COLÓQUIOS DE MATEMÁTICA DAS REGIÕES

REGIÃO NORDESTE

V Colóquio de Matemática
da Região Nordeste

PYTHON PARA MATEMÁTICOS

ANDRÉA LINS E LINS SOUZA

SBM
SOCIEDADE BRASILEIRA DE MATEMÁTICA

PYTHON PARA MATEMÁTICOS

Python para matemáticos

Sociedade Brasileira de Matemática
Presidente: Paolo Piccione
Vice- Presidente: Jaqueline Godoy Mesquita

Diretores: Walcy Santos
Jorge Herbert Soares de Lira
Daniel Gonçalves
Roberto Imbuzeiro

Editor Executivo
Ronaldo Garcia

Assessor Editorial
Tiago Costa Rocha

Capa: Pablo Diego Regino

Distribuição e vendas
Sociedade Brasileira de Matemática
Estrada Dona Castorina, 110 Sala 109 - Jardim Botânico
22460-320 Rio de Janeiro RJ
Telefones: (21) 2529-5073
http://www.sbm.org.br / email:lojavirtual@sbm.org.br

ISBN (eBook) 978-85-8337-216-5

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

Souza, Andréa Lins e Lins
Python para matemáticos [livro eletrônico] :
V Colóquio de matemática da região Nordeste /
Andréa Lins e Lins Souza. -- João Pessoa, PB :
Sbm - Sociedade Brasileira de Matemática,
2023. -- (Coleção colóquios de matemática)
PDF
Bibliografia.
ISBN 978-85-8337-216-5

1. Matemática - Estudo e ensino 2. Python
(Linguagem de programação para computadores)
I. Título III. Série.

23-163421 CDD-510

Índices para catálogo sistemático:

1. Matemática 510

Aline Graziele Benitez - Bibliotecária - CRB-1/3129

COLÓQUIOS DE MATEMÁTICA DAS REGIÕES

REGIÃO NORDESTE

V Colóquio de Matemática
da Região Nordeste

PYTHON PARA MATEMÁTICOS

ANDRÉA LINS E LINS SOUZA

1ª EDIÇÃO
2023
RIO DE JANEIRO

SBM
SOCIEDADE BRASILEIRA DE MATEMÁTICA

Para o meu amado filho Henrique!

Sumário

Prefácio **v**

1 Introdução **1**

2 Configurando o Python **3**
2.1 Verificando a instalação do Python 3
2.2 Acessando o Python 4
2.2.1 Via Terminal 4
2.2.2 Via IDE 6
2.2.3 Via Smartphone 7
2.2.4 Interpretador online 7

3 Comandos e operações básicas **9**
3.1 Tipos de dados 9
3.1.1 Inteiros (*int*) 10
3.1.2 Booleano (*bool*) 10
3.1.3 Ponto flutuante (*float*) 10
3.1.4 Complexo (*complex*) 10
3.1.5 String (*str*) 11
3.1.6 Listas (*list*) 11
3.1.7 Tuplas (*tuple*) 15
3.1.8 Dicionário (*dict*) 16
3.2 Operadores Aritméticos 16
3.3 Operadores de Atribuição 19

3.4 Funções e Classes 23
3.4.1 Funções 23
3.4.2 Classes 28
3.5 Instruções compostas 30
3.5.1 If 31
3.5.2 For 35
3.5.3 While 38
3.6 Exercícios 41

4 O módulo Math **43**
4.1 Funções piso e teto 44
4.2 Logaritmo e Exponencial 45
4.3 Funções trigonométricas 46
4.4 Aplicações 47
4.4.1 Teorema de Pitágoras 47

5 O pacote Numpy **49**
5.1 Arrays 49
5.1.1 Operações básicas usando arrays 51
5.2 Matrizes 52
5.3 Aplicações 54
5.3.1 Notação científica 54
5.3.2 Calculando integrais 54
5.3.3 Solução de sistemas lineares 55

6 O pacote Matplotlib **57**

Referências **63**

Prefácio

Esse texto foi preparado para ser utilizado no V Colóquio de Matemática da Região Nordeste, promovido pela Sociedade Brasileira de Matemática (SBM), que ocorreu, de forma presencial, entre os dias 07 e 11 de Novembro de 2022 em João Pessoa-PB, Brasil, no Departamento de Matemática da Universidade Federal da Paraíba (UFPB). O evento possui como público-alvo alunos de graduação e de pós-graduação, professores da Educação Básica, professores e/ou pesquisadores do Ensino Superior na área de Matemática. No entanto, esse texto poderá ser utilizado por qualquer um que tenha interesse pelo tema.

O minicurso foi concebido com o intuito de mostrar aos alunos, professores e profissionais da área de matemáticos como é possível utilizar uma ferramenta computacional simples e poderosa para facilitar o trabalho, seja nos estudos, auxiliando no processo de ensino/aprendizagem em sala de aula, nas pesquisas ou em outras aplicações.

A linguagem de programação Python [1] foi escolhida por ser uma linguagem de fácil acesso e aprendizagem rápida, além de ser uma linguagem de alto nível, ou seja, mais próxima da linguagem humana, o que facilita bastante a sua compreensão e aplicações.

JOÃO PESSOA, NOVEMBRO de 2022

Andréa Lins e Lins Souza

1

Introdução

As linguagens de programação possibilitam ao programador escrever comandos e algoritmos para controlar o comportamento, tanto físico quanto lógico, de um computador. Para tal, utiliza-se uma série de instruções inerentes à linguagem ou conjunto de linguagens adotadas. Cada linguagem de programação tem o seu propósito, e a melhor linguagem depende do tipo de problema que se deseja resolver e das disponibilidades de hardware.

Em um contexto geral, a matemática e as linguagens de programação estão intimamente relacionadas. Os computadores operam em sistema binário e a linguagem de programação é a responsável por estabelecer a ligação entre o sistema binário e o sistema decimal, comumente utilizado na matemática. Sendo assim, a relação entre a matemática e a programação tem uma natureza essencialmente lógica e exata.

No âmbito da Educação Básica no Brasil, foi instituída em 2017 a Base Nacional Comum Curricular (BNCC) [5], a qual explicita os objetivos de aprendizagem e desenvolvimento a serem observados no aluno ao longo das etapas e suas respectivas modalidades. Composta por dez competências gerais, a BNCC traz o ensino de linguagens de programação, além do domínio de uso de algoritmos e análise de dados, como atividades da competência de número cinco, denominada *Cultura Digital*. Essa competência reconhece que a tecnologia tem papel fundamental na formação do aluno.

Sendo assim, atividades como as propostas na BNCC, que envolvem linguagens de programação e uso de algoritmos, podem e devem ser inseridas nas aulas de matemática para auxiliar no processo de ensino/aprendizagem com o intuito de melhorar o desempenho dos alunos e prepará-los ainda mais para as

demandas dos tempos atuais.

Nesse trabalho, a linguagem de programação Python [1] foi adotada para ser apresentada aos profissionais da área da matemática, por ser uma linguagem versátil, acessível e mais próxima da linguagem humana, ou seja, de alto nível. Além disso, o Python é uma linguagem interpretada, orientada a objeto, de código aberto e com semântica dinâmica.

Criado em 2009 por Guido van Rossum, em 2014 o Python já tinha se tornado uma linguagem muito popular nos cursos introdutórios de ciência da computação das principais universidades americanas [2] e, atualmente, ele encontra-se na primeira posição dentre as linguagens de programação mais utilizadas em 2022 em diversos *rankings* internacionais, como o *IEEE Spectrum's* [3] e o *Tiobe index* [4].

O Python possui diversas aplicações, como desenvolvimento web, desenvolvimento de interfaces gráficas de usuário (GUI), administração de sistemas, aplicações científicas e numéricas, ciência de dados e tudo o que há de mais recente na área tecnológica, como inteligência artificial, *big data*, internet das coisas, etc. Ele é utilizado por atuais gigantes da tecnologia como Google, Facebook, Instagram, Spotify, Netflix, Dropbox, dentre outras, além de ser muito utilizada por pesquisadores de importantes centros de pesquisas.

Nesse contexto, o objetivo principal desse trabalho é apresentar a linguagem de programação Python sob um ponto de vista matemático, para que a mesma possa ser utilizada por professores, alunos e pesquisadores dessa área ou por qualquer pessoa que esteja estudando esse tema e/ou tenha interesse no mesmo.

O texto está organizado da seguinte forma: No Capítulo 2 será feita uma breve descrição sobre como configurar e acessar o Python no computador, smartphone e online. Os comandos e operações básicas com os dados do Python serão vistos no Capítulo 3, o qual ainda traz as definições de funções, classes e instruções compostas. Os Capítulos 4, 5 e 6 apresentam, respectivamente, o módulo Math, o pacote Numpy e o pacote Matplotlib.

2

Configurando o Python

Esse capítulo descreve brevemente como configurar o Python para utilizá-lo em um computador ou smartphone.

Nesse trabalho as atividades foram desenvolvidas utilizando a linguagem de programação Python 3, bem como alguns de seus módulos e pacotes, como o Math, o Numpy e o Matplotlib. Antes de começar a utilizar o Python é preciso verificar se ele está instalado na sua máquina. Caso ele já esteja instalado, pode-se, também, verificar a versão instalada.

2.1. Verificando a instalação do Python

Os sistemas operacionais Linux e macOS dispõem de uma versão do Python pré-instalada. Usuários do macOS podem verificar a versão do Python instalada ao acessar o terminal, após navegar pelo menu Aplicativos > Utilitários > Terminal e digitar o comando:

```
python --version
```

No Linux, esse mesmo processo pode ser feito em uma janela do Terminal. Já no Windows, é possível encontrar a versão do Python por meio do prompt de comandos do próprio Windows, digitando o comando:

```
python -V
```

Caso o valor retornado seja inferior a 3 ou não retorne um valor, pode-se tentar usar a palavra `python3` ao invés de `python` nos comandos acima. Ou seja, para macOS ou Linux:

```
python3 --version
```

e no Windows:

```
python3 -V
```

Se ainda assim continuar a retornar um valor menor que 3, é preciso instalar o Python 3 na máquina. Existem diversos tutorais na internet que ensinam como instalar o Python de forma simples e eficiente. Pode-se acessar a documentação oficial do Python, disponível em [6], para verificar como instalar o Python 3 em um sistema operacional específico.

2.2. Acessando o Python

Há diversas formas de acessar o Python em um computador, seja via terminal, através de um ambiente de desenvolvimento integrado (IDE) específico para o Python, ou através de editores de código. Também é possível acessar o Python através de smartphones. A seguir, será detalhado como proceder para usar o Python de acordo com a ferramenta adotada:

2.2.1. Via Terminal

Pode-se acessar o Python via Terminal de duas formas: acessar o interpretador diretamente ou através de um arquivo com diversos comandos (ou *scripts*).

