6º Simpósio Nacional da Formação do Professor de Matemática

INTRODUÇÃO À PROGRAMAÇÃO COM PYTHON PARA SALA DE AULA DE MATEMÁTICA

Dênis Emanuel da Costa Vargas
Jônathas Douglas Santos Oliveira
Luis Alberto D'Afonseca

ANPMat
Associação Nacional dos Professores de Matemática na Educação Básica

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

INTRODUÇÃO À PROGRAMAÇÃO COM PYTHON PARA SALA DE AULA DE MATEMÁTICA

Associação Nacional dos Professores de Matemática na Educação Básica

Presidente:
Marcela Luciano Vilela de Souza

Vice-Presidente:
Sérgio Augusto Amaral Lopes

Diretores:
Ana Luiza de Freitas Kessler
Aroldo Eduardo Athias Rodrigues
Raquel Bodart
Sumaia Almeida Ramos

6º Simpósio Nacional da Formação do Professor de Matemática

Comissão Organizadora:
Ana Luiza de Freitas Kessler
Carmen Vieira Mathias
Edson Sidney Figueiredo
Karine Faverzani Magnago
Lidiane Buligon
Marcela Luciano Vilela de Souza
Renata Magarinus
Sumaia Almeida Ramos
Valéria de Fátima Maciel Cardoso Brum

Comitê Científico:
Ana Luiza de Freitas Kessler
Carmen Vieira Mathias
Claudia Candida Pansonato
Janice Rachelli
Marcela Luciano Vilela de Souza
Renata Magarinus

Comitê Editorial:
Ana Luiza de Freitas Kessler
Aroldo Eduardo Athias Rodrigues
Fábio Simas
Jaqueline Molon
Leonardo Barichello
Letícia Rangel
Marcela Luciano Vilela de Souza
Mateus Gianni Fonseca
Raquel Bodart
Sérgio Augusto Amaral Lopes
Sumaia Almeida Ramos
Vitor Amorim

6º Simpósio Nacional da Formação do Professor de Matemática

INTRODUÇÃO À PROGRAMAÇÃO COM PYTHON PARA SALA DE AULA DE MATEMÁTICA

Dênis Emanuel da Costa Vargas
Jônathas Douglas Santos Oliveira
Luis Alberto D'Afonseca

1ª edição

2025

Rio de Janeiro

ANPMat

Introdução à programação com Python para Sala de Aula de Matemática

Copyright © 2025 Dênis Emanuel da Costa Vargas, Jônathas Douglas Santos Oliveira e Luis Alberto D'Afonseca

Direitos reservados pela Associação Nacional dos Professores de Matemática na Educação Básica. A reprodução não autorizada desta publicação, no todo ou em parte, constitui violação de direitos autorais. (Lei 9.610/98)

Projeto gráfico: Gabriel Brasil Nepomuceno

Produção editorial: Editora Pi
www.editorapi.com.br | contato@editorapi.com.br | +55 21 97748-7208

Distribuição: Associação Nacional dos Professores de Matemática na Educação Básica
www.anpmat.org.br | editoraanpmat@anpmat.org.br

ISBN: 978-65-88013-37-3

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Vargas, Dênis Emanuel da Costa
Introdução à programação com Python para sala de aula de matemática [livro eletrônico] / Dênis Emanuel da Costa Vargas, Jônathas Douglas Santos Oliveira, Luis Alberto D'Afonseca. -- 1. ed. -- Rio de Janeiro : Associação Nacional dos Professores de Matemática na Educação Básica - ANPMat, 2025.
PDF

Bibliografia.
ISBN 978-65-88013-37-3

1. Educação 2. Formação docente - Metodologias ativas 3. Matemática - Estudo e ensino 4. Python (Linguagem de programação para computadores)
I. Oliveira, Jônathas Douglas Santos.
II. D'Afonseca, Luis Alberto. III. Título.

25-267547 CDD-510.7

Índices para catálogo sistemático:

1. Matemática : Estudo e ensino 510.7

Aline Graziele Benitez - Bibliotecária - CRB-1/3129

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Sobre os autores

Dênis Emanuel da Costa Vargas
denis.vargas@cefetmg.br

Doutor em Modelagem Computacional pela UFJF, é Professor Titular do Departamento de Matemática do CEFET-MG e docente do PROFMAT, onde leciona a disciplina Recursos Computacionais no Ensino de Matemática. Possui experiência na formação inicial e continuada de professores de Matemática, atuando na orientação de mestrandos e em projetos de extensão voltados para o uso de Tecnologias Digitais de Informação e Comunicação na sala de aula de Matemática. Seu trabalho tem ênfase na criação de applets com o software GeoGebra e no desenvolvimento de algoritmos na linguagem de programação Python.

Licenciado e mestre em Matemática pela Universidade Federal de Alagoas (UFAL) e doutor em Matemática Aplicada pela Universidade Estadual de Campinas (UNICAMP). Atualmente, é professor efetivo do Departamento de Matemática do CEFET-MG (Belo Horizonte), onde atua nas disciplinas de graduação e no mestrado profissional em Matemática (PROFMAT). Desenvolve projetos de ensino e extensão, além de lecionar e orientar alunos no PROFMAT. Suas áreas de interesse incluem biomatemática, fuzzy, modelagem no ensino, tecnologias educacionais e metodologias de ensino.

Jônathas Douglas Santos de Oliveira
jonathasoliveira@cefetmg.br

Luis Alberto D'Afonseca
luis.dafonseca@cefetmg.br <https://sites.google.com/view/prof-luis-dafonseca>

Possui bacharelado, mestrado e doutorado em Matemática Aplicada pela UNICAMP. Atuou como pesquisador do Centro de Tecnologia da CGGVeritas, trabalhando em projetos de Imageamento do Pré-Sal. Atualmente é docente do Departamento de Matemática do CEFET-MG e membro do PROFMAT, onde busca desenvolver projetos que apresentem as aplicações da Matemática ou explorem recursos tecnológicos no ensino.

Sumário

Sobre os autores ii

Prefácio vi

1 Introdução 1
1.1 Python . . . 2
1.2 Interfaces de Desenvolvimento . . . 3

2 Introdução ao Python 6
2.1 Operações e Funções Matemáticas . . . 7
2.2 Armazenando Informações . . . 9
2.3 Estruturas de Dados . . . 13
2.3.1 Listas . . . 13
2.3.2 Dicionários . . . 16
2.3.3 Conjuntos . . . 16
2.4 Atividades e Exercícios . . . 18

3 Bibliotecas Matemáticas 20
3.1 Matemática . . . 21
3.2 Números Complexos . . . 22
3.3 Precisão Arbitrária e Constantes . . . 23
3.4 Computação Simbólica . . . 25
3.5 Computação Simbólica *versus* Computação Numérica . . . 29
3.6 Vetores e Matrizes . . . 31
3.7 Gerando Gráficos . . . 33
3.8 Atividades e Exercícios . . . 36

4 Introdução à Programação 38
4.1 Funções . . . 39
4.2 Decisões . . . 41
4.3 Repetições . . . 43
4.3.1 Número fixo de repetições . . . 43
4.3.2 Repetir até atingir o objetivo . . . 45
4.4 Atividades de Experimentação com Python . . . 46
4.5 Atividades e Exercícios . . . 50

5 Simulando Logo em Python 53
5.1 A Biblioteca Tartaruga . . . 54
5.1.1 Construindo Polígonos . . . 54
5.1.2 Construindo Círculos . . . 56
5.1.3 Construindo Fractais . . . 58
5.2 Atividades e Exercícios . . . 60

6 Considerações Finais 62

Respostas aos Exercícios 65

Índice Remissivo 78

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Prefácio

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Este trabalho é resultado do minicurso *Introdução à Programação com Python para Sala de Aula de Matemática* ministrado no 6º Simpósio Nacional da Formação do Professor de Matemática, realizado em 2023 pela Associação Nacional dos Professores de Matemática na Educação Básica – ANPMat e Universidade Federal do Estado do Rio de Janeiro - UniRio e reflete a experiência dos autores ministrando cursos similares em projetos de extensão e outros eventos.

Nele são apresentadas algumas das atividades desenvolvidas nesse minicurso sobre possibilidades de uso da linguagem de programação Python por professores de Matemática em sala de aula, uma vez que algoritmos têm sido tendência nos últimos anos e seu uso incentivado por pesquisas e documentos oficiais tais como a Base Nacional Comum Curricular (BNCC):

> Associado ao pensamento computacional, cumpre salientar a importância dos algoritmos e de seus fluxogramas, que podem ser objetos de estudo nas aulas de Matemática. Um algoritmo é uma sequência finita de procedimentos que permite resolver um determinado problema. Assim, o algoritmo é a decomposição de um procedimento complexo em suas partes mais simples, relacionando-as e ordenando-as, e pode ser representado graficamente por um fluxograma. (BRASIL, 2018, p.271).

O Python é uma linguagem de programação de alto nível que se tem mantido entre as mais populares dentre as diversas linguagens de programação disponíveis atualmente. Além disso, ela se destaca por ser uma das principais opções para se trabalhar com Matemática, uma vez que oferece várias possibilidades de aplicação e possui uma extensa lista de bibliotecas prontas para uso, acelerando o desenvolvimento do algoritmo.

Não é intenção deste texto dar um curso completo de programação em Python (como tantos que já se encontram disponíveis na internet), e sim focar apenas em sua parte matemática. O leitor poderá encontrar mais informações sobre o Python e alguns tutoriais, bem como fazer seu *download*, no *site* oficial da linguagem. Entretanto, para o propósito desse material, isso não será necessário, já que vamos utilizar o Google Colaboratory (ou simplesmente Colab) diretamente no navegador de internet.

Além de incluir algumas atividades adaptáveis para a sala de aula de Matemática, o livro fornece aos professores um conhecimento inicial sobre os recursos da linguagem Python, permitindo que desenvolvam suas próprias atividades e integrem a programação com Python em sua prática docente. Como ele é destinado a introduzir a programação com Python para professores de Matemática, este livro concentra-se em apresentar os fundamentos da linguagem, como sintaxe básica, estruturas de controle, manipulação de dados, criação de funções e uso de bibliotecas relevantes. Ele demonstra a aplicação de Python na resolução de problemas matemáticos e sugere maneiras de integrar a programação ao currículo de Matemática, como a utilização do Logo, por exemplo.

Para tirar o máximo proveito deste livro, é aconselhável que o professor de Matemática leitor possua conhecimentos básicos de lógica e esteja familiarizado com o uso de um computador. No entanto, o pré-requisito mais importante é ter disposição para aprender.

Acreditamos que uma efetiva integração do Python nas aulas de Matemática requer a capacitação adequada dos professores por meio de experiências de aprendizado com a linguagem. Dessa forma, esse texto foi construído buscando proporcionar boas experiências de aprendizado e exemplificar como o Python pode enriquecer as práticas educacionais dos professores de Matemática, tanto em formação inicial quanto continuada. Esperamos que o professor de Matemática leitor sinta-se mais seguro e confortável ao incorporar o Python em suas práticas docente.

Belo Horizonte, janeiro de 2024

Os Autores

1

Introdução

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Sabemos que aprender uma linguagem de programação pode ser uma experiência enriquecedora e, ao mesmo tempo, desafiadora. Assim, é válido considerar alguns pontos de atenção e pré-requisitos que podem aprimorar significativamente esse processo, tais como domínio de lógica, familiaridade com computadores, possivelmente um pouco de inglês técnico e, talvez o mais importante: persistência e paciência.

Não é intenção deste texto dar um curso completo de programação em Python, mas focar apenas em sua parte Matemática com objetivo de proporcionar experiências ao professor de Matemática leitor, exemplificando como o Python pode contribuir em sua prática docente. Além de oferecer algumas atividades que podem ser adaptadas para o contexto da sala de aula, o livro proporciona aos professores um conhecimento inicial sobre os recursos da linguagem Python para, assim, eles desenvolverem suas próprias atividades integrando a programação de computadores com a linguagem Python em suas aulas de Matemática.

No contexto educacional, a BNCC ressalta a importância dos algoritmos no desenvolvimento do pensamento computacional dos estudantes e menciona que eles podem ser objetos de estudo nas aulas de Matemática. Várias habilidades da BNCC descrevem a inclusão de algoritmos, como, por exemplo, a habilidade do Ensino Fundamental EF06MA04, que diz: "Construir algoritmo em linguagem natural e representá-lo por fluxograma que indique a resolução de um problema simples (por exemplo, se um número natural qualquer é par)". Sabemos que habilidades como essa não necessitam obrigatoriamente da implementação em linguagem de programação para serem desenvolvidas em sala de aula. No entanto, a implementação pode enriquecer a experiência educativa e deixá-la mais completa.

No âmbito do Ensino Médio, destacamos aqui duas habilidades da BNCC que recomendam um trabalho com algoritmos e programação em sala de aula. São elas a habilidade EM13MAT315, que diz "Investigar e registrar, por meio de um fluxograma, quando possível, um algoritmo que resolve um problema" e a habilidade EM13MAT405, que diz "Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática". Dessa forma, é notório o potencial do Python para contribuir efetivamente no desenvolvimento dessas e de outras habilidades da BNCC.

Assim, dividimos este texto em 6 capítulos. Após esta introdução, o Capítulo 2 – Introdução ao Python apresenta os fundamentos da linguagem Python de uma forma simples e direta. A seguir, o Capítulo 3 – Bibliotecas Matemáticas mostra as principais bibliotecas disponíveis na linguagem para realizar cálculos e manipulações matemáticas. No Capítulo 4 – Introdução à Programação são descritos os principais mecanismos para o controle do fluxo de um programa. E finalmente, o Capítulo 5 – Simulando Logo em Python utiliza o Python para simular o Logo, uma linguagem de programação dedicada especificamente para o ensino de programação capaz de realizar construções geométricas complexas de forma simples.

1.1 Python

A programação está fortemente ligada à ideia de algoritmos, que é muito importante para a Matemática. A humanidade já vem utilizando algoritmos desde a antiguidade. Aproximadamente 2500 a.C., os babilônios já utilizavam algorítimos para efetuar divisões. No século IX, Muhammad ibn Mūsā al-Khwārizmī escreveu um livro sobre soluções de equações que nos dá o nome algoritmo usado até hoje.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Porém, apenas no século XX que Alan Mathison Turing finalmente estabelece precisamente o que é um algoritmo, definindo o que hoje chamamos de máquina de Turing.

De modo simplificado e informal, podemos dizer que um algoritmo é um procedimento para resolver um problema. Esse procedimento deve garantir a solução do problema, ou verificar a impossibilidade de obtê-la, e é composto por um número finito de passos sem nenhuma ambiguidade. A principal questão aqui, que foi resolvida por Turing, é definir quais são os passos possíveis que eliminam completamente as ambiguidades.

Programar é escrever um algoritmo em uma linguagem de programação. Assim, a questão dos passos possíveis fica resolvida, pois eles são as instruções definidas na linguagem. Cada linguagem de programação tem um conjunto de instruções pré-definidas, cada uma sendo mais adequada para resolver um tipo específico de problema. Essa é a razão da existência de tantas linguagens, como, por exemplo, Assembler, Fortran, C, Algol, Java, R, Matlab, Lisp, Lua, Python, entre muitas outas.

Entre as diversas linguagens disponíveis atualmente, o Python, lançada em 1991 por Guido van Rossum, é uma das mais populares. É considerada uma linguagem de alto nível, isso é, é uma linguagem onde as instruções são mais próximas da linguagem humana em oposição a linguagens onde as instruções são mais próximas do funcionamento interno do computador. É uma linguagem de uso geral, em oposição a linguagens criadas para algum uso específico como ensino, efetuar cálculos matemáticos ou produzir páginas de internet. Uma característica fundamental do Python é que ela é uma linguagem interpretada, isto é, o interpretador Python executa os comandos assim que o usuário os apresenta. Neste material vamos tirar proveito dessa característica.

Para usarmos qualquer linguagem de programação, precisamos de um programa que a traduza para a linguagem usada de fato pelo computador, que chamamos de linguagem de máquina. Os desenvolvedores normalmente usam programas elaborados que os auxiliam em diversas formas na tarefa de criar programas de computador. Esses programas são chamados de IDEs (*Integrated Development Environment*) ou Interfaces Integradas de Desenvolvimento. Na próxima seção apresentamos uma sugestão de IDE para realizar as atividades propostas neste material.

1.2 Interfaces de Desenvolvimento

Existem diversas interfaces de desenvolvimento adequadas para o Python, elas diferem em complexidade e capacidade. Em geral, é necessário instalar no computador o interpretador Python e alguma dessas interfaces. Para simplificar esse processo vamos utilizar uma interface *online* gratuita que nos permite realizar todas as ações que desejamos sem a necessidade de instalar nenhum *software*: o Google Colaboratory, ou simplesmente Colab.

O Colab é uma IDE *online*, disponibilizada pela Google, que permite escrever e executar um código em Python diretamente no navegador. Para utilizá-la, é necessário entrar em uma conta Google e acessar o Colab. Os arquivos são salvos em uma pasta própria dentro do Google Drive e cada arquivo é chamado de notebook.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Google Colaboratory

Uma vez acessado, você deve abrir um novo notebook conforme opções exibidas nas Figuras 1 e 2. Ao ser aberto, o notebook do Colab irá abrir uma janela como a exibida na Figura 3, onde os códigos Python serão inseridos.

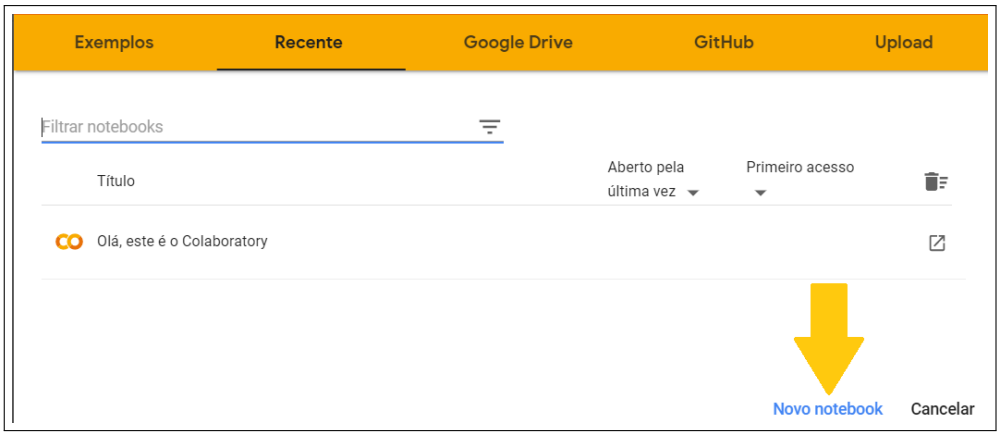


Figura 1: Uma maneira de abrir um novo notebook no Colab.

