Módulo Strings

Curso Estruturas de Dados e Algoritmos Expert

Prof. Dr. Nelio Alves
https://devsuperior.com.br

Sobre este módulo

Pré-requisitos:

- Conhecimento básico de Lógica de Programação
    - Estrutura sequencial
    - Estrutura condicional (if-else)
    - Estruturas repetitivas (while, for)
    - Arrays
    - Funções

Objetivos:

- Treinar resolução de problemas envolvendo strings.

Nota:

- Dentre os exercícios, alguns são do Leetcode, já usados por empresas tais como Amazon, Adobe, Google, Apple e Meta.

Aviso sobre os exercícios

Ao longo deste curso, dentre os exercícios com que vamos trabalhar, alguns são desafios de entrevistas, utilizados inclusive por grandes empresas como Amazon, Google, Meta, Microsoft, dentre outras.

Porém, como este capítulo sobre strings está no início do curso, momento em que não vimos ainda técnicas específicas, os exercícios sobre strings aqui neste capítulo terão que ser mais básicos, sem entrar ainda em outras técnicas específicas.

Mais à frente no curso, teremos outros exercícios sobre strings, onde poderemos resolver problemas que combinam strings com outras técnicas de programação.

Introdução à programação com strings

As strings, sequências de caracteres que representam texto, são a base através da qual a informação é transmitida, armazenada e manipulada em quase todos os sistemas computacionais. Algumas aplicações comuns são:

- Validação de dados de formulários
- Exibição de mensagens customizadas
- Comunicação entre sistemas
    - Protocolo HTTP
    - JSON
    - XML
    - CSV
- Compiladores
- Processamento de linguagem natural

Ao dominar a programação com strings, você não apenas ampliará significativamente suas habilidades de programação, mas também abrirá portas para novas oportunidades e desafios. Um sólido entendimento de como trabalhar com strings será um dos seus ativos mais valiosos.

Referência: strings em Javascript

Literais e expressões

Em Javascript, strings literais são representados como texto entre aspas duplas, aspas simples ou crases. Pode-se construir também strings com literais e expressões, por meio de concatenação e template literais (interpolação). Exemplos:

https://github.com/devsuperior/curso-eda/blob/main/strings/javascript/demo-literals.js

Imutabilidade

Em JavaScript, strings são imutáveis. Isso significa que, uma vez que uma string é criada, não é possível alterar seus caracteres individuais diretamente. Qualquer operação que pareça modificar uma string na verdade cria e retorna uma nova string.

Para lidar com problemas que precisam alterar caracteres em uma string, pode-se converter a string para um array de caracteres, daí fazer as operações necessárias, e ao final converter novamente o array para uma nova string.

```
let str = "Hello, world!";
let arr = str.split(""); // Converte a string em um array de caracteres*
arr[7] = 'W'; // Modifica um caracter* específico
str = arr.join(""); // Converte o array de volta para string
```

H	e	l	l	o	,		w	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

* Nota: em Javascript na verdade não existe o tipo caractere (como existe em Java e C# por exemplo). O array criado pela chamada do método `.split("")` na verdade será um array de strings, onde cada string será um string contendo um único caractere.

Funções de string

https://github.com/devsuperior/curso-eda/blob/main/strings/javascript/demo.js

concat
Combina o texto de duas ou mais strings e retorna uma nova string.

indexOf
Retorna o índice da primeira ocorrência de um valor especificado em uma string, ou -1 se não for encontrado.

lastIndexOf

Retorna o índice da última ocorrência de um valor especificado em uma string, ou -1 se não for encontrado.

match
Procura uma string por uma correspondência contra uma expressão regular e retorna as correspondências como um array de objetos.

matchAll
Retorna um iterador com todas as correspondências de uma string contra uma expressão regular.

replace
Substitui algumas ou todas as ocorrências de um padrão por uma substituição.

replaceAll
Similar ao replace(), mas substitui todas as ocorrências do padrão.

search
Procura uma string por uma correspondência contra uma expressão regular e retorna o índice da correspondência.

