___/ ___/ _____

VÍDEO 37 (aula 21) - Operadores de Comparação

OPERADORES DE COMPARAÇÃO (comparam 2 valores):

> -> Maior que -> Checa se um valor é maior que o outro
>= -> Maior ou igual -> Checa 2 coisas - se o valor é maior ou igual ao outro valor
< -> Menor que -> Checa se um valor é menor que o outro
<= -> Menor ou igual -> Checa 2 coisas - se o valor é menor ou igual ao outro valor

== -> Igualdade -> só checa valor // Obs.: Não é recomendável que utilizemos este
=== -> Igualdade estrita -> checa valor e o tipo
!= -> Diferente -> só checa valor // Também não vamos utilizar
!== -> Diferente estrito -> checa valor e o tipo

Obs.: Professor recomenda que não utilizemos == e !=, pois eles checam apenas o valor e não o tipo.

Utilizar SEMPRE que possível === e !==

VÍDEO 38 (aula 22) - Operadores Lógicos

OPERADORES LÓGICOS

AND - && -> E -> Todos os valores precisar ser VERDADEIROS (true) para o resultado ser VERDADEIRO (true)
OR - || -> OU -> Apenas um valor precisa ser VERDADEIRO para retornar TRUE / Só retorna FALSE se os 2 valores forem FALSE
NOT - ! -> NÃO -> Essa negação inverte uma expressão

VÍDEO 39 (aula 23) - Avaliação de curto-circuito (short-circuit)
Vamos ver coisas que podem trabalhar para termos um código melhor e mais performático.

AVALIAÇÃO DE CURTO CIRCUITO PARTE 1 - AND && - RESUMO
&& -> true && false && false && true -> Assim que encontrar um valor avaliado como FALSO, vai retornar o primeiro valor avaliado como FALSE.

&& -> true && true && true && true -> Se todas forem VERDADEIRO, vai retornar o último valor VERDADEIRO.

/* AVALIAÇÃO DE CURTO-CIRCUITO PARTE 2 - OR || - RESUMO
|| -> false || false || true || true -> Quando tiver algum VERDADEIRO, vai retornar o primeiro valor avaliado como VERDADEIRO; o último valor não é avaliado, já temos o result.

|| -> false || false || false || false -> Sempre que todas forem avaliadas em FALSE, vai retornar o último avaliado como FALSO.

___/ ___/ _____

Avaliação de curto circuito = refere-se à avaliação condicional que é interrompida assim que o resultado final é determinado. Exemplo:

true && false && true -> vai ser interrompida a avaliação e vai exibir esse valor de false como resultado; não vai precisar checar a terceira expressão, pois já sabemos que o resultado é FALSE.

VÍDEO 40 (aula 24) - if, else if e else (parte 1)

Nessa aula vamos falar das estruturas condicionais if, else if e else, que servem para desviar o fluxo da nossa aplicação quando necessário.

=== ESTRUTURA CONDICIONAL SIMPLES: IF (pode ser executado sozinho, sem precisar de else e else if)

```
const hora1 = 11;

//   condição
if (hora1 <= 12) {
    console.log('Bom dia');
}
```

=== ESTRUTURA CONDICIONAL COMPOSTA: IF, ELSE IF (Serve para caso precisamos de mais condições) - ELSE = senão

```
const hora2 = 50;

//   condição
if (hora2 >= 0 && hora2 <= 11) {
    console.log('Bom dia');
} else if (hora2 >= 12 && hora2 <= 17) {
    console.log('Boa tarde');
} else if (hora2 >= 18 && hora2 <= 23) {
    console.log('Boa noite');
} else {
    // Se a hora não tiver entre 0 e 23, quero fazer alguma coisa padrão.
    console.log('Olá'); // Se nenhuma condição acima for atendida, ele vai executar esse bloco do else.
}
```

___/ ___/ _____

VÍDEO 41 (aula 25) - if, else if e else (parte 2)
Nessa aula vamos treinar nossa lógica de programação. Ver código da aula 25 no GitHub.

COMPORTAMENTOS DAS CONDICIONAIS - ALGUMAS OBSERVAÇÕES:
O IF e o ELSE dependem um do outro
A partir do momento que o JavaScript encontrar UMA condição VERDADEIRA, ele vai executar o bloco de código e vai sair fora do bloco e seguir o fluxo normal do programa (abaixo da condicional)
Se temos 2 blocos com condição VERDADEIRA, ele vai executar o primeiro e pular fora da condição sem executar o outro bloco verdadeiro
Se quisermos checar alguma coisa fora do IF, posso criar outro IF
Se preciso checar alguma coisa independente, crio um IF independente para aquela coisa

VÍDEO 42 - Modelo HTML e CSS para exercícios posteriores

Modelo que vamos utilizar em vários exercícios do curso. Ver na pasta "modelo" no GitHub.