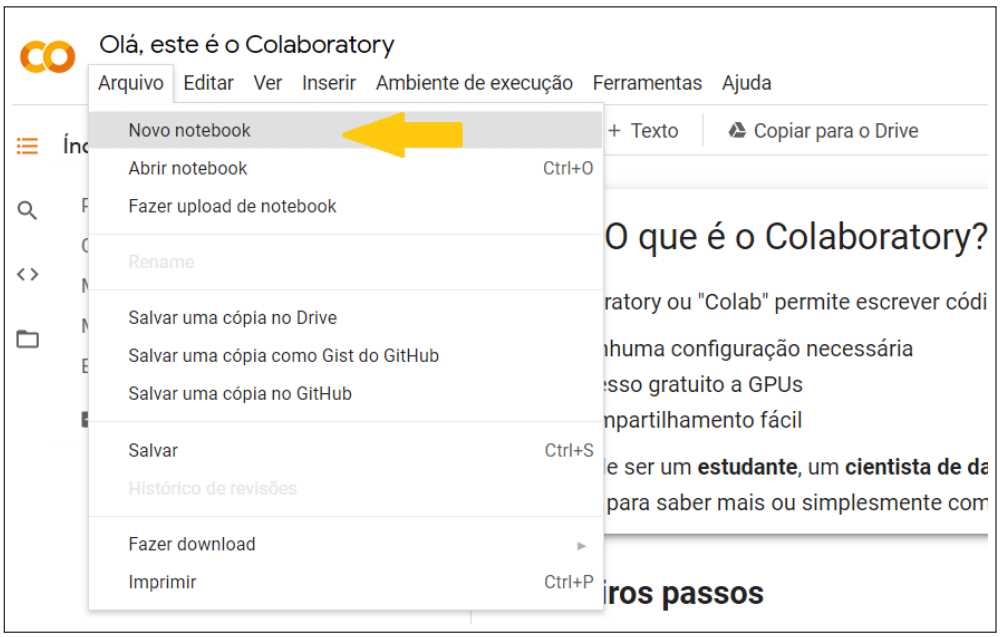


Figura 2: Outra maneira de abrir um novo notebook no Colab.

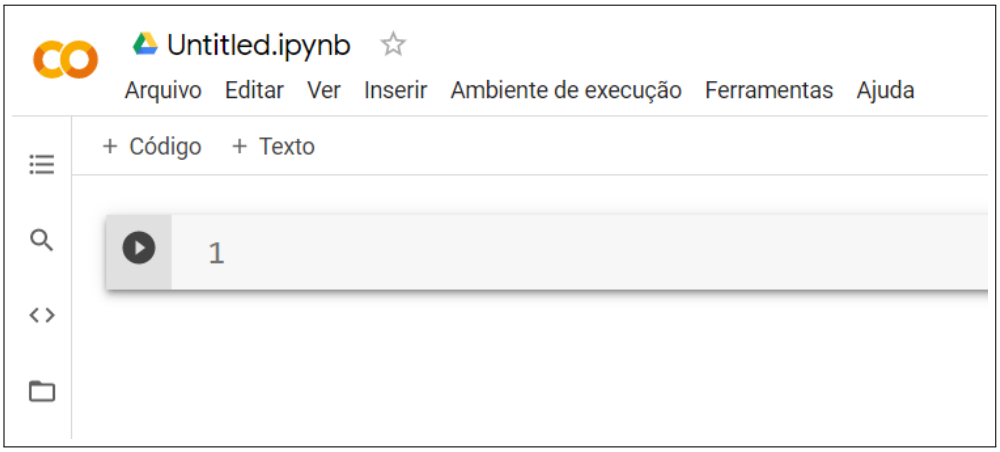


Figura 3: Notebook aberto no Colab.

Observe que o notebook é organizado em células, como podemos ver na Figura 3, onde o número 1 é o contador da célula. Cada célula pode conter texto ou código. As células com texto apresentam informação para o leitor e podem conter equações matemáticas escritas em LaTeX. Já as células com código podem conter uma ou mais linhas de instruções e são executadas quando clicamos o botão de *play* ▶ do seu lado esquerdo.

Um dos códigos mais populares quando se está aprendendo programação é exibir na tela a mensagem "Alô Mundo!". Para isso, você deve digitar `print('Alô Mundo!')` na linha 1 do seu notebook Colab. O comando `print()` exibe (ou imprime, no linguajar da computação) algum tipo de dado na tela. Para executar o código, clique em ▶. O resultado do código executado aparecerá logo abaixo, conforme mostra a Figura 4.

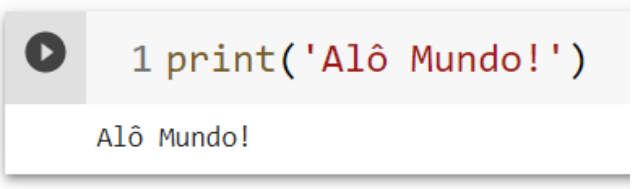


Figura 4: Meu primeiro código em Python.

Vários exemplos e atividades apresentados neste texto estão disponíveis em notebooks Colab. A pasta com todos os notebooks pode ser acessada pelo *link* a seguir.

Pasta do Google Drive com os notebooks Colab disponibilizados para o acompanhar este texto.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

2

Introdução ao Python

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Neste capítulo, vamos mergulhar nos conceitos necessários para iniciar a programação em Python. Começaremos com as operações matemáticas que alimentam cálculos precisos até o uso variáveis para armazenar dados e informações cruciais. Nos próximos capítulos vamos abordar os controladores de fluxo que dirigem o curso de execução do código, bem como apresentaremos as estruturas de repetição. Além disso, apresentaremos o conceito de função nessa linguagem.

2.1 Operações e Funções Matemáticas

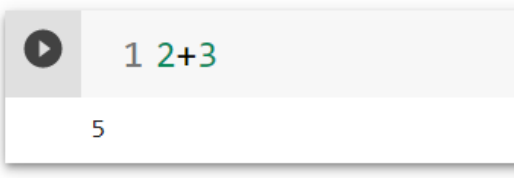


Figura 5: Exemplo de uma soma simples no Python.

O Python pode agir como uma calculadora, fazendo cálculos simples. Basta digitar uma expressão e depois executá-la, para o resultado aparecer imediatamente. A Figura 5 mostra um exemplo e a Tabela 1 mostra os operadores numéricos disponíveis no Python.

Tabela 1: Principais operações numéricas em Python.

Função	Sintaxe	Exemplo	Resultado
Soma	`+`	`2+3`	`5`
Diferença	`-`	`2-3`	`-1`
Produto	`*`	`2*3`	`6`
Quociente	`/`	`2/3`	`0.6666666666666666`
Potência	`**`	`2**3`	`8`
Quociente Inteiro	`//`	`5//3`	`1`
Resto da Divisão	`%`	`5%3`	`2`

O trecho de código abaixo apresenta alguns exemplos iniciais. Você pode digitar uma linha em cada célula do Colab e, ao executá-la, o Python retorna o resultado dos cálculos. Note que se você digitar mais do que uma expressão na mesma célula, o Colab exibe apenas o resultado da última. Mais à frente no texto, veremos como controlar o que é exibido como resposta.

```
3 * (5 + 2)  # Calcula a soma 5 + 2 e depois multiplica por 3
2**3         # Calcula o cubo de 2
10 / 3       # Calcula uma aproximação para a divisão de 10 por 3
10 // 3      # Calcula o quociente inteiro da divisão de 10 por 3
10 % 3       # Calcula o resto da divisão de 10 por 3
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Observe que o texto escrito a partir do caractere # é ignorado pelo Python e não precisa ser digitado para que ele efetue as operações. Esse texto é chamado de comentário e é usado apenas para informar o leitor humano sobre o que o código pretende fazer. É uma recomendação geral na programação escrever comentários informativos com o código para esclarecer aspectos que podem ser obscuros para outras pessoas.

Outro ponto importante: escreva os comandos começando na primeira coluna, isto é, não deixe espaços no início da linha. Mais à frente veremos que o Python usa a coluna onde os comandos começam na linha para definir a estrutura do código. Esse alinhamento do código com relação ao início da linha é chamado de indentação, e o Python vai gerar um erro se a indentação estiver errada.

Por padrão, o Python trabalha com 16 casas decimais. Entretanto, o usuário interessado pode limitar a quantidade de casas com o comando `round`.

```
round(número, quantidade de casas decimais)
```

A Figura 6 mostra um exemplo com 5 casas decimais.

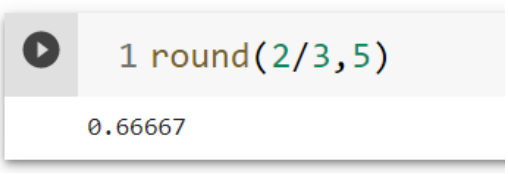


Figura 6: Limitando a quantidade de casas decimais com o comando `round`.

Para se calcular uma raiz n-ésima, basta recorrer à potência $1/n$. Assim, $\sqrt[3]{8}$ é calculado como 8**(1/3). Também pode-se usar representação decimal para o expoente. Por exemplo, $\sqrt{2}$ pode ser calculada como 2**0.5.

Tabela 2: Principais operações relacionais em Python.

Relação	Sintaxe	Exemplo	Resultado
Maior que	>	3 >4	False
Maior ou igual a	>=	3>=4	False
Menor que	<	3 <4	True
Menor ou igual a	<=	3<=4	True
Igual	==	3==4	False
Diferente	!=	3!=4	True

Também podemos verificar a relação entre valores, testando se um valor é maior, menor, igual ou diferente do que outro. A Tabela 2 mostra a lista dos principais operadores relacionais do Python. Cada um desses operadores compara os valores à esquerda e direita e retorna se a relação é verdadeira (`True`) ou falsa (`False`). Note que para o Python esses operadores funcionam como os operadores numéricos, eles recebem dois valores e retornam o resultado. Não podemos usá-los para definir relações como fazemos na Matemática para especificar conjuntos de números.

Um notebook Colab com exemplos do uso do Python como uma calculadora pode ser acessado pelo *link* a seguir.

Notebook Colab com exemplos do uso Python para efetuar operações matemáticas.

2.2 Armazenando Informações

Nesta seção, vamos apresentar como o Python armazena informações na memória do computador. Na próxima, mostramos estruturas elaboradas para armazenar informações mais complexas de modo organizado.

No Python as informações são armazenadas em variáveis, que funcionam como caixas que armazenam informações na memória do computador e servem para guardar diferentes tipos de dados. Cada variável tem um nome único e pode conter um valor específico (numérico ou não). Cuidado para não se confundir com a noção de variáveis na matemática. Aqui variável significa unicamente uma informação específica armazenada na memória.

Os dados que armazenamos podem ser de diferentes tipos como números ou texto. Os principais tipos de dados existentes no Python são:

- Números inteiros;
- Números reais, que são armazenados de forma aproximada;
- Números complexos;
- Texto, que é chamado de *strings* na programação e
- Valores booleanos, isto é, verdadeiro ou falso.

A seguir apresentamos alguns exemplos de cada um desses tipos de dados e como eles são armazenados.

Os números inteiros (int) são representados pela representação direta do número.

```
10
-15
1_345_234
1345234
```

Note que não podemos usar o ponto de separação de milhares, pois para o Python o ponto indica a parte decimal do número, como veremos a seguir. A forma de separar os milhares e facilitar a leitura do valor é usar o caractere de sublinhado _, como mostrado na penúltima linha do exemplo. Entretanto, essa separação não é obrigatória, e o usuário pode digitar o número sem qualquer separação, como mostrado na última linha do exemplo.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Obviamente, os números Reais não podem ser representados com um número infinito de casas decimais (ou binárias no caso da memória do computador). Dessa forma, eles são armazenados usando a técnica de Ponto Flutuante. Na computação, números armazenados com essa técnica são chamados de *float*. Assim, ao se trabalhar com ponto flutuante, sempre devemos lembrar que estamos usando aproximações. Por exemplo, o número 0,1 é uma dízima em binário e não pode ser representado exatamente na memória do computador. O que temos é a seguinte representação aproximada

$$0.1 \approx 0.1000000000000000055511151231257827021181583404541015625$$

Fazendo uma análise simplificada, podemos dizer que o computador consegue armazenar corretamente valores com 16 casas decimais de precisão.

Para representarmos números reais, isto é, números em ponto flutuante, devemos usar o ponto mesmo que o valor decimal seja zero, como nos exemplos a seguir.

`10.0`	$= 10$
`3.1415`	$= 3{,}1415000000$
`1_345_234.25`	$= 1.345.234{,}25$
`1345234.25`	$= 1.345.234{,}25$
`25.3e4`	$= 25,3 \times 10^4 = 253.000{,}0$

Para facilitar a representação de números muito grandes ou muito pequenos podemos usar uma notação similar à notação científica escrevendo o caractere `e` para indicar a potência de 10, como ilustrado no último exemplo.

O Python também trabalha com números complexos. Para criarmos um número complexo devemos usar a função `complex`, como no exemplo

```
complex(1, 2)
```

que cria o número $1 + 2i$. O Python, porém, usa a letra j para indicar a unidade imaginária $\sqrt{-1}$. Assim, no Python esse número é `1+2j`. No próximo capítulo apresentamos como trabalhar com os números complexos em Python.

Para que o programa possa tomar decisões, ele precisa verificar se determinadas condições são verdadeiras ou falsas. Para permitir a manipulação de expressões lógicas, o Python trabalha com valores booleanos, que podem ser verdadeiro (`True`) ou falso (`False`). Note que a primeira letra precisa ser maiúscula nos dois casos.

Textos são representados por *strings*, que são sequências de caracteres escritas entre aspas, como “Alô Mundo!”. As aspas podem ser simples ou duplas.

```
'Um exemplo de texto'
"Agora usando aspas duplas"
```

No caso de a *string* estender-se por mais do que uma linha, podemos usar três aspas, como no exemplo

```
"""Texto com várias linhas
1 - um
2 - dois
```

```
3 - três
"""
```

Podemos usar a função `type()` em Python para descobrir o tipo de dado associado a uma variável ou valor específico. Ela é particularmente útil quando você quer confirmar ou verificar o tipo de uma variável durante a execução do programa. A Figura 7 mostra um exemplo de um código executado usando a função `type()`.

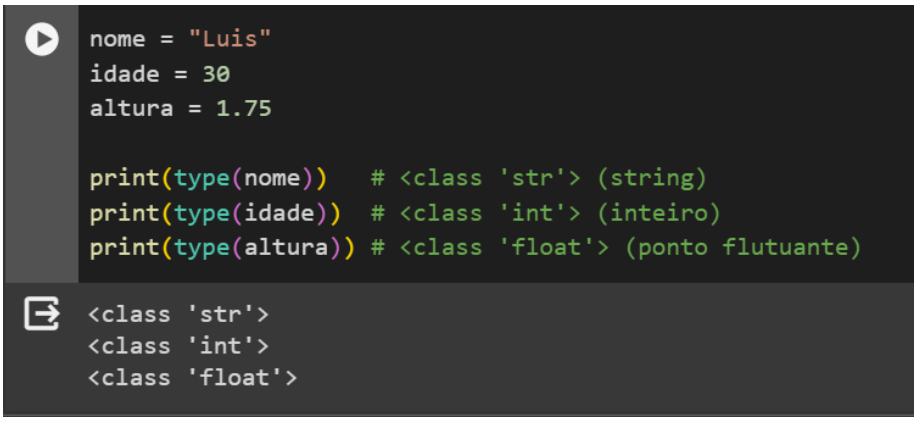


Figura 7: Exemplo: Classes em Python.

Agora que vimos os principais tipos de dados, podemos armazená-los em variáveis, usando o operador de atribuição `=`, como mostrado no exemplo abaixo.

```
a = 3
x = 16.89
nome = "Luis"
```

Cada variável tem um nome usado para identificá-la dentro do código. Esse nome pode ser qualquer sequência de letras e números, podendo conter alguns símbolos como o hífen `-` ou o sublinhado `_`, mas não pode conter espaços ou operadores como o `+`.

A atribuição de valores a variáveis em Python permite armazenar e manipular dados de maneira eficiente. Ao atribuir um valor a uma variável, o usuário está reservando um espaço na memória para armazenar esse dado. É importante notar que, em Python, você não precisa declarar explicitamente o tipo da variável, pois o interpretador infere automaticamente. Assim, a mesma variável pode armazenar diferentes tipos de dados ao longo do programa.

Anteriormente falamos da função `print()`, que é uma função de saída, uma vez que exibe informações para o usuário na tela. Já como função de entrada existe a opção `input()`. Quando você usa `input()`, o programa para e espera que o usuário digite alguma coisa no teclado. O que quer que seja digitado é tratado como uma *string* e pode ser atribuído a uma variável para uso posterior. No exemplo da Figura 8, é solicitado ao usuário que digite o seu nome.

Ao solicitar a entrada via `input()` de uma variável a e tentar somá-la ao número 1 através do código abaixo

```
a=input()
print(a+1)
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

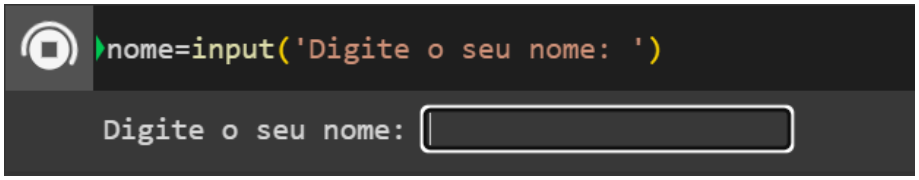


Figura 8: Ilustração da função *input*.

será exibida uma mensagem de erro dizendo que não podemos somar uma *string* com um número. No entanto, você pode converter essa *string* para outros tipos de dados, como inteiros ou *floats*, usando funções de conversão como `int()` ou `float()`. Dessa forma, o código abaixo retornaria o valor variável *a* digitada pelo usuário somada ao número 1.

```
a=int(input())
print(a+1)
```

Em outro exemplo, podemos criar um programa que peça a altura de uma pessoa (variável do tipo *float*), através do comando `altura = float(input("Digite sua altura: "))`.