Interpretador do Python

O interpretador do Python pode ser acessado via Terminal digitando o comando `python3`, como ilustra a Figura 2.1.

Observe que, nesse sistema, ao digitar apenas `python`, ele estará utilizando o Python 2.7.17, já que essa versão também encontra-se instalada no computador. No entanto, nem todos os exemplos desse minicurso irão funcionar para essa versão do Python, principalmente devido à sintaxe de alguns comandos que diferem entre o Python 2 e o Python 3.

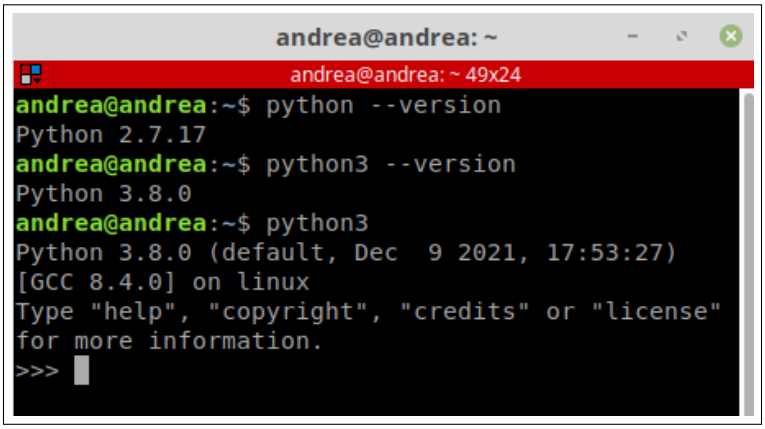


Figura 2.1: Usando o Python via Terminal no Linux.

Para sair do modo de interpretador do Python basta digitar simultaneamente a tecla `ctrl` e a letra `d`.

Arquivo `.py`

Uma outra forma de usar o Python via terminal é usar um editor de texto e salvar todos os comandos em um arquivo com extensão `.py`. Depois, pode-se acessá-lo através do terminal digitando o comando:

`python3 /endereço-do-arquivo/nome-do-arquivo.py`.

Exemplo 2.1. Considere o arquivo exemplo.py que foi armazenado no Desktop e tem os seguintes comandos:

```
1 # Importa o módulo sys
2 import sys
3 # Imprime a versão do Python instalada no sistema
4 print(f"Python {sys.version}")
```

Nesse caso, para usar o interpretador do Python 3 via terminal, eu iniciei um terminal e digitei o comando:

`python3 /home/andrea/Desktop/exemplo.py`

como mostra a Figura 2.2.

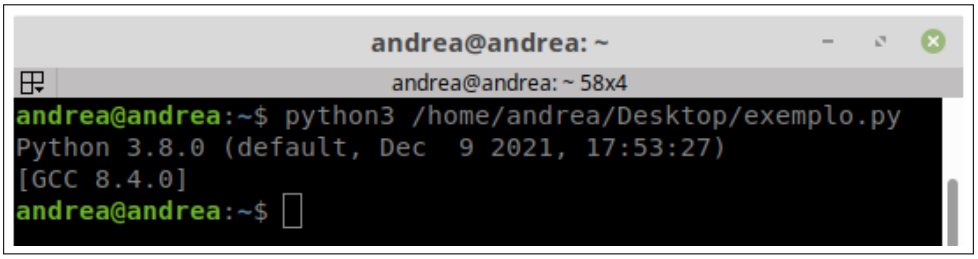


Figura 2.2: Usando o Python via Terminal no Linux através de um arquivo `exemplo.py`.

Observe que, nesse caso, o endereço do arquivo foi `/home/andrea/Desktop/`, no entanto ele pode variar de acordo com a máquina e o sistema utilizados.

Observação: Um comentário em Python é precedido pelo símbolo #. O comando precedido pelo símbolo # não será levado em consideração na execução do código. No Exemplo 2.1, as linhas 1 e 3 são exemplos de comentários.

2.2.2. Via IDE

Existem inúmeros IDEs para desenvolver programas em Python e o principal benefício de um ambiente de desenvolvimento integrado é o aumento da produtividade do desenvolvedor, pois eles reduzem o tempo de configuração do sistema, que nem sempre é uma tarefa simples; além de padronizar o processo de desenvolvimento. Como sugestão, pode-se usar a IDEs Jupyter e VS Code, mas é possível escolher a mais adequada aos objetivos.

Jupyter

Dentre os IDEs mais utilizados com o Python está o Jupyter, devido à sua versatilidade para aplicações em Ciência de Dados que utilizam grandes conjuntos de dados. Além disso, o Jupyter pode servir como uma ferramenta de demonstração, pois permite ver o resultado e editar o código ao mesmo tempo. Ele também é muito usado para fins educacionais.

Para instalar o Jupyter abra um terminal e digite o comando:

```
pip install notebook
```

Para abrir o notebook, dependendo do sistema operacional, digite um dos comandos:
`jupyter-notebook` ou `jupyter notebook`.

Para mais detalhes, consulte a documentação oficial disponível em [8].

VS Code

O Visual Studio Code é um IDE criado pela Microsoft e é muito usada por programadores de diversas linguagens de programação. Ele fornece suporte ao Python por meio da extensão Microsoft Python. Ele é altamente customizável e possui um grandioso conjunto de extensões que podem ser usadas para melhorar as funcionalidades padrão. Além das ferramentas de auto-completar e integração direta com o GIT (sistema de controle de versões).

Para saber mais sobre o VS Code acesse a documentação oficial disponível em [9].

2.2.3. Via Smartphone

Uma opção para utilizar o Python é usando o seu smartphone. Para isso, em smartphones Android deve-se instalar os aplicativos *Pydroid repository plugin* e *Pydroid 3*. Para iPhone, tem um aplicativo chamado *Juno*.

2.2.4. Interpretador online

Há diversas páginas que disponibilizam um interpretador do Python online [12, 13], como também uma ferramenta do Google, o Colab [14], o qual é muito utilizado para fins educacionais. Uma das desvantagens desse método é que requer acesso à internet de forma contínua.

3

Comandos e operações básicas

Nesse capítulo serão exibidos os tipos de dados, os comandos e operações básicas com os dados do Python através de operadores aritméticos e de atribuição. Por fim, serão definidas as funções, as classes e as instruções compostas e algumas aplicações serão apresentadas.

O Python possui uma sintaxe relativamente simples, comparado a outras linguagens de programação e é fácil de ser utilizado. No decorrer desse texto, veremos como criar códigos em Python, com algumas aplicações em matemática.

Por ser uma linguagem dinamicamente tipada, não é necessário declarar ou mudar o tipo da variável. Na seção seguinte, serão apresentados os tipos de dados do Python.

3.1. Tipos de dados

O Python possui os seguintes tipos de dados padrão:

- Inteiro (*int*)
- Booleano (*bool*)
- Ponto Flutuante ou Decimal (*float*)
- Complexo (*complex*)
- String (*str*)

- Lista (*list*)
- Tupla (*tuple*)
- Dicionário (*dict*)

Matematicamente falando, o Python possui três tipos numéricos: a) inteiros (*int* e *bool*), ponto flutuante (*float*) e complexo(*complex*), os quais serão detalhadas nas subseções a seguir:

3.1.1. Inteiros (*int*)

São os números zero, positivos e negativos sem a parte fracionária. Eles representam números em um intervalo ilimitado, sujeito apenas à memória (virtual) disponível.

Exemplo 3.1. São exemplos de números inteiros: 1, 5, -3, 4, 0, -15, ...

3.1.2. Booleano (*bool*)

Dado lógico que pode assumir apenas dois valores, falso ou verdadeiro (*False* ou *True* em Python) e na lógica computacional, podem ser considerados como 0 ou 1.

3.1.3. Ponto flutuante (*float*)

São números reais positivos e negativos, com a parte fracionária denotada pelo símbolo decimal `.` ou em notação científica `E` ou `e`.

Exemplo 3.2. Os números $1.2, 10.0, -3.55555, -79.0$, $1.23e+08$,... são exemplos de *floats*.

3.1.4. Complexo (*complex*)

São representados por $a + bj$, onde a e b são números reais. É obrigatório o uso da variável j para indicar que um número é complexo em Python.

Exemplo 3.3. Alguns exemplos de números complexos: $3 + 2j$, $5 - j$, $4j$, $1.0 + 2.5j$, ...

Além dos tipos numéricos, temos ainda dados de outros tipos, os quais serão vistos nas subseções abaixo:

3.1.5. String (*str*)

É um conjunto de caracteres dispostos numa determinada ordem, geralmente são utilizados para representar palavras, frases ou textos. Podem ser delimitados por aspas simples, três aspas simples seguidas ou aspas duplas.

Exemplo 3.4. As variáveis nome, email e telefone são do tipo *string*:

```
nome = 'Andréa Lins'
email = '''andlins@lcg.ufrj.br'''
telefone = "00009999"

# print() é um comando para exibir um conteúdo na tela
print(nome)
print(email)
print(telefone)

# Saída
Andréa Lins
andlins@lcg.ufrj.br
00009999
```

3.1.6. Listas (*list*)

Agrupam um conjunto de elementos variados, podendo conter: inteiros, *floats, strings*, outras listas e outros tipos. Elas são definidas utilizando-se colchetes [ ] para delimitá-las e vírgulas para separar os elementos.

Exemplo 3.5. No código abaixo temos dois tipos de listas, uma de nomes (*strings*) e uma de notas (*floats e ints*):

```
lista_alunos = ['Davi', 'Felipe', 'Henrique', 'Maria']
lista_notas = [9.5, 7.5, 10, 8.0]

print(lista_alunos)
print(lista_notas)

# type() é um comando usado para saber o tipo do dado
print(type(lista_alunos))
# informa o tipo do dado na primeira posição [0] da lista
print(type(lista_alunos[0]))

print(type(lista_notas))
# informa o tipo do dado na terceira posição [2] da lista
print(type(lista_notas[2]))

# Saída
['Davi', 'Felipe', 'Henrique', 'Maria']
[9.5, 7.5, 10, 8.0]
<class 'list'>
<class 'str'>
<class 'list'>
<class 'int'>
```

É possível alterar os dados existentes em uma lista, ou inserir novos dados. A seguir veremos alguns exemplos usando os comandos `append()` e `insert()`:

Exemplo 3.6. Usando o comando `append()` para acrescentar um elemento no final da lista do Exemplo 3.5:

```
# O comando append() acrescenta um elemento no final da lista
lista_alunos = ['Davi', 'Felipe', 'Henrique', 'Maria']
lista_notas = [9.5, 7.5, 10, 8.0]

new_lista_alunos = lista_alunos
new_lista_alunos.append("Laura")
print(lista_alunos)
print(new_lista_alunos)

new_lista_notas = lista_notas
new_lista_notas.append(7.0)
```

```
print(lista_notas)
print(new_lista_notas)

# Saída
['Davi', 'Felipe', 'Henrique', 'Maria', 'Laura']
['Davi', 'Felipe', 'Henrique', 'Maria', 'Laura']
[9.5, 7.5, 10, 8.0, 7.0]
[9.5, 7.5, 10, 8.0, 7.0]
```

Observe que ao acrescentar um elemento no final da lista `new_lista_alunos`, esse mesmo elemento também foi adicionado à lista `lista_alunos`. O mesmo ocorre ns lista de notas. Caso deseje que os elementos da primeira lista não sejam alterados, é preciso copiar os elementos da lista item por item.

