Aulão de Python

Resumo

Ana Júlia Lima

Desenvolvido para
Young 1 - sexta #5228

Ctrl Play
Brasil
Outubro 2025

Sumário

1 Programação:

1.1 O que é programação?

Programar é dar instruções a um computador para que ele realize tarefas específicas. Essas instruções são escritas em uma linguagem de programação, que o computador entende e executa.

```
print("Olá, mundo!")
```

1.2 Linguagens de Baixo e Alto Nível:

As linguagens variam conforme o seu **nível de abstração**, ou seja, o quão próximas estão da linguagem humana.

Tipo	**Características**	**Exemplo**
Baixo Nível	Próxima da linguagem de máquina. Difícil de ler, porém muito rápida.	Assembly, C
Alto Nível	Mais próxima do português/inglês. Fácil de aprender e manter.	Python, Java, JavaScript

Baixo Nível: ideal quando é necessário controle total do hardware.
Alto Nível: ideal para desenvolvimento rápido e intuitivo.

Python é uma **linguagem de alto nível**.

1.3 Python e seus Usos:

Python é uma linguagem **simples, poderosa e versátil**, criada em 1991 por Guido van Rossum. Seu foco principal é na **legibilidade do código** e na **facilidade de aprendizado**.

Vantagens

- Sintaxe simples, parecida com o inglês.
- Multiplataforma (Windows, Mac, Linux).
- Grande comunidade e várias bibliotecas prontas.

Usos do Python

- **Ciência de Dados:** análise de dados, gráficos, inteligência artificial.
- **Web:** sites com Django e Flask.
- **Automação:** scripts para tarefas repetitivas.
- **Jogos:** criação de jogos simples com Pygame.
- **Educação:** excelente para aprender lógica de programação.

```
nome = input("Digite seu nome: ")
print("Bem-vindo,", nome)
```

2 Declaração de Variáveis e Comentários

Variáveis

Uma variável é um **espaço na memória** usado para armazenar dados.

```
nome = "Ana"
idade = 18
altura = 1.65
```

Tipos mais comuns

Tipo	Exemplo	Descrição
int	idade = 20	Números inteiros
float	altura = 1.75	Números decimais
str	nome = ”Brunno”	Textos
bool	estudando = True	Verdadeiro ou falso

Comentários

Comentários são linhas que o Python ignora. Eles são usados para explicar o código e torná-lo mais legível.

```
# Este é um comentário
nome = "Ana"  # Armazena o nome do usuário

""" Usamos 3 aspas duplas quando o comentário passa de 1 linha, também usado para docstring. """
```

3 Boas Práticas (PEP 8)

A **PEP 8** é o guia oficial de boas práticas do Python. Ela define **padrões de escrita** que tornam o código mais limpo e fácil de manter.

Dicas principais

- **Nomes de variáveis:** use letras minúsculas e “_” para separar palavras.

  ```
  nome_completo = "Ana Souza"
  ```

- **Indentação:** use 4 espaços por nível ou simplesmente use 1 tab a cada nível.

  ```
  if idade >= 18:
      print("Maior de idade")
  ```

- **Comprimento das linhas:** limite de 79 caracteres por linha.

- **Nomes de funções:** use minúsculas e underscores.

  ```
  def calcular_media(n1, n2):
      return (n1 + n2) / 2
  ```

- **Docstrings:** explique o propósito das funções.

  ```
  def somar(a, b):
      """Retorna a soma de dois números."""
      return a + b
  ```

Para mais detalhes sobre as boas práticas, consulte a PEP 8 - Guia de estilo do Python.

4 Variáveis e Tipos de Dados em Python

Em Python, uma **variável** é um nome usado para armazenar dados na memória. O tipo da variável é definido automaticamente conforme o valor atribuído.

```
idade = 18
nome = "Ana"
altura = 1.65
```

4.1 Tipos Numéricos: `int` e `float`

- `int`: números inteiros (sem parte decimal)
- `float`: números reais (com parte decimal)

```
x = 10        # inteiro
y = 3.5       # float
```

Operações Aritméticas Básicas

```
a = 10
b = 3
print(a + b)   # soma -> 13
print(a - b)   # subtração -> 7
print(a * b)   # multiplicação -> 30
print(a / b)   # divisão -> 3.333...
print(a // b)  # divisão inteira -> 3
print(a % b)   # resto -> 1
print(a ** b)  # potência -> 1000
```

