Vscode

Segurando ALT e clicando nos valores, ele entenderá como um só e alterará todos os marcados de uma única vez;

Outra alternativa é utilizando o ctrl+f; ele vai abrir o campo com o item a ser modificado. Clica na setinha e em REPLACEe você insere o novo valor. Ao dar enter ele mudará um por um.

Ctrl + P: busca de arquivos

Ctrl+shift+p: mostra comandos do vscode

Emmet: função que consiste em vários atalhos para criar estrutura de códigos.

Mkdir: cria novas pastas

git.init: inicializa o projeto

Git branch: mostra a branch que se está

 o asterisco indica que teve alteração no arquivo que não foi enviado para o repositório no git

Path: são comandos digitáveis na linha de comando. Ele busca-os na pasta bin

Code: utilizado para rodar o comando

Code + nome da pasta abre a pasta em nova janela

INTRODUÇÃO AO JAVASCRIPT

Aula 1: O que é Javascript?

Introdução ao Javascript

Objetivos

1. Entender a história da linguagem

2. Familiarizar-se com a sintaxe

3. Utilização de recursos básicos

4. Criar seu primeiro projeto

5. Conhecer possibilidades no mercado de trabalho

História

Interpretada: linguagem interpretada, ou seja, ela está sendo rodada em tempo real. O código está sendo disponibilizado instantaneamente. Ele não precisa ser transformado em outra coisa antes de ser enviado para o browser, ao contrário de uma linguagem compilada, como C++, que precisa ser transformado. Se estivermos usando algum framework de compilação no JVS, pode se dizer que essa compilação também é em tempo real enquanto o código está sendo executado.

Baseada em Protótipos: base para maioria das estruturas de dados do JVS. Conjunto de funções e funcionalidades em comum.

Multiparadigma: peremite escolher se você quer escolher com orientação a objetos, programação funcional ou programação estruturada. Atende à sua necessidade não limitando a um paradigma. É uma linguagem flexível.

Comumente utilizada em aplicações ***web client-side*** (parte da programação que vai ter comunicação com o usuário: JVS é a linguagem da internet, da web. É a linguagem que os browsers entendem.

Segue o padrão ECMAScript: padrão para várias linguagens. Conjunto de normas que vai dizer quais funcionalidades oficialmente estão sendo lançadas naquela linguagem. Significa que o JVS está seguindo as novas atualizações.

Evolução

Linha do tempo

Descrição gerada automaticamente

Foi criado em 1995 oficialmente. Em 1997 começou a seguir o padrão ECMASprit.

ECMASprit 3 em 1999: permite o uso de expressões regulares, permitindo, por exemplo, validação de e-mails.

Try/catch: maneira de tratar erros

ECMASprit 5 em 2009: suporte ao JSON (quando quer consumir um API). Algumas funções para alguns array.

ECMASprit 6 em 2015: trouxe muita atualização para o JVS

Gráfico, Gráfico de mapa de árvore

Descrição gerada automaticamente

Muito do que se consegue fazer hoje no JVS deve a essa atualização. Permite, por exemplo, o paradigma da orientação de objetos, declare classes, que o programador consiga se familiarizar, permite formas novas de declarar variáveis e concatená-las em streaming.

Há mais evoluções, mas não tão significativas quanto a 6.

Aplicações

Diagrama

Descrição gerada automaticamente

É uma linguagem muito flexível, não limitando ao mundo web.

RECURSOS BÁSICOS:

Objetivos:

1. Se familiarizar com a sintaxe da linguagem

2. Entender a diferença entre variável e constante

3. Ser capaz de comentar seu código

Manipulando um arquivo JS

Para inserir comentários em uma linha começar com //

Para inserir comentários em várias linhas, iniciar com /\* e fechar com \*/

Ctrl + / insere comentário de linha

Variável: valor dinâmico, permitindo alteração.

CONSTANTE: valor não pode ser manipulado. A boa prática é declará-las em letras maiúsculas, facilitando diferenciação com a variável

FUNÇÕES:

Para declará-la, utiliza-se a palavra function e dá um nome a ela, abrindo parenteses para você declarar atributos

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Para obter o resultado chama o return a+b

Para ver no console, utiliza-se o console.log(a + b)

Para chamar a função, repeti-la atribuindo valores:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

CONSOLE: acessando através da tela de inspeção na página WEB, aba CONSOLE, é possível editar e inserir dados. Utilizado para mostrar ao usuário. Outra forma de acessar e debugar é utilizando o nodeJS, ao lado do desenvolvedor.

