nodejs. para compilar tecle Ctrl + B

JavaScript
- Linguagem de programação
- Criado originalmente para interagir com elementos de uma página da web, adicionar um evento de clique a um botão que mostra um alert, altera o estilo de um element.
- Não é JAVA, é um derivado da linguagem C, foi criado pensando em java
- É dinamicamente tipada e compilada JIT, var x = 0; o código é compilado quando é executado

JavaScript no cliente vs no servidor
Funciona em ambos os lugares.Mas, não exatamente da mesma maneira
- Cliente (Browser) um navegador, desktop, laptop ou dispositivo móvel
- Server (Node.js) código que é executado diretamente em um computador, geralmente um servidor.
- Cliente: É executado em um navegador da web usando tags de scripts.Tem acesso às funções e ao objeto da página web Document Object Model (DOM)
- Servidor: requer o Node.js e tem acesso a pacotes integrados e de terceiros. Normalmente usado para construir coisas como serviços da web.
- Você deve sempre ter 3 pontos.

Declarando variáveis
Existem três formas
var um = 1;
let dois = 2;
const três = 3;
var
- função com escopo
- pode ser alterado em escopo
- disponivel antes da declaração

let
- bloco com escopo
- pode ser alterado em escopo
- disponível somente após declaração

const
- bloco com escopo, como let
- não pode ser alterado
- somente disponível após a declaração

Quando devemos usar?
const por padrão
let em loops

Concatenação de Strings
Combinando duas ou mais strings, recurso que torna mais fácil formatar textos, podemos juntar uma combinação de variáveis e strings (também conhecidos como strings literais). Concatenação de strings criarão novas strings.
Normalmente se usa o operador +, o operador de adição não serve apenas para números, strings e números podem ser usados.
Ex:
```
let str1 = "ola";
let str2 = "Mundo!";
console.log(str1  +  str2);
console.log(str1  +  " Grande " +  str2)
```

compilando sairá
Olá Mundo!
Olá, Grande Mundo!

Template Literals
Formatação flexível
- a sintaxe torna as strings mais faceis de formatar e ler
- uso de marcadores $( ) para variáveis ou expressões
- respeita as quebra de linha.Não precise do caractere para incluir uma nova linha "\n"

Uso de acentos graves (``)
Templates liberais requerem apenas o caráter do acento grave ``, colocando no início e no final de uma determinada string. Não há necessidade de caracteres de aspas duplas.
Diferentes formatações de strings
Operador de concatenação +
- precisa de aspas simples ou aspas duplas
- quebras de linha com caractere de nova linha
- precisa concatenar as variáveis

Template literals
- precisa do par de acentos graves (``)
- respeita a quebra de linhas
- os marcadores inserem novos valores e expressões de variáveis

Exemplos:
```
let str1 = "JavaScript";
let str2 = "legal";
sem template literals
console.log("Estou escrevendo códigos em", str1)
console.log('Estou escrevendo códigos em ' + str1)
Com template literals
console.log(`Estou escrevendo códigos em ${str1}`);
Concatenação de variáveis
Com template literals
console.log(`Formatando strings em ${str1} e ${str2}!`);
Outro exemplo:
let bool1 = true;
const getValue = (num) =>{
   return num + num
}
console.log(`O oposto de true é ${!bool1}`)
```
Tipo de dados
JavaScript possui tipagem fraca
- Tipos simples da linguagem

Number(float), String, Boolean, Date, Function, Array e Object
- Tipos Especiais

NaN, null, undefined
Verificando tipos
- operador typeof : retorna uma string do tipo de dado primitive
- operador instanceof : retorna um booleano se um valor correspondente ao tipo dos dados.

