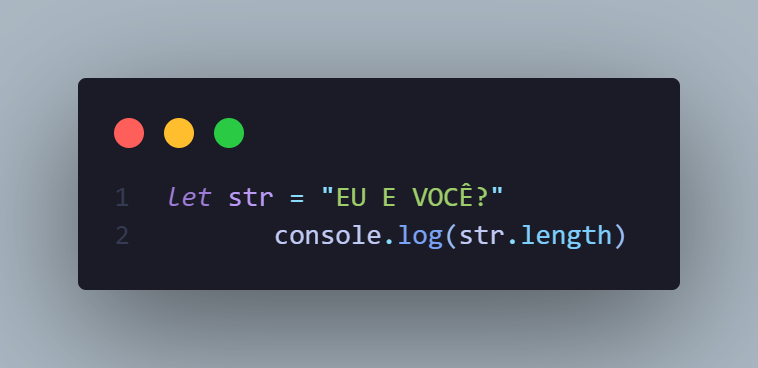
Manipulando Strings

As string na linguagem de programação desempenham um papel muito importa, sendo usada para armazenar e manipular texto, antes de entender como e quais são as formas de manipular as Strings, precisamos entender que cada caracteres dentro de uma string ocupa uma posição sequencial, a string funciona com o deslocamento 0, que seria que ela começa a contar a posição a partir de 0, ou seja 0,1,2 ...etc. segue exemplo abaixo:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | U |  | E |  | V | O | C | Ê | ? |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Nesse exemplo vemos que temos letras, espaços e caracteres especiais e todos eles entram no deslocamento 0, contando até os espaços, agora que entendemos esse conceito, vamos ver a propriedade **length**, essa propriedade conta quantos caracteres tem na string, usando o deslocamento em 1, ou seja ele começa a contar do numeral 1, que é o método comum de contagem, segue abaixo um exemplo de como usar o **lenght:**

****

Nesse exemplo estamos definindo uma variável e guardamos o valor de uma string dentro dela, logo usamos o console.log que vai exibir str.length, ou seja ele vai exibir o número de caracteres armazenados na variável, que são dez

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | U |  | E |  | V | O | C | Ê | ? |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Extraindo Caracteres da String

* at()
* charAt()
* charCodeAt
* Usando acesso de propriedade [] como array

**at:** O método at() foi introduzido na versão ECMAScript 2022 (ES13) e funciona com um único parâmetro que representa a posição do caractere que desejamos acessar. O deslocamento (ou índice) começa em **0**, ou seja, o primeiro caractere da string está na posição 0.

Nesse caso retornará o E para nós porque ele é correspondente a 0,Podemos passar também números negativos como parâmetro, que contam a partir do final da string. Por exemplo, -1 refere-se ao último caractere, -2 refere-se ao penúltimo, e assim por diante, Mas se tentarmos acessar um índice maior ou igual ao comprimento da string (ou array), o método retornará undefined, sendo negativo ou positivo

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | U |  | E |  | V | O | C | Ê | ? |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

**charAt()**: o chartAt faz o mesmo que at, a diferença é que como ele é mais antigo ele é aceito por um número maiores de navegadores, não aceita número negativo, e se tentarmos acessar um índice maior ou igual ao comprimento da string (ou array), o método retornará uma string vazia, diferente do at

**charCodeAt()**: o charCodeAt retorna um número inteiro representando o valor UTF-16 do caractere em uma posição específica de uma string

**Usando acesso de propriedade [ ]:** As [ ] funcionam igual ao charAt() podendo retorna os caracteres correspondendo ao número do índice 0, A diferença principal é que, ao acessar com colchetes, se o índice for inválido (fora do comprimento da string), ele retornará undefined, enquanto o charAt() retornará uma string vazia "":

Extraindo Partes de String

Existem 3 métodos para extrair uma parte de uma string

* Slice(start, end)
* Substring(start, end)
* Substr(start, Length)

**Slice:** o slice funciona da seguinte maneira, você passa dois parâmetros para ele de início e fim, ele vai começar a mostrar no primeiro parâmetro e terminar antes do parâmetro final.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | U |  | E |  | V | O | C | Ê | ? |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

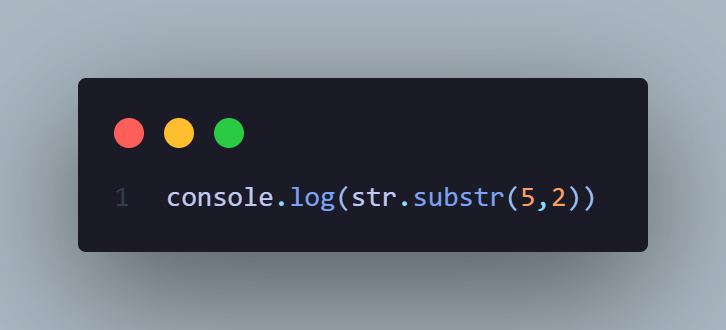
Levando o exemplo acima em consideração, se fizermos o



Ele irá mostrar “Você” começando a mostrar no 5 e parando antes do 9.