Um notebook com exemplos de criação de variáveis pode ser acessado pelo *link* a seguir.

Notebook Colab com exemplos do armazenamento de informações em variáveis.

Cabe aqui uma observação sobre a semelhança do operador de atribuição `=` e o igual, `==`. Na matemática usamos o símbolo `=` com vários significados diferentes, confiando que o leitor será capaz de identificar o significado correto pelo contexto. Mas ao escrevermos programas, precisamos evitar todas as ambiguidades. Assim, usamos símbolos diferentes para a atribuição `=` e para a comparação de valores `==`.

O operador de comparação `==` verifica se os valores são iguais e retorna verdadeiro ou falso, dependendo do caso. Por exemplo,

```
4 == 3
```

retorna falso, `False`, enquanto

```
4 == 2 + 2
```

retorna verdadeiro, `True`.

Além disso, o operador de atribuição apenas indica que o Python deve armazenar um valor em uma variável. Ele não é usado para resolver equações. Assim, é perfeitamente válido fazer

```
x = 1
x = x + 1
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Isso instrui o Python para armazenar o valor 1 na variável `x`, depois pegar o valor armazenado na variável, somar 1 a ele e armazenar o resultado novamente na variável, que passa a conter o valor 2. Observe que a expressão `x+1=x` não faz sentido e resultará em um erro de sintaxe, pois o programador está tentando atribuir `x` ao valor de `x+1`, o que não é válido. A expressão à esquerda do operador de atribuição `=` deve ser uma variável que pode armazenar um valor, e não uma expressão matemática. O *link* a seguir aponta para um notebook que compara os operadores de atribuição e de comparação.

Notebook Colab comparando o operador de atribuição `=` com o operador de comparação `==`.

2.3 Estruturas de Dados

Vimos na seção anterior que o Python armazena informações em variáveis. Até então, as informações que usamos eram um único número ou um texto, mas é comum precisarmos armazenar quantidades maiores de informação de forma organizada, para que depois possam ser acessadas com simplicidade e rapidez. Para isso usamos as estruturas de dados. Elas servem para armazenar coleções de variáveis de modo organizado e são tratadas como uma única variável pelo Python.

Para cada tipo de informação que desejamos armazenar existe uma estrutura mais adequada. O Python oferece quatro estruturas nativamente:

Listas: Uma coleção de valores indexada numericamente;

Dicionários: Uma coleção de valores indexado por chaves e

Conjuntos: Uma coleção de valores não ordenada e não indexável.

Ao longo desta seção, vamos explicar cada uma dessas estruturas.

Se desejarmos usar outras estruturas de dados, como vetores e matrizes, podemos usar bibliotecas que fornecem suas definições. No Capítulo 3 discutiremos mais sobre isso.

2.3.1 Listas

As listas permitem armazenar uma coleção ordenada de elementos. Esses elementos podem ser de diferentes tipos, como números, *strings*, ou até mesmo outras listas. A principal característica das listas é a capacidade de modificar, adicionar ou remover elementos após sua criação. Para utilizar listas em Python você deve utilizar o símbolo `[]` (colchetes) para as listas, armazenar a lista em uma variável e separar os itens da lista pela vírgula.

Vamos iniciar com um exemplo simples de lista, com três elementos, conforme mostra a Figura 9. Se quisermos acessar um item da lista criada, precisamos usar a estrutura: `nomedalista[posição]`. Por exemplo, se quisermos acessar a posição 1 da lista devemos digitar `minha_lista[1]` e aparecerá na tela:

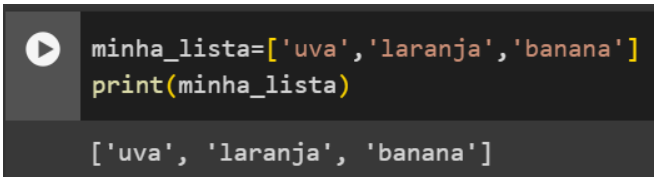


Figura 9: Primeira lista em Python.

```
laranja
```

Vale observar que a numeração dos itens de uma lista em Python começa a partir do índice 0. Isso significa que o primeiro elemento da lista tem o índice 0, o segundo tem o índice 1, e assim por diante.

Quando falamos de listas em Python, é como ter um conjunto de ferramentas versáteis. Você pode inserir um novo elemento no final da fila usando o `append()`, inserir um novo elemento na posição desejada com o `insert()`, ou até remover um elemento usando o `remove()`. Observe a Figura 10.

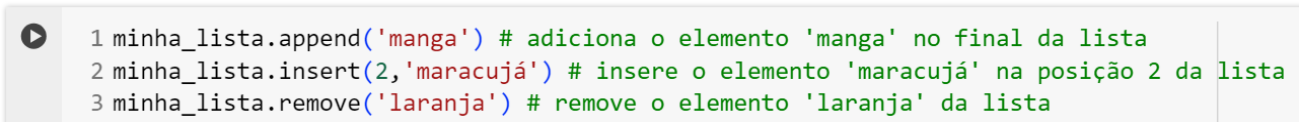


Figura 10: Métodos Listas.

Por vezes é interessante criar uma lista vazia `nomedalista=[]` e ir inserindo elementos a essa lista através do método `nomedalista.append()`. Além disso, se as entradas de uma lista são numéricas, podemos usar o comando `sum(nomedalista)` para obter a soma dos seus elementos, bem como `min(nomedalista)` e `max(nomedalista)` para obter o mínimo e o máximo da lista, respectivamente. O cálculo da média dos elementos de uma lista pode ser obtido dividindo `sum(nomedalista)` por `len(nomedalista)`, onde `len(nomedalista)` é a quantidade de elementos em uma lista.

A seguir apresentamos alguns exemplos de criação e manipulação de listas. Podemos criar listas escrevendo cada um dos seus elementos entre colchetes

```
a = [ 2, 3, 5, 7, 11, 13, 17 ]
b = [ 0, 3.14, True, "texto" ]
```

O primeiro comando cria uma lista chamada `a` que contém os primeiros 7 números primos. O segundo cria uma lista onde cada elemento é de um tipo diferente.

Podemos armazenar qualquer tipo de dado como um elemento de uma lista. Por exemplo, podemos ter listas de listas

```
c = [ 10, a, b ]
```

Esse comando cria uma lista `c` cujo primeiro elemento é o número 10, enquanto o segundo e terceiro elementos são as listas `a` e `b` criadas anteriormente.

As listas podem ser vazias

```
d = []
e = list()
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Os dois comandos fazem o mesmo, que é criar uma lista sem nenhum elemento. Isso é útil quando a lista vai ser preenchida posteriormente, por exemplo, com valores fornecidos pelo usuário.

Tendo criado uma lista, podemos usá-la como uma variável da mesma forma que as demais. Por exemplo, o comando a seguir exibe todos os elementos da lista

```
print(a)
```

Alternativamente, podemos acessar os elementos separadamente fornecendo seu índice, lembrando que o primeiro índice é zero. Por exemplo,

```
print(a[0])
```

retorna o primeiro elemento da lista `a`, que é o número 2.

Podemos pegar os primeiros elementos de uma lista usando o operador "dois pontos"

```
print(a[:4])
```

Esse comando vai retornar os 4 primeiros elementos da lista, isto é, os valores 2, 3, 5 e 7.

Se quisermos os últimos elementos da lista, contamos negativamente a partir do final

```
print(a[-3:])
```

Esse comando retorna os últimos 3 elementos da lista, isto é, os valores 11, 13 e 17.

Podemos também pegar um pedaço no meio da lista, para isso devemos fornecer os valores inicial e final dos índices

```
print(a[1:5])
```

Esse comando vai exibir os elementos da lista com índices começando em 1 e terminando antes de 5, isto é, os elementos 3, 5, 7 e 11 cujos índices são 1, 2, 3 e 4.

Quando queremos alterar o conteúdo de uma posição da lista, basta atribuir o novo valor, como neste exemplo

```
a[0] = "zero"
```

Agora o primeiro elemento da lista `a` contém a *string* `"zero"`

Outra opção é acrescentar novos elementos no final da lista. Uma opção é usar a função `append`, por exemplo,

```
a.append("fim")
```

acrescenta a *string* `"fim"` como um novo último elemento da lista `a`. Também podemos concatenar duas listas com o operador `+`. Essa operação vai criar uma nova lista composta pelos elementos da primeira lista seguidos dos elementos da segunda lista; por exemplo,

```
e = [ 1, 2, 3 ] + [ 10, 20 ]
```

cria a lista `e` composta pelos elementos 1, 2, 3, 10 e 20.

Um notebook com exemplos de manipulação de listas está disponível no *link* a seguir.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Notebook Colab com exemplos do uso de listas.

2.3.2 Dicionários

Um dicionário é uma estrutura de dados que utiliza chaves em vez de índices, como fazem as listas e tuplas. Uma chave pode ser qualquer valor imutável, por exemplo, um número qualquer ou uma *string*. Criamos dicionários colocando os pares *chave*:*valor* entre chaves. Por exemplo,

```
carro = {
    "marca": "Ford",
    "modelo": "Mustang",
    "ano": 1964
}
```

cria um dicionário com as chaves: "marca", "modelo" e "ano". A chave "marca" está associada ao valor "Ford" e assim por diante.

Podemos exibir todo o conteúdo de um dicionário

```
print(carro)
```

ou podemos acessar cada um dos seus valores separadamente

```
print(carro["ano"])
```

retorna o valor 1964.

O notebook a seguir ilustra o uso de dicionários.

Notebook Colab com exemplos de uso de dicionários.

2.3.3 Conjuntos

Já conjuntos são estruturas de dados com o mesmo significado matemático, ou seja, representa uma coleção de elementos não ordenada e sem duplicidade. Criamos conjuntos colocando os elementos entre chaves. Por exemplo,

```
j = { 3, 2, 1 }
```

cria o conjunto `j` que contém os elementos 1, 2, e 3. Podemos também usar a função `set()` para criar conjuntos a partir dos elementos de uma lista ou tupla

```
k = set([ 2, 4, 6 ])
```

Mesmo que valores sejam repetidos na criação de um conjunto, como neste caso

```
l = { 3, 2, 1, 1, 2, 3 } ,
```

o conjunto vai conter apenas uma cópia de cada um.

Não podemos acessar os elementos de um conjunto por índices ou chaves, mas, como na Matemática, podemos verificar se um elemento pertence ou não a um conjunto. Para isso usamos o comando `in`. Por exemplo,

```
3 in k
```

retorna falso, pois o conjunto `k` não contém o elemento 3 e

```
4 in k
```

retorna verdadeiro, pois o conjunto `k` contém o elemento 4.

Também podemos realizar operações com conjuntos. Considere o código

```
A = { 1, 2, 3 }
B = { 3, 4, 5 }
U = A | B
print( "União =", U )
```

As duas primeiras linhas criam os conjuntos `A` e `B` e a terceira linha cria o conjunto `U` como a união entre `A` e `B`. Por fim, o comando `print` exibe a mensagem

```
União = {1, 2, 3, 4, 5}
```

Também podemos calcular a interseção de conjuntos, como no código

```
I = A & B
print('Interseção =', I )
```

que cria o conjunto `I` como a interseção entre `A` e `B` e o `print` exibe a mensagem

```
Interseção = {3}
```

Do mesmo modo podemos calcular a diferença entre conjuntos

```
D = A - B
print('Diferença =', D)
```

O conjunto `D` é a diferença entre `A` e `B` e o `print` exibe a mensagem

```
Diferença = {1, 2}
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Também podemos verificar se um conjunto é subconjunto de outro com a função `issubset`. Observe o código

```
C = {1, 2}
D = {1, 2, 3}
print(C.issubset(D))
print(D.issubset(C))
```

As duas primeiras linhas criam os conjuntos `C` e `D`. O primeiro `print` escreve `True`, pois `C` está contido em `D`. O segundo `print`, porém, escreve `False`, pois `D` não está contido em `C`.

O notebook a seguir apresenta alguns exemplos do uso de conjuntos no Python.

Notebook Colab com exemplos de uso de conjuntos.

2.4 Atividades e Exercícios

Exercício 2.1

Use o Python para calcular os valores solicitados.

1. O quociente inteiro da divisão de 10 por 3.
2. O resto da divisão de 10 por 3.

Clique para ver a resposta

Exercício 2.2

Utilize o Python para avaliar as expressões.

a) $(3+7)^2$

b) $\dfrac{5-2\times 7^3}{5+\sqrt{3}}$

Clique para ver a resposta

Exercício 2.3

Qual o resultado da avaliação, pelo Python, da expressão `5 * 4 / 2 * 3`?

Clique para ver a resposta

Exercício 2.4

Faça um programa em que, dada a temperatura em graus Fahrenheit, retorna a temperatura em graus Célsius.

Dica: $C = \frac{5(F-32)}{9}$

Clique para ver a resposta

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

3

Bibliotecas Matemáticas

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

3.1 Matemática

```
1 import math
2 alfa=math.radians(60)
3 print(math.sin(alfa))
```

```
0.8660254037844386
```

Figura 11: A função `math.radians` passa um ângulo de graus para radianos, e a função `math.sin` calcula o seno de um ângulo em radianos.

Uma biblioteca é um conjunto de funções prontas que facilitam o desenvolvimento dos algoritmos. Para importá-las, devemos digitar `import <nome da biblioteca>`. Por exemplo, `import math` importa a biblioteca `math`, que possui várias funções matemáticas específicas. Para utilizar alguma função da biblioteca `math` (ou de qualquer outra biblioteca), devemos digitar `<nome da biblioteca>.<nome da função>`, como por exemplo `math.<nome da função>`. E, alternativamente ao que foi mencionado para calcular $\sqrt{2}$, podemos exemplificar usando o comando `math.sqrt(2)`, uma vez que a função `math.sqrt` calcula a raiz quadrada. Outro exemplo clássico de funções contidas na biblioteca `math` são as funções trigonométricas. A Figura 11 mostra um exemplo dessas funções executadas no Colab. Outras funções disponíveis da biblioteca `math`, como por exemplo `math.factorial`, podem ser acessadas nesta página. A Tabela 3 mostra algumas dessas funções.

Tabela 3: Principais operações e funções possíveis da biblioteca `math`.

Exemplo	Sintaxe
3^2	`math.pow(3,2)`
$\sqrt{2}$	`math.sqrt(2)`
$\lvert -4 \rvert$	`math.fabs(-4)`
$5!$	`math.factorial(5)`
$MDC(30,5)$	`math.gcd(30,5)`
$MMC(2,5)$	`math.lcm(2,5)`
e^2	`math.exp(2)`
e	`math.e`
π	`math.pi`
$\ln e$	`math.log(math.e)`
$\log_3 27$	`math.log(27,3)`
Hipotenusa dos catetos 3 e 4	`math.hypot(3,4)`
$\operatorname{sen} \frac{\pi}{3}$	`math.sin(math.pi/3)`
$\cos \frac{\pi}{3}$	`math.cos(math.pi/3)`
$\operatorname{tg} \frac{\pi}{3}$	`math.tan(math.pi/3)`

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

É importante observar que a biblioteca `math` em Python não lida explicitamente com representações simbólicas ou formatos específicos ao serem exibidos na tela. Quando você usa funções dessa biblioteca para realizar operações matemáticas, como `sqrt` para a raiz quadrada, o resultado que aparece na tela é geralmente um número do tipo *float* (com uma notação decimal na precisão disponível) e não o símbolo $\sqrt{2}$. Se você precisa de representações simbólicas, você pode usar a biblioteca `SymPy`, a qual será tratada adiante.

Notebook Colab com exemplos do uso da biblioteca `math`.

3.2 Números Complexos

O Python também faz operações com números complexos, onde `j` assume o lugar da unidade imaginária. A Tabela 4 mostra as operações e funções possíveis para números complexos em Python.

Tabela 4: Principais operações e funções possíveis para números complexos em Python. Suponha `a=2+3j` e `b=4-5j`.

Função	Sintaxe	Exemplo	Resultado
Soma	`+`	`a+b`	`6-2j`
Diferença	`-`	`a-b`	`-2+8j`
Produto	`*`	`a*b`	`23+2j`
Quociente	`/`	`a/b`	`-0.17073+0.53659j`
Parte Real	`número.real`	`a.real`	`2`
Parte Imaginária	`número.imag`	`a.imag`	`3`
Conjugado	`número.conjugate()`	`a.conjugate()`	`2-3j`
Valor Absoluto	`abs()`	`abs(a)`	`3.605551275463989`

Alguns comentários sobre números complexos em Python:

- A função `conjugate()` só funciona com os parênteses no final.
- Para se conseguir as 5 casas decimais na divisão da Tabela 4, foi necessário calcular a divisão antes, denominando o resultado dessa divisão por c. Assim, arredondamos com a função round as partes real e imaginária separadamente, conforme mostra a Figura 12.
- Uma função predefinida sobre o argumento de um número complexo está incluída na biblioteca `cmath`, que será mencionada adiante no texto.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
1 a=2+3j
2 b=4-5j
3 c=a/b
4 round(c.real,5)+round(c.imag,5)*1j
```

```
(-0.17073+0.53659j)
```

Figura 12: Utilizando a função round para arredondar as partes real e imaginária separadamente.

