Universidade Federal do Maranhão
Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação
Centro de Ciências Exatas e Tecnologia
Programa de Mestrado Profissional em Matemática em Rede Nacional - PROFMAT

Alice Santos de Araújo

Algoritmo e Pensamento Computacional:
python como ponte entre matemática e programação no ensino médio

São Luís - MA
2025

Universidade Federal do Maranhão
Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação
Centro de Ciências Exatas e Tecnologia
Programa de Mestrado Profissional em Matemática
em Rede Nacional - PROFMAT

Alice Santos de Araújo

ALGORITMO E PENSAMENTO COMPUTACIONAL:
PYTHON COMO PONTE ENTRE MATEMÁTICA E PROGRAMAÇÃO NO ENSINO MÉDIO

Dissertação apresentada à banca do Programa de Mestrado Profissional em Matemática (PROFMAT) da UFMA como requisito parcial para a obtenção do título de Mestre em Matemática.

Este exemplar corresponde à versão final da dissertação defendida pela aluna Alice Santos de Araújo e aprovada pela comissão julgadora.

Prof. Dr. Luís Fernando Coelho Amaral
(Orientador)

São Luís - MA
2025

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Araújo, Alice Santos de.
Algorimo e Pensamento Computacional : python como ponte entre Matemática e Programação no Ensino Médio / Alice Santos de Araújo. - 2025.
92 f.

Orientador(a): Luís Fernando Coelho Amaral.
Dissertação (Mestrado) - Programa de Pós-graduação em Rede - Matemática em Rede Nacional/ccet, Universidade Federal do Maranhão, São Luís, 2025.

1. Programação. 2. Python. 3. Bncc. 4. Gráficos. 5. Competências do Século Xxi. I. Amaral, Luís Fernando Coelho. II. Título.

Universidade Federal do Maranhão
Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação
Centro de Ciências Exatas e Tecnologia
Programa de Mestrado Profissional em Matemática
em Rede Nacional - PROFMAT

Dissertação apresentada à banca do Programa de Mestrado Profissional em Matemática (PROFMAT) da UFMA como requisito parcial para a obtenção do título de Mestre em Matemática.
Área de Concentração: Matemática na Educação Básica
Aprovada em: 19 de setembro de 2025

Prof. Dr. Luís Fernando Coelho Amaral
Orientador
Universidade Federal do Maranhão (UFMA)

Profª. Dra. Valeska Martins de Souza
Universidade Federal do Maranhão (UFMA)

Prof. Dr. João Coelho Silva Filho
Universidade Estadual do Maranhão (UEMA)

São Luís - MA
2025

AGRADECIMENTOS

Ao Programa de Mestrado Profissional em Matemática em Rede Nacional (PROFMAT), pela oportunidade de formação acadêmica e profissional, que me proporcionou uma nova forma de compreender a Matemática e refletir sobre como posso contribuir com os estudantes da Educação Básica.

Ao meu orientador, Prof. Dr. Luís Fernando, por me acompanhar desde a graduação e ensinar, através do exemplo, como ser um excelente professor, além das orientações atenciosas e apontamentos precisos que foram fundamentais para a construção deste trabalho.

Aos professores e colegas do curso, pelas discussões e contribuições que ampliaram meus horizontes e transformaram a maneira como enxergo a Matemática.

À minha mãe, pelo contínuo incentivo e sustento ao longo da vida, e por me ensinar a lição mais valiosa: o estudo como essência da vida. Estendo esse agradecimento à minha família, em especial à minha irmã Rayssa, pelo carinho, apoio e compreensão.

À Letícia, pela contribuição essencial desde os primeiros dias de minha prática pedagógica, pela escuta atenta, pelas ideias sempre pertinentes e pela ajuda generosa na escrita deste trabalho, além de sua amizade sincera.

À Jéssica, que esteve ao meu lado durante toda esta caminhada, cuja compreensão, apoio constante e presença amorosa foram fundamentais para que eu chegasse até aqui.

Por fim, a todos que, de forma direta ou indireta, contribuíram para a realização deste trabalho, deixo minha sincera gratidão.

RESUMO

Este trabalho apresenta propostas didáticas para uma eletiva do Ensino Médio que integra conteúdos de Matemática com noções introdutórias de programação. A proposta foi desenvolvida em consonância com as diretrizes da Base Nacional Comum Curricular (BNCC) e com as competências e habilidades demandadas na formação do estudante do século XXI. As atividades elaboradas buscam explorar, de forma contextualizada e interdisciplinar, a construção e a interpretação de gráficos e tabelas, promovendo o desenvolvimento do raciocínio lógico, do pensamento computacional e da autonomia na resolução de problemas. A linguagem de programação é utilizada como ferramenta de apoio à aprendizagem matemática, aproximando os estudantes de práticas contemporâneas e conectadas ao mundo digital. A proposta está alinhada à parte diversificada do currículo do Novo Ensino Médio, contribuindo para a oferta de percursos formativos flexíveis, significativos e integrados a diferentes áreas do conhecimento.

Palavras-chave: Programação; Python; BNCC; Gráficos; Competências do século XXI.

ABSTRACT

This work presents a didactic proposal for an elective course in Brazilian upper secondary education that integrates Mathematics content with introductory programming concepts. The proposal was developed in alignment with the guidelines of the *Base Nacional Comum Curricular* (BNCC) and the competencies and skills required for the education of 21st-century students. The activities are designed to explore, in a contextualized and interdisciplinary manner, the construction and interpretation of graphs and tables, fostering the development of logical reasoning, computational thinking, and autonomy in problem solving. Programming is used as a supporting tool for learning Mathematics, bringing students closer to contemporary, digitally connected practices. The proposal is consistent with the diversified part of the Novo Ensino Médio curriculum, contributing to the provision of flexible and meaningful learning pathways integrated across different areas of knowledge.

Key-words: Programming; Python; BNCC; Graphs; 21st-century skills.

SUMÁRIO

1 INTRODUÇÃO

Em uma sociedade orientada por dados, a capacidade de interpretar, produzir e implementar gráficos e tabelas configura-se como uma competência essencial à formação cidadã. A leitura de notícias, a compreensão de fenômenos econômicos, sociais e ambientais e a participação crítica em debates públicos são exemplos de situações em que o letramento em dados se faz necessário no cotidiano dos estudantes.

Nesse contexto, o Novo Ensino Médio, instituído pela Lei n.º 13.415/2017 (Brasil, 2017) e implementado em todo o território nacional a partir de 2022, propõe mudanças estruturais no currículo, incluindo a adoção dos itinerários formativos e das eletivas. Essas mudanças visam mitigar problemas históricos da educação brasileira e aproximar os conteúdos escolares das demandas do século XXI. Espera-se, assim, que os professores sejam capazes de integrar novas metodologias, recursos tecnológicos e práticas contextualizadas ao processo de ensino-aprendizagem.

Diante desse cenário, torna-se essencial o desenvolvimento de propostas didáticas que dialoguem com os princípios da Base Nacional Comum Curricular (BNCC), a qual preconiza, entre outras competências, o pensamento científico, crítico e criativo, a cultura digital, a comunicação e a construção do projeto de vida dos estudantes (Brasil, 2018).

A linguagem de programação Python surge como uma ferramenta promissora para o ensino de Matemática, especialmente no que se refere à representação e interpretação de dados. Este trabalho foi concebido em consonância com as diretrizes da BNCC e com o objetivo de utilizar as tecnologias digitais como mediadoras do conhecimento, estimulando a autonomia, a criatividade e a capacidade de resolução de problemas. Busca-se, com isso, oferecer suporte a educadores da Educação Básica interessados em integrar a programação ao ensino de Matemática.

A realização deste estudo justifica-se pela crescente necessidade de integrar a cultura digital ao ensino de Matemática na Educação Básica, em consonância com as demandas formativas do século XXI. Em uma sociedade cada vez mais orientada por dados, torna-se indispensável que os estudantes desenvolvam competências ligadas à interpretação e à produção de informações em diferentes linguagens, incluindo a programação. Do ponto de vista educacional, a escassez de materiais que articulem matemática e programação, voltados especificamente para professores da educação básica, representa uma lacuna que este trabalho busca minimizar. Além disso, ao propor uma metodologia que aproxima conceitos matemáticos da linguagem Python, este estudo contribui não apenas para a prática docente, mas também para o campo científico, oferecendo subsídios a novas pesquisas sobre a integração entre tecnologias digitais e processos de ensino-aprendizagem.

Dessa forma, este trabalho apresenta uma proposta didática de uma eletiva voltada para manipulação de gráficos e tabelas no Ensino Médio, utilizando a linguagem de programação Python como ferramenta pedagógica para facilitar a compreensão de conceitos matemáticos. A proposta visa trabalhar com problemas contextualizados, conferindo significado aos conteúdos abordados, ao promover o desenvolvimento de habilidades essenciais à formação integral do estudante da educação básica.

A metodologia adotada será de natureza qualitativa, com base em revisão bibliográfica, voltada à proposição de atividades didáticas para manipulação de gráficos e tabelas,

norteada pelas competências e habilidades da BNCC. A construção da proposta se dará a partir da análise de referenciais teóricos, documentos curriculares e experiências pedagógicas.

A presente dissertação está organizada em três seções, além desta introdução. A seção 2 apresenta os fundamentos teóricos relacionados ao Novo Ensino Médio e à Base Nacional Comum Curricular (BNCC), com ênfase na criação de eletivas e na formação integral do estudante. A seção 3 discute a linguagem de programação Python, abordando seus conceitos fundamentais e suas potencialidades no ensino das diferentes representações matemáticas. A seção 4 propõe algumas atividades didáticas voltadas para o Ensino Médio, fundamentadas na programação em Python e direcionadas ao desenvolvimento da habilidade de interpretação e construção de gráficos e tabelas. Por fim, as considerações finais destacam as contribuições do estudo, suas limitações e sugestões para pesquisas futuras.

2 MATEMÁTICA E PROGRAMAÇÃO NO NOVO ENSINO MÉDIO

A presente dissertação insere-se no contexto das transformações recentes impulsionadas pela Reforma do Ensino Médio no Brasil. Nesta seção, discute-se o Novo Ensino Médio, destacando o papel das disciplinas eletivas como componentes fundamentais da nova estrutura curricular. Além disso, argumenta-se que a integração entre Matemática e Programação pode oferecer respostas relevantes às demandas formativas do século XXI, ao promover o desenvolvimento de competências como o pensamento computacional, a resolução de problemas, o letramento digital, a criatividade, bem como a capacidade de interpretar e produzir dados.

2.1 Mudanças Legais na Educação Básica Contemporânea

A Reforma do Ensino Médio, instituída pela Lei nº 13.415/2017, promoveu alterações significativas na organização da educação básica brasileira, incluindo mudanças na Lei de Diretrizes e Bases da Educação Nacional (LDB), Lei nº 9.394/1996, e a criação da Política de Fomento à Implementação de Escolas de Ensino Médio Integral. Conhecida como Novo Ensino Médio, a reforma busca uma reorganização curricular que prioriza a formação integral dos estudantes, o desenvolvimento de competências e habilidades, e a personalização dos percursos formativos.

Conforme o disposto no Art.35-B da Lei nº 9.394/1996, incluído pela Lei nº 14.945/2024:

> Serão asseguradas aos estudantes oportunidades de construção de projetos de vida, em perspectiva orientada pelo desenvolvimento integral, nas dimensões física, cognitiva e socioemocional, pela integração comunitária no território, pela participação cidadã e pela preparação para o mundo do trabalho, de forma ambiental e socialmente responsável (Brasil, 1996).

Assim, o currículo do Novo Ensino Médio foi reorganizado em duas partes principais: a formação geral básica e os itinerários formativos. Essa estrutura prevê uma articulação entre a Base Nacional Comum e uma parte diversificada, destinada a complementar o núcleo comum. A parte diversificada tem como objetivo assegurar que os currículos reflitam as características regionais e locais da sociedade, cultura, economia e as especificidades dos educandos de cada sistema de ensino e de cada estabelecimento escolar (Brasil, 1996, art.26).

A Base Nacional Comum Curricular (BNCC), por sua vez, é um conjunto de diretrizes que orientam o currículo das escolas das redes públicas e privadas em todo o Brasil. Sua finalidade é promover a elevação da qualidade do ensino no país, por meio de uma estrutura curricular comum e obrigatória, respeitando, contudo, a autonomia assegurada pela Constituição Federal aos entes federados e às escolas (Brasil, 2018).

Na formação geral básica, o estudante tem garantida a articulação entre diferentes áreas do conhecimento. Essa etapa segue uma estrutura "tradicional", que contempla componentes curriculares que estão organizados em áreas do conhecimento, tais como Linguagens e suas Tecnologias, Ciências Humanas e Sociais e suas Tecnologias, Ciências da Natureza e suas Tecnologias, e Matemática e suas Tecnologias. As competências e

habilidades previstas na Base Nacional Comum Curricular (BNCC) constituem o eixo central dessa formação (Brasil, 2018).

Por outro lado, os Itinerários Formativos são elementos fundamentais e estratégicos para assegurar a flexibilidade do currículo no Novo Ensino Médio. Eles oferecem aos estudantes a possibilidade de construir trajetórias educacionais alinhadas aos seus interesses pessoais e projetos de vida.

> a oferta de diferentes itinerários formativos pelas escolas deve considerar a realidade local, os anseios da comunidade escolar e os recursos físicos, materiais e humanos das redes e instituições escolares de forma a propiciar aos estudantes possibilidades efetivas para construir e desenvolver seus projetos de vida e se integrar de forma consciente e autônoma na vida cidadã e no mundo do trabalho (Brasil, 2018, p.478).

Para tanto, os itinerários devem adotar metodologias que promovam a autonomia, o protagonismo juvenil e o engajamento crítico. Tais metodologias devem estar organizadas em torno de um ou mais dos seguintes eixos estruturantes, conforme disposto na figura 1:

Figura 1 — Fluxograma dos eixos estruturantes propostos no Novo Ensino Médio

Fonte: Elaboção própria (2025).

Dessa forma, os itinerários são estruturados com o objetivo de construir uma ponte entre os conteúdos trabalhados em sala de aula e as demandas do mundo contemporâneo. Investigar, idealizar e realizar projetos que gerem impacto na sociedade, no meio ambiente e na vida do próprio estudante são ações que devem permear todo o Ensino Médio. Assim, busca-se promover a articulação entre os diversos conteúdos curriculares, estabelecendo vínculos com o mundo do trabalho, com a prática social e com o exercício da cidadania.

Cada sistema de ensino tem autonomia para escolher o modelo pedagógico que será adotado, considerando as diretrizes nacionais, a legislação vigente e as características locais de sua rede de ensino. No contexto da rede estadual de ensino do Maranhão, à qual a autora desta proposta está vinculada, o modelo pedagógico escolhido foi a Escola da Escolha, desenvolvido pelo Instituto de Corresponsabilidade pela Educação (ICE). Esse modelo tem como objetivo principal proporcionar uma formação integral, alinhada aos princípios do protagonismo juvenil e à construção de projetos de vida. Portanto, ao

compreender as especificidades do Novo Ensino Médio, é possível avançar para a análise de sua implementação.

2.2 As Eletivas e a Escola da Escolha

Nesta seção, apresentaremos o modelo pedagógico Escola da Escolha, desenvolvido pelo Instituto de Corresponsabilidade pela Educação (ICE). Será destacado de que forma esse modelo contribui para o desenvolvimento da autonomia e do protagonismo juvenil, fundamentos essenciais do Novo Ensino Médio. Além disso, discutiremos as especificidades do modelo no que se refere à construção de disciplinas eletivas, com foco em como essas podem ser planejadas de modo a contribuir efetivamente para os itinerários formativos.

De acordo com o registro presente no *Caderno de Formação do ICE: Concepção do Modelo Pedagógico* (ICE, 2021b), o modelo teve origem na cidade do Recife, em Pernambuco, a partir da iniciativa de um ex-aluno do Ginásio Pernambucano. Ao retornar à escola, que fora um importante centro de referência educacional em décadas anteriores, ele encontrou um cenário de profunda decadência estrutural e pedagógica. Motivado por essa realidade, aliado ao fomento da iniciativa privada e com a permissão do poder público, iniciou-se um processo de reformulação da escola, que envolveu tanto a reestruturação física quanto a reconstrução de seu projeto pedagógico. Esse movimento deu origem à proposta de um novo modelo de escola, comprometido com uma educação mais significativa, democrática e voltada para o desenvolvimento integral dos estudantes.

Assim, em 2003, foi criado o Instituto de Corresponsabilidade pela Educação (ICE), uma entidade sem fins lucrativos, formada por representantes da sociedade civil, com o propósito de contribuir para a melhoria da educação básica no Brasil por meio de soluções educacionais inovadoras. Alinhado às diretrizes do Novo Ensino Médio e da BNCC, o modelo Escola da Escolha expandiu-se para diversas regiões do país, estando atualmente presente na maioria dos estados brasileiros. Com o jovem e seu projeto de vida como elementos centrais, o modelo propõe uma estrutura pedagógica que valoriza a parte diversificada do currículo, apresentando caminhos concretos para a implementação das transformações propostas pela reforma do Ensino Médio.

> Este modelo de educação oferece aos estudantes não apenas uma Formação Acadêmica de Excelência que lhe assegura o pleno domínio do conhecimento, mas também agrega a Formação para a Vida, ampliando suas referências sobre valores e ideais. Além disso, os apoia para enfrentar os desafios do mundo contemporâneo por meio de competências necessárias para encarar os desafios impostos pelo século XXI (ICE, 2023, p. 6).

Dessa forma, o modelo pedagógico da Escola da Escolha propõe um currículo integrado, composto por componentes curriculares tanto da Base Nacional Comum Curricular (BNCC) quanto da parte diversificada. Em consonância com a perspectiva de formação integral, a parte diversificada não deve ser compreendida como um complemento, mas sim como parte constituinte do currículo escolar.

A efetivação dessa proposta ocorre por meio da criação de componentes curriculares próprios, como as eletivas, que são planejadas para atender aos interesses, às necessidades e ao projeto de vida dos estudantes. É nesse contexto que se inserem as chamadas Metodologias de Êxito, conforme descrito a seguir:

> As Metodologias de Êxito são componentes curriculares da Parte de Formação Diversificada que exercem o papel de articuladores entre o mundo acadêmico e as práticas sociais, ampliando, enriquecendo e diversificando o repertório de experiências e conhecimentos dos estudantes (ICE, 2021a, p. 16).

Nessa perspectiva, as Metodologias de Êxito constituem a forma como se efetiva, na prática, o que é proposto nos Itinerários Formativos, considerando as singularidades e especificidades de cada realidade escolar. No modelo da Escola da Escolha, são previstas diferentes Metodologias de Êxito, tais como: projeto de vida, pós-médio, protagonismo, estudo orientado, práticas experimentais, tutoria, projeto de corresponsabilidade social e eletivas. Essas metodologias são incorporadas ao currículo e representam a estratégia pedagógica por meio da qual o modelo da Escola da Escolha concretiza as demandas de formação da Parte Diversificada. Dentre essas diversas possibilidades, este trabalho volta-se especificamente para as eletivas, compreendidas como espaços privilegiados para o desenvolvimento de experiências significativas e o aprofundamento de interesses pessoais e acadêmicos dos estudantes.

Assim, as eletivas são componentes curriculares da Parte da Formação Diversificada, inseridas nas chamadas Metodologias de Êxito, conforme previsto no modelo pedagógico da Escola da Escolha. Elas desempenham um papel fundamental na efetivação dos princípios do Novo Ensino Médio, especialmente no que se refere à flexibilização dos percursos formativos, considerando o projeto de vida de cada estudante. Em outras palavras, as eletivas possibilitam que cada aluno construa seu próprio itinerário formativo, de acordo com seus interesses, aspirações e escolhas.

Nesse sentido, as eletivas, em sua concepção, devem articular-se com as demandas previstas na Base Nacional Comum Curricular (BNCC), buscando enriquecer o repertório dos estudantes, diversificar as metodologias de ensino, estimular o protagonismo juvenil e favorecer a resolução de situações-problema. Além disso, é fundamental que essas disciplinas sejam organizadas em torno de temas interdisciplinares, de modo que os estudantes compreendam que os conhecimentos, ainda que originados em áreas distintas, são interdependentes e complementares (ICE, 2021a).

No currículo da Escola da Escolha, estão previstas três modalidades de eletivas, que se distinguem por suas naturezas e finalidades específicas: as Eletivas da Base Nacional Comum Curricular (Eletivas da Base), as Eletivas Pré-Itinerário Formativo (Eletivas Pré-IF) e as Eletivas de Itinerário Formativo (Eletivas IF).

Como mencionado anteriormente, Itinerário Formativo refere-se ao percurso que o estudante trilhará ao longo do Ensino Médio, de acordo com seu projeto de vida, podendo envolver o aprofundamento em uma área do conhecimento ou a formação técnica e profissional, conforme a organização do sistema de ensino. Dentro desse contexto, as Eletivas Pré-IF são ofertadas na 1ª série do Ensino Médio e têm como objetivo apresentar aos estudantes as possibilidades em cada área do conhecimento, auxiliando-os na escolha do itinerário a ser seguido nas séries subsequentes.

Por sua vez, as Eletivas IF são ofertadas na 2ª e 3ª séries e visam o aprofundamento dos conhecimentos dentro da área escolhida pelo estudante ao final da 1ª série, promovendo uma formação mais alinhada aos seus interesses, habilidades e objetivos futuros.

Já as Eletivas da Base assumem um papel complementar e, ao mesmo tempo, integrador. Segundo a proposta do modelo Escola da Escolha (ICE, 2021a, p. 43), recomenda-se

que essas eletivas não estejam associadas diretamente ao Itinerário Formativo do estudante, justamente para garantir a continuidade da Formação Geral Básica. Assim, as Eletivas da Base, embora sejam componentes curriculares obrigatórios, sua escolha é livre, permitindo que o estudante opte, a cada semestre, por um tema de seu interesse, a partir de um cardápio de ofertas elaborado pelos professores. As Eletivas da Base são ofertadas semestralmente, com uma carga horária de duas horas semanais.

> O papel do professor nas aulas das Eletivas da Base, Eletivas Pré-Itinerário Formativo e Eletivas de Itinerário Formativo é despertar o interesse e levar os estudantes na constante aplicação dos seus conhecimentos por meio da análise de problemas, situações e acontecimentos dentro de um contexto real, utilizando os recursos presentes nas diversas áreas do conhecimento (ICE, 2021a, p. 78).

Dessa forma, cabe ao professor a missão de elaborar, a cada semestre, uma eletiva que contribua de maneira significativa para a formação integral do estudante. Essa elaboração deve considerar tanto as competências e habilidades previstas na BNCC, quanto as competências esperadas do estudante do século XXI, como pensamento crítico, criatividade, colaboração e fluência digital.

Assim, as eletivas possibilitam que os estudantes entrem em contato com uma variedade de temas, desenvolvam competências específicas e participem de experiências de aprendizagem mais contextualizadas e relevantes. Além disso, constituem um ambiente favorável à inovação pedagógica, viabilizando o uso de tecnologias, metodologias ativas e abordagens interdisciplinares. Neste contexto de flexibilidade curricular e novas demandas para o ensino médio, é proposto uma eletiva que articule Matemática e Programação.

2.3 Matemática, Programação e as Competências do Século XXI

Considerando esse cenário, com novas normativas e estímulo a novas metodologias, a articulação entre Matemática e Programação apresenta-se como uma boa abordagem para responder às novas exigências da formação básica por meio de uma eletiva. A linguagem computacional, quando inserida de forma contextualizada e interdisciplinar, contribui para o desenvolvimento do pensamento lógico, da criatividade, da resolução de problemas e do letramento digital, habilidades consideradas essenciais para a formação do cidadão do século XXI (Valente, 2019). Nesse sentido, integrar a Programação ao ensino da Matemática não significa apenas ampliar o repertório técnico dos estudantes, mas oferecer experiências que favoreçam o raciocínio, a autonomia e a compreensão crítica dos dados que circulam na sociedade.

