura assegura que as proposições envolvidas compartilham o mesmo valor de verdade, caracterizando-se, assim, como logicamente equivalentes. O uso do bicondicional constitui um recurso fundamental nas

demonstrações formais de proposições e teoremas, uma vez que permite a construção de argumentos nos quais a correspondência bidirecional entre os enunciados é essencial para a validade lógica da inferência.

Exemplo 2.21. Apresenta-se, a seguir, um exemplo de como duas proposições podem ser combinadas por meio do conectivo lógico bicondicional, formando uma proposição composta que expressa uma relação de equivalência entre elas.

Sejam as proposições p e q tais que:

p: 'Vou ao cinema.'

q: 'Está chovendo.'

'Vou ao cinema se e somente se não estiver chovendo' pode ser escrita como $(p \leftrightarrow q)$.

Tabela 6 – Tabela verdade do bicondicional

p	q	$p \leftrightarrow q$
V	V	V
V	F	F
F	V	F
F	F	V

Fonte: Elaborado pelo autor, 2025.

É importante destacar que os conceitos lógicos abordados neste trabalho não esgotam a amplitude teórica que os envolve, mas constituem definições introdutórias fundamentais para a compreensão do funcionamento dos dados booleanos na linguagem de programação Python. Tais conceitos revelam-se particularmente relevantes na análise e implementação de estruturas lógicas de controle, como decisões condicionais e laços de repetição. Diante desse panorama, prossegue-se com a exposição da aplicação da lógica na linguagem Python.

Lógica em Python

Assim como ocorre em outras linguagens de programação, a linguagem Python fundamenta-se em princípios da lógica matemática. Dentre esses princípios, destaca-se a lógica booleana, cujo nome homenageia George Boole, matemático e filósofo britânico do século XIX, responsável pela formulação da álgebra booleana, como já abordado.

Em Python, os valores booleanos, True e False, são utilizados em estruturas de decisão, as quais possibilitam a definição de comportamentos condicionais dentro do código. Dessa forma, é possível estabelecer diferentes fluxos de execução a partir da avaliação lógica de expressões, o que torna a lógica booleana um elemento central na implementação de algoritmos que demandam tomada de decisão, dada tamanha importância, apresento a seguir os principais operadores lógicos em Python.

Exemplo 2.22. No exemplo a seguir tem operações lógicas em Python contendo algumas proposições e o valor lógico fornecido pelo programa:

```
7 == 9      # Entrada, verifica a igualdade.
False       # Saída.
1 != 2      # Entrada, verifica a diferença.
True        # Saída.
4 > 6       # Entrada, verifica se maior que.
False       # Saída.
```

Comparação

A tabela adiante, contém o conjunto de operadores associados a atividade de comparar dados com o fim de imprimir o valor lógico.

Tabela 7 – Operadores de comparação

Operador	Função	Exemplo
==	Verifica a igualdade	7 == 9
! =	Verifica a diferença	1! = 2
>	Maior que	4 > 6
<	Menor que	5 < 2
>=	Maior ou igual	6 >= 4
<=	Menor ou igual	5 <= 7

Fonte: Elaborado pelo autor, 2025.

Conectivos

Na lógica matemática, os conectivos lógicos são operadores utilizados com a finalidade de formar proposições compostas a partir de proposições simples. Essas proposições compostas mantêm uma estrutura lógica que lhes confere um valor de verdade, verdadeiro `(V)` ou falso `(F)`, determinado a partir dos valores das proposições que as compõem e do tipo de conectivo empregado.

Na linguagem de programação Python, os operadores lógicos mantêm a mesma fundamentação conceitual da lógica matemática clássica. Sua principal função consiste em estabelecer relações lógicas entre expressões booleanas, permitindo a construção de estruturas condicionais e algoritmos mais complexos e funcionais.

Esses conectivos são amplamente empregados na formulação de expressões compostas, especialmente em estruturas de controle como if, while, for, entre outras, possibilitando a tomada de decisões com base em múltiplas condições simultâneas. Os principais operadores lógicos em Python são os seguintes:

- **and**: corresponde à conjunção lógica `(e)` e retorna verdadeiro `(True)` somente quando ambas as expressões avaliadas forem verdadeiras;
- **or**: representa a disjunção lógica `(ou)`, retornando verdadeiro se ao menos uma das expressões for verdadeira;
- **not**: realiza a negação lógica `( ¬ )`, invertendo o valor lógico de uma dada expressão.
- **in**: Verifica se uma variável pertence a uma sequência específica, tipo uma lista, retornando verdadeiro se sim.

A correta aplicação desses operadores é essencial para o desenvolvimento de algoritmos eficientes, uma vez que permite o encadeamento lógico de condições e a implementação de fluxos de controle precisos, que são fundamentais tanto em aplicações simples quanto em sistemas computacionais de maior complexidade.

Exemplo 2.23. Para entender a aplicação prática dos conectivos, abaixo apresento código contendo-os:

```
a = 6              # Atribuição de variável;
b = 7              # Atribuição de variável;
a > 3 and b < 5    # Entrada.
True               # Saída.
5 in [1,3,5,7,9]   # Entrada.
True               # Saída.
```

Condicional if

É muito tradicional a matemática conter raízes lógicas altamente calcadas em lógica aristotélica, que são a base muitas vezes da linguagem humana, aproveitada para deduções lógicas simbólicas. Na lógica formal, com a estrutura condicional no formato $p \rightarrow q$, lemos "se p, então q". Isso significa que a proposição p é uma condição para que a proposição q aconteça.

Nessa esteira, a linguagem Python possui uma forma de codificar essa ideia lógico-matemática. Nela, podemos estabelecer condições para que um comando, ou vários comandos seguintes, funcionem, o que corresponderia às sentenças ou teses dependentes do acontecimento de hipóteses.

Essa estrutura formal é dada da seguinte forma:

```
if "condição booleana":
    "bloco a ser executado,
```

```
    caso a condição booleana acima
    seja satisfeita."
```

Pode-se ver que a estrutura utilizada por Python se assemelha muito à nossa linguagem natural. Essa é a característica que chamamos de linguagem de alto nível.

Em suma, na **instrução `if`**, temos uma expressão que pode ter valor lógico True ou False (booleanos), chamada teste condicional. Esses valores, em Python, têm o objetivo de criar uma estrutura de decisão, ou seja, se o resultado for True, Python executará o código após a instrução `if`; se for False, o código será ignorado (MATHEUS, 2016).

Exemplo 2.24. Temos aqui uma estrutura de decisão usando a instrução if em que o programa solicita um valor e apresenta positivo caso valor lógico verdade:

```
# Atribuição de variável real através do comando input;
x = float(input('insira um número: '))
if x > 0:               # Condição if
  print('positivo')     # Imprime positivo
  insira um número: 3   # Entrada.
positivo                # Saída.
```

Instruções if - else

A execução condicional constitui um recurso fundamental na construção de algoritmos, especialmente no que tange ao desenvolvimento de estruturas de decisão. No entanto, a utilização de uma única condição pode, em determinadas situações, limitar a capacidade do algoritmo de responder adequadamente às diversas possibilidades de fluxo lógico. Nesse cenário, a instrução else desempenha um papel essencial, ao permitir a definição de um caminho alternativo de execução para os casos em que a condição inicialmente estabelecida não é satisfeita. Tal mecanismo contribui para a elaboração de algoritmos mais dinâmicos, robustos e adaptáveis, promovendo uma maior eficiência na resolução de problemas computacionais.

A instrução else configura-se como um complemento lógico à condição inicialmente estabelecida pela instrução if. Considerando um domínio de possibilidades no qual determinadas condições satisfazem o critério definido pelo if, o else atua sobre o conjunto complementar dessas possibilidades, ou seja, sobre os casos em que a condição avaliada não é verdadeira. Em termos de lógica computacional, essa estrutura permite a delimitação clara entre os cenários que atendem à condição primária e aqueles que não a satisfazem, garantindo, assim, uma cobertura mais abrangente das possíveis situações que podem ocorrer durante a execução de um algoritmo. A utilização conjunta de if e else

é, portanto, essencial para a construção de estruturas de decisão mais robustas, capazes de oferecer respostas adequadas a diferentes contextos de execução.

Exemplo 2.25. Temos, neste caso, uma estrutura de decisão utilizando a instrução if-else, na qual o programa solicita um valor ao usuário e, com base na análise lógica dessa entrada, exibe a mensagem "positivo" caso a condição seja verdadeira, ou "negativo" caso a condição seja falsa.

```
 # Atribuição de variável real através do comando input;
x = float(input('insira um número: '))
if x > 0:               # Condição if;
  print('positivo')     # Imprime positivo;
else:                   # Condição alternativa;
  print('negativo')     # Imprime negativo;
  insira um número: -4  # Entrada.
negativo                # Saída.
```

Instruções if - elif - else

No âmbito das estruturas de decisão, há situações em que a simples utilização da construção if-else não se mostra suficiente para contemplar todas as possibilidades lógicas envolvidas em um determinado problema. Visando à elaboração de algoritmos que reproduzam de forma mais fidedigna a complexidade do processo decisório humano, introduz-se a instrução elif (abreviação de else if). Essa instrução possibilita o encadeamento de múltiplas condições, viabilizando a construção de estruturas condicionais com mais de dois ramos de execução.

Por meio desse recurso, o algoritmo é capaz de avaliar, de forma sequencial, diferentes condições lógicas e executar o bloco de instruções correspondente à primeira condição verdadeira encontrada. Tal abordagem amplia significativamente a capacidade de tomada de decisão dos algoritmos, tornando-os mais flexíveis e adaptáveis às variáveis de entrada.

Exemplo 2.26. Temos, neste caso, uma estrutura de decisão utilizando a instrução if- elif - else, na qual o programa solicita um valor ao usuário e, com base na análise lógica dessa entrada, exibe a mensagem "positivo" caso a condição seja verdadeira, "negativo" caso a condição seja falsa ou "x é igual a zero" se não ocorrer nenhuma das duas condições:

```
 # Atribuição de variável real através do comando input;
x = float(input('insira um número: '))
if x > 0:                   # Condição if;
  print('positivo')         # Imprime positivo;
```

```
elif x == 0:                # Condição elif;
  print(‘x é igual a zero’) # Imprime x é igual a zero;
else:                       # Condição alternativa;
  print(‘negativo’)         # Imprime negativo;
insira um número: 0         # Entrada.
x é igual a zero            # Saída.
```

2.2.5 Estruturas de repetição, ou LOOP

Assim como as estruturas condicionais, as estruturas de repetição, ou loops, fundamentam-se nos princípios da lógica Booleana para controlar o fluxo de execução de um algoritmo. Tais estruturas são projetadas para repetir uma determinada ação enquanto uma condição lógica permanecer verdadeira. A repetição ocorre de forma automática, sem necessidade de intervenção externa, até que uma condição impeditiva seja satisfeita, interrompendo a execução do ciclo.

No contexto da linguagem de programação Python, destacam-se as instruções ‘while’ e ‘for’ como mecanismos principais para a construção de algoritmos baseados em repetição. A instrução **while** é utilizada quando se deseja manter a execução de um bloco de código enquanto determinada condição for atendida, sendo especialmente útil em situações em que o número de iterações não é previamente conhecido.

Por outro lado, a instrução **for** permite a iteração sobre elementos de uma sequência, como listas, tuplas ou intervalos numéricos, oferecendo um controle mais explícito e direto sobre a quantidade de repetições. Ambas as estruturas são essenciais para a elaboração de algoritmos eficientes, dinâmicos e capazes de lidar com tarefas repetitivas de forma sistemática.

While

A estrutura de repetição while constitui um mecanismo de controle de fluxo amplamente utilizado em linguagens de programação, cuja principal característica reside na repetição de um bloco de instruções enquanto uma determinada condição lógica for avaliada como verdadeira. Trata-se, portanto, de um laço de repetição pré-condicional, isto é, a condição é verificada antes da execução do bloco de comandos.

Durante a execução do laço, a expressão condicional é continuamente reavaliada ao final de cada iteração. Caso a condição permaneça verdadeira, o ciclo se repete; no entanto, se a condição for avaliada como falsa, a execução do laço é imediatamente interrompida, e o controle do programa é transferido para a instrução subsequente ao bloco while.

Essa estrutura é particularmente útil em situações nas quais o número de repetições não é previamente conhecido, sendo determinado dinamicamente com base em eventos

ou valores computados em tempo de execução. Contudo, é fundamental garantir que a condição eventualmente se torne falsa, a fim de evitar a ocorrência de laços infinitos, que podem comprometer a execução do programa.

Para evitar a ocorrência de laços infinitos durante a execução de estruturas de repetição do tipo while em Python, o programador pode empregar estratégias de controle que asseguram a terminação adequada do laço. Dentre essas estratégias, destacam-se dois mecanismos amplamente utilizados: o uso de um contador de repetições e a utilização de uma marca de controle, comumente denominada flag.

1. O contador de repetições consiste em uma variável numérica que é inicializada antes do início do laço e atualizada (geralmente incrementada) a cada iteração. A condição de continuidade do while é estabelecida de forma que o laço se encerre automaticamente quando o contador atingir um valor previamente definido. Essa abordagem é especialmente apropriada quando o número de repetições desejado é conhecido ou pode ser previamente determinado.

2. Por outro lado, a utilização de uma marca de controle (`flag`) se mostra eficaz em situações em que a quantidade de repetições depende de condições dinâmicas ou imprevisíveis, como a entrada de dados pelo usuário ou o resultado de alguma verificação lógica. Nesse caso, define-se uma variável booleana que representa o estado de continuidade do laço. Enquanto essa flag estiver com valor lógico verdadeiro (`True`), o laço continuará sendo executado. Quando a condição de parada for atingida, essa variável é alterada para falso (`False`), provocando a interrupção do laço (MATHEUS, 2016).