CTRL + SHIFT + I: ABRE TELA DE INSPEÇÃO

NODEJS: maneira de enxergar o console diretamente dentro do VScode

returnEvenValues: retorna números pares

Array: lista de dados

JS NUMA PÁGINA DA WEB:

ESTRUTURA DE PROJETO

 <meta name="viewport" content="width=device-width, initial-scale=1.0">

Responsivo para todas mídias

Projetos que utiliza diversos tipos de arquivos devem ser separados por pastas (arquitetura) desenhando toda a estrutura do projeto.

Assets: ferramentas que comporão a página

Link: importa o arquivo

Rel= atributo que relaciona o arquivo que está chamando e o chamado

Stylesheet: folha de estilo

Href= referência de onde está o arquivo

Quando não se coloca nada no meio da tag, não precisa ser fechada, bastando colocar />

Se o script for muito grande, o ideal é colocar o estilo antes do encerramento da tag html

Script: tag do JS. Ele tem que ser fechado por completo, pois é possível colocar informações adicionais.

Src= origem do arquivo script

INTERAGINDO COM ELEMENTOS DO DOM

DOM: Document Object Model= Documentos que existem na página. Estrutura dos elementos dentro da janela. Árvore de elementos DOM.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Criando um computador:

Tagname: uma h1, um p, um html

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

É possível selecionar o elemento específico através da tela de inspeção:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

É possível enviar o comando direto da janela de inspeção para este elemento:

Interface gráfica do usuário, Aplicativo, Word

Descrição gerada automaticamente

CRIANDO UM CONTADOR:

Eventos: propriedades do JS

Pela regra do HTML só é possível uma única ID para cada elemento

innerHTML é o html que está dentro do elemento

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="styles.css">

    <title>Contador</title>

</head>

<body>

    <h1>Contador</h1>

    <p>Este é meu primeiro projeto JavaScript pela DIO</p>

    <div id="counter">

        <button name="subtrair" onclick="decrement()">-</button>

        <span id="currentNumber">0</span>

       <button name="adicionar" onclick="increment()">+</button>

    </div>

    <script src="scripts.js"></script>

</body>

</html>

Script

var currentNumberWrapper = document.getElementById('currentNumber');

var currentNumber = 0;

function increment() {

    currentNumber = currentNumber + 1;

    currentNumberWrapper.innerHTML = currentNumber;

}

function decrement() {

    currentNumber = currentNumber - 1;

    currentNumberWrapper.innerHTML = currentNumber;

}

Style

@import url('https://fonts.googleapis.com/css2?family=Open+Sans:wght@300;600&display=swap');

\* {

    font-family: 'Open Sans', sans-serif;

}

body {

    display: flex;

    align-items: center;

    justify-content: center;

    flex-direction: column;

    background-color: rgb(177, 225, 255);

}

button {

    height: 40px;

    width: 40px;

    border-radius: 50%;

    border: 2px solid lightcoral;

    background-color: rgb(241, 214, 214);

    color: lightcoral;

    font-size: 16pt;

    font-weight: 600;

    text-align: center;

}

#counter {

    display: flex;

}

#currentNumber {

    display: flex;

    align-items: center;

    margin: 0 20px;

    font-size: 16pt;

}

PROJETO TO-DO-LIST

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="styles.css">

    <title>To-do-list</title>

</head>

<body>

    <h1>To-do-list</h1>

    <!-- Começando a criar a página -->

    <div id="list"> <!-- Aqui é para criar os itens em lista -->

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

<form id="task-form"> <!-- Inserindo o formulário para usuário preencher -->

            <input id="task-input" name="tarefa" type="text" placeholder="Digite aqui sua tarefa" /> <!-- código pedindo para usuário inserir uma tarefa no formato texto -->

            <button type="submit">Adicionar</button> <!-- Cria o botão para adicionar a informação digitada em lista -->

        </form>

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

HTML completo do projeto:

<!DOCTYPE html> <!—dar comando emmit e clicar em html5 já vai abrir o head completo -->

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="styles.css">

    <title>To-do-list</title>

</head>

<body>

    <h1>To-do-list</h1>

    <!-- Começando a criar a página -->

    <div id="list"> <!-- Aqui é para criar os itens em lista -->

        <form id="task-form"> <!-- Inserindo o formulário para usuário preencher -->

            <input id="task-input" name="tarefa" type="text" placeholder="Digite aqui sua tarefa" /> <!-- código pedindo para usuário inserir uma tarefa no formato texto -->

            <button type="submit">Adicionar</button> <!-- Cria o botão para adicionar a informação digitada em lista -->