As competências do século XXI referem-se a um conjunto de conhecimentos, habilidades, atitudes e valores considerados essenciais para que os indivíduos atuem de forma ativa, crítica, criativa e colaborativa em uma sociedade em constante transformação. Em um cenário marcado pela globalização, pelos avanços tecnológicos acelerados, pelas mudanças nas relações de trabalho e pela complexidade dos desafios sociais e ambientais, tais competências visam preparar os estudantes não apenas para responder a demandas atuais, mas também para antecipar e construir futuros possíveis. Como afirma Valente, Freire e Arantes (2018, p. 24):

> As habilidades do Século XXI deverão incluir uma mistura de atributos cognitivos, intrapessoais e interpessoais como colaboração e trabalho em equipe, criatividade e imaginação, pensamento crítico e resolução de problemas, que os estudantes aprenderão por intermédio de atividades mão-na-massa, realizadas com o apoio conceitual desenvolvido em diferentes disciplinas.

Em 2015, a Organização para Cooperação e Desenvolvimento Econômico (OCDE) divulgou o projeto *The Future of Education and Skills: Education 2030* (OECD, 2018), cujo objetivo é fornecer respostas aos desafios emergentes da educação para o século XXI. O documento destaca que, diante da acelerada globalização e do rápido desenvolvimento tecnológico, as escolas devem preparar os estudantes para profissões que ainda não existem, tecnologias que ainda não foram inventadas e problemas que sequer são conhecidos atualmente. Diante disso, a OCDE propõe a seguinte reflexão: que conhecimentos, habilidades, atitudes e valores os estudantes de hoje precisarão para moldar o futuro?

> Os alunos precisarão aplicar seus conhecimentos em circunstâncias desconhecidas e em constante evolução. Para isso, precisarão de uma ampla gama de habilidades, incluindo habilidades cognitivas e metacognitivas (por exemplo, pensamento crítico, pensamento criativo, aprender a aprender e autorregulação); habilidades socioemocionais (por exemplo, empatia, autoeficácia e colaboração); e habilidades práticas e físicas (por exemplo, uso de novos dispositivos de tecnologia da informação e comunicação) (OECD, 2018, p. 5, tradução própria).

Na mesma perspectiva, a BNCC afirma que a construção de conhecimentos deve ser articulada ao desenvolvimento de habilidades e à formação de atitudes e valores, em conformidade com a LDB. Esse processo tem como foco o desenvolvimento de competências, entendidas como a capacidade de mobilizar conhecimentos (conceitos e procedimentos), habilidades (práticas, cognitivas e socioemocionais), atitudes e valores para resolver demandas complexas da vida cotidiana, do pleno exercício da cidadania e do mundo do trabalho (Brasil, 2019, p. 8). Essas competências são de natureza transversal, ou seja, devem ser trabalhadas em todas as etapas da Educação Básica e em todas as áreas do conhecimento. Elas formam a base para o desenvolvimento integral dos estudantes, preparando-os para os desafios contemporâneos.

Enquanto a OCDE oferece uma perspectiva internacional voltada para o futuro da educação e das habilidades necessárias em uma sociedade globalizada, a BNCC incorpora essas orientações ao contexto educacional brasileiro, traduzindo-as em competências que respeitam as especificidades culturais, sociais e econômicas do país. Ambas as propostas convergem ao reconhecer que o desenvolvimento de competências como pensamento crítico, resolução de problemas, criatividade e letramento digital é essencial para a formação de cidadãos preparados para enfrentar os desafios do século XXI. Dessa forma, a BNCC pode ser compreendida como uma materialização, em âmbito nacional, dos princípios mais amplos defendidos por organismos internacionais como a OCDE.

A BNCC estabelece dez competências gerais da Educação Básica, embora todas sejam relevantes, este trabalho destaca duas competências:

> 2. Exercitar a curiosidade intelectual e recorrer à abordagem própria das ciências, incluindo a investigação, a reflexão, a análise crítica, a imaginação e a criatividade, para investigar causas, elaborar e testar hipóteses, formular e resolver problemas e criar soluções (inclusive tecnológicas) com base nos conhecimentos das diferentes áreas.
> 5. Compreender, utilizar e criar tecnologias digitais de informação e comunicação de forma crítica, significativa, reflexiva e ética nas diversas práticas sociais (incluindo as escolares) para se comunicar, acessar e disseminar informações, produzir conhecimentos, resolver problemas e exercer protagonismo e autoria na vida pessoal e coletiva (Brasil, 2018, p.9).

A competência 2 promove o pensamento científico, de forma integrada, a competência 5 promove a cultura digital. Evidencia-se ambas, pois estão diretamente relacionadas à proposta de integrar matemática e programação no ensino médio, ao desenvolver essas competências espera-se que o estudante seja um sujeito autônomo intelectualmente, que busque soluções criativas, utilize tecnologias de maneira responsável, sendo capaz de resolver problemas de forma crítica.

Dessa forma, o foco da matemática, de acordo orientação da BNCC, deve estar na construção de uma visão integrada e aplicada à realidade, considerando as vivências cotidianas dos estudantes e os impactos da tecnologia em suas vidas. Segundo o documento:

> [...] quando a realidade é a referência, é preciso levar em conta as vivências cotidianas dos estudantes do Ensino Médio – impactados de diferentes maneiras pelos avanços tecnológicos, pelas exigências do mercado de trabalho, pelos projetos de bem viver dos seus povos, pela potencialidade das mídias sociais, entre outros (Brasil, 2018, p. 528).

Nessa perspectiva, enfatiza-se a importância de que o ensino da Matemática dialogue com as linguagens da cultura digital e contribua com respostas aos desafios contemporâneos. Essa orientação aproxima-se de diretrizes internacionais, como o relatório da OCDE, que reconhecem o papel central das competências digitais e do pensamento computacional na formação dos estudantes para o século XXI (OECD, 2018).

Ademais, a BNCC reforça que o uso de tecnologias digitais deve ser utilizado como recurso tanto para investigação matemática como para dar continuidade ao desenvolvimento do pensamento computacional iniciado no Ensino Fundamental (Brasil, 2018, p. 528). Assim, ao integrar a Programação ao ensino da Matemática por meio de uma proposta didática contextualizada, busca-se atender a essas demandas, conectando o conhecimento escolar às linguagens e ferramentas do cotidiano dos estudantes e às exigências atuais da sociedade.

Em 2006, Jennete Wing, professora de Ciência da Computação da Universidade de Columbia, popularizou o conceito de Pensamento Computacional, em que afirma ser uma habilidade analítica fundamental para todas as crianças, assim como leitura, escrita e aritmética (Wing, 2016). Segundo a autora, trata-se de um processo que envolve a formulação e resolução de problemas por meio da decomposição em partes menores, da identificação de padrões, do raciocínio recursivo, da construção de algoritmos e da sua posterior testagem.

A BNCC amplia essa concepção ao afirmar que o pensamento computacional "envolve as capacidades de compreender, analisar, definir, modelar, resolver, comparar

e automatizar problemas e suas soluções, de forma metódica e sistemática, por meio do desenvolvimento de algoritmos"(Brasil, 2018, p. 474).

Além disso, na área de Matemática, Brasil (2018, p. 529) afirma que "os estudantes devem desenvolver habilidades relativas aos processos de investigação, de construção de modelos e de resolução de problemas". Logo, construir o pensamento computacional concomitante ao pensamento matemático potencializa a construção de ambos, visto que são pensamentos que exigem o desenvolvimento de habilidades mutuamente dependentes. Dessa forma, ao programar, é possível experimentar concretamente conceitos matemáticos que, por muitas vezes, permanecem abstratos durante o processo de aprendizagem.

A articulação entre Matemática e Programação, fundamentada pelas competências da BNCC, oferece uma alternativa para o processo de ensino e aprendizagem no Ensino Médio. Ao propor o desenvolvimento do Pensamento Computacional e do Pensamento Matemático, oferece uma oportunidade aos estudantes assumirem uma postura mais ativa e investigativa, em que se desenvolvem não só habilidades técnicas, mas também habilidades analíticas, colaborativas e criativas.

Entre os diversos conteúdos possíveis para integrar Matemática e Programação, optou-se por trabalhar com a leitura, interpretação e produção de gráficos e tabelas. Essa escolha se justifica tanto pelo alinhamento com as habilidades previstas na BNCC quanto pela relevância desse tema na vida cotidiana e no mundo do trabalho.

Em uma sociedade orientada por dados, em que a tomada de decisões depende da capacidade de compreender representações gráficas, torna-se essencial formar indivíduos aptos a compreender os fenômenos ao seu redor. Sob essa perspectiva, este trabalho tem como foco o desenvolvimento da Competência Específica 4 da área de Matemática e suas Tecnologias, prevista na BNCC, que propõe que os estudantes devem ser capazes de:

> Compreender e utilizar, com flexibilidade e precisão, diferentes registros de representação matemáticos (algébrico, geométrico, estatístico, computacional etc.), na busca de solução e comunicação de resultados de problemas (Brasil, 2018, p. 538).

Essa competência evidencia a importância dos estudantes mobilizarem diferentes registros de um mesmo objeto matemático como ferramenta para resolver problemas em contextos diversos, incluindo situações do cotidiano, socioambientais e tecnológicas. Compreender e transitar entre essas representações é, portanto, um aspecto essencial do pensamento matemático, pois possibilita aos estudantes a construção de significados mais profundos e conectados aos conteúdos. Além disso, a forma como o aluno representa e organiza suas ideias ao resolver um problema revela não apenas seu domínio conceitual, mas também seus modos de pensar e raciocinar.

Nesse cenário, a Programação surge como um caminho para o desenvolvimento dessa competência. Ao construir algoritmos, criar visualizações e testar soluções por meio de linguagens computacionais, os estudantes são constantemente desafiados a representar informações matemáticas de maneiras distintas, estabelecendo conexões entre o pensamento abstrato e sua aplicação prática. Assim, espera-se que o trabalho com códigos, gráficos e modelos computacionais reforce o desenvolvimento das seguintes habilidades previstas na BNCC:

- (EM13MAT401) Converter representações algébricas de funções polinomiais de 1º grau em representações geométricas no plano cartesiano, distinguindo os casos nos

quais o comportamento é proporcional, recorrendo ou não a softwares ou aplicativos de álgebra e geometria dinâmica.

- (EM13MAT402) Converter representações algébricas de funções polinomiais de 2º grau em representações geométricas no plano cartesiano, distinguindo os casos nos quais uma variável for diretamente proporcional ao quadrado da outra, recorrendo ou não a softwares ou aplicativos de álgebra e geometria dinâmica, entre outros materiais.

- (EM13MAT403) Analisar e estabelecer relações, com ou sem apoio de tecnologias digitais, entre as representações de funções exponencial e logarítmica expressas em tabelas e em plano cartesiano, para identificar as características fundamentais (domínio,imagem, crescimento) de cada função.

- (EM13MAT405) Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

- (EM13MAT406) Construir e interpretar tabelas e gráficos de frequências com base em dados obtidos em pesquisas por amostras estatísticas, incluindo ou não o uso de softwares que inter-relacionem estatística, geometria e álgebra.

Portanto, o presente trabalho propõe a utilização do espaço curricular destinado às Eletivas da Base, em conformidade com suas diretrizes, para associar Matemática e Programação, com foco no desenvolvimento de habilidades em modelar e resolver problemas matemáticos com base em representações múltiplas e digitais, integrando o uso de tecnologias. A proposta apresenta-se como um manual introdutório para professores interessados em utilizar a linguagem de programação Python como ferramenta pedagógica, mesmo que não possuam experiência prévia com programação. Dessa forma, busca-se promover o pensamento computacional e estimular a utilização de recursos computacionais como apoio à investigação matemática, favorecendo uma aprendizagem mais ativa, significativa e alinhada às demandas contemporâneas.

3 INTRODUÇÃO À LINGUAGEM PYTHON: CONCEITOS BÁSICOS

Diante da proposta de integrar Matemática e Programação por meio de uma eletiva, esta seção tem como objetivo apresentar conceitos introdutórios da linguagem Python, fornecendo ao leitor uma base para compreender e aplicar os recursos que serão utilizados na Seção 4. A escolha dessa linguagem fundamenta-se tanto em suas características técnicas quanto em seu potencial pedagógico, destacando-se como uma ferramenta acessível e versátil para apoiar a aprendizagem matemática de forma prática e contextualizada.

Em 1989, em busca de um passatempo que o mantivesse ocupado durante a semana do Natal, Guido van Rossum iniciou o desenvolvimento de um projeto que, futuramente, se tornaria uma das linguagens de programação mais utilizadas no mundo: o Python. Segundo o próprio Van Rossum, o nome da linguagem foi inspirado em um de seus programas de televisão favoritos, o programa de humor *Monty Python's Flying Circus* (Rossum, 1996).

Python é um software livre, ou seja, pode ser utilizado gratuitamente. Uma de suas maiores vantagens está na versatilidade: a linguagem roda em diversos sistemas operacionais, como Windows, Linux e Mac OS. Além disso, é possível programar em Python diretamente do navegador, sem a necessidade de instalar qualquer software, por meio de plataformas como o Google Colab (Google, 2024). Essa ferramenta baseada na nuvem permite que o usuário escreva e execute códigos online, oferecendo uma interface intuitiva, suporte para colaboração em tempo real e até mesmo a possibilidade de uso em smartphones.

Criada com o objetivo de ser simples, rápida e acessível, a linguagem Python reúne, em um único ambiente, uma grande variedade de aplicações. Ela pode ser utilizada para análise e ciência de dados, desenvolvimento de animações 3D, criação de aplicativos para desktop e dispositivos móveis, desenvolvimento de aplicações científicas e acadêmicas, inteligência artificial, machine learning, jogos digitais, sites, chatbots, entre outros recursos. Essa diversidade a torna uma linguagem poderosa tanto para iniciantes quanto para profissionais experientes.

Uma característica que torna Python especialmente atrativa para quem está começando a programar é sua sintaxe clara e legível. Diferente de outras linguagens que exigem diversos símbolos para marcar o fim de comandos ou delimitar blocos de código, Python adota uma estrutura mais limpa e intuitiva:

> Outras linguagens de programação utilizam inúmeras marcações, como ponto (.) ou ponto e vírgula (;), no fim de cada linha, além dos marcadores de início e fim de bloco como chaves ( ) ou palavras especiais (begin/end). Esses marcadores tornam os programas um tanto mais difíceis de ler e felizmente não são usados em Python (Menezes, 2010, p.22)

Outro ponto forte da linguagem é a vasta coleção de bibliotecas disponíveis. Essas bibliotecas oferecem funções pré-programadas e amplamente testadas por outros desenvolvedores, o que permite ao programador focar diretamente na resolução dos problemas que deseja tratar, sem a necessidade de escrever todo o código do zero.

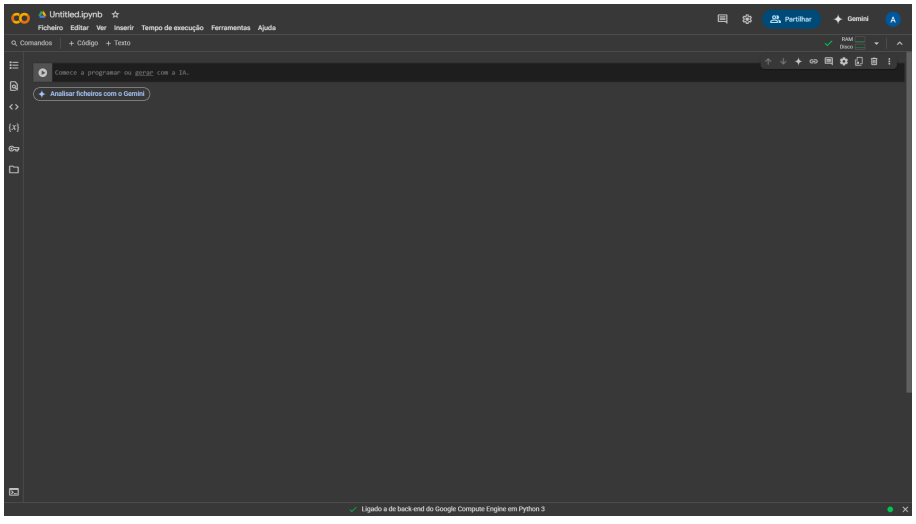
Como destaca Menezes (2010), "Python foi escolhida por facilitar o processo de aprendizagem e, ao mesmo tempo, oferecer grande potencial de desenvolvimento".

Portanto, a escolha por Python no contexto da educação básica se justifica por diversas razões: trata-se de uma ferramenta gratuita, de fácil acesso, que pode ser utilizada mesmo em contextos de infraestrutura limitada como escolas que não possuem laboratórios de informática. Além disso, sua sintaxe acessível reduz a frustração dos estudantes, que não precisam se preocupar com erros causados por detalhes como a ausência de um ponto e vírgula. Por tudo isso, a introdução da programação por meio da linguagem Python, especialmente em articulação com a matemática, pode representar uma excelente porta de entrada para estudantes explorarem o mundo da tecnologia, desenvolverem habilidades de resolução de problemas e, quem sabe, descobrirem afinidade com futuras carreiras na área da computação, obtendo dessa forma competências fundamentais para o século XXI.

3.1 Estrutura Básica da Linguagem e Comandos Iniciais

Este trabalho foi desenvolvido utilizando o ambiente de programação em nuvem Google Colaboratory, conhecido como Google Colab. Por ser um ambiente online, ele elimina a necessidade de instalação de softwares ou configurações locais, permitindo que o usuário apenas abra um bloco de notas e comece a programar diretamente do navegador, sem se preocupar em salvar ou baixar arquivos manualmente. Essa praticidade otimiza o tempo e facilita o acesso, especialmente em contextos educacionais nos quais nem sempre há infraestrutura adequada disponível. Na Figura 2, observa-se a interface do ambiente Google Colab, utilizada no desenvolvimento deste trabalho.

Figura 2 — Interface do ambiente Google Colab.



Fonte: Elaboração própria (2025).

Por se tratar de um material voltado para a Educação Básica, com a proposta de servir como um manual introdutório para professores em seu primeiro contato com a linguagem de programação, foram selecionadas apenas algumas estruturas e funções consideradas essenciais, com base nas necessidades observadas durante o desenvolvimento dos códigos apresentados na seção 4. A intenção é oferecer uma abordagem acessível e prática, sem

a pretensão de esgotar o conteúdo da linguagem. Para os que desejarem aprofundar seus conhecimentos em Python, recomenda-se a leitura da obra de Lutz (2013): Programming Python ou a consulta ao site oficial da linguagem: https://www.python.org/.

A seguir, serão apresentadas algumas estruturas básicas da linguagem Python, com o objetivo de familiarizar o leitor com seus principais elementos sintáticos e operacionais. Ao começar a escrever códigos em Python, é importante observar algumas regras fundamentais da linguagem, pequenos descuidos podem impedir o funcionamento correto do programa. Dessa forma, destacam-se alguns cuidados essenciais:

a) letras maiúsculas e minúsculas são diferentes, ou seja, `Print`, `PRINT` e `print` são interpretados como coisas diferentes.

b) parênteses não são opcionais, o uso de parênteses é obrigatório em determinadas estruturas, como na função `print()` ou `def()`.

c) deve-se evitar colocar diferentes instruções na mesma linha; é importante escrever cada comando separadamente e respeitar a indentação sempre que necessário, a menos que se saiba exatamente como fazer isso corretamente.

d) atenção à pontuação numérica: em Python, utiliza-se o ponto (`.`) em vez da vírgula (`,`) para representar números decimais. Por exemplo, para indicar quinze centésimos, deve-se escrever `0.15`, e não `0,15`.

A função `print` e Python como calculadora

A função `print` é uma das mais utilizadas em Python. Sua principal finalidade é exibir na tela mensagens, resultados de cálculos ou qualquer outro conteúdo especificado entre os parênteses. Veja o exemplo a seguir:

```
1 print("Olá,mundo!")
```

Código 1 — Print

Uma das vantagens do Python em relação a outras linguagens, como C, é a simplicidade no processo de exibição de resultados. Não é necessário utilizar comandos adicionais para visualizar a saída. Isso faz com que Python seja especialmente útil para quem está iniciando e deseja, por exemplo, utilizar a linguagem como uma calculadora:

```
1     3 + 4
2     7
```

Código 2 — Adição

Para visualizar a saída do código, ou seja, para que o interpretador execute as instruções e exiba o resultado, basta pressionar as teclas de atalho `Ctrl + Enter`. Esse

comando executa a célula ativa no Google Colab, e em muitos casos, especialmente em operações simples como cálculos, não é necessário utilizar a função `print`, pois o resultado é exibido automaticamente na saída da célula.

```
12 - 3 + 5
14
```

Código 3 — Adição e Subtração

A multiplicação é representada por um asterisco (*) e a divisão, pela barra (/):

```
4*5
20
```

Código 4 — Multiplicação

```
30/5
6
```

Código 5 — Divisão

Para elevar um número a um expoente, utilizamos dois asteriscos (**). Por exemplo:

```
4**3
64
```

Código 6 — Exponenciação

Enquanto que o resto da divisão entre dois números inteiros utiliza-se o símbolo (%)

```
67%3
1
```

Código 7 — Resto da divisão

Quando lidamos com expressões numéricas em Python, o uso dos parênteses segue a mesma lógica da matemática: eles servem para alterar a ordem das operações quando necessário. Assim como nos cálculos tradicionais, a ordem das operações é fundamental e pode impactar diretamente o resultado obtido. Observe o exemplo a seguir:

```
10 + 15*8
130
```

Código 8 — Expressão numérica

```
(10 + 15)*8
200
```

Código 9 — Expressão numérica

Variável e Atribuição

Em programação, as variáveis são utilizadas para armazenar valores. Elas funcionam como "espaços nomeados" na memória do computador, permitindo guardar dados que poderão ser reutilizados ao longo do programa. Para isso, utiliza-se o símbolo de igualdade (=) entre o nome da variável e o valor que se deseja armazenar. Esse processo é chamado de atribuição, isto é, um valor é atribuído a uma variável, dizemos que a variável "recebeu"um valor.

Como exemplo, vejamos a resolução de um problema simples para calcular o valor de um salário após um aumento percentual. Suponha que um salário inicial de R$750,00 receba um aumento de 15%. Qual será o valor final após o reajuste?

```
salário = 750
aumento = 15
print(salário + salário*aumento/100)
```

Código 10 — Aumento de salário

Uma forma alternativa de realizar esse cálculo, sem utilizar variáveis, seria:

```
print(750 + 750*15/100)
```

Código 11 — Alternativa para aumento de salário

Embora ambas as abordagens produzam o resultado correto, o uso de variáveis torna o código mais legível, flexível e fácil de modificar. Ao atribuir valores às variáveis, podemos alterar o salário ou o percentual de aumento independentemente, sem a necessidade de reescrever a expressão matemática.

Mais adiante, será apresentada a função `input()`, que permite solicitar ao usuário que insira valores diretamente, o que torna o programa mais dinâmico e interativo.

Nome das variáveis

Na linguagem de programação Python, os nomes de variáveis devem, obrigatoriamente, começar com uma letra ou com o caractere sublinhado ("_"). Após o primeiro caractere, é permitido utilizar letras, números e o próprio "_". O quadro 1 a seguir apresenta alguns exemplos de nomes válidos e inválidos para variáveis:

Quadro 1 – Exemplos de nomes válidos e inválidos para variáveis em Python

Nomes válidos	Nomes inválidos
"nome"	"2nome"(não pode começar com número)
"idade_aluno"	"nome@aluno"(caractere inválido)
"_variavel"	"nome aluno"(espaço não permitido)
"mediaFinal"	"class"(palavra reservada do Python)
"x1", "valor_total"	"nome-aluno"(hífen não permitido)

Fonte: Elaboração própria (2025).