A estrutura de comando é basicamente dividida na condição e o bloco de comandos tendo a opção de usar a instrução else para interromper o loop (ROJAS; KOSTIN, 2018).

Exemplo 2.27. Apresento aqui uma estrutura de repetição do tipo while, utilizando o método do contador para encerrar o loop.

```
soma = 0                    # Soma dos valores;
i = 0                       # Quantidade de valores lidos;
while i<5:                # Condição de continuidade do loop;
      i+ = 1                # Adicione 1 a qtd de valores lidos;
      x = float(input('digite um valor: '))
      soma+ = x             # Adicione o valor lido a soma
      media = soma/5        # Atribuição da média;
      # Imprime valor da média junto com mensagem;
      print('a média dos valores é {:5.2f}'.format(media))
```

Na estrutura de repetição apresentada acima, temos um algoritmo construído com o objetivo de calcular a média aritmética de 5 valores. O número 5 representa a quantidade total de execuções, após as quais o loop é interrompido.
Média Aritmética

Exemplo 2.28. Apresento aqui uma estrutura de repetição do tipo while, utilizando o método 'flag' para encerrar o loop. Antes, a título de formalidade, Segundo Morgado (2013), a média aritmética (x_m, simples) de uma lista de n números $x_1, x_2, \ldots, x_n$ é definida por

$$x_m = \frac{x_1 + x_2 + \cdots + x_n}{n}.$$

Isto posto, vamos ver o seguinte código:

```
soma = 0                                        # soma dos valores
i = 0                                           # quantidade de valores lidos
x = float(input('digite um valor: '))
while x >= 0: # enquanto o valor for >=0
    soma += x
    i += 1
    x = float(input('digite um valor: ')) # o próximo Valor
media = soma / i
print('a média dos valores é {:5.2f}'.format(media))
```

Esse programa permanece solicitando novos valores até o momento em que um valor negativo é fornecido. Esse valor negativo é a *flag*.

Comando break

Embora o comando while seja amplamente utilizado e eficiente na construção de estruturas de repetição, ele não representa a única alternativa disponível para a implementação de algoritmos com comportamento iterativo. O comando break, por exemplo, constitui um recurso adicional que permite o controle do fluxo de execução em estruturas de repetição, especialmente quando o número de iterações é indeterminado. Nesse contexto, realiza-se uma verificação condicional dentro do laço, e, caso a condição especificada seja satisfeita, a repetição é interrompida de forma imediata, forçando a saída do laço antes que a condição natural de término seja atingida (ROJAS; KOSTIN, 2018).

O comando break possui uma ampla gama de aplicações no desenvolvimento de algoritmos, destacando-se, entre elas, sua utilização em processos de busca em estruturas de dados, como listas, conjuntos e dicionários. Nesses contextos, a busca é iniciada sequencialmente, e, ao identificar o elemento desejado, o comando break é acionado para inter-

romper imediatamente o laço de repetição. Essa estratégia evita iterações desnecessárias, promovendo uma execução mais eficiente do algoritmo. Tal otimização é especialmente relevante no campo da computação, onde o desempenho e a economia de recursos são fatores fundamentais.

Ao estabelecer uma relação entre esse recurso tecnológico e o universo matemático caracterizado por uma vasta gama de propriedades rigorosamente definidas nos diferentes conjuntos numéricos, observa-se um amplo leque de possibilidades para a aplicação do comando break. Nesse contexto, conforme delineado neste trabalho, a implementação de algoritmos voltados ao cálculo de fórmulas matemáticas por meio da linguagem de programação Python configura-se como uma estratégia pedagógica promissora. Tal abordagem não apenas contribui para a consolidação do raciocínio lógico e matemático, mas também transforma o processo de ensino-aprendizagem em uma experiência mais interativa, criativa e alinhada às demandas contemporâneas da educação digital.

Exemplo 2.29. Calcular a média aritmética dos valores inseridos até que surja um número negativo, o breack é se o valor for negativo, nesse caso sai do while:

```
soma = 0                # Soma dos valores;
i = 0                   # Quantidade de valores lidos;
while True:             # Enquanto verdade !
    x = float(input('digite um valor: '))
    se x <= 0:
    breack              # se x for negativo, saia do while;
    soma += x
    i += 1
    media = soma/i
    # Imprime mensagem com o valor da média;
    print('a média dos valores é {:5.2f}'.format(media))
```

Comando For

A estrutura de repetição 'for' é amplamente utilizada em linguagens de programação para realizar iterações sobre elementos de uma sequência ou de um objeto iterável, o que também pode ser feito em Python. Seu principal objetivo é percorrer, de maneira ordenada, os itens contidos em uma coleção de dados, como listas, tuplas, strings ou intervalos numéricos, executando um bloco de comandos a cada iteração.

Diferentemente de outras estruturas de repetição, como o 'while', que depende de uma condição lógica para sua continuidade, o laço 'for' pressupõe que o número de repetições ou, ao menos, a quantidade de elementos a serem percorridos, é previamente conhecido

ou determinado pela estrutura de dados fornecida. Dessa forma, o laço 'for' está fortemente associado à ideia de controle iterativo baseado em uma sequência finita e ordenada (ROJAS; KOSTIN, 2018).

Como abordado, esse tipo de comando percorre objetos, dados estruturados do tipo lista por exemplo, em Python existe o comando `range( )`. Esse comando tem como funcionalidade gerar uma progressão aritmética com início e fim.

Definição 2.1. Os objetos range são gerados pela função `range()`, esses objetos são percorridos pelo comando 'for'. O intervalo percorrido pela função `range()` é fechado no início e aberto no fim. O primeiro elemento está incluso e o último não.

Exemplo 2.30. Segue o modelo de construção de objetos range, tais estruturas podem ser geradas assim:

```
range ( início , fim , passo )
início                # Primeiro elemento do intervalo;
fim                   # Último elemento do intervalo;
passo                 # Razão da PA, se não informado, será 1;
```

Como muitos dos objetos percorridos pelo comando for seguem a estrutura de uma progressão aritmética, apresenta-se, a seguir, a definição matemática dessa ferramenta, com o objetivo de favorecer uma melhor compreensão de suas funcionalidades.

Definição 2.2. De acordo com Morgado (2013),Uma progressão aritmética `(PA)` é uma sequência na qual a diferença entre cada termo e o termo anterior é constante. Essa diferença constante é chamada razão da progressão e representada pela letra.

Só para fixar as ideias, as sequências $(2, 4, 6, 8, 10, \dots)$ e $(9, 6, 3, 0, -3, -6, \dots)$ são progressões aritméticas com razões 2 e -3, respectivamente.

Exemplo 2.31. Segue exemplo de estrutura de repetição utilizando o comando 'for' que utiliza a função `range()` para gerar o objeto a ser percorrido:

```
# Percorre intervalo [1,15) gerando PA de razão 2;
for i in range(1,15,2):
# Imprime PA incluindo o 1º elemento e excluindo o último;
    print(i,end = ' ')
1 3 5 7 9 11 13            # Saída.
```

A linguagem de programação Python apresenta inúmeras intersecções com a matemática, o que a torna uma ferramenta valiosa no processo de ensino-aprendizagem dessa

disciplina. Dentre seus diversos recursos, destaca-se o comando for, que, aliado à função `range()`, permite representar e manipular progressões aritméticas de maneira prática e contextualizada. Essa funcionalidade possibilita a abordagem de conceitos matemáticos de forma concreta, favorecendo a construção do conhecimento por meio da experimentação e da resolução de problemas. Assim, a utilização pedagógica de Python pode contribuir significativamente para o desenvolvimento do pensamento lógico e algébrico dos estudantes, especialmente no contexto do Ensino Básico, por meio de práticas didáticas criativas, interativas e alinhadas à cultura digital.

Comando continue

O comando em questão permite que o programa avance para a próxima iteração de um laço de repetição, podendo, inclusive, interagir com estruturas aninhadas, como laços internos. Em determinadas situações, é possível estabelecer condições lógicas que resultam na não execução de certas ações, promovendo a seleção de valores que serão efetivamente processados. Um exemplo comum consiste na omissão de valores múltiplos de 5, os quais podem ser excluídos da saída do programa por meio de instruções condicionais, a fim de refinar o comportamento do algoritmo conforme os objetivos definidos.

Exemplo 2.32. Segue aqui exemplo de estrutura de repetição aplicando o comando 'for' aprimorado pela instrução 'continue':

```
for i in range ( 1 , 4 ):
        for j in range ( 1 , 4 ):
                if i == j:
                        continue
                print ( i , j )
```

Neste programa quando o valor de i for igual ao de j o comando 'continue' não imprime os valores e passa para a próxima etapa produzindo no exemplo em questão o seguinte resultado:

```
1 2
1 3
2 1
2 3
3 1
3 2
```

Exemplo 2.33. Neste exemplo, vamos mostrar o passo a passo de como desenvolver uma calculadora usando conceitos bãsicos apresentados até aqui. Para melhor entendimento, vamos elaborar uma calculadora de soma de matrizes 2×2:

1. **Conceito Matemático.** Sejam duas matrizes A e B de mesma ordem $m \times n$:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}.$$

   A soma $C = A + B$ é definida elemento a elemento da seguinte forma:

$$c_{ij} = a_{ij} + b_{ij}.$$

   Ou seja, somamos cada posição correspondente das matrizes.

2. **Representando Matrizes em Python.** Em Python, representamos uma matriz como uma lista de listas:

```
A = [[1, 2],
     [3, 4]]

B = [[5, 6],
     [7, 8]]
```

   Neste caso, cada lista interna representa uma linha da matriz. Ademais, estamos representando as matrizes

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad \text{e} \quad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}.$$

3. **Criar uma Matriz Resultado.** Precisamos de uma matriz para armazenar a soma.

   Inicialmente, podemos criar uma matriz vazia:

```
C = []
```

4. **Percorrer as Linhas.** Usamos um laço `for` para percorrer cada linha da matriz.

```
for i in range(len(A)):

# Comentario: aqui o comando len() retorna a
  quantidade de elementos de uma lista
```

No qual:

- `len(A)` retorna a quantidade de linhas.
- `i` representa o índice da linha.

5. **Percorrer as Colunas.** Para cada linha, percorremos as colunas:

```
linha = []
for j in range(len(A[0])):
```

Aqui:

- `len(A[0])` retorna a quantidade de colunas.
- `j` representa o índice da coluna.

6. **Somar os Elementos.** Agora realizamos a soma elemento a elemento:

```
soma = A[i][j] + B[i][j]
linha.append(soma)
```

Explicação:

- `A[i][j]` acessa o elemento da linha i e coluna j.
- `B[i][j]` acessa o elemento correspondente.
- `append()` adiciona o resultado na nova linha.

7. **Adicionar a Linha na Matriz Resultado.** Depois de completar a linha:

```
C.append(linha)
```

Agora a matriz resultado começa a ser construída.

8. **Código Completo.**

```
A = [[1, 2],
[3, 4]]

B = [[5, 6],
[7, 8]]
```

```
C = []

for i in range(len(A)):
   linha = []
   for j in range(len(A[0])):
      soma = A[i][j] + B[i][j]
      linha.append(soma)
      C.append(linha)

print(C)
```

9. **Saída Esperada.**

```
[[6, 8],
[10, 12]]
```

Podemos observar que a soma de matrizes em Python segue exatamente a definição matemática: somamos elemento a elemento utilizando dois laços de repetição.

A linguagem de programação Python configura-se como uma ferramenta potencialmente eficaz para o trabalho com propriedades matemáticas no contexto educacional. Desde a introdução de seus conceitos básicos, como variáveis primitivas, estruturas de decisão e comandos de repetição, observa-se um alinhamento significativo com os conteúdos matemáticos abordados no Ensino Básico.

Diversas evidências apontam para a aplicabilidade pedagógica dessa linguagem na mediação de conceitos abstratos, promovendo um ensino mais dinâmico, interativo e centrado na construção ativa do conhecimento por parte dos estudantes. Além de facilitar a compreensão de conteúdos como aritmética, lógica dedutiva e sequências numéricas, Python permite a automação de fórmulas e procedimentos matemáticos, ampliando as possibilidades de exploração prática e significativa desses temas.

Nesse sentido, a utilização de Python no ensino de Matemática contribui para a construção de uma didática contemporânea, que integra os saberes matemáticos às competências digitais, respondendo de forma eficaz às demandas de um cenário educacional cada vez mais inserido em um universo tecnológico em constante transformação.

3 TEORIA DO BIG O

O uso de algoritmos requer uma busca pela execução eficiente, ou seja, a velocidade que as instruções operam e quanto tempo leva para o programa rodar. Para medir tal eficiência alguns conceitos matemáticos são as ferramentas que auxiliam nessa operação. Daí temos a teoria do Big O que nada mais é do que conceitos matemáticos que servem de parâmetro para classificar um algoritmo em bom, ruim, ótimo ou ideal, tudo através de conceitos de comportamento assintótico de algumas funções conhecidas.

3.1 Algoritmos

Atualmente, o conceito de algoritmo encontra-se cada vez mais associado à área da Ciência da Computação, embora sua origem esteja fundamentada na matemática. Trata-se de uma sequência finita e bem definida de instruções ou regras, cuja finalidade é a resolução de problemas específicos ou a realização de tarefas de forma sistemática e eficiente. Essa aproximação entre algoritmos e a Computação evidencia a importância desses procedimentos na construção de soluções computacionais e no desenvolvimento de tecnologias da informação.

Primeiramente precisamos entender o que é um algoritmo em termos computacionais e para isso apresento aqui a definição de algoritmo.

Definição 3.1. Um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída (CORMEN et al., 2002).

