___/ ___/ _____

Vídeo 7 - console.log()

console.log -> serve para exibir coisas na tela, debugar código, etc.

Dentro dessa função posso colocar qualquer coisa que ainda não vimos, como uma variável, um objeto.

Se quisermos exibir algum texto, por exemplo, devemos envolver ele entre aspas, seja ela aspas simples, duplas ou crase:

```
console.log('Olá mundo');
```

O console.log é uma das maneiras de descobrirmos o valor de uma variável, de um objeto, de um array e coisas do tipo.

Todo texto dentro de aspas simples, duplas ou crases é considerado uma STRING.

Também podemos exibir números na tela, e, nesse caso, não utilizamos aspas:

```
console.log(1234); -> número inteiro
console.log(15.85); -> número real, com ponto flutuante
console.log(1234, 15.85, 'Olá mundo');
```

Dentro do console.log podemos mandar exibir vários valores, separando os valores que queremos exibir por vírgula. Mais um exemplo:

```
console.log('Meu nome é "Júnior". Estou aprendendo JavaScript às', 5, 'da manhã.');
```

Resultado:
Meu nome é "Júnior". Estou aprendendo JavaScript às 5 da manhã.

Vídeo 11 (pasta aula 4) - Comentários de código

Comentários são trechos de código que são ignorados pelo motor do JavaScript. É como uma anotação para explicar algo do seu código. Para isso, utilizamos duas barras:

```
// Este é um comentário -> duas barras; só funciona com 1 linha de código
/* Este é outro comentário */ -> barra asterisco e asterisco barra para mais de 1 linha de código.
```

Obs.: Também podemos usar comentários para trechos de códigos que eu não quero que seja executado.

___/ ___/ _____

Vídeo 12 (pasta aula 5) - Navegador vs Node (HTML + JavaScript)

Para incluirmos JavaScript dentro do HTML, podemos colocar a tag <script></script> tanto dentro do nosso <head>, quanto no final do HTML, logo acima do </body>.

Não é uma boa prática colocar a tag <script> dentro do nosso <head>, pois isso pode retardar o carregamento da página. Quando o navegador detectar essa tag, ele vai tentar executar os scripts que estão ali dentro. Pode ser uma má experiência para o usuário, que pode ficar esperando com uma página em branco até que todo o script seja executado.
Uma melhor prática é colocar nosso <script> no final do nosso HTML, antes do fechamento do body (</body>).

Mesmo assim, nosso código pode ficar extremamente longo e confuso quando misturamos o código HTML com o JavaScript. O ideal é que separemos a parte do JavaScript do nosso HTML.
Para isso, digitamos 'script' no nosso VSCode na parte do HTML e escolhemos a opção 'script:scr'. Fica assim:

<script src="index.js"></script> -> nosso arquivo externo com nossos códigos JS.

Quando falamos de Frontend a gente está falando de uma página que é carregada e executada no computador do cliente (pessoa que está acessando).
Já o Backend é um servidor que vai servir a página, servir os dados para determinada página.

Mesmo o Chrome e o node.js tendo o mesmo motor (V8), eles se comportam de maneira diferente. Por exemplo, se formos no console do navegador e digitarmos:

alert('Olá mundo); -> ao mandarmos executar, aparece uma janelinha com a mensagem.

Se colocarmos o mesmo comando no node.js, ele não será executado e aparecerá um erro.

Vídeo 13 (pasta aula 6) - Variáveis com let

Variável serve para salvarmos determinado dado, determinado valor dentro da memória.
Variável, como o próprio nome diz, é algo que pode variar, algo que pode ser modificado.
Para criarmos uma variável, temos 2 maneiras: var e let;
Neste curso vamos utilizar apenas let, pois var é mais antigo.
Fica assim:

let nome = 'João'; -> nome é o rotúlo da variável e está recebendo 'João'.

```
console.log(nome, 'nasceu em 1984.');
console.log('Em 2000', nome, 'conheceu Maria.');
console.log(nome, 'casou-se com Maria em 2012.');
console.log('Maria teve 1 filho com', nome, 'em 2015.');
console.log('O filho de', nome, 'se chama Eduardo.');
```

Se eu mandar executar o código, em todo local que está a variável nome vai aparecer escrito João. Resultado:

```
João nasceu em 1984.
Em 2000 João conheceu Maria.
João casou-se com Maria em 2012.
Maria teve 1 filho com João em 2015.
O filho de João se chama Eduardo.
```

Obs.: separando por vírgula não é a melhor maneira de fazer isso; mais para frente vamos ver como concatenar (juntar) e veremos mais sobre template strings utilizando crase.