4.2 Strings (`str`)

Strings são cadeias de caracteres delimitadas por aspas simples ou duplas. Elas são **imutáveis**, ou seja, não podem ser alteradas diretamente.

```
nome = "Ana Souza"
```

Funções e Métodos Úteis

- `len(s)` – retorna o tamanho da string.
- `s.count('a')` – conta quantas vezes um caractere aparece.
- `s.index('a')` – retorna o índice da primeira ocorrência.
- `s.find('a')` – semelhante a index, mas retorna -1 se não encontrar.
- `s.upper()` – converte para maiúsculas.
- `s.lower()` – converte para minúsculas.
- `s.split()` – separa em uma lista de palavras.

```
texto = "Python é incrível"
print(len(texto))      # 17
print(texto.upper())   # PYTHON É INCRÍVEL
print(texto.split())   # ['Python', 'é', 'incrível']
```

Entrada de Dados

```
nome = input("Digite seu nome: ")
print("Olá,", nome)
```

4.3 Booleanos (`bool`)

O tipo `bool` representa valores lógicos: `True` ou `False`.

```
maior_de_idade = True
chovendo = False

print(type(maior_de_idade))  # <class 'bool'>
```

Eles são frequentemente usados em condições:

```
idade = 17
if idade >= 18:
    print("Maior de idade")
else:
    print("Menor de idade")
```

5 Listas

Listas são coleções mutáveis e ordenadas. Elas podem armazenar qualquer tipo de dado (inclusive misturado).

```
numeros = [1, 2, 3, 4]
nomes = ["Ana", "Brunno", "João"]
```

Principais métodos e operações

```
lista = [10, 20, 30]

print(lista[0])     # acesso por índice -> 10
lista.append(40)    # adiciona ao final
lista.insert(1, 15) # insere na posição 1
del lista[2]        # remove pelo índice
lista.pop()         # remove o último elemento
lista.remove(15)    # remove o valor 15
lista.sort()        # ordena crescente
lista.sort(reverse=True)  # ordena decrescente
nova = sorted(lista)       # cria nova lista ordenada
lista.reverse()     # inverte a ordem
```

6 Matrizes (Listas de Listas)

Matriz é uma lista dentro de outra lista. Usada para representar tabelas ou dados bidimensionais.

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

print(matriz[0][2])  # acessa linha 0, coluna 2 -> 3
```

7 Tuplas (`tuple`)

Tuplas são semelhantes às listas, mas são **imutáveis**. São usadas quando não queremos alterar os dados.

```
tupla = (10, 20, 30, 40)

print(tupla[1])       # 20
print(len(tupla))     # 4
print(tupla.count(10))# 1
print(tupla.index(40))# 3
```

8 Conjuntos (`set`)

Conjuntos são coleções **não ordenadas e sem elementos repetidos**. Eles são úteis para eliminar duplicatas e realizar operações matemáticas de conjuntos.

```
numeros = {1, 2, 2, 3, 4}
print(numeros)  # {1, 2, 3, 4}

numeros.add(5)
numeros.remove(2)
```

Operações matemáticas:

```
a = {1, 2, 3}
b = {3, 4, 5}

print(a | b)  # união -> {1, 2, 3, 4, 5}
print(a & b)  # interseção -> {3}
print(a - b)  # diferença -> {1, 2}
```

9 Dicionários (`dict`)

Dicionários armazenam dados em pares **chave: valor**. Cada chave deve ser única, e os valores podem ser de qualquer tipo (inclusive listas, tuplas ou outros dicionários).

```
pessoa = {
    "nome": "Ana",
    "idade": 19,
    "altura": 1.65,
    "materias": ["Python", "Cálculo", "Física"]
}
```

Acesso e Métodos

```
print(pessoa["nome"])        # Acesso direto
print(pessoa.get("idade"))   # Acesso seguro

print(pessoa.keys())         # Retorna todas as chaves
print(pessoa.values())       # Retorna todos os valores
print(pessoa.items())        # Retorna pares (chave, valor)
```

Você também pode adicionar ou alterar valores:

```
pessoa["cidade"] = "Recife"
pessoa["idade"] = 20
```

10 Diferenças, Usos e Escolha de Estrutura

- **Lista:** coleção ordenada e mutável — ideal para sequências genéricas.
- **Tupla:** coleção ordenada e imutável — ideal para dados fixos (ex: coordenadas).
- **Set:** coleção não ordenada, sem duplicatas — ideal para garantir unicidade.
- **Dicionário:** coleção de pares chave:valor — ideal para representar entidades com atributos.

Estrutura	**Mutável**	**Ordenada**
Lista	Sim	Sim
Tupla	Não	Sim
Set	Sim	Não
Dicionário	Sim	Sim (desde o Python 3.7)

11 Estruturas Condicionais

As estruturas condicionais permitem que o programa tome decisões com base em uma condição (verdadeira ou falsa). Em Python, usamos `if`, `elif` e `else` para controlar o fluxo.

11.1 Operadores de Comparação

Operador	**Significado**
`==`	Igual a
`!=`	Diferente de
`>`	Maior que
`<`	Menor que
`>=`	Maior ou igual a
`<=`	Menor ou igual a

Também é possível combinar condições com:

- `and` — todas as condições devem ser verdadeiras.
- `or` — pelo menos uma deve ser verdadeira.
- `not` — inverte o valor lógico (verdadeiro $\leftrightarrow$ falso).

```
idade = 18
if idade >= 18 and idade < 60:
    print("Adulto")
```

11.2 Comandos `if`, `elif` e `else`

- `if` — executa um bloco se a condição for verdadeira.
- `elif` — avalia uma nova condição se a anterior for falsa.
- `else` — executa quando nenhuma das condições anteriores é verdadeira.

```
nota = float(input("Digite sua nota: "))

if nota >= 7:
    print("Aprovado")
elif nota >= 5:
    print("Recuperação")
else:
    print("Reprovado")
```

11.3 Condicional Aninhada

É possível colocar uma estrutura condicional dentro de outra.

```
idade = int(input("Digite sua idade: "))

if idade >= 18:
    print("Maior de idade")
    if idade >= 60:
        print("Idoso")
else:
    print("Menor de idade")
```

12 Estruturas de Repetição

As estruturas de repetição permitem executar um bloco de código várias vezes.

12.1 Laço `for`

Usado quando sabemos **quantas vezes** queremos repetir algo. Percorre uma sequência (como lista, tupla, string ou intervalo numérico).

```
for i in range(5):
    print("Repetição número:", i)
```

O comando `range(início, fim, passo)` gera uma sequência de números.

```
for numero in range(1, 10, 2):
    print(numero)  # 1, 3, 5, 7, 9
```

Também pode ser usado para percorrer listas:

```
nomes = ["Ana", "Brunno", "Lucas"]
for nome in nomes:
    print("Olá,", nome)
```

12.2 Laço `while`

Usado quando **não sabemos exatamente quantas vezes** a repetição deve ocorrer, mas queremos repetir enquanto uma condição for verdadeira.

```
contador = 0
while contador < 5:
    print("Contando:", contador)
    contador += 1
```

12.3 Diferenças entre `for` e `while`

Aspecto	for	while
Uso principal	Quando se sabe o número de repetições	Quando depende de uma condição
Controle	Automático (via range ou sequência)	Manual (condição + atualização da variável)
Exemplo típico	Percorrer listas, strings	Ler dados até o usuário digitar "sair"

12.4 Comandos `break`, `continue` e `pass`

- `break` — interrompe completamente o laço atual.
- `continue` — pula a iteração atual e vai para a próxima.
- `pass` — não faz nada (usado como "código vazio" temporário).

```
# Exemplo com break
for i in range(10):
    if i == 5:
        break
    print(i)  # imprime 0 a 4

# Exemplo com continue
for i in range(5):
    if i == 2:
        continue
    print(i)  # pula o 2

# Exemplo com pass
for i in range(3):
    pass  # usado quando ainda não há código a ser escrito
```

12.5 Exemplo Prático — Menu com Loop

```
while True:
    print("\nMenu:")
    print("1 - Dizer Olá")
    print("2 - Mostrar números")
    print("3 - Sair")

    opcao = input("Escolha uma opção: ")

    if opcao == "1":
        print("Olá, mundo!")
    elif opcao == "2":
        for i in range(1, 6):
            print(i)
    elif opcao == "3":
        print("Saindo...")
        break
    else:
        print("Opção inválida!")
```

13 Funções em Python

Funções são blocos de código que realizam uma tarefa específica. Elas servem para **organizar, reutilizar e simplificar o código**.

13.1 Para que servem

As funções:

- Evitam repetição de código.
- Tornam o programa mais legível e organizado.
- Facilitam a manutenção e testes.