Exemplo 3.1. A seguir, será apresentado um exemplo de algoritmo cujo propósito é realizar a ordenação de elementos numéricos, ilustrando sua funcionalidade e aplicabilidade prática.

```
a = float(input(‘digite um número: ’)) # Atibuição variável a;
b = float(input(‘digite um número: ’)) # Atribuição de variável b;
c = float(input(‘digite um número: ’)) # Atribuição de variável c;
d = [a,b,c]                            # Lista com a, b e c;
def insertion(arr):                    # Ambiente insertion;
    for i in range(1, len(arr)):       # for em objeto gerado;
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

```
    return arr
insertion(d)
```

O modelo de programa apresentado acima solicita a entrada de três valores numéricos e depois os apresenta como uma sequência ordenada crescente.

Algoritmos, uma tecnologia

A palavra tecnologia pode ser definida como 'conjunto dos conhecimentos científicos, dos processos e métodos, na criação e utilização de bens e serviços' (HOUAISS, 2010). A definição de algoritmo, enquanto método ou procedimento sistemático executado por computadores para a realização de tarefas específicas como, por exemplo, a ordenação de dados, é fundamental no campo da Ciência da Computação. Para ilustrar a importância da eficiência algorítmica, pode-se recorrer a uma analogia com o automobilismo: mesmo quando os carros de Fórmula 1 possuem configurações idênticas em termos de mecânica e potência, o desempenho final dependerá, em grande medida, da habilidade e técnica de pilotagem de cada competidor. Da mesma forma, diferentes algoritmos podem ser empregados para solucionar um mesmo problema computacional; no entanto, alguns se destacam por sua maior eficiência. Assim, um algoritmo mais bem estruturado e otimizado pode proporcionar tempos de execução significativamente menores, ressaltando a relevância da escolha criteriosa da abordagem algorítmica em função dos requisitos da tarefa a ser executada. Comparando dois algoritmos utilizados para ordenação, o primeiro **ordenação por inserção** que leva um tempo aproximado $c_1 n^2$ para ordenar n itens. O segundo de **ordenação por intercalação**, com um tempo aproximado $c_2 n \lg n$.

Supondo que um computador A execute um bilhão de instruções por segundo, usando ordenação por inserção com código que exige $2n^2$ instruções para ordenar n números.Por outro lado um computador B executa 10 milhões de instrução por segundo, usando ordenação por intercalação com código $50n \lg n$ para ordenar n números. Seja n igual a um milhão de números, o resultado do tempo de cada computador será.

$$\text{Computador A: } \frac{2 \cdot (10^6)^2}{10^9} = 2000 \text{ segundos.}$$

$$\text{Computador B: } \frac{50 \cdot 10^6 \lg 10^6}{10^7} \simeq 100 \text{ segundos.}$$

O problema apresentado mostra que a eficiência de um algoritmo é mais relevante do que a velocidade do computador o que mostra como isso pode ser uma ferramenta usada para melhorar desempenho (CORMEN et al., 2002).

Análise de algoritmos

Em computação outros recursos influenciam na análise de um algoritmo tais como memória, largura de banda de comunicação, hardware e tudo isso vai afetar o tempo de computação e por se busca um algoritmo mais eficiente.

Usando um modelo de computação genérico com um único processador, a RAM (random-acess machine - máquina de acesso aleatório) como nossa tecnologia de implementação em que os algoritmos serão implementados como programas de computador. No modelo de RAM, as instruções são executadas uma após outra.

Quando um algoritmo passa pelo processo de análise o que se procura saber é quanto ele é rápido na execução de uma determinada tarefes,ou seja, como vimos no exemplo anterior, mesmos um computador com uma configuração melhor, um algorítimo mais eficiente pode superá-lo.

Análise da ordenação por inserção

Alguns fatores vão determinar o tempo despendido pelo procedimento INSERTION-SORT tais como a quantidade de elementos da entrada ou quantos elementos da sequência já se encontram ordenados.

Definindo o tempo de execução de um programa como uma função do tamanho de sua entrada, ou seja, para entrada, sequência de elementos a ordenar, teremos um tempo associado de ordenação. Por isso é muito importante definir o tamanho de entrada e o tempo de execução'.

Tamanho da entrada: Depende do problema que está em execução, por exemplo o tamanho do arranjo n para ordenação.

Tempo de execução: É o número de operações primitivas ou etapas executadas, por exemplo: um período constante de tempo é exigido para executar cada linha de um pseudocódigo. Uma linha pode demorar mais do que outra.

A ordenação por inserção utiliza uma abordagem incremental tendo ordenado o subarranjo $A[1...j-1]$, inserimos o elemento isolado $A[j]$ em seu lugar apropriado, formando o subarranjo ordenado $A[1...]$.

Na análise da Inserção precisamos levar em consideração tanto o pior caso, arranjo em ordem decrescente, bem como o melhor caso quando o arranjo já está ordenado. O tempo de execução do pior caso de um algoritmo é um limite superior sobre o tempo de execução para qualquer entrada. Conhecê-lo nos dá uma garantia de que o algoritmo nunca irá demorar mais tempo. Não precisamos fazer nenhuma suposição baseada em fatos sobre o tempo de execução, e temos a esperança de que ele nunca seja muito pior (CORMEN et al., 2002).

3.2 Notação assintótica

A matemática, mais uma vez, desempenha um papel essencial ao oferecer ferramentas conceituais para o avanço de outras áreas do conhecimento, como a Ciência da Computação. Nesse contexto, as notações assintóticas configuram-se como funções matemáticas fundamentais para a análise da eficiência de algoritmos. Elas permitem descrever, de maneira formal, o comportamento do tempo de execução ou do consumo de recursos computacionais em função do crescimento da entrada. É uma ferramente eficiente na análise de algoritmos e para isso dispomos das notações Θ, O, Ω, o, ω.

Notação Θ

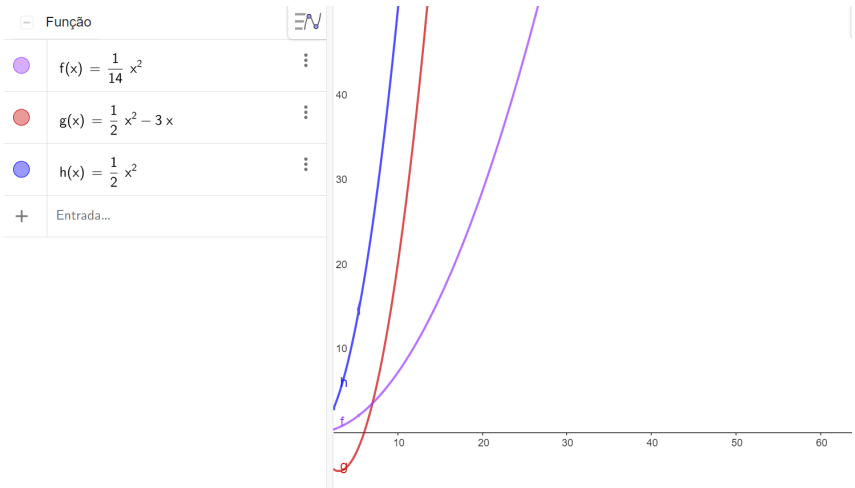
A análise assintótica de algoritmos está intrinsecamente relacionada ao conceito matemático de limite, uma vez que busca descrever o comportamento de funções que expressam o custo computacional. A notação Θ é responsável por fornecer simultaneamente um limite superior `(pior caso)` e um limite inferior `(melhor caso)`, segue definição formal.

Definição 3.2. Para uma dada função $g(n)$, denotamos por $\Theta(g(n))$ o conjunto de funções:

$$\Theta(g(n)) = \{f(n) : \exists\ c_1,\ c_2,\ \text{e}\ n_0\ \text{tais que}\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\ \text{para todo n} \geq n_0\}\,,$$

ou seja, $g(n)$ é um limite assintoticamente restrito para $f(n)$, a partir de um certo n_0 os valores da função $f(n)$ ficam limitados inferiormente por $c_1 g(n)$ e superiormente por $c_2 g(n)$, onde c_1 e c_2 são constantes positivas.

Figura 2 – Gráfico da notação Θ



Fonte: Elaborado pelo autor, 2025.

Exemplo 3.2. seja $\frac{1}{2}n^2 - 3n = \Theta(n^2)$, precisamos encontrar

$$c_1 \text{ e } c_2 \text{ e } n_0 \text{ tais que: } c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \ \forall \ n \geq n_0.$$

Os valores que satisfazem a definição são:

$$c_1 = \frac{1}{14},\ c_2 = \frac{1}{2} \text{ e } n_0 = 7.$$

Notação O

Como exposto anteriormente, a notação (Θ) é utilizada para descrever com precisão o comportamento assintótico de algoritmos, estabelecendo simultaneamente limites inferior e superior para sua complexidade. No entanto, em diversas situações analíticas, é fundamental considerar o desempenho no pior cenário possível, a fim de garantir a robustez e previsibilidade do sistema. Nesse contexto, a análise do pior caso busca determinar o tempo máximo de execução que um algoritmo pode demandar à medida que o tamanho da entrada cresce indefinidamente. Para esse fim, emprega-se a notação assintótica 'Grande O' (O), a qual estabelece um limite superior para a função de complexidade e será formalmente definida a seguir.

Definição 3.3. Para uma dada função $g(n)$, denotamos por $Og((n))$, o grande O, o conjunto das funções

$$O(g(n)) = \{f(n) : \exists \text{ c e } n_0 \text{ tais que } 0 \leq f(n) \leq cg(n) \ \forall \ n \geq n_0\}.$$

A notação O fornece um limite superior sobre uma função, dentro de um fator constante, ou seja, para todos os valores n à direita de n_0, o valor da função $f(n)$ está em ou abaixo de $g(n)$.

Figura 3 – Gráfico da notação O

Fonte: Elaborado pelo autor, 2025.

Exemplo 3.3. Seja $\frac{1}{2}n^2 - 3n \in O(n^2)$

Os valores de c e n_0 que satisfazem a definição são $c = \frac{1}{2}$ e $n_0 = 7$.

Notação Ω

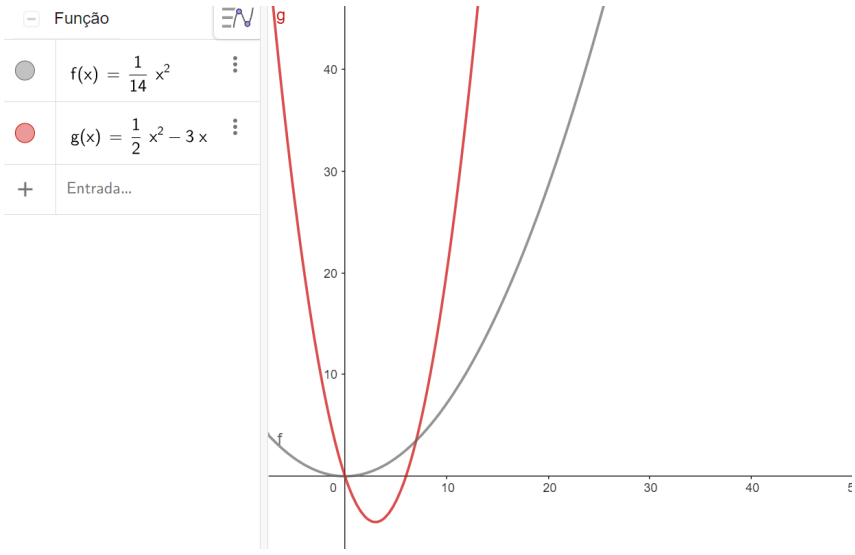
De maneira complementar à notação assintótica O (grande), que estabelece um limite superior para o tempo de execução de um algoritmo no pior caso, a notação (Ω) (ômega) é empregada para representar um limite inferior assintótico. Essa notação descreve o comportamento do algoritmo no melhor caso, ou seja, sob condições ideais de entrada.

Dessa forma, Ω fornece uma garantia de desempenho mínimo, sendo fundamental na análise comparativa de algoritmos quanto à sua eficiência teórica em cenários favoráveis.

Definição 3.4. A notação Ω fornece um limite assintótico inferior, ou seja, esta associado ao melhor desempenho de um algoritmo. Para uma determinada função $g(n)$, denotamos por $\Omega(g(n))$ o conjunto de funções:

$$\Omega(g(n)) = \{f(n) : \exists \text{ c e } n_0 \text{ tais que } 0 \leq cg(n) \leq f(n), n \geq n_0\}.$$

Figura 4 – Gráfico da notação Ω



Fonte: Elaborado pelo autor, 2025.

Para todos os valores de n à direita de n_0, o valor de $f(n)$ está em ou acima de $g(n)$, é um limite inferior, melhor caso para execução do algoritmo.

Exemplo 3.4. Seja $\frac{1}{2}n^2 - 3n \in \Omega(n^2)$.

Os valores de c e n_0 que satisfazem a definição são $c = \frac{1}{14}$ e $n_0 = 7$.

Notação o

Dando continuidade à análise assintótica do comportamento de algoritmos, introduz-se a notação o (letra minúscula), que representa um limite superior estrito para uma função.

Aqui está uma versão acadêmica aprimorada da definição, com maior rigor matemático e clareza formal:

Definição 3.5 (Pequeno-o assintótico)**.** Seja $g(n)$ uma função positiva. Definimos a notação $o(g(n))$ como o conjunto de todas as funções $f(n)$ que satisfazem:

$$o(g(n)) = \{f(n) : \forall\ c > 0,\ \exists\ n_0 \in \mathbb{N} \text{ tal que } 0 \leq f(n) < cg(n),\ \forall\ n \geq n_0\}.$$

A notação $o(g(n))$ estabelece um limite superior assintótico estrito, onde $f(n)$ torna-se insignificante em relação a $g(n)$ quando n tende ao infinito. Em contraste com a notação $\mathcal{O}(g(n))$, que requer apenas que $f(n) \leq cg(n)$ para alguma constante $c > 0$, a condição $o(g(n))$ exige que a desigualdade valha para toda constante positiva c, demonstrando uma dominância assintótica mais forte.

Exemplo 3.5. Seja $1000n^2 \in o\left(n^3\right)$, para todo valor de c, um n_0 que atende a definição é:

$$n_0 = \left\lceil \frac{1000}{c} \right\rceil + 1.$$

Notação ω

De maneira análoga, porém conceitualmente oposta à notação o (minúsculo), a notação ω (ômega minúsculo) é empregada na análise assintótica para descrever um limite inferior estrito de uma função.

Assim, ω fornece um limite inferior que é estritamente maior que $f(n)$, independentemente do valor atribuído à constante c, sendo útil na caracterização de funções cujo crescimento excede estritamente uma determinada taxa assintótica.

Segue abaixo uma definição formal em termos matemáticos:

Definição 3.6. Para uma determinada função $\omega(g(n))$, o pequeno ω, denotamos por $\omega(g(n))$ o conjunto de funções:

$$\omega(g(n)) = \{f(n) : \forall \text{ constante positiva c, existe } n_0 > 0 \text{ tal que } 0 \leq cg(n) \leq f(n), n \geq n_0\}.$$

Ou seja, $cg(n)$ é limite inferior de $f(n)$ para toda constante c positiva e não apenas para alguma como na notação Ω.

Exemplo 3.6. Seja $\frac{1}{1000}n^2 \in \omega(n)$, para todo valor de c, um n_0 que atende a definição é:

$$n_0 = \lceil 1000c \rceil + 1.$$

(CORMEN et al., 2002)

3.3 Big O no ensino básico

A inserção da teoria da notação Big O no contexto da educação básica apresenta desafios significativos, principalmente em virtude da ausência, nos conteúdos curriculares de matemática, de uma abordagem explícita do conceito de limite. Esse conceito é fundamental para a compreensão do comportamento assintótico de funções, o que dificulta a plena assimilação dos princípios que fundamentam a análise da complexidade algorítmica. Dessa forma, a limitação conceitual imposta pela grade curricular vigente constitui um obstáculo à introdução formal e rigorosa dessa temática no ensino fundamental e médio.

Para alcançar resultados mais satisfatórios no ensino da teoria da complexidade algorítmica, é essencial que se realize um estudo preliminar com os alunos sobre as noções de limite, incluindo os conceitos de limites infinitos e no infinito. Esta abordagem proporcionaria uma base sólida para a compreensão do comportamento assintótico das funções, facilitando a assimilação dos princípios fundamentais da análise de algoritmos. Ademais, é necessário que o ensino de algoritmos seja abordado de maneira abrangente, destacando não apenas a construção teórica, mas também a implementação prática por meio da programação de códigos. O domínio de uma linguagem de programação adequada se torna, portanto, um componente crucial para a aplicação dos conceitos discutidos em sala de aula, possibilitando uma compreensão mais profunda e efetiva da eficiência algorítmica e do comportamento computacional.

Superadas as etapas iniciais, a teoria da notação Big O se configura como uma ferramenta relevante para a aplicação de conceitos matemáticos fundamentais, tais como funções, exponenciais, logaritmos e fatoriais. Esse vínculo entre a teoria algorítmica e conceitos matemáticos oferece uma abordagem integrada, permitindo aos alunos perceberem a utilidade prática dos fundamentos matemáticos em contextos computacionais. Além disso, a introdução da notação Big O pode exercer um papel motivador significativo, despertando o interesse dos alunos pela área de informática e incentivando-os a aprofundar seus estudos nas áreas relacionadas ao desenvolvimento e análise de algoritmos.

4 ALGORITMOS DE ORDENAÇÃO

No âmbito da ciência da computação, é comum a ocorrência de problemas cuja resolução exige o desenvolvimento de algoritmos capazes de oferecer soluções eficientes. Dentre as aplicações mais frequentes, destaca-se o processo de ordenação de sequências, que consiste na reorganização de elementos segundo uma determinada ordem, geralmente crescente ou decrescente. Os algoritmos concebidos para esse propósito são denominados algoritmos de ordenação, e representam uma classe fundamental de ferramentas computacionais utilizadas na resolução de diversos problemas. A ordenação é, frequentemente, uma etapa preliminar essencial em inúmeros procedimentos computacionais, sendo, portanto, de grande relevância tanto do ponto de vista teórico quanto prático. A análise da eficiência desses algoritmos, especialmente sob a ótica da complexidade computacional, constitui um aspecto central na avaliação de seu desempenho.

4.1 Por que ordenar?

A utilização de algoritmos de ordenação justifica-se, em grande medida, pela necessidade de organizar informações de modo cronológico, alinhando-se à forma como os seres humanos percebem e estruturam o tempo. Nesse contexto, torna-se essencial ordenar conjuntos de dados em uma linha temporal, sobretudo em aplicações que demandam análise sequencial ou acompanhamento histórico de eventos.

A ordenação, também denominada classificação de registros, consiste na disposição de elementos em ordem crescente ou decrescente, com o objetivo de facilitar sua manipulação, recuperação e análise. Dentre os principais benefícios desse processo, destaca-se a possibilidade de realizar buscas e consultas de forma mais eficiente, uma vez que conjuntos ordenados permitem a aplicação de algoritmos de pesquisa com desempenho superior em relação a estruturas desordenadas.

A literatura da ciência da computação apresenta uma ampla variedade de algoritmos de ordenação, os quais empregam diferentes estratégias e técnicas, refletindo a complexidade e a diversidade do problema. A relevância histórica da ordenação é notável, sendo um tema recorrente tanto em pesquisas acadêmicas quanto em aplicações de engenharia. A eficiência desses algoritmos pode ser impactada por diversos fatores, tais como o conhecimento prévio sobre as chaves de ordenação, a estrutura dos dados satélites, a hierarquia de memória do sistema computacional e as especificidades do ambiente de software utilizado (CORMEN et al., 2002).

4.2 Tipos de algoritmos de Ordenação

Dentre os algoritmos utilizados para a ordenação de dados, alguns se destacam como opções mais adequadas dependendo das características específicas dos dados a serem manipulados. A escolha da técnica de ordenação mais eficiente está diretamente relacionada

a fatores como o tamanho da entrada, a organização prévia dos elementos e os requisitos de desempenho do sistema. As metodologias empregadas por esses algoritmos variam, podendo basear-se na comparação direta entre elementos adjacentes ou na aplicação da estratégia de dividir para conquistar, frequentemente implementada por meio de funções recursivas.

Entre os algoritmos mais conhecidos nesse contexto, destacam-se o Insertion Sort, que avalia a posição relativa de cada elemento na sequência em comparação aos demais, bem como o Merge Sort e o Quick Sort, que utilizam a abordagem recursiva para decompor o problema original em subproblemas menores, resolvê-los de forma independente e, em seguida, combinar suas soluções. Esses algoritmos são amplamente estudados e aplicados devido à sua relevância teórica e eficácia prática na resolução de problemas de ordenação em diferentes domínios computacionais.

Insertion Sort

O algoritmo de ordenação por inserção, ou Insertion Sort, destaca-se por sua simplicidade operacional e por apresentar desempenho satisfatório quando aplicado a conjuntos de dados de pequena dimensão. Sua lógica baseia-se na varredura sequencial da estrutura de dados, da esquerda para a direita, inserindo cada elemento na posição correta em relação aos elementos previamente ordenados, conforme o critério de ordenação previamente definido. Em cada iteração, o algoritmo compara o elemento atual com os anteriores, deslocando-os, se necessário, até encontrar a posição apropriada para inserção. Como resultado, a subsequência à esquerda da posição atual permanece continuamente ordenada. Embora sua complexidade assintótica de tempo, no pior caso, seja quadrática $O(n^2)$, o Insertion Sort revela-se eficiente em situações em que o número de elementos é reduzido ou quando os dados se encontram parcialmente ordenados, sendo, por isso, amplamente utilizado em aplicações específicas e como base para algoritmos mais complexos (CORMEN et al., 2002).

Exemplo 4.1. Considere uma sequência numérica de entrada composta pelos elementos $[3, 9, 7, 5, 1]$. O objetivo é ordená-los em ordem crescente, utilizando um algoritmo que reposicione cada elemento conforme o critério de ordenação previamente estabelecido. Para isso, os elementos são selecionados sequencialmente e, a cada etapa, inseridos na posição adequada em relação aos elementos anteriormente processados. Este processo de reordenação garante que, ao final de todas as iterações, a sequência esteja completamente ordenada de forma crescente.