Diferentemente de algumas linguagens de programação que utilizam exclusivamente o conjunto de caracteres ASCII - padrão americano que não aceita acentos ou letras como "ç" - o Python adota, por padrão, a codificação UTF-8 (Unicode Transformation Format - 8 bits). Essa codificação permite a utilização de praticamente todos os caracteres dos alfabetos conhecidos. Dessa forma, Python oferece aos programadores maior flexibilidade, permitindo o uso de acentos e caracteres especiais sem a necessidade de adaptações adicionais.

Tipos de variáveis: numéricas, string e lógico

Ao programar, é fundamental identificar o tipo de variável com a qual se está trabalhando, pois conhecer sua natureza evita erros decorrentes do uso inadequado em diferentes funções. A seguir, serão apresentados alguns dos principais tipos de variáveis em Python:

Variável Numérica

Uma variável é considerada numérica quando armazena números inteiros ou decimais (também chamados de ponto flutuante ou *float*). Dependendo da operação a ser realizada, pode ser necessário especificar se a variável é inteira ou não, utilizando a função `int(nome_da_variável)`. Vale destacar que, em Python, os números decimais são representados com ponto (.) em vez de vírgula (,). Por exemplo, o número dois vírgula um (2,1) deve ser escrito como 2.1.

Variável String

Uma variável é do tipo *string* quando armazena informações textuais, ou seja, uma sequência ordenada de caracteres. Para melhorar a visualização das informações são enviadas à tela para o usuário, é comum utilizar a composição de strings, que consiste em combinar ou substituir partes de texto por outras strings. Por exemplo,

ou simplesmente podemos utilizar em uma só sentença:

```
altura = 1.65
idade = 17
print("Maria tem %d anos" % idade)
```

Código 12 — Composição string número inteiro

```
print("Maria tem %.2f m de altura" % altura)
```

Código 13 — Composição string número decimal

```
print("Maria tem %d anos e %.2f m de altura" % (idade, altura))
```

Código 14 — Composição string

O símbolo % indica a composição da string com uma variável. Os marcadores de posição %d e %f indicam, respectivamente, que naquela posição será inserido um valor inteiro e um valor decimal. Já o marcador %.2f especifica que o número decimal deve ser exibido com duas casas decimais após a vírgula.

Saída Esperada:

```
Maria tem 17 anos e 1.65 m de altura
```

Variável do tipo lógico

Em algumas situações deseja-se verificar condições, controlar o fluxo de execução ou simplesmente comparar valores. Para isso, existem as variáveis do tipo lógico (também chamadas de booleanas), em que só pode ser dois valores como resposta: `True` (verdadeiro) ou `False` (falso).

Para realizar comparações lógicas, utilizamos operadores relacionais. Observe o quadro 2 com alguns operadores:

Quadro 2 – Operadores relacionais

Operador	Operação
==	igual
!=	diferente
>	maior que
<	menor que
>=	maior ou igual que
<==	menor ou igual que

Fonte: Elaboração própria (2025).

Ao trabalhar com um conjunto de dados para a plotagem de um gráfico, é fundamental que as variáveis envolvidas possuam o mesmo número de elementos. Em Python, existe

uma função simples e prática para contar a quantidade de elementos em um conjunto, como uma lista: a função len(nome_da_variável). Assim, para evitar erros de execução durante a criação do gráfico, uma abordagem eficiente é verificar se os conjuntos de dados possuem o mesmo tamanho utilizando:

`len(variável_1) == len(variável_2)`

Se o resultado da comparação for True, a consistência dos dados estará garantida, evitando falhas ao gerar o gráfico.

Entrada de variáveis

Até o momento, os programas construídos utilizavam valores definidos diretamente no código, ou seja, informações já conhecidas previamente pelo programador. No entanto, uma das grandes vantagens de um programa é justamente sua capacidade de receber diferentes valores de entrada sem que seja necessário alterar o código a cada nova execução.

Para isso, utilizamos a função `input`, que permite solicitar dados ao usuário durante a execução do programa. O valor digitado pelo usuário é então capturado pelo programa quando a tecla `ENTER` é pressionada. A seguir, apresentamos um exemplo simples, no qual o programa solicita um número e imprime o valor digitado:

```
1 número = input("Digite um número: ")
2 print(número)
```

Código 15 — input

É importante destacar que a função `input` retorna sempre um valor do tipo `string`, isto é, uma cadeia de caracteres. Mesmo que o usuário digite um número, o valor será interpretado como texto. Isso pode gerar problemas quando se deseja realizar operações matemáticas com os dados fornecidos. Para contornar essa limitação, é necessário converter o valor retornado para um tipo numérico, utilizando as funções `int` (para números inteiros) ou `float` (para números decimais).

Veja, por exemplo, um programa que solicita a temperatura em graus Celsius (°C) e realiza a conversão para Fahrenheit (°F):

```
1 celsius = input("Digite a temperatura em Celsius: ")
2 celsius = float(celsius)
3 fahrenheit = (celsius * 9/5) + 32
4 print("A temperatura em Fahrenheit é:", fahrenheit)
```

Código 16 — Conversão temperatura

Uma alternativa mais concisa e elegante é realizar a conversão diretamente na leitura do valor, como mostra o exemplo a seguir:

```python
celsius = float(input("Digite a temperatura em Celsius: "))
fahrenheit = (celsius * 9/5) + 32
print("A temperatura em Fahrenheit é:", fahrenheit)
```

Código 17 — Conversão temperatura

Dominar o uso de variáveis e saber manipular entradas é essencial para o desenvolvimento de programas mais interativos e versáteis. Além disso, à medida que os programas se tornam mais complexos, torna-se necessário lidar com situações em que decisões precisam ser tomadas com base nos dados fornecidos pelo usuário, ou em que ações devem ser repetidas automaticamente. Nesse sentido, exploraremos a seguir as estruturas de condição e repetição, ferramentas fundamentais para controlar o fluxo de execução dos programas e torná-los mais inteligentes e eficientes.

Condições e Repetições

Em programação, estruturas de repetição (também chamadas de laços ou loops) permitem que um bloco de código seja executado diversas vezes. Já as estruturas condicionais possibilitam controlar o fluxo de execução do programa com base em critérios previamente definidos.

A combinação dessas duas estruturas torna possível construir programas mais inteligentes e flexíveis: a repetição é responsável por executar padrões, enquanto a condição decide quais caminhos seguir. Assim como na Matemática há regras que permitem gerar padrões e estabelecer generalizações, na programação isso ocorre por meio da lógica e do controle de fluxo.

Estruturas condicionais

Essenciais para tomada de decisões, em Python existem algumas estruturas condicionais como `if` e `else` que auxiliam no fluxo de execução do programa. Assim, dependendo da situação é interessante definir quais partes do programa devem ser executadas de acordo com uma condição.

A estrutura condicional mais utilizada é o `if`, do inglês *"se"*, funciona da seguinte forma: se a condição é verdadeira, então execute o bloco de código. Sua sintaxe é:

```
if <condição>:
  bloco
```

Vejamos um exemplo de um código que solicita do usuário um número e verifica se é par ou ímpar:

```
número = input("Digite um número: ")
número = int(número)
if número % 2 == 0:
  print("O número é par")

if número % 2 !=0:
  print ("O número é ímpar")
```

Código 18 — if

Observe que o bloco de código deve começar mais à direita da linha anterior que iniciou o bloco, a isso chama-se indentação, esse espaço em branco no início da linha é o que determina o início e o fim dos blocos de códigos nos quais pertencem à estrutura `if`. Esse processo deixa o código mais legível, organizado e evita ambiguidade sobre o que pertence a qual bloco. Em outras linguagens como C, utiliza-se o bloco dentro de chave, enquanto em outras utiliza-se `begin/end`

Uma alternativa para evitar utilizar duas vezes o `if`, é optar pela estrutura `else`. Em certos casos em que a segunda condição é o inverso da primeira, podemos utilizar um artifício que simplifica o programa.

Assim, um código mais elegante do exemplo acima seria:

```
número = input("Digite um número: ")
número = int(número)
if número % 2 == 0:
  print("O número é par")

else:
  print ("O número é ímpar")
```

Código 19 — else

Estruturas de repetição (`while` e `for`)

A repetição é a base de diversos programas. Ela é utilizada quando se deseja executar um conjunto de comandos mais de uma vez. Em Python, destacam-se duas estruturas de repetição: `while` e `for`, que serão apresentadas a seguir.

A estrutura `while`, do inglês enquanto, executa um bloco de código enquanto a condição lógica especificada for verdadeira. A sintaxe é:

```
while <condição>:
   bloco
```

É importante atentar-se à indentação: em Python, a indentação define quais comandos pertencem ao bloco de repetição. Caso não seja respeitada, ocorrerá erro de execução.

Considere o exemplo a seguir, no qual se deseja imprimir os números de 1 a 10:

```
x = 1
while x<= 10:
  print(x)
  x = x + 1
```

Código 20 — while

Cada linha de código representa um comando. Note que, se a linha `print(x)` estivesse abaixo de `x = x + 1`, a saída começaria a partir do número 2. A instrução `x = x + 1` atualiza o valor da variável `x` a cada iteração, somando 1 ao seu valor anterior. Essa operação permite contar quantas vezes o laço foi executado, por isso, `x` é chamada de contador.

Contadores são ferramentas úteis na construção de programas, pois permitem acompanhar, medir ou controlar o número de execuções. No exemplo a seguir, é apresentada uma progressão geométrica com razão igual a 2 na qual deseja-se descobrir quantos termos são menores que 1000:

```
termo = 1
contador = 0

while termo < 1000:
  print(termo)
  termo = termo* 2
  contador = contador + 1

print("Total de termos:", contador)
```

Código 21 — while sequência

Saída Esperada:

```
1
2
4
8
16
32
64
128
256
512
Total de termos: 10
```

Por outro lado, uma ferramenta igualmente importante no contexto das estruturas de repetição é o uso de acumuladores. Enquanto os contadores geralmente incrementam uma variável com um valor constante (como somar 1 a cada repetição), os acumuladores têm como objetivo acumular valores ao longo da execução de um laço. Isso pode ser feito

por meio de operações de soma, multiplicação ou outras formas de combinação de valores sucessivos.

No exemplo anterior, a variável `termo` pode ser considerada um acumulador, pois ela armazena o resultado de valores que vão sendo somados a cada iteração.

Uma maneira prática e eficiente de implementar acumuladores e contadores em Python é por meio dos operadores de atribuição. Esses operadores simplificam a modificação de valores armazenados em variáveis, tornando o código mais elegante, compacto e legível. Além disso, contribuem para a redução de erros e da repetição desnecessária de expressões.

Veja alguns exemplos de operadores de atribuição no quadro 3:

Quadro 3 – Operadores de Atribuição em Python

Operador	Exemplo	Equivalência
=	x = 5	Atribui o valor 5 à variável x
+=	x += 2	x = x + 2
-=	x -= 3	x = x - 3
*=	x *= 4	x = x * 4
/=	x /= 2	x = x / 2
//=	x //= 2	x = x // 2 (divisão inteira)
%=	x %= 3	x = x % 3 (resto da divisão)
**=	x **= 2	x = x ** 2 (potência)

Fonte: Elaboração própria (2025).

Assim, uma forma mais elegante de escrever o código apresentado anteriormente, utilizando operadores de atribuição, é a seguinte:

```
1 termo = 1
2 contador = 0
3
4 while termo < 1000:
5   print(termo)
6   termo *= 2
7   contador += 1
8
9 print("Total de termos:", contador)
```

Código 22 — while sequência com operadores de atribuição

Dessa forma, a variável `termo` é um acumulador, e o operador $*=$ serve para multiplicar e acumular os valores numéricos a cada iteração. Um acumulador pode ser compreendido como uma "caixa" onde, a cada vez que um valor atende a determinada condição, esse valor é somado (ou multiplicado) ao que já se encontra armazenado na variável.

Diferentemente do contador que apenas registra quantas vezes uma condição foi satisfeita, o acumulador permite realizar somatórios, produtos ou relações de recorrência, sendo uma ferramenta essencial para diversas operações em algoritmos.

Saber utilizar de forma eficiente os contadores e acumuladores amplia consideravelmente o potencial dos códigos desenvolvidos. Essas ferramentas estão no cerne do pensamento computacional, pois possibilitam ao programador visualizar cada iteração e compreender o processamento da informação passo a passo. Esse processo fortalece a capacidade de pensar algoritmicamente, ou seja, projetar uma sequência clara e lógica de ações que viabilizam a resolução de problemas. Dessa forma, o uso de acumuladores e contadores contribui para a construção de soluções mais eficazes e estruturadas.

A estrutura de repetição `while`, comum a diversas linguagens de programação, permite que um bloco de código seja executado enquanto uma condição for verdadeira. No entanto, essa estrutura pode apresentar limitações quando se deseja percorrer sequências bem definidas de dados, como listas, vetores ou intervalos numéricos.

Nesses casos, a linguagem Python oferece a estrutura de repetição `for`, que se destaca por sua sintaxe mais compacta, legível e segura, especialmente quando o número de repetições é conhecido ou quando se deseja iterar sobre uma sequência de elementos. O laço `for` é amplamente utilizado em operações que envolvem coleções de dados, como listas, pois permite acessar diretamente cada item da sequência sem a necessidade de gerenciar índices ou contadores externos.

A estrutura de repetição `for` funciona de forma semelhante ao `while`, mas foi projetada especificamente para percorrer elementos de sequências, como listas. Isso a torna especialmente útil ao lidar com grandes conjuntos de dados, como será o caso ao gerar visualizações e gráficos a partir de listas numéricas.

Sua sintaxe básica é:

```
for <elemento> in <sequência>:
    bloco
```

A estrutura `for` possui grande potencial quando combinado com listas, pois permite acessar e manipular cada item de maneira simples e organizada. Por isso é fundamental compreender o que são listas em Python e como elas podem ser utilizadas no processamento de dados.

Listas

As listas são um tipo de estrutura de dados que permitem armazenar múltiplos elementos, inclusive de tipos diferentes. Uma característica importante das listas é que seus elementos podem ser alterados, adicionados ou removidos após sua criação, o que as torna estruturas *mutáveis*. Além disso, as listas mantêm a ordem em que os elementos foram inseridos, sendo, portanto, *estruturas ordenadas*. Assim, podemos defini-las como coleções de itens que são, simultaneamente, mutáveis e ordenadas.

As listas oferecem diversas possibilidades no processamento de dados, especialmente em contextos que envolvem visualização gráfica, como será explorado na seção seguinte.

A sintaxe para criar uma lista envolve o uso de colchetes `[ ]`, dentro dos quais os elementos são separados por vírgulas.

Considere o exemplo a seguir, em que a variável `lista` recebe uma lista vazia, ou seja, sem nenhum elemento:

No próximo exemplo, temos uma lista que contém três elementos:

```
lista = [ ]
```

Código 23 — Lista vazia

```
X = [37, 82, 14]
```

Código 24 — Lista com três elementos

Como a lista `X` possui três elementos, dizemos que seu tamanho é 3. Cada elemento da lista pode ser acessado por meio de seu índice (posição). É importante lembrar que, em Python, a indexação começa em 0. Assim, `X[0]` retorna o valor 37, `X[1]` retorna 82, e assim por diante.

Outra forma de acessar os elementos de uma lista é utilizando um laço `for`, como mostra o exemplo a seguir:

```
for n in X:              # Percorre cada elemento da lista X
   print(n)
```

Código 25 — Acessar elementos da lista com `for`

Saída esperada:

```
37
82
14
```

As listas são amplamente utilizadas para armazenar resultados obtidos por meio de laços de repetição, isto é, quando se deseja iterar operações e salvar as saídas em forma de lista. Para isso, é necessário iniciar com uma lista vazia, que servirá como recipiente para os resultados, e utilizar o método `append()`, responsável por adicionar elementos ao final da lista.

Observe o exemplo a seguir:

```
X = [37, 82, 14]
X.append(5)
X
```

Código 26 — Adicionar elemento ao final da lista

Saída esperada:

```
[37, 82, 14, 5]
```

Sequência de números

A função `range()` em Python é responsável por gerar uma sequência de números inteiros e é especialmente útil em estruturas de repetição como o laço `for`.

Sua sintaxe é dada por:

```
range(start, stop, step)
```

`start`: valor inicial da sequência (opcional; por padrão, é 0);
`stop`: valor final (obrigatório); o número indicado não é incluído na sequência;
`step`: incremento entre os valores (opcional; por padrão, é 1).

Em resumo, a função `range()` gera uma progressão aritmética de inteiros, começando em `start`, parando antes de `stop`, e avançando de acordo com o valor de `step`. Caso apenas o `stop` seja especificado, os valores padrão para `start` e `step` serão utilizados.

Portanto, há três formas principais de utilizar a função `range()`:

```
range(stop)
range(start, stop)
range(start, stop, step)
```

Veja os exemplos:

```
1 list(range(7))
```

Código 27 — `range(stop)`

Saída esperada:

```
1 [0, 1, 2, 3, 4, 5, 6]
```

No caso `range(start, stop)`, temos:

```
1 list(range(2, 10))
```

Código 28 — `range(start, stop)`

Saída esperada:

```
1 [2, 3, 4, 5, 6, 7, 8, 9]
```

No caso `range(start, stop, step)`, temos:

```
1 list(range(10, 0, -1))
```

Código 29 — `range(start, stop, step)`

Saída esperada:

```
1 [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

No entanto, uma limitação importante da função `range()` é que o parâmetro `step` aceita apenas números inteiros. Por exemplo, não é possível utilizar `range(1, 3, 0.5)` para obter a sequência $1.0, 1.5, 2.0, 2.5$, pois isso resultará em um erro de tipo:

```
TypeError: 'float' object cannot be interpreted as an integer
```

Essa limitação pode representar um entrave ao se trabalhar com representações mais refinadas, como na plotagem de funções matemáticas com domínio contínuo, em que são necessários valores decimais. Nesse contexto, introduz-se a biblioteca `NumPy`, uma poderosa ferramenta da linguagem Python voltada à computação numérica.

A função `numpy.arange()`, pertencente a essa biblioteca, possui comportamento semelhante ao `range()`, com a vantagem de aceitar valores de ponto flutuante para o parâmetro `step`. Assim, é possível gerar intervalos com precisão decimal, como mostra o exemplo a seguir:

```python
import numpy as np
valores = np.arange(1, 3, 0.5)
valores
```

Código 30 — `np.arange()`

Saída esperada:

```
[1.  1.5 2.  2.5]
```

Para ambientes em que o uso de bibliotecas externas não é viável, é possível contornar essa limitação utilizando uma construção baseada em compreensão de listas:

```python
valores = [i * 0.5 for i in range(2, 7)]
valores
```

Código 31 — list comprehension

Nesse caso, o `range(2, 7)` gera os inteiros de 2 a 6 e, ao multiplicar cada um por $0,5$, obtêm-se os valores desejados, ou seja, uma sequência de 1,0 a 3,0 com passo de $0,5$:

Saída esperada:

```
[1.0, 1.5, 2.0, 2.5, 3.0]
```

Embora essa abordagem seja menos direta, ela permite gerar listas com incrementos fracionários utilizando apenas os recursos básicos da linguagem. Ambas as estratégias, com ou sem o uso de bibliotecas, promovem uma reflexão sobre a escolha de ferramentas de acordo com as exigências de cada problema, aspecto essencial no desenvolvimento do pensamento computacional.

Lista por compreensão

A *list comprehension*, ou compreensão de listas, é uma forma concisa, elegante e eficiente de criar listas em Python. Essa estrutura permite construir uma nova lista a partir de um iterável (como outra lista, um intervalo gerado por `range()`, entre outros), utilizando apenas uma linha de código.

Trata-se de uma alternativa compacta às estruturas tradicionais com `for`, favorecendo a clareza e a legibilidade do código. Em outras palavras, é possível gerar uma lista a

partir de outra coleção de dados por meio de uma expressão dentro de colchetes, seguida de um laço.

Observe o exemplo abaixo, que gera uma lista com os quadrados dos números de 0 a 4:

```
quadrados = [x**2 for x in range(5)]
quadrados
```

Código 32 — list comprehension

Saída esperada:

```
[0, 1, 4, 9, 16]
```

Até aqui, exploramos conceitos essenciais da linguagem Python que permitem armazenar dados, realizar repetições, tomar decisões e organizar informações em estruturas como listas. Esses elementos são a base para resolver uma ampla gama de problemas computacionais simples.

No entanto, à medida que os programas se tornam maiores ou que certas operações precisam ser reutilizadas em diferentes contextos, torna-se necessário adotar estratégias que favoreçam a organização, reutilização e modularização do código.

Nesse cenário, entram em cena as `funções` definidas pelo usuário, construídas com a palavra-chave `def`. As funções permitem agrupar um conjunto de instruções sob um nome, possibilitando que esse bloco de código seja executado sempre que necessário, com ou sem parâmetros de entrada, e opcionalmente retornando um resultado.

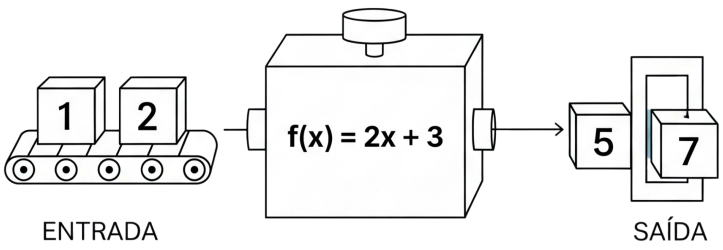
A seguir, estudaremos como definir e utilizar funções em Python, entendendo sua sintaxe, aplicabilidade e importância na construção de algoritmos mais estruturados e reutilizáveis.

3.2 Introdução ao Uso de Funções em Python

Em Matemática, uma função é uma relação entre dois conjuntos, em que cada elemento do domínio associa-se a um único elemento da imagem. Em outras palavras, para cada elemento do conjunto de entrada, existe uma correspondência com um único elemento do conjunto de saída.

Por exemplo, considere $f : \mathbb{R} \to \mathbb{R}$, uma função definida por $f(x) = 2x + 3$, como ilustrado na Figura 3.

Figura 3 — Representação gráfica de uma função afim



Fonte: Elaboração própria (2025).

De forma semelhante, em Python, uma função também estabelece uma relação entre entradas (parâmetros) e uma saída (valor retornado). A diferença está no contexto: enquanto na Matemática lidamos com símbolos e expressões, na programação operamos com instruções e algoritmos. Pensar em funções na programação é como imaginar uma máquina que recebe ingredientes (parâmetros), executa uma receita (código) e entrega um resultado (valor de retorno).

Em Python, utiliza-se o comando `def` para definir uma função. Sua sintaxe básica é:

```
def nome_da_função(parametro1, parametro2, ...):
```

Uma construção comum é o uso do comando `return`, que encerra a execução da função e devolve um valor. Esse valor pode ser de qualquer tipo: um número, uma string, uma lista, um dicionário, uma tupla, um valor booleano ou mesmo o resultado de uma expressão mais complexa. É importante respeitar a indentação e a sintaxe correta, como os dois pontos e os parênteses envolvendo os parâmetros.

A seguir, apresenta-se um exemplo de função que calcula a média aritmética entre três valores numéricos:

```
1 def média(n1, n2, n3):
2     return (n1 + n2 + n3) / 3
3
4 resultado = média(7.5, 8.0, 6.0)
5 print("Média final:", resultado)
```

Código 33 — Função para calcular média

Saída esperada:

```
Media final: 7.166666666666667
```

O uso de funções em Python promove a reutilização do código, reduz a repetição de instruções e contribui significativamente para a organização e clareza dos programas. Como afirma Menezes:

> Funções são especialmente interessantes para isolar uma tarefa específica em um trecho de programa. Isso permite que a solução de um problema seja reutilizada em outras partes do programa, sem precisar repetir as mesmas linhas (Menezes, 2010, p.162).

Veja o exemplo a seguir em que construímos um código capaz de calcular o valor de uma expressão algébrica simples. É solicitado ao usuário um valor para a variável x, em seguida, o programa calcula o valor da função para esse x e retorna o resultado, o programa calcula o valor da função para esse x e retorna com o resultado. Considere $f : \mathbb{R} \to \mathbb{R}$, uma função definida por $f(x) = 2x^2 + 3x - 5$.

A função pode ser implementada em Python da seguinte maneira:

```python
def f(x):
    return 2 * x**2 + 3 * x - 5

# Exemplo de uso:
n = float(input("Digite o valor de x: "))
resultado = f(n)
print(f"f({n})={resultado}")
```

Código 34 — Função

Compreender o funcionamento das funções em Python é um passo essencial para desenvolver programas mais organizados, reutilizáveis e eficientes. Ao estabelecer relações entre entradas e saídas, as funções em programação guardam estreita semelhança com o conceito matemático de função, o que favorece conexões significativas para os estudantes. Além disso, ao escrever suas próprias funções, o aluno é convidado a estruturar soluções, antecipar resultados e refletir sobre os processos envolvidos, o que contribui para o desenvolvimento do pensamento computacional e para a aprendizagem matemática.

A partir desse entendimento, torna-se possível explorar recursos ainda mais poderosos oferecidos pela linguagem Python, como as bibliotecas. Essas coleções de funções e estruturas prontas ampliam as possibilidades de aplicação da linguagem, permitindo ao estudante focar na resolução de problemas reais sem a necessidade de reescrever códigos complexos. A seguir, discutiremos o papel das bibliotecas em Python, com destaque para NumPy e Matplotlib, que são amplamente utilizadas na articulação entre programação e conceitos matemáticos.

3.3 Bibliotecas Úteis para o Ensino de Gráficos e Funções

Bibliotecas são coleções de módulos, funções e objetos prontos para o uso, desenvolvidas para facilitar e acelerar o desenvolvimento de programas. Elas funcionam como uma espécie de "caixa de ferramentas"para programadores, permitindo reutilizar códigos já testados e otimizados, sem a necessidade de criar o código do zero para solucionar problemas comuns.

As bibliotecas podem ser criadas por programadores individuais, comunidades de código aberto, equipes de desenvolvimento ou empresas e organizações. Essas bibliotecas são disponibilizadas em repositórios públicos, como o Python Package Index (PyPI), que é um repositório oficial de softwares para a linguagem Python (Python Software Foundation, 2025).

Portanto, ao contrário do que muitos iniciantes pensam, não é necessário escrever todas as funcionalidades do zero. O uso de bibliotecas torna eficiente a manipulação de arquivos, o tratamento de dados, a realização de operações matemáticas, a construção e visualização de gráficos, entre outras tarefas, conforme as necessidades específicas de cada projeto.

O uso de bibliotecas está relacionado ao conceito de modularização, ou seja, organizar o código em partes menores e reutilizáveis. À medida que o código se torna mais complexo, escrever tudo do zero em cada projeto torna-se inviável. Por isso, comunidades e desenvolvedores passaram a criar bibliotecas para resolver tarefas comuns, compartilhando conhecimento e facilitando o trabalho dos programadores.

Dentre as inúmeras bibliotecas disponíveis em Python, este trabalho destaca três que articulam muito bem os conceitos de matemática e programação: `NumPy`, `Matplotlib` e `Pandas`. Nas próximas seções, exploraremos o que são essas bibliotecas, seus principais recursos e, na seção 4, aplicaremos seu uso em uma proposta educacional que promove o desenvolvimento do pensamento matemático.

NumPy

NumPy (Numerical Python) é uma biblioteca de código aberto fundamental para a computação científica em Python. Seu principal diferencial é a capacidade de armazenar e manipular dados em múltiplas dimensões de maneira rápida e eficiente, superando o desempenho das listas tradicionais da linguagem. Além disso, oferece um conjunto robusto de funções que operam diretamente sobre essas estruturas de dados, o que facilita a realização de operações matemáticas e estatísticas vetorizadas. NumPy constitui a base para grande parte do ecossistema de ciência de dados em Python, sendo utilizado em diversas bibliotecas amplamente empregadas em pesquisa e desenvolvimento, como Pandas e Matplotlib.

A origem da biblioteca remonta ao final da década de 1990, quando Travis E. Oliphant, então estudante de pós-graduação na área de imagens biomédicas, buscava ferramentas computacionais para analisar grandes volumes de dados provenientes de exames de ressonância magnética e ultrassom. À época, conheceu a linguagem Python e sua extensão numérica chamada Numeric, criada por Jim Hugunin em 1995. Impressionado pelas potencialidades da linguagem, Oliphant passou a utilizá-la em sua pesquisa, estendendo a funcionalidade do Numeric conforme suas necessidades. Quando encontrava limitações, recorria a códigos escritos em linguagens como C e Fortran, adaptando-os à nova estrutura (Oliphant, 2006).

Esse processo de aprimoramento contínuo, aliado ao caráter colaborativo do software livre, culminou em 2005 na criação do NumPy. A biblioteca foi desenvolvida com base no código do Numeric, incorporando melhorias e novas funcionalidades a partir das necessidades reais de pesquisadores que utilizavam Python em seus trabalhos científicos. Muitos desses colaboradores não possuíam formação formal em ciência da computação, mas contribuíram ativamente com soluções e ideias, movidos pelo interesse comum em criar ferramentas acessíveis, eficientes e abertas para a pesquisa científica. Desde então, o NumPy consolidou-se como um dos pilares da computação numérica moderna, sendo amplamente utilizado no ensino, na pesquisa e no desenvolvimento tecnológico.

Em programação, um *array* é uma estrutura de dados utilizada para armazenar e acessar coleções de elementos do mesmo tipo. Dependendo da sua dimensão, pode ser interpretado como um vetor (unidimensional), uma matriz (bidimensional) ou, de forma mais geral, como um *array* de múltiplas dimensões. No NumPy, esse tipo de estrutura é representado pelo objeto `ndarray`, que constitui o núcleo da biblioteca.

A principal vantagem em utilizar *arrays* é a eficiência na execução de operações matemáticas, especialmente quando se lida com grandes volumes de dados. O uso dessa estrutura permite aplicar cálculos de forma vetorizada, isto é, diretamente sobre coleções inteiras, sem a necessidade de laços repetitivos (*loops*). Além de tornar o código mais conciso e legível, essa abordagem reduz o tempo de execução, o que é essencial em tarefas como geração de gráficos ou processamento de dados científicos. Por essa razão, em

diversos contextos, os *arrays* do NumPy substituem com vantagens as listas tradicionais do Python.

Importação da biblioteca NumPy

Em Python, as bibliotecas não são carregadas automaticamente. Esse comportamento visa manter os programas mais leves e eficientes, utilizando apenas os recursos estritamente necessários. Para utilizar qualquer biblioteca, é preciso importá-la previamente no início do código, por meio do comando `import`, que informa ao interpretador que o programa fará uso de funções ou recursos pertencentes àquela biblioteca. O exemplo a seguir demonstra como importar a biblioteca NumPy:

```
1 import numpy as np
```

Código 35 — importação da biblioteca NumPy

Como o nome completo da biblioteca pode ser extenso, é comum atribuir um apelido mais curto, como `np`, o que facilita sua utilização ao longo do código. Isso permite acessar as funcionalidades do NumPy utilizando a notação `np.função()`, prática amplamente adotada na comunidade Python por tornar o código mais legível e conciso.

No quadro 4 são apresentadas algumas das principais funções do pacote NumPy que serão utilizadas na seção 4. Como a proposta deste trabalho envolve a criação e manipulação de dados para representação gráfica, essas funções serão fundamentais para gerar intervalos, aplicar transformações matemáticas e estruturar conjuntos de valores. Cada uma dessas funções será retomada nas seções posteriores, durante o estudo da biblioteca Matplotlib, quando seu uso será exemplificado em contextos práticos.

Quadro 4 – Principais funções da biblioteca NumPy

Função (NumPy)	**O que faz**	**Exemplo de uso**
`np.array()`	Cria um array a partir de uma lista ou tupla	`np.array([1, 2, 3])`
`np.linspace()`	Gera valores igualmente espaçados em um intervalo	`np.linspace(0, 10, 100)`
`np.arange()`	Gera valores com passo fixo dentro de um intervalo	`np.arange(0, 10, 0.5)`
`np.sin()`	Aplica a função seno a cada elemento do array	`np.sin(x)`
`np.cos()`	Aplica a função cosseno a cada elemento do array	`np.cos(x)`
`np.tan()`	Aplica a função tangente a cada elemento do array	`np.tan(x)`
`np.exp()`	Aplica a função exponencial (e^x)	`np.exp(x)`
`np.log()`	Aplica o logaritmo natural (ln)	`np.log(x)`
`np.sqrt()`	Calcula a raiz quadrada de cada elemento	`np.sqrt(x)`
`np.abs()`	Retorna o valor absoluto de cada elemento	`np.abs(x)`
`np.pi`	Constante matemática π	`np.sin(np.pi / 2)`

Fonte: Elaboração própria (2025).

Pandas

Outra biblioteca fundamental para quem utiliza a linguagem de programação Python é a `Pandas`. Trata-se de uma biblioteca de código aberto (*open source*) desenvolvida com o objetivo de tornar a manipulação e a análise de dados mais simples, poderosa e flexível. A `Pandas` foi criada por Wes McKinney em 2008, enquanto trabalhava na AQR Capital Management, para suprir a necessidade de uma ferramenta eficiente para análise quantitativa de dados financeiros. O nome "Pandas"deriva de *panel data*, um termo estatístico usado para se referir a conjuntos de dados organizados em duas dimensões (observações ao longo do tempo e entre diferentes unidades). Em 2009, a biblioteca foi lançada publicamente como software livre e, desde 2015, é mantida por uma comunidade ativa com apoio da organização *NumFOCUS* (Pandas Development Team, 2024).

A biblioteca `Pandas` permite trabalhar com tabelas de dados em diversos formatos, como arquivos `.csv`, planilhas do Excel, bancos de dados em SQL, entre outros. Sua versatilidade a torna útil em diferentes áreas do conhecimento, como ciência de dados, estatística, finanças ou qualquer área que envolva a análise de grandes volumes de dados, realizando-a de forma organizada e visual.

No contexto educacional, `Pandas` oferece aos estudantes um primeiro contato com a análise e manipulação de dados estruturados em forma de tabela. Isso permite associar o uso da programação ao trabalho com informações numéricas e categóricas, promovendo habilidades como a leitura e interpretação de dados. Além disso, quando combinada com estruturas básicas da programação, como o laço `for`, permite que os alunos acompanhem as iterações de maneira estruturada, organizando os resultados em tabelas de fácil leitura.

O objetivo deste trabalho não é explorar profundamente todas as funcionalidades da biblioteca, mas oferecer um contato inicial que permita associá-la a conceitos matemáticos já presentes no currículo escolar. A ideia principal é preparar os dados com o `Pandas` e, posteriormente, associá-los à sua representação gráfica com a biblioteca `Matplotlib`, que será abordada na próxima seção.

A estrutura de dados da biblioteca `Pandas` está baseada em dois principais objetos: `Series` e `DataFrame`. O primeiro representa uma estrutura unidimensional, semelhante a uma lista, com índices rotulados. O segundo, o `DataFrame`, é uma estrutura bidimensional, mais comumente utilizada, que se assemelha a uma planilha eletrônica, composta por linhas e colunas identificadas. Por essa razão, o `DataFrame` será o foco principal desta introdução, por ser ideal para organizar, visualizar e manipular dados.

Importação da biblioteca Pandas

Assim como no uso de qualquer biblioteca externa em Python, é necessário realizar sua importação no início do código. A convenção mais comum é utilizar o apelido `pd`, como mostrado abaixo:

```
1 import pandas as pd
```

Código 36 — importação da biblioteca Pandas

O uso de um nome abreviado torna o código mais conciso, especialmente quando se utilizam diversas funções da biblioteca ao longo do programa.

DataFrame

A principal estrutura de dados da biblioteca é o `DataFrame`, cuja sintaxe básica é:

```
DataFrame(data, index, columns, dtype, copy)
```

Os principais parâmetros são:

- `data`: fonte principal dos dados (pode ser uma lista, dicionário, array, outro `DataFrame` ou arquivo externo).
- `index` (opcional): rótulos para as linhas. Caso não seja definido, o índice padrão é numérico (`0, 1, 2, ...`).
- `columns` (opcional): rótulos para as colunas.
- `dtype` (opcional): tipo de dado dos elementos (`int`, `float`, `str`, etc.).
- `copy` (opcional): se `True`, cria uma cópia dos dados para evitar alterações indesejadas no objeto original.

Para fins didáticos, será utilizada apenas a opção `columns`, a fim de organizar melhor a tabela.

No exemplo a seguir, criaremos uma tabela que exibe os números naturais de 1 a 10, seus quadrados e seus cubos, utilizando o laço `for` para gerar os dados e organizando-os posteriormente em um `DataFrame`:

```
import pandas as pd
dados = []

for i in range(1,11):
  dados.append([i,i**2,i**3])

tabela = pd.DataFrame(dados,columns=['Número','Quadrado','Cubo'])
print(tabela)
```

Código 37 — Pandas

O código acima gera a seguinte tabela:

```
   Numero  Quadrado  Cubo
0       1         1     1
1       2         4     8
2       3         9    27
3       4        16    64
4       5        25   125
5       6        36   216
6       7        49   343
7       8        64   512
8       9        81   729
9      10       100  1000
```

O uso do `Pandas` permite aos estudantes organizar dados de maneira estruturada, facilitando a interpretação de informações numéricas e categóricas em formato de tabela. Essa organização, por sua vez, abre caminho para análises mais visuais, tornando-se especialmente útil quando associada à representação gráfica. Nesse sentido, a seguir será abordada a biblioteca `Matplotlib`, que possibilita a criação de diferentes tipos de gráficos a partir dos dados previamente estruturados, ampliando as possibilidades de visualização e compreensão dos fenômenos matemáticos e do mundo real.

Matplotlib: Criação de Gráficos

Matplotlib é uma biblioteca de código aberto (open source) voltada para a criação de gráficos estáticos, animados e interativos na linguagem Python. Foi criada em 2003 por John D. Hunter, que se inspirou em funcionalidades gráficas do MATLAB, uma linguagem de programação projetada para matemática computacional, análise de dados, visualização e desenvolvimento de algoritmos (Mathworks, 2024). Hunter considerava o MATLAB bastante eficiente na geração de gráficos visualmente atrativos. No entanto, à medida que suas necessidades de manipulação de dados se tornaram mais complexas, identificou limitações nessa ferramenta. Diante disso, desenvolveu a Matplotlib como alternativa em Python, linguagem já conhecida por sua simplicidade sintática.

De acordo com Matplotlib Development Team (2025), Matplotlib foi concebida com a filosofia de que gráficos simples devem poder ser criados com poucos comandos, até mesmo com apenas uma linha de código. Desde então, consolidou-se como uma das bibliotecas mais utilizadas para visualização de dados em Python, permitindo a criação de gráficos variados, como gráficos de linha, barras, histogramas, dispersão, gráficos 3D, mapas de calor, entre outros.

A documentação oficial da biblioteca é bastante completa e detalha todas as funcionalidades, métodos e argumentos disponíveis. Nesta seção, serão apresentados apenas os comandos considerados essenciais para o contexto desta proposta didática. Para aprofundamentos, recomenda-se a consulta ao site oficial (https://matplotlib.org/stable/), que oferece uma ampla variedade de exemplos e tutoriais sobre o uso da Matplotlib.

Importação da biblioteca Matplotlib

Para utilizar os recursos gráficos da biblioteca Matplotlib, é necessário realizar sua importação no início do código. A estrutura mais comum envolve a importação do módulo `pyplot`, responsável por funções de plotagem, com o apelido `plt`, como mostra o exemplo a seguir:

```
1 import matplotlib.pyplot as plt
```

Código 38 — Importação da biblioteca Matplotlib

O uso de um nome abreviado, como `plt`, torna o código mais limpo e prático, especialmente em situações em que diversas funções de plotagem são utilizadas. Essa convenção é amplamente adotada pela comunidade Python, garantindo padronização e legibilidade no desenvolvimento de gráficos.

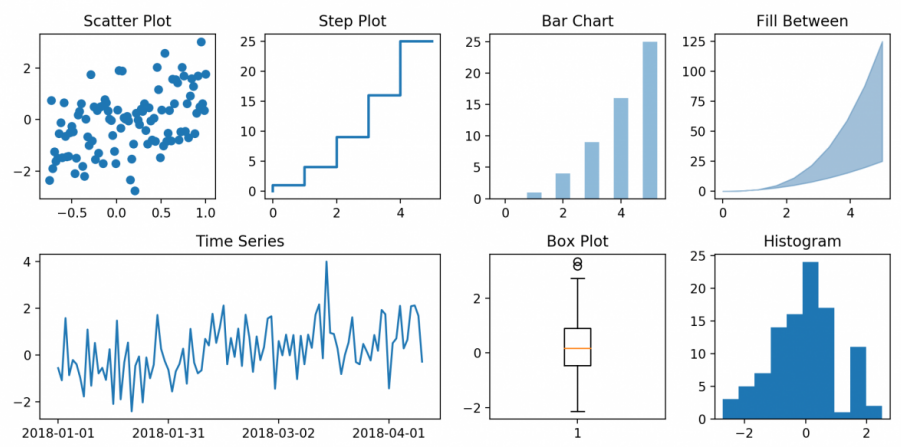
É importante compreender a diferença entre importar o pacote principal (`matplotlib`) e importar o módulo específico `matplotlib.pyplot`. O pacote `matplotlib` representa a biblioteca como um todo, composta por diversos módulos responsáveis por funcionalidades variadas relacionadas à construção e manipulação de gráficos. Contudo, em contextos práticos e educacionais, raramente é necessário importar todo o pacote, pois isso não fornece acesso direto às funções de criação de gráficos. Por esse motivo, é mais comum importar apenas o submódulo `pyplot`.

O `pyplot` fornece uma interface simplificada, inspirada no estilo do software MATLAB, com funções como `plot()`, `title()`, `xlabel()`, `ylabel()` e `show()`. Essas funções são amplamente utilizadas na criação e exibição de gráficos em ambientes educacionais e científicos. Dessa forma, ao importar apenas o `pyplot`, o programador carrega os recursos essenciais de maneira mais leve e objetiva, otimizando a construção de visualizações com clareza e eficiência.

Tipos de Gráficos

Abaixo estão alguns exemplos de gráficos que podem ser feitos utilizando a biblioteca Matplotlib: gráfico de dispersão (scatter plot), gráfico de linhas (line plot), gráfico de barras (bar plot), histograma (histogram), gráfico de caixa (box plot), entre outros.

Figura 4 — Tipos de gráficos da biblioteca Matplotlib



Fonte: Coding Blocks (2021)

Gráfico de linhas

O comando principal para a criação de gráficos simples é o `plot()`, utilizado para gerar gráficos de linha. A biblioteca Matplotlib trabalha com gráficos 2D construídos a partir de estruturas chamadas arrays. Um array pode ser entendido como um "vetor"ou "matriz", ou seja, uma estrutura de dados que armazena uma sequência de elementos organizados em posições numeradas chamadas índices. Em Python, uma lista é um exemplo de array, embora, para cálculos com maior desempenho, seja recomendada a utilização de arrays fornecidos por bibliotecas como o NumPy.

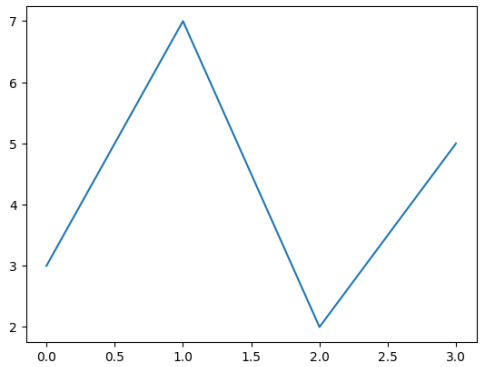
Para criar um gráfico, é necessário definir uma lista (ou array), escolher o tipo de gráfico desejado e utilizar o comando `plt.show()`, que é responsável por exibir o gráfico na tela. Até que esse comando seja executado, o Python mantém o gráfico em memória, mas não o renderiza visualmente para o usuário. Veja o exemplo a seguir:

```
1 import matplotlib.pyplot as plt
2 lista = [3,7,2,5]
3
4 # Plotando o gráfico
5 plt.plot(lista)
6 plt.show()
```

Código 39 — gráfico de linha

Esse trecho de código produz a saída mostrada a seguir:

Figura 5 — Gráfico de linha gerado com Matplotlib



Fonte: Elaboração própria (2025).

Todo gráfico é formado por coordenadas horizontais e verticais. No exemplo acima, como foi especificada apenas uma variável, o Matplotlib atribuiu automaticamente os valores do eixo horizontal, iniciando em 0 até $n - 1$, em que n representa o número de elementos da lista. É fundamental garantir que os arrays utilizados em um gráfico tenham o mesmo comprimento. Caso contrário, o gráfico não será gerado corretamente.

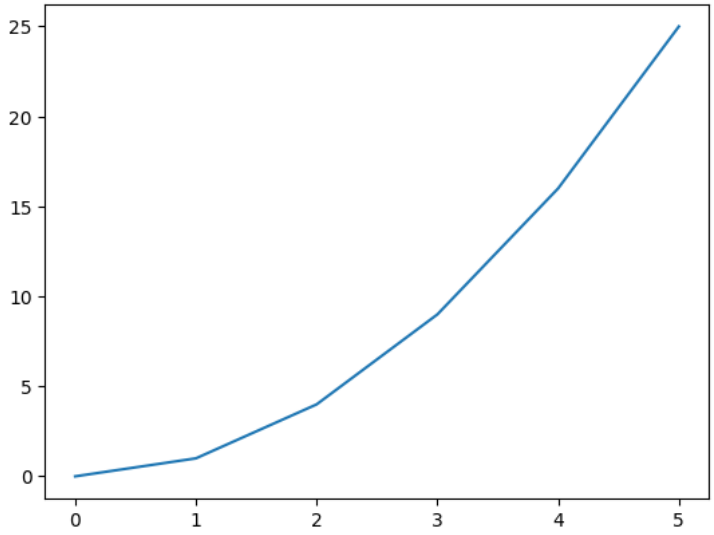
A seguir, apresenta-se um exemplo de construção do gráfico de uma função quadrática. Considere a função $f : \mathbb{R} \to \mathbb{R}$ definida por $f(x) = x^2$. O código abaixo utiliza a estrutura `range` para gerar os valores de entrada:

```
1 x = range(6)
2 y = [xi**2 for xi in x]
3
4 # Plotando o gráfico
5 plt.plot(x,y)
6 plt.show()
```

Código 40 — Gráfico utilizando range

A execução do código acima produz o seguinte gráfico:

Figura 6 — Gráfico gerado com `range`



Fonte: Elaboração própria (2025).

Observa-se que o gráfico não representa visualmente uma parábola suave, como seria esperado da função quadrática. Em vez disso, o que se vê é uma sequência de segmentos de reta conectando os poucos pontos calculados. Isso ocorre porque estão sendo utilizados apenas seis pontos para construir a curva, o que resulta em uma representação poligonal pouco precisa.

Para obter uma curva mais contínua e próxima da forma parabólica, é necessário aumentar a quantidade de pontos no domínio. No entanto, a função `range` trabalha exclusivamente com números inteiros e não permite a definição de incrementos fracionários. Para contornar essa limitação, pode-se utilizar a função `arange`, da biblioteca `NumPy`, que funciona de maneira semelhante ao `range`, mas aceita intervalos com valores decimais.

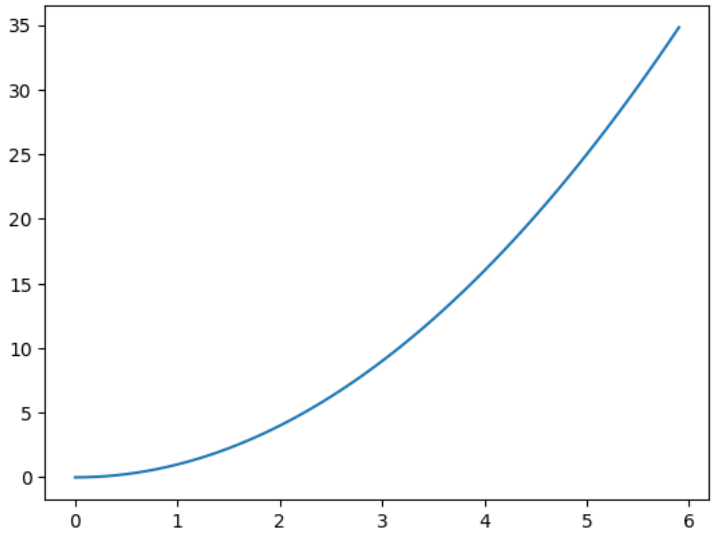
No exemplo a seguir, a função é avaliada no intervalo $0 \leqslant x \leqslant 5$, com incremento de 0,1, o que proporciona uma maior densidade de pontos e, consequentemente, uma representação gráfica mais suave:

```
1 import numpy as np
2
3 x = np.arange(0,5,0.1)
4 y = [xi**2 for xi in x]
5
6 # Plotando o gráfico
7 plt.plot(x,y)
8 plt.show()
```

Código 41 — Gráfico utilizando arange

A execução do código acima resulta no gráfico a seguir:

Figura 7 — Gráfico gerado com `np.arange()`



Fonte: Elaboração própria (2025).

Neste segundo gráfico, a função quadrática apresenta uma curvatura mais fiel à sua forma teórica. A utilização de intervalos menores, como os fornecidos por `np.arange()`, é fundamental para representar graficamente o comportamento de funções contínuas de forma mais precisa.

Uma alternativa ao uso de incrementos fixos, como ocorre com o comando `arange`, consiste em determinar diretamente a quantidade de pontos que se deseja dentro de um intervalo específico. Para isso, a biblioteca `NumPy` oferece a função `linspace()`, que gera uma sequência de valores igualmente espaçados entre dois limites.

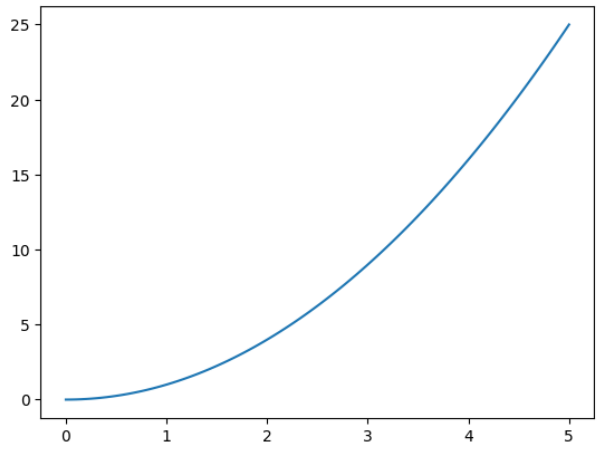
Suponha que se deseja gerar 100 pontos igualmente distribuídos no intervalo $0 \leqslant x \leqslant 5$. O código a seguir demonstra essa construção:

```
1 import numpy as np
2
3 x = np.linspace(0,5,100)
4 y = [xi**2 for xi in x]
5
6 # Plotando o gráfico
7 plt.plot(x,y)
8 plt.show()
```

Código 42 — Gráfico utilizando linspace

O código gera o seguinte gráfico da Figura 8:

Figura 8 — Gráfico gerado com `np.linspace()`



Fonte: Elaboração própria (2025).

É importante notar que, nos exemplos anteriores, utilizamos listas por compreensão para calcular os valores de y. No entanto, uma alternativa mais apropriada e reutilizável, especialmente quando se deseja trabalhar com diferentes expressões matemáticas, consiste em definir a função explicitamente por meio da palavra-chave `def`.

Contudo, para que essa abordagem funcione corretamente, é necessário garantir que os valores de entrada estejam em uma estrutura que aceite operações vetoriais, como os arrays fornecidos por `NumPy`. Caso contrário, ao utilizar listas criadas com `list(range(0,6))`, por exemplo, haverá erro de execução, pois a operação `x**2` não é compatível com objetos do tipo `range` nem com listas convencionais em operações matemáticas diretas.

Assim, para contornar essa limitação e garantir o funcionamento adequado da função, recomenda-se utilizar `np.arange()` ou `np.linspace()`, conforme demonstrado anteriormente. Com isso, é possível construir gráficos a partir de funções definidas da seguinte forma:

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  def f(x):
5      return x**2
6
7  x = np.linspace(-10,10,100)
8  y=[]
9  for i in x:
10   y.append(f(i))
11
12 # Plotando o gráfico
13 plt.plot(x, y, color='green')
14 plt.grid()
15 plt.show()
```

Código 43 — Gráfico utilizando def

Nesse exemplo, a função `f(x)` é aplicada individualmente a cada elemento do vetor `x`, composto por 100 valores igualmente espaçados no intervalo de `-10` a `10`. O resultado é armazenado na lista `y`, que é então utilizada para gerar um gráfico suave e contínuo da função quadrática. Esse processo destaca como a combinação das bibliotecas `NumPy` e `matplotlib` pode facilitar a representação visual de funções matemáticas de forma precisa e eficiente.

Compreender como construir o gráfico de uma função matemática em Python, utilizando estruturas compatíveis com operações vetoriais, é o primeiro passo para uma representação visual eficiente. No entanto, além da correta definição da função e da escolha adequada dos valores de entrada, a clareza do gráfico pode ser significativamente aprimorada por meio de elementos visuais como cores, marcadores e a rotulagem dos eixos. Esses recursos não apenas tornam a visualização mais atrativa, como também facilitam a interpretação dos dados apresentados. A seguir, serão exploradas algumas possibilidades de personalização oferecidas pela biblioteca `matplotlib`.

Elementos do gráfico: cores, marcadores e eixos

Após a construção correta de um gráfico, é possível aprimorar sua legibilidade e atratividade por meio da personalização de seus elementos visuais. A biblioteca `matplotlib.pyplot` permite a modificação de cores, estilos de linha, marcadores e a adição de informações como título, rótulos nos eixos e legenda. A seguir, são apresentados os principais recursos disponíveis para essa personalização.

A função `plot()` permite construir gráficos de linha a partir de dois conjuntos de valores, correspondentes aos eixos x e y. Além desses dois argumentos obrigatórios, podem ser especificados parâmetros adicionais para controle estético. Abaixo, apresenta-se um exemplo da sintaxe comumente utilizada:

```
plt.plot(x, y, color='blue', marker='o', linestyle='--')
```

Por padrão, caso não se especifique uma cor, a linha será exibida em azul. No entanto, o parâmetro `color` permite a definição de qualquer cor desejada. As cores podem ser informadas por meio de nomes em inglês, abreviações com uma única letra ou códigos hexadecimais. O quadro 5 apresenta alguns exemplos:

Quadro 5 – Cores no `matplotlib`

Cor	Nome (string)	Código abreviado
Azul	`'blue'`	`'b'`
Vermelho	`'red'`	`'r'`
Verde	`'green'`	`'g'`
Preto	`'black'`	`'k'`
Amarelo	`'yellow'`	`'y'`
Roxo	`'purple'`	—
Laranja	`'orange'`	—

Fonte: Elaboração própria (2025).

Além das cores pré-definidas pela biblioteca, é possível utilizar códigos hexadecimais para definir cores com maior precisão, como por exemplo `#FF5733`. Esses códigos podem ser consultados em plataformas online, como o site `https://htmlcolorcodes.com`, que disponibiliza uma paleta interativa para a seleção e cópia da cor desejada.

A utilização de marcadores também pode ser bastante útil, especialmente quando se deseja evidenciar os pontos do gráfico ou diferenciar visualmente diferentes conjuntos de dados. Os marcadores contribuem para tornar a leitura e interpretação mais claras, sobretudo em gráficos com poucos dados ou com múltiplas curvas sobrepostas. Em contextos comparativos, eles auxiliam na distinção entre as diferentes representações. O quadro 6 apresenta alguns exemplos de marcadores disponíveis:

Quadro 6 – Marcadores

Código	**Símbolo**	**Descrição**
$'o'$	●	Círculo
$'\wedge'$	▲	Triângulo para cima
$'s'$	■	Quadrado
$'x'$	$\wedge$	Cruz
$'+'$	$+$	Sinal de mais
$'*'$	★	Asterisco
$'D'$	♦	Losango

Fonte: Elaboração própria (2025).

Ademais, é possível modificar o estilo da linha por meio do parâmetro `linestyle`, que altera o traçado entre os pontos do gráfico. O quadro 7 apresenta alguns exemplos de estilos disponíveis:

Quadro 7 – Estilos de linha

Código	**Exemplo visual**	**Descrição**
`'-'`	———————	Linha contínua
`'- -'`	– – – –	Linha tracejada
`':'`	. . . . .	Linha pontilhada
`'-.'`	–·–·–	Traço e ponto

Fonte: Elaboração própria (2025).

No que se refere à melhoria da legibilidade e compreensão dos gráficos, também é possível incluir informações adicionais como título, rótulo dos eixos e grade. O comando `plt.title('Título do gráfico')` adiciona um título à figura, enquanto os comandos `plt.xlabel('Eixo X')` e `plt.ylabel('Eixo Y')` permitem nomear os eixos horizontal e vertical, respectivamente. Por padrão, os gráficos não exibem linhas de grade, entretanto, é possível ativá-las com o comando `plt.grid(True)`.

Além da curva principal representada pela função, pode ser desejável adicionar linhas auxiliares ao gráfico, como o eixo x (linha horizontal em $y = 0$) ou o eixo y (linha vertical em $x = 0$). A biblioteca `matplotlib.pyplot` oferece comandos específicos para inserir essas linhas de forma prática:

```
plt.axhline(y=0, color='black', linestyle='--')
plt.axvline(x=0, color='black', linestyle='--')
```

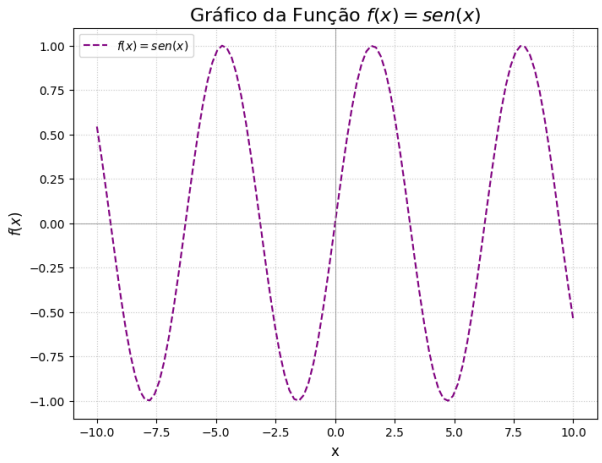
Esses recursos visuais contribuem significativamente para a interpretação de características relevantes da função, como simetria, raízes, interseções ou variações em torno de determinados pontos. A seguir, apresenta-se um exemplo de gráfico da função seno, no qual são combinados os principais elementos abordados nesta subseção:

```python
import matplotlib.pyplot as plt
import numpy as np

def plot_função(func):
  x = np.linspace(-10, 10, 100)
  y = [func(i) for i in x]

  plt.figure(figsize=(8, 6))
  plt.plot(x, y, label='$f(x) = sen(x)$', color='purple', linestyle='--')
  plt.title('Gráfico da Função $f(x) = sen(x)$', fontsize=16)
  plt.xlabel('x', fontsize=12)
  plt.ylabel('$f(x)$', fontsize=12)
  plt.grid(True, linestyle=':', alpha=0.7)
  plt.axhline(0, color='gray', linewidth=0.5) # Adiciona eixo x
  plt.axvline(0, color='gray', linewidth=0.5) # Adiciona eixo y
  plt.legend()
  plt.show()

def minha_função(x):              # Define a função f(x) = sen(x)
    return np.sin(x)

plot_função(minha_função)         # Chama a função para plotar o gráfico
```

Código 44 — função seno

O código gera o seguinte gráfico:

Figura 9 — Função Seno



Fonte: Elaboração própria (2025).

Portanto, a biblioteca `matplotlib`, como vimos ao longo desta seção, oferece uma variedade de recursos que permitem a construção e personalização de gráficos matemáticos com grande flexibilidade e qualidade visual. Ao integrar a linguagem Python ao ensino de Matemática, especialmente por meio de recursos gráficos como os apresentados, abre-se um espaço fértil para o desenvolvimento de competências do século XXI, como o pensamento computacional, a interpretação de dados, o raciocínio lógico e a autonomia na resolução de problemas. Além disso, a visualização gráfica de funções e dados contribui para tornar os conceitos matemáticos mais acessíveis e significativos aos estudantes do Ensino Médio.

Na seção 4 apresenta-se a proposta didática de uma eletiva que materializa o uso do Python no contexto escolar. A sequência de atividades foi elaborada com base nos princípios discutidos até aqui, buscando não apenas aprofundar conteúdos matemáticos, mas também proporcionar experiências investigativas, criativas e conectadas ao mundo real. O objetivo é oferecer aos estudantes um ambiente de aprendizagem interdisciplinar, colaborativo e alinhado às competências do século XXI.

4 PROPOSTA DIDÁTICA: CONSTRUINDO GRÁFICOS COM PYTHON NO ENSINO MÉDIO

Esta seção apresenta uma proposta didática para uma eletiva focada no ensino da manipulação de gráficos e tabelas por meio da linguagem de programação Python. As atividades foram elaboradas a fim de proporcionar aos estudantes um primeiro contato com a programação, ao mesmo tempo em que aprofundam conteúdos matemáticos do Ensino Médio, especialmente relacionados à construção e interpretação de gráficos de funções. Além disso, a proposta foi elaborada para ser facilmente adaptável a outros temas, levando em consideração que as eletivas devem ser trabalhadas de forma interdisciplinar, promovendo a articulação entre diferentes áreas do conhecimento, e também de forma transdisciplinar, integrando saberes para a compreensão e resolução de problemas complexos.

Para facilitar a aplicação das atividades, recomenda-se o uso da plataforma gratuita Google Colaboratory (Google, 2024), um serviço de nuvem que permite a escrita e execução de códigos em Python diretamente pelo navegador, sem a necessidade de instalação de softwares. Essa característica torna possível o uso em diferentes dispositivos, inclusive smartphones, ampliando o acesso dos estudantes.

Nesta seção, são apresentadas sete atividades que envolvem a criação de gráficos e tabelas, abordando temas diversos do currículo de Matemática do Ensino Médio, com ênfase no estudo de funções. Com o objetivo de desenvolver as competências específicas da área de Matemática e suas Tecnologias, conforme estabelecido pela Base Nacional Comum Curricular (BNCC), cada atividade está acompanhada das habilidades correspondentes que são mobilizadas durante sua execução.

As atividades são estruturadas a partir de questões contextualizadas, elaboradas com o apoio de inteligência artificial, e baseiam-se em conjuntos de dados fictícios inspirados nos Objetivos de Desenvolvimento Sustentável (ODS) da Organização das Nações Unidas. Por meio da modelagem matemática desses dados, espera-se que o estudante construa significado sobre os conceitos abordados e desenvolva a capacidade de resolver problemas.

Sugere-se que o desenvolvimento das atividades inicie com a construção conjunta do código, considerando que esta é a primeira experiência dos estudantes com a programação. Nessa etapa, ocorre uma interação contínua entre professor e alunos, que parte do problema contextualizado para a introdução dos conceitos fundamentais de Python. O professor orienta a escrita do código, explicando detalhadamente cada comando e auxiliando na identificação e correção de possíveis erros durante a construção linha a linha.

Após a finalização do código, os estudantes respondem a perguntas de reflexão que estimulam a análise dos resultados obtidos. Em seguida, promove-se um momento de debate e troca de ideias, que denominamos “Socialização”, no qual os alunos compartilham suas interpretações e aprofundam a compreensão dos conceitos trabalhados.

4.1 Atividade 1: Familizariando com a Biblioteca Matploblib

Tempo estimado: 100 min
Objetivos da atividade:

- Compreender e utilizar funções básicas da biblioteca matplotlib para a construção de gráficos em Python, a partir de dados contextualizados.
- Desenvolver habilidades de leitura, construção e interpretação de tabelas e gráficos de frequências, utilizando dados fictícios relacionados aos Objetivos de Desenvolvimento Sustentável (ODS).

Habilidades da BNCC

- (EM13MAT405) Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.
- (EM13MAT406) Construir e interpretar tabelas e gráficos de frequências com base em dados obtidos por meio de pesquisas amostrais, utilizando ou não softwares que inter-relacionem estatística, geometria e álgebra.
- (EM13MAT501) Investigar relações numéricas expressas em tabelas para representá-las no plano cartesiano, identificando padrões, formulando conjecturas e generalizando algebricamente essas relações, reconhecendo, quando aplicável, a representação de uma função polinomial do 1º grau.

Matplotlib é uma biblioteca em Python que possibilita criar gráficos e visualizar dados. No estudo de funções, a representação gráfica constitui parte fundamental. Em diferentes áreas de conhecimento, para melhor interpretar e conjecturar a cerca de um conjunto de dados, organiza-se em um plano cartesiano e busca-se encontrar regularidades e padrões.

Inicialmente, propõe-se a construção conjunta de um código para a geração de um gráfico, envolvendo a interação entre professor e estudantes a partir de um problema contextualizado. Durante essa etapa, o professor apresenta a problemática e introduz conceitos fundamentais de Python, como listas, tamanho de listas e elementos do gráfico.

Em seguida, os estudantes analisam o gráfico gerado, promovendo um debate sobre sua interpretação e as informações extraídas. Posteriormente, a turma é dividida em grupos, e cada grupo recebe um problema relacionado à construção de gráficos, vinculado a algum Objetivo de Desenvolvimento Sustentável (ODS). Após a elaboração dos gráficos, os grupos compartilham seus resultados com a turma, respondendo às perguntas propostas sobre a análise dos dados apresentados.

Contextualização: Campanha de Consumo Consciente

O problema relaciona dados envolvendo a ODS 12 - Consumo e Produção Responsáveis. Uma organização ambiental lançou uma campanha na internet para incentivar o consumo consciente de energia elétrica, utilizando a *hashtag* #EconomizeEnergia. Um mês após o lançamento, a equipe responsável decidiu analisar o impacto da campanha nas redes sociais. Para isso, um analista reuniu o número de vezes que a *hashtag* foi mencionada a cada dia ao longo do mês.

Tabela 8 — Menções da *hashtag* #EconomizeEnergia

Dia	Menções	Dia	Menções	Dia	Menções
1	125	11	1125	21	432
2	140	12	1754	22	398
3	189	13	1623	23	312
4	215	14	1540	24	289
5	261	15	1602	25	370
6	274	16	1485	26	1054
7	263	17	1298	27	1510
8	732	18	643	28	1402
9	856	19	473	29	1389
10	995	20	452	30	1356

Fonte: Elaboração própria (2025).

Os dados foram organizados na tabela 8 e devem ser representados em um gráfico para facilitar a análise.

1º Passo: Definir valores para os eixos por meio de listas

Por ser um primeiro contato, a ideia é que construção das listas seja realizada manualmente. Em seguida, levantar o questionamento: há uma função no Python que possibilite criar lista de maneira mais eficiente?

```
dia= [1,2,3,4,5,6,7,8,9,10,11,12,12,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30]
menção = [125, 140, 189, 215, 261, 274, 263, 732, 856, 995, 1125,
    1754, 1623, 1540, 1602, 1485, 1298, 643, 473, 452, 432, 398,
    312, 289, 370,1054, 1510, 1402, 1389, 1356]
```

Código 45 — listas

Se utilizarmos a função `list(range(1,30,1))`, temos:

```
dia= list(range(1,30,1))
menção = [125, 140, 189, 215, 261, 274, 263, 732, 856, 995, 1125,
    1754, 1623, 1540, 1602, 1485, 1298, 643, 473, 452, 432, 398,
    312, 289, 370,1054, 1510, 1402, 1389, 1356]
```

Código 46 — listas range

2º Passo: Verificar se as listas possuem o mesmo tamanho

Para que o gráfico seja gerado corretamente, as listas que representam os eixos (por exemplo, dia e menções) devem possuir a mesma quantidade de elementos. Caso contrário, ocorrerá um erro, pois não será possível estabelecer uma correspondência entre os valores.

Para verificar essa condição, utiliza-se a função `len(nome_da_variavel)`, que retorna o número de elementos de uma lista.

A comparação entre os tamanhos das listas pode ser feita utilizando o operador de igualdade (==). Se a saída da verificação for 'True', significa que as listas possuem o

mesmo número de elementos e podem ser utilizadas na plotagem do gráfico. Caso o resultado seja 'False', indica-se que há uma inconsistência nos dados, o que pode causar erros.

```
len(dia) == len(menção)
```

Código 47 — tamanho das listas

3º Passo: Plotar o gráfico

Conforme apresentado na seção anterior, a biblioteca Matplotlib oferece uma ampla variedade de opções para a visualização de dados. Para um primeiro contato, propõe-se a utilização do gráfico de dispersão, que representa a relação entre duas variáveis por meio de pontos, permitindo uma análise visual da distribuição dos dados.

Inicialmente, é necessário importar a biblioteca Matplotlib. No entanto, como seu nome é extenso, é comum utilizá-la com a abreviação plt, o que torna o código mais fluído e legível. Para criar um gráfico de dispersão, utiliza-se a função `plt.scatter()`, que plota os pontos com base nas coordenadas fornecidas. Já a função `plt.show()` é responsável por executar o programa e exibir o gráfico gerado.

```
import matplotlib.pyplot as plt

plt.scatter(dia, menção)
plt.show()
```

Código 48 — plotar o gráfico

4º Passo: Ajustes no gráfico

Para uma melhor visualização, a biblioteca Matplotlib oferece funções como `plt.xlabel()` e `plt.ylabel()`, responsáveis por adicionar legendas aos eixos x e y, respectivamente. O termo label, do inglês, significa legenda, indicando a função desses comandos. Da mesma forma, `plt.title()`, cujo nome significa título em inglês, permite adicionar um título ao gráfico.