        </form>

        <div id="tasks"></div>

    </div>

    <script src="scripts.js"></script>

</body>

</html>

Analisando o Script

// os código abaixo vão criar o formulário e o id para receber as tarefas do usuário //

const form = document.getElementById('task-form');

const taskList = document.getElementById('tasks');

form.onsubmit = function (e) {

    e.preventDefault();

    // criação do input //

    const inputField = document.getElementById('task-input');

    addTask(inputField.value); // comando que permite a inclusão da informação digitada //

    form.reset(); // função que vai limpar o formulário após envio //

};

// Criação das constantes em lista //

function addTask(description) {

    const taskContainer = document.createElement('div');

    const newTask = document.createElement('input');

    const taskLabel = document.createElement('label');

    const taskDescriptionNode = document.createTextNode(description);

    // comandos para criar a lista das tarefas digitadas com o checkbox //

    newTask.setAttribute('type', 'checkbox');

    newTask.setAttribute('name', description);

    newTask.setAttribute('id', description);

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

// Código para que o texto digitado seja inserido na página //

    taskLabel.setAttribute('for', description);

    taskLabel.appendChild(taskDescriptionNode);

    // Criação do container que vai armazenar a lista de tarefas //

    taskContainer.classList.add('task-item');

    taskContainer.appendChild(newTask);

    taskContainer.appendChild(taskLabel);

    // o comando abaixo é quem vai criar a lista das tarefas digitadas //

    taskList.appendChild(taskContainer);

}

Analisando o Style.css

/\* caminho para JS buscar novas fontes \*/

@import url('https://fonts.googleapis.com/css2?family=Open+Sans:wght@300;600&display=swap');

\* { /\* tipos de fontes \*/

    font-family: 'Open Sans', sans-serif;

}

/\* Dando padrão para o corpo da página \*/

body {

  display: flex;

  align-items: center;

  justify-content: center;

  flex-direction: column;

}

/\* Informando que tipo de texto será aceito e como será a caixa para inserção das tarefas \*/

input [type="text"] {

    padding: 5px;

    border-radius: 5px;

    border: 1px solid black;

}

/\* Estilizando o botão adicionar \*/

button{

  border: 1px solid rgb(105, 197, 255);

  border-radius: 5px;

  background-color: rgb(227, 248, 255);

  color: rgb(105, 197, 255);

  text-align: center;

  padding: 5px;

}

/\* Padrão para todos dados digitados  \*/

#tasks {

    display: flex;

    flex-direction: column; /\* se os dados ficarão em linha (row) ou colunas\*/

    margin-top: 20px; /\* distância da caixa de digitação \*/

  }

  /\* padrão de cada item digitado \*/

  .task-item {

    display: flex;

    align-items: center;

    margin-bottom: 10px;

  }

  /\* padrão da caixinha + a informação digitada \*/

  :checked + label {

    text-decoration: line-through; /\* linha no texto \*/

    color: grey;

  }

FRAMEWORKS E BIBLIOTECAS

Forma, Seta

Descrição gerada automaticamente

JS puro = Vanilla JS

Frameworks: são ferramentas desenvolvidas por terceiros para agilizar o desenvolvimento do código

Os principais são VueJS e Angular

React: é uma biblioteca utilizada como base de desenvolvimento de grandes projetos

jQuery está caindo em desuso

SINTAXE BÁSICA EM JAVASCRIPT

O que é o JS?

O JavaScript é uma linguagem de programação de alto nível (se assemelha mais à sintaxe humana), que integra o desenvolvimento de apps e páginas web.

É uma das linguagens de programação mais usadas no mundo do desenvolvimento.

Como surgiu?

Em 1995 o programador Brendan Eich que trabalhava na Netscape, implementou uma linguagem de programação chamada Mocha, depois o nome mudou para LiveScript, mas na época a linguagem Java era popular, então mudaram LiveScript para JavaScript como jogada de marketing.

A confusão entre JavaScript e Java é culpa dos marketeiros.

Para quê serve?

Para criar scripts dinâmicos que realizam a interação de apps ou páginas web.

Imagina que o HTML é a estrutura, o CSS é o design, o JavaScript é a parte interativa do desenvolvimento.

Onde Usar?

Podemos usar o JavaScript de várias formas, seu uso mais comum se dá no desenvolvimento de apps e páginas web, juntamente com as linguagens HTML e CSS.

Nesse curso usaremos para estudar e criar projetos.

CONFIGURANDO E DESENVOLVENDO

Ferramentas e extensões

Durante o curso usaremos o VSCode(IDE) e a extensão Live Server.

