Engenharia de Software
Computacional thinking with Python
Aula 05 –
Funções e Estrutura bidimensional

Prof. Dr. Francisco Elânio

Funções

O propósito de uma função em Python é encapsular um conjunto de instruções que realizam uma tarefa específica e podem ser reutilizadas.

Principais objetivos das funções em Python:

Reutilização de código: Funções permitem escrever um conjunto de instruções uma vez e reutilizá-lo em várias partes do programa. Isso reduz a redundância e torna o código mais eficiente e fácil de manter.

Funções

Principais objetivos das funções em Python:

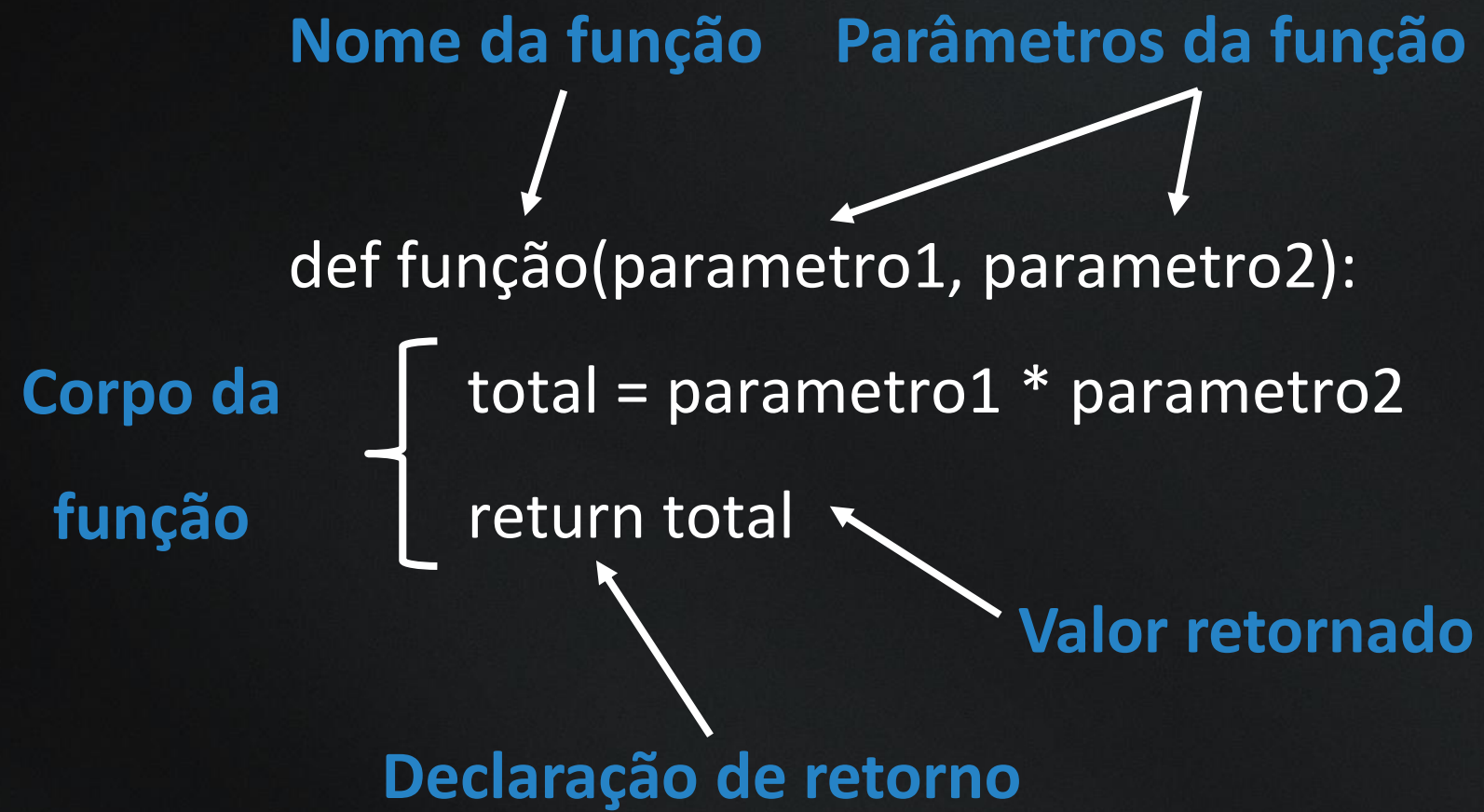
Abstração: Funções permitem abstrair detalhes de implementação complexos e fornecer uma interface simples para o usuário. Isso facilita o uso de funcionalidades complexas sem precisar entender todos os detalhes internos.