Posso declarar uma variável e não inicializar ela, não atribuir nenhum valor a ela. Ao mandarmos exibir seu valor na tela, vai retornar como resultado 'undefined'. Ex:

```
let nome2;
console.log(nome2);
```

Resultado:
undefined -> a variável foi declarada, mas não tem valor no momento.

Posso declarar a variável e posteriormente atribuir um valor a ela:

```
let nome2;
nome2 = 'José'
console.log(nome2); -> Resultado: José
```

Também podemos modificar o valor da variável:

```
nome2 = 'José'
console.log(nome2) -> Resultado: José
nome2 = 'Otávio'
console.log(nome2) -> Resultado: Otávio
```

Obs.: uma coisa que não podemos fazer é redeclarar a variável, declarar de novo alguma variável que já foi declarada. Isso vai gerar erro no nosso código.
Ver no código do nosso index.js da nossa aula 6 sobre variáveis para saber sobre as regras de criação de variáveis ou nos nossos PDFs do Curso em Vídeo do curso de JavaScript.

___/ ___/ _____

Vídeo 14 (aula 7) - Constantes com const

É chamada no JS de variável constante, que é uma variável que não muda o seu valor.
Não podemos modificar o valor de uma constante. Não podemos também declarar uma constante e depois tentar atribuir um valor a ela. Ela já precisa ser criada tendo algum valor.

Obs.: ver nos códigos das aulas para saber as regras de nome de let e const.

Para declararmos uma constante, fazemos:

```
const nome = 'João'
nome = 'João' -> Resultado: isso vai retornar um erro, pois não podemos alterar valor de uma constante.
console.log(nome)
```

Como já dissemos, não podemos modificar o valor de uma constante. Isso é um comportamento muito importante.

Quando a variável precisar ter seu valor alterado, utilizamos let;
Quando queremos que o valor não se modifique, utilizamos const.

Também podemos pegar o valor de uma variável constante e atribuir seu valor a outra variável ou outra constante. Exemplo:

```
const primeiroNumero = 5;
const segundoNumero = 10;
const resultado = primeiroNumero * segundoNumero; // Multiplicando a primeira constante pela segunda
const resultadoDuplicado = resultado * 2
console.log(resultado) // Resultado: 50
console.log(resultadoDuplicado) // Resultado: 100 (o valor de resultado * 2)
```

Note que continuo evoluindo o meu valor, em nenhum momento estou tentando alterar ele. No caso de quisermos alterar o valor, utilizamos let no lugar de const:

```
let resultadoTriplicado = resultado * 3 // Resultado: 50 * 3 = 150
resultadoTriplicado = resultadoTriplicado + 5 // Resultado: 155
```

A variável let perdeu seu valor inicial, que era de 150 e passou a valer 155.

Para sabermos o tipo de determinada variável, podemos fazer, por exemplo:

```
console.log(typeof(primeiroNumero)); -> Resultado: number
```

___/ ___/ _____

Se fazermos, por exemplo:

```
const primeiroNumero = '5'
const segundoNumero = 10
console.log(primeiroNumero + segundoNumero);
```

Resultado: 510
Dá esse resultado pois o primeiro dado é do tipo STRING e o segundo é NUMBER. Sendo assim, o JavaScript CONCATENA (junta) os valores em vez de somar.

Vídeos 14 a 17 (aula 8) - Exercícios com const e let -> fizemos tudo, ver no código index.js da aula 8.

Vídeo 18 (aula 9) - Let vs Var - Primeira diferença

A primeira diferença entre VAR e LET é:

Com VAR -> posso redeclarar uma variável, como podemos ver nas linhas 1 e 2 do nosso código;
Com o LET -> não é possível redeclarar uma variável, como podemos ver nas linhas 5 e 6. Isso irá gerar um erro ao executarmos nosso código, dizendo que nosso LET já foi declarado.

