# Exercícios Strings

[**Exercícios Strings**](#_1qehg8vycegm) **1**

[**Determine se duas strings são anagramas (contêm as mesmas letras em ordens diferentes):**](#_6r3sgwb1m0p6) 1

[Dado um conjunto de números, determine se é possível encontrar um subconjunto cuja soma seja igual a um determinado valor:](#_as0bzeqm3fnk) 2

[**Contar a frequência de cada caractere em uma string dada**](#_xtr53b881ugz) 3

[Encontrar a maior subsequência comum entre duas strings.](#_6j0dq4qnv3q5) 4

[**Contar quantas substrings distintas podem ser formadas a partir de uma string dada.**](#_mjozuhyevw3w) 5

[Dado um dicionário e uma frase, determinar se a frase pode ser formada usando palavras do dicionário.](#_7bj8fblbrj1n) 5

[**Compressão de String: Implementar um algoritmo para comprimir uma string repetindo caracteres consecutivos.**](#_76p77z5ojvdk) 6

[Strings Rotacionadas: Verificar se uma string é uma rotação cíclica de outra string.](#_2o6k31ddgae8) 7

[**Operações de Edição: Determinar o número mínimo de operações de inserção, remoção ou substituição necessárias para transformar uma string em outra.**](#_uz35btv91dad) 8

[Substrings Palindrômicas: Encontrar todas as substrings palindrômicas em uma string.](#_8496uxotapvm) 9

[**Pares de Anagramas: Encontrar todos os pares de palavras em uma lista que são anagramas entre si.**](#_bp85ah9t0mwv) 9

[String Permutável (Anagrama): Determinar se duas strings podem ser permutadas uma na outra trocando os caracteres de lugar.](#_nk2o5x9iqdd) 10

[**Exercício para verificar se uma palavra pode se tornar um palíndromo:**](#_yeb9x3eipsxa) 11

[Exercício verificar se uma palavra é um palíndromo:](#_yfz3wporpvm7) 12

[**Exercício para remover todas as consoantes de uma string:**](#_yu7p2n3fl9iv) 12

[Exercício colocar uma string em ordem alfabética:](#_s0z9t49newqn) 13

[**Exercício converter letras minúsculas em maiúsculas e maiúsculas em minúsculas:**](#_ur544giez7u6) 13

[Exercício para mostrar as letras que mais se repetem em uma string:](#_qe0qcrukiqde) 14

## Determine se duas strings são anagramas (contêm as mesmas letras em ordens diferentes):

Digite a primeira string: ana

Digite a segunda string: naa

As strings são anagramas.

def are\_anagrams(str1, str2):

str1 = str1.replace(" ", "").lower()

str2 = str2.replace(" ", "").lower()

if len(str1) != len(str2):

return False

char\_count = {}

for char in str1:

if char in char\_count:

char\_count[char] += 1

else:

char\_count[char] = 1

for char in str2:

if char in char\_count:

char\_count[char] -= 1

if char\_count[char] == 0:

del char\_count[char]

else:

return False

return len(char\_count) == 0

# Exemplo de uso

string1 = input("Digite a primeira string: ")

string2 = input("Digite a segunda string: ")

if are\_anagrams(string1, string2):

print("As strings são anagramas.")

else:

print("As strings não são anagramas.")

## Dado um conjunto de números, determine se é possível encontrar um subconjunto cuja soma seja igual a um determinado valor:

nums = [3, 34, 4, 12, 5, 2]

target = 9

É possível encontrar um subconjunto com soma 9

def subset\_sum(nums, target\_sum):

n = len(nums)

dp = [[False] \* (target\_sum + 1) for \_ in range(n + 1)]

for i in range(n + 1):

dp[i][0] = True

for i in range(1, n + 1):

for j in range(1, target\_sum + 1):

if j >= nums[i - 1]:

dp[i][j] = dp[i - 1][j] or dp[i - 1][j - nums[i - 1]]

else:

dp[i][j] = dp[i - 1][j]

return dp[n][target\_sum]

# Exemplo de uso

nums = [3, 34, 4, 12, 5, 2]

target = 9

if subset\_sum(nums, target):

print("É possível encontrar um subconjunto com soma", target)

else:

print("Não é possível encontrar um subconjunto com soma", target)