Exemplo 3.7. Usando o comando `append()` para acrescentar cada elemento no final da lista, individualmente:

```
lista_alunos = ['Davi', 'Felipe', 'Henrique', 'Maria']
lista_notas = [9.5, 7.5, 10, 8.0]
# len() fornece a quantidade de elementos da lista
print(len(lista_alunos))
print(lista_alunos)

# Saída
4
['Davi', 'Felipe', 'Henrique', 'Maria']

# Criando a nova lista vazia
new_lista_alunos = []
# Adicionando os dados à nova lista
new_lista_alunos.append(lista_alunos[0])
new_lista_alunos.append(lista_alunos[1])
new_lista_alunos.append(lista_alunos[2])
new_lista_alunos.append(lista_alunos[3])
new_lista_alunos.append("Laura")

print(lista_alunos)
print(new_lista_alunos)

# Saída
```

```
['Davi', 'Felipe', 'Henrique', 'Maria']
['Davi', 'Felipe', 'Henrique', 'Maria', 'Laura']
```

Para uma lista com apenas quatro elementos, como a do exemplo, essa é uma forma razoável de resolver esse problema. No entanto, existem métodos automatizados para realizar a tarefa de copiar item por item de uma lista, como veremos na Subseção 3.5.2.

Exemplo 3.8. O comando `insert(POSIÇÃO, DADO)` acrescenta um elemento (`DADO`) na posição (`POSIÇÃO`) da lista

```
# Usando o comando insert() para acrescentar elementos em uma
    determinada posição da lista:
lista_alunos = ['Davi', 'Felipe', 'Henrique', 'Maria']
lista_notas = [9.5, 7.5, 10, 8.0]

new_lista_alunos = lista_alunos
print(lista_alunos)
new_lista_alunos.insert(3, "Laura")
print(new_lista_alunos)

new_lista_notas = lista_notas
print(lista_notas)
new_lista_notas.insert(3, 7.0)
print(new_lista_notas)

# Saída
['Davi', 'Felipe', 'Henrique', 'Maria']
['Davi', 'Felipe', 'Henrique', 'Laura', 'Maria']
[9.5, 7.5, 10, 8.0]
[9.5, 7.5, 10, 7.0, 8.0]
```

Caso queira alterar um dado, basta saber a posição do dado na lista e fazer a substituição. No exemplo acima, digamos que a nota de Felipe, que corresponde à posição 1 na lista, não é 7.5, mas sim 8.5, então basta fazer como no exemplo a seguir:

Exemplo 3.9. Alterando o dado na posição 1 da lista:

```
lista_notas = [9.5 , 7.5 , 10 , 7.0 , 8.0]
new_lista_notas[1] = 8.5
print(new_lista_notas)

# Saída
[9.5, 8.5, 10, 7.0, 8.0]
```

3.1.7. Tuplas (*tuple*)

Assim como a Lista, a Tupla é um tipo que agrupa um conjunto de elementos. No entanto, há uma diferença em sua forma de definição: utiliza-se parênteses ( ) ao invés de colchetes como nas listas. Após a definição, Tuplas não podem ser modificadas, ou seja, elas são imutáveis, diferentemente das Listas.

Exemplo 3.10. Tuplas em Python:

```
tupla_alunos = ('Davi', 'Felipe', 'Henrique', 'Maria')
tupla_notas = (9.5, 7.5, 10, 8.0)

print(tupla_alunos)
print(tupla_notas)

print(type(tupla_alunos))
print(type(tupla_notas))

# Comentar os comandos abaixo:
#tupla_notas[1] = 8.5    # comando inválido para o tipo tupla
#tupla_notas.append(8.5) # comando inválido para o tipo tupla

# Descomentar os comandos abaixo:
new_list = [tupla_notas[:]]
print(new_list)
# new_list.append(8.5)
# print(new_list)

## Outra opção:
# new_list = [tupla_notas[0], tupla_notas[1],tupla_notas[2],
   tupla_notas[3]]
```

```
# print(new_list1)
# new_list1.append(8.5)
# print(new_list1)

# Saída
('Davi', 'Felipe', 'Henrique', 'Maria')
(9.5, 7.5, 10, 8.0)
<class 'tuple'>
<class 'tuple'>
[(9.5, 7.5, 10, 8.0)]
```

3.1.8. Dicionário (*dict*)

Por ser um tipo de dado bastante flexível, ele é utilizado para agrupar elementos através da estrutura de chave e valor, onde a chave é o primeiro elemento seguido por dois pontos e pelo valor. Por exemplo:

Exemplo 3.11. Criando um dicionário cujas chaves são nomes (*strings*) e os valores são notas (*ints e floats*):

```
dict_notas = {'Davi': 9.5, 'Felipe': 8.5, 'Henrique': 10, '
    Maria':8.0}
print(dict_notas)
print(type(dict_notas))

# Saída
{'Davi': 9.5, 'Felipe': 8.5, 'Henrique': 10, 'Maria': 8.0}
<class 'dict'>
```

Para saber mais detalhes sobre os tipos de dados, pode-se consultar a documentação oficial do Python 3, disponível em [7].

3.2. Operadores Aritméticos

O Python funciona muito bem como uma calculadora e as operações básicas são realizadas através dos seguintes operadores:

- Adição (+)

- Subtração (-)
- Multiplicação (*)
- Divisão (/)
- Divisão inteira (//)
- Módulo ou resto da divisão (%)
- Potenciação (**)
- Radiciação (**)

No exemplo a seguir é possível verificar como usar cada um desses operadores para realizar operações através do Python.

Exemplo 3.12. Uso de operadores aritméticos:

```
# Soma: a + b
print("2 + 3 =", 2+3)

# Subtração: a - b
print("4 - 5 =", 4-5)

# Multiplicação: a * b
print("5 * 4 * 3 * 2 * 1 =", 5 * 4 * 3 * 2 * 1)

# Divisão: a / b; b diferente de zero (b != 0)
print("10 / 6.5 =", 10 / 6.5)

# Divisão inteira: a // b; b != 0
print("10 // 6.5 =", 10 // 6.5)

# Módulo: a % b
print("10 % 6 =", 10 % 6)

# Potenciação: a ** b
print("10 ** 6 =", 10 ** 6)

```

```
# Radiciação: a ** (1/b); b inteiro diferente de zero
print("8**(1/3) =", 8**(1/3))
print("8**(-1/3) =", 8**(-1/3))
print("(1/8)**(1/3) =", (1/8)**(1/3))

# Saída
2 + 3 = 5
4 - 5 = -1
5 * 4 * 3 * 2 * 1 = 120
10 / 6.5 = 1.5384615384615385
10 // 6.5 = 1.0
10 % 6 = 4
10 ** 6 = 1000000
8**(1/3) = 2.0
8**(-1/3) = 0.5
(1/8)**(1/3) = 0.5
```

Em Python é possível utilizar vários tipos de operadores em uma mesma expressão, no entanto deve-se estar atento à ordem em que eles são escritos, para obter o resultado esperado. A convenção matemática é adotada para operadores matemáticos em Python. Podemos lembrar essa ordem usando o acrônimo PEMDAS: Parênteses, Expoentes, Multiplicação e Divisão (da esquerda para a direita), Adição e Subtração (da esquerda para a direita).

Exemplo 3.13. Calcular em Python $3 + 2 * 4$, retorna 11, já o resultado de $(3 + 2) * 4$ é 20. Escrevendo em linguagem Python, teríamos:

```
# Expressões com resultados diferentes
print(3+2*4)
print((3+2)*4)

# Saída
11
20
```

3.3. Operadores de Atribuição

No Python, também é possível atribuir nomes às variáveis para substituí-las pelos operandos e utilizar os operadores que controlam como a atribuição será realizada.

O símbolo de igual `=` corresponde ao operador de atribuição. Isto é, dada uma variável `n`, ao fazermos `n = valor` atribuímos o `valor` à variável `n`. Deste modo, `n` passa a ter um sinônimo de valor. Se fizermos `n = 13`, como no Exemplo 3.14, ao inserir o comando para imprimir `n`, o sistema retornará o valor `13`. A seguir, é possível verificar o operador de atribuição e seu equivalente em Python:

Operador	*Equivalente a*
$=$	$n = 1$
$+=$	$n = n + 1$
$-=$	$n = n - 1$
$*=$	$n = n * 1$
$/=$	$n = n/1$
$\%=$	$n = n\%1$

Exemplo 3.14. Atribuindo o valor 13 à variável n e executando os demais comandos, usando operadores de atribuição, obteremos os resultados abaixo:

```
n=13
print(n)
n -= 15
print(n)
n**=4
print(n)
n/=4
print(n)
n%=3
print(n)
n+=12
print(n)
n*=0.5
print(n)
```

```
15
16 # Saída
17 13
18 -2
19 16
20 4.0
21 1.0
22 13.0
23 6.5
```

Assim como as demais linguagens de programação, o Python possui palavras reservadas, as quais não poderão ser utilizadas em definições, nome de funções e classes, etc. A Tabela 3.1 contém as palavras reservadas no Python.

and	as	assert	break	class	continue	def	del	elif	else	except
False	finally	for	from	global	if	import	in	is	lambda	None
nonlocal	not	or	pass	raise	return	True	try	while	with	yield

Tabela 3.1: Palavras reservadas no Python.

Caso alguma dessas palavras seja utilizada no código, o sistema irá retornar um erro, então não precisa preocupar-se em decorá-las.