Funções

Principais objetivos das funções em Python:

Organização do código: Funções ajudam a organizar o código em partes menores e mais gerenciáveis. Isso torna o código mais legível, compreensível e fácil de manter, especialmente em projetos grandes e complexos.

Revisão sobre funções



Agenda da Aula

1. Calcule a média de uma lista de números.
2. Encontre o maior valor em uma lista de números.
3. Verifique se todos os elementos de uma lista são positivos.
4. Calcule a soma dos quadrados de uma lista de números.
5. Conte quantas vezes um determinado elemento aparece em uma lista.
6. Inverta uma lista.
7. Verifique se uma lista está ordenada de forma crescente.
8. Remova os elementos duplicados de uma lista.

Exercício 1

1. Calcule a média de uma lista de números.

```
def calcular_media(lista):
    return sum(lista) / len(lista)

numeros = [10, 20, 30, 40, 50]
media = calcular_media(numeros)
print("A média dos números é:", media)
```

Exercício 2

2. Encontre o maior valor em uma lista de números.

```
def encontrar_maior(lista):
    return max(lista)

numeros = [10, 20, 30, 40, 50]
maior = encontrar_maior(numeros)
print("O maior número é:", maior)
```

Exercício 3

3. Retorne os números negativos e positivos da lista.

```
def separar_positivos_negativos(lista):
    positivos = [num for num in lista if num > 0]
    negativos = [num for num in lista if num < 0]
    return positivos, negativos

numeros = [10, -20, 30, 40, 50, -3, 7, 8]
positivos, negativos = separar_positivos_negativos(numeros)

print("Números positivos:", positivos)
print("Números negativos:", negativos)
```

Exercício 4

4. Calcule a soma dos quadrados de uma lista de números.

```
def soma_quadrados(lista):
    return sum(num ** 2 for num in lista)

numeros = [1, 2, 3, 4, 5]
soma_quadr = soma_quadrados(numeros)
print("A soma dos quadrados é:", soma_quadr)
```

Exercício 5

5. Conte quantas vezes um determinado elemento aparece em uma lista.

```
def contar_elemento(lista, elemento):
    return lista.count(elemento)

numeros = [1, 2, 3, 4, 5, 1, 2, 3]
elemento = 3
ocorrencias = contar_elemento(numeros, elemento)
print("O elemento", elemento, "aparece", ocorrencias, "vezes na lista.")
```

Exercício 6

6. Inverta uma lista.

```
def inverter_lista(lista):
    return lista[::-1]

numeros = [1, 2, 3, 4, 5]
lista_invertida = inverter_lista(numeros)
print("Lista invertida:", lista_invertida)
```

Exercício 7

7. Verifique se uma lista está ordenada de forma crescente.

```
def esta_ordenada_crescente(lista):
    return lista == sorted(lista)

numeros = [1, 2, 3, 4, 5]
esta_ordenada = esta_ordenada_crescente(numeros)
print("A lista está ordenada de forma crescente?", esta_ordenada)
```

Exercício 8

8. Remova os elementos duplicados de uma lista.

```
def remover_duplicados(lista):
    return list(set(lista))

numeros = [1, 2, 3, 4, 5, 1, 2, 3]
sem_duplicados = remover_duplicados(numeros)
print("Lista sem elementos duplicados:", sem_duplicados)
```

