JavaScript possui tipagem dinâmica, no caso ele define o tipo enquanto o programa está rodado, ao contrário de outras linguagens como C, C#, Java, onde o tipo é definido ao declarar uma variável.  
Tipos primitivos JavaScript -> Number, string, null, boolean, undefined

Importante é adicionar o script no HTML para localizar o arquivo, nesse caso na HEAD utilize <script src= “Local”></script>

Utilizar na declaração de variáveis var, let ou const, além disso é importante lembrar que seria Case sensitive, ou seja, JavaScript considera variáveis diferentes se utilizar maiúsculo e minúsculo.

var 🡪 Declara uma variável, opcionalmente, inicializando-a com um valor.

let 🡪 Declara uma variável local de escopo do bloco, opcionalmente, inicializando-a com um valor.

const 🡪 Declara uma constante de escopo de bloco, apenas de leitura.

O essencial é declarar a variável no estilo camelCase, começando com uma primeira palavra minúscula e caso necessite de uma segunda deixar junto e iniciar em maiúscula.

Caso seja necessário alterar os valores da variável, utilizar o Let, se não basta utilizar o const na declaração que funcionará sem problemas e ela não poderá ser alterada.

O tipo objeto seria para juntar informações de variáveis em um mesmo lugar.

let pessoa = {};// Definindo um objeto vazio

Para acessar um valor dentro de um objeto, basta adicionar o nome do objeto.nomeValor.  
Exemplo: pessoa.nome

Para definir o que terá dentro do objeto seria propriedade: O que irá dentro da propriedade (Valor).

Seria a combinação de chave e valor, a chave mais conhecida como identificado;

Exemplo:

let pessoa = {

nome: 'Guilherme',

idade: 20

};

Array 🡪 Conjunto de dados que pode ser acessado por um índice.  
Índice 🡪 Posição exata onde se encontra essa informação.

let familia = []; //Definindo um array vazio

Exemplo:

let familia = [26,45,50,17]; //Definindo um array

console.log(familia[0]);

Não precisa estar definido com apenas um tipo dentro dele, pode inserir tipos diferentes de valores.

Nomear funções deve pensar em um verbo + um substantivo por fins didáticos

Exemplo:

function resetaCor(){

};

Assim basta inserir os parâmetros se necessitar e para chamar a função para utilizar o nome com os parênteses.  
Caso a função precise retornar algo, basta utilizar o return.

Tipos de operadores

Operadores aritméticos(matemáticos)

Operadores de atribuição

Operadores de comparação

Operadores lógicos

Operadores BitWise

Operadores aritméticos

+, -, /, \*, %, \*\* -> Exponencial.

++ -> Incremento

-- -> Decremento

Operador de atribuição

= -> Atribui um valor a uma propriedade.

+=

-=

Operadores de igualdade

=== -> Compara o valor e o tipo são iguais

== -> Compara apenas se o valor é igual ignorando o tipo.

//Operador ternário

//Exemplo: Cliente possui mais de 100 reais, premium, se não normal.

let dinheiro = 100;

let tipo = dinheiro > 100 ? 'premium' : 'normal';

console.log(tipo);

Operadores lógicos

&& -> Retorna true se os dois operandos forem verdadeiros

|| -> Retorna true se um dos operandos for verdadeiro

! -> Retorna à negação do que estiver sendo comparado

Comparações não booleanas

Resultado de uma comparação lógica nem sempre será true ou false.

Falsy: Undefined, null, 0, false, ‘’, NaN

Truthy : Qualquer outro não citado acima.

False || true -> true

False || ‘Guilherme’ -> Guilherme

False || 1 -> 1

False || 1 || 3 -> 1 //Avalia apenas a primeira comparação

Exemplo:

var corPersonalizada = “Vermelho”

var corPadrao = “Azul”

var corPerfil = corPersonalizada || corPadrao

Condicionais -> De acordo com o valor da variável ou de acordo com resposta da condição imposta irá efetuar um bloco de comando.

**If...Else**

Exemplo: Horário estiver entre 06:00 até 12:00 🡪 Bom dia

Se estiver entre 12:00 até às 18:00 🡪 Boa tarde

Caso contrário 🡪 Boa noite

If(Condição) {

// Caso positivo

}else if(Outra condição){

//Caso a outra condição for positiva

}else{

//Caso nenhuma condição seja positiva

}

Var hora = 10;

If(hora>=6 && hora<12) {

console.log(“Bom dia!”)

}else if(hora>=12 hora<18){

console.log(“Boa tarde!”)

}else{

console.log(“Boa noite!”)