split
Divide uma string em um array de strings usando um separador especificado.

slice
Extrai uma parte de uma string e retorna uma nova string.

substring
Retorna uma parte da string entre os índices de início e fim.

toLowerCase
Converte uma string para letras minúsculas.

toUpperCase
Converte uma string para letras maiúsculas.

trim
Remove espaços em branco do início e do fim de uma string.

trimStart
Remove espaços em branco do início de uma string.

trimEnd
Remove espaços em branco do final de uma string.

charAt
Retorna o caractere em um índice específico.

charCodeAt
Retorna um número indicando o valor Unicode do caractere em um determinado índice.

startsWith
Verifica se uma string começa com os caracteres de uma string especificada, retornando true ou false.

endsWith
Verifica se uma string termina com os caracteres de uma string especificada, retornando true ou false.

includes
Verifica se uma string contém os caracteres de uma string especificada, retornando true ou false.

padStart
Preenche a string original com um determinado caractere (ou conjunto de caracteres) no início da string até alcançar o comprimento desejado.

padEnd
Preenche a string original com um determinado caractere (ou conjunto de caracteres) no final da string até alcançar o comprimento desejado.

Referência: strings em Java

Literais e expressões

Em Java, strings literais são representadas como texto entre aspas duplas. Pode-se construir também strings com literais e expressões, por meio de concatenação, formatação, ou com a classes StringBuilder (ou StringBuffer). Exemplos:

https://github.com/devsuperior/curso-eda/blob/main/strings/java/src/DemoLiterals.java

Imutabilidade

Em Java, strings são imutáveis. Isso significa que, uma vez que uma string é criada, não é possível alterar seus caracteres individuais diretamente. Qualquer operação que pareça modificar uma string na verdade cria e retorna uma nova string.

Para lidar com problemas que precisam alterar caracteres em uma string, pode-se converter a string para um array de caracteres, daí fazer as operações necessárias, e ao final converter novamente o array para uma nova string.

```
String str = "Hello, world!";
char[] arr = str.toCharArray(); // Converte a string em um array de caracteres
arr[7] = 'W'; // Modifica um caracter específico
str = new String(arr); // Converte o array de volta para string
```

Funções de string

Funções da classe String

https://github.com/devsuperior/curso-eda/blob/main/strings/java/demo-string1.java
https://github.com/devsuperior/curso-eda/blob/main/strings/java/demo-string2.java

Informações Básicas:

length(): Retorna o comprimento da string.

isEmpty(): Verifica se a string está vazia.

Comparação:

equals(Object anObject): Compara esta string com o objeto especificado.

equalsIgnoreCase(String anotherString): Compara esta String com outra String, ignorando considerações de caso.

compareTo(String anotherString): Compara duas strings lexicograficamente.

compareToIgnoreCase(String str): Compara duas strings lexicograficamente, ignorando diferenças de caso.

startsWith(String prefix): Verifica se a string começa com o prefixo especificado.

endsWith(String suffix): Verifica se a string termina com o sufixo especificado.

contains(CharSequence s): Retorna true se e somente se esta string contiver a sequência de caracteres especificada.

Busca:

indexOf(int ch): Retorna o índice da primeira ocorrência do caractere especificado.

indexOf(int ch, int fromIndex): Retorna o índice da primeira ocorrência do caractere especificado, começando a busca no índice especificado.

indexOf(String str): Retorna o índice da primeira ocorrência da substring especificada.

indexOf(String str, int fromIndex): Retorna o índice da primeira ocorrência da substring especificada, começando a busca no índice especificado.

lastIndexOf(int ch): Retorna o índice da última ocorrência do caractere especificado.

lastIndexOf(int ch, int fromIndex): Retorna o índice da última ocorrência do caractere especificado, procurando para trás a partir do índice especificado.

lastIndexOf(String str): Retorna o índice da última ocorrência da substring especificada.

lastIndexOf(String str, int fromIndex): Retorna o índice da última ocorrência da substring especificada, procurando para trás a partir do índice especificado.

