console.log("") --> imprime o conteúdo
Utiliza do mesmo princípio do System.out.println e a concatenação para imprimir múltiplos dados
Outra maneira é usando interpolação:
console.log(`o ${variavel} tem ${variavel2}`)

---------- Inserção no htlm ----------
<script src=""></script>
Normalmente no body. Colocar no final melhora a velocidade.

---------- Comentários ----------
//
/* */

---------- Tipos de Dados ----------

+String --> cadeia de caracteres
""
''
`` --> template literals
permite strings de múltiplas linhas
provê uma maneira fácil de interpolar variáveis e expressões em strings. O método é chamado de interpolação e sua sintaxe é ${...}.

Se colocarmos aspas simples, então colocamos a string delimitada por aspas duplas. O inverso também vale.
As crases permitem usar tanto simples como duplas no meio do texto. Além disso, elas permitem múltiplas linhas. Por fim, permitem incluir expressões dentro da própria string.
O caracter \ torna caracteres especiais como ", ' e o próprio \ em caracteres normais. Além disso, podemos usar o \ para quebrar uma string em duas linhas (no código).

*** String Methods.
Apesar de Strings serem dados primitivos, o javascript considera elas como objetos quando executa métodos
Todos os métodos de strings retornam uma nova string, não alterando a string que chamou o método.

string.length --> comprimento da string
.slice(start, end) --> retorna o pedaço da string especificado. (end não é incluso)
se um dos parâmetros for negativo, é contado desde o fim da string.
ao omitir o últmimo parâmetro, irá pegar até o fim da string
.substring(start, end) --> similiar à slide() com a diferença que valores negativos são tratados como 0.
.substr(start, length) --> novamente similar, apenas muda o segundo parâmetro.
se um dos parâmetros for negativo, é contado desde o fim da string.
.replace("isso","por isso") --> substitui um determinado valor com outro. Substitui apenas o primeiro que encontrar.
/palavra/i --> serve para ignorar o case sensitive
/palavra/g --> para substituir todas os padrões que encontrar
.toUpperCase()
.toLowerCase()
.concat() --> para concatenar várias strings as separa, em ordem, por virgula
.trim() --> remove espaços do fim e inicio
.trimStart()
.trimEnd()
.padStart(quantidade,"string") e padEnd(quantidade, "string") --> adiciona no início ou fim, respectivamente, a string especifica caso o comprimento da string que chamou o método for menor que quantidade. Caso os caracteres em string não sejam suficientes, começa a repetí-los.
.chatAt(num) --> retorna o caractere no índice especificado
retorna uma string vazia se estiver além do comprimento da string
.charCodeAt(num) --> retorna o unicode do caractere no índice especificado
[] --> retorna undefined se estiver além do comprimento da string
apenas leitura
.split("") --> quebra a string nos caracteres passados como parâmetro como uma string
"" retornará um array de caracteres
se o separador for omitido retornará um array com a string como único elemento.
.join("") --> junta strings separadas. Recebe como argumento o separador das diferentes strings na string final.

.indexOf() --> retorna o index da primeira ocorrência da string passada como argumento.
retorna -1 se não encontrar nada
aceita um número como segundo parâmetro, o qual especifica o index de início da busca
.lastIndexOf() --> retorna o index da última ocorrência da string passada como argumento.
retorna -1 se não encontrar nada
aceita um número como segundo parâmetro, o qual especifica o index de início da busca, mas busca desse index até o inicio da string, pois o método procura do fim para o começo.
.search() --> similar ao indexOf, mas não aceita segundo parâmetro e aceita expressões regulares
.match() --> procura na string usando uma expressão regular
retorna um array, contendo os itens encontrados. (/g necessário para retornar todos, senão retonar apenas o primeiro.
.includes() --> retorna verdadeiro se contém uma determina substring
aceita um segundo parâmetro, especificando o index de inicio
.startsWith() --> retorna verdadeiro se a string começa com o valor passado.
aceita um argumento opcional, um número, indicando o início da busca.
.endsWith() --> retorna verdadeiro se a string termina com o valor passado.
aceita um argumento opcional, um número, indicando a largura para procurar(desde o início)