A biblioteca `cmath` possui funções específicas para números complexos. A Figura 13 mostra um exemplo no Colab da função `cmath.phase` da biblioteca `cmath`. Ela calcula o argumento de um número complexo em radianos. Uma lista com funções disponíveis da biblioteca `cmath` pode ser acessada nessa página.

```
1 import cmath
2 z=2+3j
3 print(cmath.phase(z))
```

```
0.982793723247329
```

Figura 13: A função `cmath.phase` calcula o argumento de um número complexo em radianos.

O *link* a seguir leva para um notebook que ilustra o uso de números complexos no Python.

Notebook Colab com exemplos do uso da biblioteca de números complexos.

3.3 Precisão Arbitrária e Constantes

Algumas constantes matemáticas podem ser acessadas com as bibliotecas `math` e `cmath`. Os comandos `math.pi` (ou `cmath.pi`) e `math.e` (ou `cmath.e`) exibem os valores de π e e com 15 casas decimais. Se desejar mais precisão, utilize o pacote de multiprecisão, como mostra o código abaixo:

```
from mpmath import mp # biblioteca matemática de multi-precisão
mp.dps = 50           # número de dígitos
print(mp.pi)
```

O valor de π obtido com o código acima é

$$3,14159265358979323846264338327950288419716939937751$$

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Observe que também podemos importar uma biblioteca, função, classe ou constante que esteja dentro de outra, como foi o caso do código descrito acima: `from mpmath import mp`. A biblioteca `mp` foi importada de dentro da `mpmath`. De modo geral, podemos fazer isso através da seguinte sintaxe:

```
from <nome da biblioteca> import <nome da biblioteca, função, classe ou
    constante>
```

Para o caso de importar tudo que está dentro da biblioteca, podemos simplesmente colocar o caractere asterisco (*) da seguinte forma

```
from <nome da biblioteca> import *
```

Note que essa não é a forma recomendada, pois pode levar a conflitos entre nomes de funções e variáveis. Ela só deve ser usara para testes ou exemplos muito simples.

Isso produz o mesmo resultado do comando `import <nome da biblioteca>`. Entretanto, ao usar `from <nome da biblioteca> import *`, as funções não precisam ser chamadas como `<nome da biblioteca>.<nome da função>`. Nesse caso, elas podem ser chamadas diretamente sem especificar o nome da biblioteca de origem. Por exemplo, o código da Figura 13 produz o mesmo resultado se escrito como

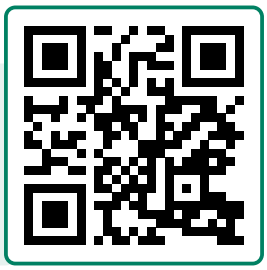
```
from cmath import *
z=2+3j
print(phase(z))
```

O notebook a seguir traz exemplos do uso da biblioteca `mpmath`.

Notebook Colab com exemplos do uso da biblioteca para controlar a precisão.

Outras constantes podem ser importadas da biblioteca `SciPy`, que é bem conhecida e com bastantes funções científicas. Como ela é muito vasta e tem muitas funções úteis, à medida que elas forem sendo necessárias, vamos abordando cada uma separadamente. Uma documentação completa da biblioteca `SciPy` encontra-se no *link* abaixo.

Página da biblioteca SciPy



Como, aqui, queremos apenas as constantes da biblioteca `SciPy`, podemos importá-las especificamente, digitando

```
import scipy.constants
```

ou, equivalentemente,

```
from scipy import constants
```

A Tabela 5 exibe algumas constantes disponíveis na biblioteca `SciPy.Constants` (uma lista completa pode ser acessada nessa página). Observe que os nomes das constantes podem ter diferentes significados dependendo da biblioteca utilizada. Por exemplo, a letra `e`, que na biblioteca `math` refere-se ao número de Euler e na biblioteca `constants` refere-se à carga elétrica elementar.

Tabela 5: Algumas constantes disponíveis na biblioteca `SciPy.Constants`.

Constante	Comando	Valor
π	`constants.pi`	3.141592653589793
Carga Elétrica Elementar	`constants.e`	$1.602176634 \times 10^{-19}$
Razão áurea	`constants.golden`	1.618033988749895
Constante gravitacional	`constants.g`	9.80665
Velocidade da luz	`constants.c`	299792458

3.4 Computação Simbólica

Outra biblioteca útil para o professor de matemática é a `SymPy`, que permite ao Python trabalhar como um Sistema Algébrico Computacional (CAS, do inglês Computer Algebra System), isto é, permite o trabalho com matemática simbólica. Essa biblioteca define as operações básicas dos números reais e as funções de uso comum. Além disso, é capaz de fazer manipulações algébricas como simplificações, fatorações ou expansões, resolver equações algébricas e operações do Cálculo como limites, derivadas e integrais.

Página da biblioteca SymPy

Como um exemplo, a Figura 14a mostra a soma de duas raízes quadradas fatoradas, enquanto as Figuras 14b e 14c mostram as funções `sympy.factor`, que fatora uma expressão e `sympy.expand`, que expande uma expressão.

Repare que nas Figuras 14a, 14b e 14c não utilizamos o comando `print`. Em geral, o Python exibe o valor correspondente à última linha (e apenas ele), sem a utilização do comando `print`. No caso da biblioteca `SymPy`, podemos usar a função `display` da biblioteca `IPython.display` para obtermos o

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
1 import sympy
2 a=sympy.sqrt(8)
3 b=sympy.sqrt(20)
4 a+b
```

$2\sqrt{2}+2\sqrt{5}$

(a) Somando duas raízes.

```
1 import sympy
2 a=sympy.sqrt(8)
3 b=sympy.sqrt(20)
4 sympy.factor(a+b)
```

$2\left(\sqrt{2}+\sqrt{5}\right)$

(b) Fatorando uma expressão.

```
1 import sympy
2 a=sympy.sqrt(8)
3 b=sympy.sqrt(20)
4 c=sympy.sqrt(10)
5 sympy.expand((a+b)/c)
```

$\frac{2\sqrt{5}}{5}+\sqrt{2}$

(c) Expandindo uma expressão.

Figura 14: Exemplos do uso da biblioteca `SymPy`.

estilo LaTeX. A Figura 15 mostra um exemplo e alguns comentários relativos a esses códigos são exibidos abaixo.

```
1 from sympy import *
2 from IPython.display import display
3 x,y=symbols('x y')
4 display(Eq((x+y)**2,expand((x+y)**2)))
5 display(Eq((x+y)**3,expand((x+y)**3)))
6 display(Eq((x+y)**4,expand((x+y)**4)))
```

$$(x+y)^2 = x^2+2xy+y^2$$
$$(x+y)^3 = x^3+3x^2y+3xy^2+y^3$$
$$(x+y)^4 = x^4+4x^3y+6x^2y^2+4xy^3+y^4$$

Figura 15: Usando a função `display` da biblioteca `IPython.display` para obter o estilo LaTeX e visualizar mais de uma linha simultaneamente.

```
from sympy import *
from IPython.display import display
x,y=symbols('x y')
# Comando que atribui variável simbólica ao x e y.
# Se fosse usado ''import sympy'', esse comando deveria ser
# ''x,y=sympy.symbols('x y')''.

display(Eq((x+y)**2,expand((x+y)**2)))
display(Eq((x+y)**3,expand((x+y)**3)))
display(Eq((x+y)**4,expand((x+y)**4)))
# O comando ''Eq(a,b)'' produz a equação a=b.
# Nesse caso, a=(x+y)**4 e b sua expansão.
# Lembrando que se fosse usado ''import sympy'', os comandos deveriam ser
# precedidos de ''sympy.'', ou seja,
# ''display(sympy.Eq((x+y)**4,sympy.expand((x+y)**4)))''.
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Nos comentários do exemplo anterior, vimos que o comando `Eq(a,b)` monta a equação $a = b$. Utilizamos esse comando quando desejamos expressar uma igualdade entre duas expressões simbólicas. Existe uma outro comando, bastante útil, utilizado para resolver equações simbólicas ou sistemas de equações. Essa função é a função `solve(equação,variável)`. É importante destacar que, para utilizar o comando `solve` você deve passar a equação para o formato *equação=0* e especificar em que variável a equação deve ser resolvida. Por exemplo, se quisermos resolver a equação $x + 2 = 0$, devemos digitar o seguinte comando `sympy.solve(x+2,x)`. Por outro lado, se quisermos encontrar as soluções da equação $x^2 = 9$, primeiro devemos passar essa equação para a forma padrão $x^2 - 9 = 0$ e digitar `sympy.solve(x^2-9,x)`. O código a seguir apresenta a solução da equação $x^2 = 9$ em Python.

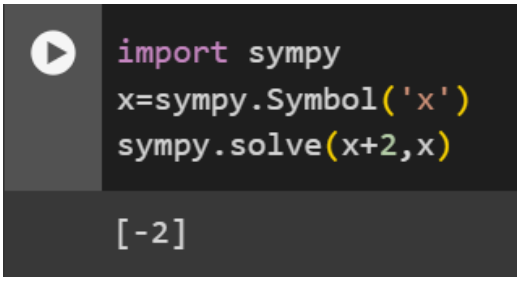
```
import sympy
x=sympy.Symbol('x')
sympy.solve(x**2-9,x)
```

A saída fornecida, após a execução do código pelo Colab é: `[-3,3]`.

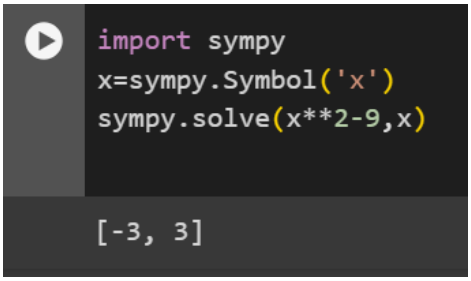
Você poderia também criar a equação através do comando `Eq` e depois resolver, conforme mostra o código seguinte.

```
import sympy
x=sympy.Symbol('x')
equacao = sympy.Eq(x**2, 9)
sympy.solve(equacao,x)
```

Apresentamos a seguir alguns exemplos de códigos em Python, bem como a saída fornecida pelo Colab, para resolver algumas equações. A Figura 16 ilustra como resolver as equações $x+2 = 0$ e x^2-9.



(a) Solução da equação $x + 2 = 0$.



(b) Solução da equação $x^2 = 9$.

Figura 16: Exemplos do uso da biblioteca `SymPy` para resolver equações.

A Figura 17 ilustra como resolver o sistema linear

$$\begin{cases} x + y = 5 \\ x - y = 3 \end{cases} \tag{3.1}$$

Observe que a solução é apresentada na forma de uma lista com cada raiz. Para visualizar cada solução separadamente, você deve acessar cada item da lista da forma usual em Python.

Existe outro comando especial, dentro da biblioteca `SymPy` para resolver sistemas lineares. No caso do SymPy, é o `linsolve`. Você pode explorar mais esse comando.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
import sympy
x,y=sympy.symbols('x y')
sympy.solve((x+y-5,x-y-3),(x,y))
```

```
{x: 4, y: 1}
```

Figura 17: Exemplo do uso da biblioteca `SymPy` para resolver o Sistema 3.1.

A biblioteca `SymPy` é uma das mais interessantes para desenvolver atividades relacionadas à álgebra. Manipulações algébricas tais como aquelas realizadas para se demonstrar identidades trigonométricas, cálculos de funções compostas e outras similares podem ser realizadas com essa biblioteca. Ela possui muitas outras funções que podem ser úteis em atividades na sala de aula, inclusive para os professores de ensino superior com comandos de Cálculo Diferencial e Integral (Figuras 18 e 19). A Figura 20 mostra, por exemplo, o comando `solveset` que exibe o conjunto solução de equações. Outras tantas funções interessantes podem ser obtidas nessa página.

```
1 from sympy import *
2 from IPython.display import display
3 x,y,dy,dx=symbols('x y dy dx')
4 f = exp(x**2)
5 display(Eq(y,f))
6 display(Eq(dy/dx,diff(f, x)))
```

$$y = e^{x^2}$$
$$\frac{dy}{dx} = 2xe^{x^2}$$

Figura 18: A função `diff` calcula a derivada de f em relação a x.

```
1 from sympy import *
2 from IPython.display import display, Math
3 x=symbols('x')
4 f = exp(-x)
5 I1=integrate(f)
6 I2=integrate(f, (x, 0, oo))
7 display(Math('f(x)=' + latex(f)))
8 display(Math('\int f(x)dx =' + latex(I1)  + '+c'))
9 display(Math('\int_0^{+\infty} f(x)dx =' + latex(I2)))
```

$$f(x) = e^{-x}$$
$$\int f(x)dx = -e^{-x} + c$$
$$\int_0^{+\infty} f(x)dx = 1$$

Figura 19: Integral da função f. A função `Math` da biblioteca `IPython.display` (juntamente com a função `latex`) permite imprimir fórmulas em LaTeX no Colab de modo centralizado.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
1 from sympy import *
2 x,a,b,c=symbols('x a b c')
3 solveset(a*x**2+b*x+c, x)
```

$$\left\{-\frac{b}{2a}-\frac{\sqrt{-4ac+b^2}}{2a}, -\frac{b}{2a}+\frac{\sqrt{-4ac+b^2}}{2a}\right\}$$

Figura 20: Resolvendo uma equação quadrática com a função `solveset`.

3.5 Computação Simbólica *versus* Computação Numérica

Um ponto importante que precisa estar claro é a diferença entre computação numérica e computação simbólica. Ambas fazem cálculos matemáticos, são extremamente importantes e normalmente se complementam. Porém, os métodos empregados e resultados obtidos, em geral, são muito diferentes.

A computação simbólica envolve a manipulação de símbolos e expressões matemáticas exatamente, sem aproximações. Ela lida com fórmulas e equações matemáticas, realizando operações de forma analítica. Inclui operações como simplificação de expressões, derivação e integração simbólica, solução exata de equações, manipulação de polinômios, e transformação de expressões. Além disso, trabalha com valores exatos e mantém a precisão matemática.

Já a computação numérica envolve a aproximação de soluções matemáticas usando métodos numéricos. Ela lida com números reais e complexos e utiliza algoritmos para obter soluções aproximadas para problemas matemáticos. Inclui operações como solução numérica de equações diferenciais, integração numérica, interpolação, otimização e simulações numéricas. Sua principal característica é trabalhar com valores aproximados devido a limitações de precisão numérica e erros de arredondamento.

Para que ambas as metodologias fiquem claras vamos compará-las com alguns exemplos simples. Para os códigos a seguir assumimos que as bibliotecas `math` e `SymPy` foram importadas

```
import math
import sympy
```

Agora podemos usar a biblioteca padrão para calcular uma raiz quadrada

```
math.sqrt(8)
```

Esse comando retorna o valor

$$2.8284271247461903$$

Efetuando o mesmo calculo simbolicamente

```
sympy.sqrt(8)
```

obtemos o resultado

$$2\sqrt{2}$$

Note que o resultado da computação simbólica é o resultado matemático correto, enquanto o resultado da computação numérica é uma aproximação. Vemos então que se desejamos realizar manipulações

matemáticas devemos usar a computação simbólica. Porém, se precisamos do valor, por exemplo, para especificar as medidas de um objeto a ser construído fisicamente, precisamos do valor numérico e precisamos usar a computação numérica.

O notebook a seguir apresenta códigos que ilustram as diferenças entre as duas técnicas.

Notebook Colab com a comparação entre `math` e `SymPy`.

Para realizarmos operações simbólicas precisamos criar variáveis simbólicas. No `sympy` usamos a função `symbols`

```
x, y = sympy.symbols("x y")
```

Essa função recebe os nomes das variáveis simbólicas que serão criadas e retorna variáveis Python que as armazenam. Não é necessário que os nomes das variáveis simbólicas e Python coincidam, mas é uma boa prática colocar nomes associados para ser possível compreender o código.

Tendo criado as variáveis simbólicas, podemos realizar operações entre elas e criar expressões simbólicas. Por exemplo, o código

```
p = x**3 + 2*x**2 - 3*x + x*y
```

cria o polinômio $p(x, y) = x^3 + 2x^2 - 3x + xy$.

Note que aqui a variável x tem o significado usado na Matemática, isto é, ela pode assumir qualquer valor numérico, inclusive derivar e integrar em relação à x. Em oposição, a variável `x` é uma variável Python, ou seja, ela aponta para uma região da memória do computador que armazena as informações necessárias para representar a variável x.

Tendo criado uma expressão simbólica, podemos realizar várias operações matemáticas nela. Por exemplo, podemos fatorar o polinômio p

```
factor(p)
```

que produz o resultado $x(x^2 + 2x - 3 + y)$. Podemos também integrar p em relação a x

```
integrate(p, x)
```

que produz o resultado $\frac{x^4}{4} + \frac{2x^3}{3} + x^2\left(\frac{y}{2} - \frac{3}{2}\right)$.

Os notebooks a seguir contém exemplos de computação simbólica. O primeiro manipula um polinômio e o segundo explora uma função de duas variáveis

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Notebook Colab usando a SymPy para manipular um polinômio.

Notebook Colab usando a SymPy para manipular uma função de duas variáveis.

3.6 Vetores e Matrizes

Outra biblioteca importante é a `NumPy`, que permite ao Python trabalhar com vetores e matrizes. Uma documentação completa sobre as funções da biblioteca `NumPy` pode ser acessada através do QR Code abaixo.

Página da biblioteca NumPy.

Um detalhe que ainda não foi mencionado é que podemos importar uma biblioteca e fornecer para ela o nome que você quiser, escrevendo `import <biblioteca> as <nome>`. A biblioteca `NumPy`, por exemplo, é geralmente chamada de `np`. Nesse caso, ao usar uma função da biblioteca, devemos usar `np.<função>`.

Vamos mostrar agora algumas funções da biblioteca `NumPy`, onde uma matriz pode ser acessada com a função `array`. A Figura 21 mostra o resultado executado no Colab de um código com as matrizes A e B definidas abaixo juntamente com as operações `A+B`, `A*B` e `A@B`, cujas definições são comentadas na própria figura.