É possível usar letras, dígitos e alguns caracteres especiais para nomear as variáveis, funções e classes, no entanto os nomes sempre devem começar por letras ou sublinhado (_). A seguir, temos alguns exemplos de nomes de variáveis:

Exemplo 3.15. Possíveis nomes de variáveis em Python.

```
1 x = 4
2 y = 7
3 z = -10
4 print(x + y + z)
5 print(x * y)
6
7 # Saída
8 1
9 28
```

Exemplo 3.16. Utilizando operadores aritméticos para converter minutos em horas:

```
minutos = 700
horas = minutos/60
print(minutos, "minutos corresponde a", horas, "horas")
minutos = 500
horas = minutos/60
print(minutos, "minutos corresponde a", horas, "horas")

# Saída:
700 minutos corresponde a 11.666666666666666 horas
500 minutos corresponde a 8.333333333333334 horas
```

Podemos fazer com que a saída acima esteja mais próximo da linguagem humana ao utilizar outros operadores aritméticos:

```
minutos=700
horas=minutos/60
horas_inteiras = minutos // 60
minutos_restantes = minutos % 60
print(minutos,"minutos corresponde a", horas_inteiras, "horas e
    ", minutos_restantes, "minutos")

minutos=500
horas=minutos/60
horasinteiras=minutos//60
minutosrestantes=minutos%60
print(minutos,"minutos corresponde a", horasinteiras, "horas e"
    , minutosrestantes, "minutos")

# Saída
700 minutos corresponde a 11 horas e 40 minutos
500 minutos corresponde a 8 horas e 20 minutos
```

É possível realizar as operações aritméticas para números complexos através do Python, como mostra o exemplo a seguir:

Exemplo 3.17. Operações básicas com números complexos:

```
z1 = 3 + 4j
```

```
z2 = 2 - 2j
soma = z1 + z2
sub  = z1 - z2
mult = z1 * z2
div  = z1 / z2
print(soma)
print(sub)
print(mult)
print(div)
print(z1.real + z2.real)
print(z1.imag + z2.imag)
print(z1.conjugate())

#Saída
(5+2j)
(1+6j)
(14+2j)
(-0.25+1.75j)
5.0
2.0
(3-4j)
```

Essas operações também podem ser realizadas para outros tipos de variáveis, além das numéricas. Para saber o tipo de uma variável, basta usar o comando `type`, como no exemplo seguinte:

Exemplo 3.18. Considerando as variáveis `minutos, horas e horas_inteiras` do Exemplo 3.16 e a variável `soma` do Exemplo 3.17, obtemos:

```
print(type(minutos))
print(type(horas))
print(type(horas_inteiras))
print(type(soma))
# Saída
<class 'int'>
<class 'float'>
<class 'int'>
<class 'complex'>
```

3.4. Funções e Classes

O Python suporta o uso de funções e classes, mas nem por isso torna-as necessárias, como em outras linguagens de programação. Vamos tomar um exemplo clássico da computação, o programa "Hello, World!", que é bastante utilizado por programadores para testar o sistema ou como um exemplo de código minimalista da linguagem. Esse programa em Python pode ser escrito sem a necessidade de criar funções ou classes, da seguinte maneira:

```
# programa "Hello, World!" em Python
print("Hello, World!")
```

Em outras linguagens, como C++ e Java, a sintaxe é um pouco diferente. Em C++ esse mesmo programa requer uma função e necessita de um compilador instalado na máquina.

```
# programa " Hello, World!" em C++
#include <iostream>
int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Em Java, todo o código deve ser inserido em uma classe, como a seguir:

```
# programa " Hello, World!" em Java
class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

3.4.1. Funções

Uma função em Python é uma sequência de comandos que executa alguma tarefa. A sua principal finalidade é nos ajudar a organizar programas em pedaços que correspondam a uma solução do problema. As funções recebem nomes, que podem ser qualquer um escolhido pelo usuário, com exceção das palavras reservadas dos Python.

A sintaxe de uma definição de função é:

```
def NOME( PARAMETROS ):
    COMANDOS
```

No exemlo a seguir, vamos criar uma função para calcular a soma de dois objetos a e b:

Exemplo 3.19. Somar duas variáveis.

```
# função para calcular a soma entre dois números a e b
def sum(a, b):
    return (a + b)

a = int(input('Digite o primeiro número: '))
b = int(input('Digite o segundo número: '))

print(f'A soma de {a} e {b} é {sum(a, b)}')

# Saída
Digite o primeiro número:
3 # digite um número qualquer e pressione a tecla enter
Digite o segundo número:
-4 # digite outro número qualquer e pressione a tecla enter
A soma de 3 e -4 é -1
```

É possível usar qualquer tipo de variável usando a função `sum()`. Por exemplo:

Exemplo 3.20. Somar variáveis de diferentes tipos.

```
# Inteiros
ai = 3
bi = 5
print(f'A soma de {ai} e {bi} é {sum(ai, bi)}')

# Reais
af = float(input('Digite o primeiro número real: ')) # Digitar
    o número e em seguida pressionar a tecla Enter
bf = float(input('Digite o segundo número real: '))  # Digitar
    o número e em seguida pressionar a tecla Enter
```

```
9
10 print(f'A soma de {af} e {bf} é {sum(af, bf)}')
11
12 # Complexos
13 ac =  5 + 4j
14 bc = -3 + 2j
15
16 print(f'A soma de {ac} e {bc} é {sum(ac, bc)}')
17
18 # Diversos tipos numéricos
19 ad = 5.0
20 bd = 4
21 somad = sum(ad, bd)
22 print(f'A soma de {ad} e {bd} é {somad}')
23 print(type(ad))
24 print(type(bd))
25 print(type(somad))
26
27 # Strings
28 a_s = "le"
29 b_s = "tras"
30 soma_s = sum(a_s, b_s)
31 print(f'A soma de {a_s} e {b_s} é {soma_s}')
```

No caso dos números reais nesse exemplo, como eles são inseridos pelo usuário, deve-se atentar para inserir números reais válidos, senão o programa retorna um erro.

É possível, também, usar funções definidas em Python para diversos tipos de soluções, como arredondar números definindo a quantidade de casas decimais, ou usar notação científica, por exemplo.

Pode-se arredondar um número para quantas casas decimais se queira usando as funções `round()` ou usando `f strings`:

1. Usando a função `round()`:
   `round(número, casas_decimais)`

2. Usando `f strings`:
   `print(f'número:.casas_decimaisf')`

Exemplo 3.21. Arredondar números decimais.

```
# função para calcular a soma entre a e b
def sum(a, b):
    return (a + b)
ar = 4.6666666666
br = 7.9999999999
# Usando a função sum
somar = sum(ar, br)
print(f'A soma de {ar} e {br} é {somar}')
# Usando a função round
# arredondando para 1 casa decimal
print(f'A soma de {ar} e {br} é {round(somar,1)}')
# Usando f strings
# arredondando para 2 casas decimais
print(f'A soma de {ar} e {br} é {somar:.2f}')

# Saída
A soma de 4.6666666666 e 7.9999999999 é 12.6666666665
A soma de 4.6666666666 e 7.9999999999 é 12.7
A soma de 4.6666666666 e 7.9999999999 é 12.67
```

É possível visualizar um número em notação científica, utilizando funções do Python. No exemplo anterior, poderíamos acrescentar o código a seguir para obtermos o resultado escrito em notação científica:

Exemplo 3.22. Notação Científica.

```
# Utilizando a função format
print('{:e}'.format(somar))
# Usando f strings
print(f'{somar:e}')
# Usando f strings com o número de casas decimais
print(f'{somar:.2e}')

# Saída
1.266667e+01
1.266667e+01
1.27e+01
```

Aplicações:

Criar uma função para calcular a área de um círculo de raio r e outra para calcular o perímetro desse mesmo círculo, cujas fórmulas estão descritas na Figura 3.1.

Exemplo 3.23. Funções para calcular a área e o perímetro de um círculo dado o raio r.

```
# Calcula a área do círculo dado o raio
def area(radius):
    return radius**2*3.14

# Calcula o perímetro do círculo dado o raio
def perimeter(radius):
    return 2*radius*3.14

# Para utilizar as funções basta definir um valor para o raio
   r  e instanciar a função como nos comandos abaixo:
# Definindo o raio r = 3
r = 3
# Calcular a área de um círculo de raio r = 3
a = area(r)
# Imprimir o resultado
print(a)
# Saída
28.26
```

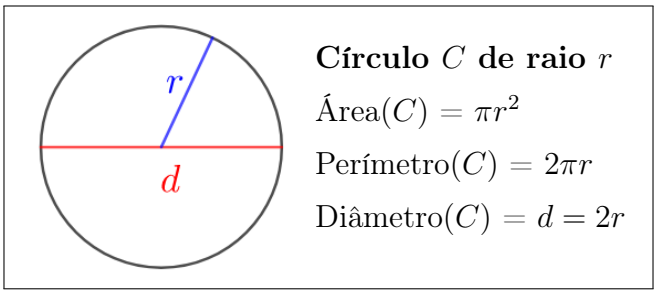


Figura 3.1: Fórmulas para calcular a área, o perímetro e o diâmetro de um círculo de raio r.

```
18
19 # Calcular o perímetro de um círculo de raio 3 e imprimir o
       resultado
20 print(perimeter(r))
21 # Saída
22 18.84
```

3.4.2. CLASSES

No Python, a criação de classes é bem simples e é possível definir os seus atributos e métodos. A sintaxe de uma definição de classe em Python é:
`class NOME:`
`COMANDO(S)`

Para criar classes que contenha novos objetos com um estado inicial predeterminado, cria-se um método especial chamado `__init__()`, o qual pode ter parâmetros para maior flexibilidade. Neste caso, os argumentos fornecidos na invocação da classe serão passados para o método `__init__()`:
`def __init__(self, PARÂMETROS):`
`self.data = []`

Para construir a nossa primeira classe com o Python, vamos criar uma classe para calcular a área e o perímetro de um círculo dado o raio, aproveitando as funções definidas no Exemplo 3.23.

Dessa forma, a classe círculo pode ser construída como mostra o exemplo abaixo.

Exemplo 3.24. Classe para calcular a área e o perímetro de um círculo dado o raio r.

```
1 # Classe círculo
2 class Circle():
3     def __init__(self, r):
4         self.radius = r
5
6     # calcula a área do círculo
7     def area(self):
8         return self.radius**2*3.14
```

```
9
10      # calcula o perímetro do círculo
11      def perimeter(self):
12          return 2*self.radius*3.14
13
14  NewCircle = Circle(8)
15  print(NewCircle.area())
16  print(NewCircle.perimeter())
17
18  # Saída
19  200.96
20  50.24
```

Desafio: Na classe círculo acima, insira uma função que calcula o *diâmetro* do círculo dado o raio r e mostre a saída.