+number --> número
inteiros
reais
not a number (NaN)
Infinity --> infinito, escrito desta forma.

expoentes: 123e5 --> 123 . 10^5

Números em javascript são sempre floats de 64 bits.
A adição de um número e uma string resulta na concatenação.
20 + 30 + "40" = "7040"
"20" + 30 + 40 = "203040"
O interpretado trabalha da esquerda para a direita.

Operações com números em forma de string funcionam, menos na adição.

isNaN(number); --> true, se não for um número
Number() --> retorna um número, convertido a partir de seu argumento.
Quando uma data é passada retorna a quantidade de milisegundos desde 1.1.1970
parseFloat() --> analisa a string passada e retorna um número. Espaços são permitidos. Apenas o primeiro número é retornado.
parseInt() --> analisa a string passada e retorna um número inteiro. Espaços são permitidos e apenas o primeiro número é retornado

*** Number methods
Assim como no caso de strings, métodos estão disponíveis para números no javascript, mesmo que eles sejam primitivos.

number.toString() --> retorna o número como uma string. (funciona com variáveis, literais e expressões.
.toExponential() --> retorna uma string, com o número arredondado e escrito usando notação exponencial.
Recebe um parâmetro opcional, indicando a quantidade de casas decimais. Se não for especificado, javascript não irá arredondar.
.toFixed(number) --> retorna uma string, com o número escrito com uma quantidade específica de casas decimais.
.toPrecision(number) --> retorna uma string, com o número escrito com um determinado comprimento.
.valueOf() --> retorna um número como um número. (util para objetos number)

Number.MAX_VALUE
Number.MIN_VALUE
Number.POSITIVE_INFINITY --> retornado em overflow
Number.NEGATIVE_INFINITY --> retornado em overflow
Number.NaN

+boolean --> booleano
    true
    false

+Undefined
    undefined --> indefinido
        não existe

+Null
    null --> nulo (objeto sem nada)
    diferente de undefined
        Existe, mas não tem nada dentro

+Object
    Objeto, com propriedades/atributos e funcionalidades/métodos.
    Como criar um objeto:
    {
        propriedade: "valor",
        outra: "valor2",

        andar: function(){
            console.log('andar')
        }
    }

    O par nome: valor é chamado de propriedade. Podemos acessar as propriedades de um objeto de duas maneiras:
        objeto.propriedade ou objeto["propriedade"]

    Acessamos métodos assim:
        objeto.metodo();
        Sem o parêntesis irá retornar a definiçao da função.

    Ao que o this se refere depende de onde estiver sendo usado ou chamado.
        Sozinho ou em uma função, , se refere ao objeto global.

    Comparar dois objetos sempre retorna falso.

+Array
    vetor

    Entre colchetes, items separados por vírgula.
    Os tipos dos elementos podem ser diferentes.
    É prática comum declarar arrays usando const.

    Primitivos --> não são objetos.
                   são imutáveis.
        String
        Number
        Boolen
        undefined
        Symbol
        BigInt

    Estruturais
        Object
            Array
            Map
            Set
            Date
            ...
        Function

    Estruturais primitivos
        null

------------------- Variáveis ------------------
Como criar variáveis:

var --> declara uma variável como global. Evita-se usá-lo no js moderno.
var clima = "Quente"

let --> mais moderno do que em relação ao var.
let clima = "Quente"

const --> não pode mudar seu valor durante o programa. Precisam ter seu valor dado no momento da declaração. Mas podemoos redeclarar.
const clima = "Quente"
Sempre declarar variáveis que sabemos que seu valor não mudará usando const.
É possível mudar os elementos de um array constante. Mas é impossível reatribuir o array. A mesma coisa com objetos constantes.