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

Na `NumPy`, existe outra biblioteca chamada `LinAlg`[1] com diversas funções de Álgebra Linear. Por exemplo, o código abaixo imprime o determinante e a matriz inversa de A, bem como a resolução do

[1] Abreviação de *Linear Algebra*.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
1 import numpy as np
2 A = np.array([[1,2],[3,4]])
3 B = np.array([[5,6],[7,8]])
4 print('A+B=') # Soma usual de matrizes
5 print(A+B)
6 print('A*B=') # Produto termo a termo
7 print(A*B)
8 print('A@B=') # Produto usual de matrizes
9 print(A@B)
```

```
A+B=
[[ 6  8]
 [10 12]]
A*B=
[[ 5 12]
 [21 32]]
A@B=
[[19 22]
 [43 50]]
```

Figura 21: Exemplo de operações com matrizes usando a biblioteca `NumPy`.

sistema linear $Ax = b$, onde

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad \text{e} \qquad b = \begin{bmatrix} 3 \\ 7 \end{bmatrix}.$$

```
import numpy as np

A = np.array([[1,2],[3,4]])
b = np.array([[3],[7]])

print('det(A)=') # Determinante de A
print(np.linalg.det(A))

print('A^-1=') # Inversa de A
print(np.linalg.inv(A))

print('x=') # Solução do Sistema Linear Ax=b
print(np.linalg.solve(A,b))
```

Abaixo seguem algumas formas usuais de criar e operar com vetores no `NumPy`.

Listando elemento a elemento

```
x = np.array( [ 1, 2, 5, 6] )
```

Utilizando uma função

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
u = np.linspace( -5, 5, 100 ) # cria um vetor de 100 elementos
                              # igualmente espaçados entre -5 e 5

v = np.zeros( (5, 7) )        # cria uma matriz de zeros de
                              # tamanho 5x7

w = np.arange( 0, 1, 0.1)     # cria um vetor com os termos de
                              # uma progressão aritmética, cujo
                              # primeiro elemento é 0 e a razão 0.1,
                              # com último elemento menor que 1.
```

Operações com Vetores

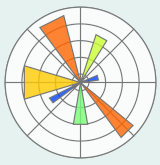
```
y = x**2 + 2*x - 3
y = array([0,  5, 32, 45])

f = np.sin(x)
f = array([0.84147098, 0.90929743, -0.95892427, -0.2794155])

g = np.exp(x)
g = array([2.71828183, 7.3890561, 148.4131591, 403.42879349])
```

O notebook a seguir contém exemplos do uso da `NumPy`.

Notebook Colab usando o NymPy para criar e manipular vetores.

3.7 Gerando Gráficos

Uma biblioteca bastante útil para se fazer gráficos é a `Matplotlib`, que dá acesso a vários tipos de gráficos tais como gráficos de linha, de dispersão, de barras, histogramas, de *pizza*, entre outros, permitindo a personalização dos mesmos. Seu pacote mais comum é o `Pyplot`. Uma documentação completa sobre as funções da biblioteca `Matplotlib` pode ser acessada através do QR Code abaixo.



Página da bibioteca Matplotlib

Vamos começar fazendo um simples gráfico de linha 2D ligando coordenadas de alguns pontos. Comumente chamamos a biblioteca `matplotlib.pyplot` pelo nome `plt`. O comando `plot` faz um

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

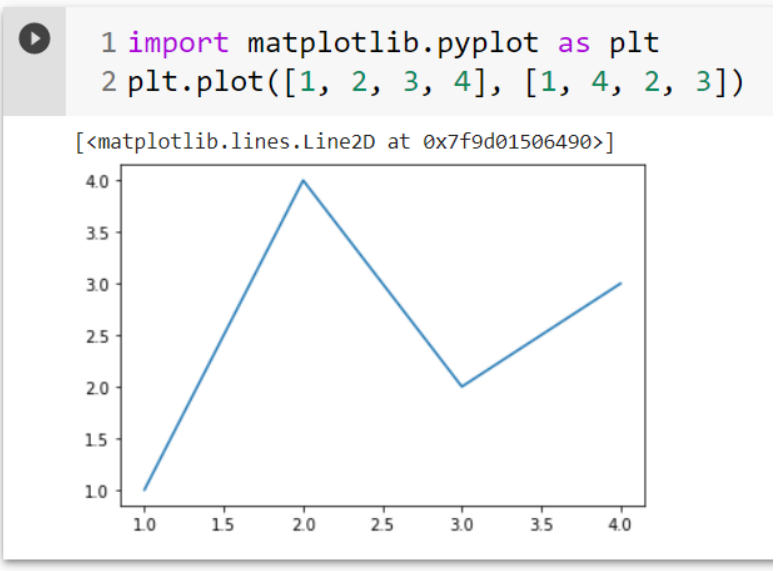


Figura 22: Gráfico de linha 2D ligando coordenadas de alguns pontos.

simples gráfico de linha 2D ligando os pontos de coordenadas $(1,1)$, $(2,4)$, $(3,2)$ e $(4,3)$, sendo a primeira lista a das abcissas e a segunda a das ordenadas.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [1, 4, 2, 3])
```

Para fazer um gráfico de uma função de uma variável, devemos importar também a biblioteca `NumPy` para criar uma lista discreta de abcissas, preferencialmente com espaçamento pequeno entre os pontos para simular a continuidade da função. A Figura 23 foi gerada com o código abaixo. Repare que existem funções em diferentes bibliotecas que retornam o mesmo resultado. Por exemplo, a função `sin` que calcula o seno existe tanto na biblioteca `NumPy` quando na `Math`.

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

# Esse comando linspace do Numpy toma 100 pontos igualmente
# espaçados no intervalo [-2pi,2pi].
x = np.linspace(-2*np.pi, 2*np.pi, 100)

# Esses comandos plotam os pontos de coordenadas
# (x,sen(x)) e (x,cos(x)).
# O label dá o nome que irá aparecer na legenda.
plt.plot(x, np.sin(x), label='Senoide')
plt.plot(x, np.cos(x), label='Cossenoide')
plt.xlabel('x')                  # nome do eixo x.
plt.ylabel('y')                  # nome do eixo y.
plt.title("Exemplo de Gráfico") # título do gráfico.
plt.legend()                     # Exibe a legenda.
plt.grid(True)                   # Exibe a malha.
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

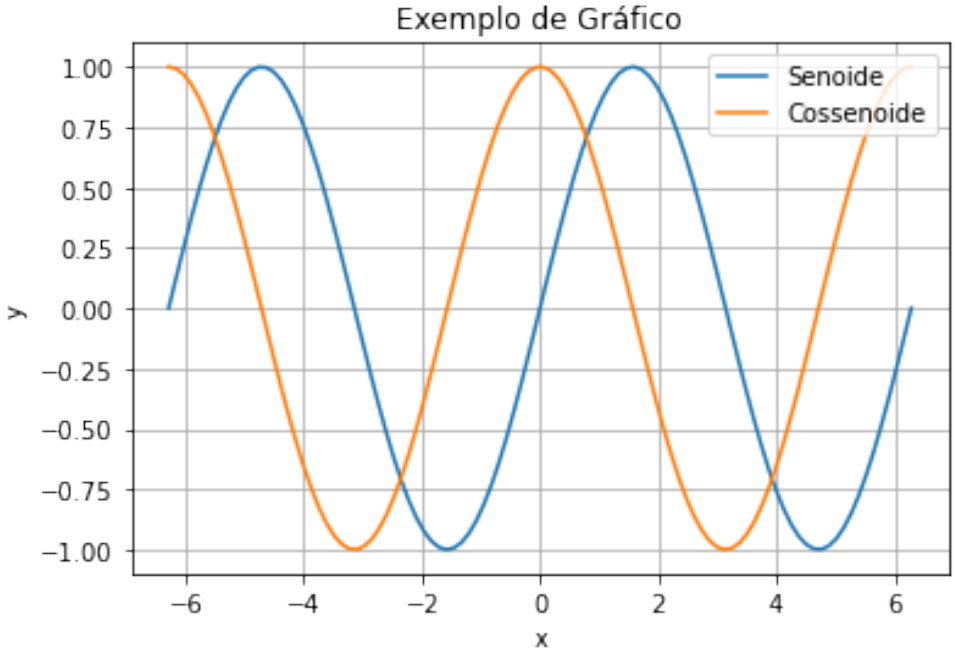


Figura 23: Gráfico das funções seno e cosseno gerado pelo Python.

No caso de gráficos 3D, devemos importar também a biblioteca `mplot3d` integrante da `mpl_toolkits`. O código comentado abaixo dá um exemplo de uma superfície de um paraboloide hiperbólico (Figura 24).

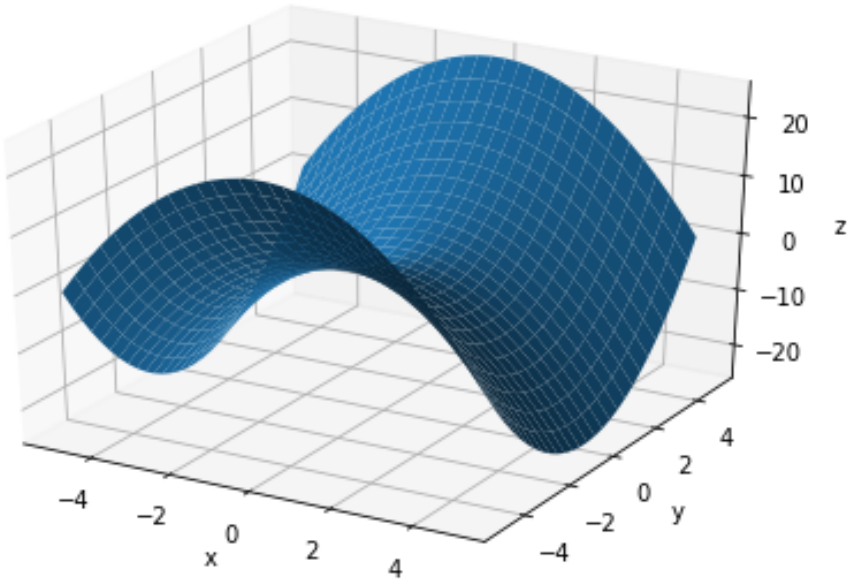


Figura 24: Paraboloide hiperbólico gerado pelo Python.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
```

```
# cria lista com 30 pontos igualmente espaçados
# no intervalo [-5,5].
x = np.linspace(-5, 5, 30)
y = np.linspace(-5, 5, 30)
X, Y = np.meshgrid(x, y) # Malha com as listas de valores (x,y)
Z = Y**2-X**2            # paraboloide hiperbólico

# As 3 linhas abaixo plotam a superfície 3D de pontos (x,y,z)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z)
ax.set_xlabel('x')  # nome do eixo x
ax.set_ylabel('y')  # nome do eixo y
ax.set_zlabel('z')  # nome do eixo z

# otimiza o layout da figura para caber os nomes dos eixos
fig.tight_layout()
```

Notebook Colab usando NymPy e MatPlotLib para desenhar o gráfico de uma função.

Notebook Colab usando NymPy e MatPlotLib para desenhar polígonos.

3.8 Atividades e Exercícios

Exercícios sobre a biblioteca matemática padrão

Exercício 3.1
Crie um código em Python que, dado um arco, em radianos, calcula o cosseno e a tangente desse arco. **Clique para ver a resposta**

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Exercício 3.2

Crie um código em Python que, dado um arco, em graus, calcula o cosseno e a tangente desse arco.

Clique para ver a resposta

Exercício 3.3

Crie um código em Python que, dado o raio de um círculo, calcula o seu comprimento e sua área.

Clique para ver a resposta

Exercícios sobre a biblioteca SymPy

Exercício 3.4

Use o SymPy para encontrar as raízes dos polinômios.

a) $p_1 = x^2 - 4$

b) $p_2 = x^2 - x - 2$

c) $p_3 = 2x^3 - 4x^2 - 10x + 12$

d) $p_4 = x^4 - x^3 - 7x^2 + x + 6$

Clique para ver a resposta

Exercício 3.5

Use o SymPy para fatorar os polinômios.

a) $p_1 = 2x^2 - 4x - 6$

b) $p_2 = x^4 - 4x^3 - 7x^2 + 22x + 24$

c) $p_3 = x^5 + x^4 - 9x^3 - 13x^2 + 8x + 12$

Clique para ver a resposta

Exercício 3.6

Use a biblioteca Sympy para completar quadrados do polinômio $x^2 - 4x + 7$. **Clique para ver a resposta**

Exercícios sobre a geração de gráficos

Exercício 3.7

Faça um programa, usando a biblioteca NumPy e a biblioteca Matplotlib, que faz o gráfico da função $y = x^3 - 3x^2 + 3x$ no intervalo $[-10, 10]$. **Clique para ver a resposta**

4

Introdução à Programação

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Os códigos exibidos no capítulo anterior são chamados sequenciais, ou seja, eles exibem resultados a partir da sequência em que são chamados. Neste capítulo, vamos apresentar como podemos organizar o código em funções que nos permitem dividir a complexidade de uma tarefa em etapas mais simples. Também iremos abordar de que forma informamos ao programa como ele deve tomar decisões e como realizar tarefas que envolvem a repetição dos mesmos passos diversas vezes.

4.1 Funções

Nesta seção, trataremos de outra estrutura importante em Python: as funções. Primeiramente vale destacar que Função na programação não é o mesmo que na Matemática.

Função na programação nada mais é que um conjunto de instruções que ajudam a organizar e modularizar o código, tornando-o mais legível, eficiente e fácil de manter. Uma função pode ou não receber parâmetros e pode ou não retornar valores. Uma característica importante desse tipo de estrutura é que podemos chamar funções em diferentes partes do código, possibilitando a reutilização da lógica contida nelas. Esse aspecto favorece a modularização do código, contribuindo para uma estrutura mais clara e organizada.

Para definir funções usamos a palavra-chave `def`. O código abaixo exibe um exemplo da sintaxe de uma função que recebe dois parâmetros e retorna um valor. Ela executa quatro comandos e depois retorna o valor calculado.

```
def nome_da_funcao( parametro1, parametro2 ):
    comando_1
    comando_2
    comando_3
    comando_4
    return valor_de_retorno
```

Observando o exemplo anterior, percebemos que existe um recuo das cinco linhas abaixo da primeira, isto é, elas não estão alinhadas à esquerda em relação à margem do alinhamento vertical da palavra-chave `def`. Chamamos isso de indentação, ou seja, espaços ou tabulações no início de uma linha de código para indicar a estrutura do programa. O Python utiliza a indentação para definir blocos de código em vez de chaves (comumente usadas em outras linguagens de programação). Isso significa que a função vai executar apenas os comandos que estiverem indentados após sua definição. A indentação pode ser criada com a tecla *Tab* ⇆ do computador ou simplesmente apertando a tecla *Enter* no final da primeira linha.

Como já mencionado, tradicionalmente o primeiro programa que um estudante de programação cria é o "Hello World!", ou "Alô Mundo!". Aqui vamos criar uma função que nos diz "Alô Mundo!", que não recebe nenhum parâmetro nem retorna nenhum valor.

```
def alo_mundo():
  print('Alô Mundo!')
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Ao digitarmos esse código em uma célula do Colab e o executarmos, aparentemente nada acontece. Na verdade, o Python criou a função e a guardou na memória. Para que ela seja executada precisamos "chamar" a função usando o nome dela.

```
alo_mundo()
```

Agora o Python busca na memória a função `alo_mundo` definida anteriormente e a executa escrevendo "Alô Mundo!" na tela.

Nosso próximo passo é escrever uma função que recebe um parâmetro. Neste caso, vamos passar um nome para a função e ela vai desejar bom dia para a pessoa cujo nome foi passado.

```
def bom_dia(nome):
  print('Bom dia,', nome, '!')
```

Novamente precisamos primeiro definir a função e depois chamá-la para que o Python a execute.

Se desejamos retornar um valor usamos o comando `return`, como já foi mostrado no primeiro exemplo dessa seção. Vamos dar outro exemplo: criar uma função com nome `soma` que recebe como parâmetros dois números `a` e `b` e retorna o valor da soma desses dois números.