```
# Exemplo simples de função
def saudacao():
    print("Olá! Seja bem-vindo(a).")

# Chamando a função
saudacao()

#Usando um for e lista dentro da função
nomes_ls = ["Ana", "Brunno", "Lucas"]

def saudacao(nomes):
    for nome in nomes:
        print("Olá! Seja bem-vindo", nome)

saudacao(nomes_ls)
```

13.2 Estrutura básica de uma função

```
def nome_da_funcao(parametros):
    """Comentário opcional (docstring) explicando o que ela faz."""
    # Bloco de código
    return resultado
```

Componentes:

- `def` — palavra-chave que indica a criação de uma função.
- **Nome da função** — identifica o que ela faz.
- **Parâmetros (argumentos)** — valores que a função recebe.
- `return` — valor retornado pela função.

13.3 Entrada (Argumentos)

As funções podem receber dados através dos parâmetros.

```
def apresentar(nome):
    print("Olá,", nome)

apresentar("Ana")
apresentar("Brunno")
```

Também é possível ter múltiplos parâmetros:

```
def soma(a, b):
    print("A soma é:", a + b)

soma(10, 5)
```

13.4 Saída (Retorno)

Usamos o comando `return` quando queremos que a função devolva um resultado.

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

resultado = soma(3, 4)
print("Resultado:", resultado)
```

Sem o `return`, a função apenas executa, mas não devolve valor.

13.5 Função com Entrada e Saída

```
def media(n1, n2):
    media_final = (n1 + n2) / 2
    return media_final

a = float(input("Primeira nota: "))
b = float(input("Segunda nota: "))
print("A média é:", media(a, b))
```

13.6 Funções Compostas (Chamando outras funções)

Uma função pode chamar outras funções — isso ajuda a dividir o problema em partes menores.

```
def quadrado(n):
    return n ** 2

def soma_quadrados(a, b):
    return quadrado(a) + quadrado(b)

print(soma_quadrados(2, 3))  # 13
```

Aqui, `soma_quadrados` depende da função `quadrado` — esse é um exemplo de **função composta**.

13.7 Funções Recursivas

Uma função é dita **recursiva** quando ela chama a si mesma. Elas são úteis para resolver problemas que podem ser divididos em subproblemas menores, como cálculos matemáticos.

```
# Exemplo: cálculo do fatorial de um número
def fatorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * fatorial(n - 1)

print(fatorial(5))  # 120
```

```
#Para observar cada iteração:
def fatorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        fatorial_atual = n * fatorial(n - 1)
        print("Número: ", n, "Resultado atual: ", fatorial_atual)
        return fatorial_atual

print(fatorial(5))

#Número:  2 Resultado atual:  2
Número:  3 Resultado atual:  6
Número:  4 Resultado atual:  24
Número:  5 Resultado atual:  120
120
```

Como funciona:

- Se `n` for 1 ou 0, a função para (caso base).
- Caso contrário, ela se chama novamente até chegar ao caso base.

13.8 Diferenças entre Tipos de Função

Tipo	Descrição	Exemplo de uso
Simples	Executa uma ação sem retornar valor	`print_mensagem()`
Com retorno	Processa e devolve um resultado	`soma(a, b)`
Composta	Chama outras funções dentro dela	`soma_quadrados(a, b)`
Recursiva	Chama a si mesma até uma condição base	`fatorial(n)`

13.9 Resumo prático

- **def**: cria a função.
- **Argumentos**: valores de entrada.
- **return**: devolve o resultado.
- **Funções compostas**: combinam outras funções.
- **Funções recursivas**: chamam a si mesmas.

14 Arquivos e Acessos

Em Python, podemos **criar, ler, escrever e manipular arquivos** de texto. Isso é muito útil para salvar dados de forma permanente (por exemplo, registros, notas, listas, etc.).

14.1 Abertura de Arquivos

Usamos a função `open()` para abrir um arquivo.

```
arquivo = open("dados.txt", "modo")
```

Modos de abertura mais comuns

Modo	Significado	Descrição
`'r'`	Read	Abre para leitura (erro se o arquivo não existir).
`'w'`	Write	Cria ou sobrescreve o arquivo.
`'a'`	Append	Adiciona conteúdo ao final sem apagar o anterior.
`'r+'`	Read + Write	Leitura e escrita sem apagar o conteúdo existente.
`'b'`	Binary	Abre em modo binário (imagens, PDFs, etc.).