}

**Switch...case**

Switch(variável){

Case ‘Opção’: Código a ser executado

break

}

Exemplo:

var teste = “comum”

Switch(variável){

Case ‘comum’:

console.log(“Caso comum”)

break;

Case ‘gerente’:

console.log(“Caso gerente”)

break;

Case ‘diretor’:

console.log(“Caso diretor”)

break;

case default:

console.log(“Caso não reconhecido”)

}

Laços de repetição em JavaScript : For, while, Do while, For..in, For..of

**Laço For**

for(declaração ; condição ; incremento){

}

**Laço While**

while(condição){

}

**Laço Do While**

Do{

}while(condição)

**Laço For..in**

Irá percorrer todo o array ou objeto de forma que podemos buscar uma informação específica ou mostrar tudo.

Exemplo:

Const pessoa = {

nome:’Guilherme’

idade:20

}

For(var chave in pessoa){

Console.log(chave);

}

Const cores = [‘Vermelho’,’Azul’,’Verde’]

For(var índice in cores){

Console.log(cores[indice])

}

**Laço For..of**

Const cores = [‘Vermelho’,’Azul’,’Verde’]

For(var cor of cores){

Console.log(cores[cor])

}

Ou seja se deseja localizar um índice seria utilizando o for..in, se precisa localizar especificamente os dados presentes em um array, basta utilizar o for..of

É possível inserir funções dentro de objetos e novos objetos dentro dos mesmos.

Exemplo:

Const telefone{

Marca: “Motorola”,

tamanhoTela: {

vertical: 200,

horizontal: 150

},

capacidadeBateria: 1000,

ligar: function(){

console.log(“Realizando ligação...”)

}

}

Funções de fábrica – Factory Functions

Exemplo:

Function criarCelular(marcaCelular,tamanhoTela,capBateria){

Return {

marcaCelular,

tamanhoTela,

capBateria,

ligar (){

console.log(“Realizando ligação...”)

}

}

}

Exemplo de criar um objeto nessa função:

Var celular1 = criarCelular(“Motorola”,5.5,5000);

Constructor function -> Função que funciona como construtor.

//Pascal Case -> UmDois,Testar,Carro

Function Celular(marcaCelular,tamanhoTela,capacidadeBateria){

This.marcaCelular = marcaCelular;

This.tamanhoTela = tamanhoTela;

This.capacidadeBateria = capacidadeBateria;

This.ligar = function(){

Console.log(“Fazendo ligação...”);

}

}

Const cel = new Celular(“Motorola”, 5.5, 5000)

Natureza dinâmica de objetos

No caso os objetos em JavaScript são dinâmicos, assim é possível incluir novas propriedades, funções e alterar valores durante o desenvolvimento do código.

Para deletar uma propriedade, basta utilizar o delete obj.propriedade

Clonar objetos

Const novoObj = Object.assign({Possibilidade de adicionar mais propriedades}, nome do obj)

Const obj2 = {...celular}

Math -> Biblioteca matemática - <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Math>

Math.random() - > Retorna um número aleatório.

Math.max(num1,num2,num3,num4) -> Retorna o maior valor dentre um intervalo

Math.min(num1,num2,num3,num4) -> Retorna o menor valor dentre um intervalo

String -> Métodos de uma string -> https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global\_Objects/String

Tipo primitivo  
const msg = “Teste de mensagem”

Tipo objeto  
const msg2 = new String(“Teste da segunda mensagem com obj”)

Msg.length -> Retorna o tamanho do string.

Msg[indice] -> Acessa um caractere do string

Msg.includes(“verifica”) -> Verifica se na mensagem aquele string ou caractere está incluído, retornando true ou false.

Msg.startsWith(“verifica”) -> Verifica se a mensagem começa com aquela string.

Msg.endsWith(“verifica”) -> Verifica se a mensagem termina com aquela string.

Msg.indexOf(“verifica”) -> Verifica o início do índice daquela string na mensagem.

Msg.replace(“local”,”valorTrocado”) -> Altera a frase local para a valorTroca desde que esteja na mensagem.

Msg.trim() -> Retira espaços do começo e do final

Msg.split(‘ ’) -> Separa a mensagem ao encontrar um espaço.

Template literal -> //Object {}, boolean True or false, string “” ‘’, Template literal -> ``

Escape notation em JavaScript -> \n, \’ ‘\, entre outros.

Const msg = ` Essa é uma mensagem de ‘teste’

para o curso realizado` -> Nesse caso toda a formação com espaços, quebras de linha e aspas é mantido.

Possibilidade de durante a mensagem utilizar ${variável/função/método/...} para trazer uma informação do Javascript no texto.

Date – Trabalhando com datas -> https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global\_Objects/Date

Const data1 = new Date(); -> Retorna data atual

Const data2 = new Date(“March 11 2022 18:00”);

Const data3 = new Date(2022,2,11,18,30);

Data3.get -> É possível extrair diversas informações

Data3.set -> É possível adicionar/alterar diversas informações

Data3.toDateString() -> Converte a data retornando em String.