```
def soma(a, b):
    resultado = a + b
    return resultado
```

Para efetuarmos uma soma precisamos chamar a função passando dois valores para ela. Por exemplo, digitando `soma(2,3)` e executando o código, obtemos o valor 5 na tela, conforme mostra a Figura 25.

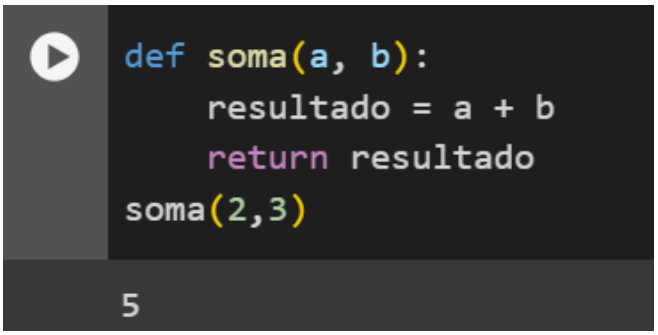


Figura 25: Função `soma` executada no Colab.

Um detalhe sobre o uso do `return` é que ele não precisa ser único e não precisa ser o último comando de uma função. Podemos usá-lo diretamente no momento em que encontramos o resultado que deve ser retornado.

Vale relembrar que não precisamos criar funções que já existem disponíveis nas bibliotecas do Python e que podemos importá-las de diversas formas. Por exemplo, com o comando `import math`, com o comando `import math as m` (usando o apelido `m`) ou com o comando `from math import sin, pi` (importando apenas funções específicas). Nesses casos, para calcular sen(π) usamos a função seno das seguintes formas, respectivamente: `math.sin(math.pi)`, `m.sin(m.pi)` e `sin(pi)`.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Como já foi mencionado, ao importarmos funções específicas com o comando `from math import sin, pi`, podemos chamar a função seno e o π sem a necessidade de prefixos.

Por fim, podemos mudar o nome da função que estamos importando. Mas essa opção deve ser usada com muito cuidado para não tornar o código ilegível depois de algum tempo. Com essa opção podemos importar a função seno e traduzir seu nome para o português

```
from math import sin as sen, pi
```

A função agora é executada com o comando

```
sen(pi)
```

O notebook do *link* a seguir apresenta alguns exemplos do uso de funções das bibliotecas, bem como de definição e criação de novas funções.

Notebook Colab com exemplos de uso e definição de funções.

4.2 Decisões

Esta seção está dedicada a tratar dos controladores de fluxo em Python. Falaremos do `if` e `else`, que funcionam permitindo que trechos de código sejam executados ou ignorados, dependendo de condições lógicas.

Os comandos if-else (*se* e *senão* em português) são operadores lógicos e funcionam como os diretores de tráfego do código, decidindo qual caminho seguir, baseado nas condições que você define. Basicamente, a estrutura de decisão é do tipo "Se p então q. Se não p, então r". Essas estruturas permitem que o programa escolha diferentes caminhos baseado em condições especificadas. Por exemplo, se quisermos testar se uma pessoa é maior de idade, podemos executar o seguinte código:

```
idade=int(input('Digite a sua idade: '))
if idade >= 18: # se a idade é maior ou igual à 18
    print("Você é maior de idade!")
else: # senão, ou seja, se a idade é menor que 18
    print("Você é menor de idade.")
```

É importante notar que tudo o que vier abaixo do `if` e `else` deve estar indentado, ou seja, só serão executados nas condições `if` e `else` os comandos indentados após suas respectivas linhas. Além disso, podemos ter múltiplas condições usando `elif` (abreviação de "else if"), conforme o código a seguir:

```
idade=int(outro exemplo('Digite a sua idade: '))
if idade < 18: # se a idade é menor que 18
    print("Você é menor de idade.")
```

```
elif idade == 18: # senão, se a idade é igual à 18
    print("Você tem exatamente 18 anos!")
else: # senão, ou seja, a idade é maior que 18
    print("Você é maior de idade.")
```

Embora o símbolo `:` no final do `if`, `else` e `elif` seja o mesmo utilizado na criação de listas e sublistas, seu significado é diferente. Aqui ele marca o início de um bloco de código indentado que depende de uma condição. No contexto de listas, ele é utilizado para delinear intervalos ou subsequências dentro de uma estrutura de dados.

O próximo código é um programa em Python que solicita ao usuário a inserção de um número inteiro e, em seguida, determina se o número é par ou ímpar. Nesse caso, o programa exibe claramente na tela se o número dado é par ou ímpar. Esse é mais um exemplo usando `if-else`. Repare que esse código pode ser utilizado pelo professor de Matemática para desenvolver a habilidade EF06MA04 da BNCC mencionada anteriormente, que diz: "Construir algoritmo em linguagem natural e representá-lo por fluxograma que indique a resolução de um problema simples (por exemplo, se um número natural qualquer é par)".

Exemplo 4.1

Faça um programa que leia um número inteiro e retorne se esse número é par ou ímpar.

```
n = int(outro exemplo('Digite o valor de n: '))
if n % 2 == 0:
  print(n, 'é par')
else:
  print(n, 'é ímpar')
```

Podemos criar uma função e inserir os controladores de fluxo para tomar decisão dentro dela. Função Tomando Decisão: verificando se um número é par

```
def par( n ):
  if n % 2 == 0:
    print( n, 'é par')
  else:
    print( n, 'não é par')
```

Usando a Função

```
par(2)
2 é par

par(3)
3 não é par
```

Outro exemplo de uma estrutura de decisão é o código abaixo, que classifica um triângulo quanto aos lados. Esse código pode ser utilizado pelo professor de Matemática para contribuir no desenvolvimento

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

da Habilidade EF06MA19 da BNCC que diz: "Identificar características dos triângulos e classificá-los em relação às medidas dos lados e dos ângulos".

Exemplo 4.2

Faça um programa que leia as medidas dos lados de um triângulo e o classifica quanto aos lados.

```
a = float(input("Digite o valor do lado a: "))
b = float(input("Digite o valor do lado b: "))
c = float(input("Digite o valor do lado c: "))

# Nesse caso está verificando se um lado é maior que a
# soma dos demais. Se isso for verdade, as 3 medidas de
# lados não formam um triângulo.
if a>=b+c or b>=a+c or c>=a+b: # or é o conectivo lógico ou
    print("Nao formam um triângulo")
elif a == b and a == c: # and é o conectivo lógico e
    print("Formam um triângulo equilátero")
elif a==b or a==c or b==c:
    print("Formam um triângulo isósceles")
else:
    print("Formam um triângulo escaleno")
```

O notebook a seguir ilustra o uso dos controladores de fluxo `if`, `elif` e `else` em processos de tomada de decisões.

Notebook Colab com exemplos do uso do `if`, `elif` e `else`.

4.3 Repetições

Quando surge a necessidade de realizar tarefas repetitivas em um código Python, não é preciso envolver-se em repetições manuais. Podemos aproveitar recursos que simplificam esse processo, transformando tarefas repetitivas em *loops* contínuos. Esses *loops* continuam a executar até que uma condição específica seja atendida

Nesta seção, abordaremos duas estruturas de repetição em Python: `for` e `while`.

4.3.1 Número fixo de repetições

O for (significa para em inglês) é uma ferramenta poderosa para automatizar tarefas repetitivas e processar dados em coleções de forma eficiente. Ele ajuda a tornar o código mais limpo e legível, especialmente em situações onde a repetição é necessária. Ele é muito útil quando você precisa iterar

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

(ou percorrer) sobre uma coleção de itens, como uma lista, tupla, *string* ou até mesmo um intervalo numérico.

A estrutura básica de um *loop* for é assim:

```
for item in colecao:
    # faça algo com o item  (deve estar indentado)
```

O `range()` é uma função bastante útil para criar sequências numéricas em Python e muitas vezes é utilizado em conjunto com a estrutura `for` para iterar sobre essa sequência. A sintaxe básica do `range()` é `range(início, final, passo)`. O valor inicial da sequência, por padrão, é 0; e o intervalo entre os valores, por padrão, é 1.

A sequência `range(10)` começa em 0 para antes de 10

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$$

A sequência `range(1, 10)` começa em 1 para antes de 10

$$1, 2, 3, 4, 5, 6, 7, 8, 9$$

A sequência `range(1, 10, 2)` começa em 1 para antes de 10 com passo 2

$$1, 3, 5, 7, 9$$

Podemos agora usar a sequência para construir uma repetição, como no código seguinte

```
for i in range(1, 6):
    print(i)
```

Esse laço de repetição `for` imprimirá os números de 1 a 5. Note que o valor final (6) não é incluído.

O programa a seguir imprime na tela a tabuada do 7.

```
for n in range(1, 11):
    v = 7 * n
    print('7 x ',n,'=',v)
```

A Figura 26 mostra o código executado no Colab .

O código a seguir mostra um exemplo não numérico. Nele listamos as vogais do alfabeto.

```
vogais = [ "a", "e", "i", "o", "u" ]
for v in vogais:
    print( v )
```

Algumas vezes precisamos de um controle mais elaborado das repetições, o Python fornece-nos dois comandos para isso. O `break` permite-nos interromper as repetições terminando prematuramente o `for`. O `continue` interrompe o ciclo atual pulando todas as instruções ainda não executadas e iniciando diretamente o próximo ciclo.

O notebook a seguir apresenta alguns exemplos do uso da instrução `for`.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
1 for n in range(1, 11):
2     v = 7 * n
3     print('7 x ',n,'=',v)
```

```
7 x  1 = 7
7 x  2 = 14
7 x  3 = 21
7 x  4 = 28
7 x  5 = 35
7 x  6 = 42
7 x  7 = 49
7 x  8 = 56
7 x  9 = 63
7 x  10 = 70
```

Figura 26: Código que imprime na tela a tabuada do 7.

Notebook Colab com exemplos de repetição com o uso do comando `for`.

4.3.2 Repetir até atingir o objetivo

O `while` (significa enquanto em inglês) é outra estrutura de controle de fluxo em Python, mas ao contrário do for, ele não depende explicitamente de uma sequência de elementos. Em vez disso, ele executa um bloco de código repetidamente enquanto uma condição específica estipulada é verdadeira.

A estrutura básica de um *loop* `while` é o seguinte:

```
while condicao:
    # faça algo enquanto a condição for verdadeira (indentado)
```

O bloco de código escrito dentro do `while` é repetido enquanto a condição fornecida for avaliada como verdadeira. Se a condição se tornar falsa, a execução do código dentro do `while` é interrompida, e o programa continua com o próximo bloco de código após o `while`.

O código a seguir imprimirá os números de 0 a 4. O contador é incrementado a cada iteração, e o *loop* continua enquanto o contador for menor que 5.

```
contador = 0
while contador < 5:
    print(contador)  #indentado
    contador += 1
```

A cada iteração, o contador é atualizado através do comando `contador += 1`, que é equivalente a `contador = contador + 1`, sendo apenas uma forma mais concisa de escrever a mesma operação.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

A seguir vemos um exemplo do cálculo de fatoriais.

Exemplo 4.3

Crie uma função para calcular $n!$ de duas formas: uma usando `for` e outra usando `while` Usando `for`

```
def fatorial(n):
    f = 1
    for k in range( 1, n+1 ):
        f = f * k
    return f
```

Usando `while`

```
def fatorial(n):
    f = 1
    while n > 1:
        f = f * n
        n = n - 1
    return f
```

Notebook Colab com exemplos de repetição com o uso do comando `while`.

4.4 Atividades de Experimentação com Python

Através de atividades de experimentação e investigação, os estudantes podem criar conjecturas para identificar propriedades e relações entre objetos matemáticos que estão sendo estudados em sala de aula.

Uma possibilidade é verificar a validade do Binômio de Newton

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^{n-i}\, y^i$$

para diferentes valores dos expoentes. O código abaixo calcula o Binômio de Newton de 2 até 9. A Figura 27 mostra o resultado do código no Colab.

```
from sympy import *
from IPython.display import display

x, y = symbols('x y')

for n in range(2,10):
```

```
    a = (x+y)**n
    b = expand((x+y)**n)
    display(Eq(a,b))
```

```
1 from sympy import *
2 from IPython.display import display
3
4 x,y=symbols('x y')
5
6 for n in range(2,10):
7     a=(x+y)**n
8     b=expand((x+y)**n)
9     display(Eq(a,b))
```

$$(x+y)^2 = x^2 + 2xy + y^2$$
$$(x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$$
$$(x+y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$$
$$(x+y)^5 = x^5 + 5x^4y + 10x^3y^2 + 10x^2y^3 + 5xy^4 + y^5$$
$$(x+y)^6 = x^6 + 6x^5y + 15x^4y^2 + 20x^3y^3 + 15x^2y^4 + 6xy^5 + y^6$$
$$(x+y)^7 = x^7 + 7x^6y + 21x^5y^2 + 35x^4y^3 + 35x^3y^4 + 21x^2y^5 + 7xy^6 + y^7$$
$$(x+y)^8 = x^8 + 8x^7y + 28x^6y^2 + 56x^5y^3 + 70x^4y^4 + 56x^3y^5 + 28x^2y^6 + 8xy^7 + y^8$$
$$(x+y)^9 = x^9 + 9x^8y + 36x^7y^2 + 84x^6y^3 + 126x^5y^4 + 126x^4y^5 + 84x^3y^6 + 36x^2y^7 + 9xy^8 + y^9$$

Figura 27: Binômio de Newton com expoentes variando 2 até 9 no Colab.

Outra possibilidade está na análise combinatória e probabilidade. Nesse caso, deve-se importar a biblioteca `Itertools`. O código abaixo, por exemplo, exibe todas as permutações com os algarismos 1, 2, 3 e 4.

```
import itertools
p = itertools.permutations([1, 2, 3, 4])
j = 1
for i in list(p):
    print(j,i)
    j += 1
```

Ao inserir o parâmetro 2 na 2ª linha do código acima da seguinte forma

```
p = itertools.permutations([1, 2, 3, 4],2)
```

o algoritmo irá imprimir todos os arranjos de 2 elementos formados com os números de 1 a 4. Trocando a função `permutations` por `combinations` da seguinte forma

```
p = itertools.combinations([1, 2, 3, 4],2)
```

o algoritmo irá imprimir todos as combinações de 1 a 4 tomados 2 a 2. Mais informações da biblioteca `Itertools` podem ser encontradas nesta página.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Outros exemplos de códigos para atividades de experimentação são apresentados a seguir:

Exemplo 4.4

Se uma moeda é lançada 5 vezes, qual a probabilidade de sair "cara" 3 vezes?

```
import itertools

# Calcula todos os resultados possiveis de 5 lancamentos
# de cara (C) e coroa (K)
p = itertools.product('KC', repeat=5)

j = 1
k = 1

for i in list(p):
    if i.count('C')==3: # se a lista tiver 3 caras
        print(j,i,'Sim','({})'.format(k))
        k+=1
    else:
        print(j,i)
    j+=1
```

Exemplo 4.5

(OBMEP-2019) Em uma caixa há cinco bolas idênticas, com as letras O, B, M, E e P. Em uma segunda caixa há três bolas idênticas, com as letras O, B e M. Uma bola é sorteada da primeira caixa e, a seguir, outra bola é sorteada da segunda caixa. Qual é a probabilidade de que essas bolas tenham a mesma letra?

```
C1=['O', 'B', 'M', 'E', 'P']
C2=['O', 'B', 'M']

k = 1
t = 1

for i in range(len(C1)):
    for j in range(len(C2)):
        if C1[i] == C2[j]:
            print(t, C1[i], C2[j], 'Sim ({})'.format(k))
            k+=1
        else:
            print(t,C1[i],C2[j])
        t += 1
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Exemplo 4.6

Em uma experiência aleatória foi lançado duas vezes um dado. Considerando que o dado é honesto, qual a probabilidade de obter a soma dos lançamentos igual a 5?

```
D1 = list(range(1, 7))
D2 = list(range(1, 7))

k = 1
t = 1

for i in range(len(D1)):
    for j in range(len(D2)):
        if D1[i]+D2[j] == 5:
            print(t, D1[i], D2[j], 'Sim ({})'.format(k))
            k+=1
        else:
            print(t,D1[i],D2[j])
        t += 1
```

Exemplo 4.7

Quantos anagramas da palavra AMAR possui as vogais juntas?

```
import itertools
p = itertools.permutations(['A','M','A','R'])

f = []

for i in list(p):
    for j in range(len(i)-1):
        if (i[j]=='A') and (i[j+1]=='A') and (i not in f):
            f.append(i)
            print(''.join(i))
```

Em relação aos exemplos 4.5 e 4.6, podemos fazer sorteios para simular essa probabilidade através da biblioteca `Random`, que fornece números aleatórios. O professor de Matemática pode utilizar esses códigos em suas aulas para auxiliar no desenvolvimento das seguintes Habilidades da BNCC:

- (EF06MA30) Calcular a probabilidade de um evento aleatório, expressando-a por número racional (forma fracionária, decimal e percentual) e comparar esse número com a probabilidade obtida por meio de experimentos sucessivos.

- (EF07MA34) Planejar e realizar experimentos aleatórios ou simulações que envolvem cálculo de probabilidades ou estimativas por meio de frequência de ocorrências.

- (EF09MA20) Reconhecer, em experimentos aleatórios, eventos independentes e dependentes e calcular a probabilidade de sua ocorrência, nos dois casos.

Os códigos abaixo simulam experimentos aleatórios para os exemplos 4.5 e 4.6.

```
import random
C1 = ['O', 'B', 'M', 'E', 'P']
C2 = ['O', 'B', 'M']
N = 100000
i = 0
j = 0
while (i < N):
    if random.choice(C1) == random.choice(C2):
        j+=1
    i+=1
probabilidade=j/i
print(probabilidade)
```

```
import random
D1=list(range(1, 7))
D2=list(range(1, 7))
N = 100000
i = 0
j = 0
while (i < N):
    if random.choice(D1)+random.choice(D2)==5:
        j+=1
    i+=1
probabilidade=j/i
print(probabilidade)
```

4.5 Atividades e Exercícios

Exercícios sobre definição de funções

Exercício 4.1
Escreva uma função em Python que retorne a média aritmética de dois números. **Clique para ver a resposta**

Exercícios sobre tomada de decisão

Exercício 4.2
Faça um programa que solicita um número para o usuário e retorna se o número é positivo, negativo ou nulo. **Clique para ver a resposta**

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Exercício 4.3 (GUANABARA)
Desenvolva um programa em Python que leia o peso e a altura de uma pessoa, calcule seu Índice de Massa Corporal (IMC)

$$\text{IMC} = \frac{\text{peso}}{\text{altura}^2}$$

e mostre sua condição, conforme as informações da tabela

IMC	Condição
Abaixo de 18,5	Abaixo do Peso
Entre 18,5 e 25	Peso Ideal
25 até 30	Sobrepeso
30 até 40	Obesidade
Acima de 40	Obesidade Mórbida

Clique para ver a resposta

Exercício 4.4
Crie uma função que, dados os coeficientes, calcule as raízes de uma equação de grau 2. **Clique para ver a resposta**

Exercícios sobre repetições

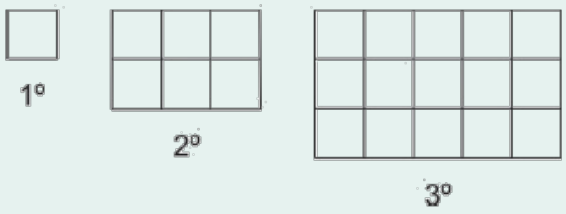
Exercício 4.5
Desenvolva um programa no qual o usuário fornece o primeiro termo, a razão de uma PA e o número de termos. No final, esse programa deve mostrar os n termos da progressão e sua soma. **Clique para ver a resposta**

Exercício 4.6
Desenvolva um programa no qual o usuário fornece o primeiro termo, a razão de uma progressão geométrica e o número de termos. No final, o programa mostra os n termos da progressão e sua soma. **Clique para ver a resposta**

Exercício 4.7
Faça um programa que, dados os catetos, forneça o valor da hipotenusa de um triângulo retângulo. Nesse programa você deve testar se os lados do triângulo retângulo dado são positivos. Caso os valores dos catetos forem negativos, mostrar na tela a mensagem "Os valores dos catetos devem ser positivos". **Clique para ver a resposta**

Exercício 4.8 (OBMEP 2008 ADAPTADA)
Com quadradinhos de lado 1 cm, constrói-se uma sequência de retângulos acrescentando-se, a cada

etapa, uma linha e duas colunas ao retângulo anterior. A figura a seguir mostra os três primeiros retângulos dessa sequência. Qual é o perímetro do 100^{o} retângulo dessa sequência?



Faça um programa em Python que resolva esse problema da OBMEP, ou seja, um programa que imprima o perímetro desse retângulo. **Clique para ver a resposta**

Exercício 4.9

Faça um programa que imprima na tela os n primeiros números da sequência de Fibonacci. **Clique para ver a resposta**

Exercício 4.10

Crie um algoritmo em Python para verificar se uma lista de números não nulos é uma PG e, se for, imprimir na tela a razão. **Clique para ver a resposta**

Exercício 4.11

Crie um algoritmo e experimente a seguinte proposição: Em uma PG finita de n termos, sendo S a soma dos termos, S' a soma de seus inversos e P o produto dos elementos, temos que

$$P^2 = \left(\frac{S}{S'}\right)^n.$$

Você acha que essa proposição é verdadeira ou falsa? **Clique para ver a resposta**

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

5

Simulando Logo em Python

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Logo (ou SuperLogo) é uma linguagem criada por Seymour Papert voltada para o ensino de programação dentro do ambiente escolar e que fez muito sucesso como ferramenta de apoio ao ensino de Matemática entre as décadas de 70 e 90. Basicamente, ela implementa uma tartaruga que responde aos comandos do usuário para se movimentar, construindo figuras com sua trajetória.

5.1 A Biblioteca Tartaruga

Vamos agora mostrar como simular o Logo no Python. Para isso, será necessário instalar a biblioteca Tartaruga no Colab através do recurso de instalação `pip`. Digite