Aplicações:

Nos exemplos seguintes, vamos criar funções e classes usando o Python para solucionar problemas comuns da matemática:

Exemplo 3.25. Código em Python para calcular a soma dos quadrados dos n primeiros números naturais:

```
1  # Programa para calcular a soma dos quadrados dos n primeiros n
       úmeros naturais
2  def squareSum(n):
3      sum = 0
4      for i in range(1, n+1):
5          sum = sum + (i+i)
6      return sum
7
8  n = 4
9  print(squareSum(n))
10 # Saída
11 30
12 n = 10
```

Exemplo 3.26. Soma mínima de fatores de um número natural n: Para encontrar a soma mínima de fatores de n, primeiro calcularemos todos os fatores e a sua soma correspondente e, depois, encontraremos o mínimo entre eles. Sendo assim, a soma dos fatores primos do produto corresponde à soma mínima de fatores do número.

Por exemplo: O número 12 pode ser fatorado como:

$$\begin{aligned} 12 = 12 * 1 \quad &= 12 + 1 \quad &&= 13, \\ 12 = 2 * 6 \quad &= 2 + 6 \quad &&= 8, \\ 12 = 3 * 4 \quad &= 3 + 4 \quad &&= 7 \text{ e} \\ 12 = 2 * 2 * 3 \quad &= 2 + 2 + 3 \quad &&= 7. \end{aligned}$$

Logo, a soma mínima de fatores do número 12 é 7.

Podemos escrever uma função para calcular essa soma utilizando a linguagem Python da seguinte forma:

```
#Programa para encontrar a soma mínima de fatores de n natural
def findMinSum(n):
    sum = 0
    #Encontrar fatores do número e adicioná-los à soma
    i = 2
    while (i*i <= n ):
        while (n % i == 0 ):
            sum += i
            n /= i
        i+=1
    sum += n
    return sum

n = 12
print(findMinSum(n))
```

3.5. INSTRUÇÕES COMPOSTAS

Nessa seção vamos usar as instruções `if`, `for` e `while` no Python para solucionar problemas comuns na matemática.

Instruções compostas contém outras instruções, ou grupos de instruções e afetam ou controlam a execução dessas outras instruções de alguma maneira. As instruções `if, for` e `while` implementam construções tradicionais de controle do fluxo de execução. Vamos estudar cada uma delas a seguir:

3.5.1. If

A instrução `if` é usada para execução condicional. A seguir temos três exemplos da sintaxe do `if`:

1.
```
if EXPRESSÃO:
    COMANDO(S)
```

2.
```
if EXPRESSÃO_1:
    COMANDO(S)
elif EXPRESSÃO_2:
    COMANDO(S)
else:
    COMANDO(S)
```

3.
```
if EXPRESSÃO_1 or EXPRESSÃO_2 and EXPRESSÃO_3:
    COMANDO(S)
```

Como exemplo de aplicação, vamos estudar os operadores de comparação e lógicos em Python.

Operadores de Comparação

São usados para comparar se dois valores são iguais (‘==‘), diferentes (‘!=‘), uma maior que outro (‘>‘), um maior ou igual ao outro (‘>=‘), um menor que outro (‘<‘), ou um menor ou igual ao outro (‘<=‘). Ele podem ser utilizados, como no exemplo a seguir:

Exemplo 3.27. Operadores de comparação em Python:

```
n1 = 7
n2 = 3

if n1 == n2:
    print(f'{n1} é igual a {n2}')

if n1 != n2:
    print(f'{n1} é diferente de {n2}')

if n1 > n2:
    print(f'{n1} é maior que {n2}')

if n1 >= n2:
    print(f'{n1} é maior ou igual a {n2}')

if n1 < n2:
    print(f'{n1} é menor que {n2}')

if n1 <= n2:
    print(f'{n1} é menor ou igual a {n2}')

# Saída

7 é diferente de 3
7 é maior que 3
7 é maior ou igual a 3
```

Poderíamos escrever o código acima de uma forma mais "enxuta", da seguinte forma:

```
if n1 < n2:
    print(f'{n1} é menor que {n2}')
elif n1 > n2:
    print(f'{n1} é maior que {n2}')
else:
    print(f'{n1} e {n2} são iguais')

# Saída
7 é maior que 3
```

Operadores Lógicos:

Permitem construir um tipo de teste muito útil em qualquer programa Python: os testes lógicos. O Python tem três tipos de operadores lógicos: `and, or` e `not`.

- `and`: retorna `True` se ambas as afirmações forem verdadeiras;
- `or`: retorna `True` se uma das afirmações for verdadeira;
- `not`: retorna `False` se o resultado for verdadeiro.

Exemplo 3.28. Operadores lógicos em Python:

```
n1 = 7
n2 = 3
n3 = -2

# Usando and
if n1 > n2 and n2 > n3:
  print(f"{n1} > {n2} > {n3}")
if n1 > 3 and n2 < 8:
    print("As Duas condições são verdadeiras")

# Saída
7 > 3 > -2
As Duas condições são verdadeiras

# Usando or
if n1 > n2 or n1 > n3:
  print(f"{n1} > {n2} ou {n1} > {n3}")
if n1 > 4 or n2 <= 8:
    print("Uma ou duas das condições são verdadeiras")

# Saída
7 > 3 ou 7 > -2
Uma ou duas das condições são verdadeiras


# Usando not
```

```
if not (n3 > n2):
  print(f"{n3} <= {n2}")
if not (n3 > n2 or n3 > n1):
  print(f"{n3} < {n2} e {n3} < {n1}")
if not (n1 < 100 and n2 < -5):
    print("Inverte o resultado da condição entre os parenteses")

# Saída
-2 <= 3
-2 < 3 e -2 < 7
Inverte o resultado da condição entre os parenteses
```

Exemplo 3.29. Determinar se um triângulo é equilátero, isósceles ou retângulo. Nesse exemplo, os tamanhos dos lados a, b e c não nulos de um triângulo serão fornecidos pelo usuário. Vamos escrever uma função usando a instrução `if` para determinar se o triângulo é equilátero, isósceles ou retângulo. Sabendo que:

a) Um triângulo é isósceles quando possui os dois lados iguais;

b) Um triângulo equilátero é também isósceles;

c) Um triângulo é retângulo se $h^2 = c_1^2 + c_2^2$, onde h é a hipotenusa e c_1^2 e c_2^2 são os catetos.

Para solucionar esse problema em Python, podemos escrever um código da seguinte forma:

```
def triangulo():
    print("Entre com o tamanho dos lados a, b e c do triangulo")
    a = float(input("Lado a:"))
    b = float(input("Lado b:"))
    c = float(input("Lado c:"))
    if (a==b) or (b == c) or (a == c):
      print("triangulo isosceles")
    if (a == b) and (b == c) and (a == c):
      print("triangulo equilatero")
    if ((a*a) == (b*b + c*c)) or ((b*b) == (a*a + c*c)) or ((c*c) == (a*a + b*b)):
      print("triangulo retangulo")
```

```
12
13 # Chamando a função triangulo
14 triangulo()
15
16 # Um exemplo de saída
17 Entre com o tamanho dos lados a, b e c do triangulo
18 Lado a:3
19 Lado b:4
20 Lado c:5
21 triangulo retangulo
```

Observe que esse código contém alguns problemas:

1. Se o triângulo não for isósceles, equilátero ou retângulo, nenhuma mensagem será retornada.

2. Se a, b e c são lados de um triângulo. Então:

   (a) nenhum deles pode ser nulo.

   (b) a soma de dois lados deve ser maior que o outro lado que não participou da soma.

   No entanto, o usuário não está sendo alertado sobre os valores, já que o mesmo pode cometer enganos.

Desafio: O que você faria para solucionar os problemas acima? Tente escrever a solução no código em Python.

3.5.2. For

A instrução `for` é usada para iterar sobre os elementos de uma sequência, que pode ser uma *string*, tupla ou lista, ou outro objeto iterável.

Sintaxe:

```
for ELEMENTO in ITERADOR :
    COMANDO(S)
```

Ou

```
for CONTADOR in range(INÍCIO, FIM, PASSO):
```

```
COMANDO(S)
```

Se o passo de contagem for omitido, o programa fará uso de passo 1 por padrão.

A contagem apresentada no `range()` será até `FIM - 1`.

Exemplo 3.30. Função para calcular a soma dos n primeiros números naturais:

```
# Calcula a soma dos n primeiros números naturais
def Sum(n):
    sum = 0
    for i in range(n+1): # Nesse caso (n+1) é o fim
        sum = sum + i
        print(sum)
    return sum

n = 5
Sum(n)

# Saída
0
1
3
6
10
15
```

Exemplo 3.31. Criar um código em Python para calcular a soma dos quadrados dos n primeiros números naturais:

```
#Função para calcular a soma dos quadrados dos n primeiros nú
   meros naturais
    def squareSum(n):
        sum = 0
        for i in range(1, n+1): # Nesse caso 1 é o início e (n
   +1) é o fim
            sum = sum + (i*i)
        return sum

```

```
    # Digitar um número natural e em seguida pressionar a tecla Enter
    n = int(input('Digite o valor de n: '))  # O comando int(), transforma um número em inteiro.
    print(f'A soma dos quadrados dos {n} primeiros números naturais é {squareSum(n)}')
```

Exemplo 3.32. Função para calcular a soma dos números naturais pares de 2 até n:

```
# Calcula a soma dos números naturais pares de 2 até n
def evenSum(n):
    sum = 0
    for i in range(2, n+1, 2): # Nesse caso 2 é o início, (n+1) é o fim e o passo é 2.
        sum = sum + i
    return sum

n = 8
evenSum(n)
```

Desafio: Como você faria a soma dos números ímpares de 1 até n?

Exemplo 3.33. Usando o for para resolver o problema das listas, visto nos Exemplos 3.6 e 3.7:

```
# O comando append() acrescenta um elemento no final da lista
lista_alunos = ['Davi', 'Felipe', 'Henrique', 'Maria']

new_lista_alunos = []
for i in range(len(lista_alunos)) :
  new_lista_alunos.append(lista_alunos[i])
print(lista_alunos)
new_lista_alunos.append("Laura")
print(lista_alunos)
print(new_lista_alunos)

# Saída
['Davi', 'Felipe', 'Henrique', 'Maria']
['Davi', 'Felipe', 'Henrique', 'Maria']
['Davi', 'Felipe', 'Henrique', 'Maria', 'Laura']
```

3.5.3. WHILE

A instrução `while` é usada para execução repetida enquanto uma expressão é verdadeira.

Sintaxe:

```
while (CONDIÇÃO LÓGICA):
    COMANDO(S)
```

Exemplo 3.34. Calcular a soma mínima de fatores de um número natural n.

Para encontrar a soma mínima de fatores de n, primeiro calcularemos todos os fatores e a sua soma correspondente e, depois, encontraremos o mínimo entre eles. Sendo assim, a soma dos fatores primos do produto corresponde à soma mínima de fatores do número.