Obs.: ver códigos da aula 9 para saber mais.

VÍDEO 19 (aula 9) - Tipos de dados primitivos
Ver no código da aula 9 no GitHub para saber mais. (Professor se confundiu na numeração das aulas)
STRING: cadeia de caracteres; posso ter tanto com aspas simples, quanto com aspas duplas ou com crase.

NUMBER: um número normal; O JavaScript não diferencia números inteiros e números com ponto flutuante, são todos do mesmo tipo (NUMBER).

UNDEFINED: Indefinido; a variável recebe esse valor automaticamente quando apenas declaramos ela, sem atribuir nenhum valor. Não aponta pra lugar nenhum na memória. Nunca vou criar uma variável e configurar o valor dela para undefined. Quem faz isso é o JavaScript automaticamente quando não definimos um valor para a variável.

NULL: Nulo -> também não aponta pra lugar nenhum na memória
Utilizamos quando queremos desconfigurar uma variável, quando queremos colocar o valor da variável como nulo; Eu explicitamente digo que eu não quero que essa variável aponte pra lugar nenhum na memória.

BOOLEAN: só tem 2 possibilidades de valor: TRUE ou FALSE

___/ ___/ _____

VÍDEO 20 (aula 10) - Operadores aritméticos, de atribuição e incremento
Ver códigos da aula 10 no GitHub para mais informações e também na aula sobre operadores do curso de JavaScript do Curso em Vídeo para saber sobre os operadores e sobre a ordem de procedência.

Convertendo STRING em NUMBER:

let num = '10.5' -> STRING
== Métodos mais antigos ==

num = parseInt('10.5') -> converte a STRING em um número inteiro; Resultado: 10
num = parseFloat('10.5') -> converte p/ número de ponto flutuante; Resultado: 10.5

== Método mais moderno ==
num = Number('10.5') -> converte STRING para NUMBER; Resultado: 10.5

Obs.: Se tentarmos converter, por exemplo:
let nome = Number('Luiz') -> Resultado: NaN, ele n vai conseguir fazer a conversão.

VÍDEO 21 (aula 11) - alert, confirm e prompt (Navegador)
Vamos ver 3 funções do navegador que vão nos ajudar a fazer alguns exercícios.
Vamos criar um documento HTML index.html e linkar nosso script script.js neste documento.
Não temos alerta no node, por isso estamos trabalhando no navegador. Fazemos assim:

window.alert('Olá, mundo'); -> exibe uma janela de alerta.

No navegador, temos o elemento central, como se fosse o pai de todos os elementos, que é o objeto window (janela do navegador). Dentro desse objeto window eu tenho essas funções, como alert(), confirm(), prompt(), etc.

Assim como fazemos console.log(), estamos chamando o método log que está dentro do objeto console.

window.confirm('Deseja mesmo apagar a mensagem?'); -> serve para confirmar coisas do usuário se ele realizar determinada ação, como apagar um post.
Nessa janela teremos 2 botões: um de 'OK' e outro de 'Cancelar'.

Caso o usuário clique em 'OK', retorna o valor 'true'. Se o usuário clicou em 'OK', é executada a ação que ele pediu.
Caso o usuário clique em 'Cancelar', retorna o valor 'false'. Se o usuário clicou em 'Cancelar', a ação não é executada.

Outra janela que temos é a prompt, que vai pedir pro usuário digitar alguma coisa. Fica assim:

window.prompt('Digite o seu nome'); -> pede que o usuário digite algo

Quando uma função está dentro de um objeto, chamamos ela de MÉTODO. Se tiver fora do objeto, chamamos de FUNÇÃO.

Sempre que eu chamo uma função em JavaScript, sempre que eu executo qualquer ação, como um console.log('Olá'), essa função pode ou não me retornar um valor. Quando falamos em retornar, estou dizendo que essa função pode me dar um valor de volta.

Por exemplo: no alert(), por padrão, ela não retorna nada. Sempre o que vai retornar vai ser undefined, que não aponta pra lugar nenhum na memória.
Já a função confirm() me retorna alguma coisa: quando clicamos em 'OK' me retorna o valor true, quando clicamos em 'Cancelar' me retorna o valor false.