VÍDEO 44 - Exercício: unindo tudo aprendido até aqui e mais
Vamos refazer o exercício de Tabela IMC que fizemos na parte básica do curso de um modo mais profissional.

REVER CÓDIGOS DO EXERCÍCIO NO GITHUB, POIS NÃO ENTENDI DIREITO

VÍDEO 45 (aula 27) - Operação ternária
Pode substituir IF e ELSE no código de modo a encurtar nosso código.

condição ? 'valor para verdadeiro' : 'valor para falso'

Vamos supor que o usuário tem 999 pontos. Se o usuário tem 1000 ou mais pontos = usuário VIP, menos de 1000 = usuário normal

```
const pontuacaoUsuario = 1000;
const nivelUsuario = pontuacaoUsuario >= 1000 ? 'Usuário VIP' : 'Usuário normal'
console.log(nivelUsuario)
```

___/ ___/ _____

VÍDEO 46 (aula 28) - Objeto Date
Serve para trabalharmos com datas no JavaScript.

```
// Criando data atual
const data1 = new Date(); // Não específicamos nenhum parâmetro, então ele vai criar a data atual
console.log(data1.toString());


// Criando data específica
const data2 = new Date(2023, 3, 25); // Vai criar uma data no dia especificado. -> ano, mês, dia, hora, minuto, segundo, milésimo de segundo // Obs.: mês no JS começa com 0 e vai até 11
console.log(data2.toString());


// Outro modo de criar data especificada - utilizando string
const data3 = new Date('2020-07-28 15:30:15'); // ano-mês-dia hora:minuto:segundo // Aqui podemos colocar a data normal utilizando string
console.log(data3.toString());

console.log('Dia', data3.getDate()) // Pegando o número do dia no mês // Resultado: Dia 28
console.log('Mês', data3.getMonth() + 1) // Pegando o mês // Resultado: Mês 7
console.log('Ano', data3.getFullYear()) // Pegando o ano // Resultado: Ano 2020
console.log('Hora', data3.getHours()) // Pegando a hora // Resultado: Hora 15
console.log('Minuto', data3.getMinutes()) // Pegando minutos // Resultado: Minuto 30
console.log('Dia da semana', data3.getDay() + 1) // Pegando o dia da semana // Resultado: Dia da semana 3


-----
========== DIAS NO JAVASCRIPT ==========
0 = Domingo
1 = Segunda
2 = Terça
3 = Quarta
4 = Quinta
5 = Sexta
6 = Sábado

========== MESES NO JAVASCRIPT ==========
0 = Janeiro, 1 = Fevereiro, 2 = Março, 3 = Abril, 4 = Maio, 5 = Junho, 6 = Julho, 7 = Agosto, 8 = Setembro, 9 = Outubro, 10 = Novembro, 11 = Dezembro
```

___/ ___/ _____

VÍDEO 47 (aula 29) - Switch/case

Ver código da aula 29 sobre Switch/case no GitHub para saber mais.

VÍDEO 48 (aula 30) - Exercício com switch e date

Exercício para mostrar o dia da semana, dia do mês, mês, ano e hora na página HTML.

REVER!!!!!!

Ver código da aula 30 para saber mais.

VÍDEO 49 (aula 31) - Mais diferenças entre var e let/const

Como já vimos, não podemos redeclarar uma variável com let
LET tem escopo de bloco (tudo que tiver dentro de chaves) -> {... isso é um bloco}
LET -> caso eu crie o let dentro de um bloco, quem está fora do bloco não consegue acessar
VAR só tem escopo de função
O único escopo que VAR respeita é o escopo de função

Você pode acessar o valor das variáveis que estão no closure (entorno da função). Ela primeiro vai procurar o valor da variável dentro dela, caso não encontre ela procura fora da função.

Podemos acessar o que está fora da função, mas o que está fora não consegue acessar o que está dentro.

Caso a variável esteja definida dentro da função, ela não vai poder ser acessada de fora.