Por exemplo: Para o número 12 temos as seguintes fatorações e as respectivas soma dos fatores:

$$\begin{aligned} 12 = 12 * 1 \quad &=> 12 + 1 \quad = 13, \\ 12 = 2 * 6 \quad &=> 2 + 6 \quad = 8, \\ 12 = 3 * 4 \quad &=> 3 + 4 \quad = 7, \text{ e} \\ 12 = 2 * 2 * 3 &=> 2 + 2 + 3 \ = 7. \end{aligned}$$

Logo, a soma mínima de fatores do número 12 é 7.

Podemos escrever uma função para calcular essa soma, utilizando a linguagem Python, da seguinte forma:

```
#Programa para encontrar a soma mínima de fatores de n natural
def findMinSum(n):
    sum = 0
    #Encontrar fatores do número e adicioná-los à soma
    i = 2
    while (i*i <= n ):
        while (n % i == 0 ):
            sum += i
            n /= i
        i+=1
    sum += n
    return sum
```

```
13
14 n = 120
15 print( findMinSum(n))
```

Exemplo 3.35. Solução de Sistemas Lineares. Sistemas lineares compostos por n equações lineares e n variáveis tem como forma geral:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \qquad\quad \vdots \qquad\qquad \vdots \qquad\quad \vdots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

Os a_{ij} são os coeficientes.

A solução desses sistemas consistem em determinar os valores de x, de modo que eles satisfaçam todas as equações ao mesmo tempo. Um dos métodos para solucionar esse tipo de sistema é o *Método de Eliminação de Gauss*, o qual consiste em transformar o sistema linear original, em um sistema linear equivalente com matriz triangular superior, que apresentam solução imediata.

Seja $Ax = B$ um sistema linear, deve-se aplicar uma sequência de operações elementares escolhidas entre:

- Trocar a ordem de duas equações do sistema;
- Multiplicar uma equação do sistema por uma constante não nula;
- Adicionar um múltiplo de uma equação a uma outra equação;

Dessa forma, obtemos um novo sistema $Ux = C$, equivalente ao sistema $Ax = B$.

O Método é definido pela aplicação dos seguintes passos:

1. Determinar a Matriz Completa do Sistema - Obter a matriz completa ou aumentada do sistema original.
2. Triangulação - Transformar a matriz A em uma matriz triangular superior por meio de uma sequência de operações elementares nas linhas da matriz A.

3. Retro-substituição - Cálcular os coeficientes $x_1, x_2, ..., x_n$, solução de $Ax = B$, a partir da solução do último componente, x_n, e a substituição regressiva nas equações anteriores.

A seguir temos um código em Python que, dada uma matriz A, calcula a sua matriz triangular superior:

```
'''Recebe uma matriz A (n x n) e
  retorna a matriz zerada na diagonal inferior esquerda. '''
def pivotamento(A):
    for i in range(len(A) - 1):
        for j in range(i + 1,len(A)):
            n1 = A[j][i]
            if n1 == 0:
                for k in range(i,len(A)):
                    if A[k][i] != 0:
                        swap(A, i, k)
                        n1 = A[j][i]
            n2 = A[i][i]
            if n2 == 0:
                continue
            div = n1 / n2
            A[j] = sumVector(multVector(A[i], -div), A[j])
    return A

def multVector(v, x):
    newV = []
    for i in range(len(v)):
        newV += [v[i] * x]
    return newV

def sumVector(v1, v2):
    newV = []
    for i in range(len(v1)):
        newV += [v1[i] + v2[i]]
    return newV

def swap(A, i, j):
    newI = []
    newJ = []
```

```
34      for index in range(len(A) + 1):
35          newI += [A[j][index]]
36          newJ += [A[i][index]]
37      A[i] = newI
38      A[j] = newJ
39
40  def printMatrix(A):
41    for i in A:
42      print('\t'.join(map(str, i)))
43    print("\n")
44
45  A = [[1.0, -1.0, 2.0, -1.0, -8.0], [2.0, -2.0, 3.0, -3.0, -20],
        [1.0, 1.0, 1.0, 0.0, -2.0], [1.0, -1.0, 4.0, -3.0, 4.0]]
46  printMatrix(A)
47  printMatrix(pivotamento(A))
```

3.6. Exercícios

1) Faça uma função que some os números inteiros a partir de 1 até que a soma não seja maior que 5000. Imprima o valor para o qual isto ocorreu e o resultado final.

2) Faça uma função que soma os n termos de uma Progressão Aritmética de razão r. Dados os valores de entrada r, a_1 e n, imprima o resultado na tela.

3) Complete o algoritmo visto no Exemplo 3.35 para que ele resolva o Passo 3 do método de eliminação de Gauss e retorne a solução do sistema.

4

O módulo Math

Nesse capítulo será apresentado o módulo Math do Python e como utilizá-lo em aplicações matemáticas.

O módulo `math` [10] faz parte da *Python Standard Library* [11]. Isso significa que todas as funções do módulo `math` estão disponíveis em qualquer instalação do Python. Ele possibilita acessar funções matemáticas definidas pelo padrão da linguagem de programação C. Esse módulo permite operações apenas entre números inteiros, ou pontos flutuantes e as funções fornecidas retornam valores em pontos flutuantes, exceto quando explicitamente indicados de outra forma. Para números complexos, deve-se utilizar o módulo `cmath`, cujas funções possuem a mesma nomenclatura das funções do módulo `math`.

Para usar o módulo `math` basta importá-lo integralmente em seu programa através do comando:

```
1 # Comando para importar o módulo math
2 import math
```

Ou importar apenas as funções que serão utilizadas:

```
1 # Comando para importar as funções log, log10, exp, e, pow e
    sqrt do módulo math
2 from math import log, log10, exp, e, pow, sqrt
```

Para conhecer todas as funções disponíveis nesse módulo, pode-se consultar a documentação oficial disponível em [10].

Nas seções a seguir serão apresentadas algumas funções matemáticas e sua correspondente em Python, utilizando o módulo `math`.

4.1. FUNÇÕES PISO E TETO

A função piso, denotada por $\lfloor x \rfloor$, converte um número real x no maior número inteiro menor ou igual a x, enquanto a função teto, denotada por $\lceil x \rceil$ converte um número real x no menor número inteiro maior ou igual a x. Ou seja,

$$\lfloor x \rfloor = \max \{m \in \mathbb{Z} \mid m \leq x\}, \tag{4.1}$$

$$\lceil x \rceil = \min \{n \in \mathbb{Z} \mid n \geq x\}. \tag{4.2}$$

Por exemplo, o número $2,1$ tem como piso o número 2 e como teto o número 3, ou seja, $\lfloor 2,1 \rfloor = 2$ e $\lceil 2,1 \rceil = 3$. Já o número $-2,1$ tem piso igual a -3 e teto igual a -2, ou seja, $\lfloor -2,1 \rfloor = -3$ e $\lceil -2,1 \rceil = -2$.

Na linguagem Python, é possível obter os valores do piso e do teto de um número utilizando, respectivamente, as funções `floor()` e `ceil()` do módulo `math`, como pode ser visto nos Exemplos 4.1 e 4.2.

Exemplo 4.1. Função piso usando o módulo `math`:

```
#Importar o módulo math
import math as mt

# Imprimir o valor do piso de diferentes números decimais
print ("O valor do piso de 2.1 é: ", mt.floor(2.1))
print ("O valor do piso de 2.7 é: ", mt.floor(2.7))
print ("O valor do piso de -2.1 é: ", mt.floor(-2.1))
print ("O valor do piso de -2.7 é: ", mt.floor(-2.7))

#Saída:
O valor do piso de 2.1 é:  2
O valor do piso de 2.7 é:  2
O valor do piso de -2.1 é:  -3
O valor do piso de -2.7 é:  -3
```

Exemplo 4.2. Função teto usando o módulo `math`:

```
#Importar o módulo math
import math as mt
```

```
3
4 a = 2.1
5 b = 2.7
6
7 # Imprimir o valor do teto dos números decimais a, b, -a e -b
8 print ("O valor do teto de",a,"é: ", mt.ceil(a))
9 print ("O valor do teto de",b,"é: ", mt.ceil(b))
10 print ("O valor do teto de",-a,"é: ", mt.ceil(-a))
11 print ("O valor do teto de",-b,"é: ", mt.ceil(-b))
12
13 # Saída:
14 O valor do teto de 2.1 é:  3
15 O valor do teto de 2.7 é:  3
16 O valor do teto de -2.1 é:  -2
17 O valor do teto de -2.7 é:  -2
```

4.2. Logaritmo e Exponencial

Além de adição, subtração, multiplicação e divisão, pode-se calcular expoentes e logaritmos com o Python. As funções exponenciais e logarítmicas são importadas do módulo `math`. A seguir veremos algumas delas:

- `e`: constante e base dos logaritmos naturais (número de Euler ou Néper).
- `exp(x)`: função para calcular uma exponenciação do tipo e^x.
- `log(x)`: calcula o logaritmo natural de x.
- `log10(x)`: calcula o logaritmo de x na base 10.

É possível renomear uma função no momento que ela está sendo importada. Portanto, para renomear a função `log(x)` que calcula o logaritmo natural, para `ln(x)`, podemos importar a função `log` como `ln`, como no exemplo a seguir:

Exemplo 4.3. Logaritmos usando o módulo `math`:

```
# Importando as funções log10 e log do módulo math
# Renomeando a função log como ln
from math import log10, log as ln

x = ln(0.1)
print(x)
x = log10(0.1)
print(x)

# Saída
-2.3025850929940455
-1.0
```

4.3. Funções trigonométricas

As função trigonométrica são amplamente usadas para resolver muitos problemas de trignometria. Dentre as funções trigonométricas do módulo `math`, estão:

- `cos(x)`: Retorna o cosseno de x em radianos.
- `sin(x)`: Retorna o seno de x em radianos.
- `tan(x)`: Retorna a tangente de x em radianos.
- `acos(x)`: Retorna o arco cosseno de x em radianos.
- `asin(x)`: Retorna o arco seno de de x em radianos.
- `atan(x)`: Retorna o arco tangente de x em radianos.

Para ver a lista completa, consulte a documentação oficial do Python disponível em [7].

Exemplo 4.4. Usando funções trigonométricas do módulo `math`:

```
#Importar o módulo math
import math as mt

a = mt.sin(mt.pi/2)
b = mt.cos(mt.pi)
print(mt.pi)
print(a)
print(b)
print(mt.asin(a))
print(mt.acos(b))

# Saída
3.141592653589793
1.0
-1.0
1.5707963267948966
3.141592653589793
```

4.4. Aplicações

4.4.1. Teorema de Pitágoras

O teorema de Pitágoras relaciona os comprimentos dos lados de qualquer triângulo retângulo. Na geometria euclidiana, o teorema afirma que:

Teorema 4.5 (Teorema de Pitágoras)**.** *O quadrado do comprimento da hipotenusa de um triângulo retângulo é igual à soma dos quadrados dos comprimentos dos outros dois lados.*

Por definição, a hipotenusa é o lado oposto ao ângulo reto, e os dois lados que o formam são os catetos. Na Figura 4.1 o lado c é a hipotenusa, a e b são os catetos.