Extração:

charAt(int index): Retorna o caractere no índice especificado.

substring(int beginIndex): Retorna uma nova string que é uma substring desta string.

substring(int beginIndex, int endIndex): Retorna uma nova string que é uma substring desta string, começando do índice inicial até o final do índice especificado.

split(String regex): Divide esta string ao redor das correspondências da expressão regular dada.

split(String regex, int limit): Divide esta string ao redor das correspondências da expressão regular dada com um limite no número de substrings resultantes.

Modificação:

trim(): Retorna uma cópia da string, com espaços em branco iniciais e finais omitidos.

toLowerCase(): Converte todos os caracteres da string para minúsculas.

toUpperCase(): Converte todos os caracteres da string para maiúsculas.

replace(char oldChar, char newChar): Retorna uma nova string resultante da substituição de todos os caracteres antigos na string por novos caracteres.

replaceAll(String regex, String replacement): Substitui cada substring desta string que corresponde à expressão regular dada com a substituição dada.

replaceFirst(String regex, String replacement): Substitui a primeira substring desta string que corresponde à expressão regular dada com a substituição dada.

Conversão:

toCharArray(): Converte esta string para um novo array de caracteres.

valueOf(Various Types): Retorna a representação em string do argumento passado (sobrecarregada para vários tipos).

Outros:

matches(String regex): Diz se ou não esta string corresponde à expressão regular dada.

concat(String str): Concatena a string especificada ao final desta string.

String.join(delimiter, elements): Retorna uma string que será a sequência dos elementos dados, usando o delimiter como separador.

String.format(Locale l, String format, Object... args): Retorna uma string formatada.

Funções da classe StringBuilder ou StringBuffer

Em Java, StringBuilder e StringBuffer são classes que permitem construir strings de forma eficiente. StringBuilder é mais rápido, porém não é thread-safe. StringBuffer é mais lento, porém é thread-safe.

append(String str)
Adiciona a string especificada ao final do string builder.

insert(int offset, String str)
Insere a string no string builder na posição dada.

replace(int start, int end, String str)
Substitui a parte da string entre os índices start e end pela string dada.

delete(int start, int end)
Remove os caracteres entre os índices start e end.

reverse()
Inverte a sequência de caracteres no string builder.

Referência: strings em Python

Literais e expressões

Em Python, strings literais são representadas como texto entre aspas duplas ou aspas simples. Pode-se construir também strings com literais e expressões, por meio de concatenação e interpolação. Exemplos:

https://github.com/devsuperior/curso-eda/blob/main/strings/python/demo-literals.py

Imutabilidade

Em Python, strings são imutáveis. Isso significa que, uma vez que uma string é criada, não é possível alterar seus caracteres individuais diretamente. Qualquer operação que pareça modificar uma string na verdade cria e retorna uma nova string.

Para lidar com problemas que precisam alterar caracteres em uma string, pode-se converter a string para uma lista de caracteres, daí fazer as operações necessárias, e ao final converter novamente a lista para uma nova string.

```
my_str = "Hello, world!"
my_list = list(my_str) // Converte a string em uma lista de caracteres*
my_list[7] = 'W' // Modifica um caracter* específico
my_str = ''.join(my_list) // Converte o array de volta para string
```

* Nota: em Python na verdade não existe o tipo caractere (como existe em Java e C# por exemplo). A lista criada pela chamada da função `list(..)` na verdade será uma lista de strings, onde cada string será um string contendo um único caractere.

Funções de string

https://github.com/devsuperior/curso-eda/blob/main/strings/python/demo.py

Funções build-in

len(string):
Retorna o comprimento de uma string.

str(object):
Converte um objeto em string.

format(value, format_spec):
Formata um valor de acordo com uma especificação de formato.

print(string):

Imprime a string no console.