## Contar a frequência de cada caractere em uma string dada

Digite uma string: teste

't': 2

'e': 2

's': 1

def contar\_caracteres(string):

# Cria um dicionário vazio para armazenar a contagem de caracteres

contador = {}

# Itera pelos caracteres na string

for char in string:

# Ignora espaços em branco

if char != ' ':

# Incrementa o contador para o caractere atual

contador[char] = contador.get(char, 0) + 1

return contador

# Solicita a entrada do usuário

entrada = input("Digite uma string: ")

# Chama a função para contar os caracteres

resultado = contar\_caracteres(entrada)

# Exibe o resultado

for char, count in resultado.items():

print(f"'{char}': {count}")

## Encontrar a maior subsequência comum entre duas strings.

Digite a primeira string: teste

Digite a segunda string: testando

A maior subsequência comum é: test

def maior\_subsequencia\_comum(s1, s2):

m = len(s1)

n = len(s2)

# Cria uma matriz para armazenar as soluções intermediárias

dp = [[0] \* (n + 1) for \_ in range(m + 1)]

# Preenche a matriz dp usando programação dinâmica

for i in range(1, m + 1):

for j in range(1, n + 1):

if s1[i - 1] == s2[j - 1]:

dp[i][j] = dp[i - 1][j - 1] + 1

else:

dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

# Reconstrói a subsequência comum a partir da matriz dp

subsequencia = ""

i, j = m, n

while i > 0 and j > 0:

if s1[i - 1] == s2[j - 1]:

subsequencia = s1[i - 1] + subsequencia

i -= 1

j -= 1

elif dp[i - 1][j] > dp[i][j - 1]:

i -= 1

else:

j -= 1

return subsequencia

# Solicita as entradas do usuário

string1 = input("Digite a primeira string: ")

string2 = input("Digite a segunda string: ")

# Chama a função para encontrar a maior subsequência comum

resultado = maior\_subsequencia\_comum(string1, string2)

# Exibe o resultado

print("A maior subsequência comum é:", resultado)

## Contar quantas substrings distintas podem ser formadas a partir de uma string dada.

Digite uma string: davi

Quantidade de substrings distintas: 10

def contar\_substrings\_distintas(string):

n = len(string)

substrings = set()

# Loop externo para determinar o início da substring

for i in range(n):

substring = ""

# Loop interno para determinar o fim da substring

for j in range(i, n):

substring += string[j]

substrings.add(substring)

return len(substrings)

# Solicita a entrada do usuário

entrada = input("Digite uma string: ")

# Chama a função para contar as substrings distintas

resultado = contar\_substrings\_distintas(entrada)

# Exibe o resultado

print("Quantidade de substrings distintas:", resultado)

## Dado um dicionário e uma frase, determinar se a frase pode ser formada usando palavras do dicionário.

dicionario = ["eu", "gosto", "de", "programar"]

Digite uma frase: eu gosto

A frase pode ser formada usando palavras do dicionário.

def pode\_formar\_frase(dicionario, frase):

# Converte o dicionário para um conjunto para facilitar a busca

dicionario\_set = set(dicionario)

palavras\_frase = frase.split()

for palavra in palavras\_frase:

if palavra not in dicionario\_set:

return False

return True

# Dicionário de palavras

dicionario = ["eu", "gosto", "de", "programar"]

# Solicita a entrada do usuário

frase = input("Digite uma frase: ")

# Chama a função para verificar se a frase pode ser formada usando palavras do dicionário

resultado = pode\_formar\_frase(dicionario, frase)

# Exibe o resultado

if resultado:

print("A frase pode ser formada usando palavras do dicionário.")

else:

print("A frase não pode ser formada usando palavras do dicionário.")