No Exemplo 4.6 vamos escrever um código em Python para calcular a medida da hipotenusa c dados dois lados a e b de um triângulo retângulo.

Exemplo 4.6. Calcula a medida da hipotenusa c dados dois lados a e b de um triângulo retângulo:

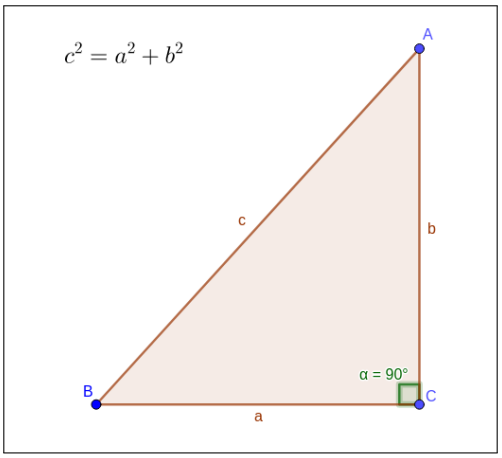


Figura 4.1: Triângulo retângulo.

```
#Importa o módulo Math para usar a função sqrt
from math import sqrt

# Atribuir valores para os catetos a e b
a = float(input("Digite o primeiro cateto: "))
b = float(input("Digite o segundo cateto: "))

# Usa a função sqrt() para calcular o valor da hipotenusa pelo
    Teorema de Pitágoras
c = sqrt((a**2) + (b**2))
# Outra forma usando a função pow
#c = sqrt(pow(a,2)+pow(b,2))

# Imprime o valor da hipotenusa
print("Hipotenusa c: ", c)

# Saída
Digite o primeiro cateto:
10
Digite o segundo cateto:
11
Hipotenusa c:  14.866068747318506
```

5

O pacote Numpy

Esse capítulo apresenta o pacote Numpy e algumas de suas funcionalidades através de aplicações da matemática.

O Numpy (*Numerical Python*) [15] é um pacote fundamental para computação científica. Seu objeto principal é o vetor n-dimensional (ndarray). Esse vetor, também chamado de tensor, deve conter todos os elementos do mesmo tipo, ou seja, deve ser homogêneo. Há diversas funções disponíveis no NumPy, as quais podem ser consultadas na documentação oficial disponível em [16].

Para instalar o `NumPy` em um computador, deve-se abrir um terminal e digitar o comando:

```
pip install numpy
```

Feito isso, é possível usá-lo para criar e manipular objetos, como será apresentado nas seções a seguir:

5.1. Arrays

Para criar um array basta usar o comando `array`, como no exemplo a seguir:

Exemplo 5.1. Criando um *array* simples usando o NumPy:

```
1 # Importando o pacote Numpy
2 import numpy as np
3
4 a = np.array([1, 2, 3, 4, 5, 6])
5
6 # Imprime todos os elementos do array
```

```
print(a)
# Imprime somente o primeiro elemento do array
print(a[0])

# Saída
[1 2 3 4 5 6]
1
```

O exemplo seguinte mostra diversas outras formas para criar *arrays* com o NumPy:

Exemplo 5.2. Outras formas de criar *arrays* usando o NumPy:

```
# Array com todos os elementos iguais a 0 (zero)
a0 = np.zeros(2)
print(a0)
# Saída
[0. 0.]

# Array com todos os elementos iguais a 1 (um)
a1 = np.ones(4)
print(a1)
# Saída
[1. 1. 1. 1.]

# Array vazio
a2 = np.empty(2)
print(a2)
# Saída
[-5.73021895e-300  6.90945411e-310]

# Array com elementos inteiros iniciando em 0 com n elementos:
    arange(n)
a3 = np.arange(4)

# Array com elementos igualmente espaçados: arange(valor
    inicial, valor final, passo)
a4 = np.arange(2, 9, 2)

```

```
25 # Array com valores linearmente espaçados em um intervalo espec
       ífico: linspace(valor inicial, valor final, número de
       elementos)
26 a5 = np.linspace(0, 10, num=5)
27
28 # Especificando o tipo de dado, o padrão (default) é float(np.
       float64)
29 a6 = np.ones(2, dtype=np.int64)
30
31 print(a3)
32 print(a4)
33 print(a5)
34 print(a6)
35
36 # Saída
37 [0 1 2 3]
38 [2 4 6 8]
39 [ 0.   2.5  5.   7.5 10. ]
40 [1 1]
```

5.1.1. Operações básicas usando arrays

É possível usar os operadores aritméticos para realizar operações com *arrays*, como veremos no exemplo seguinte:

Exemplo 5.3. Operações básicas com *arrays* usando o NumPy:

```
1  # Importando o pacote Numpy
2  import numpy as np
3
4  data = np.array([1, 2])
5  ones = np.ones(2, dtype=int)
6  soma = data + ones
7  subt = data - ones
8  mult = data * data
9  divi = data / data
10 print(soma)
11 print(subt)
12 print(mult)
```

```
print(divi)

# Saída
[2 3]
[0 1]
[1 4]
[1. 1.]
```

5.2. MATRIZES

É possível criar listas de listas com o NumPy, para criar um array 2D (matriz) e acessar os seus elementos, como no exemplo abaixo:

Exemplo 5.4. Usando o NumPy para criar e acessar elementos de uma matriz:

```
# Importando o pacote Numpy
import numpy as np

data = np.array([[1, 2], [3, 4], [5, 6]])
print(data)
# Transposta da matriz
print("\n",np.transpose(data))
# Criando uma matriz como o comando matrix
M = np.matrix(data)
print("\n",M)
print("\n",np.transpose(M))
# Saída
[[1 2]
 [3 4]
 [5 6]]

 [[1 3 5]
 [2 4 6]]

 [[1 2]
 [3 4]
 [5 6]]

 [[1 3 5]
```

```
 [2 4 6]]

# Acessando o dado do elemento 0 na posição 1
print("\n",data[0, 1])
# Acessando os dados dos elementos 0 até 2, na posição 0.
print("\n",data[0:2, 0])
# Saída
2
[1 3]

# shape retorna o número de linhas e de colunas da matriz
forma  = data.shape
# size retorna o número de elementos da matriz
num_elem = data.size

print("\n", forma)
print("\n", num_elem)
# Saída
 (3, 2)
 6

# Os três comandos abaixo são idênticos: acessa elementos da
   posição 1 até o final (3).
print("\n",data[1:3])
print("\n",data[1:forma[0]])
print("\n",data[1:])
# Saída
 [[3 4]
 [5 6]]

 [[3 4]
 [5 6]]

 [[3 4]
 [5 6]]
```

5.3. APLICAÇÕES

5.3.1. NOTAÇÃO CIENTÍFICA

Pode-se usar o NumPy para transformar um número decimal para a notação científica:

Exemplo 5.5. Usando o NumPy para escrever números em notação científica:

```
import numpy as np
a = 12
b = -3
n = 100
c = ((a+b)**n)*n**n
print(np.format_float_scientific(c))
# especificando o número de casas decimais
print(np.format_float_scientific(c, 4))

# Saída
2.6561398887587477e+295
2.6561e+295
```

5.3.2. CALCULANDO INTEGRAIS

No Exemplo 5.6 vamos calcular a seguinte integral:

$$\int_{-\infty}^{0} e^{-x^2} dx \tag{5.1}$$

Essa integral está representada na Figura 5.1 como a área marrom do gráfico.

Exemplo 5.6. Calcula a integral de uma função f de a até b usando a regra dos trapézios com n intervalos:

```
# Importando o pacote Numpy
import numpy as np

# Integra f de a a b, usando a regra dos trapézios com n
    intervalos.
def integracao(f, a, b, n=100):
```

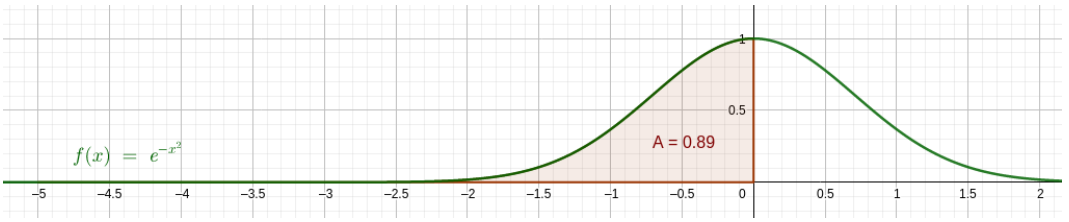


Figura 5.1: Gráfico da função $f(x) = e^{-x^2}$ (verde) e área (marrom) correspondente ao valor aproximado da integral de f entre -5 e 0.

```
    x = np.linspace(a, b, n+1) # Coordenadas dos intervalos
    h = x[1] - x[0]            # Espaçamento entre intervalos
    I = h*(sum(f(x)) - 0.5*(f(a) + f(b)))
    return I

# Definindo o integrando
def funcao(x):
    return np.exp(-x**2)

menos_infinito = -5  # Aproximação de menos infinito
I = integracao(funcao, menos_infinito, 0, n=1000)
print('Valor da integral:', I)

# Saída:
Valor da integral: 0.8862269254513772
```

5.3.3. Solução de sistemas lineares

Usando o NumPy, é possível encontrar a solução de sistemas lineares através do comando `linalg.solve`.

Exemplo 5.7. Calcular a solução do sistema de equações

$$x_0 + 2x_1 = 1$$

$$3x_0 + 5x_1 = 2$$

:

```
# Importando o pacote Numpy
import numpy as np

a = np.array([[1, 2], [3, 5]])
b = np.array([1, 2])
x = np.linalg.solve(a, b)
print(x)

# Para verificar se a solução está correta (True), use o
    comando:
s = np.allclose(np.dot(a, x), b)
print(s)

# Saída
[-1.  1.]
True
```

6

O pacote Matplotlib

Nesse capítulo será usado o pacote Matplotlib para gerar gráficos e visualizar dados em Python.

O pacote Matplotlib [17] é útil para visualizar dados em Python. Com ele é possível desenhar gráficos de diversas formas, fazer figuras interativas e exportá-los em diferentes formatos. Para conhecer as opções disponíveis no Matplotlib consulte a documentação oficial disponível [18].

Para instalar o `Matplotlib` em um computador, deve-se abrir um terminal e digitar o comando:

```
pip install matplotlib
```

Feito isso, é possível importar o Matplotlib em códigos do Python, usando o comando:

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

Por padrão, é recomendável nomeá-la como `plt`.