====== GUARDANDO O QUE FOI RETORNADO PELO CONFIRM ======

Exemplo:
const confirma = window.confirm('Realmente deseja apagar?')

// O que vai ser guardado na nossa variável está relacionado com os botões que apertarmos; Se apertamos 'OK' o valor true vai para a nossa const confirma. Se eu apertar 'Cancelar', o valor false vai para dentro da const confirma.

// Se clicarmos em 'OK', por exemplo, e depois fazermos um console.log(confirma), o resultado que vai aparecer é true. O que retornar pela função confirm() vai ser o valor da nossa variável.

Obs.: o texto não tem nada a ver com o que vai ser retornado pela função, é apenas algo que vai ser exibido na janelinha de texto.

___/ ___/ _____

Exercício proposto pelo professor:

```
const num1 = prompt('Digite um número')
const num2 = prompt('Digite outro número')
const resultado = num1 + num2
alert(`O resultado foi de ${resultado}`)
```

Pegar o que for digitado nos prompt e fazer uma conta de ADIÇÃO.
Assim que o usuário digitar os 2 números nos respectivos prompts, quero que apareça um alerta com o resultado da conta.

Resolvi o exercício. Ver na pasta 'exercícios' dentro da Aula 11.

VÍDEO 23 (aula 12) - Exercício com variáveis

Ver código da aula12, aula12.js, para ver a resolução e ver os comentários.

Neste vídeo o professor propôs o seguinte exercício:
Queremos que a varA tenha o valor de varB, varB tenha o valor de varC e varC tenha o valor de varA.

// Resolução com método antigo:

```
let varA = 'A'; // B
let varB = 'B'; // C
let varC = 'C'; // A

let varATemp = varA;
varA = varB;
varB = varC;
varC = varATemp;
```

console.log(varA, varB, varC); -> Resultado: B C A

// Resolução com método mais moderno (vamos ver isso mais pra frente no curso):

```
[varA, varB, varC] = [varB, varC, varA]
```

B C A

console.log(varA, varB, varC); -> Resultado: B C A

___/ ___/ _____

VÍDEO 24 (aula 13) - Mais sobre strings

Como já vimos, uma string é um texto. Se quisermos colocar aspas duplas dentro de aspas simples, podemos. Se quisermos colocar aspas simples dentro de aspas duplas, podemos também. Ex:

```
let umaString = 'Um "texto"';
let umaString = "Um 'texto'";
```

===== ASPAS SIMPLES DENTRO DE ASPAS SIMPLES OU ASPAS DUPLAS DENTRO DE ASPAS DUPLAS =====

let umaString = "Um \ "texto\""; -> serve para escaparmos o caractere dentro das aspas;

===== SE QUISER EXIBIR A BARRA QUE ESTÁ DENTRO DA STRING =====
let umaString = "Um \\texto"; -> Resultado: Um \texto

===== STRINGS SÃO INDEXADAS =====
Cada índice representa um caractere da nossa STRING. Para sabermos o caractere dentro de determinado índice, fazemos:

```
let umaString = "Um texto";
console.log(umaString[0]) -> utilizamos colchetes para saber o caractere que está em determinado índice
// Resultado: U
```

===== FUNÇÃO PARA PEGAR ELEMENTO DE DETERMINADO ÍNDICE charAt() =====
Faz exatamente a mesma coisa de colocarmos o índice dentro de [ ];

```
let umaString = "Um texto";
console.log(umaString.charAt(5)) -> vai retornar o elemento que está no índice 5
// Resultado: x
```

===== FUNÇÃO PARA CONCATENAR STRINGS concat() =====
Faz exatamente o que o sinal de + faz;

```
let umaString = "Um texto";
console.log(umaString.concat(' em um lindo dia'))
// Resultado: Um texto em um lindo dia
```

===== OUTRAS MANEIRAS DE CONCATENAR =====

```
console.log(umaString + ' em um lindo dia');
console.log(`${umaString} em um lindo dia`); -> utilizando TEMPLATE STRING

// Resultado: Um texto em um lindo dia
```