## Compressão de String: Implementar um algoritmo para comprimir uma string repetindo caracteres consecutivos.

Digite uma string: carro

String original: carro

String comprimida: caro

def compress\_string(s):

compressed = ""

count = 1

for i in range(1, len(s)):

if s[i] == s[i - 1]:

count += 1

else:

compressed += s[i - 1]

count = 1

compressed += s[-1]

return compressed

# Exemplo de uso

entrada = input("Digite uma string: ")

compressed\_string = compress\_string(entrada)

print("String original:", entrada)

print("String comprimida:", compressed\_string)

## Strings Rotacionadas: Verificar se uma string é uma rotação cíclica de outra string.

Digite a primeira string: carro

Digite a segunda string: rocar

As strings são rotações cíclicas uma da outra.

def sao\_rotacoes\_ciclicas(string1, string2):

# Verifica se as strings têm o mesmo comprimento

if len(string1) != len(string2):

return False

# Concatena a primeira string consigo mesma para tratar todas as rotações possíveis

string1\_dupla = string1 + string1

# Verifica se a segunda string está contida na string1 duplicada

if string2 in string1\_dupla:

return True

else:

return False

# Solicita as entradas do usuário

string1 = input("Digite a primeira string: ")

string2 = input("Digite a segunda string: ")

# Chama a função para verificar se uma string é uma rotação cíclica da outra

resultado = sao\_rotacoes\_ciclicas(string1, string2)

# Exibe o resultado

if resultado:

print("As strings são rotações cíclicas uma da outra.")

else:

print("As strings não são rotações cíclicas uma da outra.")

## 

## Operações de Edição: Determinar o número mínimo de operações de inserção, remoção ou substituição necessárias para transformar uma string em outra.

Digite a primeira string: davi

Digite a segunda string: da

Número mínimo de operações de edição: 2

def operacoes\_edicao(string1, string2):

m = len(string1)

n = len(string2)

# Cria uma matriz para armazenar as soluções intermediárias

dp = [[0] \* (n + 1) for \_ in range(m + 1)]

# Preenche a matriz dp usando programação dinâmica

for i in range(m + 1):

for j in range(n + 1):

if i == 0:

dp[i][j] = j

elif j == 0:

dp[i][j] = i

elif string1[i - 1] == string2[j - 1]:

dp[i][j] = dp[i - 1][j - 1]

else:

dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1])

return dp[m][n]

# Solicita as entradas do usuário

string1 = input("Digite a primeira string: ")

string2 = input("Digite a segunda string: ")

# Chama a função para determinar o número mínimo de operações de edição

resultado = operacoes\_edicao(string1, string2)

# Exibe o resultado

print("Número mínimo de operações de edição:", resultado)

## Substrings Palindrômicas: Encontrar todas as substrings palindrômicas em uma string.

Digite uma string: radar

Substrings palindrômicas encontradas: {'d', 'a', 'ada', 'radar', 'r'}

def substrings\_palindromicas(string):

def expand\_from\_center(left, right):

while left >= 0 and right < len(string) and string[left] == string[right]:

palindromo\_substring.add(string[left:right + 1])

left -= 1

right += 1

palindromo\_substring = set()

for i in range(len(string)):

# Caso a substring seja de tamanho ímpar

expand\_from\_center(i, i)

# Caso a substring seja de tamanho par

expand\_from\_center(i, i + 1)

return palindromo\_substring

# Solicita a entrada do usuário

entrada = input("Digite uma string: ")

# Chama a função para encontrar as substrings palindrômicas

resultado = substrings\_palindromicas(entrada)

# Exibe o resultado

print("Substrings palindrômicas encontradas:", resultado)

## Pares de Anagramas: Encontrar todos os pares de palavras em uma lista que são anagramas entre si.

Digite uma lista de palavras separadas por espaço: carro teto rocar

Pares de anagramas encontrados:

carro - rocar

def sao\_anagramas(palavra1, palavra2):

return sorted(palavra1) == sorted(palavra2)

def pares\_de\_anagramas(lista\_palavras):

pares = []

for i in range(len(lista\_palavras)):

for j in range(i + 1, len(lista\_palavras)):

if sao\_anagramas(lista\_palavras[i], lista\_palavras[j]):

pares.append((lista\_palavras[i], lista\_palavras[j]))

return pares

# Solicita a entrada do usuário

entrada = input("Digite uma lista de palavras separadas por espaço: ")

lista\_palavras = entrada.split()

# Chama a função para encontrar os pares de anagramas

resultado = pares\_de\_anagramas(lista\_palavras)

# Exibe o resultado

if resultado:

print("Pares de anagramas encontrados:")

for par in resultado:

print(par[0], "-", par[1])

else:

print("Nenhum par de anagramas encontrado.")

