­­­­­Atividade Avaliativa JavaScript

Kauã Frenedozo e Raíssa Bernardo

1. A) Sim. As Strings podem ser manipuladas como arrays, pois existem métodos que podem fazer essa função.

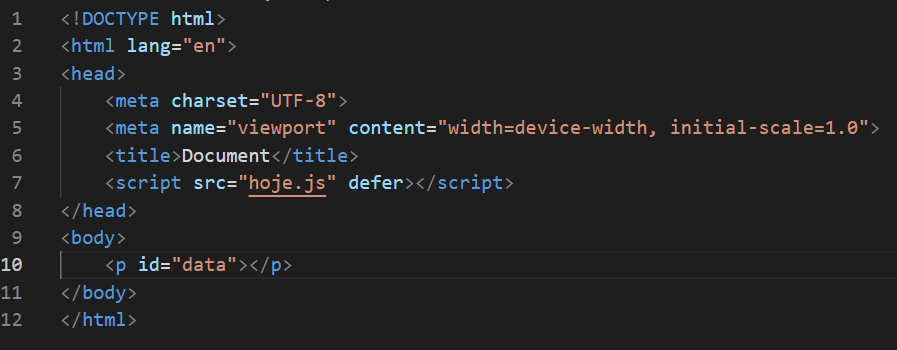
Por exemplo, quando usamos slipt(). Esse método divide a String em partes, de acordo com um padrão fornecido (vírgula, por exemplo) e transforma esses dados em uma lista ordenada, que é armazenada em um array. Quando retornada, retorna como array também.

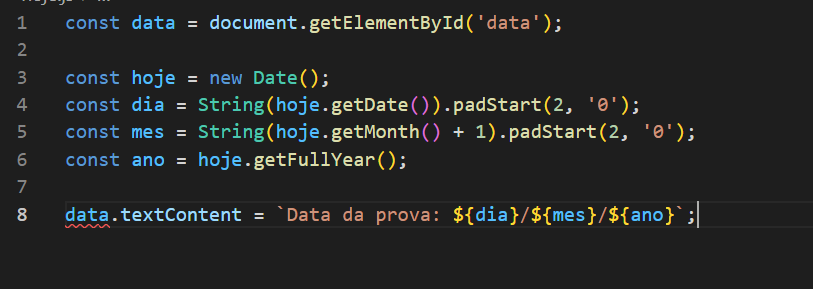
1. Math.floor() 🡪 arredonda os números decimais para cima, exemplo: 3.4 vira 4.0;

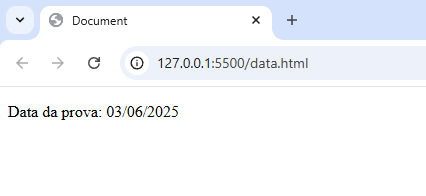
Math.ceil() 🡪 arredonda os números decimais para baixo, exemplo: 3.6 vira 3.0;

Math.round() 🡪 arredonda os números decimais para a casa mais próxima, exemplo: 3.4 vira 3.0 e 3.6 vira 4.0.

1. As datas são instanciadas pela classe Date(). Exemplo:







1. A) Em JS, expressão regular ou regex, são como um “controle” de padrões de Strings. Essas expressões podem garantir que haja consistência de dados e padrões de texto. Por exemplo: o CPF deve ter 11 dígitos, isso é um padrão. Isso melhora a qualidade do input e faz a validação da String.

B) O método .test verifica se a String corresponde ao padrão que foi imposto pelo regex, ou seja, ele serve para validar se ela está dentro do que é esperado.

1. A) Programação Assíncrona é quando decidimos a ordem em que o código roda, sem travarmos ele. Ou seja: rodamos ele por completo, executando o mais demorado sem que o mais rápido trave. Isso melhora a fluência do fluxo de código.

B) O JS é single-threaded, isso significa que as tarefas que são mais extensas e difíceis de concluir podem bloquear que as outras tarefas rodem, porque só roda uma linha por vez. O que ocasiona na lentidão do fluxo de código.

C) Promises 🡪 algo que será resolvido depois, é como se fosse um assunto futuro. Usamos .then() para lidar com as respostas;

Async await 🡪 é uma forma mais “limpa” de codar em assíncrono. Mas, no fim, utilizamos promises.

Callbacks 🡪 são funções que serão utilizadas como argumentos de outras funções depois.

Gosto de pensar na relação deles como uma árvore geneológica:

1. O Callback é a “avó”, pois é a mais antiga. Isso o torna mais difícil de ler.
2. Para que isso fosse melhorado, temos a “mãe”: o promises. Ele é uma versão melhorada do call-back, mais fácil de fazer.
3. Por fim, temos a “filha”: o async await. Ela ainda carrega os traços da mãe (usa o promises), mas é mais legível e simplificada.
4. Resumidamente, event loop é o que organiza a fila de execução de tarefas. Ele prioriza o código síncrono primeiro e depois processa as callbaks em ordem, sem travar nada.
5. Ele vai validar a entrada de dados, usando o formato ano-mês-dia (regex) e criar uma data com a classe Date(). Se for válido, ele vai pegar a data de hoje e comparar com a data colocada no input, transformando o resultado em dias (1000 \* 60 \* 60 \*24). Assim, ele vai retornar a diferença em dias.

A saída no console será: A data fornecida está a 0 dias de hoje.

1. 8, 3, 4, 7 🡪 porque eles são síncronos, então são exibidos primeiro;

1 ,6 🡪 porque são as promisses, ou seja, mais rápidas;

5, 2 🡪 são o setTimeOut, ou seja, atrasadas. Primeiro o 5 (1 segundo) e depois o 2 (2 segundos).

ORDEM COMPLETA DE EXIBIÇÃO: 8, 3, 4, 7, 1, 6, 5, 2