Métodos de string

capitalize():
Capitaliza a primeira letra da string.

upper():
Converte todos os caracteres da string para maiúsculas.

lower():
Converte todos os caracteres da string para minúsculas.

title():
Converte a primeira letra de cada palavra para maiúscula.

swapcase():
Inverte a caixa de todos os caracteres na string.

strip([chars]):
Remove espaços em branco (ou caracteres especificados) do início e do fim da string.

rstrip([chars]):
Remove espaços em branco (ou caracteres especificados) do final da string.

lstrip([chars]):
Remove espaços em branco (ou caracteres especificados) do início da string.

find(sub[, start[, end]]):
Retorna o índice mais baixo na string onde substring sub é encontrado.

rfind(sub[, start[, end]]):
Retorna o índice mais alto onde a substring sub é encontrado.

index(sub[, start[, end]]):
Similar ao find(), mas lança uma exceção se sub não for encontrado.

rindex(sub[, start[, end]]):
Similar ao rfind(), mas lança uma exceção se sub não for encontrado.

count(sub[, start[, end]]):
Conta o número de ocorrências de uma substring.

replace(old, new[, count]):
Substitui todas as ocorrências da substring old por new.

split(sep=None, maxsplit=-1):

Divide a string em uma lista de strings usando sep como delimitador.

rsplit(sep=None, maxsplit=-1):
Divide a string da direita para a esquerda.

join(iterable):
Junta os elementos de um iterável em uma única string, intercalada por uma string separadora.

startswith(prefix[, start[, end]]):
Retorna True se a string começar com o prefixo especificado.

endswith(suffix[, start[, end]]):
Retorna True se a string terminar com o sufixo especificado.

isalpha():
Retorna True se todos os caracteres na string são alfabéticos.

isdigit():
Retorna True se todos os caracteres na string são dígitos.

isspace(): Retorna
True se todos os caracteres na string são espaços em branco.

islower():
Retorna True se todos os caracteres em letras na string são minúsculas.

isupper():
Retorna True se todos os caracteres em letras na string são maiúsculas.

istitle():
Retorna True se a string é um título.

isnumeric():
Retorna True se todos os caracteres na string são numéricos.

isdecimal():
Retorna True se todos os caracteres na string são decimais.

Referência: strings em C#

Literais e expressões

Em C#, strings literais são representadas como texto entre aspas duplas. Pode-se construir também strings com literais e expressões, por meio de concatenação, interpolação, ou com a classes StringBuilder. Exemplos:

https://github.com/devsuperior/curso-eda/tree/main/strings/csharp/demo-literals

Imutabilidade

Em C#, strings são imutáveis. Isso significa que, uma vez que uma string é criada, não é possível alterar seus caracteres individuais diretamente. Qualquer operação que pareça modificar uma string na verdade cria e retorna uma nova string.

Para lidar com problemas que precisam alterar caracteres em uma string, pode-se converter a string para um array de caracteres, daí fazer as operações necessárias, e ao final converter novamente o array para uma nova string.

```
string str = "Hello, world!";
char[] arr = str.ToCharArray(); // Converte a string em um array de caracteres
arr[7] = 'W'; // Modifica um caracter específico
str = new string(arr); // Converte o array de volta para string
```

Funções de strings

https://github.com/devsuperior/curso-eda/tree/main/strings/csharp/demo

Funções da classe String

string.Concat:
Concatena duas ou mais strings.

string.Compare:
Compara duas strings e retorna um valor indicando sua relação.

string.Contains:
Verifica se a string contém uma substring especificada.

string.EndsWith:
Verifica se a string termina com uma substring especificada.

string.IndexOf:
Retorna a posição de índice da primeira ocorrência de uma substring.

string.LastIndexOf:
Retorna a posição de índice da última ocorrência de uma substring.

string.Insert:
Insere uma string em uma posição especificada.

string.Remove:
Remove uma série de caracteres de uma string.

string.Replace:
Substitui todas as ocorrências de uma substring por outra substring.

string.Split:
Divide uma string em um array de strings com base em delimitadores especificados.

string.StartsWith:
Verifica se a string começa com uma substring especificada.

string.Substring:
Retorna uma substring que começa em uma posição de caractere especificada.

string.ToLower:
Converte a string para letras minúsculas.

string.ToUpper:
Converte a string para letras maiúsculas.

string.Trim:
Remove todos os caracteres de espaço em branco do início e do fim da string.

string.TrimStart:
Remove todos os caracteres de espaço em branco do início da string.

string.TrimEnd:
Remove todos os caracteres de espaço em branco do final da string.