## String Permutável (Anagrama): Determinar se duas strings podem ser permutadas uma na outra trocando os caracteres de lugar.

Digite a primeira string: carro

Digite a segunda string: rocar

As strings são permutáveis uma da outra.

def sao\_permutaveis(string1, string2):

return sorted(string1) == sorted(string2)

# Solicita as entradas do usuário

string1 = input("Digite a primeira string: ")

string2 = input("Digite a segunda string: ")

# Chama a função para determinar se as strings são permutáveis

resultado = sao\_permutaveis(string1, string2)

# Exibe o resultado

if resultado:

print("As strings são permutáveis uma da outra.")

else:

print("As strings não são permutáveis uma da outra.")

## Exercício para verificar se uma palavra pode se tornar um palíndromo:

Digite uma palavra: ovov

Pode se tornar um palíndromo ao reorganizar as letras.

def can\_become\_palindrome(word):

# Remove espaços em branco e converte para letras minúsculas para evitar problemas de comparação

cleaned\_word = word.replace(" ", "").lower()

# Conta a frequência de cada caractere na palavra

char\_count = {}

for char in cleaned\_word:

if char in char\_count:

char\_count[char] += 1

else:

char\_count[char] = 1

# Verifica quantos caracteres têm frequência ímpar

odd\_count = 0

for count in char\_count.values():

if count % 2 != 0:

odd\_count += 1

# Uma palavra pode se tornar um palíndromo se tiver no máximo um caractere com frequência ímpar

return odd\_count <= 1

# Exemplos de uso

word = input("Digite uma palavra: ")

if can\_become\_palindrome(word):

print("Pode se tornar um palíndromo ao reorganizar as letras.")

else:

print("Não pode se tornar um palíndromo ao reorganizar as letras.")

## Exercício verificar se uma palavra é um palíndromo:

Digite uma palavra: ovo

ovo é um palíndromo.

def is\_palindrome(word):

word = word.lower() # Convertendo a palavra para letras minúsculas

reversed\_word = word[::-1] # Revertendo a palavra

return word == reversed\_word

# Exemplo de uso

input\_word = input("Digite uma palavra: ")

if is\_palindrome(input\_word):

print(f"{input\_word} é um palíndromo.")

else:

print(f"{input\_word} não é um palíndromo.")

## Exercício para remover todas as consoantes de uma string:

Digite uma string: davi

String sem consoantes: ai

def remove\_consonants(input\_string):

vowels = "AEIOUaeiou"

result = ''.join(char for char in input\_string if char in vowels)

return result

# Exemplo de uso

text = input("Digite uma string: ")

result = remove\_consonants(text)

print("String sem consoantes:", result)

## Exercício colocar uma string em ordem alfabética:

Digite uma string: davi

String em ordem alfabética: adiv

def sort\_string(input\_string):

sorted\_chars = sorted(input\_string)

sorted\_string = ''.join(sorted\_chars)

return sorted\_string

# Exemplo de uso

text = input("Digite uma string: ")

sorted\_text = sort\_string(text)

print("String em ordem alfabética:", sorted\_text)

## Exercício converter letras minúsculas em maiúsculas e maiúsculas em minúsculas:

Digite uma string: davi

String com letras alternadas: DAVI

def swap\_case(input\_string):

swapped\_string = input\_string.swapcase()

return swapped\_string

# Exemplo de uso

text = input("Digite uma string: ")

swapped\_text = swap\_case(text)

print("String com letras alternadas:", swapped\_text)

## Exercício para mostrar as letras que mais se repetem em uma string:

Digite uma string: daavi

Letras mais comuns: ['a']

Frequência máxima: 2

def most\_common\_letters(input\_string):

char\_count = {}

for char in input\_string:

if char.isalpha(): # Considerar apenas letras do alfabeto

char = char.lower() # Tratar letras maiúsculas e minúsculas igualmente

if char in char\_count:

char\_count[char] += 1

else:

char\_count[char] = 1

max\_count = max(char\_count.values())

most\_common = [char for char, count in char\_count.items() if count == max\_count]

return most\_common, max\_count

# Exemplo de uso

text = input("Digite uma string: ")

common\_letters, max\_count = most\_common\_letters(text)

print("Letras mais comuns:", common\_letters)

print("Frequência máxima:", max\_count)