**Curso JavaScript**

**Comentários:**

* // - uma linha
* /\* \*/ - mais de uma linha

**Identificadores:**

* Podem começar com letra, $ ou \_
* Não podem começar com números, conter espaços e ser palavras reservadas
* É possível utilizar letras, números, acentos, símbolos

**Variáveis:**

* **var:** estará acessível em qualquer escopo
* **let:** estará acessível apenas no escopo onde foi criada, para cima
* **const:** seu valor não pode ser mais alterado ao ser declarado

**Data Types:**

* Number
  + Infinity, NaN
* String
* Boolean
* Null
* Undefined
* Object
  + Array
* Function

Se necessário utilizar o typeof para saber qual o tipo

**String > Número:**

* Number.parseInt(n)
* Number.parseFloat(n)
* Number(n)

**Número > String:**

* String(n)
* n.toString()

**Fomatando string:**

* `${n}` // usa template string, necessário utilizar crase
* + n // usa concatenação
* n.lenght
* n.toUpperCase()
* n.toLowerCase()

**Formatando números:**

* n.toFixed()
* n.toFixed().replace(‘.’, ‘,’)
* n.to.LocaleString(‘pt-BR’, {style: ‘currency’, currency: ‘BRL’’})

**Operadores:**

* aritiméticos
  + +
  + –
  + \*
  + /
  + % //resto da divisão inteira
  + \*\* //ao quadrado
* Atribuição
  + var a = 6 \* 2 / d
  + n += 4
  + n ++ (existe o pré e pós incremento ++n ou n++)
    - n++: usa e depois incrementa
    - ++n: incrementa e depois usa
* Relacionais
  + >
  + <
  + >=
  + <=
  + ==
  + !=
  + Identidade: ===
* Lógicos
  + ! (negação)
  + && (conjunção)
  + || (disjunção)
* Ternário
  + ?
  + :
  + Ex: teste ? true : false
* Typeof (não será visto nesse curso)
* Precedência: () -> \*\* -> \* / % -> + -
* Precedência: relacionais
* Precedência: ! -> && -> ||
* Bitwise
  + 11 (00001011) **&** 14 (00001110) = 10 (00001010) 🡪 Retorna valor 1 ou 0 se tiver equivalência entre os valores, se não tiver equivalência retorna 0
  + 10 (00001010) **|** 11 (00001011) = 10 (00001011) 🡪 Retorna valor 1 ou 0 se tiver equivalência entre os valores, se não tiver equivalência retorna 1
  + 10 (00001010) **^** 11 (00001011) = 1 (00000001) 🡪 Se tiver equivalência, retorna 0, se não tiver equivalência e tiver 1, retorna 1
  + Deslocamento de bit:
    - **N1 << 2** deslocamento para a esquerda (dobra o valor)
    - **N1 >> 1** deslocamento para a direita (reduz o valor pela metade)
* Spread
  + Espalha/quebra um array e devolve elemento a elemento
  + let n3 = [...n1] 🡪 copiar um array para outro
  + let n3 = [...n1,... n2] 🡪 copia duas arrays
  + const jogador3 = {...jogador1,...jogador2} 🡪 copia sempre as propriedades do último objeto, concatenando com as que existem apenas no primeiro objeto
  + const objs2 = [...document.getElementByTagName] 🡪 transforma em array

**DOM – Document Object Model**

* **Árvore DOM**
  + window
    - location
    - document
      * html
        + head

meta, title, etc

* + - * + body

h1, p, div, etc

strong

* + - history
* **Métodos de acesso:**
  + Marca: getElementsByTagName()
  + ID: getElementByID()
  + Nome: getElementsByName()
  + Classe: getElementsByClassName()
  + Seletor, mais prático para alterações rápidas
    - Obtém o elemento que eu identificar para ele
    - querySelector()
      * retorna o primeiro element que encontrar da chave especificada
    - querySelectorAll()
      * retorna todos os elementos que encontrar da chave especificada
      * posso especificar mais de um, por exemplo, (“div, p”) ou ("div[class]") ou ("div > p")
* **Eventos**

Eventos são adicionados com addEventListener. O tipo do evento vai entre aspas, como 'click', 'mouseover'...Javascript dom events list

* + onclick, mouseenter, mouseout, click

**Condições:**

* **Tipos de condição:**
  + Condição **simples**:
    - if (condição) {true}
  + Condição **composta**:
    - if (condição) {true} else {false}
  + Condições **aninhadas**: fazer condição dentro de outra condição
    - if (cond1) {bloco1} else if (cond2) {bloco2} else {bloco3}}
  + Condição **Múltipla**: tem possibilidade de valores múltiplos
    - switch (expressão) {  
      caso valor 1: break  
      caso valor 2: break

caso valor 3: break

caso valor 4: break  
}

**Repetições:**

* Estrutura de repetição com teste lógico no **início**
  + while (condição){}
* Estrutura de repetição com teste lógico no **final**
  + do {} while (condição)
* Estrutura de repetição com **variável de controle**
  + for (let i = 0; i < num.length; i++){}
  + for(n in num){} 🡪 retorna os índices dentro da coleção
  + for(n of num){} 🡪 retorna os elementos dentro da coleção

**Variáveis Compostas:**

Devem ser capazes de armazenar vários valores em uma mesma estrutura.

* var a = []
  + var a = array (vetor)
  + [] = elementos
  + 0,1,2 = índices
* Um array é uma variável que tem vários elementos. Cada elemento teu seu valor e sua chave de identificação
* Exemplo:
  + let num = [5, 8, 4]
    - índices: 0, 1, 2
* num[3] = 6 = insere um novo elemento no índice escolhido
* num.push(7) = insere um número
* num.length
* num.sort() = coloca os elementos em ordem crescente
* for(let pos in num){} = lê-se “para cada pos em num”. Funciona apenas para arrays e objetos
* num.indexOf() = procura no vetor onde está o valor. Se não tiver o valor será retornado valor -1

**Funções:**

É um bloco de comandos em que eu posso executar quando eu quiser ou precisar. São heterogêneas, podem guardar valores de diferentes tipos.