As instruções de instalação e configuração estarão disponíveis através de um pdf no repositório.

BOAS PRÁTICAS:

Externar os arquivos de script através do src



Ao segurar o ctrl em cima do nome do arquivo a ser externado, ele dá opção de criá-lo automaticamente.

Carregar o script ao final, antes do fechamento do </body>

Texto

Descrição gerada automaticamente

COMO FUNCIONA A TIPAGEM NO JS

A tipagem funciona como uma regra de uso de dados. Quanto mais forte for a tipagem, mais obrigatório é a declaração do tipo de dado. A tipagem em JavaScript é fraca, a declaração dos dados acontece de modo dinâmico.

Ex.: Ao criarmos uma variável com valor entre aspas (“valor”) o JavaScript já converte o dado para o tipo String.

Ex.: var numero = 1;

// o JavaScript já converte o valor 1 para o tipo Number.

DECLARAÇÃO DE VARIÁVEIS

Tipos primitivos

As variáveis em JavaScript podem guardar tipos de dados que chamamos de tipos primitivos.

Variáveis podem guardar valores dos tipos:

Boolean: verdadeiro ou falso;

Null: nulos;

undefined;

Number;

String;

Array;

Object;

Function.

Se eu mudar a instrução para typeof, eu consigo ver que tipo de variável trata-se através da janela de inspeção do navegador:

Texto

Descrição gerada automaticamente Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

O que são variáveis?

São dados que variam.

Imagina uma caixa, tu pode colocar quase qualquer coisa dentro dela, pode ser uma lista de nomes, cartões com números, objetos no geral...

Variáveis funcionam dessa forma,, como uma caixa que guarda valores.

Existem 3 modos de declarar as variáveis em JavaScript:

var – genérico, escopo global e local, pode ter seu valor alterado, se não tiver um valor inicial será tratada como null;

let – escopo local de bloco, pode ter seu valor alterado, se não tiver um valor inicial será tratada como null;

const – escopo local de bloco, somente leitura, o valor inicial é obrigatório e não pode ser alterado.

ESCOPO

O escopo em JavaScript define a limitação e visibilidade de um bloco de código.

Escopo global – quando a variável é declarada fora de qualquer bloco, sua visibilidade fica disponível em todo o código.

Escopo local – quando a variável é declarada dentro de um bloco, sua visibilidade pode ficar disponível ou não. Por convenção utiliza-se a variável let dentro da função.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

No exemplo daria erro, pois o console.log ficou fora. Ele deve ser colocado junto com o let na function.

Regras de Uso de Variáveis

• Iniciar com letras, underscore \_ ou cifrão $; Não iniciar com número.

Ex.: var 1nome ❌

var nome || var \_nome ✔

• Não usar espaços (use o camelCase (quando a variável tem mais de um nome, iniciamos com minúsculo no primeiro e Começamos com letra maiúscula a partir do segundo nome) ou \_);

Ex.: var nome completo ❌

var nomeCompleto || var nome\_completo✔

• Não usar palavras reservadas;

Ex.: var function ❌

• Declarar variáveis no topo do bloco de código.

DIFERENÇAS ENTRE ATRIBUIÇÃO, COMPARAÇÃO E COMPARAÇÃO IDÊNTICA

Atribuição

O sinal de igualdade “=“ em JavaScript, significa atribuição.

Ex.:

como declarar: var nome = “meu nome”;

como ler: variável nome recebe o valor meunome;

Comparação

Para fazermos uma comparação de valores em JavaScript usamos “==“.

Sempre retornará um valor bolleano

Ex.:

como declarar: "0" == 0; ;

como ler: “0” tem o valor igual a 0?

// nesse caso retorna true

Comparação idêntica

Para fazermos uma comparação de valores e tipos em JavaScript usamos “===“.

Ex.:

como declarar: "0" === 0; ;

como ler: “0” tem o valor e o tipo idêntico a 0?