Se não termos um valor para uma variável, o seu valor será undefined.

JS é fracamente tipado. Logo, o tipo de um variável pode mudar em meio ao programa.
Hoisting --> pega a declaração da variável e coloca no início do bloco. Logo podemos ter undefined mesmo antes da declaração de uma variável. Além disso, mesmo se tiver fora do escopo. Isso ocorre com variáveis var.
O mesmo não ocorre com let e const. As variáveis são disponíveis apenas no escopo local e não podem ser referenciadas antes de serem declaradas.

JS é case-sensitive. Aceita a cadeia de caracteres Unicode.
Camel_case

É possível redeclarar variáveis usando var, sendo que o valor delas não são apagados. O mesmo não é possível usando let ou const.

---------- Funções ----------

Obs: dá para criar uma variável dentro de uma função sem usar as palavras chaces
e ela acabar por existir em todo o código. Não faça isso.

Declaração de funções:

```
function nomeFuncao(){
    // código
}
```

Execução:

```
nomeFuncao()
```

Parâmetros são colocados entre os parêntesis e não é necessário declarar tipo de retorno.

---------- Function expression or Function anonymous

```
const sum = function(){

}
```

Execução:

```
sum()
```

--------- Parâmetros e argumentos

```
const sum = function(number1, number2){

}
```

sum(2,3) --> 2 e 3 são argumentos
--> number1 e number2 são parâmtros

--------- Retorno
Nesta construção sum não receberá total, sum receberá a função.

```
const sum = function(number1, number2){
    let total = number1 + number2
    return total
```

```
}

    Para receber total, devemos criar uma variável e:
    const retorno = sum(1,2)

--------- Function Hoisting
    Hoisting ocorre com funções também.
        Porém não ocorre com funções do tipo Function expression

--------- Arrow Function ( tipo expression)

const sayMyname = () => {
    console.log('Daniel')
}

Maneira mais enxuta de se declarar uma função


--------- Callback function
    É uma função passada como parâmetro

--------- function constructor
    Expressão new (usada na frente da função, automáticamente retornando um objeto)
    Cria um novo objeto
    this keyword

--------- Global Function
    Funções globais podem ser executadas via console

    globalThis.nomeFuncao = nomeFuncao
    globalThis e window são equivalentes no console
---------- Manipulando Dados ----------

------ Prototype
Javascript é dita como uma linguagem baseada em protótipos

Cadeia de protótipos
__proto__ --> propriedade de um objeto que nos permite acessar todas as propriedades de uma
maneira bem clara.

------ Type conversion e Type coersion

Type conversion --> explicitamente altera um dado para outro
Type coersion --> o javascript força a troca
    '9' + 5 --> '9' + '5' --> '95' Type coersion
     Number('9') + 5 --> 9 + 5 --> 14 Type conversion

------ Manipulando strings e Números
String para número
    Number(string)

Numero para string
    String(number)

Fixar casas decimais
    number.toFixed(N) --> fixa a quantidade de casas decimais de number em N

Criando array com construtor
    new Array() --> como argumentos recebe os valores

    .length funciona com arrays

Strings para Arrays
    Array.from("string")

Manipulando Arrays
    Adicionar um item no fim
        vetor.push("")
    Adicionar no começo
```

```
        vetor.unshift("")
    Remover do fim
        vetor.pop()
    Remover do começo
        vetor.shift()
    Pegar alguns elementos do array
        vetor.slice(1,3) --> começa em 1, não 0
    Remover 1 ou mais elementos
        vetor.splice(começo, quantidade) --> começa em 0
    Encontrar posição de um elemento
        vetor.indexOf("")

---------- Expressões e Operadores ----------

Toda expressão no javascript pode ou não terminar com ;

new --> expressão para criar um novo objeto

typeof nomeVariavel --> retorna o tipo da variável

delete objeto.propriedade --> procura pela propriedade especificada de um objeto e a deleta.

------- Operadores Aritméticos
    Operações padrão não sofrem alteração (incluindo resto e incremento/decremento)
    Exponencial: **

------- Operadores de Comparação
    Igual a: ==
    Diferente d: !=
        (comparam apenas o valor)
        1 == "1"  --> true
        1 != "1"  --> false

    Estritamente igual a: ===
    Estritamente diferente de: !==
        (comparam o valor e o tipo)
        1 === "1" --> false
        1 !== "1" --> true

--------
Operadores de maior e menor se mantêm.
Operadores de atribuição se mantêm.
Operadores lógicos se mantêm.

-------- Operadores de String
    Comparação --> usamos o ==
    Concatenação --> usamos o +
                += funciona, com o mesmo princípio

-------- Falsy e Truthy

Falsy --> quando um valor é considerado false em contextos onde um booleano é obrigatório
    false
    0
    -0
    ""
    null
    undefined
    NaN

Truthy --> quando um valor é considerado true me contetos onde um booleano éobrigatório
    true
    {}
    []
    1
    3.23
    "0"
    "false"
    -1
```

```
    Infinity
    -Infinity

Em ambos os casos ocorre type coersion

--------------------------- Controle de Fluxo -----------------------------

----- if...else

if(true){

} else {

}

----- Switch

switch(expression) {
    case 'a':
        //código
        break;
    default:
        break;
}


----- Throw
    Disparar erros

function sayMyName(name = ""){
    if(name === ""){
        throw new Error("Nome é necessário")
    }
}

podemos disparar qualquer mensagem, não apenas um erro

----- Try/Catch
try{
    sayMyName()
} catch(e){
    console.log(e) --> vai imprimir "Nome é necessário"
}

Um erro que seja disparado e não pego irá ocasionar um erro de Uncaught

------------------------- Estruturas de repetição ----------------------

break e continue --> mesmos comportamentos


----- For
for(let i = 0; i < 10; i++){

}

----- While
while(true){
    break;
}

----- for...of

let name = 'Mayk'

for(let char of name){
    console.log(char)
}
    // Irá imprimir Mayk letra a letra
```