**Substring:** O método substring() é semelhante ao slice(). A principal diferença é que, no substring(), valores negativos são tratados como zero. Portanto, se passarmos um valor como -11 para ele, será interpretado como 0;

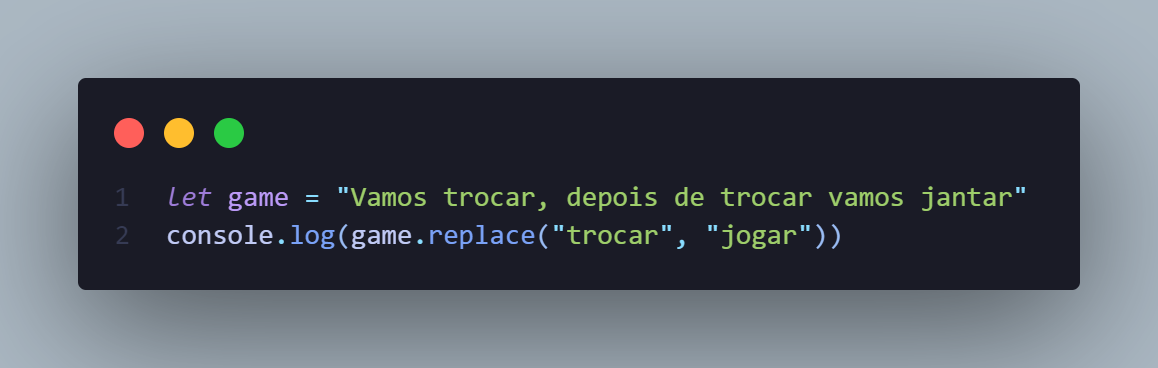
**Substr:** O método substr() é semelhante ao slice(), mas os parâmetros funcionam de forma diferente. O primeiro parâmetro define a posição inicial de onde a extração começará, e o segundo parâmetro indica quantos caracteres serão retornados a partir dessa posição, segue exemplo



Nesse exemplo, a extração começará no índice 5 e continuará por 2 caracteres. Portanto, se a string original contiver o texto adequado, o console exibirá, por exemplo, "VO".

Substituindo o conteúdo de uma string

Para substituir o conteúdo de uma string, temos o replace(), que tem como parâmetros o valor que vai ser substituído, e o valor que vai substituir, mas ele substitui só a primeira correspondência



Apenas a primeira ocorrência de "jogar" será substituída, resultando na frase: "Vamos trocar, depois de jogar vamos jantar".

Para substituir **todas** as correspondências de uma string, é necessário usar o sinalizador /g (correspondência global) com uma expressão regular. Nesse caso, todas as ocorrências da palavra "jogar" serão substituídas por "trocar", e a frase resultante será: "Vamos trocar, depois de trocar vamos jantar".

Outro problema que pode surgir é que o replace() diferencia letras maiúsculas de minúsculas. Isso significa que ele não substituirá uma palavra se houver diferenças na capitalização. Para resolver isso, podemos utilizar o sinalizador /i (insensível a maiúsculas e minúsculas), o que permite que tanto letras maiúsculas quanto minúsculas sejam substituídas. Dessa forma, a palavra "Jogar" com "J" maiúsculo, por exemplo, também será substituída por "trocar", mesmo que a busca seja feita por "jogar" em letras minúsculas.

Se quisermos substituir **todas** as ocorrências de uma palavra, independentemente de maiúsculas ou minúsculas, podemos combinar os sinalizadores /g e /i. Isso permitirá que todas as variações da palavra, como "Jogar", "jogar" e "JOGAR", sejam substituídas por "trocar", resultando em uma substituição completa e insensível à capitalização.

Outros Métodos

**toLowerCase()**: Transforma toda a string em minúsculas.

**toUpperCase()**: Transforma toda a string em maiúsculas.

**concat()**: Junta duas ou mais strings e pode substituir o operador +.

**trim()**: Elimina os espaços em branco de ambos os lados da string.

**trimStart()**: Elimina os espaços em branco do início da string.

**trimEnd()**: Elimina os espaços em branco do final da string.

**repeat()**: Retorna uma nova string com a repetição da string original pelo número de vezes especificado no parâmetro.

**split()**: Divide uma string em um array de substrings com base em um separador especificado. Cada elemento no array será uma parte da string original, dependendo do delimitador. Se o separador for um espaço (" "), as palavras serão separadas.