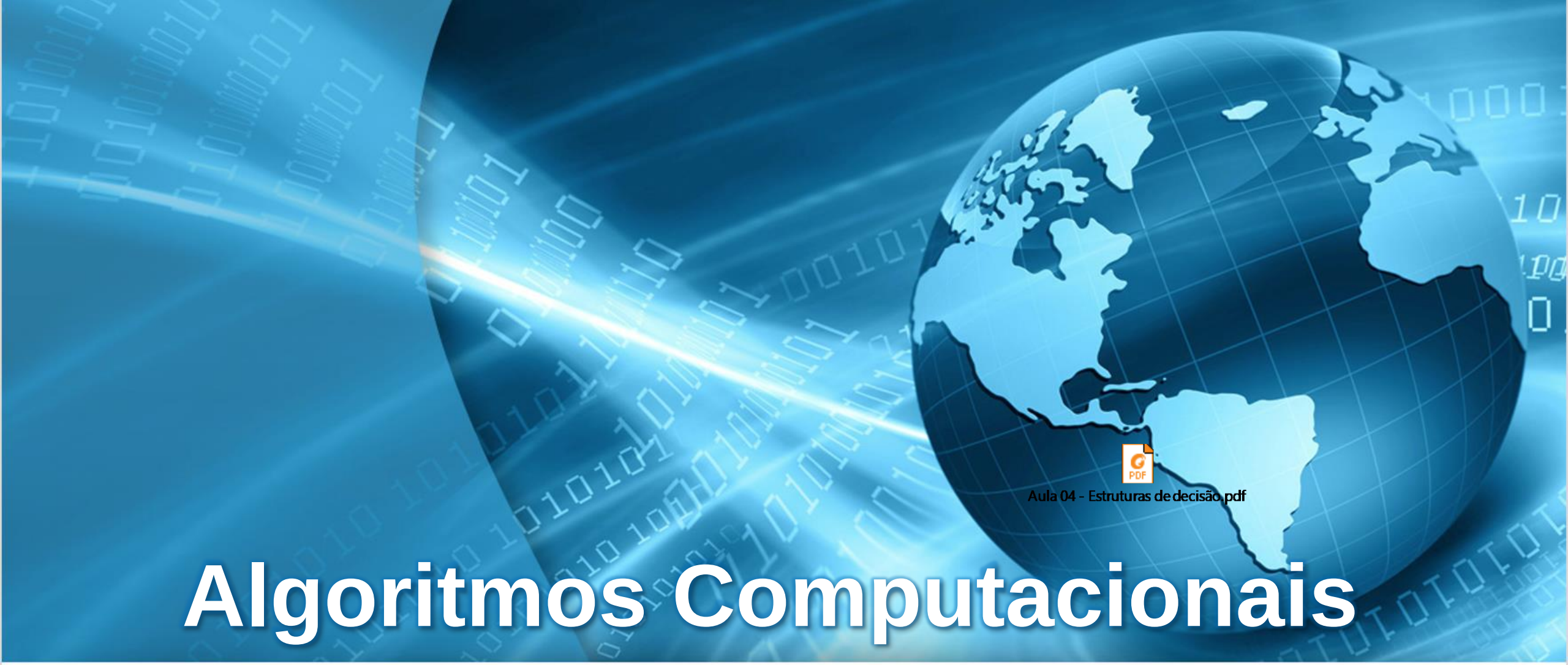

Algoritmos Computacionais

Conteúdo: Strings

Prof. Dsc. Giomar Sequeiros
giomar@eng.uerj.br

strings

Strings

- Uma string é uma sequência de caracteres. Elas são representadas usando aspas simples ('') ou aspas duplas (""). Exemplos:

```
string1 = 'Olá, mundo!'
string2 = "Eu sou uma string."
string3 = "12345"
string4 = 'Este é um exemplo de string com "aspas duplas" dentro dela.'

# Imprimindo as strings
print(string1)  # Saída: Olá, mundo!
print(string2)  # Saída: Eu sou uma string.
print(string3)  # Saída: 12345
print(string4)  # Saída: Este é um exemplo de string com "aspas duplas" dentro dela.
```

Strings: concatenação

- As strings em Python são imutáveis, o que significa que não podem ser modificadas após serem criadas.
- A operação de concatenação de strings é realizada utilizando o operador +. Ele permite combinar duas ou mais strings em uma única string.
- Exemplo:

```
# Concatenação de strings
string1 = 'Olá, '
string2 = 'mundo!'
concatenada = string1 + string2
print(concatenada)  # Saída: Olá, mundo!
```