Exemplos:
```
Number
let idade = 33;
```

```
console.log(idade);
Float
let precoMouse = 19.99;
console.log(precoMouse);
String
let nome = "Léia";
console.log(nome);
Boolean
let nomeVerdadeiro = true;
console.log(nomeVerdadeiro);
let nomeFalso = false;
console.log(nomeFalso);
Date
let dataHoje = new Date();
console.log(dataHoje);
Function
let resultado = adicionarNumeros(5,2);

function adicionarNumeros(num1, num2) {
    return num1 + num2;
}

console.log(resultado);
Array
let frutas = ["Banana" , "Laranja", "Abacaxi"];
console.log(frutas);

Object
const pessoa = {
    nome: "Léia",
    sobrenome: "Santos",
    idade:35
};
console.log(pessoa);
Dados especiais
NaN (é number)
let num = NaN
console.log(typeof num); // number
console.log(num == NaN); // falso
console.log(num === NaN); // falso
Null
let variavelQualquer = null;
console.log(variavelQualquer);
Undefined
let carro;
carro = undefined;
console.log(carro);
typeof
console.log(typeof 1);
console.log(typeof 'leia santos');
```

instaceof
```
class Carro {};
let carro = new Carro();
console.log(carro instaceof Carro);
```
Igualdade Estrita
```
console.log(20 == 20) // comparando valor e tipo por isso ==
console.log('20' == 20);
```
Igualdade Solta
```
console.log(20 = 20); // compara valores
console.log('1' = 1);
```

Operações matematicas adicionais
```
let num1 = 100;
console.log(num1 % 1500) ;resto
console.log(++num1); incrementa
console.log(--num1); decrementa
```

O objeto Math
Para operações matemáticas, conjunto de constantes e funções, executa operações trigonométricas, logarítmicas e muito mais
Ex:
```
let num1 = 100;

console.log(Math.PI); Pi
console.log(Math.sqrt(num1)); raiz quadrada
console.log(Math.tan(num2)); tangente
console.log(Math.abs(num2)); valor absoluto
```

Números e strings
Conversão entre numbers e strings
parseInt( ) e parseFloat( )
Converte strings numeradas em números, adiciona caracteres não numéricos e pode ter resultados indesejados, parseFloat( ) são para números de ponto flutuante e números com pontos decimais
toString( ) converte números em strings numéricas

Exemplo conversão:String (inteiros: parseInt( ))
```
var numeroDecimal = parseInt("50");
console.log(numeroDecimal);
console.log(typeof numeroDecimal);

var hexadecimal = parseInt("0xF")
console.log(hexadecimal)
```
Exemplo de conversão:String(floats:parseFloat( ))
```
var mouse = parseFloat("29.90");
console.log(mouse);
console.log(typeof mouse);
```

Exemplo de converão:Numeros:(Strings:toString( ))
```
var num = 33;
var idade = num.toString();
```

```
console.log(idade);
console.log(typeof idade);
```

DATAS
Criando o objeto date
Retornando a data atual

```
const now = new Date()
```

Definindo data e hora específicas, a contagem do mês começa com zero!

```
const randomDate = new Date(2015, 3, 12, 6, 25, 58;
```

Definindo uma data especifica- hora definida para

```
const win95Launch = new Date(1995, 7, 24);
console.log(randomDate);
console.log(win95Launch);
```

Configurando Valores

```
const now = new Date();

now.setFullYear(2014);  * define ano
now.setMonth(3); Abril * meses  (a contagem começa em zero)
now.setDate(4); * define o dia

now.setHours(4); * 24horas
now.setMinutes(24); *minutos
now.setSeconds(46); *segundos

console.log(now);
```

Retornando Valores

```
const now = new Date();
console.log(now.getMonth());  * Todas as funções possui a palavra GET
console.log(now.getTime()); * Milissegundos desde 1 jan de 1970
console.log(now.getDay()); * dia da semana (Domingo = 0)
```

Alguns exemplos:

```
const agora = new Date()
console.log(agora)
```

Retorna a data do dia que você está manipulando.

Comparando Valores em javaScript
O javaScript suporta os operadores comuns
< para número menor ou mais próximo do inicio do valor
<= para número menor ou igual ou mais próximo do inicio do valor
> para um número maior ou mais longe do inicio do valor
>= para maior ou igual número ou igual

Conversões de igualdade e tipos de dados
JavaScript converte automaticamente os tipos de dados em muitas instâncias. *Pode levar a bugs no código.
Dois operadores de comparação disponíveis
== verifica a igualdade, independente do tipo de dados

Ex: ' 42 ' == 42 é true!
=== verifica valores iguais e tipos de dados
Ex: ' 42 ' === 42 é false
Operador não igual
!= verifica a não igualdade, independente do tipo de dados
!== verifica se há valores e tipos de dados não iguais

Declarações if
Ex:

```
const status = 200;
if (status === 200) {
    console.log('ok');
} else if (status === 400){
    console.log('Error');
} else {
    console.log('Unknown Status')
}
```