Funções da classe StringBuilder

Em C#, StringBuilder é uma classe que permite construir strings de forma eficiente.

Append:
Acrescenta uma representação de string de um objeto especificado ao final desta instância.

Insert:
Insere uma string ou representação de objeto em uma posição especificada da instância atual.

Remove:
Remove um intervalo de caracteres da instância atual.

Replace:
Substitui todas as ocorrências de uma substring por outra substring dentro da instância atual.

Expressões regulares

Definição e exemplos iniciais

Expressões regulares, ou abreviadamente em Inglês: regex, oferecem uma maneira poderosa de validar, substituir ou pesquisar **padrões de texto**.

As expressões regulares equivalem aos autômatos finitos, que é um formalismo capaz de especificar linguagens regulares, bem como verificar se alguma sentença pertence a essa linguagem. Linguagens regulares é o tipo mais simples de linguagem formal na hierarquia de Chomsky.

Para adquirir a competência de se elaborar expressões regulares complexas, é preciso bastante estudo (o que pode corresponder a boa parte de uma disciplina de faculdade), mais um tempo de experiência prática. Entretanto, é possível que um programador lide com expressões regulares em seu cotidiano, mesmo não sendo um especialista nesta área, desde que saiba encontrar o que precisa na web e nas ferramentas de IA, e que saiba os fundamentos básicos para conferir e testar as regex.

Dica de ChatGPT para outras linguagens: peça a expressão ou código equivalente na outra linguagem desejada.

Exemplo 1: queremos verificar se um CEP é valido, ou seja, se ele possui 8 dígitos, podendo ou não ter um hífen entre o quinto e sexto dígito.

```
function validateCEP(cep) {
  const regex = /^\d{5}-?\d{3}$/;
  return regex.test(cep);
}

console.log(validateCEP("12345-678")); // true
console.log(validateCEP("12345678")); // true
console.log(validateCEP("1234-5678")); // false
```

Exemplo 2: queremos remover de um CPF qualquer caractere que não seja um dígito.

```
function removeNonDigits(string) {
    const regex = /\D/g;
    return string.replace(regex, "");
}

console.log(removeNonDigits("94923784799"));
console.log(removeNonDigits("213.445.034-82"));
```

Exemplo 3: verifique se um dado domínio termina com .br

```
function validateBrDomain(domain) {
    const regex = /\.br$/;
    return regex.test(domain);
}

console.log(validateBrDomain("batata.com.br")); // true
console.log(validateBrDomain("banana.com"));  // false
```

Exemplo 4: encontre todos endereços de email em um texto.

```
function findEmails(string) {
    const regex = /\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b/g;
    const result = string.match(regex);
    if (result) {
        return result;
    }
    return [];
}

const text = "Para mais informações, contate-nos em contato@exemplo.com ou
suporte@exemplo.com.br.";

const emails = findEmails(text);

if (emails.length > 0) {
    console.log("Emails encontrados:");
    for (let i = 0; i < emails.length; i++) {
        console.log(emails[i]);
    }
}
else {
    console.log("Nenhum email encontrado.")
}
```

Referência sobre regex

Caracteres Especiais

.
Corresponde a qualquer caractere, exceto quebra de linha. Por exemplo, /.at/ corresponde a "cat", "bat", e "rat".

\d
Corresponde a qualquer dígito de 0 a 9.

\D
Corresponde a qualquer caractere que não seja um dígito de 0 a 9.

\w
Corresponde a qualquer caractere alfanumérico, incluindo o sublinhado (_). Equivalente a [A-Za-z0-9_].

\W
Corresponde a qualquer caractere que não seja alfanumérico.