Strings: concatenação

- As strings em Python são imutáveis, o que significa que não podem ser modificadas após serem criadas.
- A operação de concatenação de strings é realizada utilizando o operador +. Ele permite combinar duas ou mais strings em uma única string.
- Sintaxe:

```
string_concatenada = string1 + string2
```

- A variável string_concatenada irá conter o resultado da concatenação das strings string1 e string2.
- Também podemos concatenar mais de duas strings utilizando o operador + repetidamente:

```
string_concatenada = string1 + string2 + string3
```

Strings: concatenação

- Exemplos:

```
# Concatenação de strings
string1 = 'Olá, '
string2 = 'mundo!'
concatenada = string1 + string2
print(concatenada)  # Saída: Olá, mundo!
```

```
nome = "João"
sobrenome = "Silva"
mensagem = "Olá, " + nome + " " + sobrenome + "! Bem-vindo."
print(mensagem)
```

Strings: concatenação

- O operador * em Python pode ser usado para realizar a operação de repetição em strings. Ele permite criar uma nova string que consiste na repetição de uma string original por um determinado número de vezes.

- Aqui está a sintaxe do operador * em strings::

```
string_repetida = string * n
```

- Exemplo:

```
texto = "Olá! "
texto_repetido = texto * 3
print(texto_repetido)
```

- Observe que o valor de n precisa ser um número inteiro não negativo. Se n for zero ou negativo, o resultado será uma string vazia.

Strings: indexação

- A indexação em strings é uma operação que permite acessar caracteres individuais em uma string com base na sua posição. Em Python, os caracteres em uma string são indexados a partir do índice 0, ou seja, o primeiro caractere está na posição 0, o segundo caractere está na posição 1 e assim por diante.
- Sintaxe:

```
caractere = string[indice]
```

- Onde:
  - string é a string da qual você deseja obter o caractere,
  - indice é o número que representa a posição do caractere desejado.

Strings: indexação

- Exemplo

```
# Indexação de strings
string = 'Python'
print(string[0])    # Saída: 'P'
print(string[2])    # Saída: 't'
print(string[-1])   # Saída: 'n'
```

Strings: fatiamento

- O fatiamento (slicing) de strings é uma operação que permite extrair partes específicas de uma string. Com o fatiamento podemos obter uma subsequência de caracteres de uma string original com base nos índices de início, fim e passo especificados. Sintaxe

```
string[início:fim:passo]
```

- Onde:
  - **início**: Índice de início da fatia (inclusive). O caractere correspondente ao índice de início será incluído na fatia resultante.
  - **fim**: Índice de fim da fatia (exclusivo). O caractere correspondente ao índice de fim não será incluído na fatia resultante.
  - **passo**: Tamanho do passo (opcional). O valor padrão é 1, o que significa que todos os caracteres no intervalo serão incluídos. Um passo de 2 significa que a cada segundo caractere será incluído, e assim por diante.

Strings: fatiamento

- Exemplos:

```
# Fatiamento de strings
string = 'Python é uma linguagem de programação'
print(string[7:9])     # Saída: 'é'
print(string[0:6])     # Saída: 'Python'
print(string[:6])      # Saída: 'Python'
print(string[14:])     # Saída: 'linguagem de programação'
print(string[::2])     # Saída: 'Ptoéua lnuae epgaaç'
```

Strings: fatiamento

- Mais exemplos:

```
string = "Python é uma linguagem de programação"

# Obter os primeiros 6 caracteres da string
print(string[:6])  # Saída: "Python"

# Obter os caracteres do índice 7 até o 12 (exclusivo)
print(string[7:12])  # Saída: "é uma"

# Obter os últimos 10 caracteres da string
print(string[-10:])  # Saída: "programação"

# Obter os caracteres em posições pares
print(string[::2])  # Saída: "Pto éua lnaemdepormo"

# Obter a string reversa
print(string[::-1])  # Saída: "ãçãoamargorp ed meganuil amu é nohtyP"
```

Exemplo 1

- Escreva uma função **contar_vogais(texto)** que receba uma string como parâmetro e retorne o número de vogais presentes nessa string. A função deve considerar tanto letras maiúsculas quanto minúsculas.

```
def contar_vogais(texto):
    """
    Retorna o número de vogais numa cadeia
    """
    i = 0
    cont = 0 # contador de vogais
    while i < len(texto):
        if texto[i] in 'aeiouAEIOU':
            cont +=1
        i+=1

    return cont
```