Pode ser aplicado em arrays também.

----- for...in
loop em cima de um objeto

```
let person = {
    name: 'John',
    age: 30,
    weight: 88.6
}

for(let property in person){
    console.log(property)
    console.log(person[property])
}
```

a variável property irá armazenar o identificador de uma propriedade a cada iteração
o segundo console faz acesso ao valor da propriedade

---------- DOM (Document Object Model) ----------
É o HTML convertido para um Objeto JavaScript
API que representa e interage com o HTML.

Serve para manipular o HTML com o JavaScript

**** getElementById

```
const element = document.getElementById('nome-do-Id');
```

**** getElementsByClassName

```
const elements = document.getElementsByClassName('nome-da-classe');
```

elements é uma html colection.

**** getElementsByTagName

```
const elements = document.getElementsByTagName('nome-da-tag');
```

elements é uma html colection.

**** querySelector (seletores usados no css)

```
const element = document.querySelector('.exemplo');
```

Pega o primeiro elemento com o seletor dado.

**** querySelectorAll

```
const elements = document.querySelectorAll('.exemplo');
```

Pega todos os elementos com o seletor dado.
elements é um NodeList. Possível usar o forEach

Dado um element obtido por algum dos métodos acima

**** textContent --> retorna todo conteúdo visível, incluindo o de nós filhos

```
element.textContent = 'Texto'; ---> altera o conteúdo do elemento
                += 'Texto'; ---> concatena a string no final do conteúdo
```

O método também retorna o texto quando invocado em outras situações.