Além disso, a função `plt.grid(True)` insere uma grade no gráfico, facilitando a visualização e a associação entre as variáveis. Embora esses ajustes sejam opcionais, eles são essenciais para construir um gráfico mais completo e informativo, destacando seus elementos principais.

```
import matplotlib.pyplot as plt

plt.scatter(dia, menção)
plt.xlabel("Dia do Mês")
plt.ylabel("Quantidade de Menções")
plt.title("Menções da Hashtag #EconomizeEnergia ao Longo do Mês")
plt.grid(True)
plt.show()
```

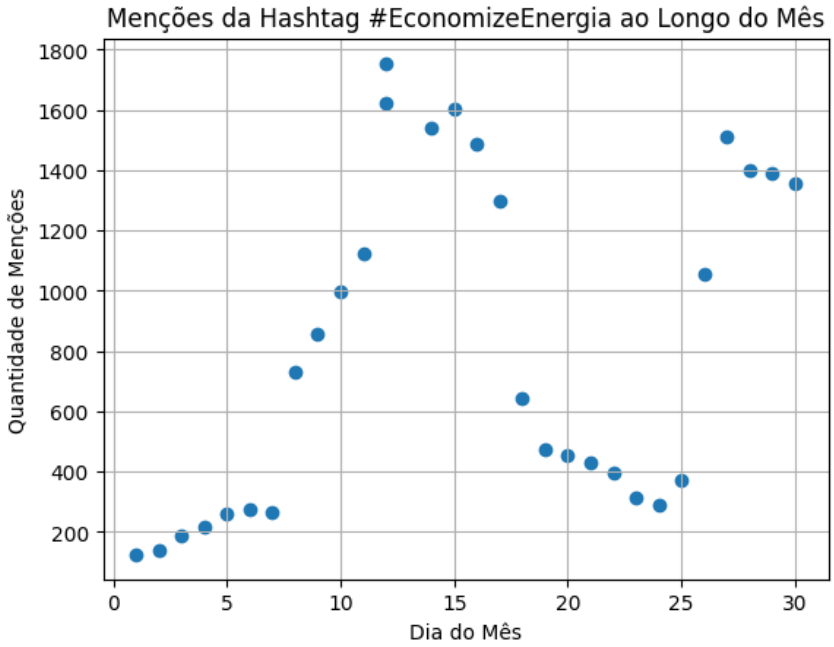
Código 49 — Ajustes gráfico

Por fim, o código final fica da seguinte forma, e sua versão compilada pode ser observada na Figura 10.

```
import matplotlib.pyplot as plt
#Listas
dia= [1,2,3,4,5,6,7,8,9,10,11,12,12,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30]
menção = [125, 140, 189, 215, 261, 274, 263, 732, 856, 995,
          1125, 1754, 1623, 1540, 1602, 1485, 1298, 643, 473, 452,
          432, 398, 312, 289, 370, 1054, 1510, 1402, 1389, 1356]

# Plotando o gráfico
plt.scatter(dia, menção)
plt.xlabel("Dia do Mês")
plt.ylabel("Quantidade de Menções")
plt.title("Menções da Hashtag #EconomizeEnergia ao Longo do Mês")
plt.grid(True)
plt.show()
```

Código 50 — Código final

Figura 10 — Gráfico de Dispersão: Campanha de Consumo Consciente



Fonte: Elaboração própria (2025).

Perguntas para Reflexão: Análise do Gráfico

a) Em quantos dias a *hashtag* foi mencionada mais de 1200 vezes?

b) Qual foi o dia com maior número de menções? E o dia com menor número de menções?

c) Identifique períodos em que as menções à *hashtag* aumentaram e períodos em que diminuíram. O que pode ter causado essas variações?

d) Entre os dias 6 e 7 e os dias 25 e 26, qual intervalo apresentou a maior variação no número de menções?

e) Com base nos dados analisados, quais estratégias poderiam ser adotadas para aumentar o impacto da campanha e incentivar mais pessoas a praticarem o consumo consciente de energia?

Produção

Durante a etapa de produção, os estudantes recebem um novo problema e constroem o gráfico de forma autônoma, recomenda-se disponibilizar o guia da biblioteca como apoio. Pode-se apresentar a função `help()`, que permite visualizar os argumentos disponíveis, os parâmetros opcionais e explicações sobre o funcionamento das funções. Dessa forma, os estudantes podem explorar diferentes possibilidades de personalização do gráfico, como a escolha de cores, definição de intervalos, uso de marcadores, entre outros ajustes. Essa abordagem oferece oportunidades adicionais de exploração e aplicação dos conhecimentos, incentivando a autonomia e o pensamento crítico na análise e construção de gráficos.

Socialização

Na etapa final da atividade, os estudantes devem apresentar suas produções aos colegas, expondo o gráfico construído e respondendo às perguntas de reflexão com base na análise dos dados. Além disso, devem compartilhar suas dificuldades e descobertas ao longo do processo. Essa etapa é fundamental, pois permite que os estudantes sintetizem o que foi aprendido, desenvolvendo não apenas a compreensão dos conceitos matemáticos, mas também a capacidade de reflexão e pensamento crítico sobre o mundo ao seu redor.

4.2 Atividade 2: Padrões Numéricos e Algoritmos de Repetição

Tempo estimado: 100 min
Objetivos da atividade:

- Reconhecer padrões matemáticos em diferentes tipos de sequências numéricas e construir algoritmos que gerem seus termos.
- Introduzir a estrutura de repetição `for` como recurso fundamental para automatizar processos e iterar instruções em um programa.
- Explorar representações gráficas das sequências geradas, promovendo a análise e interpretação de seus comportamentos de crescimento.

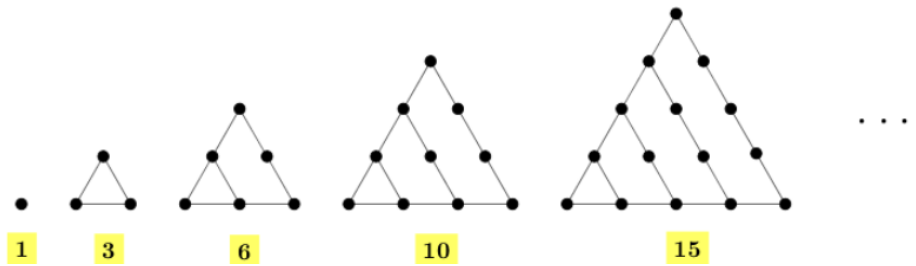
Habilidades da BNCC

- (EM13MAT507) Identificar e associar progressões aritméticas (PA) a funções afins de domínios discretos, para análise de propriedades, dedução de algumas fórmulas e resolução de problemas.
- (EM13MAT508) Identificar e associar progressões geométricas (PG) a funções exponenciais de domínios discretos, para análise de propriedades, dedução de algumas fórmulas e resolução de problemas.
- (EM13MAT405) Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

Contextualização: Números Triangulares

Chamamos de números triangulares aqueles que podem ser expressos como a soma de uma sequência de números naturais consecutivos iniciando pelo número 1. Esses números podem ser representados por arranjos de pontos dispostos de forma a formar triângulos equiláteros. Na Figura 11 apresenta-se a representação geométrica dos cinco primeiros números triangulares.

Figura 11 — Números Triangulares



Fonte: Clubes da OBMEP (2020)

Com base nessa representação, obtemos a seguinte tabela:

Tabela 9 — Tabela de Números Triangulares

Número Triangular	Expressão	Resultado
$1^{\underline{o}}$	1	1
$2^{\underline{o}}$	$1+2$	3
$3^{\underline{o}}$	$1+2+3$	6
$4^{\underline{o}}$	$1+2+3+4$	10
$5^{\underline{o}}$	$1+2+3+4+5$	15
$6^{\underline{o}}$	$1+2+3+\cdots+6$	21
$7^{\underline{o}}$	$1+2+3+\cdots+7$	28
$8^{\underline{o}}$	$1+2+3+\cdots+8$	36
$9^{\underline{o}}$	$1+2+3+\cdots+9$	45
$10^{\underline{o}}$	$1+2+3+\cdots+10$	55

Fonte: Elaboração própria (2025).

Considere uma sequência com n números triangulares, e seja r_n a quantidade de pontos no n-ésimo triângulo (isto é, o resultado da soma). Observa-se que essa sequência pode ser descrita por uma relação de recorrência:

$$r_n = r_{n-1} + n$$

De fato, podemos verificar essa relação nos primeiros termos:

$$\begin{aligned} r_2 &= r_1 + 2 = 1 + 2 = 3 \\ r_3 &= r_2 + 3 = 3 + 3 = 6 \\ r_4 &= r_3 + 4 = 6 + 4 = 10 \\ &\vdots \end{aligned}$$

Com base nessa relação, propõe-se a seguinte atividade: Elabore um programa em Python que determine o $20^{\underline{o}}$ número triangular.

```
1 resultado=0
2
3 for n in range(1,21):
4     resultado = resultado + n
5
6 print("O 20º número triangular é:",resultado)
```

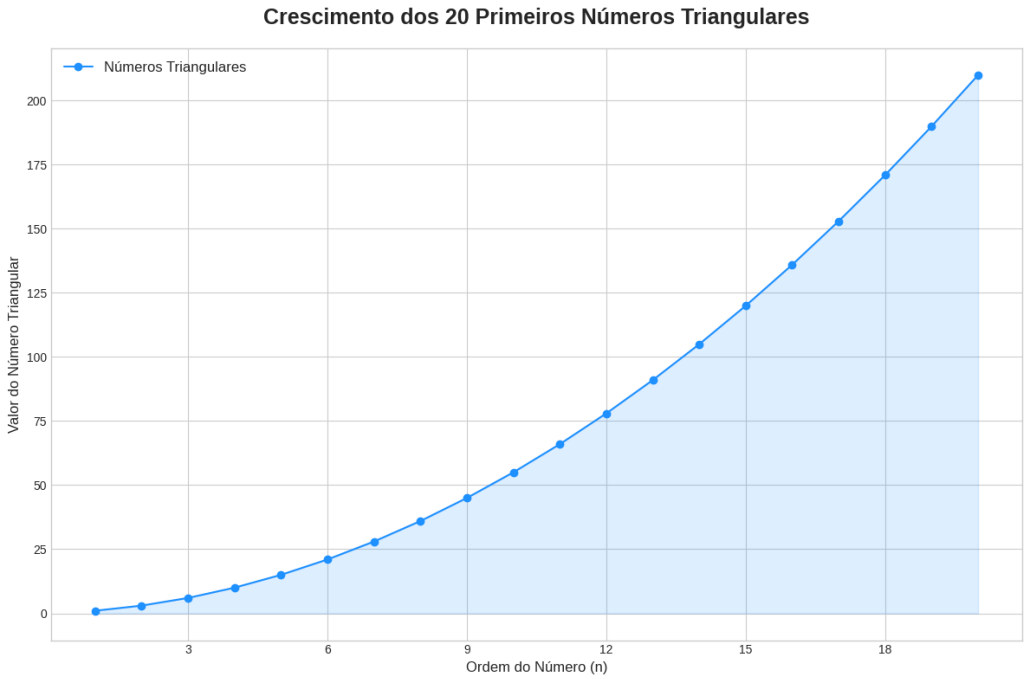
Código 51 — Números Triangulares

Saída esperada:

```
O 20º número triangular é: 210
```

O exemplo dos números triangulares apresenta uma situação em que o crescimento da sequência não é proporcional. Por isso, sua representação não corresponde a uma função polinomial do $1^{\underline{o}}$ grau. Como mostra o gráfico da Figura 12.

Figura 12 — Gráfico Números Triangulares



Fonte: Elaboração própria (2025).

O gráfico acima foi gerado a partir do código abaixo:

```
import matplotlib.pyplot as plt

triangular=0
y=[]  # Cria uma lista vazia que será usada para armazenar
    # os resultados gerados no laço for

for n in range(1,21):
    triangular = triangular + n
    y.append(triangular)    #Armazena o número triangular na lista y a cada repetição

x = list(range(1,21))          #Define os elementos do eixo x

# Plotando o gráfico
plt.plot(x, y,marker='o', linestyle='-', color='dodgerblue', label='Números Triangulares')
# Adiciona rótulos aos eixos x e y
plt.xlabel('Ordem do Número (n)')
plt.ylabel('Valor do Número Triangular')
# Adiciona um título claro e informativo
plt.title('Crescimento dos 20 Primeiros Números Triangulares')
plt.grid(True)
plt.show()
```

Código 52 — Gráfico Números Triangulares

Produção

Nesta etapa, os estudantes são convidados a aplicar os conhecimentos adquiridos em programação para realizar operações aritméticas e gerar sequências numéricas. Para isso, utilizam operadores básicos $(+, -, *)$ explorando a construção de listas em Python como forma de estruturar e armazenar os dados gerados.

As sequências numéricas selecionadas abrangem diferentes tipos, como aritméticas, geométricas e de recorrência. A proposta consiste em identificar o padrão de formação de cada sequência e implementar um código em Python capaz de gerar os 50 primeiros termos. As sequências a serem exploradas são:

1. $(3, 8, 13, 18, 23, 28, 33, 38, ...)$
2. $(1, 4, 7, 10, 13, 16, 19, 22, 25, ...)$
3. $(1, 7, 14, 28, 35, 42, 49, ...)$
4. $(2, 6, 18, 54, 162, 486, ...)$
5. Sequência de Fibonacci: $(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...)$
6. Quadrados perfeitos: $(0, 1, 4, 9, 16, 25, 36, 49, 64, 81, ...)$
7. Cubos perfeitos: $(1, 8, 27, 64, 125, 216, 343, 512, 729, ...)$

Concluída a implementação dos códigos e a geração das listas, os estudantes são orientados a representar graficamente os dados, utilizando a biblioteca `matplotlib.pyplot`. A visualização gráfica permite investigar o comportamento de cada sequência e refletir sobre as características do crescimento envolvido. A partir dessa análise, os estudantes devem responder às seguintes questões:

a) O que o gráfico revela sobre o crescimento da sequência? O crescimento ocorre de forma constante, acelerada ou irregular? É possível identificar algum padrão?

b) O que o gráfico revela sobre o crescimento da sequência? O crescimento ocorre de forma constante, acelerada ou irregular? É possível identificar algum padrão?

c) Quais das sequências apresentadas podem ser modeladas por funções afins? Quais se aproximam de funções exponenciais? Justifique suas respostas com base na análise gráfica.

d) Ao comparar duas ou mais sequências em um mesmo gráfico, qual delas apresenta crescimento mais rápido? A partir de qual ponto uma sequência ultrapassa a outra?

Socialização

Na etapa de socialização, os estudantes compartilham suas produções com a turma, apresentando os códigos desenvolvidos, os gráficos gerados e suas respostas às questões de reflexão. Esse momento visa promover a troca de experiências, o reconhecimento coletivo das dificuldades enfrentadas e das estratégias utilizadas, bem como o aprofundamento na análise dos dados a partir da mediação entre pares.

4.3 Atividade 3: Função Afim

Tempo estimado: 100 min
Objetivos da atividade:

- Interpretar e resolver problemas envolvendo função polinomial do 1º grau.
- Converter representações algébricas de funções polinomiais de 1º grau em representações geométricas no plano cartesiano.

Habilidades da BNCC

- (EM13MAT401) Converter representações algébricas de funções polinomiais de 1º grau em representações geométricas no plano cartesiano, distinguindo os casos nos quais o comportamento é proporcional, recorrendo ou não a softwares ou aplicativos de álgebra e geometria dinâmica.
- (EM13MAT405) Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

Além da capacidade de plotar gráficos, a linguagem Python destaca-se por possibilitar a realização de cálculos mais complexos, superando, nesse aspecto, as funcionalidades de uma calculadora convencional. Nesse sentido, ao propor a resolução e elaboração de problemas, torna-se essencial que os estudantes compreendam e utilizem adequadamente a notação de função.

Assim como a Atividade 1, inicialmente, propõe-se a construção conjunta de um código para o cálculo de determinada função, envolvendo a interação entre professor e estudantes a partir de um problema contextualizado. Durante essa etapa, o professor apresenta a problemática e relembra conceitos estudados envolvendo função polinomial do 1º grau.

Contextualização: Plástico demais no planeta!

O problema relaciona dados envolvendo a ODS 12 - Consumo e Produção Responsáveis. Uma fábrica de garrafas PET recicladas transforma plástico coletado em garrafas reutilizáveis. A quantidade de garrafas produzidas depende da quantidade de plástico reciclado. Sabe-se que a cada 1 kg de plástico reciclado, produzem-se 25 garrafas, mas há uma perda fixa de material equivalente a 10 garrafas durante o processo. Logo, podemos modelar a função da seguinte forma: seja $f : \mathbb{R} \to \mathbb{R}$

$$f(x) = 25 \cdot x - 10$$

em que x representa a massa de plástico reciclado em quilogramas e $f(x)$ a quantidade de garrafas produzidas.

1º Passo: A função def em Python

A estrutura `def` em Python define um bloco de código que executa uma tarefa específica e pode ser reutilizado diversas vezes ao longo da resolução de um problema. Essa funcionalidade é especialmente útil, pois evita a repetição da expressão analítica de uma função matemática sempre que for necessário utilizá-la. Dessa forma, dado um valor de x, é possível obter seu correspondente $f(x)$, isto é, calcular a imagem da função. Da

mesma maneira, também é viável realizar o processo inverso: conhecendo o valor de y, tal que $y = f(x)$, pode-se determinar o valor de x.

```
def f(x):
  return 25*x-10
```

Código 53 — a função def

2º Passo: Cálculo de imagens de função

Suponha que foram utilizados 4kg de plástico reciclado, pergunta-se: quantas garrafas foram produzidas ao final do processo?

Temos $x = 4$ e $f(x) =?$

```
def f(x):
  return 25*x-10
x = 4
f(x)
```

Código 54 — f(x)

Saída esperada:

90

3º Passo: Cálculo de pré-imagens

A pré-imagem de um valor y em uma função $f(x)$ é o conjunto de todos os valores x para os quais $f(x) = y$, isto é, significa determinar os valores de entrada (x) que produzem um valor conhecido de saída (y).

Suponha que foram produzidas 290 garrafas durante um processo, pergunta-se: quantos quilogramas de plástico reciclado foram utilizados?

$$\begin{aligned} f(x) &= y \\ 25 \cdot x - 10 &= y \\ 25 \cdot x &= y + 10 \\ x &= \frac{y + 10}{25} \end{aligned}$$

```
y = 290
x = (y+10)/25

x
```

Código 55 — f(x)

Saída esperada:

12

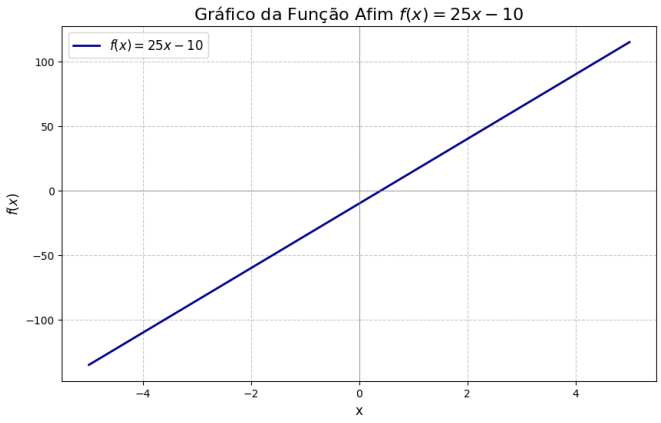
4º Passo: Gráfico da função

```python
import matplotlib.pyplot as plt

def f(x):
  return 25*x-10

x = list(range(0,10))
y=[]
for n in x:
  y.append(f(n))
# Plota a função
plt.plot(x, y, label='$f(x) = 25x - 10$', color='darkblue')
# Adiciona rótulos aos eixos
plt.xlabel('x', fontsize=12)
plt.ylabel('$f(x)$', fontsize=12)
# Adiciona um título ao gráfico
plt.title('Gráfico da Função Afim $f(x) = 25x - 10$', fontsize=16, fontweight='bold')
# Adiciona uma grade para facilitar a leitura dos valores
plt.grid(True, linestyle='--')
# Adiciona linhas de referência para os eixos x e y no zero
plt.axhline(0, color='gray', linewidth=1)
plt.axvline(0, color='gray', linewidth=1)
# Adiciona a legenda para identificar a linha plotada
plt.legend(fontsize=12)
# Exibe o gráfico
plt.show()
```

Código 56 — Gráfico função afim

O gráfico da Figura 13 foi gerado a partir do código acima:

Figura 13 — Gráfico função afim



Fonte: Elaboração própria (2025).

Perguntas para Reflexão:

a) Quantas garrafas são produzidas com 9,5 kg de plástico reciclado?

b) Represente graficamente a função para valores de plástico reciclado entre 0 e 10 kg.

c) O que significa obter um valor negativo de garrafas produzidas? Por exemplo, quantas garrafas serão produzidas com 0,2kg de plástico reciclado?

d) Se a meta da fábrica for produzir ao menos 100 garrafas, qual deve ser o mínimo de plástico reciclado?

Produção

Nesta etapa, os estudantes devem, com base nas expressões algébricas fornecidas, definir um domínio adequado e construir os respectivos gráficos por meio de código em linguagem Python. A representação gráfica gerada serve de base para a análise do comportamento das funções, com foco na identificação de suas principais características.

Seja $f : \mathbb{R} \to \mathbb{R}$. Considere as seguintes funções:

a) $f(x) = 2x - 4$

b) $f(x) = 2x$

c) $f(x) = -x + 1$

d) $f(x) = -4x + 2$

e) $f(x) = 3$

Após a construção dos gráficos, os estudantes deverão responder às seguintes questões:

1. Classifique cada função quanto ao seu comportamento: é crescente, decrescente ou constante?

2. Em qual(is) das funções apresentadas o comportamento pode ser considerado proporcional? Justifique

3. Determine o(s) zero(s) da função f, isto é, (s) valor(es) de x tal que $f(x) = 0$.

4. Calcule o valor de $f(x)$ quando $x = 0$ e interprete esse valor no contexto do gráfico

5. Escolha duas das funções listadas, represente-as no mesmo plano cartesiano e analise:

   - Os gráficos se interceptam? Em caso afirmativo, identifique o(s) ponto(s) de interseção.
   - Qual dos gráficos apresenta maior inclinação? O que essa inclinação revela sobre o coeficiente angular da função?

Socialização

Na etapa de socialização, os estudantes devem apresentar à turma o gráfico construído com duas funções de sua escolha, juntamente com os códigos utilizados, os gráficos gerados e as respostas obtidas na etapa de análise. Este momento tem como objetivo favorecer a troca de experiências, o reconhecimento coletivo de dificuldades e estratégias adotadas, além de promover o aprofundamento conceitual por meio da mediação entre pares.

4.4 Atividade 4: Função a partir de uma tabela

Tempo estimado: 100 min
Objetivos da atividade:

- Identificar a expressão algébrica que representa uma função a partir de uma tabela de valores.
- Converter representações algébricas de funções polinomiais de 1º grau em representações geométricas no plano cartesiano.

Habilidades da BNCC

- (EM13MAT501) Investigar relações entre números expressos em tabelas para representá-los no plano cartesiano, identificando padrões e criando conjecturas para generalizar e expressar algebricamente essa generalização, reconhecendo quando essa representação é de função polinomial de 1º grau.
- (EM13MAT401) Converter representações algébricas de funções polinomiais de 1º grau em representações geométricas no plano cartesiano, distinguindo os casos nos quais o comportamento é proporcional, recorrendo ou não a softwares ou aplicativos de álgebra e geometria dinâmica.
- (EM13MAT405) Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

Contextualização: Questão ENEM 2024

A atividade tem como ponto de partida uma questão retirada da prova de Matemática da reaplicação do Exame Nacional do Ensino Médio (ENEM), edição de 2024 (questão 141, Caderno 6 — Cinza). Segundo o enunciado:

É comum pensarmos na equivalência entre a idade de um animal de estimação, no caso de cães e gatos, e de um ser humano. De acordo com as diretrizes de idade criadas pela American Animal Hospital Association (AAHA), o International Cat Care e a American Association of Feline Practitioners (AAFP), a última fase da vida de um gato é chamada de geriátrica e começa aos 15 anos de vida do animal. A Tabela 14 apresenta os primeiros anos da fase geriátrica da equivalência entre a idade do gato e a idade de um humano.

Figura 14 — Questão 141 da reaplicação do ENEM 2024 (Caderno 6 – Cinza)

Idade do gato (ano)	Idade equivalente de um humano (ano)
15	76
16	80
17	84
18	88
19	92
20	96
21	100
22	104
23	108
24	112
25	116

Sabe-se que o gato mais velho do mundo morreu ao completar 38 anos de vida. Considere que o padrão observado na tabela se mantém.