Exemplo 2

- Escreva uma função **inverter_string(texto)** que receba uma string como parâmetro e retorne a string invertida, ou seja, os caracteres em ordem inversa.

```
def inverter_string(texto):
    """
    Retorna uma string em ordem inversa
    """
    saida = ''
    i = len(texto)-1
    while i >=0:
        saida += texto[i]
        i-=1
    return saida
```

Outra solução

```
def inverter_string_v2(texto):
    """
    Inverte uma string usando fatiamento
    """
    texto_invertido = texto[::-1]

    return texto_invertido
```

Exemplo 3

- Escreva uma função **validar_palindromo(texto)** que receba uma string como parâmetro e verifique se ela é um palíndromo. A função deve retornar True se a string for um palíndromo e False caso contrário. A função deve ignorar espaços em branco.

```
def validar_palindromo(texto):
    """
    Retorna uma string invertida
    """
    return remover_espacos(texto) == remover_espacos(inverter_string(texto))
```

Exercício 1

- Escreva uma função **remover_vogais(texto)** que receba uma string como parâmetro e retorne a mesma string, porém sem vogais.

```
def remover_vogais(texto):
    """
    Retorna uma string com as vogais removidas
    """
    saida = ''
    i = 0
    while i < len(texto):
        if texto[i] not in 'aeiouAEIOU':
            saida += texto[i]
        i+=1

    return saida
```

Exercício 2

- Escreva uma função **remover_espacos(texto)** que receba uma string como parâmetro e retorne a mesma string, porém sem espaços em branco.

```
def remover_espacos(texto):
    """
    Retorna uma string sem espaços
    """
    saida = ''
    i = 0
    while i < len(texto):
        if texto[i] != ' ':
            saida += texto[i]
        i+=1

    return saida
```

Comando in

- O comando **in** é utilizado para verificar se um determinado valor está presente em uma sequência, como uma string. Ele retorna True se o valor estiver presente e False caso contrário. Exemplo

```
texto = "Olá, mundo!"
if "a" in texto:
    print("A letra 'a' está presente no texto.")
```

```
frase = "A vida é bela!"
if "vida" in frase:
    print("A palavra 'vida' está presente na frase.")
```

Exercício 2

- Escreva uma função **contar_ocorrencias(texto, palavra)** que receba uma string texto e uma palavra palavra como parâmetros, e retorne o número de vezes que a palavra ocorre no texto. A função deve ser case-insensitive, ou seja, não deve fazer distinção entre letras maiúsculas e minúsculas.

Funções úteis strings

- As strings em Python possuem vários métodos embutidos que permitem realizar diversas operações e manipulações. Algumas são:
  - **len(string):** Retorna o comprimento da string, ou seja, o número de caracteres que ela contém.
  - **string.lower():** Retorna uma nova string com todos os caracteres em minúsculas.
  - **string.upper():** Retorna uma nova string com todos os caracteres em maiúsculas.
  - **string.strip():** Retorna uma nova string com os espaços em branco removidos do início e do final da string.
  - **string.split(separador):** Divide a string em substrings com base em um separador especificado e retorna uma lista das substrings resultantes.
  - **string.replace(antigo, novo):** Substitui todas as ocorrências de uma substring "antigo" por uma nova substring "novo" e retorna a string resultante.
  - **string.find(substring):** Retorna o índice da primeira ocorrência de uma substring na string. Se a substring não for encontrada, retorna -1.
  - **string.startswith(prefixo):** Verifica se a string começa com um determinado prefixo e retorna True ou False.
  - **string.endswith(sufixo):** Verifica se a string termina com um determinado sufixo e retorna True ou False.