```
!pip install ColabTurtlePlus
```

no Colab e depois execute. Quando aparecer a mensagem "Successfully installed ColabTurtlePlus", pronto. Você agora pode utilizar os recursos do Logo no Python, simulando a famosa tartaruga no Colab.

5.1.1 Construindo Polígonos

Vamos começar com um código simples: fazer a tartaruga desenhar um triângulo equilátero com sua trajetória. O professor de Matemática pode utilizar esse código em suas aulas para auxiliar no desenvolvimento da Habilidade EF07MA28 da BNCC, que diz: "Descrever, por escrito e por meio de um fluxograma, um algoritmo para a construção de um polígono regular (como quadrado e triângulo equilátero), conhecida a medida de seu lado".

Mesmo após instalada, primeiramente devemos importar a biblioteca denominada `ColabTurtlePlus.Turtle`, geralmente como a letra `t` (de tartaruga). Vamos criar uma função chamada `caminho(l,a,t)`, que irá determinar o caminho da tartaruga `t` com lado `l` e ângulo de virada `a`. Observe que, nesse caso, a estrutura de definição da função não tem o `return`, pois essa função não retorna nenhum valor (apenas executa comandos). O código abaixo contém os demais comentários dessa primeira simulação do Logo.

Exemplo 5.1

Crie um código para fazer a tartaruga desenhar um triângulo equilátero com sua trajetória.

```
import ColabTurtlePlus.Turtle as t

def caminho(l,a,t):
    t.forward(l)   # tartaruga anda pra frente "l" unidades
    t.left(a)      # tartaruga vira para esquerda "a" graus

t.clearscreen()    # limpa a área a ser desenhada
t.setup(800,500)   # define o tamanho do retângulo
t.showborder()     # mostrar a borda do retângulo
t.speed(5)         # velocidade da tartaruga
t.shape("turtle")  # formato de tartaruga
t.color("white")   # cor da tartaruga
t.bgcolor("black") # cor de fundo
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
t.pensize(5)        # largura do traço

l=100               # tamanho do lado do triângulo equilátero
a=120               # ângulo externo do triângulo equilátero

for _ in range(3): # percorre os 3 lados do triângulo
    caminho(l,a,t)
```

Uma observação sobre o código do Exemplo 5.1 está no comando

```
for _ in range(3):
```

Essa instrução tem a finalidade de repetir o caminho três vezes para percorrer o triângulo. Entretanto, o caminho não foi definido em função de uma variável de iteração, tradicionalmente identificada pelas letras `i`, `j`, `k` ou `n`. Nesse caso, podemos simplesmente usar o caractere *underline* (_), que teremos o efeito desejado, sem a necessidade de uma variável adicional que não será utilizada no *loop*.

Podemos avançar e construir um programa para fazer a tartaruga desenhar um polígono regular qualquer com sua trajetória. Essa implementação do algoritmo pode ser usada pelo professor de Matemática para auxiliar no desenvolvimento da Habilidade EF09MA15 da BNCC, que diz: "Descrever, por escrito e por meio de um fluxograma, um algoritmo para a construção de um polígono regular cuja medida do lado é conhecida, utilizando régua e compasso, como também *softwares*".

Para melhor aproveitar a área do retângulo 800×500, podemos definir a posição inicial da tartaruga em um círculo de diâmetro 450. O código exibido abaixo contém alguns comentários sobre o programa sugerido. A Figura 28 mostra os dois polígonos desenhados como resultados desses códigos executados no Colab.

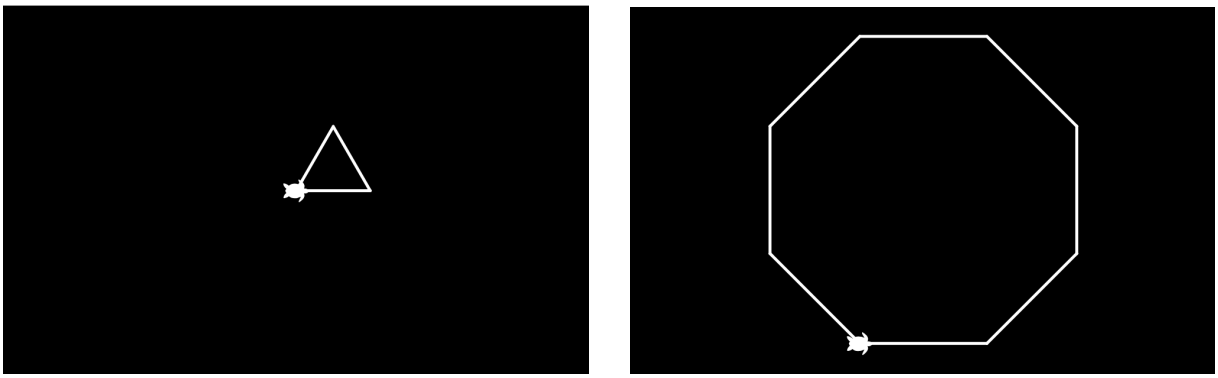
Exemplo 5.2

Crie um programa que faça a tartaruga desenhar um polígono regular de n lados com sua trajetória.

```
import ColabTurtlePlus.Turtle as t
import math

def caminho(l,a,t):
    t.forward(l)
    t.left(a)

n=8 # definindo 8 lados para o polígono
l=450*math.sin(math.pi/n) # lado de um poligono regular
                          # inscrito em um circulo de
                          # diametro 450
ai=180*(n-2)/n  # angulo interno

t.clearscreen()
t.setup(800,500)
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
t.showborder()


t.speed(5)
t.shape("turtle")
t.color("white")
t.bgcolor("black")
t.pensize(5)

t.up() # levanta a tartaruga para não desenhar até chegar
       # ao ponto de partida

# Os 3 comandos a seguir posicionam o polígono de tal
# forma que seu lado mais baixo fique na horizontal
t.right(90+180/n)
t.forward(450/2)
t.left(90+180/n)

t.down()  # baixa a tartaruga para comecar a desenhar
a = 180-ai # angulo externo

for _ in range(n):
    caminho(l,a,t)
```



(a) Triângulo. (b) Octógono.

Figura 28: Triângulo e Octógono desenhados no Colab com os códigos dos Exemplos 5.1 e 5.2.

5.1.2 Construindo Círculos

Também podemos fazer a tartaruga desenhar um círculo com sua trajetória e colorir regiões. Vamos construir um programa para fazer a tartaruga desenhar a bandeira do Japão. O código exibido a seguir contém comentários sobre alguns comandos do programa sugerido, e a Figura 29 mostra o resultado no Colab.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Exemplo 5.3

Crie um programa que faça a tartaruga desenhar a bandeira do Japão com sua trajetória.

```
import ColabTurtlePlus.Turtle as t

t.clearscreen()
t.setup(800,500)
t.showborder()
t.shape("turtle")
t.color("red")
t.speed(5)
t.up()
t.setposition(0,-100) # define a posição inicial da tartaruga
                      # em coordenadas, sendo a origem no
                      # centro do retângulo
t.down()
t.begin_fill() # inicia o preenchimento da região com a
               # cor desejada
t.circle(100)  # círculo de raio 100
t.end_fill()   # finaliza o preenchimento da região com a
               # cor desejada
t.hideturtle() # oculta a tartaruga para não aparecer no
               # desenho final
```

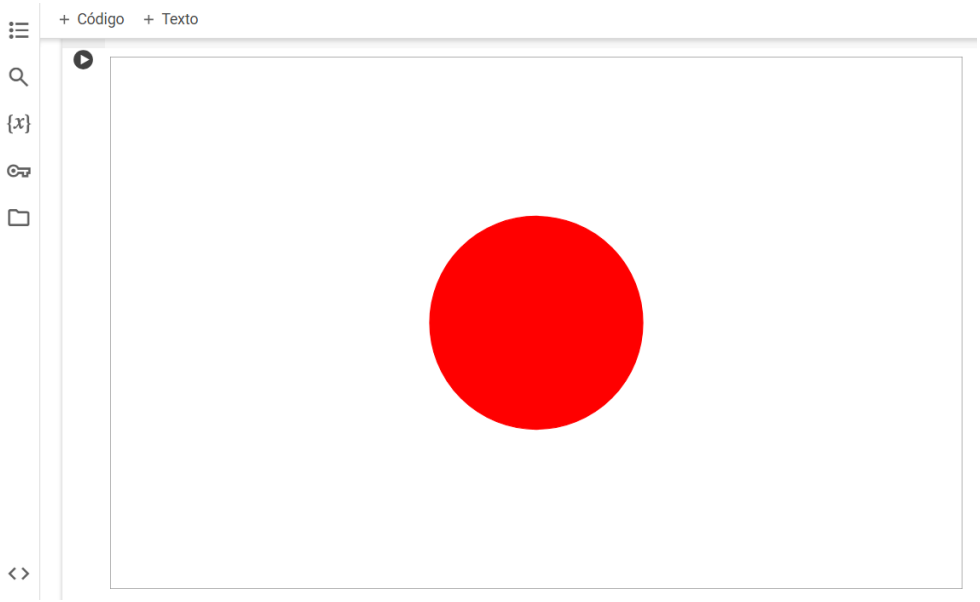


Figura 29: Bandeira do Japão desenhada no Colab com o código do exemplo 5.3.

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

5.1.3 Construindo Fractais

Uma das aplicações mais interessantes da tartaruga do Logo é o desenho de fractais. Fractais são estruturas geométricas que possuem autossimilaridade, isto é, suas partes repetem os traços do todo em infinitas escalas. Existem diversos exemplos de fractais, muitos deles encontrados na natureza. Vamos dar dois exemplos aqui: a árvore fractal e a curva de Koch.

Começaremos com a árvore fractal, cujos galhos e suas ramificações possuem autossimilaridade com o tronco principal. Para implementar fractais, usaremos funções recursivas, isto é, funções que chamam a si mesmas. A propriedade de recursividade nos fractais é exatamente a autossimilaridade. Por exemplo, em um galho da árvore pode ser observada uma réplica em miniatura semelhante ao todo. Segue um exemplo de uma possibilidade de código para a construção de uma árvore fractal.

Exemplo 5.4

Crie um programa que faça a tartaruga desenhar uma árvore fractal, com sua trajetória.

```
import ColabTurtlePlus.Turtle as t

def arvore(l,t):
    if l > 5:
        t.forward(l)
        t.right(20)
        arvore(l-10,t) # recursividade
        t.left(40)
        arvore(l-10,t) # recursividade
        t.right(20)
        t.backward(l)

t.clearscreen()
t.setup(800,500)
t.showborder()
t.shape("turtle")
t.color("green")
t.bgcolor("yellow")
t.speed(10)
t.pensize(5)

t.left(90) # virando a tartaruga para cima, pois ela sempre
           # começa virada para a direita


t.up()
t.backward(150)
t.down()

arvore(75,t)
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Outro fractal que mostraremos aqui é a curva de Koch, de autoria do matemático sueco Helge von Koch. Trata-se de um fractal que começa com uma linha reta dividida em três pedaços iguais. Em seguida, traça-se um triângulo equilátero que tem como base o terço central. Removendo essa base, obtemos uma linha composta por 4 segmentos, cada qual com ⅓ do comprimento inicial. Fazendo isso recursivamente (para cada um desses 4 segmentos), obtemos a curva de Koch. Tomando um triângulo equilátero e fazendo a curva de Koch em cada um dos seus lados, obtemos o famoso fractal do floco de neve de Koch. Abaixo temos uma possibilidade de código para a construção desse floco de neve. A Figura 30 mostra os dois fractais desenhados como resultados desses códigos executados no Colab.

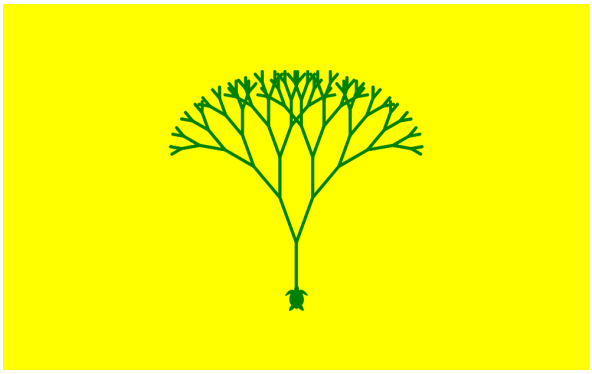
Exemplo 5.5

Crie um programa que faça a tartaruga desenhar o floco de neve de Koch com sua trajetória.

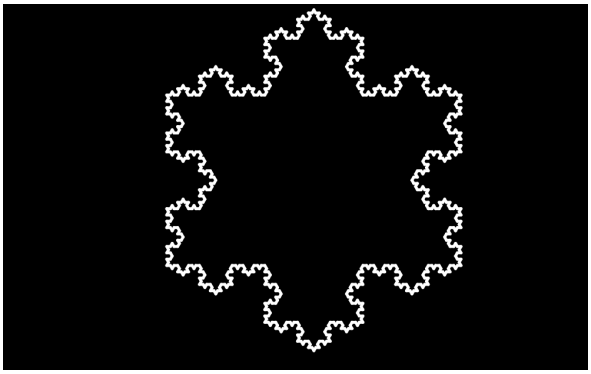
```
import ColabTurtlePlus.Turtle as t

def koch(t, n, l):
  if n == 0:
    t.forward(l)
  else:
    n = n - 1
    l = l / 3
    koch(t, n, l)
    t.left(60)
    koch(t, n, l)
    t.right(120)
    koch(t, n, l)
    t.left(60)
    koch(t, n, l)

t.clearscreen()
t.setup(800,500)
t.showborder()
t.shape("turtle")
```

```
t.color("white")
t.bgcolor("black")
t.speed(10)
t.pensize(5)

t.up()
t.setposition(-200,125)
t.down()
t.hideturtle()

n = 4 # número de repetições para a curva de Koch
l = 400
```

```
for _ in range(3):
    koch(t, n, l)
    t.right(120)
```



(a) Árvore fractal.



(b) Floco de neve de Koch.

Figura 30: Árvore fractal e floco de neve de Koch desenhados no Colab com os códigos dos Exemplos 5.4 e 5.5.

Uma documentação com outras opções de funções e parâmetros da biblioteca `ColabTurtlePlus` pode ser acessada nessa página.

O notebook a seguir traz os códigos apresentados neste capítulo.

Notebook Colab com as atividades com o Logo.

5.2 Atividades e Exercícios

Exercício 5.1

Modifique o código do Exemplo 5.1 para que a tartaruga desenhe um quadrado com sua trajetória. **Clique para ver a resposta**

Exercício 5.2

Modifique o código do Exemplo 5.2 para que a tartaruga desenhe 6 polígonos regulares de 3 até 9 lados com sua trajetória, todos inscritos na mesma circunferência e cada um de uma cor diferente, bem como a própria circunferência **Clique para ver a resposta**

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Exercício 5.3
Crie um programa que faça a tartaruga desenhar os anéis olímpicos, símbolo dos Jogos Olímpicos, com sua trajetória. **Clique para ver a resposta**

Exercício 5.4
Crie um programa que faça a tartaruga desenhar a bandeira da França com sua trajetória. **Clique para ver a resposta**

Exercício 5.5
Crie um programa que faça a tartaruga desenhar o fractal Triângulo de Sierpinski com sua trajetória. **Clique para ver a resposta**

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

6

Considerações Finais

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

O objetivo deste trabalho foi mostrar algumas possibilidades de uso da linguagem de programação Python por professores de Matemática em sala de aula. Vale lembrar que programação e Tecnologias Digitais da Comunicação e Informação (TDIC) têm sido tendência nos últimos anos na sala de aula de Matemática, além de recomendada pela Base Nacional Comum Curricular. Também destacamos que os livros de Matemática do Programa Nacional do Livro Didático (PNLD 2021) trazem introdução à programação, algoritmos e fluxogramas, associadas ao pensamento computacional.

Como já mencionado, não foi intenção desse texto dar um curso completo de programação em Python. Para isso, o leitor interessado pode se valer dos vários livros e materiais que já se encontram disponíveis, muitos deles gratuitamente na internet. Nosso foco aqui foi auxiliar o professor de Matemática, tanto em formação inicial quanto continuada, a superar os desafios de aprender a linguagem de programação Python de tal forma que ela possa auxiliar seu trabalho em suas práticas com TDIC.