```
[3, 9, 7, 5, 1] # Número 1 comparado com os demais;
[1, 3, 9, 7, 5] # Número 1 posicionado na ordem adequada;
[1, 3, 5, 9, 7] # Número 5 posicionado na ordem adequada;
[1, 3, 5, 7, 9] # Número 7 posicionado na ordem adequada, ordenada.
```

Análise do Insert Sort

Para analisar a eficiência do algoritmo Insertion Sort, é fundamental compreender seu procedimento de execução. O algoritmo percorre os índices do array $1, 2, 3, ..., n-1$, comparando cada elemento com o valor a ser ordenado e, em seguida, inserindo-o na posição adequada. Esse processo é repetido para todos os elementos da sequência, resultando, ao final, em uma lista ordenada.

A operação de inserção requer um tempo que depende do tamanho do subarray já ordenado. Em outras palavras, o tempo necessário para inserir um elemento aumenta conforme o número de elementos previamente ordenados, o que pode afetar a eficiência do algoritmo.

Vamos analisar a situação onde nós chamamos insert e o valor que está sendo inserido em um subarray é menor que todos os elementos.

Exemplo 4.2. Inserir 1 no subarray $[3, 4, 5, 8, 10]$, os elementos terão que se deslocar um índice para direita.

Ou seja, um *subarray* com k elementos, todos eles podem ter de se deslocar em uma posição. Admitindo que o número de linhas seja um constante c, poderia levar $c.k$ linhas para inserir em uma *subarray* de k elementos.

Sendo assim temos que para $k = 1$, $k = 2$, $k = 3$ até $k = n - 1$, onde o valor que está sendo inserido é menor que todos os elementos do subarray a sua esquerda, o tempo gasto pode ser encontrado pela expressão:

$$c \cdot 1 + c \cdot 2 + c \cdot 3 + \cdots c \cdot (n-1) = c \cdot (1 + 2 + 3 + \cdots + (n-1)).$$

Podemos observar que a soma dos índices corresponde à soma dos termos de uma progressão aritmética de razão 1.

$$S_n = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2},$$

em que o último termo é $(n-1)$, logo fazendo a substituição temos,

$$c\frac{(n-1+1)(n-1)}{2} = \frac{cn^2}{2} - \frac{cn}{2}.$$

Usando a notação Θ, descartando o termo de baixa ordem $\frac{c \cdot n}{2}$ e os fatores constantes c e $\frac{1}{2}$, o resultado do tempo para a ordenação por inserção é $\Theta(n^2)$.

Não é toda situação que a ordenação por inserção levará um $\Theta(n^2)$, podendo ser menor a depender o quanto o *subarray* já está ordenado (CORMEN et al., 2002).

Exemplo 4.3. Seja o array $[2, 4, 5, 6, 8]$, onde os quatro primeiros já estão ordenados. No primeiro teste, nós vemos que 8 é maior que 6, os elementos no subarray precisam se deslocar para a direita. Essa chamada de insert leva só um tempo constante. Supondo que cada chamada insert leve um tempo constante c num total de $(n-1)$ chamadas, logo o tempo total de *ordenção por inserção* é $c(n-1)$, ou seja, $\Theta(n)$ e não $\Theta(n^2)$, conforme Khan Academy (KHAN ACADEMY, 2024).

Tempos de execução da ordenação por inserção

A seguir, são apresentadas algumas situações relacionadas ao tempo de execução na ordenação de elementos utilizando o método Insertion Sort. Em cada uma dessas situações, existe uma função matemática responsável por determinar sua eficiência:

- Pior caso: $\Theta(n^2)$;
- Melhor caso: $\Theta(n)$;
- Casos médio para um arranjo aleatório: $\Theta(n^2)$;
- Caso de arranjo 'quase ordenado': $\Theta(n)$.

Merge Sort

Ainda quando se busca ordenar elementos, sem deixar lado a eficiência, o método 'merge sort' se apresenta como uma ótima opção, visto que se baseia em recursão, que emprega a técnica de dividir e conquistar. O problema é dividido em vários subproblemas menores, semelhantes ao problema original. Esses subproblemas são resolvidos recursivamente, e, em seguida, suas soluções são combinadas com o objetivo de construir uma solução para o problema original.

Quando um algoritmo contém uma chamada recursiva a si próprio, seu tempo de execução pode, frequentemente, ser descrito por uma equação de recorrência, que expressa o tempo total de execução de um problema de tamanho n em termos do tempo de execução de entradas menores. A partir dessa recorrência, é possível utilizar ferramentas matemáticas para resolvê-la e, assim, estabelecer limites teóricos sobre o desempenho do algoritmo (CORMEN et al., 2002).

Exemplo 4.4. Seja a sequência $(3, 2, 7, 5, 4, 6, 8, 1)$, pelo método de dividir e conquistar temos como resultado o procedimento:

```
[3,2]-[7,5]-[4,6]-[8,1]   # Sequência dividida em 4 subgrupos;
[2,3,5,7]-[1,4,6,8]       # Elementos ordenados e reposicionados;
[1,2,3,4,5,6,7,8]         # Sequência ordenada.
```

Conforme evidenciado no exemplo anterior, observa-se que o paradigma adotado para a ordenação dos elementos segue uma estrutura bem definida, composta por três etapas principais, descritas a seguir:

1. Dividir o problema em um número de subproblemas que sejam partes menores do mesmo problema.

2. Conquistar os subproblemas resolvendo-os recursivamente. Se eles forem pequenos o suficiente, resolva os subproblemas como problemas base.

3. Combinar as soluções dos subproblemas em uma solução para o problema original.

Análise do Merge Sort

Quando um algoritmo envolve chamadas recursivas a si próprio, seu tempo de execução pode frequentemente ser modelado por meio de uma equação de recorrência, a qual expressa o tempo total de execução para um problema de tamanho n em termos do tempo de execução para instâncias menores do mesmo problema. A partir dessa equação, é possível utilizar ferramentas matemáticas adequadas para resolver a recorrência e, dessa forma, estabelecer limites assintóticos que caracterizam o desempenho do algoritmo (CORMEN et al., 2002).

Uma recorrência para o tempo de execução de um algoritmo de dividir e conquistar se baseia nos três passos do paradigma básico. Consideramos $T(n)$ o tempo de execução sobre um problema de tamanho n. Se o tamanho do problema for pequeno o bastante, digamos $n \leq c$ para alguma constante c, a solução direta demorará um tempo constante $\Theta(1)$. Vamos supor que o problema seja dividido em a subproblemas, cada um dos quais com $\frac{1}{b}$ do tamanho do problema original. Se levarmos o tempo $D(n)$ para dividir o problema em subproblemas e o tempo $C(n)$ para combinar as soluções dadas aos subproblemas na solução para o problema original, obteremos a recorrência:

$$T(n) = \begin{cases} \Theta(1) & : \ n \leq c, \\ aT(n/b) + D(n) + C(n) & : \text{ no caso contrário.} \end{cases} \tag{4.1}$$

Tempos de execução da ordenação por Merge Sort

A seguir, são apresentadas algumas situações relacionadas ao tempo de execução na ordenação de elementos utilizando o método Merge Sort. Em cada uma dessas situações, existe uma função matemática responsável por determinar sua eficiência. Ao compararmos o crescimento das funções associadas aos algoritmos Merge Sort e Insertion Sort, é perceptível que funções logarítmicas ou quase-lineares apresentam um crescimento menos acentuado, resultando em menor tempo de execução quando comparadas a funções de crescimento quadrático.

- Pior caso: $\Theta(n \log n)$;
- Melhor caso: $\Theta(n \log n)$;
- Casos médio: $\Theta(n \log n)$.

Quicksort

Ordenação rápida, é um algoritmo de ordenação com tempo de execução pior caso $\Theta(n^2)$ sobre um arranjo de entrada de n números (CORMEN et al., 2002).

Apesar de ser lento no tempo de execução do pior caso, é uma boa opção prática para ordenação em razão de sua eficiência média. Seu tempo de execução esperado é $\Theta(n \lg n)$.

O algoritmo Quicksort, assim como o Merge Sort, fundamenta-se no paradigma de Dividir e Conquistar, o qual organiza o processo de ordenação de um subarranjo típico [p. . . r] em três etapas principais: divisão, conquista e combinação.

No Merge Sort, a etapa de divisão é relativamente simples, sendo realizada de forma direta e uniforme. A maior parte do esforço computacional concentra-se na etapa de combinação, responsável por intercalar os subarranjos previamente ordenados. Por outro lado, no Quicksort, ocorre o inverso: o principal trabalho é realizado na etapa de divisão, que envolve o particionamento eficiente do arranjo em torno de um elemento pivô, de modo que os elementos menores fiquem à esquerda e os maiores à direita. A etapa de combinação, nesse caso, é praticamente inexistente, visto que a ordenação é obtida implicitamente ao final das chamadas recursivas.

- Dividir: O arranjo $A[p, \ldots, r]$ é particionado em dois subarranjos $A[p, \ldots, q-1]$ e $A[q+1, \ldots, r]$ onde cada elemento de $A[p, \ldots, q-1]$ é menor que ou igual a $A[q]$, chamado *pivô*, que por sua vez, é igual ou menor a cada elemento de $A[q+1, \ldots, r]$.
- Conquistar: Os dois subarranjos $A[p, \ldots, q-1]$ e $A[q+1, \ldots, r]$ são ordenados por chamadas recursivas a quicksort.
- Combinar: Como os subarranjos são ordenados localmente, não é necessário nenhum trabalho para combiná-los. O arranjo $A[p, \ldots, r]$ inteiro agora está ordenado.

Exemplo 4.5. Seja o array $[9, 7, 5, 11, 12, 2, 14, 3, 10, 6]$, escolhendo o elemento mais a direita como pivô, ou seja $A[6]$. Ordenando recursivamente todos os elementos das subarrays $[5, 2, 3]$ e $[12, 7, 14, 9, 10, 11]$, com o 6 como pivô, obtemos:

```
[2,3,5] [6] [7,9,10,11,12] # Conquistar;
[2,3,5,6,7,9,10,11,12]     # Combinar, array ordenado.
```

Função partition

Definição 4.1. A função partition deve particionar o subarray array $[p, \ldots, r]$ de forma que todos os elementos em array $[p, \ldots, q-1]$ sejam menores ou iguais a array $[q]$ (o pivô) e todos os elementos em array $[q+1, \ldots, r]$ sejam maiores que array$[q]$, e ela retorna o índice q de onde o pivô termina.

Análise do Quick Sort

Assim como analisamos a eficiência dos algoritmos Insertion Sort e Merge Sort, podemos, neste ponto, abordar as situações mais recorrentes relacionadas ao tempo de execução do Quicksort.

- **Tempo de execução pior caso:** Quando a ordenação rápida tem sempre as partições o mais desbalanceada possível, então a chamada original leva tempo cn para alguma constante c, a chamada recursiva sobre $n-1$ elementos leva $c(n-1)$ de tempo, a chamada recusiva sobre $n-2$ elementos leva $c(n-2)$ de tempo, e assim por diante (CORMEN et al., 2002).

  $$cn + c(n-1) + c(n-2) + ... + 2c = c(n + (n-1) + (n-2) + ... + 2),$$

  aplicando a soma de todos os termos de uma P.A temos a expressão:

  $$c((n+1)\frac{n}{2} - 1).$$

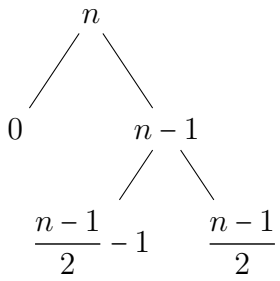
  Ordenação rápida no pior caso roda com o tempo $\Theta(n^2)$.

- **Tempo de execução no melhor caso:** Ocorre quando as partições bem balanceadas: os seus tamanhos são iguais ou até 1 de cada uma. O primeiro caso ocorre se o subarranjo tem um número ímpar de elementos e o pivô está no meio depois do particionamento e cada partição tem $\frac{(n-1)}{2}$ elementos. O último caso ocorre se o subarranjo tem um número par de elementos n e uma partição tem $\frac{n}{2}$ elementos com a outra tendo $\frac{n}{2} - 1$. Em ambos os casos, cada partição tem no máximo $\frac{n}{2}$ elementos, e a árvore do tamanho do subproblema parece a árvore do subproblema do um tipo fundir de mesclagem, com as partições de tempo parecendo fundir com o tempo.

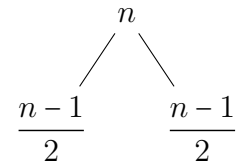
$$
\begin{aligned}
& cn \\
& \leq 2 \cdot \tfrac{cn}{2} = cn \\
& \leq 4 \cdot \tfrac{cn}{4} = cn \\
& \leq 8 \cdot \tfrac{cn}{8} = cn \\
& \cdots \\
& < n \cdot c = cn.
\end{aligned}
\tag{4.2}
$$

Usando a notação Big Θ obtemos o mesmo resultado que a ordenação Merge Sort $\Theta(n \log n)$.

- **Tempo de execução caso médio:** Quando executamos o quicksort sobre um arranjo de entrada aleatório, é improvável que o particionamento sempre ocorra do mesmo modo em todo nível. Temos algumas divisões bem-equilibradas e outras muito desequilibradas. No caso médio, PARTITION produz uma mistura de divisões "boas"e "ruins". Em uma árvore de recursão para uma execução do caso médio de PARTITION, as divisões boas e ruins estão distribuídas aleatoriamente ao longo da árvore.



A figura acima mostra as divisões em dois níveis consecutivos na árvore de recursão. Na raiz da árvore, o custo é n para particionamento e os subarranjos produzidos têm tamanhos $n-1$ e 1: o pior caso. No nível seguinte, o subarranjo de tamanho $n-1$ é particionado no melhor caso em dois subarranjos de tamanho $\frac{n-1}{2} - 1$.



Um único nível de uma árvore de recursão que está muito bem equilibrada. Em ambas as partes, o custo de particionamento para os subproblemas mostrados dos subarrajos de tamanho diferente de zero é $\Theta(n)$.

A combinação da divisão ruim seguida pela divisão boa produz três subarranjos de tamanhos O, $\frac{(n-1)}{2} - 1$ e $\frac{(n-1)}{2}$, a um custo de particionamento combinado

$$\Theta(n-1) + \Theta(n) = \Theta(n). \tag{4.3}$$

o tempo de execução do Quick-Sort, quando os níveis se alternam entre divisões boas e ruins, é semelhante ao tempo de execução para divisões boas sozinhas, ou seja, $\Theta(n \log n)$, mas com uma constante ligeiramente maior oculta pela notação de O (CORMEN et al., 2002).

4.3 Recorrência

No contexto da análise de algoritmos recursivos, a compreensão dos procedimentos utilizados para avaliar sua eficiência exige, necessariamente, o entendimento do conceito matemático de recorrências. Essas equações desempenham um papel fundamental na modelagem do tempo de execução, ao expressar o custo computacional de um algoritmo em função do tamanho das instâncias de entrada.

Muitas sequências são definidas recursivamente, isto é, por recorrência. Uma regra que permite calcular qualquer termo em função do(s) antecessor(es) imediato(s) (MORGADO; CEZAR, 2013).

Exemplo 4.6. Aqui, pode-se observar alguns exemplos de como uma sequência pode ser expressa por meio de uma relação de recorrência:

Seja a sequência (x_n) de números ímpares $1, 3, 5, 7, 9, \cdots$ pode ser definida por

$$x_{n+1} = x_n + 2,\ n \geq 1,\ x_1 = 1.$$

Qualquer progressão aritmética (x_n) de razão r e primeiro termo a pode ser definida por

$$x_{n+1} = x_n + r,\ n \geq 1,\ x_1 = a.$$

Recorrências Lineares de Primeira Ordem

Conforme discutido anteriormente, uma recorrência consiste em uma equação que define os termos de uma sequência a partir de sua posição n, sendo n um número natural. Tal equação estabelece uma forma geral capaz de determinar, de maneira sistemática, os valores correspondentes de cada termo da sequência.

Do ponto de vista matemático, a equação que define uma relação de recorrência pode ser de primeiro ou segundo grau. Essa distinção está diretamente relacionada à ordem da recorrência, classificando-a, respectivamente, como de primeira ou segunda ordem. A seguir, apresenta-se a definição formal de uma recorrência de primeira ordem.

Definição 4.2. Uma recorrência de primeira ordem expressa x_{n+1} em função de x_n. Ela é dita linear se (e somente se) essa for do primeiro grau.

Exemplo 4.7. As recorrências $x_{n+1} = 2x_n - n^2$ e $x_{n+1} = nx_n$ são lineares e a recorrência $x_{n+1} = x_n^2$ não é linear. As duas últimas são ditas homogêneas pois não possuem termo independente de x_n.

Exemplo 4.8. Resolva a recorrência $x_{n+1} = nx_n$, $x_1 = 1$.
Solução:

$$\begin{aligned} x_2 &= 1x_1, \\ x_3 &= 2x_2, \\ x_4 &= 3x_3, \\ &\dots \quad \dots \quad \dots \\ x_n &= (n-1)x_{n-1}. \end{aligned} \tag{4.4}$$

Daí, multiplicando obtemos $x_n = (n-1)!x_1$. Como $x_1 = 1$, temos que $x_n = (n-1)!$.

Equações recorrentes de primeira ordem são fundamentais para analisar a complexidade de algoritmos de ordenação recursivos, como merge sort, quick sort e heap sort. Essas equações descrevem o tempo de execução de um algoritmo em termos de chamadas recursivas menores.

Entretanto, nem sempre tais equações são diretamente dedutíveis a partir da descrição do algoritmo. Em diversos casos, é necessário recorrer a técnicas matemáticas que permitam transformar ou resolver recorrências não triviais, de forma a descrever seu comportamento assintótico. Uma dessas técnicas é o método de transformação, também conhecido como método da substituição, que consiste em assumir uma solução candidata para a recorrência e demonstrar, por meio de substituição e indução matemática, que ela satisfaz a equação para todos os valores de entrada a partir de um determinado ponto.

Teorema 4.1. *Segundo Morgado (2013), Se a_n é uma solução não nula da recorrência $x_{n+1} = g(n)x_n$, então a substituição $x_n = a_n y_n$ transforma a recorrência $x_{n+1} = g(n)x_n + h(n)$ em $y_{n+1} = y_n + h(n)[g(n)a_n]^{-1}$.*

Demonstração. A substituição $x_n = a_n y_n$, transforma $x_{n+1} = g(n)x_n + h(n)$ em $a_{n+1}y_{n+1} = g(n)a_n y_n + h(n)$.

Mas, $a_{n+1} = g(n)a_n$, pois a_n é solução de $x_{n+1} = g(n)x_n$. Portanto a equação se transforma em:

$$g(n)a_n y_{n+1} = g(n)a_n y_n + h(n),$$

Ou seja,

$$y_{n+1} = y_n + h(n)[g(n)a_n]^{-1}.$$

□

Recorrências Lineares de Segunda Ordem

Equações recorrentes que dependem de dois termos anteriores são classificadas como equações lineares de segunda ordem. Tais equações apresentam propriedades análogas às das equações algébricas de segundo grau, especialmente no que se refere à resolução por meio da equação característica. Essa similaridade permite a aplicação de métodos analíticos consolidados para a determinação de soluções gerais e particulares.

Ainda que as recorrências lineares de primeira ordem predominem na análise de algoritmos de ordenação, o estudo das recorrências lineares de segunda ordem revela-se igualmente relevante. A compreensão desses modelos mais complexos de dependência recursiva proporciona uma visão mais aprofundada da análise de algoritmos, especialmente em casos que envolvem estratégias adaptativas, otimizações estruturais ou algoritmos híbridos. Dessa forma, o conhecimento sobre recorrências de segunda ordem amplia a capacidade analítica do pesquisador ao lidar com problemas computacionais de maior complexidade. A forma geral de uma equação de recorrência linear de segunda ordem com coeficientes constantes pode ser expressa da seguinte maneira:

$$a_n = r_1 a_{n-1} + r_2 a_{n-2} + f(n),$$

em que a_n representa o termo geral da sequência, r_1 e r_2 são constantes reais, e $f(n)$ é uma função conhecida de n, caracterizando a parte não homogênea da equação. No caso particular em que $f(n) = 0$, a equação é classificada como homogênea. Esse tipo de recorrência é amplamente utilizado na modelagem de problemas computacionais e matemáticos, sendo sua solução obtida por meio da análise da equação característica associada.

Seja a recorrência da forma $x_{n+2} + px_{n+1} + qx_n = 0$, $q \neq 0$, com p e q constantes, uma recorrência linear homogênea de segunda ordem. Para cada recorrência de segunda ordem associamos uma equação do segundo grau do tipo

$$r^2 + pr + q = 0,$$

chamada equação característica.

Exemplo 4.9. Apresenta-se, a seguir, um exemplo de equação de recorrência linear de segunda ordem, no qual é possível determinar explicitamente as raízes da equação característica associada.

A recorrência $x_{n+2} = x_{n+1} + x_n$ tem equação característica $r^2 = r + 1$. As raízes da equação característica são:

$$r_1 = \frac{1+\sqrt{5}}{2},\ r_2 = \frac{1-\sqrt{5}}{2}.$$

A resolução de equações de recorrência lineares de segunda ordem com coeficientes constantes fundamenta-se na análise das raízes da equação característica associada. O teorema correspondente estabelece que, conhecidas essas raízes, é possível determinar a forma geral da solução da recorrência para todo $n \in \mathbb{N}$, mediante a construção de expressões que variam conforme a natureza das raízes sejam elas reais e distintas, reais e coincidentes ou complexas conjugadas.

Teorema 4.2. *Se as raízes de* $r^2 + pr + q = 0$ *são* r_1 *e* r_2*, então* $a_n = C_1 r_1^n + C_2 r_2^n$ *é solução da recorrência* $x_{n+2} + px_{n+1} + qx_n = 0$*, quaisquer que seja os valores das constantes* C_1 *e* C_2*.*

Demonstração. Substituindo $a_n = C_1 r_1^n + C_2 r_2^n$ na recorrência $x_{n+2} + px_{n+1} + qx_n = 0$, obtemos:

$$C_1 r_1(r_1^2 + pr + q) + C_2 r_2(r_2^2 + pr + q) = C_1 r_1^n 0 + C_2 r_2^n 0 = 0.$$

□

5 UMA PROPOSTA DE UTILIZAÇÃO DA LINGUAGEM DE PROGRAMAÇÃO PYTHON NO ENSINO DE MATEMÁTICA

Conforme discutido anteriormente, a busca por metodologias de ensino que promovam uma aprendizagem significativa e contextualizada dos conteúdos matemáticos tem se intensificado, especialmente diante da crescente presença de temas contemporâneos como inteligência artificial, robótica e programação no contexto educacional e tecnológico. Nesse cenário, o presente trabalho propõe uma abordagem para o ensino de Matemática fundamentada na integração com a linguagem de programação Python, com o objetivo de explorar conexões entre conceitos matemáticos e aplicações computacionais. Tal proposta visa favorecer o desenvolvimento de competências analíticas, lógicas e tecnológicas por parte dos estudantes, alinhando o ensino da Matemática às demandas da sociedade digital.

Local da experimentação

As atividades foram desenvolvidas na Escola Alzira da Fonseca Breuel, com turmas do 3º ano do Ensino Médio regular, no turno da tarde. As aulas ocorreram de forma presencial, sendo realizadas no espaço da biblioteca da escola, tendo em vista a ausência de um laboratório de informática na instituição. Para a execução da proposta, foram utilizados os seguintes recursos pedagógicos: 26 notebooks, material teórico em PDF, programas de computador, aplicativos para smartphone, Google Colab, tutor e quadro.

5.1 Sequência didática

Sequência de aulas Python e MATEMÁTICA.

ESCOLA: Alzira da Fonseca Breuel	**TURMA:** 3º anos E,F,G e H
DISCIPLINA: Matemática	**ASSUNTO:** Python E MATEMÁTICA
AULAS PREVISTAS: 12 aulas	**TEMPO POR AULA:** 50 minutos

CONTEÚDOS

1. Noções de informática básica;
2. Algoritmos;
3. Linguagem de programação Python;
4. Noções de lógica;
5. Juros Simples e Juros Compostos;

6. Volumes de Sólidos Geométricos;

7. Média aritmética, Moda e Mediana;

8. Construção dos algoritmos das fórmulas matemáticas.

OBJETIVOS

proposta tem como objetivo integrar o ensino de Matemática aos fundamentos da informática e da programação, utilizando a linguagem Python como ferramenta pedagógica. Para isso, busca-se ensinar conceitos básicos de informática, desenvolver habilidades voltadas à linguagem de programação, introduzir a construção de algoritmos e o uso da linguagem Python no contexto educacional, além de explorar as tendências contemporâneas que articulam Matemática e Informática. Pretende-se, ainda, desenvolver diversas competências nos estudantes, especialmente a capacidade de compreender e utilizar, com flexibilidade e precisão, diferentes registros de representação matemática como os algébrico, geométrico, estatístico e computacional na busca por soluções de problemas e na comunicação de resultados. Também se visa introduzir os alunos no universo tecnológico, ensinando códigos de programação e aplicando-os no desenvolvimento de algoritmos voltados à automatização de fórmulas matemáticas, por meio do uso de variáveis como dados de entrada e assim construir uma aprendizado significativo e duradouro.

RECURSOS

Quadro, marcadores coloridos em pelo menos três cores, notebooks, programa Python, conta no Google Colab e material teórico em apostilas em PDF.

AULA 1

NOÇÕES BÁSICAS EM INFORMÁTICA

TEMPO:

2 AULAS DE 50 MINUTOS

ORGANIZAÇÃO DA TURMA:

Alunos organizados, sentados à mesa central da biblioteca, com computadores e notebooks em mãos.

TENDÊNCIA EM EDUCAÇÃO MATEMÁTICA:

Matemática e Informática, computadores e linguagem de programação.

INTRODUÇÃO:

A proposta consiste na abordagem dos conceitos fundamentais da informática, destacando sua evolução histórica, principais funcionalidades e, sobretudo, sua ampla aplicabilidade nos diversos contextos da sociedade contemporânea.

DESENVOLVIMENTO:

1. Definição de computador e suas funcionalidades;
2. As principais operações, processamento e armazenamento de dados;
3. Definição da parte física, HARDWARE, e a parte abstrata, SOFTWARE;
4. Manuseio de hardwares, monitor, teclado, gabinete e mouse;
5. Sistema operacional WINDOWS e suas funcionalidades;
6. Memórias primária e a secundária. A importância da memória RAM.

CONCLUSÃO:

Mostrar que o computador, composto por um conjunto de componentes interdependentes, configura-se como uma ferramenta de ampla aplicabilidade, atendendo a múltiplas finalidades no cotidiano e em contextos profissionais. Entre suas principais funções, destacam-se o armazenamento de documentos, imagens e vídeos; a produção de textos e planilhas; a realização de pesquisas; o envio e recebimento de arquivos por meio de correio eletrônico; bem como a utilização de sistemas informatizados em instituições bancárias, estabelecimentos comerciais, supermercados e ambientes corporativos. Além disso, possibilita a execução de jogos, cálculos complexos e a edição de conteúdos audiovisuais. Assim, torna-se evidente que o domínio do uso do computador é uma competência fundamental na sociedade contemporânea.

AULA 2

ALGORITMOS

TEMPO:

2 AULAS DE 50 MINUTOS

ORGANIZAÇÃO DA TURMA:

Alunos organizados, sentados à mesa central da biblioteca, com computadores e notebooks em mãos.

TENDÊNCIA EM EDUCAÇÃO MATEMÁTICA:

Matemática e Informática, computadores e linguagem de programação.

INTRODUÇÃO:

Mostrar que algoritmo é uma sequência finita, ordenada e precisa de instruções que visam à resolução de um problema ou à execução de uma tarefa específica. Esse conceito não se restringe ao âmbito da computação, estando presente em diversas atividades cotidianas. Exemplificar que uma receita para preparar um bolo, que orienta passo a passo os procedimentos a serem realizados, ou o caminho escolhido para chegar à escola, que envolve uma sequência lógica de decisões, configuram algoritmos. Tais conjuntos de instruções organizadas logicamente são o que denominamos *algoritmos*.

DESENVOLVIMENTO:

1. Algoritmos matemáticos: algoritmo da divisão Euclidiana, algorítimo da adição, algoritmo do mmc, fórmula usada para encontrar os juros simples;

2. Definição computacional para algoritmo;

3. Baixar o aplicativo QPython nos celulares;

4. Atribuição de variáveis.

CONCLUSÃO :

Para concluir a aula de forma prática, utilizaremos o QPython, um ambiente de desenvolvimento Python para dispositivos móveis. O QPython permite que você escreva, edite e execute códigos Python diretamente no seu celular, facilitando o aprendizado e a execução interativa de algoritmos.

AULA 3

Noções da linguagem de programação Python

TEMPO:

2 AULAS DE 50 MINUTOS

ORGANIZAÇÃO DA TURMA:

Alunos organizados, sentados à mesa central da biblioteca, com computadores e notebooks em mãos.

TENDÊNCIA EM EDUCAÇÃO MATEMÁTICA:

Matemática e Informática, computadores e linguagem de programação.

INTRODUÇÃO:

Falar sobre a história da linguagem de programação Python e seu criador Guido van Rossum. Mostrar a necessidade de uma linguagem intermediária que seja capaz de proporcionar uma comunicação eficaz entre seres humanos e computadores. Nesse contexto, surge a linguagem de programação, códigos que permitem que sejam dadas instruções aos programas computacionais.

DESENVOLVIMENTO:

1. Instalação do Python no sistema operacional WINDOWS;
2. Interação com ambiente de aprendizado e edição de códigos. Criar e salvar arquivos;
3. Dados primitivos simples e estruturados em Python: strings, numéricos, booleanos, listas;
4. Operações Aritméticas em Python;
5. Funções *print*, dados de saída, e *input*, dados de entrada.

CONCLUSÃO :

Finalização da Aula com Atividade Prática.

Para encerrar a aula, proponha aos alunos a construção de um programa simples com a capacidade de calcular o lucro de 20% sobre o valor do faturamento. Essa atividade tem como objetivo reforçar os conceitos trabalhados em sala, como entrada de dados, cálculos e saída de informações.