- Ver documentação para mais: https://docs.python.org/3/library/stdtypes.html#string-methods

Funções úteis strings - exemplos

- **len(string):** Retorna o comprimento da string.

```
texto = "Olá, mundo!"
comprimento = len(texto)
print(comprimento)  # Saída: 13
```

- **string.lower():** Retorna uma nova string com todos os caracteres em minúsculas.

```
texto = "Olá, Mundo!"
texto_min = texto.lower()
print(texto_min)  # Saída: olá, mundo!
```

Funções úteis strings - exemplos

- **string.upper():** Retorna uma nova string com todos os caracteres em maiúsculas.

```
texto = "Olá, Mundo!"
texto_up = texto.upper()
print(texto_up)  # Saída: OLÁ, MUNDO!
```

- **string.strip():** Retorna uma nova string com os espaços em branco removidos do início e do final da string.

```
texto = "   Olá, mundo!   "
texto_strip = texto.strip()
print(texto_strip)  # Saída: Olá, mundo!
```

Funções úteis strings - exemplos

- **string.split(separador):** Divide a string em substrings com base em um separador especificado e retorna uma lista das substrings resultantes.

```
frase = "Python é uma linguagem de programação"
palavras = frase.split(" ")
print(palavras)  # Saída: ['Python', 'é', 'uma', 'linguagem', 'de', 'programação']
```

- **string.replace(antigo, novo):** Substitui todas as ocorrências de uma substring "antigo" por uma nova substring "novo" e retorna a string resultante.

```
texto = "Olá, mundo!"
texto_novo = texto.replace("mundo", "Python")
print(texto_novo)  # Saída: Olá, Python!
```

Funções úteis strings - exemplos

- **string.find(substring):** Retorna o índice da primeira ocorrência de uma substring na string. Se a substring não for encontrada, retorna -1.

```
texto = "Olá, mundo!"
indice = texto.find("mundo")
print(indice)  # Saída: 5
```

- **string.startswith(prefixo):** Verifica se a string começa com um determinado prefixo e retorna True ou False.

```
texto = "Olá, mundo!"
resultado = texto.startswith("Olá")
print(resultado)  # Saída: True
```

- **string.endswith(sufixo):** Verifica se a string termina com um determinado sufixo e retorna True ou False.

```
texto = "Olá, mundo!"
resultado = texto.endswith("!")
print(resultado)  # Saída: False
```

Exemplo 1

- Implemente uma função remover_espacos(texto) que receba uma string como parâmetro e retorne a mesma string, porém sem espaços em branco.

```
def remover_espacos(texto):
    """
    Remove os espaços em branco de uma string.
    """
    return texto.replace(" ", "")
```

Exemplo 2

- Crie uma função contar_ocorrencias(texto, palavra) que receba uma string texto e uma palavra palavra como parâmetros, e retorne o número de vezes que a palavra ocorre no texto. A função deve ser case-insensitive, ou seja, não deve fazer distinção entre letras maiúsculas e minúsculas.

```
def contar_ocorrencias(texto, palavra):
    """
    Conta o número de ocorrências de uma palavra em uma string.
    A contagem é case-insensitive, ou seja, não faz distinção entre letras
    maiúsculas e minúsculas.
    """
    texto = texto.lower()
    palavra = palavra.lower()
    return texto.count(palavra)
```

Exercício 1

- Escreva uma função remover_caracteres_repetidos(texto) que remova todos os caracteres que aparecem mais de uma vez consecutiva em um texto. Por exemplo, se o texto for "aabbbccccddee", o resultado deverá ser "abcde".

Exercício 1 - solução

- Escreva uma função remover_caracteres_repetidos(texto) que remova todos os caracteres que aparecem mais de uma vez consecutiva em um texto. Por exemplo, se o texto for "aabbbccccddee", o resultado deverá ser "abcde".

```
def remover_caracteres_repetidos(texto):
    """
    Remove caracteres consecutivos repetidos em um texto.
    """
    resultado = ""
    tamanho = len(texto)
    i = 0
    while i < tamanho:
        resultado += texto[i]  # Adiciona o caractere atual ao resultado
        # Verifica se o próximo caractere é igual ao atual
        while i < tamanho - 1 and texto[i] == texto[i + 1]:
            i += 1 # Se for igual, avança para o próximo caractere

        i += 1  # Passa para o próximo caractere

    return resultado
```

Exercício 1 - solução

- Teste

```
texto = "aabbbccccddee"
resultado =
remover_caracteres_repetidos(texto)
print(resultado)   # Saída: "abcde"
```

Desafio

- Crie uma função cifrar_texto(texto, chave) que cifre um texto usando a Cifra de César. A cifra deve deslocar cada letra do texto original para a direita de acordo com o valor da chave. Por exemplo, se a chave for 3, a letra 'a' será substituída por 'd', 'b' por 'e', e assim por diante.

- Exemplo para o texto **“Olá, mundo!”** e chave **3** a saida será: **"Rod, pxqgr!"**