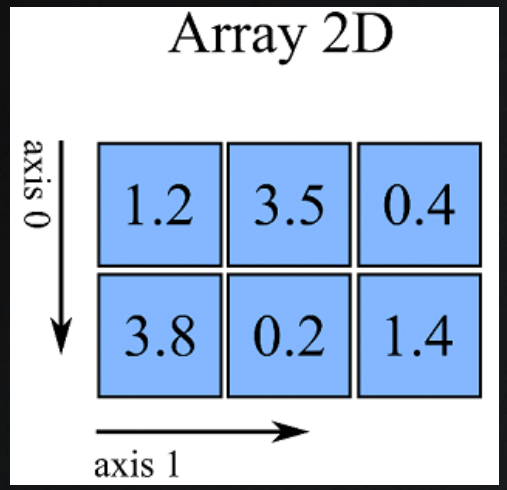
Estrutura Bidimensional

Uma estrutura de dados bidimensional é uma coleção organizada de elementos dispostos em linhas e colunas, formando uma grade ou matriz.

É frequentemente usada para representar dados que têm duas dimensões, como uma tabela, uma imagem, um mapa ou uma grade de assentos em um teatro.

A principal característica de uma estrutura bidimensional é que cada elemento é acessado utilizando dois índices: um para a linha e outro para a coluna.

Estrutura Bidimensional




```
matriz = [
    [1.2, 3.5, 0.4],
    [3.8, 0.2, 1.4]
]
```

Índice	Sexo	Idade
0	M	38
1	F	25
2	F	32
3	M	18

```
dados = [
    [1, 38],
    [0, 25],
    [1, 32],
    [0, 18]
]
```

Estrutura Bidimensional

Lendo um dataframe e transformando dados para uma matriz bidimensional.

- Baixe os dados sobre falhas em máquinas em csv.
- Lendo um dataframe
- Verifique os dados no dataframe
- Selecione apenas as variáveis Torque e Velocidade de Rotação.
- Crie uma estrutura bidimensional para duas variáveis com apenas as cinco primeiras linhas.
- Mostre o resultado da matriz.

Estrutura Bidimensional

- Lendo um dataframe

```
pip install panda

import pandas as pd

dados = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/manutenção_preventiva/manutencao_preditiva.csv')
```

Estrutura Bidimensional

- Verifique os dados no dataframe

	UDI	ID Produto	Tipo	Temperatura Ar [K]	Temperatura Processo [K]	Velocidade Rotacao [rpm]	Torque [Nm]	Desgaste Ferramenta [min]	Alvo	Tipo da Falha
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	No Failure
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	No Failure
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	No Failure
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	No Failure
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	No Failure

Estrutura Bidimensional

- Selecione apenas as variáveis Torque e Velocidade de Rotação.

```
dados_selec = dados[['Velocidade Rotacao [rpm]', 'Torque [Nm]']]
dados_selec
```

	Velocidade Rotacao [rpm]	Torque [Nm]
0	1551	42.8
1	1408	46.3
2	1498	49.4
3	1433	39.5
4	1408	40.0
...	...	...
9995	1604	29.5
9996	1632	31.8
9997	1645	33.4
9998	1408	48.5
9999	1500	40.2

Estrutura Bidimensional

- Crie uma estrutura bidimensional para duas variáveis com apenas as cinco primeiras linhas.

```
cinco_linhas = dados_selec.head(5)
cinco_linhas
```

	Velocidade Rotacao [rpm]	Torque [Nm]
0	1551	42.8
1	1408	46.3
2	1498	49.4
3	1433	39.5
4	1408	40.0

Estrutura Bidimensional

- Mostre o resultado da matriz.

cinco_linhas.values

```
array([[1551. ,   42.8],
       [1408. ,   46.3],
       [1498. ,   49.4],
       [1433. ,   39.5],
       [1408. ,   40. ]])
```

Estrutura Bidimensional

- Calcule a soma dos elementos da matriz

```
import numpy as np

soma_elementos = np.sum(cinco_linhas)

print("A soma dos elementos da matriz é:", soma_elementos)
```

Estrutura Bidimensional

- Encontre o valor mínimo em cada linha de uma matriz

```
import numpy as np

valores_minimos_por_linha = np.min(cinco_linhas, axis=1)

print("Valores mínimos em cada linha da matriz:",
valores_minimos_por_linha)
```