Formas alternativas de escrever declarações if
* não é { } necessário if se estiver usando uma única linha
Ex:

```
const status = 200;
if (status === 200) console.log('ok');
else if (status === 400)console.log('Error');
else console.log('Unknown Status');
```

Ternário
Ex:

```
const message = (status === 200) ? 'ok' : 'Error'
```

< Menor
<= Menor igual
> Maior
>= Maior igual
== valores iguais
=== valores e tipagem igual
&& E
|| OU
!== Não igual
! negação
Notas booleanas
Valores falsos implícitos
Strings *strings vazias testadas como falsas
Objects * objetos null e undefined são testadas como falsos
Numbers * 0 testa como falso

Criando Array
O que é um Array?
Listas ou coleções de valores: arrays podem conter muitos valores diferentes de vários tipos de dados diferentes.

Cada elemento possui um index: um index é um valor numérico único que representa os dados dentro de um array.
Array length: depois que um array é criado, você pode verificar seu tamanho a qualquer momento usando a propriedade length. Ex: arrayName.length
Retornando o tamanho de um array
Ex:

```
let arrayLength = 5
let arr1 = [];
let arr2 = Array(arrayLength);
console.log(arr1.length)
console.log(arr2.length)
```

Criando um array
Ex:

```
let animais = ['cachorro','gato','girafa','cavalo','macaco'];
console.log(animais);
let idiomas = new Array('Ingles','Portugues', 'coreano', 'Japones','Frances');
console.log(idiomas);
```

Acessar elemento de um array
Ex:

```
let idiomas = new Array('Ingles','Portugues', 'coreano', 'Japones','Frances');
let ingles = idiomas[0];
console.log(ingles)
```

Medindo tamanho de um array(length)
Ex:

```
let frutas = ['Banana','Laranja', 'Maçã','Jaca', 'Morango','Pera'];
console.log(frutas.length);
```

Acessar o último elemento:
Ex:

```
let frutas = ['Banana','Laranja', 'Maçã','Jaca', 'Morango','Pera'];
let pera = frutas[frutas.length -1];
console.log(pera);
```

Adicionando dados para um Array
Durante a criação de um array, você pode criar um array com dados de uma instrução.
Após a criação de um array, você pode adicionar dados para um array após terem sido criados. Este método requer uma atribuição de um valor para um índice. Os índices que já possuem um valor podem ser substituídos. Acompanhar o tamanho de um array é importante caso ele tenha um tamanho fixo.
Como posso acessar os dados? Os valores de um array podem ser acessados por seu índice.
Ex:
Adicionando dados durante a criação de um array

```
let arr1 = ["A", true, 2];
console.log(arr1[0]);
console.log(arr1[1]);
```

Ex:
Adicionando dados depois da criação de um array

```
let arrayLength = 2;
let arr2 = Array(arrayLength);
```

arr2[0] = "Value at index 0";
console.log(arr2[0]);
console.log(arr2[1]); * não possui um valor no índice
Length e Index
Length: Soma dos espaços que foram alocados para valores no array
Index: Enumera os valores de um array começando o seu índice em 0.
A propriedade length sempre será maior do que a propriedade index
Porque no index começa em 0.
O índice do último item de um array será o tamanho subtraído por 1.

Métodos do Array
Manipulando arrays
Push e pop - afeta o final do array
array.push(values) adiciona um ou mais valores ao final de um array e retorna o novo tamanho desse array.
array.pop( ) remove o último valor de um array e retorna o valor do array removido.
Ex:

let arr1 = ["A", true, 2];
console.log(arr1.push("new value")); * adiciona um valor
console.log(arr1);
console.log(arr1.pop()); * remove o último valor
console.log(arr1);

Shift e Unshift - afeta o início do array
array.shift( ) remove o primeiro valor do array e retorna o valor removido.
array.unshift(values) adiciona um ou mais valores no início de um array e retorna valor do tamanho do array atualizado.
Ex:
let arr1 = ["A", true, 2];
console.log(arr1.unshift("hello"));
console.log(arr1);
console.log(arr1.shift()); *remove o primeiro valor.
console.log(arr1);