Uma função pode receber parâmetros e retornar um resultado.

* function nome (parâmetros){} 🡪 declaração da função
* nome() 🡪 chamada da função
* return 🡪 se eu quiser que a função retorne algo. Para ao retornar mesmo havendo outros comandos
* () 🡪 parâmetros que entrarei, por exemplo, (p1, p2)
  + Necessário colocar um valor ao chamar a função
  + Caso existam mais de um parâmetro e eu não passar algum, ele ficará como “undefined” e o retorno será NaN.
  + Nesse caso posso passar no parâmetro mesmo o valor (valor padrão do parâmetro)
    - function soma(*n1*=0, *n2*){}
* **Funções parametrizadas (REST):** não preciso especificar a quantidade de parâmetros
  + function soma (...valores){

return valores.lenght

}

console.log(soma(10,5,2,15,8))

* **Funções anônimas:** não possuem um nome associado ao seu conteúdo, são chamadas em tempo de execução.
  + let teste = function(v1, v2){

return v1 + v2

}

console.log(teste(10,5))

* + **função construtor:** usada em funções mais simples
    - let teste = new Function(“v1”, “v2”, “return v1 + v2”)

console.log(teste(10,5))

* **Arrow Functions:** mecanismo conciso para representar funções anônimas. Usa uma notação que lembra uma seta. Pega os parâmetros e envia para {}.
  + const soma = (param) => {retorno}
  + Se tiver apenas um parâmetro não precisamos utilizar os parênteses
    - const nome = *n* => {return *n*}
  + Não é necessário usar o return e as {} se for uma operação simples
    - const add = *n* => *n* + 10
* **Funções dentro de outra função:** 
  + const soma = (...*valores*) => {

const somar = *val* => {

let res = 0

for(const v of *val*){

res += v

}

return res

}

return somar(*valores*)

}

* **Funções geradoras:**

Uma função geradora tem o seu retorno adiado até o momento que a gente precisa desse retorno. É o “pilar” da programação assíncrona dentro do JS.

Chamei a função e não preciso de todo o processamento para retornar alguma coisa no final e entregar a execução para o programa, eu posso ir retornando coisas ao longo da execução.

Quem tem o controle da execução da função é a própria função. Quando eu a chamo no primeiro momento, ela vai me retornar um iterador e vai me retornar sempre que houver uma declaração yield.

function\* perguntas(){

    const nome = yield 'Qual seu nome?'

    const esporte = yield 'Qual seu esporte favorito?'

    return 'Seu nome é ' + nome + ', seu esporte favorito é ' + esporte

}

const itp = perguntas()

console.log(itp.next().value)

console.log(itp.next('Sara').value)

console.log(itp.next('Vôlei').value)

------------------------------------------------------------------------------------------------

function\* contador(){

    let i = 0

    while(true){

        yield i++

        if(i > 5)

            break

    }

}

const itc = contador()

/\*

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

    console.log(itc.next().value)

}

\*/

for(let c of itc){

    console.log(c)

}

* **Método MAP:**

Tem a funcionalidade de percorrer arrays (coleções) inteiramente. Quando eu precisar trabalhar elemento por elemento de uma coleção.

Itera toda a coleção! Ou seja, não para no meio da execução da função.

array.map(elemento da coleção que está iterando, índice, array)

* **Operador THIS:**

Faz referência a um elemento, normalmente utilizado quando eu quero utilizar o mesmo parâmetro, mas sem alterar seu nome.

function aluno(nome, nota){

this.nome = nome

this.nota = nota

}

**Objetos:**

Para declarar uma variável é possível declarar com {}. É possível inserir diretamente um identificador.

São variáveis que guardam valores (atributos) e podem guardar funcionalidades (métodos).

* let amigo = {nome:’José’, sexo:’M’, peso:85.4, engordar(p){}}
  + engordar(p){} = cria um parâmetro “engordar”, que guarda uma função (p){}

**OUTROS:**

* .getHours()
* .getDay()
* .getFullYear()
* **Imagens:**
  + .setAttribute('id', 'foto')
    - .setAttribute('src', 'homem.jpg')
  + res.appendChild(img)
* **Radio button:**
  + **Texto

    O conteúdo gerado por IA pode estar incorreto.**Interface gráfica do usuário

    O conteúdo gerado por IA pode estar incorreto.Tem que ser pego pelo name, todos os rádios precisam ter o mesmo name - var fsex = document.getElementsByName('sexo')
  + Para selecionar o primeiro, segundo, etc, radio é necessário chamar por colocação - if (fsex[0].checked)
  + Lembrar de colocar *checked em um dos inputs*
* **Elementos HTML**
  + Ao pegar um elemento DOM usando getElementById(), estamos obtendo uma REFERÊNCIA ao elemento HTML, não ao seu VALOR diretamente
  + Caso eu use **.value** retornarei o valor atual como STRING
  + Caso eu use **.valueOf** retornarei o valor PRIMITIVO do objeto
  + .value:
    - Sem usar para elementos HTML
    - Usar para obter o conteúdo do campo
  + .valueOf
    - Se já convertido, não é necessário utilizar
* isNaN(n)
* **Modo estrito**
  + “use strict”
* **break**: para o programa
* **continue**
* **export**
* **Ctrl+K e depois Ctrl+C** 🡪 Comenta tudo
* **Alt + seta para baixo/cima** 🡪 Move a linha selecionada