Estrutura Bidimensional

- Calcule a média de cada coluna de uma matriz.

```
import numpy as np

medias_por_coluna = np.mean(cinco_linhas, axis=0)

print("Média de cada coluna da matriz:", medias_por_coluna)
```

Estrutura Bidimensional

- Transponha a matriz.

```
import numpy as np

matriz_transposta = np.transpose(cinco_linhas)
matriz_transposta
```

	0	1	2	3	4
Velocidade Rotacao [rpm]	1551.0	1408.0	1498.0	1433.0	1408.0
Torque [Nm]	42.8	46.3	49.4	39.5	40.0

Exercícios com estrutura Bidimensional

Usando o dataframe anterior. Faça as seguintes tarefas:

1. Selecione dois pequenos dataframes: o primeiro contendo as variáveis temperatura do ar e torque, o segundo contendo as variáveis temperatura do processo e velocidade de rotação.
2. Crie uma estrutura bidimensional para os dois dataframes, o primeiro dataframe deve conter as 10 primeiras linhas, o seguno datafame deve conter as linhas de 15 a 25.
3. Crie uma estrutura bidimensional para os dois dataframes e mostre a matriz.

Exercícios com estrutura Bidimensional

Usando o dataframe anterior. Faça as seguintes tarefas:

4. Calcule a soma dos elementos da matriz
5. Encontre o valor mínimo e máximo em cada linha de uma matriz
6. Calcule a média de cada coluna de uma matriz
7. Transponha a matriz
8. Agora volte a calcular a média de cada coluna da matriz.

Subalgoritmos

É um algoritmo usado por outro algoritmo como parte da operação do segundo algoritmo.

Um algoritmo para encontrar o valor mediano em uma lista de números pode incluir a classificação dos números como um sub-algoritmo: Há uma abundância de algoritmos para classificar, e, e as especificidades da classificação não importa para o algoritmo "valor médio", apenas que os números são classificados quando o sub-algoritmo é feito.

Subalgoritmos - Exemplo

```
def calcular_soma(numeros):
    """Esta função calcula a soma dos números em uma lista."""
    soma = 0
    for numero in numeros:
        soma += numero
    return soma

def calcular_media(numeros):
    """Esta função calcula a média dos números em uma lista."""
    soma = calcular_soma(numeros)
    media = soma / len(numeros)
    return media

numeros = [10, 20, 30, 40, 50]
media = calcular_media(numeros)
print("A média dos números é:", media)
```

Subalgoritmos - Exemplo

```python
def calcular_soma(numeros):
    """Esta função calcula a soma dos números em uma lista."""
    soma = 0
    for numero in numeros:
        soma += numero
    return soma

def calcular_media(numeros):
    """Esta função calcula a média dos números em uma lista."""
    soma = calcular_soma(numeros)
    media = soma / len(numeros)
    return media

numeros = [10, 20, 30, 40, 50]
media = calcular_media(numeros)
print("A média dos números é:", media)
```

Subalgoritmos - Exemplo

```
def encontrar_mediano(lista):
    """Esta função encontra o valor mediano em uma lista de números."""
    lista_ordenada = sorted(lista)

    tamanho = len(lista_ordenada)
    if tamanho % 2 == 0:
        mediano = (lista_ordenada[tamanho // 2 - 1] + lista_ordenada[tamanho // 2]) / 2
    else:
        mediano = lista_ordenada[tamanho // 2]

    return mediano

numeros = [7, 3, 9, 2, 5, 8, 4, 6, 1]
mediano = encontrar_mediano(numeros)
print("O valor mediano da lista é:", mediano)
```

“O que sabemos é uma gota; o que ignoramos é um oceano.”
(Issac Newton)

Referências

- ASCENCIO, A. F. G, CAMPOS, E. A. V. Fundamentos da Programação de Computadores: algoritmos, Pascal, C/C++ e Java, 2ª Edição, São Paulo: Pearson 2007.
- FURGERI, Sérgio. Introdução à Programação em Python. São Paulo: Editora Senac, 2021.
- MENEZES, Nilo. Introdução à Programação em Python. São Paulo: Novatec, 2019
- SALVETTI, Dirceu Douglas; BARBOSA, Lisbete Madson. Algoritmos. São Paulo: Pearson, 2004.