\s
Corresponde a qualquer caractere de espaço em branco, incluindo espaço, tabulação, alimentação de formulário e quebras de linha.

\S
Corresponde a qualquer caractere que não seja de espaço em branco.

\b
Corresponde a uma fronteira de palavra, como o espaço entre duas palavras.

\B
Corresponde a uma posição que não é uma fronteira de palavra.

^
Corresponde ao início de uma string, ou ao início de uma linha se a flag m for utilizada.

$
Corresponde ao final de uma string, ou ao final de uma linha se a flag m for utilizada.

[]
Usado para especificar um conjunto de caracteres. Por exemplo, [abc] corresponde a qualquer um dos caracteres "a", "b" ou "c".

Quantificadores

*
Corresponde a zero ou mais ocorrências do elemento anterior.

+
Corresponde a uma ou mais ocorrências do elemento anterior.

?
Corresponde a zero ou uma ocorrência do elemento anterior.

{n}
Corresponde exatamente a n ocorrências do elemento anterior.

{n,}
Corresponde a n ou mais ocorrências do elemento anterior.

{n,m}
Corresponde de n a m ocorrências do elemento anterior, sendo ambas inclusivas.

Caracteres de Escape

\
É usado para escapar um metacaractere, permitindo que você corresponda ao próprio caractere, como \. para corresponder a um ponto literal, ou \/ para corresponder a uma barra, ou \\ para corresponder a uma barra invertida.

Grupos e Intervalos

()
Usado para agrupar caracteres e capturar o texto correspondente.

(?:)
Agrupa caracteres sem capturar o texto correspondente.

|
Operador OR, corresponde a qualquer um dos padrões separados por ele.

Flags

g
Realiza uma correspondência global, ou seja, encontra todas as correspondências, em vez de parar após a primeira correspondência.

i
Torna a correspondência insensível a maiúsculas e minúsculas.

m
Multilinha, faz com que ^ e $ correspondam ao início e ao fim de cada linha, em vez de apenas o início e o fim de toda a string.

s
Permite que o . corresponda também a quebras de linha.

u
Trata o padrão como uma sequência Unicode.

y
Realiza uma correspondência "sticky", buscando correspondências apenas a partir do índice indicado pela propriedade lastIndex do objeto RegExp.

Algumas regex comuns

Exemplo de prompt para ChatGPT:
Elabore uma regex para telefone no padrão do Brasil. Explique o padrão adotado.

Atenção: repare que todas as regex a seguir começam e terminam com **^** e **$** respectivamente. Isso significa que a string toda deverá ser verificada para o padrão da regex, não podendo conter outros conteúdos antes e depois do padrão.

Email (simplificada)
```
/^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$/
```

URL (simplificada)
```
/^((https?|ftp):\/\/)?(([a-z0-9-]+\.)+[a-z]{2,}|((\d{1,3}\.){3}\d{1,3}))(:(\d{2,5}))?(\/[a-z0-9-\.\/?#@&%;=~_|]*)?$/
```

Telefone brasileiro
```
/^(?:\+55\s?)?(?:\([1-9]{2}\)|[1-9]{2})\s?(?:9\s?)?[6789][0-9]{3}\s?-?\s?[0-9]{4}$/
```

CEP brasileiro
```
/^\d{5}-?\d{3}$/
```

Data no formato aaaa-mm-dd (simplificada)
```
/^\d{4}-\d{2}-\d{2}$/
```

Data no formato dd/mm/aaaa (com validação de dia entre 1 e 31, e mês de 1 a 12, podendo ou não ter zero à esquerda dos valores menores que 10)

```
/^(0?[1-9]|[12][0-9]|3[01])\/(0?[1-9]|1[012])\/(19|20)\d\d$/
```

Senha “forte” (pelo menos 8 caracteres, com pelo menos uma letra maiúscula, uma minúscula, um número e um caractere especial)

```
/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/
```

Número inteiro

```
/^[-+]?\d+$/
```

Número de ponto flutuante

```
/^[-+]?\d*\.?\d+$/
```

IPv4

```
/^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/
```