// HOISTING
// No JS acontece algo estranho com variáveis declaradas com VAR: Quando criamos variáveis com VAR o JS faz algo chamado elevação (hoisting), ele eleva a declaração da variável.
// Ele vai ler o código, vai detectar todos os locais onde você está criando sua variável com VAR, vai pegar essa variável e vai declarar essa variável no topo do arquivo pra você.
// Apenas a declaração da variável é levada ao topo do código, NÃO SEU VALOR. O valor continua no local onde a variável foi escrita.

// RESUMINDO: Apenas a declaração da variável é içada para o topo, não o valor.

// ISSO NÃO OCORRE HOISTING COM LET! -> Se uso uma variável antes de declarar ela, vai me retornar um erro

___/ ___/ _____

VÍDEO 50 (aula 32) - Atribuição via desestruturação (Arrays)
é uma técnica que permite extrair valores de arrays ou objetos e atribuí-los a variáveis de forma mais simples.
Com a desestruturação de arrays, você pode extrair valores de um array e atribuí-los a variáveis individuais, com base na posição deles no array.

Exemplo:

```
const numeros = [1, 2, 3];
const [primeiro, segundo, terceiro] = numeros;

console.log(primeiro); // Saída: 1
console.log(segundo); // Saída: 2
console.log(terceiro); // Saída: 3
```

Neste exemplo, [primeiro, segundo, terceiro] é a sintaxe de desestruturação que extrai os valores do array numeros e os atribui às variáveis primeiro, segundo e terceiro.

VÍDEO 51 (aula 33) - Atribuição via desestruturação (Objetos)
Com a desestruturação de objetos, você pode extrair valores de propriedades de um objeto e atribuí-los a variáveis com o mesmo nome das propriedades.

Exemplo:

```
const pessoa = { nome: 'João', idade: 30 };
const { nome, idade } = pessoa;

console.log(nome); // Saída: 'João'
console.log(idade); // Saída: 30
```

Neste exemplo, { nome, idade } é a sintaxe de desestruturação que extrai os valores das propriedades nome e idade do objeto pessoa e os atribui às variáveis de mesmo nome.

VÍDEO 52 (aula 34) - Estrutura de repetição - For (clássico)
Estruturas de repetição fazem coisas repetitivas pra não termos que ficar repetindo código.

```
for (let i = 0; i <= 5; i++) {
    console.log(`Linha ${i}`)
}
```

Resultado: Linha 0, Linha 1, Linha 2, Linha 3, Linha 4, Linha 5

___/ ___/ _____

VÍDEO 53 (aula 35) - Exercícios com FOR
Ver no GitHub a resolução e comentários.

VÍDEO 54 - DOM e árvore do DOM
O objeto geral é o window. Ele é o pai de todos os elementos no HTML.
O window é o topo da cadeia dentro do navegador, antes dele não tem mais nada.
O document seria nosso documento HTML.

O document é child de window e window é parente de document.
Dentro de document, temos HTML (child) e dentro de HTML temos head e body, que são seus child.

Sempre que falarmos DOM, estou me referindo ao navegador e estou me referindo a objetos que me permitem manipular elementos dentro da página HTML.

VÍDEO 55 (aula 36) - FOR IN - Estrutura de repetição

FOR IN (ARRAY) -> percorre o ÍNDICE do ARRAY:

```
const frutas = ['Pêra', 'Maçã', 'Uva'];

for (let indice in frutas) { // Está percorrendo o índice do array frutas
    console.log(frutas[indice]) // Resultado: Pêra Maçã Uva
}
```

FOR IN (OBJETO) -> percorre a PROPRIEDADE do OBJETO

```
const pessoa = {
    nome: 'Luiz',
    sobrenome: 'Otávio',
    idade: 30
};

for (let propriedade in pessoa) {
    console.log(propriedade, pessoa[propriedade]) / Resultado: nome Luiz, sobrenome Otávio, idade 30
}
```

___/ ___/ _____

VÍDEO 56 (aula 37) - FOR OF - Estrutura de repetição

Esse FOR é específico para objetos iteráveis, como strings, arrays, etc.

FOR OF -> percorre o ELEMENTO da string, array, etc.

```
const nomes = ['Luiz', 'Otávio', 'Henrique']

for (let valor of nomes) {
    console.log(valor); // Resultado: Luiz, Otávio, Henrique
}


// O FOR OF retorna apenas o valor, não o índice.
```

____/ ____/ _____

____/ ____/ _____

____/ ____/ _____

____/ ____/ _____