14.2 Escrita em Arquivos

```
# Cria ou sobrescreve o arquivo
arquivo = open("dados.txt", "w")
arquivo.write("Olá, mundo!\n")
arquivo.write("Aprendendo Python com arquivos.\n")
arquivo.close()
```

14.3 Leitura de Arquivos

```
arquivo = open("dados.txt", "r")
conteudo = arquivo.read()    # Lê todo o arquivo
print(conteudo)
arquivo.close()
```

Outros métodos úteis:

- `read()` – lê todo o conteúdo.
- `readline()` – lê uma linha por vez.
- `readlines()` – cria uma lista com todas as linhas.

```
arquivo = open("dados.txt", "r")
for linha in arquivo:
    print(linha.strip())  # remove quebras de linha
arquivo.close()
```

14.4 Fechando Arquivos e Boas Práticas

Sempre feche o arquivo após o uso com `close()`, ou utilize o comando `with`, que fecha o arquivo automaticamente:

```
with open("dados.txt", "r") as arquivo:
    for linha in arquivo:
        print(linha.strip())
```

Essa é a maneira mais recomendada, pois evita erros e libera o arquivo corretamente.

14.5 Escrita e Leitura de Números

Para gravar números, converta-os em string. Para ler, converta de volta ao tipo desejado.

```
# Escrita
with open("numeros.txt", "w") as arq:
    for i in range(1, 6):
        arq.write(str(i) + "\n")

# Leitura
with open("numeros.txt", "r") as arq:
    for linha in arq:
        print(int(linha))
```

14.6 Resumo Prático

- **Arquivos:** usar `open()` com modos `r`, `w`, `a`.
- **with open:** boa prática que fecha o arquivo automaticamente.

15 Módulos e Práticas com Bibliotecas

Módulos são arquivos Python que contêm **funções, classes ou variáveis** que podemos importar e reutilizar em outros programas.

15.1 Importando Módulos

```
import math

print(math.sqrt(25))  # raiz quadrada
print(math.pi)        # constante pi
```

O Python já possui muitos módulos prontos (como `math`, `random`, `datetime`), e também é possível criar os seus próprios.

15.2 Importando com Apelido (`as`)

É comum usar apelidos para facilitar o uso de módulos longos.

```
import numpy as np
import pandas as pd

print(np.sqrt(16))
```

Aqui, usamos `np` e `pd` como abreviações — uma prática padrão na comunidade Python.

15.3 Importando Funções Específicas

Também é possível importar apenas o que for necessário.

```
from math import sqrt, ceil

print(sqrt(9))   # 3.0
print(ceil(4.2)) # 5
```

15.4 Criando seu próprio módulo

Basta criar um arquivo `.py` com funções e importá-lo em outro script.

```
# arquivo: utilidades.py
def saudacao(nome):
    return f"Olá, {nome}!"

# arquivo principal
import utilidades

print(utilidades.saudacao("Ana"))
```

15.5 Boas Práticas com Módulos e Bibliotecas

- Organize funções relacionadas em um mesmo arquivo.
- Use nomes claros e coerentes para módulos e funções.
- Evite nomes iguais a bibliotecas padrão (como `random.py`).
- Sempre documente suas funções com `docstrings`.
- Instale bibliotecas externas com o comando:

  ```
  pip install nome_da_biblioteca
  ```

15.6 Resumo Prático

- **Módulos:** organizam código em partes reutilizáveis.
- **import:** traz funções ou bibliotecas para o programa.
- **as:** cria apelidos para facilitar o uso.
- **from import:** importa funções específicas.

Referências

- Python Software Foundation. *Python 3 Documentation*. Disponível em: https://docs.python.org/3/. Acesso em: outubro de 2025.
- Van Rossum, G.; Drake, F. L. *The Python Language Reference*. Python Software Foundation, 2024. Disponível em: https://docs.python.org/3/reference/index.html.
- Python Enhancement Proposals (PEP). *PEP 8 – Style Guide for Python Code*. Disponível em: https://peps.python.org/pep-0008/. Acesso em: outubro de 2025.
- Real Python. *Python Tutorials and Guides*. Disponível em: https://realpython.com/. Acesso em: outubro de 2025.
- W3Schools. *Python Tutorial*. Disponível em: https://www.w3schools.com/python/. Acesso em: outubro de 2025.