**** innerText --> retorna o texto visível em um nó, não incluindo nós filhos

```
element.innerText = 'Texto'; --> muda o texto interno do elemento html
```

**** innerHTML

```
element.innerHTML = '<small>texto</small>' --> se quisermos usar html precisamos usar este método
```

**** value

```
element.value = "alterando-valor"
```

Manipula o valor em um input.

**** setAttribute
element.setAttribute('nome-atributo', 'valor');

O método também retorna o valor.

**** removeAttribute
element.removeAttribute('nome-atributo');

Remove o atributo.

Outros

**** createElement('element')

Cria um elemento vazio./

**** element.appendChild('childElement')

Faz append de childElement como filho de element.

Outra maneira de fazer o mesmo que este método é usar string literals e ir usando o operador de concatenação no innerHtml.

**** element.cloneNode()

Clona element. Clona apenas a casca, sem o conteúdo.
Se passarmos true como argumento, um clone de tudo, incluindo o conteúdo, é feito. Clone de profundidade.

**** remove()

Remove o elemeto que chama o método.

---------- Eventos ----------
Eventos são ações que acontecem a partir de interações do usuário.
Javascript nos permite executar códigos quando algum evento ocorre. Fazemos isso usando handler attributes:

<element event="codigo javascript">
event é um handler attribute

onclick --> handler attribute que executa o código especificado quando se clica no elemento
currentTarget --> referencia o elemento que está chamando o evento
target --> varia de acordo com onde se clicou dentro do elemento
classlist --> lista as classes
onchange --> quando o elemento HTML é mudado
onload --> quando o browser termina de carregar a página
onkeydown --> quando o usuário aperta alguma tecla
onmouseover --> quando o cursor passa pelo elemento
onmouseout --> quando o cursor sai do elemento

onsubmit --> evento de envio de um formulário.
form.onsubmit = function (event) {
event.preventDefault(); --> impede que o formulário seja enviado, no caso de ser uma função passada para onsubmit
}
Essa alternativa faz com que apenas o último definido seja impedido de mandar o formulário. Para fazer mais de um, usamos addEventListener:
addEventListener('nomedoevento',function(event);
nomedoevento = submit, neste exemplo

EventListener --> fica observando o elemento. Podemos passar, quando adicionamos um EventListener, qual evento deverá ser escutado e qual ação tomar quando ele acontecer.
'nomedoevento' pode ser click, input.

---------- Useful Methods ----------

**** write --> imprime no html o que estiver entre parêntesis. Deve apenas ser utilizado para testes.
document.write();

**** alert --> cria um alerta na tela e imprime o conteúdo
window.alert();

Especificar o window é opcional.

**** window.print() --> imprime (literalmente) o conteúdo da página

---------- Regex no Javascript ----------
Expressões regulares
Tecnologia utilizada em todas as linguagens para buscar por padrões

Leitura da esquerda para a direita, um caractere de cada vez.
Necessário conhecer os caracteres reservados da tecnologia.

/ expression / flags

Para usar caracteres especiais usamos o \ antes

const re = /foo/ --> as barras determinam uma expressão regular
const re = new RegExp(/foo/) --> ou "/foo/" como argumento

g --> global, busca no texto todo
/foo/g

frase --> pesquisa exata
^frase --> inicio da string
Dev$ --> fim da string

[] --> procura qualquer um (caracteres dentro do colchete)
[A-Z] --> procura caracteres no intervalo
[^xyz] --> não tem x,y,z.

^4\d{0,15} --> inicia com um 4, seguido de 15 digitos
5[1-5] --> inicia com um 5, seguido de um digito entre 1 e 5

| --> ou lógico

----- Funções que aceitam regex como argumento:
match() --> agrupa os padrões em um array
search() --> pesquisa se existe ou não o padrão
replace() --> substitui os padrões por um novo valor

---------- Boas práticas ----------
const em variáveis que não forem mudar
CamelCase

---------- Bibliotecas Externas ----------
imask.js.org --> criação de máscaras sobre elementos input