Disponível em: https://canaldopet.ig.com.br. Acesso em: 28 nov. 2021 (adaptado).

Fonte: INEP (2024)

De acordo com os dados apresentados, a idade em que o gato mais velho do mundo morreu é equivalente a qual idade, em anos, de um humano?

Uma forma simples de obter a resposta consiste em plotar um gráfico a partir da tabela. Observa-se que, a cada ano que passa, a idade do gato cresce 1 ano, enquanto a idade equivalente em anos humanos aumenta 4 anos. Considerando que a tabela se inicia com o gato aos 15 anos, e deseja-se calcular a idade equivalente aos 38 anos, é necessário iterar esse processo 23 vezes, correspondente à diferença entre 38 e 15.

Para isso, é preciso utilizar listas para armazenar os valores das variáveis e, posteriormente, representar graficamente os dados com a biblioteca `Matplotlib`. A função `range` pode ser utilizada para gerar a sequência de idades do gato, enquanto o método `append` permite adicionar elementos ao final das listas. A estrutura de repetição `for` será empregada para repetir o processo o número de vezes definido pela variável `interação`.

```
gato = list(range(15,39))
valor_inicial = 76
interação = 24
humano = []
for n in range(interação):
    h = valor_inicial + 4*n
    humano.append(h)
humano
```

Código 57 — Idade gato equivalente idade humano

Saída esperada:

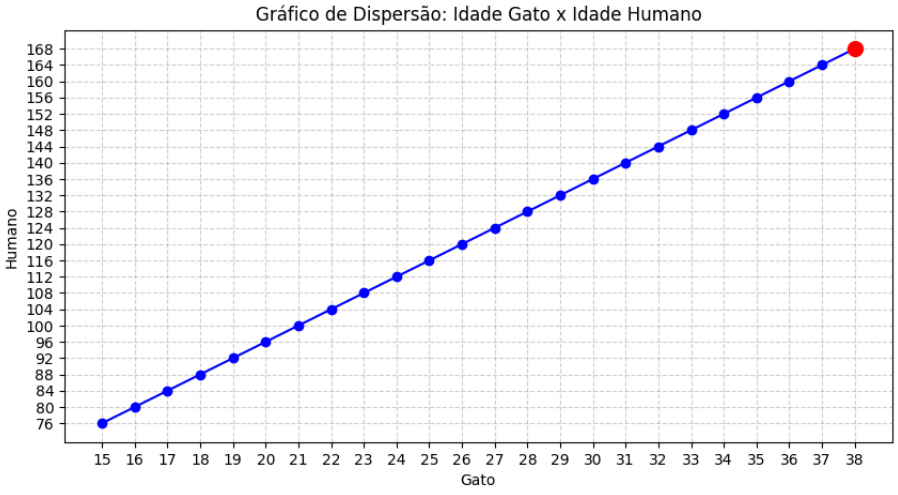
```
[76, 80, 84, 88, 92, 96, 100, 104, 108, 112, 116, 120, 124,
128, 132, 136, 140, 144, 148, 152, 156, 160, 164, 168]
```

Assim, ao plotarmos o gráfico utilizando a biblioteca `Matplotlib`, podemos observar na Figura 15 a relação visual entre a idade do gato e a idade equivalente em humanos.

```
import matplotlib.pyplot as plt

plt.plot(gato, humano,marker='o', linestyle='-', color='b')
plt.xlabel("Gato")
plt.xticks(gato)  # Isso garante que todos os valores de x apareçam no eixo
plt.yticks(humano)
plt.ylabel("Humano")
plt.grid(True, linestyle="--", alpha=0.6)
plt.title("Gráfico de Dispersão: Idade Gato x Idade Humano")
plt.plot(38, 168, 'ro')
plt.show()
```

Código 58 — Gráfico Enem 2024

Figura 15 — Gráfico gerado pelo código



Fonte: Elaboração própria (2025)

Portanto, a idade em que o gato mais velho do mundo morreu é equivalente a 168 anos em idade de humano.

Note que a idade do humano cresce de maneira proporcional em relação à idade do gato, logo podemos expressar algebricamente a relação entre as duas variáveis, em que a idade do humano depende da idade do gato. Assim, seja $f : \mathbb{R} \to \mathbb{R}$ tal que $f(x) = ax + b$, em que a e b são constantes reais. Considere x a idade do gato e $f(x)$ a idade equivalente de um humano. Devemos determinar os coeficientes a e b para determinar a expressão algébrica.

Sabemos que a taxa de variação a é dada por

$$a = \frac{y_2 - y_1}{x_2 - x_1}$$

em que $y = f(x), \forall x \in \mathbb{R}$. Pela tabela temos que, $x_1 = 15$ e $y_1 = 76$, enquanto que $x_2 = 16$ e $y_2 = 80$. Vamos encontrar a taxa de variação utilizando Python: Pelo código anterior, `gato` e `humano` são listas, logo podemos percorrer essas listas operando com seus índices. Por exemplo, `gato[0]` representa o primeiro valor da lista (15 anos), enquanto `humano[1]` representa o segundo valor da lista (80 anos).

```
1 a = (humano[1]-humano[0])/(gato[1]-gato[0])
2 a
```

Código 59 — Taxa de variação

Saída esperada:

```
4.0
```

Para obter b, basta substituir a em $y = ax + b \Rightarrow b = y - ax$

```
1 b = H[1]-a*G[1]
2 b
```

Código 60 — Termo independente

Saída esperada:

```
16.0
```

Assim,

```
1 a=(humano[1]-humano[0])/(gato[1]-gato[0])
2 b = H[1]-a*G[1]
3 print("Logo y=ax+b e y= %dx + %d"(a,b))
```

Código 61 — Expressão algébrica

Saída esperada:

```
Logo y=ax+b e y= 4x + 16
```

Produção

Nesta etapa, os estudantes, com base nas situações apresentadas a seguir e com o auxílio do Python como calculadora, deverão determinar a expressão algébrica que relaciona as duas variáveis envolvidas. Para isso, deverão observar os dados fornecidos,

identificar a taxa de variação (coeficiente angular) e o valor inicial (coeficiente linear), e então escrever a função afim correspondente.

Em seguida, os estudantes poderão utilizar o Python para testar a função construída, substituindo valores para verificar se a expressão representa corretamente a relação entre as variáveis. Por fim, deverão construir o gráfico da função com o auxílio da biblioteca matplotlib, analisando seu comportamento visual e a coerência com os dados apresentados inicialmente.

1. Um trabalhador recebe um salário fixo mensal mais um valor proporcional às horas extras trabalhadas. A Tabela 10 mostra o valor que ele recebe dependendo do número de horas extras no mês:

Tabela 10 — Relação entre horas extras e salário

Horas extras	**Salário (R$)**
0	1500
2	1620
4	1740
6	1860

Fonte: Elaboração própria (2025).

Qual é a expressão da função que relaciona o salário $S(h)$ com o número de horas extras h?

2. Uma empresa cobra uma taxa fixa de entrega e mais um valor por item transportado. A Tabela 11 mostra o custo total $C(n)$, em reais, para diferentes quantidades de itens entregues:

Tabela 11 — Relação entre quantidade de itens entregues e custo

Itens entregues	**Custo (R$)**
1	18
3	26
5	34
7	42

Fonte: Elaboração própria (2025).

Qual é a expressão da função que relaciona o custo total $C(n)$ com o número de itens entregues n?

Socialização

Na etapa de socialização, os estudantes devem elaborar um problema simples do cotidiano que envolva duas variáveis relacionadas por uma função afim. Para isso, deverão montar uma tabela com pelo menos cinco pares de valores que representem essa relação e determinar a expressão algébrica que relaciona as variáveis. Em seguida, os estudantes utilizarão o Python para construir o gráfico da função criada, analisando e explicando brevemente o significado desse gráfico no contexto do problema proposto.

4.5 Atividade 5: Função Quadrática

Tempo estimado: 100 min
Objetivos da atividade:

- Compreender comportamento de funções quadráticas em contextos reais, com ênfase na identificação e interpretação do ponto de máximo.
- Promover a articulação entre conceitos matemáticos, sustentabilidade alimentar e planejamento urbano.
- Desenvolver a autonomia dos estudantes na análise de dados e no fortalecimento do pensamento computacional.

Habilidades da BNCC

- (EM13MAT402) Converter representações algébricas de funções polinomiais de 2º grau em representações geométricas no plano cartesiano, distinguindo os casos nos quais uma variável for diretamente proporcional ao quadrado da outra, recorrendo ou não a softwares ou aplicativos de álgebra e geometria dinâmica, entre outros materiais.
- (EM13MAT503) Investigar pontos de máximo ou de mínimo de funções quadráticas em contextos envolvendo superfícies, Matemática Financeira ou Cinemática, entre outros, com apoio de tecnologias digitais.

Nesta atividade, os estudantes modelarão a produtividade de uma horta por meio de uma função quadrática e utilizarão Python para representar graficamente esse comportamento, relacionando o Objetivo de Desenvolvimento Sustentável (ODS) 2: Fome Zero e Agricultura Sustentável . A abordagem favorece a leitura de gráficos, a análise de funções com ponto de máximo e a tomada de decisão fundamentada em dados simulados.

Contextualização: Quando colher para obter o máximo da produção?

Com o aumento das iniciativas de agricultura urbana em cidades brasileiras, muitas comunidades têm buscado métodos sustentáveis de produção de alimentos em pequenos espaços. Acompanhar a produtividade de uma plantação ao longo do tempo é essencial para planejar o melhor momento de colheita e o replantio. Uma horta comunitária foi implantada em um bairro urbano. O coletivo local registrou a produtividade (em kg de hortaliças por metro quadrado) durante várias semanas de cultivo. Observou-se que, após certo tempo, a produtividade começou a cair, possivelmente devido à exaustão dos nutrientes do solo.

A equipe deseja usar a matemática para descobrir o melhor momento de colheita e planejar as próximas safras de forma mais eficiente.

Dessa forma, a turma será convidada a analisar a produtividade de hortaliças em uma horta comunitária ao longo de 9 semanas. Foi observada uma tendência de crescimento inicial na produção, atingindo um pico, e depois um declínio, conforme o solo vai perdendo nutrientes.

A função que representa esse comportamento é:

$$f(t) = -0,5t^2 + 4t + 2$$

em que $f(t)$ representa a produtividade (em kg por metro quadrado) e t é o tempo (em semanas).

Produção

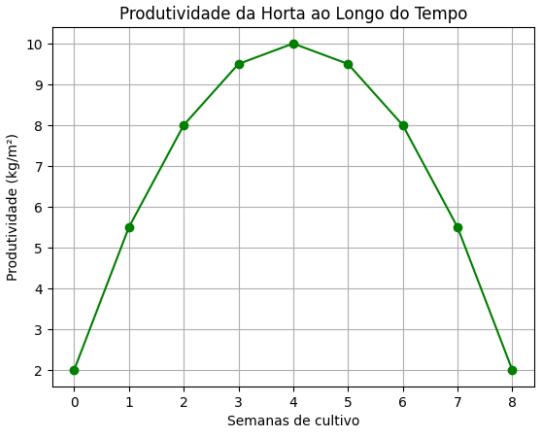
Os estudantes utilizarão Python para construir o gráfico e responder às questões interpretativas a partir dele.

```
import matplotlib.pyplot as plt
t = list(range(0,9))
def f(t):
  return -0.5*t**2 + 4*t + 2

resultado = []
for i in t:
  resultado.append(f(i))
  print("Quando i=",i,"tem-se f(t) =",f(i))

plt.plot(t, resultado, marker='o', color='green')
plt.title("Produtividade da Horta ao Longo do Tempo")
plt.xlabel("Semanas de cultivo")
plt.ylabel("Produtividade (kg/m²)")
plt.grid(True)
plt.show()
```

Código 62 — Função Quadrática

O gráfico da Figura 16 abaixo é gerado a partir do código acima:

Figura 16 — Produtividade horta



Fonte: Elaboração própria (2025).

Para analisar o ponto de máximo da parábola, consideramos que, dada a função $f : \mathbb{R} \to \mathbb{R}$ tal que $f(x) = ax^2 + bx + c$, as coordenadas do vértices x_v e y_v podem ser obtidas por meio das expressões:

$$x_v = \frac{-b}{2a} \text{ e } y_v = \frac{-\Delta}{4a}$$

Ou simplesmente, como $y = f(x)$, segue $y_v = f(x_v)$. Daí, temos

```
# Cálculo do vértice (máximo)
x_v = -4 / (2 * -0.5)  # fórmula do x do vértice: -b/2a
y_v = f(x_v)
print("As coordenadas do vértice são:",x_v,y_v)
```

Código 63 — Ponto de máximo

Saída Esperada:

```
As coordenadas do vertice sao: 4.0 10.0
```

Perguntas para Reflexão: Análise do Gráfico

1. Em que semana ocorre o ponto de máxima produtividade?

2. Qual a produtividade nesse ponto?

3. O que acontece com a produção após essa semana?

4. Se os responsáveis pela horta decidissem colher na 8ª semana, a colheita seria eficiente?

5. Como a matemática pode ajudar a tomar decisões sustentáveis na agricultura urbana?

6. Suponha que a equipe da horta decidiu investir em práticas sustentáveis, como a adubação natural e a rotação de culturas. Com isso, espera-se que a queda na produtividade ao longo do tempo seja mais lenta. Um novo modelo é proposto para simular esse cenário: $f(t) = -0,3t^2 + 3,8t + 2$ Você diria que esse modelo é mais sustentável? Justifique sua resposta com base na análise matemática e nas práticas agrícolas mencionadas.

7. Note que o gráfico possui um formato segmentado, com traços retos entre os pontos. Isso acontece porque ele foi gerado a partir de uma lista com 8 elementos, representando as semanas. Agora imagine que a variável t deixe de ser uma lista e passe a ser um array com valores mais densos, como no comando np.linspace(0, 8, 100). O que muda na aparência do gráfico? Por que essa mudança acontece? Que vantagens isso pode trazer para a interpretação dos dados?

Socialização

Nesta etapa, os estudantes compartilham com a turma as análises e conclusões obtidas a partir da modelagem da produtividade da horta, apresentando o gráfico gerado, a identificação do ponto de máximo e a interpretação das variações ao longo das semanas. As apresentações devem incluir as respostas às perguntas de reflexão, destacando como os dados e representações gráficas embasaram as decisões propostas para o manejo sustentável da produção. O momento é conduzido de modo a incentivar a comparação entre os resultados dos diferentes grupos, especialmente na avaliação do modelo alternativo com menor taxa de decréscimo de produtividade, favorecendo o debate sobre estratégias agrícolas mais sustentáveis. Espera-se que essa troca de interpretações possibilite a consolidação dos conceitos sobre funções quadráticas e amplie a compreensão sobre o uso da matemática como ferramenta para a análise e tomada de decisões em contextos reais.

4.6 Atividade 6: Função Exponencial

Tempo estimado: 100 min
Objetivos da atividade:

- Compreender o comportamento de uma função exponencial em contextos ambientais e analisar criticamente o impacto do crescimento descontrolado de resíduos plásticos

- Contruir e interpretar gráficos, desenvolvendo a consciência socioambiental por meio da matemática aplicada.

Habilidades da BNCC

- (EM13MAT304) Resolver e elaborar problemas com funções exponenciais nos quais seja necessário compreender e interpretar a variação das grandezas envolvidas, em contextos como o da Matemática Financeira, entre outros.

- (EM13MAT403) Analisar e estabelecer relações, com ou sem apoio de tecnologias digitais, entre as representações de funções exponencial e logarítmica expressas em tabelas e em plano cartesiano, para identificar as características fundamentais (domínio,imagem, crescimento) de cada função.

- (EM13MAT405) Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

Nesta atividade, os estudantes irão modelar, por meio de uma função exponencial, a concentração de microplásticos em uma região oceânica ao longo de 15 anos, relacionando os Objetivos de Desenvolvimento Sustentável 12 e 14, que tratam do Consumo e Produção Responsáveis e Vida na Água, respectivamente. Serão simulados dois cenários: Sem políticas de contenção (crescimento acelerado); Com políticas de contenção (redução da taxa de crescimento).

Contextualização: A multiplicação invisível de microplásticos nos oceanos

Microplásticos são fragmentos de plástico com menos de 5 mm que se acumulam nos oceanos devido à degradação de resíduos maiores e ao descarte inadequado de produtos industriais e cosméticos. Estudos indicam que a concentração de microplásticos tem aumentado exponencialmente nas últimas décadas, representando uma ameaça invisível à vida marinha e à saúde humana.

A quantidade de partículas plásticas microscópicas nos oceanos vem crescendo de maneira alarmante, impulsionada pelo consumo excessivo, pelo descarte incorreto de resíduos e pela fragmentação de materiais plásticos maiores.

A simulação é realizada utilizando a biblioteca `matplotlib` para construir um gráfico comparativo entre dois cenários, considerando que a concentração inicial de microplásticos na água é de $0,2$ mg/L:

- Cenário pessimista: nenhuma política ambiental é adotada, e a concentração de microplásticos cresce a uma taxa de 18% ao ano;

- Cenário otimista: políticas ambientais eficazes são implementadas, reduzindo a taxa de crescimento anual para 8%.

Os estudantes devem utilizar esse modelo para gerar e analisar os gráficos, interpretar os coeficientes da função exponencial, identificar o ano em que o limite ambiental seguro é ultrapassado e, por fim, discutir a importância de decisões sustentáveis fundamentadas em evidências matemáticas.

Modelagem Matemática do Crescimento

Seja a a concentração inicial de microplásticos na água, i a taxa de crescimento anual, e C_t a concentração total após t anos. Podemos modelar esse crescimento de forma sucessiva:

No primeiro ano, a concentração inicial cresce a uma taxa i, ou seja:

$$C_1 = a + a \cdot i = a(1 + i)$$

No segundo ano, a nova concentração C_1 também cresce a uma taxa i, resultando em:

$$C_2 = C_1 + C_1 \cdot i = C_1(1 + i) = a(1 + i)^2$$

Por indução, no ano t, a concentração de microplásticos será dada por:

$$C_t = a(1 + i)^t$$

Aplicando essa fórmula aos dois cenários descritos:

- **Cenário pessimista** ($i = 0,18$):

$$C_t = 0,2 \cdot (1 + 0,18)^t = 0,2 \cdot 1,18^t$$

- **Cenário otimista** ($i = 0{,}08$):

$$C_t = 0,2 \cdot (1 + 0,08)^t = 0,2 \cdot 1,08^t$$

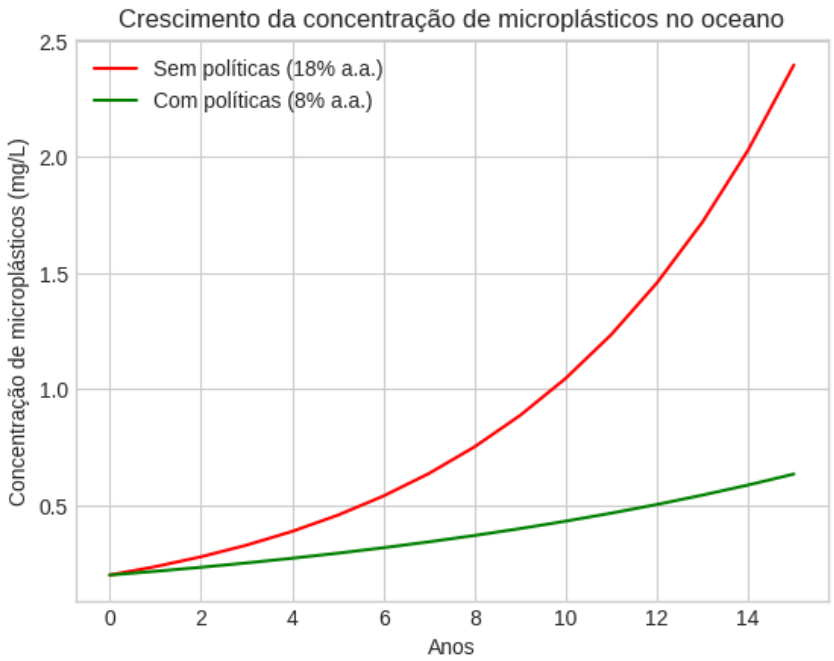
Produção

Com base na modelagem matemática apresentada, a implementação em Python permitirá a simulação e visualização gráfica do crescimento da concentração de microplásticos nos dois cenários. O código a seguir ilustra essa construção:

```python
import matplotlib.pyplot as plt

# Lista de anos (de 0 a 15)
anos = list(range(0, 16))

# Cenário 1: sem políticas de contenção (crescimento de 18% ao ano)
micro1 = [0.2 * (1.18)**t for t in anos]  # 0.2 mg/L no ano inicial

# Cenário 2: com políticas de contenção (crescimento de 8% ao ano)
micro2 = [0.2 * (1.08)**t for t in anos]

# Gráfico
plt.plot(anos, micro1, label='Sem políticas (18% a.a.)', color='red')
plt.plot(anos, micro2, label='Com políticas (8% a.a.)', color='green')
plt.xlabel("Anos")
plt.ylabel("Concentração de microplásticos (mg/L)")
plt.title("Crescimento da concentração de microplásticos no oceano")
plt.grid(True)
plt.legend()
plt.show()
```

Código 64 — Função Exponencial

Figura 17 — Crescimento da concentração de microplásticos no oceano



Fonte: Elaboração própria (2025).

O gráfico gerado representa o comportamento da função exponencial em ambos os contextos. É possível observar na Figura 17 como pequenas variações na taxa de crescimento produzem grandes diferenças na concentração ao longo do tempo. Essa análise pode servir como ponto de partida para reflexões críticas sobre políticas ambientais, planejamento sustentável e tomada de decisões fundamentadas em dados.

Perguntas para Reflexão

1. Qual é a concentração prevista de microplásticos após 15 anos em cada cenário?
2. Em que ano a concentração sem políticas dobra o valor inicial? E com políticas?
3. Como o gráfico expressa a diferença entre as taxas de crescimento?
4. Suponha que o limite seguro seja 0,6 mg/L. Em qual ano esse limite é ultrapassado em cada cenário?
5. O que esse modelo matemático nos ajuda a compreender sobre o papel das políticas públicas e da conscientização ambiental?
6. Proponha um novo cenário com uma taxa ainda menor (ex: 3% ao ano) e discuta possíveis ações que poderiam justificar essa redução.

Socialização

Nesta etapa, os estudantes apresentam os resultados obtidos a partir da modelagem matemática e da implementação em Python, compartilhando com a turma as análises desenvolvidas para cada cenário. As apresentações devem contemplar a interpretação dos gráficos produzidos, as respostas às perguntas de reflexão e as implicações ambientais identificadas. O momento é conduzido de modo a promover o diálogo entre os grupos, incentivando a comparação das diferentes estratégias propostas e a argumentação fundamentada em cálculos com equações exponenciais e representações gráficas. Espera-se que, por meio dessa troca, os estudantes consolidem a compreensão do comportamento das funções exponenciais e ampliem sua capacidade de avaliar criticamente problemas socioambientais sob a perspectiva matemática.

4.7 Atividade 7: Decaimento Radioativo

Tempo estimado: 200 min
Objetivos da atividade:

- Compreender a função exponencial decrescente no contexto do decaimento radioativo, bem como a aplicação de logaritmos para a resolução de problemas inversos.

- Utilizar a linguagem Python como ferramenta de simulação e visualização de fenômenos naturais, articulando conhecimentos matemáticos e computacionais em uma abordagem interdisciplinar.

Habilidades da BNCC

- (EM13MAT304) Resolver e elaborar problemas com funções exponenciais nos quais seja necessário compreender e interpretar a variação das grandezas envolvidas, em contextos como o da Matemática Financeira, entre outros.