____/ ____/ _____

===== PESQUISAR ÍNDICE DE DETERMINADO VALOR indexOf() =====

```
let umaString = "Um texto";
console.log(umaString.indexOf('t') -> queremos saber o índice da letra t
// Resultado: 3
```

** Caso retorne -1, significa que não encontramos nenhum índice associado àquele valor.

Não coloquei tudo da aula nesse PDF; ver o resumo no código da aula 13 no nosso GitHub)

VÍDEO 25 (aula 13) - Exercícios com strings

Neste vídeo, o professor propôs fazer várias perguntas relacionadas ao nome que digitarmos no prompt.
Vamos responder essas perguntas utilizando todos os conhecimentos do vídeo anterior.

Ver dentro da pasta exercicios da aula13 pra ver a resolução.

VÍDEO 26 (aula 14) - Mais sobre numbers

Não coloquei tudo da aula nesse PDF; ver o resumo no código da aula 14 no nosso GitHub)

VÍDEO 27 (aula 15) - Objeto Math

Temos esse objeto disponibilizado pelo próprio JavaScript que é o objeto Math, que tem várias coisas relacionadas com matemática.
Nesta aula vamos algumas coisas que podemos fazer utilizando esse objeto.

Neste PDF vou deixar apenas alguns vistos no vídeo e os usados com mais frequência.
Para mais informações, ver código da aula 15 no GitHub.

___/ ___/ _____

```
// MÉTODOS PARA ARREDONDAR NÚMEROS -> floor(), ceil(), round()
let num1 = 9.54578;
let arredondar1 = Math.floor(num1); // Método floor() - Arredonda para baixo (deixa ele como um número inteiro)
console.log(arredondar1); // Resultado: 9

let arredondar2 = Math.ceil(num1); // ceil() - Arredonda para cima
console.log(arredondar2);  // Resultado: 10

let arredondar3 = Math.round(num1); // round() - Arredonda automaticamente para número inteiro mais próximo - Ex: 9.49 ele vai arredondar para 9; 9.55 ele vai arredondar para 10
console.log(arredondar3); // Resultado: 10



// MÉTODOS PARA SABERMOS O MAIOR OU MENOR NÚMERO EM UMA LISTA DE NÚMEROS -> max() e min()
console.log(Math.max(1, 2, -15, 80, 47, 12, 37, 10)); // max() - Ele pega o maior número dentro de uma sequência de números // Resultado: 80
console.log(Math.min(-8, 4, 74, 12, 15, -47, 60, 32, 13)); // min() - Pega o menor número dentro de uma sequência de números // Resultado: -47


// MÉTODO PARA GERAR NÚMEROS ALEATÓRIOS -> random() -> O 1 NUNCA é incluído
const aleatorio = Math.random(); // random() - Vai gerar um número aleatório entre 0 e 1, mas o 1 NUNCA é incluído
console.log(aleatorio);

const aleatorio2 = Math.round(Math.random() * 100) // random() - Para gerar um número aleatório entre 0 e 1 multiplicado por 100 - Note que também arredondamos utilizando o método round()
console.log(aleatorio2);
```

VÍDEO 28 (aula 15) - Exercício com numbers e Math

Vamos fazer um exercício com numbers e Math e vamos começar a trabalhar com algo mais próximo de como é feito no mundo real.

Ver código da aula 15, pasta exercicios, no GitHub para saber mais.

___/ ___/ _____

VÍDEO 29 (aula 16) - Arrays (básico)

Podemos imaginar arrays como se fossem uma lista de coisas.
Imagina que precisamos colocar numa variável só uma lista de nomes de alunos.
Ficaria assim:

const alunos = ['Luiz', 'Maria', 'João'] -> Luiz = 0, Maria = 1, João = 2

Geralmente, vamos tentar manter as coisas organizadas e colocar só um tipo de dado dentro do array, que no caso acima, são strings.

Arrays também são indexados, mas diferente das strings, eles são indexados pelo seu elemento:

0 1 2
const alunos = ['Luiz', 'Maria', 'João']

Ver códigos da aula 16 no GitHub para ver todos os detalhes sobre arrays.