Data3.toTimeString() -> Converte a data retornando em String com tempo local.

Data3.toISOString() -> Converte a data retornando em String no formato de banco de dados.

Adicionar elementos em um array;

Inserir no inicio: array.unshift(“Adição”);

Inserir no meio: array.splice(“Indice”,Possivel deletar,”Adição”);

Inserir no final: array.push(“Adição”);

Encontrar elementos em um array(Tipo primitivo)

Exemplo:

Const numeros = [1,2,3,4];

numeros.indexOf(2);

//Retorna o índice do valor que está sendo procurado.

numeros.lastIndexOf(Valor) -> Procura onde esse valor apareceu por último

números.includes(valor) -> Verifica se o valor está presente no array.

Encontrar elementos em um array(Tipo referência)

Const marcas = [{id :1, nome: “a”}, {id :2, nome: “b”}, {id :3, nome: “c”}]

Nesse caso includes não funciona.

Var marca = marcas.find(function(marca){

return marca.nome === “a”;

})

Find JavaScript -> <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/find>

Arrow Functions

Exercício de cima utilizando arrow function

Const marcas = [{id :1, nome: “a”}, {id :2, nome: “b”}, {id :3, nome: “c”}]

Var marca = marcas.find( marca => marca.nome === “a”;)

Removendo elementos do array

Const num = [1,2,3,4,5,6];

Remover no final: num.pop();

Remover no inicio: num.shift();

Remover na posição deseja: num.splice(“Indice”, “Quantidade a ser removida”);

Esvaziando um array

var num = [1,2,3,4,5,6];

Solução 1 ->

Números = []; -> Não pode ser constante

Solução 2 (Melhor forma – Simples e funcional)->

Num.length = 0;

Solução 3 ->

Num.splice(0,números.length);

Solução 4 ->

While(num.length>0){

Num.pop()

}

Combinando/Dividir um array

Const primeiro = [1,2,3];

Const segundo = [4,5,6];

Combinar -> const comb = primeiro.concat(segundo);

Dividir -> const dividido = combinado.slice(1,3) Ou combinado.slice(1), sempre cortará a partir do primeiro parâmetro até o segundo

Tipo referência também irá alterar se utilizar essas funções.

**Imersão DEV**

Variáveis -> Referencias para valores definidos que serão utilizados pelo computador.

Console.log -> Utilizado para mostrar algo no console

toFixed -> Utilizado para limitar as casas decimais

function nomeDaFunção(){

} -> Assim terá uma função criada para ser utilizada quando algo for clicado ou para chama-la durante a execução do código.

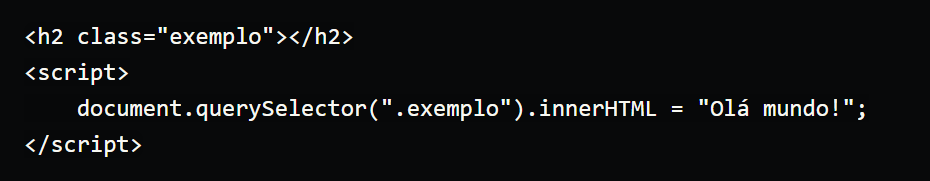
document.getElementById(“ID do elemento”).value -> Retorna o valor de um elemento do HTML de acordo com o ID.

parseInt() -> Transforma a string em inteiro

parseFloat() -> Transforma a string em um número real.

Utilizar o innerHTML quando precisar inserir uma informação dentro de uma tag selecionada no html.

querySelector() -> Retorna o primeiro elemento dentro do documento que foi especificado pelo seletor. O querySelector() aceita os atributos ids, classes e outros parametros de seleção.



getElementById() -> Retorna apenas um elemento selecionando ele pelo ID.

Math.random() -> Traz um número aleatório entre 0 e 1, pode ser multiplicado de acordo com a forma que necessita, como se deseja um número de 1 a 10, deve multiplicar por 11, assim por diante.

É possível utilizar document.write para escrever algo no HTML e aproveitar para utilizar as tags com classes formatadas.

Para criar um array, basta utilizar o [“elemento1”, “elemento2”,..]

Forma de inserir um novo elemento no array

Exemplo:

teste.push(“Novo elemento”);  
Verificar se o final da string termina com algo que desejamos ou não:

nomeVariavel.endsWith(“”);

Tabelas

Para criar uma tabela tem a tag principal <table>

Após isso podemos utilizar <thead> para construir a primeira linha como um cabeçalho da tabela.

Para criar linhas e colunas nesse cabeçalho é necessário utilizar o <tr> e <th>

Sendo <tr> para criar linhas

E <th> para criar as colunas.

Podemos também utilizar <tbody> para demonstrar todo o corpo da tabela.

Onde para criar linhas e colunas seria com <tr> e <td>

Sendo <tr> para linhas

E <td> para colunas