```
F = float(input('Digite o faturamento: '))
L = F*0.2
print(L)
exit()
```

AULA 4

ESTRUTURA DE DECISÃO, ESTRUTURA DE REPETIÇÃO

TEMPO:

2 AULAS DE 50 MINUTOS

ORGANIZAÇÃO DA TURMA:

Alunos organizados, sentados à mesa central da biblioteca, com computadores e notebooks em mãos.

TENDÊNCIA EM EDUCAÇÃO MATEMÁTICA:

Matemática e Informática, computadores e linguagem de programação.

INTRODUÇÃO:

Iniciar a aula destacando a importância da tomada de decisões no cotidiano, utilizando exemplos relacionados ao contexto do público ouvinte — como escolher a melhor rota para ir à escola, decidir como utilizar o dinheiro da mesada ou escolher as roupas de acordo com o clima. Esses exemplos mostram que tomamos decisões constantemente com base em condições e objetivos.

Mostrar que essa necessidade de tomar decisões também está presente no contexto computacional. Programas e sistemas precisam avaliar situações e definir qual caminho seguir com base em determinadas condições. É nesse cenário que surgem os algoritmos com estruturas de decisão.

DESENVOLVIMENTO:

1. Lógica matemática, conceitos, princípios e conectivos;
2. Dados Booleano em Python, operações com valor lógico, conectivos;
3. Estruturas de decisão em Python, funções *if, elif e else*;
4. Estruturas de repetição, comandos *while, for e continue* .

CONCLUSÃO :

Finalizar aula construindo algoritmos envolvendo fórmulas matemáticas, usando tanto estruturas que envolvam a tomada de decisões relacionadas a uma condição específica, bem como repetições.

AULA 5

CONTEÚDOS MATEMÁTICOS

TEMPO:

2 AULAS DE 50 MINUTOS

ORGANIZAÇÃO DA TURMA:

Aula expositiva em sala de aula, com uso do quadro e lápis de cores diferentes. Disponibilização de material em PDF com as fórmulas matemáticas relacionadas aos conceitos ministrados.

TENDÊNCIA EM EDUCAÇÃO MATEMÁTICA:

Resolução de problemas: mostrar que as fórmulas matemáticas são ferramentas tecnológicas utilizadas para solucionar problemas.

INTRODUÇÃO:

Levando em consideração o conteúdo programático de Matemática para o terceiro ano do Ensino Médio do estado de Pernambuco, revisar os seguintes assuntos: porcentagem, juros simples, juros compostos, áreas de figuras planas, volumes de sólidos geométricos, média aritmética, mediana e moda.

DESENVOLVIMENTO:

1. Matemática Financeira: porcentagem, juros simples e composto;
2. Geometria Plana: Áreas de figuras planas;
3. Geometria Espacial: Volumes de sólidos geométricos;
4. Estatística: Média Aritmética, Moda e Mediana.

CONCLUSÃO :

Resolução de ficha de exercícios contendo problemas relacionados aos assuntos trabalhados. Solicitar que os alunos apresentem os cálculos realizados por meio das fórmulas matemáticas.

AULA 6

Algoritmos de fórmulas matemáticas em Python

TEMPO:

2 AULAS DE 50 MINUTOS

ORGANIZAÇÃO DA TURMA:

Alunos organizados, sentados à mesa central da biblioteca, com computadores e notebooks em mãos.

TENDÊNCIA EM EDUCAÇÃO MATEMÁTICA:

Matemática e Informática, computadores e linguagem de programação.

INTRODUÇÃO:

Abordar a relevância das fórmulas matemáticas na resolução de problemas, evidenciando como a utilização da linguagem de programação Python pode otimizar esse processo por meio da construção de algoritmos eficientes.

DESENVOLVIMENTO:

1. Recurso educacional Colab e suas funcionalidades;
2. Dividir a turma em grupos para realizar atividade de construção de códigos de programação;
3. Auxiliar os alunos na construção dos algoritmos bem como corrigir erros de sintaxe.

CONCLUSÃO :

Disponibilizar aos grupos um tempo de 10 minutos para apresentarem os resultados e demonstrarem o funcionamento dos algoritmos construídos. Oportunizar aos alunos um momento para que compartilhem suas experiências com o uso da linguagem de programação Python no aprendizado dos conteúdos matemáticos.

5.2 Resultados

Aula 01, 03/10/2024

Devido à ausência de uma sala de informática na instituição, a aula foi realizada nas dependências da biblioteca escolar. Os estudantes foram organizados ao redor da mesa central, utilizando três notebooks disponibilizados para fins didático-práticos, o que possibilitou a interação direta com os recursos tecnológicos durante as atividades propostas.

A aula teve início com uma explanação acerca da relevância do domínio da informática para a realização de atividades cotidianas, bem como para a qualificação profissional e a inserção no mercado de trabalho. Na sequência, realizou-se uma sondagem diagnóstica com os discentes a fim de conhecer suas vivências anteriores com o uso de computadores. Observou-se que a maioria dos alunos não possui computador em casa, sendo que apenas dois declararam estar frequentando um curso de informática.

Na sequência, foi realizada uma breve explanação sobre a evolução histórica dos computadores, enfatizando as transformações tecnológicas ocorridas ao longo das décadas. Destacou-se que os primeiros computadores possuíam grandes dimensões, chegando a ocupar o espaço de uma sala inteira, devido às limitações tecnológicas da época. Com o avanço da ciência e da engenharia computacional, foi possível a miniaturização dos componentes, resultando no desenvolvimento de dispositivos cada vez mais compactos, potentes e portáteis, como os utilizados atualmente.

Na etapa de desenvolvimento da aula, foi projetado material teórico sobre os conceitos fundamentais da informática, utilizando-se de um projetor de imagens para facilitar a compreensão dos alunos. A abordagem inicial focou na explicação da importância das partes físicas e lógicas dos sistemas computacionais, destacando a função da CPU e a relação entre hardware e software. Foram apresentados, também, exemplos de dispositivos de entrada, como o mouse e o teclado, e dispositivos de saída, como a tela e a impressora, ressaltando suas funções específicas e a interação que possibilitam entre o usuário e o sistema.

Na sequência, apresentei os sistemas operacionais mais amplamente utilizados, como Linux, Windows, Android e iOS, discutindo suas principais características e funcionalidades. Enfatizei a importância do sistema operacional como componente essencial para a execução dos programas de computador, destacando seu papel na gestão de recursos do sistema e na coordenação das interações entre o hardware e o software. Também abordei a relevância da interface do usuário nos sistemas operacionais, ressaltando sua versatilidade e interatividade, que contribuem para a eficiência e a usabilidade do sistema.

Para finalizar essa parte da aula, abordamos os diferentes tipos de memória, destacando as distinções entre memória primária e memória secundária. Foi enfatizada a relevância da memória RAM (Random Access Memory) e da memória ROM (Read-Only

Memory), explicando suas funções específicas no sistema computacional. A memória RAM foi caracterizada como uma memória volátil, cuja principal função é armazenar temporariamente dados e instruções durante a execução de programas. Por sua vez, a memória ROM foi descrita como uma memória não volátil, responsável pelo armazenamento permanente de dados essenciais, como o firmware, que é necessário para o processo de inicialização e funcionamento básico do sistema.

Para reforçar a assimilação dos conteúdos, foi proporcionado um momento de interação prática, no qual os alunos foram organizados em três grupos, sendo que cada grupo utilizou um notebook. Durante essa atividade, os estudantes puderam abrir e fechar programas, acessar a internet e criar arquivos, promovendo a aplicação concreta dos conceitos abordados em sala. Além disso, uma atividade foi proposta, a ser realizada durante a aula, com o objetivo de reforçar o aprendizado e estimular a participação ativa dos alunos no processo de ensino-aprendizagem.

Atividade

Criar uma pasta, abrir um documento no word, escrever um texto e salvar documento nessa pasta.

A realização dessa aula resultou em uma melhoria notável na participação, atenção e engajamento dos alunos. Foi possível observar um aumento significativo no interesse dos estudantes ao perceberem a aplicação dos conceitos matemáticos no contexto da informática, o que despertou maior curiosidade e motivação. Todos demonstraram um forte desejo de interagir com os computadores disponibilizados.

Entretanto, é importante destacar que, apesar das limitações estruturais de uma escola pública, o entusiasmo dos alunos foi evidente. Em um ambiente mais apropriado, como uma sala de informática equipada com um computador para cada aluno, os resultados poderiam ser ainda mais eficazes, possibilitando uma aprendizagem mais aprofundada e um aproveitamento máximo dos recursos disponíveis.

Aula 02, 07/10/2024

O objetivo principal desta aula foi proporcionar aos alunos a compreensão do conceito de algoritmos, um termo que tem se tornado cada vez mais relevante devido aos avanços tecnológicos na área da computação.

Embora a proposta da sequência didática sugerisse a realização das aulas na biblioteca, com o uso de computadores, a indisponibilidade tanto da biblioteca quanto dos equipamentos impediu a implementação dessa atividade. Como alternativa, a aula foi realizada na sala de aula, adaptando-se à situação disponível.

Inicialmente fiz a pergunta:

O que é um algoritmo?

Surgiram as seguintes respostas:

‘Um aplicativo’

‘Usado para fazer pesquisa’

‘Tem no celular’

Observou-se que, embora os alunos tivessem uma compreensão preliminar sobre algumas funcionalidades de um algoritmo, não conseguiam definir o termo de maneira precisa. Nesse contexto, para introduzir o conceito de algoritmo, adotei uma abordagem prática. Iniciei com uma pergunta à aluna Emilly, que comercializa bolos de pote durante os intervalos para obter uma renda, questionando-a sobre o procedimento necessário para preparar o bolo. Ela descreveu os passos de forma sequencial e clara, demonstrando domínio do processo. Em seguida, propus a questão sobre o trajeto que os alunos percorriam diariamente para chegar à escola, ressaltando que esse procedimento era familiar para todos. A partir dessas analogias cotidianas, foi possível apresentar a definição formal de algoritmo, estabelecendo uma conexão entre o conceito teórico e as experiências práticas dos alunos.

Definição 5.1. São instruções sequenciais que possibilitam a solução de um problema.

Nesse ponto da aula mostrei que fórmulas matemáticas são algoritmos utilizados para resolver problemas e apresentei algunas exemplos.

Exemplo 5.1.

$a = b \cdot d + r$, algorítimo da divisão euclidiana.

$j = \dfrac{c \cdot i \cdot t}{100}$, fórmula do cálculo do juros simples sobre a aplicação de certo capital c a uma taxa i por um período t.

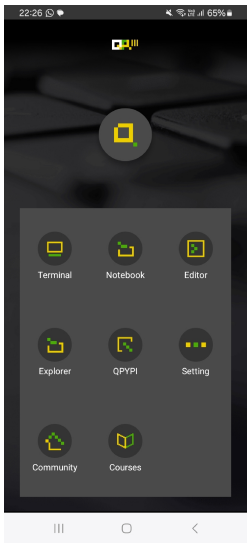
$A = a^2$, fórmula usada para calcular a área de um quadrado com lado medindo a.

Subsequentemente, foi apresentado o conceito de algoritmo no contexto computacional.

Definição 5.2. Um algoritimo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída. É uma ferramenta computacional utilizada para resolver problemas como o de ordenar uma sequência de números.

Na sequência, com o objetivo de proporcionar uma experiência prática na criação de algoritmos, solicitei aos alunos que, sempre que possível, realizassem o download do aplicativo QPython em seus smartphones pessoais. Contudo, nem todos conseguiram atender à solicitação devido à indisponibilidade de acesso à internet em alguns dispositivos.

Figura 5 – Aula 02



Fonte: Elaborado pelo autor, 2024.

Durante a aula, foi demonstrado aos alunos como atribuir variáveis e utilizar as funções print e input, com ênfase na interação entre o programa e o usuário. Também foi abordado o uso de dados do tipo string, detalhando sua manipulação e aplicabilidade na construção de algoritmos.

Para fixar o conteúdo, foi proposta uma atividade na qual os alunos deveriam criar um algoritmo no editor que exibisse uma mensagem de boas-vindas. Os resultados foram, em sua maioria, satisfatórios, embora alguns alunos tenham apresentado dificuldades, especialmente no que se refere ao uso correto de dados do tipo string, com a omissão das aspas, e na sintaxe da função input. Observou-se, ainda, uma dificuldade em compreender o mecanismo de automação: alguns alunos, ao invés de inserir a resposta na função input para a pergunta "Qual o seu nome?", colocaram diretamente seu próprio nome no código. Para solucionar essas dificuldades, foi feita uma nova explicação sobre a função input, esclarecendo sua finalidade de solicitar dados ao usuário durante a execução do programa.

Foi possível identificar um engajamento crescente por parte dos alunos, evidenciado pelo interesse despertado durante a construção dos códigos, o que demonstrou uma maior compreensão sobre o funcionamento da automação e a aplicação prática dos conceitos de programação.

Aula 03, 14/10/2024

Para iniciar a aula sobre a linguagem de programação Python, foi realizada uma breve apresentação histórica, destacando seu criador e o desenvolvimento da linguagem ao longo do tempo. Foi abordada a relevância da linguagem Python no contexto da comunicação entre programadores e máquinas, enfatizando sua simplicidade e eficácia. Também foi discutida a linguagem binária, composta pelos valores ’0’ e ’1’, explicando

sua base no funcionamento fundamental dos computadores. Por fim, ressaltou-se que a linguagem Python, por ser de alto nível, se aproxima da linguagem humana, facilitando a aprendizagem e tornando a programação mais acessível aos iniciantes.

Na sequência, apresentei o ambiente de desenvolvimento da linguagem Python, explicando o procedimento para sua instalação no sistema operacional Windows. Abordei os tipos de dados disponíveis na linguagem, destacando a diferença entre valores numéricos inteiros e reais. Em seguida, introduzi as operações aritméticas, esclarecendo os operadores aritméticos utilizados em Python para realizar cálculos. Para concluir essa parte teórica, recapitulei o uso das funções print e input, abordando sua função na interação com o usuário e na exibição de resultados no ambiente de programação.

Para consolidar os conceitos teóricos apresentados, introduzi o recurso Google Colab, uma plataforma virtual que possibilita a construção e execução de códigos Python de maneira interativa. Proporcionei aos alunos a oportunidade de aplicar os conceitos aprendidos, permitindo que praticassem a programação no referido ambiente. Ao final, foi proposto um exercício, com a finalidade de reforçar os conhecimentos adquiridos e estimular a aplicação prática dos mesmos.

Exercício

Construir um algorítimo de programação para calcular 20% sobre o valor de determinado faturamento.

Coloquei no quadro o modelo de uma fórmula para efetuar tal operação e pedi para que eles continuassem o algorítimo no colab.

$$L = F * 0.2$$

Nesta questão, as variáveis F e L representam, respectivamente, o faturamento e o lucro. Os conceitos abordados envolvem porcentagem, multiplicação por números racionais, atribuição de variáveis, utilização de dados numéricos do tipo float, e o uso das funções print e input.

Mais uma vez, alguns erros foram apresentados na construção do algoritmo, principalmente devido ao esquecimento de caracteres. No entanto, não foram observadas grandes dificuldades na elaboração do código.

A seguir a figura apresenta um dos resultados:

Figura 6 – Exercício

Fonte: Elaborado pelo autor, 2024.

Aula 04, 04/11/2024

Essa aula foi ministrada nas três turmas do 3º ano: 3º E, 3º G e 3º F, com duração de duas aulas de 50 minutos em cada turma. Desta vez, consegui utilizar a biblioteca, que estava com uma estrutura melhor devido aos aulões preparatórios para o ENEM que estavam ocorrendo nesse ambiente.

No início da aula, foram apresentadas algumas definições fundamentais de lógica matemática, enfatizando sua relevância para a compreensão das estruturas de decisão na linguagem Python, bem como para o entendimento do uso de dados booleanos.

Abordei as funções if, elif e else dentro da estrutura de decisão. Apresentei as definições dos tipos de execução: condicional simples, alternativa e encadeamento condicional, além de mostrar alguns exemplos de códigos de programação. Como material teórico, utilizei ROJAS, Alexandre; KOSTIN, Sergio. Introdução à Programação Python. Logo em seguida, no Google Colab, criamos códigos e testamos suas funcionalidades.

Figura 7 – Estrutura de decisão

Fonte: Elaborado pelo autor, 2024.

Exemplo 5.2.

Na construção desse algorítimo trabalhamos conceitos matemáticos relacionados com a reta Real.

Verifiquei que, apesar do uso de recursos tecnológicos, ainda houve dispersão por parte de um ou outro aluno, como se o tema não despertasse interesse. Por outro lado, uma parte da turma contribuiu mais com a aula e mostrou-se mais participativa. Do total de 16 alunos presentes, 10 demonstraram interesse, o que corresponde a 62,5

Por fim, com o objetivo de fixar os conceitos trabalhados e os conteúdos matemáticos relacionados a números inteiros, números reais, medidas e razão entre grandezas, dividi a turma em dois grupos e propus a seguinte atividade.

Atividade

Um endocrinologista deseja controlar a saúde de seus pacientes e, para isso se utiliza de um índice de massa corporal (IMC). Sabendo-se que o IMC é calculado através da fórmula abaixo:

$$\text{IMC} = \frac{\text{Peso}}{\text{Altura}^2} \tag{5.1}$$

Onde o peso é dado em quilogramas (kg) e a altura em metros (m). Escreva um programa em Python que apresente o nome do paciente e sua respectiva faixa de risco, com base na seguinte tabela:

Tabela 8 – Classificação do Índice de Massa Corporal (IMC)

IMC	Faixa de risco
Abaixo de 20	Abaixo do peso
A partir de 20 até 25	Normal
Acima de 25 até 30	Excesso de peso
Acima de 30 até 35	Obesidade
Acima de 35	Obesidade mórbida

Fonte: Adaptado de tabelas de classificação do IMC.

Para que todos pudessem participar da atividade, cada aluno do grupo ficou responsável, ao menos, pela construção de uma linha de execução. Quando ocorria um erro de sintaxe, eu auxiliava na correção, e, por meio da prática, fomos construindo os aprendizados juntos.

Percebi que, apesar das dificuldades estruturais e da carência de recursos tecnológicos, uma vez que nesta aula dispúnhamos apenas de três computadores, o resultado dessa experiência foi uma aula participativa, com troca de experiências e protagonismo dos alunos. Foi possível trabalhar conceitos matemáticos por meio de códigos de programação, trazendo um sentido prático ao conteúdo abordado. Em resumo, percebi que essa abordagem é um recurso viável para o ensino de conceitos matemáticos.

Exemplo 5.3. Aqui é apresentado o modelo de algoritmo construído para solucionar o problema:

```
# Solicita os dados do usuário
nome = input(‘Entre com seu nome: ’)
# Corrigindo a conversão para float
peso = float(input(‘Entre com seu peso (kg): ’))
altura = float(input(’Entre com sua altura (m): ’))
# Calcula o Índice de Massa Corporal (IMC)
imc = peso / (altura ** 2)
# Determina a faixa de risco com base no IMC
if imc < 20:
    faixa_risco = ‘abaixo do peso’
elif 20 <= imc < 25:
    faixa_risco = ‘com peso normal’
elif 25 <= imc <= 30:
    faixa_risco = ‘com excesso de peso’
elif 30 < imc <= 35:
    faixa_risco = ‘obeso’
else:
    faixa_risco = ‘com obesidade mórbida’
# Exibe o resultado
print(f‘{nome} está {faixa_risco}.’)
```

Não foi possível iniciar os conceitos de estrutura de repetição devido ao esgotamento do tempo de aula. A continuidade do plano de aula 04 ficou para o próximo encontro.

Aula 04, 18/11/2024

Dando continuidade ao plano de aula 04, realizamos a atividade na biblioteca da escola, onde os alunos utilizaram notebooks. Além disso, foi utilizado um projetor de imagem para facilitar a visualização do material teórico pertinente à aula.

No início da aula, apresentei os conceitos relacionados às estruturas de repetição, abordando as funções while e for, além das diferentes maneiras de interromper os loops. Para tornar esses conceitos mais acessíveis, utilizei o exemplo de soma contínua de valores.

$$1 + 6 + 7 + 2 + ...$$

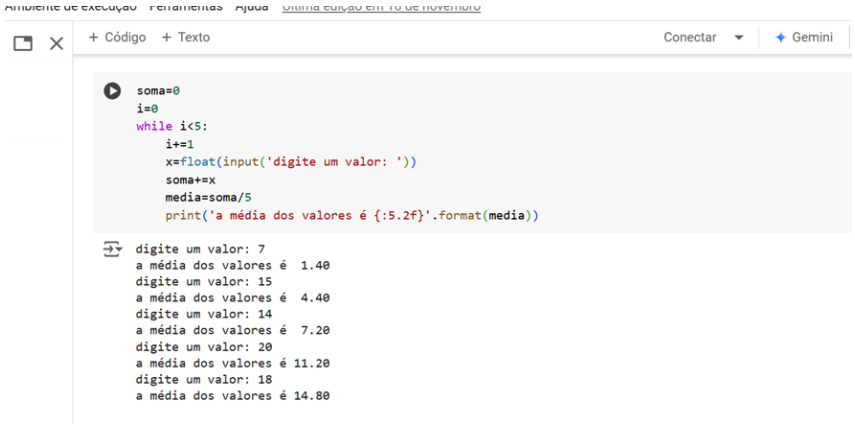
Expliquei que é possível criar programas utilizando a função while para coletar informações de maneira contínua, até que uma condição de interrupção seja atingida. Um

exemplo ilustrativo apresentado foi o preenchimento de uma lista até atingir a quantidade determinada de pessoas, devido ao limite de vagas disponíveis.

Em seguida, no ambiente Google Colab, os alunos construíram algoritmos utilizando a função while.

Dado que os conceitos de média aritmética já haviam sido trabalhados na unidade, os alunos desenvolveram um algoritmo para calcular a média aritmética de cinco números.

Figura 8 – Estrutura de repetição



Fonte: Plataforma google colab, elaborado pelo autor, 2024.

Os resultados obtidos com a abordagem que integrou conceitos matemáticos ao uso de ferramentas tecnológicas indicaram um aumento significativo no interesse dos alunos pela aula, bem como uma participação mais ativa nas atividades propostas. Essa metodologia também despertou nos alunos o desejo de continuar com aulas desse tipo. Foi possível observar a compreensão da variação da média à medida que novos valores eram inseridos no cálculo.

Os alunos apresentaram poucas dificuldades na construção dos códigos. A principal dificuldade observada foi a compreensão do objetivo da linguagem de programação, que visa automatizar processos. Muitos alunos, por exemplo, já inseriam valores numéricos diretamente na string "digite um número:", sem utilizar a função apropriada para a entrada de dados. Expliquei que a função do algoritmo é fornecer comandos à máquina para realizar tarefas de forma automatizada.

A proposta de ensinar conceitos matemáticos por meio da linguagem de programação revelou-se uma abordagem satisfatória e eficaz.

Aula 05 e Aula 06

Em virtude do calendário letivo restrito no final do ano, que incluiu a preparação dos alunos para o ENEM, as avaliações do SAEPE e a semana de provas no início de dezembro, não foi possível a aplicação dos planos de aula 05 e 06 no período previsto.

6 CONCLUSÃO

Conforme proposto neste trabalho, a linguagem de programação Python foi empregada como uma ferramenta didática no ensino de matemática. Durante a aplicação dessa metodologia, foram identificadas algumas dificuldades, como a falta de acesso a computadores pelos alunos em suas residências e a inexistência de um laboratório de informática na escola. Além disso, o número de aulas precisou ser ajustado em função da carga horária reduzida no final do ano letivo, devido à realização de provas externas (ENEM, SAEPE, SIMULADO DO SAEPE) e das avaliações do 4º bimestre.

O uso de ferramentas tecnológicas despertou um maior interesse dos alunos pelas aulas, tornando-os mais participativos e atentos ao conteúdo abordado. Alunos que demonstravam desinteresse nas aulas tradicionais, mas possuíam afinidade com a tecnologia, se destacaram ao compartilhar seus conhecimentos, auxiliando os colegas na execução das atividades.

O ensino das noções fundamentais de Python, devido à simplicidade de sua linguagem, foi bem recebido pelos alunos. Não houve grandes dificuldades para que compreendessem os tipos de dados, a construção de códigos e a funcionalidade de algumas funções da linguagem.

A prática de programação, adotada como metodologia ativa, favoreceu o engajamento dos alunos na resolução de problemas, na construção de algoritmos para representar fórmulas matemáticas e na exibição de mensagens. Esse processo permitiu a correção de erros conceituais e a consolidação do conhecimento. Embora alguns erros recorrentes tenham surgido, como o esquecimento de caracteres ou a falta de compreensão sobre o funcionamento da automação (em especial na função input, onde, ao invés de solicitar dados ao usuário, os alunos forneciam as informações diretamente no código), esses desafios foram superados ao longo da atividade.

O ensino da matemática, por meio desse modelo, mostrou-se imersivo, criando um ambiente de aprendizado dinâmico e interativo. Conceitos matemáticos como números reais, razão, índice, porcentagem e média aritmética adquiriram um significado prático e contextualizado. Além disso, a construção dos algoritmos para as fórmulas matemáticas contribuiu para uma abordagem mais eficaz das definições dos conteúdos.

Com o objetivo de fortalecer os argumentos deste trabalho, apresenta-se como recurso educacional para o ensino de conteúdos matemáticos calculadoras interativas, elaboradas por meio da aplicação da linguagem de programação Python e do desenvolvimento de algoritmos. Tal ferramenta possibilita uma abordagem prática e dinâmica dos conceitos matemáticos, promovendo a integração entre tecnologia e aprendizado.

REFERÊNCIAS

ALENCAR FILHO, E. **Iniciação à lógica matemática**. São Paulo: Nobel, 2002.

ASHOKA; ALANA. **Protagonismo: a potência de ação da comunidade escolar**. São Paulo: Ashoka; Alana, 2017.

BRASIL. **Lei n. 9.394, de 20 de dezembro de 1996**. Estabelece as diretrizes e bases da educação nacional. Brasília, DF: Presidência da República, 1996. Disponível em: `http://www.planalto.gov.br/ccivil_03/leis/l9394.htm`. Acesso em: 23 ago. 2024.

BRASIL. **Base Nacional Comum Curricular**. Brasília, DF: MEC/SEB, 2018. Disponível em: `http://basenacionalcomum.mec.gov.br/`. Acesso em: 23 ago. 2024.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos: teoria e prática**. Tradução de Vandenberg D. de Souza. Rio de Janeiro: Elsevier, 2002.

DOMINGUES, Higino H.; IEZZI, Gelson. **Álgebra moderna**. 4. ed. reform. São Paulo: Atual, 2003.

HEFEZ, Abramo. **Aritmética**. Rio de Janeiro: SBM, 2016.

HOUAISS, Antônio. **Minidicionário da língua portuguesa**. Rio de Janeiro: Objetiva, 2010.

KHAN ACADEMY. **Insertion sort**. Disponível em: `https://pt.khanacademy.org/computing/computer-science/algorithms/insertion-sort/a/insertion-sort`. Acesso em: 23 ago. 2024.

KENSKI, V. M. **Educação e tecnologias: o novo ritmo da informação**. Campinas: Papirus, 2010.

LIMA, E. L.; CARVALHO, P. C. P.; WAGNER, E.; MORGADO, A. C. **A matemática do ensino médio: volume 1**. Rio de Janeiro: SBM, 1998.

LIMA, Elon Lages. **Números e funções reais**. Rio de Janeiro: SBM, 2013.

MAGALHÃES, Alexandre Luís Levada. **Fundamentos de lógica matemática**. São Carlos: UAB-UFSCar, 2011.

MATHEUS, Eric. **Python crash course**. São Paulo: Novatec, 2016.

MORGADO, Augusto Cesar; CEZAR, Paulo Pinto Carvalho; CARVALHO, João Bosco Pitombeira; FERNANDEZ, Pedro. **Análise combinatória e probabilidade**. 9. ed. Rio de Janeiro: SBM, 1991.

MORGADO, Augusto Cesar; CEZAR, Paulo Pinto Carvalho. **Matemática discreta**. Rio de Janeiro: SBM, 2013.

MUNIZ, Antônio Caminha. **Fundamentos de cálculo**. 2. ed. Rio de Janeiro: SBM, 2022.

OPENAI. **ChatGPT (modelo GPT-5)**. 2026. Disponível em: https://chat.openai.com. Acesso em: 26 fev. 2026.

PRENSKY, Marc. **Digital natives, digital immigrants**. *On the Horizon*, v. 9, n. 5, p. 1–6, 2001.

ROJAS, Alexandre; KOSTIN, Sergio. **Introdução à programação Python**. Rio de Janeiro: Ciência Moderna, 2018.

SANTOS, Maria. **Educação e tecnologia: desafios e perspectivas**. São Paulo: Editora XYZ, 2020.

VALENTE, José Armando. **O computador na sociedade do conhecimento**. Campinas: UNICAMP/NIED, 2016.

WAGNER, Eduardo; MORGADO, Augusto Cesar; LIMA, Elon Lages; CARVALHO, Paulo Cezar Pinto. **A matemática do ensino médio: volume 1**. 8. ed. Rio de Janeiro: SBM, 2005.

WAGNER, Eduardo; MORGADO, Augusto Cesar; LIMA, Elon Lages; CARVALHO, Paulo Cezar Pinto. **A matemática do ensino médio: volume 3**. 7. ed. Rio de Janeiro: SBM, 2016.