Concat
Une dois arrays para criar um novo array
Ex:
let arr1 = ["A", true, 2];
let arr2 = ["B", false, 3];
let newArr = arr1.concat(arr2)
let newArr2 = arr2.concat([1,2,3])
console.log(newArr);
console.log(newArr2);

Laços em javascript
Execute o código várias vezes
- valor codificado
- iterar em uma lista
- verificar se algo é true

Tipo de loops comuns
- while
- for
- for ... of

Laço while
Ex:
```
const nomes = ['cintia', 'glaucia', 'diego'];
let index = 0;
while (index < nomes.length){
    const nome = nomes [index];
    console.log(nome);
    index++;
}
```
Laço for
Ex:
```
const nomes = ['cintia', 'glaucia','diego'];
for(let index = 0; index < nomes.length; index++){
    const nome = nomes[index];
    console.log(nome);
}
```

Escolhendo o laço certo
Laço while : ao chamar uma função que retorna falso ou nulo quando for concluída
Laço for: realizar loop um número conhecido de vezes
Laço for ... of : iteração de uma coleção de itens

O que são funções?
Um bloco de código que executa uma tarefa de rotina usando uma série de instruções
Ex:
printThanks - imprime uma mensagem fixa para todos.
```
function printThanks() {
    console.log("Thanks for shopping");
    console.log("discounts expira dec 1");
}
```
computePrice - retorna o resultado personalizado com base nos dados de entrada fornecidos para computação.
```
function computePrice(cost, discount) {
    let reduction = cost*discount;
    console.log("You saved $" + reduction);
    return cost-reduction;
}
```

Porque as funções são úteis?
Legibilidade e concisão: digamos que uma tarefa leve 10 linhas de código, e é usado 5 vezes em seu programa, reduza o tamanho do código, torne a funcionalidade clara.
Capacidade de manutenção: precisa corrigir um bug nas instruções de execução? ou mudar o código de execução? faça isso em um só lugar(não em 5 lugares separados)

Definição da sintaxe de uma função em js

function é uma palavra reservada que identificada como uma função
O nome da função é usado para inovar(executar)funções
parâmetros da função são marcadores de posição para entrada
return(retorno) da função define pontos de saída, espaço reservado para dados de saída(resultado)
Uso da função : invocação
Definição
tem { }, sem ponto e vírgula, é um plano para as etapas que precisam ser executadas para executar a tarefa
Invocação
( ) operador, ponto e vírgula terminado
É uma instrução que executa o código da função, parâmetros substituídos por argumentos (entradas), resultando em execução personalizada.

Exemplo de funções:

```
function raizQuadrada(numero) {
    return numero * número;
}
console.log(raizQuadrada(10));

function somaNumb(numero) {
    return numero + numero;
}
console.log(somaNumb(10));

function descontoProduto(preco, desconto) {
    let resultado = preco * desconto;
    console.log("Voce economizou.....:R$  " + resultado);
     return preco - desconto;
}
console.log(descontoProduto(100, 10));
```

Arrow functions
(param) => { return param};

- são expressadas por =>
- às vezes são chamadas de funções de seta gorda / fat arrow functions
- altera para o conteúdo this
- possui suporte a valores de retorno implícitos
- deve ser atribuído a uma variável ou ser usado imediatamente
- redução de caracteres digitados

Exemplo:

```
const func = (a,b) => {
   let sum = a+b;
   return sum;
}
console.log(func(1,2))
```

O que é JSON?
json = javaScript Object Notation
A linguagem é independente e legível pelo usuário.

Depende apenas de 2 estruturas de dados que são encontradas em todas as linguagens de programação modernas.
Coleção de pares nome/valor, comparável às objetos(objects)
Lista ordenada de valores(values) comparando a arrays pense na capacidade de empacotar coleções de objetos em um único arquivo para transferência.

Objetos em javaScript
O que são objetos?
No mundo real está cheio de objetos!
Ex: livros, músicas, bibliotecas, animais...
Objetos reais tem atributos associados
Ex: book title, book author, book availability (at library)
Objetos reais tem ações associadas
Ex: verificar o livro(da biblioteca)
O que são objetos em javaScript?
representações de objetos do mundo real usando código
Ex: bool, song, library, playlist
Objetos javaScript tem propriedades associadas
Ex: book has title, author, isAvailable
Objetos javaScript tem métodos associados
Ex: you can checkin, checkOut a book