Esta apostila utilizou os notebooks do Google Colaboratory diretamente no seu navegador de internet. Outra opção popular de notebook *online* para Python é o Jupyter, encontrado em <https://jupyter.org>. O leitor interessado também pode instalar alguma IDE em seu computador, como, por exemplo, Pycharm, IDLE ou Spyder.

O Python tem na Ciência de Dados um de seus maiores triunfos, principalmente por sua capacidade de fazer análises de grandes volumes de dados (Big Data). Uma das principais bibliotecas de análise de dados é a Pandas. O leitor interessado em aprofundar seus conhecimentos em sua aplicação na Ciência de Dados pode acessar essa biblioteca em <https://pandas.pydata.org>.

Atualmente, Inteligencias Artificiais (IAs) tais como o ChatGPT, Copilot, Gemini, além da IA integrada ao próprio Colab, entre outras opções, estão sendo muito utilizadas para fornecer e melhorar códigos de programação e podem ser uma grande aliada na aprendizagem e no aprimoramento em Python.

No que diz respeito à programação de computadores e ao pensamento computacional para aulas de Matemática, seguem algumas sugestões de materiais disponíveis que podem servir como excelentes referências para professores interessados em integrar esses recursos em suas aulas. Um deles é o módulo de Pensamento Computacional do projeto Livro Aberto de Matemática, que propõe diversos problemas matemáticos e mostra o processo de resolução através de diferentes representações tais como textuais, visuais e na linguagem de programação Portugol. Conteúdos complementares do livro Pensamento Computacional da coleção Livro Aberto de Matemática podem ser acessados nesse site. Há outras atividades também que, embora não envolvam necessariamente programação e sejam mais voltadas para grandes conceitos e problemas da computação, podem ser encontradas no *site* do projeto Computação Desplugada. Tal *site* oferece uma coleção gratuita de atividades para a Educação Básica, abordando conceitos e problemas da computação sem a necessidade de utilizar computadores ou equipamentos eletrônicos.

Por fim, esse material foi desenvolvido para professores que estão iniciando seus estudos em programação com Python, com foco na sala de aula de Matemática. Ao longo dele pensamos em algoritmos que pudessem ser feitos com e por estudantes programando em Python, permitindo ao professor trabalhar o pensamento computacional enquanto se discute Matemática. Esperamos que esse texto possa ajudar o leitor nessa jornada de ensinar Matemática através da programação com Python, deixando o professor mais seguro e confortável ao incorporá-lo em suas práticas docente. Fica aqui o convite para o professor interessado experimentar essas e outras atividades de programação no contexto da sala de aula.

Respostas aos Exercícios

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Capítulo 2

2.1

a)

```
10 // 3
```

b)

```
10 % 3
```

2.2

a)

```
3 + 7 ** 2
```

b)

```
( 5 - 2 * 7**3 ) / ( 5 + 3**(1/2) )
```

2.3 O resultado obtido é $30 = 5 \times \frac{4}{2} \times 3$

2.4 O programa a seguir solicita que o usuário digite um valor, converte o texto que o usuário digitar em um número, faz a conversão e escreve uma frase respondendo qual o valor em graus Célsius.

```
F = float(input('Entre com o valor de F \n'))
C = (5 * (F-32)) / 9
print(F, 'graus Fah equivale a ',C, 'graus Celsius')
```

Capítulo 3

3.1 Aqui podemos usar a biblioteca matemática padrão e realizar os cálculos diretamente.

```
import math

a = float(input('Digite o valor do arco em radiano '))

print('cos(a) =', math.cos(a))
print('tg(a)  =', math.tan(a))
```

3.2 A solução é a mesma do exercício anterior, só precisamos converter a medida do arco para radianos.

```
import math

g = float(input('Digite o valor do arco em graus '))
a = math.radians(g)
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
print('cos(a) =', math.cos(a))
print('tg(a)  =', math.tan(a))
```

3.3 Vamos usar a biblioteca matemática padrão para nos fornecer o valor de π.

```
import math

r = float(input('Digite o valor do raio '))

print('Comprimento =', 2 * math.pi * r)
print('Área        =', math.pi * r**2)
```

3.4 Em cada item o código importa a biblioteca SymPy, cria a variável simbólica x, define o polinômio e usa a função `solve` para calcular as raízes.

a)

```
import sympy as s
x = s.symbols('x')
p1 = x**2 - 4
s.solve(p1)
```

b)

```
import sympy as s
x = s.symbols('x')
p2 = x**2 - x - 2
s.solve(p2)
```

c)

```
import sympy as s
x = s.symbols('x')
p3 = 2*x**3 - 4*x**2 - 10*x + 12
s.solve(p3)
```

d)

```
import sympy as s
x = s.symbols('x')
p4 = x**4 - x**3 - 7*x**2 + x + 6
s.solve(p4)
```

3.5 Em cada caso, vamos definir o polinômio como uma expressão simbólica e usaremos a função `factor` para obter a forma fatorada.

a)

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
import sympy as s
x = s.symbols('x')
p1 = 2*x**2 - 4*x - 6
s.factor(p1)
```

b)

```
import sympy as s
x = s.symbols('x')
p2 = x**4 - 4*x**3 - 7*x**2 + 22*x + 24
s.factor(p2)
```

c)

```
import sympy as s
x = s.symbols('x')
p3 = x**5 + x**4 - 9*x**3 - 13*x**2 + 8*x + 12
s.factor(p3)
```

3.6 Para encontrar a forma que desejamos, é preciso encontrar a e b tais que

$$x^2 - 4x + 7 = (x - a)^2 + b$$

Para resolver isso com o SymPy, vamos criar as variáveis simbólicas x, a e b, usá-las para criar as expressões $p = x^2 - 4x + 7$ e $q = (x - a)^2 + b$, criar uma equação igualando p e q, resolver a equação para encontrar os valores de a e b e finalmente substituir esses valores em q.

```
import sympy as s

x, a, b = s.symbols('x a b')

p = x**2 - 4*x + 7
q = (x - a)**2 + b

solucao = s.solve(s.Eq(p, q), (a, b))

q = q.subs(solucao)
q
```

3.7

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

x = np.linspace(-10, 10, 100)
y = x**3 - 3*x**2 + 3*x
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title("Gráfico da Função $y=x^3-3x^2+3x$")
plt.grid(True)
```

Capítulo 4

4.1 A função a seguir recebe os valores a e b e retorna sua média aritmética

```
def media(a, b):
    return ( a + b ) / 2
```

Para calcular a média de dois valores chamamos a função com o código

```
media( 3, 5 )
```

4.2 Uma solução para esse problema é:

```
numero = float(input("Digite um número: "))

if numero > 0:
   print(f'O número {numero} é positivo')
elif numero < 0:
   print(f'O número {numero} é negativo' )
else:
   print('O número é zero')
```

4.3

```
peso = float(input('Entre com o peso da pessoa em kg: '))
H    = float(input('Entre com a altura da pessoa, em m: '))

IMC = peso/(H**2)

if IMC < 18.5:
  print(f'Seu IMC é {IMC} e você está abaixo do peso')

elif 18.5 <= IMC < 25:
  print(f'Seu IMC é {IMC} e você está no peso ideal')

elif 25 <= IMC < 30:
  print(f'Seu IMC é {IMC} e você está em sobrepeso')

elif 39 <= IMC < 40:
  print(f'Seu IMC é {IMC} e você está em obesidade')

else:
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
  print(f'Seu IMC é {IMC} e você está com obesidade mórbida')
```

4.4 A função a seguir recebe os coeficientes a, b e c e calcula o delta. Se o delta for não for negativo, a equação possui duas raízes reais que são calculadas por Bhaskara. Se delta for negativo, ele é transformado em um número complexo e duas raízes complexas são calculadas.

```
import cmath
import math

def raizes(a, b, c):

    delta = math.pow(b,2) - 4*a*c

    if delta >= 0:
        x1 = (-b+math.sqrt(delta)) / 2
        x2 = (-b-math.sqrt(delta)) / 2
        return (x1, x2)

    else:
        delta = complex(delta)
        x1 = (-b+cmath.sqrt(delta)) / 2
        x2 = (-b-cmath.sqrt(delta)) / 2
        return (x1, x2)
```

Para calcular as raízes de uma equação chamamos a função com o código

```
raizes( 1, 3, 2 )
```

4.5 Existem diversas formas de realizar a mesma tarefa. Nesta solução o programa calcula cada termo pela fórmula do termo geral da PA e armazena esse valor em uma lista. No final, o primeiro *print* exibe todos os valores armazenados na lista, e o segundo, a soma desses termos.

```
a1 = float(input('Entre com o primeiro termo da PA '))
r  = float(input('Entre com a razão da PA '))
n  = int(  input('Entre com o número de termos da PA '))

lista = [] # Cria uma lista vazia

for n in range(1, n+1):
    elemento = a1 + (n - 1) * r
    lista.append(elemento) # armazena cada termo na lista

print('Os termos da PA são', lista)
print('A Soma dos termos é', sum(lista))
```

4.6 A solução deste exercício é similar à do anterior, basta substituir a fórmula do termo geral.

```
a1 = float(input('Entre com o primeiro termo da PG '))
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
q  = float(input('Entre com a razão da PG '))
n  = int(  input('Entre com o número de termos da PG '))

lista = [] # Cria uma lista vazia

for n in range(1, n+1):
    elemento = a1 * q**(n-1)
    lista.append(elemento)

print(f" A PG pedida é {lista} e a soma dos seus termos é {sum(lista)}")
```

4.7

```
b = float(input('Entre com o valor do cateto b '))
c = float(input('Entre com o valor do cateto c '))

d = c**2 + b**2
a = d**0.5

if b > 0 and c > 0:
  print('A hipotenusa desse triângulo retângulo é', a)

else:
  print('Os valores de b e c devem ser positivos')
```

4.8 Primeiro vamos criar uma função que calcula o perímetro do n-ésimo retângulo, depois imprimimos o perímetro do 100^{o} retângulo. A função recebe o número de iterações n e calcula os comprimentos dos lados somando os incrementos de cada interação, depois calcula e retorna o perímetro.

```
def perimetro_retangulo(n):
    a_1      = 1
    r_linha  = 1
    r_coluna = 2

    for n in range (1, n+1):
        a_linha  = a_1 + (n-1) * r_linha
        a_coluna = a_1 + (n-1) * r_coluna

    perimetro = 2 * (a_linha*1 + a_coluna*1)

    return perimetro

print(f"O perímetro do 100 retângulo é {perimetro_retangulo(100)}")
```

4.9 Sabemos que a sequência de Fibonacci é uma sequência numérica F_n, onde $F_1 = F_2 = 1$ e $F_n = F_{n-1} + F_{n-2}$ para $n \geq 2$. Existem diversas formas de resolver esse problema, sendo uma delas o programa a seguir:

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
n = int(input("Numero de termos da Sequência de Fibonacci: "))

f = [1, 1] # Lista dos valores da sequência de Fibonacci,
           # iniciando com 1 e 1

for i in range(2, n+1):      # para i assume os valores inteiros de 2 até n
    f.append(f[i-1]+f[i-2])  # Acrescenta um novo termo à lista com a soma
                             # dos dois anteriores

print(f'Os {n} primeiros termos da Sequência de Fibonacci são:{f}')
```

O mesmo código da sequência de Fibonacci pode ser escrito usando `while` no lugar de `for`.

```
n = int(input("Numero de termos da Sequência de Fibonacci: "))
f=[1,1]
i=2
while i<n: # enquanto o i for menor que n, faça:
    f.append(f[i-1]+f[i-2]) # adicione o próximo termo à lista f
    i+=1 # soma 1 ao i
print('Os', n, 'primeiros termos da Sequência de Fibonacci são:',f)
```

4.10 Vamos resolver este problema criando uma função que recebe uma lista, testa se cada termo é zero e se a divisão dele com seu antecessor é sempre igual. Note o uso do `return` no primeiro `if` para terminar a execução da função, uma vez que já temos a resposta para o teste. O `break` cumpre uma função similar terminando antecipadamente o `for`.

```
def teste_se_e_uma_PG(lista):

    if lista[0] == 0:
        print('A sequência não é uma PG')
        return

    q  = lista[1] / lista[0]
    pg = True

    for i in range(2, len(lista)):
        if lista[i] == 0 or lista[i]/lista[i-1] != q:
            pg = False
            break

    if pg:
        print('A sequência é uma PG de razão', q)
    else:
        print('A sequência não é uma PG')
```

Tendo definido uma função que realiza o teste podemos criar uma lista e fazer o teste.

```
L = [2, 6, 18, 54, 162]
```

```
teste_se_e_uma_PG(L)
```

4.11

```
import numpy as np
n=5
a1=2
q=3
l=[a1]
for i in range(n-1):
    l.append(l[i]*q)
inversos=[1/i for i in l]
S=sum(l)
S_inv=sum(inversos)
P=np.prod(l)
if P**2==(S/S_inv)**n:
    print('Verdadeiro')
else:
    print('Falso')
```

A proposição é verdadeira.

Capítulo 5

5.1

```
import ColabTurtlePlus.Turtle as t
def caminho(l,a,t):
    t.forward(l)
    t.left(a)
t.clearscreen()
t.setup(800,500)
t.showborder()
t.speed(5)
t.shape("turtle")
t.color("white")
t.bgcolor("black")
t.pensize(5)
l=100
a=90
for _ in range(4):
    caminho(l,a,t)
```

5.2

```
import ColabTurtlePlus.Turtle as t
import math
def caminho(l,a,t):
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

```
    t.forward(l)
    t.left(a)
t.clearscreen()
t.setup(800,500)
t.showborder()
t.speed(10)
t.shape("turtle")
t.bgcolor("black")
t.pensize(5)
cores = ["red","purple","blue","green","orange","yellow"]
n=3
for i in range(6):
    l=450*math.sin(math.pi/n)
    ai=180*(n-2)/n
    t.color(cores[i])
    t.up()
    t.right(90+180/n)
    t.forward(450/2)
    t.left(90+180/n)
    t.down()
    a=180-ai
    for _ in range(n):
        caminho(l,a,t)
    t.up()
    t.setposition(0,0)
    t.down()
    n+=1
t.up()
t.setposition(0,-225)
t.down()
t.color("white")
t.circle(225)
```

5.3

```
import ColabTurtlePlus.Turtle as t
t.clearscreen()
t.setup(800,500)
t.showborder()
t.shape("turtle")
t.speed(10)
t.pensize(5)
x=[-200,0,200,-100,100]
y=[-25,-25,-25,-150,-150]
cores = ["blue","black","red","yellow","green"]
for i in range(5):
    t.up()
```

```
    t.setposition(x[i],y[i])
    t.down()
    t.color(cores[i])
    t.circle(100)
t.hideturtle()
```

5.4

```
import ColabTurtlePlus.Turtle as t
t.clearscreen()
t.setup(800,500)
t.showborder()
t.shape("turtle")
t.speed(10)
t.up()
t.setposition(-400,-250)
t.down()
cores = ["blue","red"]
for i in range(2):
    t.color(cores[i])
    t.begin_fill()
    for j in range(4):
        if j%2 == 0:
            l=800/3
        else:
            l=500
        t.forward(l)
        t.left(90)
    t.end_fill()
    t.forward(1600/3)
t.hideturtle()
```

5.5

```
import ColabTurtlePlus.Turtle as t
def sierpinski(l,n):
    if n == 0:
        t.begin_fill()
        for _ in range(3):
            t.forward(l)
            t.left(120)
        t.end_fill()
    else:
        l=l/2
        sierpinski(l,n-1)
        t.up()
        t.forward(l)
        t.down()
```

```
        sierpinski(l,n-1)
        t.up()
        t.left(120)
        t.forward(l)
        t.right(120)
        t.down()
        sierpinski(l,n-1)
        t.up()
        t.right(120)
        t.forward(l)
        t.left(120)
        t.down()
t.clearscreen()
t.setup(800,500)
t.showborder()
t.shape("turtle")
t.speed(10)
t.up()
t.setposition(-250,-200)
t.down()
l=500
n=3
sierpinski(l,n)
```

6º Simpósio Nacional da Formação do Professor de Matemática / Rio de Janeiro - RJ / set. 2023

Referências Bibliográficas

BORBA, M. C.; PENTEADO, M. G. Informática e Educação Matemática. 6a edição. ed. Belo Horizonte: Editora Autêntica, 2019.

BRASIL. MINISTÉRIO DA EDUCAÇÃO. Base Nacional Comum Curricular. Brasília, 2018. vi

GUANABARA, G. Curso em Vídeo. Disponível em: <https://www.youtube.com/playlist?list=PLHz_AreHm4dlKP6QQCekuIPky1CiwmdI6>. 51

MARCONDES, G. A. B. Matemática com Python: Um Guia Prático. São Paulo: Novatec Editora, 2018.

PAIVA, S. Introdução à Programação e ao Pensamento Computacional Usando a Linguagem Python e Portugol Studio Univali. Rio de Janeiro: Editora Ciência Moderna, 2021.

PONTE, J. P.; BROCARDO, J.; OLIVEIRA, H. Investigações Matemáticas na Sala de Aula. 4a edição. ed. Belo Horizonte: Editora Autêntica, 2019.

SAHA, A. Doing Math with Python: Use Programming to Explore Algebra, Statistics, Calculus, and More! San Francisco, CA - USA: No Starch Press, 2015.

Índice Remissivo

Algoritmo, vi, 2, 3, 21, 29, 42, 47, 52, 54, 55, 63

Colab, vi, 3–5, 7, 9, 12, 13, 16, 18, 21–24, 28, 30, 31, 33, 36, 40, 41, 43, 45–47, 54, 56, 57, 60, 63

Fluxograma, vi, 2, 42, 54, 55, 63

Google Colaboratory, vi, 3, 4, 63

Notebook, 3–5, 9, 12, 13, 15, 16, 18, 22–24, 30, 31, 33, 36, 41, 43–46, 60, 63

6º Simpósio Nacional da Formação do Professor de Matemática

Realização e Organização

ANPMat

Associação Nacional dos Professores de Matemática na Educação Básica

Distribuição

Editora ANPMat