No exemplo a seguir, o Matplotlib será usado para visualizar dados gerados através do módulo NumPy, o qual foi apresentado no Capítulo 5.

Exemplo 6.1. Criando gráficos simples com o Matplotlib usando dados gerados no NumPy:

```
#Importando a Matplotlib
import matplotlib.pyplot as plt
#Importando o NumPy
import numpy as np

#Criando arrays com os dados
a = np.array([2, 1, 5, 7, 4, 6, 8, 14, 10, 9, 18, 20, 22])
```

```
b = np.sort(a)

#Plotando os arrays a e b em figuras separadas
plt.plot(a)
plt.show()
plt.plot(b,color='red')
plt.show()

# Plotando os arrays na mesma figura
plt.plot(a)
plt.plot(b,color='red')
plt.show()
```

A Figura 6.1 ilustra os resultados gerados com o código acima. Figura 6.1a mostra o gráfico gerado através dos comandos das linhas 11 e 12 do código, já as linhas 13 e 14 têm como resultado o gráfico gerado na Figura 6.1b e, por fim, a Figura 6.1c exibe o gráfico gerado pelos comandos das linhas 17, 18 e 19. Observe que, para gerar uma figura com todos os gráficos foi necessário incluir o comando `plt.show()` somente uma vez, após todos os `plt.plot()`.

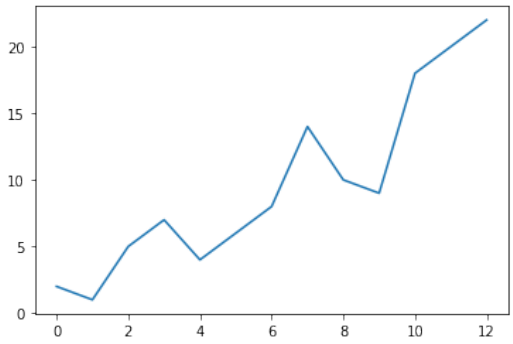
Exemplo 6.2. Gráficos simples com diferentes cores e estilos com o Matplotlib usando sequências de dados igualmente espaçados geradas no NumPy através da função linspace():

```
# Importando a Matplotlib
import matplotlib.pyplot as plt
# Importando o NumPy
import numpy as np

# Criando arrays com os dados
# linspace(inicio,fim,quantidade) cria uma quantidade de dados
    uniformemente espaçados ao longo de um intervalo [inicio,fim
    ] especificado.
x = np.linspace(0, 5, 10)
y = np.linspace(0, 10, 10)

# Plotando os dados
plt.plot(x, y, 'purple') # linha
plt.plot(y, x, 'o') # pontos
```

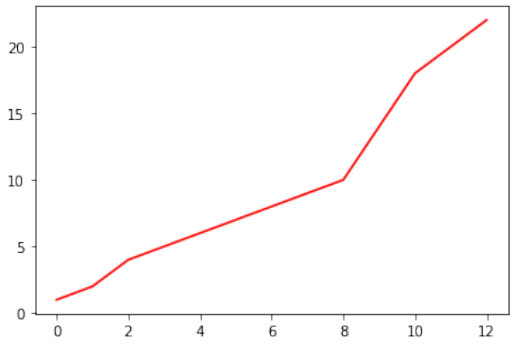
```
plt.plot(x+y, y, '--') # tracejado
plt.plot(y, y, '-o') # linha e pontos
plt.show()
```

Exemplo 6.3. Gráficos de superfícies com o Matplotlib:

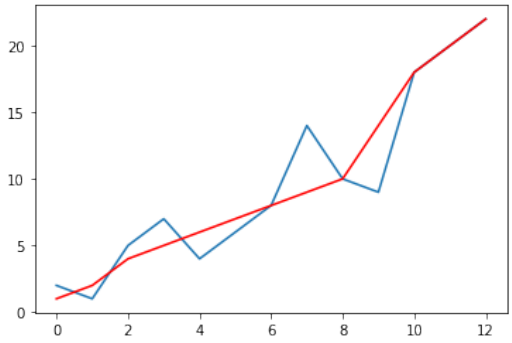
```
#Importando a Matplotlib
import matplotlib.pyplot as plt
#Importando o NumPy
import numpy as np

fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(projection='3d')
```



(a) Gráfico único.



(b) Gráfico único.



(c) Composição de gráficos.

Figura 6.1: Gráficos gerados com o Matplotlib.

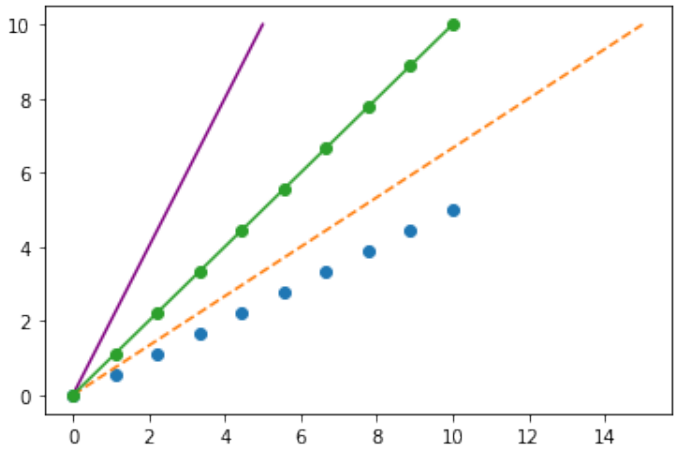


Figura 6.2: Gráficos com diferentes estilos gerados com o Matplotlib.

```
X = np.arange(-5, 5, 0.15)
Y = np.arange(-5, 5, 0.15)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.cos(R)

ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='viridis')
plt.show()
```

A Figura 6.3 ilustra o gráfico da superfície gerada com o código acima.

Para salvar uma figura, pode-se utilizar o comando:
`plt.savefig('nome_da_imagem.png')`

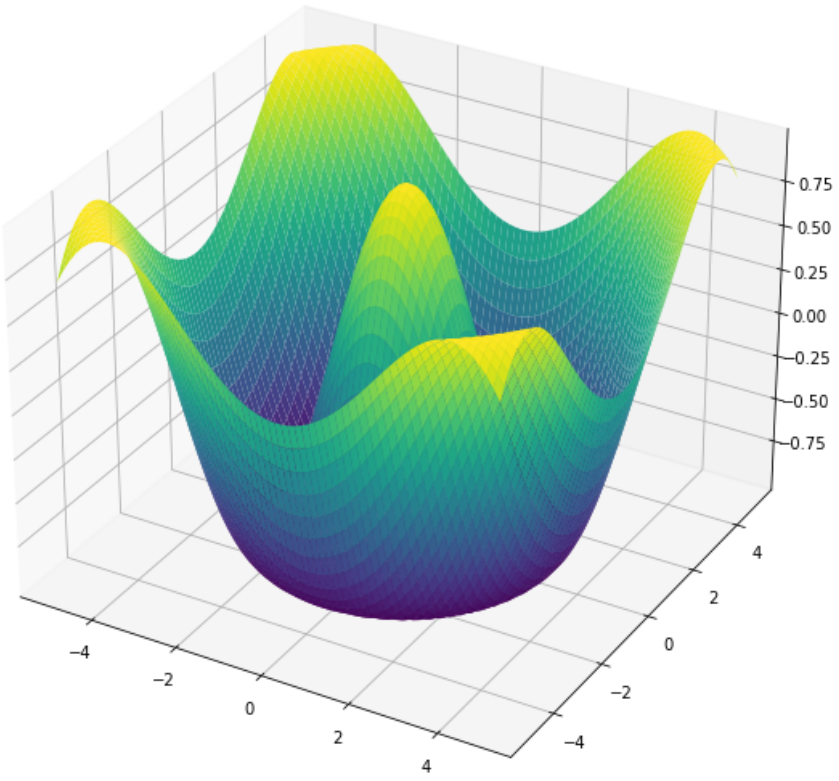


Figura 6.3: Gráfico de superfície com o Matplotlib.

Referências Bibliográficas

[1] VAN ROSSUM, G. & DRAKE, F. (2009). *Python 3 Reference Manual.* CreateSpace.

[2] GUO, P. (2014).*Python Is Now the Most Popular Introductory Teaching Language at Top U.S. Universities.* Communications of the ACM. Disponível em `https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext`. Acessado em 20 de Outubro de 2022.

[3] CASS, S. (2022). *Top Programming Languages 2022.* Disponível em `https://spectrum.ieee.org/top-programming-languages-2022`. Acessado em 19 de Outubro de 2022.

[4] *Tiobe index for October 2022.* Disponível em `https://www.tiobe.com/tiobe-index/`

[5] *Base Nacional Comum Curricular.* Disponível em `http://basenacionalcomum.mec.gov.br/`. Acessado em 20 de Outubro de 2022.

[6] *Configurações e Uso do Python.* Disponível em `https://docs.python.org/pt-br/3/using/index.html`. Acessado em 21 de Outubro de 2022.

[7] *A Referência da Linguagem Python.* Disponível em `https://docs.python.org/pt-br/3/reference/index.html`. Acessado em 21 de Outubro de 2022.

[8] *Installing Jupyter - Get up and running on your computer* Disponível em https://jupyter.org/install.html. Acessado em 21 de Outubro de 2022.

[9] *Getting Started with Python in VS Code* Disponível em https://code.visualstudio.com/docs/python/python-tutorial. Acessado em 21 de Outubro de 2022.

[10] *math — Funções matemáticas* Disponível em https://docs.python.org/pt-br/3/library/math.html. Acessado em 18 de Setembro de 2022.

[11] *The Python Standard Library* Disponível em https://docs.python.org/3/library/#the-python-standard-library. Acessado em 18 de Setembro de 2022.

[12] *Online Python* Disponível em https://www.online-python.com/. Acessado em 22 de Outubro de 2022.

[13] *Programiz - Python Online Compiler* Disponível em https://www.programiz.com/python-programming/online-compiler/. Acessado em 22 de Outubro de 2022.

[14] *Welcome to Colaboratory* Disponível em https://colab.research.google.com/. Acessado em 22 de Outubro de 2022.

[15] *NumPy documentation* Disponível em https://numpy.org/doc/stable/index.html. Acessado em 22 de Outubro de 2022.

[16] *NumPy Reference* Disponível em https://numpy.org/doc/stable/reference/index.html#reference. Acessado em 22 de Outubro de 2022.

[17] *Matplotlib: Visualization with Python* Disponível em https://matplotlib.org/. Acessado em 22 de Outubro de 2022.

[18] *Matplotlib 3.7.1 documentation* Disponível em https://matplotlib.org/stable/index.html. Acessado em 22 de Outubro de 2022.

ISBN 978-85-8337-216-5

SBM
SOCIEDADE BRASILEIRA DE MATEMÁTICA