07

Para saber mais: Codificação de strings

Acabamos de ver que usamos o tipo `string` sempre que queremos trabalhar com dados de texto. Mas se pararmos para pensar, vários idiomas utilizam caracteres diferentes, como acentos e ideogramas. Como as linguagens de programação lidam com isso? E o que dizer dos *emojis* :question:? Você já visitou algum site e notou que os caracteres dos textos não pareciam corretos, que no lugar de alguns deles aparecia um sinal de interrogação, quadrados ou traços?

Isso tudo tem a ver com a **codificação de caracteres**, ou *character encoding*. Nas últimas décadas, foram desenvolvidos diversos conjuntos de caracteres especiais, cada um com seus próprios códigos, para que pessoas que escrevem e leem em linguagens diferentes do inglês pudessem utilizar computadores com seus próprios idiomas. E como isso funciona?

Para que o computador consiga **decifrar** um caractere especial, é preciso utilizar um sistema específico que tenha basicamente um código para cada caractere, e que o computador possa acessá-lo para fazer a conversão - uma ideia similar a que está por trás da criptografia.

Foram desenvolvidos diversos conjuntos de caracteres, desde os específicos de cada linguagem como Western, Latin-US, Japanese e assim por diante, até o ASCII (*American Standard Code for Information Interchange* ou ”Código Padrão Americano para o Intercâmbio de Informação”). e a partir de 2007 foi adotado o formato Unicode. O padrão UTF (de *Unicode Transformation Format* ou “formato de conversão de unicode”, em tradução livre) é utilizado como padrão na web até hoje.

O Unicode tem códigos específicos para “cifrar” e “decifrar” caracteres de mais de 150 idiomas antigos e modernos, e também diversos outros conjuntos de

caracteres como símbolos matemáticos e inclusive *emojs*. A [Wikipedia (https://en.wikipedia.org/wiki/List_of_Unicode_characters)](https://en.wikipedia.org/wiki/List_of_Unicode_characters) tem uma lista extensa de todas as tabelas com os códigos Unicode e os caracteres, como por exemplo os que estão abaixo:

caractere	UTF-16	descrição oficial
$	U+0024	DOLLAR SIGN
A	U+0041	LATIN CAPITAL LETTER A
✅	U+2705	CHECK MARK
あ	U+3041	HIRAGANA LETTER SMALL A

Podemos testar a transformação/conversão do código Unicode em caractere utilizando o `console.log()`. Faça o teste:

```
const cifrao = '\u0024'
const aMaiusculo = '\u0041'
const tique = '\u2705'
const hiragana = '\u3041'

console.log(cifrao)
console.log(aMaiusculo)
console.log(tique)
console.log(hiragana)
```

COPIAR CÓDIGO

Os caracteres `\u` no início do código são **caracteres de escape** que usamos para sinalizar ao JavaScript de que estamos falando de códigos Unicode, e não de strings de texto usuais.

O JavaScript usa, por padrão, o UTF-16. O número 16 está relacionado aos espaços em bits ocupados por cada caractere, 16 neste caso. Não vamos nos

aprofundar na relação entre tipos de dados e espaço de memória ocupado por cada tipo - você pode pesquisar mais sobre o assunto, assim como sobre o que são caracteres de escape! - mas por enquanto é bacana vermos na prática como o Unicode funciona.

Bancos de dados podem aceitar outros tipos de codificação de caracteres, o que faz sentido se pensarmos que o UTF-16 utiliza uma quantidade relativamente grande de espaço em memória para salvar cada caractere. 16 bits parece pouco, mas algumas vezes os bancos precisam salvar quantidades enormes de dados! Porém, com as tecnologias de armazenamento e tráfego de dados que temos hoje, esta já não é uma preocupação tão grande, a não ser em casos muito específicos. Já não é muito comum utilizar uma codificação diferente da UTF em bancos mesmo em caso de grandes volumes de dados, mas sempre vai depender **muito** do caso.

Mais detalhes precisos e documentação sobre o Unicode na página da Unicode Foundation (https://home.unicode.org/).