- (EM13MAT305) Resolver e elaborar problemas com funções logarítmicas nos quais seja necessário compreender e interpretar a variação das grandezas envolvidas, em contextos como os de abalos sísmicos, pH, radioatividade, Matemática Financeira, entre outros;

- (EM13MAT405) Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

Resíduos nucleares são produzidos em diversas áreas, como usinas de energia, hospitais e indústrias. Mesmo em pequenas quantidades, esses materiais podem permanecer perigosos por longos períodos, devido à radiação que continuam emitindo ao longo do tempo. O processo pelo qual uma substância radioativa perde gradualmente sua atividade é chamado de decaimento radioativo, sendo o conceito de meia-vida definido como o tempo necessário para que a substância perca metade da sua radioatividade. Esse processo é classicamente modelado por funções exponenciais decrescentes, e sua análise pode envolver o uso de logaritmos.

Nesta atividade, os estudantes irão simular o decaimento de dois elementos radioativos com diferentes meias-vidas, calcular o tempo necessário para que atinjam níveis considerados seguros e refletir sobre os desafios éticos, científicos e ambientais relacionados ao descarte de lixo nuclear.

A atividade será desenvolvida em duas etapas. Na primeira, busca-se que os estudantes compreendam intuitivamente o conceito de meia-vida e percebam que o modelo exponencial é adequado para descrever esse tipo de comportamento. Na segunda, espera-se que eles resolvam equações logarítmicas para estimar o tempo necessário para que a radioatividade atinja determinado nível, desenvolvendo a capacidade de interpretação e análise de dados científicos, por meio do uso de tabelas com base em modelos matemáticos e computacionais.

Contextualização: Descobrindo a meia-vida de um elemento radioativo

Nesta atividade, os estudantes são convidados a assumir o papel de cientistas em formação diante de um conjunto de dados experimentais referentes à quantidade de um isótopo radioativo desconhecido ao longo do tempo. O objetivo consiste em analisar os dados fornecidos, identificar a meia-vida do elemento e construir um modelo matemático que descreva seu decaimento radioativo.

A Tabela 12 apresenta dados simulados de um experimento que mede a quantidade (em gramas) de um elemento radioativo ao longo de 30 anos. A partir desses dados, espera-se que os estudantes analisem o comportamento do decaimento radioativo e, por meio de observação e modelagem, estimem a meia-vida aproximada do elemento.

Tabela 12 — Variação da quantidade de material radioativo ao longo do tempo

Tempo (anos)	Quantidade de material (g)
0	100
5	71
10	50
15	35
20	25
25	18
30	12

Fonte: Elaboração própria (2025).

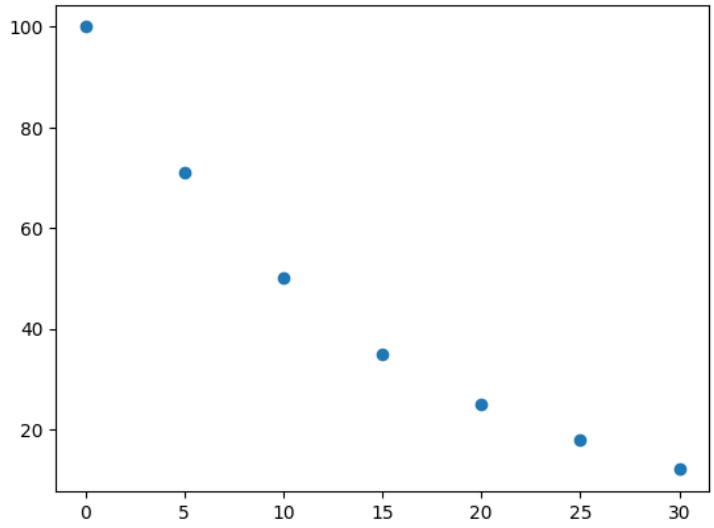
Produção

Com base na tabela apresentada, os estudantes devem construir o gráfico que representa a quantidade de material radioativo em função do tempo e observar o comportamento da curva gerada. A partir do gráfico, obtido como na Figura 18, e dos dados numéricos, os estudantes devem estimar em que intervalo de tempo a quantidade de material se reduz aproximadamente à metade, permitindo, assim, a determinação da meia-vida do isótopo em análise.

```
1 import matplotlib.pyplot as plt
2 # Dados
3 tempo = [0, 5, 10, 15, 20, 25, 30]
4 quantidade = [100, 71, 50, 35, 25, 18, 12]
5
6 # Plotagem dos pontos experimentais
7 plt.plot(tempo, quantidade, 'o', label='Dados experimentais')
8 plt.show()
```

Código 65 — Gráfico do material radioativo ao longo do tempo

Figura 18 — Dados experimentais



Fonte: Elaboração própria (2025).

Modelagem Matemática

O número e (aproximadamente $2,718$) é um número irracional que aparece naturalmente sempre que há crescimento ou decaimento contínuo, ou seja, que não ocorre em etapas, mas a cada instante, sem parar.

Um elemento radioativo se desintegra aos poucos. Em cada segundo, uma fração do que ainda existe se transforma. Assim, como ocorre no crescimento de juros compostos, o novo produto é obtido a partir do anterior, como no caso do decaimento radiotivo está perdendo a cada segundo, considere r a taxa de decaimento por unidade de tempo. Assim temos que se medir de segundo a segundo pode utiliza a função:

$$Q(t) = Q_0(1 - r)^t$$

em que $Q(t)$ representa a quantidade de material radioativo em relação tempo t, Q_0 é a quantidade inicial de material e r a taxa de decaimento. Mas se essa desintegração for contínua, ou seja, em cada microscópico instante, uma fração infinitamente pequena se perde, o comportamento é diferente. Aí a fórmula se transforma em:

$$Q(t) = Q_0 e^{-kt} \tag{4.1}$$

em que k é a constante de decaimento.

Portanto, o número e aparece quando a mudança acontece a todo instante, em vez de acontecer de tempos em tempos. Como o decaimento radioativo acontece de forma contínua, o modelo matemático que melhor o representa é uma função exponencial com base e.

A partir daí, se soubermos a meia-vida de um elemento podemos responder bastantes questões a respeito do comportamento desse elemento. Assim, seja T a meia-vida do elemento químico, logo

$$Q(T) = \frac{Q_0}{2} \tag{4.2}$$

Substituindo 4.2 em 4.1, temos

$$\begin{aligned} Q(t) &= Q_0 e^{-kt} \\ Q(T) &= Q_0 e^{-kT} \\ \frac{Q_0}{2} &= Q_0 e^{-kT} \end{aligned}$$

Como $Q_0 \neq 0$ segue

$$\begin{aligned} \frac{1}{2} &= e^{-kT} \\ \ln\left(\frac{1}{2}\right) &= \ln(e^{-kT}) \\ \ln(1) - \ln(2) &= -kT \cdot \ln(e) \\ 0 - \ln(2) &= -kT \\ k &= \frac{\ln 2}{T} \end{aligned}$$

Assim, a constante de decaimento é dada por $k = \frac{\ln 2}{T}$. Mas

$$e^{-kt} = e^{-\frac{\ln 2}{T}t} = (e^{-\ln 2})^{\frac{t}{T}} = (e^{\ln 2^{-1}})^{\frac{t}{T}} = (e^{\ln(\frac{1}{2})})^{\frac{t}{T}} = \left(\frac{1}{2}\right)^{\frac{t}{T}}$$

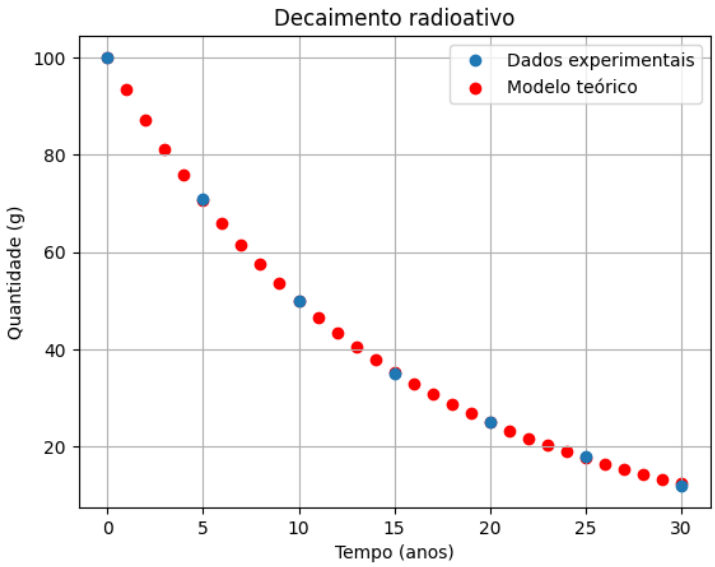
Portanto, temos que a função que modela o decaimento radioativo é

$$Q(t) = Q_0 \left(\frac{1}{2}\right)^{\frac{t}{T}}$$

em que t é o tempo e T é a meia-vida do elemento químico.

Observação: Note que o tempo t deve ser contínuo, então ao utilizar listas como nas atividades anteriores teremos um conjunto discreto que ocasionará um erro ao aplicar na fórmula. Para contornar o problema devemos utilizar a biblioteca Numpy, em que a função `range`, vira a função `np.arange` que funciona de maneira similar. Assim, temos na Figura 19 o gráfico gerado a partir do código abaixo.

Figura 19 — Decaimento utilizando um conjunto discreto



Fonte: Elaboração própria (2025).

```python
import matplotlib.pyplot as plt
import numpy as np

# Dados
tempo = [0, 5, 10, 15, 20, 25, 30]
quantidade = [100, 71, 50, 35, 25, 18, 12]

# Plotagem dos pontos experimentais
plt.plot(tempo, quantidade, 'o', label='Dados experimentais')
# Teste com meia-vida aproximada de 10 anos
T = 10
Q0 = 100
t = np.arange(0,31)
modelo = Q0 * (1/2)**(t / T)

# Plot da função modelada
plt.scatter(t, modelo, color='red',label='Modelo teórico')
plt.title('Decaimento radioativo')
plt.xlabel('Tempo (anos)')
plt.ylabel('Quantidade (g)')
plt.legend()
plt.grid()
plt.show()
```

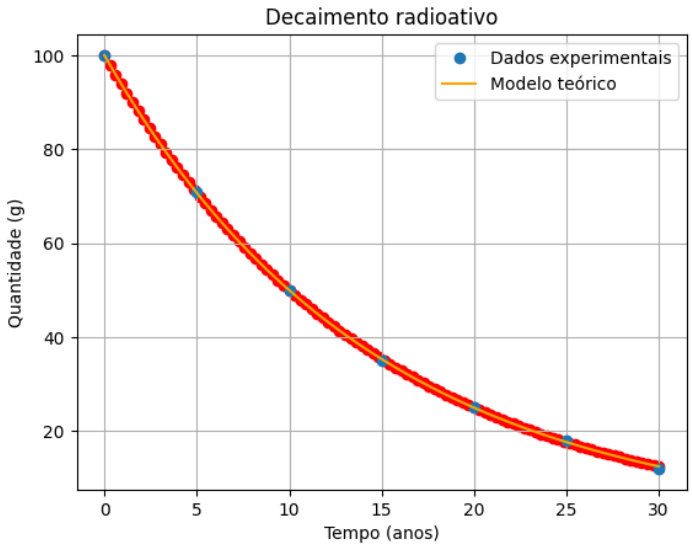
Código 66 — Teste com meia-vida com conjunto discreto

Se utilizarmos a função da biblioteca Numpy `linspace` que é usada para gerar um vetor de números linearmente espaçados entre dois pontos. Daí, obteremos um comportamento como demonstrado na Figura 20.

```
import matplotlib.pyplot as plt
import numpy as np
# Dados
tempo = [0, 5, 10, 15, 20, 25, 30]
quantidade = [100, 71, 50, 35, 25, 18, 12]
# Plotagem dos pontos experimentais
plt.plot(tempo, quantidade, 'o', label='Dados experimentais')
# Teste com meia-vida aproximada de 10 anos
T = 10
Q0 = 100
t_continuo = np.linspace(0, 30, 100)
modelo = Q0 * (1/2)**(t_continuo / T)
# Plot da função modelada
plt.scatter(t_continuo,modelo,color='red')
plt.plot(t_continuo, modelo, label='Modelo teórico', color='orange')
plt.title('Decaimento radioativo')
plt.xlabel('Tempo (anos)')
plt.ylabel('Quantidade (g)')
plt.legend()
plt.grid()
plt.show()
```

Código 67 — Teste com meia-vida com conjunto contínuo

Figura 20 — Decaimento utilizando um conjunto contínuo



Fonte: Elaboração própria (2025).

Perguntas para Reflexão

1. Comparando os dados da tabela, em quais anos a quantidade de material está mais próxima da metade da quantidade anterior? A redução observada se mantém constante em valores absolutos ou depende da quantidade restante de material?

2. A partir da análise dos dados, aproximadamente quanto tempo é necessário para que a substância se reduza à metade? Você identifica algum padrão nesse comportamento? Que tipo de função pode representar esse fenômeno?

3. Ao construir o gráfico com os dados fornecidos, que tipo de curva é possível observar? Ela se assemelha a uma função linear, quadrática ou exponencial?

4. Com base na função ajustada ao comportamento da substância, qual seria a quantidade esperada de material após 40 anos?

5. Se a quantidade inicial de material fosse de 200 gramas, como o gráfico se modificaria?

Etapa 2

Contextualização: Césio-137 e Plutônio

Resíduos nucleares podem permanecer radioativos por centenas ou milhares de anos, o que torna seu descarte um desafio ambiental, ético e tecnológico. Entender quanto tempo um material leva para se tornar menos perigoso é fundamental para políticas públicas de segurança. Os elementos radioativos Césio-137 e Plutônio-239 possuem meia-vida de 30 anos e 24000 anos, respectivamente.

Nesta atividade, o estudante deve acompanhar o decaimento de dois elementos radioativos reais, ano a ano, construindo uma tabela com os valores de sua radioatividade restante ao longo do tempo. Ao fim, poderá comparar a eficácia de diferentes estratégias de armazenamento, com base nos dados obtidos.

Produção

Construa uma tabela em Python cujas colunas são: ano, quantidade restante, porcentagem do material que resta e fração decaída até o momento. Para tal, usaremos a biblioteca Pandas. Observe o comportamento ao longo dos anos do elemento Césio-137:

```python
import numpy as np
import pandas as pd

# Dados iniciais
Q0 = 100  # massa inicial
T = 30.17  # meia-vida do Cesio-137, em anos
k = np.log(2) / T

anos = np.arange(0, 151, 10)  # de 0 a 150 anos, de 10 em 10
dados = []

for t in anos:
    Qt = Q0 * np.exp(-k * t)
    percentual = (Qt / Q0) * 100
    decaido = 100 - percentual
    dados.append([t, round(Qt, 2), round(percentual, 2), round(decaido, 2)])

tabela = pd.DataFrame(dados, columns=["Ano", "Q(t) (g)", "% restante", "% decaido"])
print(tabela)
```

Código 68 — Tabela Césio-137

Saída Esperada:

```
    Ano  Q(t) (g)  % restante  % decaido
0     0    100.00      100.00       0.00
1    10     79.37       79.37      20.63
2    20     63.00       63.00      37.00
3    30     50.00       50.00      50.00
4    40     39.69       39.69      60.31
5    50     31.50       31.50      68.50
6    60     25.00       25.00      75.00
7    70     19.84       19.84      80.16
8    80     15.75       15.75      84.25
9    90     12.50       12.50      87.50
10  100      9.92        9.92      90.08
11  110      7.87        7.87      92.13
12  120      6.25        6.25      93.75
13  130      4.96        4.96      95.04
14  140      3.94        3.94      96.06
15  150      3.13        3.13      96.88
```

Para gerar a tabela do Plutônio-239 basta mudar a meia-vida (T) para 24000 anos. Em consequência, deve-se ajustar o limite de ano de 151 para algo mais adequado como 100000 anos, com intervalos de milhares de anos.

Perguntas para Reflexão

1. Em que momento o material atinge menos de 10% da sua quantidade original?
2. Em que momento ele se torna praticamente inofensivo (ex: $<1\%$)?
3. Qual elemento representa um risco maior para o futuro?

Socialização

Após a construção da tabela de decaimento e a resolução das questões, os estudantes são convidados a refletir e discutir os impactos sociais, ambientais e éticos do uso de elementos radioativos, com foco especial no Césio-137 e no Plutônio-239.

Para promover a socialização dos conhecimentos, podem ser formados grupos que podem compartilhar suas reflexões de diferentes formas: por meio de debates orientados, apresentações orais, cartazes argumentativos ou textos escritos coletivos. O professor atua como mediador, incentivando a escuta ativa, o respeito à diversidade de opiniões e o aprofundamento crítico das ideias.

5 CONSIDERAÇÕES FINAIS

Esta dissertação teve como objetivo apresentar uma proposta didática para uma eletiva do Ensino Médio, voltada à exploração de conceitos matemáticos por meio da linguagem de programação Python, em consonância com as diretrizes da BNCC e as demandas da formação contemporânea.

Ao longo do trabalho, foram desenvolvidas sete atividades interdisciplinares que articulam conteúdos de Matemática com fundamentos básicos de programação. A proposta buscou oferecer aos estudantes um primeiro contato com a linguagem de programação de forma acessível e contextualizada, promovendo, ao mesmo tempo, o desenvolvimento de habilidades como pensamento crítico, letramento digital, pensamento computacional, interpretação de dados, raciocínio lógico e autonomia na resolução de problemas, todas apontadas como essenciais para o estudante do século XXI.

A proposta também está em conformidade com a parte diversificada do currículo do Novo Ensino Médio, especialmente no que se refere aos itinerários formativos, que permitem a construção de percursos flexíveis e integrados. A linguagem de programação, nesse contexto, aparece como uma boa ferramenta para o desenvolvimento de práticas pedagógicas interdisciplinares, com potencial de engajar os estudantes em situações de aprendizagem mais significativas.

Espera-se que a implementação dessa proposta contribua para ampliar o repertório metodológico dos professores, oferecendo-lhes subsídios práticos para integrar programação e Matemática em sala de aula. Do ponto de vista dos estudantes, a expectativa é que as atividades favoreçam a construção de uma aprendizagem mais crítica, criativa e contextualizada, fortalecendo competências indispensáveis para a formação integral no século XXI.

Como desdobramento futuro, sugere-se a implementação piloto da eletiva em escolas de Ensino Médio, preferencialmente naquelas que já contam com carga horária ampliada ou com cursos técnicos na área de tecnologia, bem como a realização de estudos que investiguem os efeitos da abordagem sobre o desempenho e o engajamento dos estudantes. Também seria relevante adaptar a proposta para outros níveis de ensino e para diferentes contextos educacionais.

Em um cenário educacional em constante transformação, espera-se que esta proposta contribua para fortalecer práticas pedagógicas inovadoras, capazes de preparar os estudantes para os desafios do mundo contemporâneo por meio de uma aprendizagem significativa, crítica e transformadora, alinhada às competências exigidas na atualidade.

REFERÊNCIAS

BRASIL. **Base Nacional Comum Curricular**. Brasília: Ministério da Educação, 2018. http://basenacionalcomum.mec.gov.br/. Acesso em: 01 abr. 2025.

_____. **Lei nº 13.415, de 16 de fevereiro de 2017**. [S.l.: s.n.], 2017. https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2017/lei/l13415.htm. Acesso em: 18 mai. 2025.

_____. **Lei nº 9.394, de 20 de dezembro de 1996**. [S.l.: s.n.], 1996. https://www.planalto.gov.br/ccivil_03/leis/l9394.htm. Acesso em: 18 mai. 2025.

CLUBES DA OBMEP. **Números Triangulares**. [S.l.: s.n.], 2020. https://clubes.obmep.org.br/blog/sala-de-leitura-numeros-triangulares/. Acesso em: 20 jul. 2025.

CODING BLOCKS. **Matplotlib: Visualization with Python**. [S.l.: s.n.], 2021. https://coding-blocks.github.io/DS-NOTES/1.2%20Introduction%20to%20Matplotlib.html. Acesso em: 20 jul. 2025.

GOOGLE. **Google Colaboratory**. [S.l.: s.n.], 2024. https://colab.research.google.com. Acesso em: 21 abr. 2025.

HUNTER, J. D. Matplotlib: A 2D graphics environment. **Computing in Science & Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

ICE, Instituto de Corresponsabilidade pela Educação. **Caderno de formação: inovações em conteúdo, método e gestão – metodologias de êxito**. 5. ed. Recife: Instituto de Corresponsabilidade pela Educação, 2021.

_____. **Caderno memória e concepção: concepção do modelo pedagógico**. 5. ed. Recife: Instituto de Corresponsabilidade pela Educação, 2021.

_____. **Livreto Institucional ICE**. [S.l.: s.n.], 2023. https://icebrasil.org.br/wp-content/uploads/2023/08/Livreto-Institucional-ICE.pdf. Acesso em: 19 mai. 2025.

INEP, Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira. **Exame Nacional do Ensino Médio: reaplicação 2024 – Caderno 6: Cinza, questão 141**. Brasília: INEP, 2024. https://download.inep.gov.br/enem/provas_e_gabaritos/2024_PV_reaplicacao_PPL_D2_CD6.pdf. Acesso em: 31 jul. 2025.

LUTZ, Mark. **Learning python: Powerful object-oriented programming**. [S.l.]: "O'Reilly Media, Inc.", 2013.

MATHWORKS. **MATLAB: The Language of Technical Computing**. [S.l.: s.n.], 2024. https://www.mathworks.com/products/matlab.html. Acesso em: 19 jun. 2025.

MATPLOTLIB DEVELOPMENT TEAM. **Histórico do Projeto Matplotlib**. [S.l.: s.n.], 2025. https://matplotlib.org/stable/project/history.html. Acesso em: 19 jun. 2025.

MENEZES, Nilo Ney Coutinho. **Introdução à programação com Python: algoritmos e lógica de programação para iniciantes**. São Paulo: Novatec Editora, 2010. ISBN 978-85-7522-250-8.

OECD. **The Future of Education and Skills: Education 2030 – The Future We Want**. [S.l.: s.n.], 2018. `https://www.oecd.org/content/dam/oecd/en/publications/reports/2018/06/the-future-of-education-and-skills_5424dd26/54ac7020-en.pdf`. Acesso em: 28 jun. 2025.

OLIPHANT, Travis E et al. **Guide to numpy**. [S.l.]: Trelgol Publishing USA, 2006. v. 1.

PANDAS DEVELOPMENT TEAM. **About pandas**. [S.l.: s.n.], 2024. `https://pandas.pydata.org/about/`. Acesso em: 25 jun. 2025.

PYTHON SOFTWARE FOUNDATION. **Python Package Index (PyPI)**. [S.l.: s.n.], 2025. `https://pypi.org/`. Acesso em: 19 jun. 2025.

ROSSUM, Guido Van. **Foreword for Programming Python**. [S.l.: s.n.], 1996. `https://www.python.org/doc/essays/foreword/`. Accessed on 1 May 2025. Acesso em: 1 mai. 2025.

VALENTE, José Armando. Pensamento Computacional, Letramento Computacional ou Competência Digital? Novos desafios da educação. **Revista de Educação**, v. 16, n. 43, p. 147–165, 2019.

VALENTE, José Armando; FREIRE, Fernanda Maria Pereira; ARANTES, Flávia Linhalis (Ed.). **Tecnologia e educação: passado, presente e o que está por vir**. Campinas, SP: NIED/UNICAMP, 2018. P. 406. Publicação digital (e-book) no formato PDF. ISBN 978-85-88833-10-4.

WING, Jeannette. Pensamento computacional: um conjunto de atitudes e habilidades que todos, não só cientistas da computação, ficaram ansiosos para aprender e usar. **Revista Brasileira de Ensino de Ciência e Tecnologia**, v. 9, n. 2, 2016.