// nesse caso retorna false pois o primeiro entre aspas é uma string

OPERADORES ARITMÉTICOS, RELACIONAIS E LÓGICOS

Operadores aritméticos

São tipos de operadores matemáticos com valor numérico:

• + adição;

• - subtração;

• \* multiplicação;

• / divisão real;

• % divisão inteira; retorna o resto da divisão (5/2 vai retornar 1)

• \*\* potenciação;

Operadores relacionais

São tipos de operadores que consultam a relação entre valores:

• > maior que;

• < menor que;

• >= maior ou igual a;

• <= menor ou igual a;

Operadores lógicos

São tipos de operadores que consultam valores lógicos:

• && - “e” – considera que todos os valores sejam true;

• || - “ou” – considera que qualquer valor seja true;

• ! - “não” – inverte o valor de true para false ou vice-versa;

Para saber mais

Tipagem –

https://danvitoriano.medium.com/tipagem-dinâmica-no-javascript-e3551a445b38

Variáveis –

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/

Grammar\_and\_types

Escopo –

https://imasters.com.br/desenvolvimento/escopos-em-javascript

Operadores –

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/>

Vetores ou arrays

Arrays são um tipo de lista, ou matriz de variáveis, onde cada variável possui um índice. Os valores podem ser de vários tipos.

Imagina que um array é uma caixa com várias outras caixas dentro e cada uma contendo algum valor;

Ex.: let array = [‘string’, 1, true...]

Array

O array deve ser declarado entre colchetes “[]”, e podem guardar qualquer valor dentro de seus índices: inclusive outros arrays.

Ex.: let array = ['string', 1, true, false, [‘array1], [‘array2’]...]

Obs.: O índice só é acessado por um número inteiro, onde 0 é o primeiro índice. Cada índice é separado por vírgula.

Manipulando Arrays

Ao ser declarado, o Array traz consigo uma série de métodos para manipulá-lo.

• forEach() – itera um array;

• push() – add item no final do array;

• pop() – remove item no final do array;

• shift() – remove item no início do array;

• unshift() – add item no início do array;

• indexOf() – retorna o índice de um valor;

• splice() – remove ou substitui um item pelo índice;

• slice() – retorna uma parte de um array existente;

Objetos

Dados que possuem propriedades e valores que definem suas características. Deve ser declarado entre chaves “{}”.

Ex.: imagine uma xícara azul. Ela tem cor, pode ter vários tamanhos e funções(quero cafééé). Pode ser declarada assim:

var xicara = {

cor: ‘azul’,

tamanho: ‘p’,

funcao: tomarCafe()

}

Manipulando objetos

As propriedades de objetos podem ser atribuídas à variáveis, facilitando a manipulação do objeto. Chamamos isso de desestruturação.

Ex.: var xicara = {cor: ‘azul’, tamanho: ‘p’, funcao: tomarCafe()}

var cor = xicara.cor;

var tamanho = xicara.tamanho;

var funcao = tomarCafe();

Outra forma de fazer a desestruturação é utilizando chaves ao declarar a variável:

Ex.: var xicara = {cor: ‘azul’, tamanho: ‘p’, funcao: tomarCafe()}

var { cor, tamanho, funcao } = xicara;

Estruturas condicionais

São instruções para realizar determinadas tarefas a partir de uma condição, seja de decisão ou repetição;

Ex.: Um jogo precisa mudar o placar toda vez que um jogador marca pontuação;

Estruturas de decisão

if

Podemos usar as palavras reservadas “if” para estabelecer uma condição:

Ex.: var jogador1 = 0;

var jogador2 = 0;

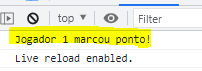
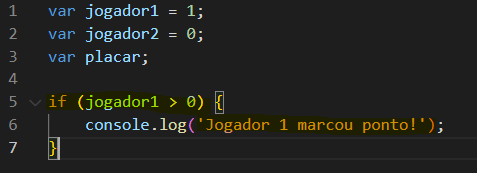
if (jogador1 > 0) {

console.log(‘jogador1 marcou ponto’);

}

// como ler: se o jogador1 tiver valor maior que 0

ele marcou ponto.



Else = se não

No caso de a condição não ser atendida podemos usar o “else” :

Ex.: var jogador1 = 1;

var jogador2 = 0;

if (jogador1 > 0) {

console.log(‘jogador1 marcou ponto’);

} else {

console.log(‘ninguem marcou ponto’);

}

else if

Caso haja mais de uma condição usamos o “else if“.

Ex.: var jogador1 = 1;

var jogador2 = 0;

if (jogador1 > 0) {

console.log(‘jogador1 marcou ponto’);

} else if (jogador2 > 0) {

console.log(‘jogador2 marcou ponto’);

} else {

console.log(‘ninguém marcou ponto);

}

ninho de if

Podemos também usar o “if” dentro de um outro “if”, chamamos isso de aninhamento de if’s ou ninho de if’s.

Ex.: if (jogador1 = -1) {

if (jogador1 > 0) {

console.log(‘jogador 1 marcou ponto’);

} else {

console.log(‘ninguém marcou ponto’);

}

} else {

console.log(‘jogador inválido’);

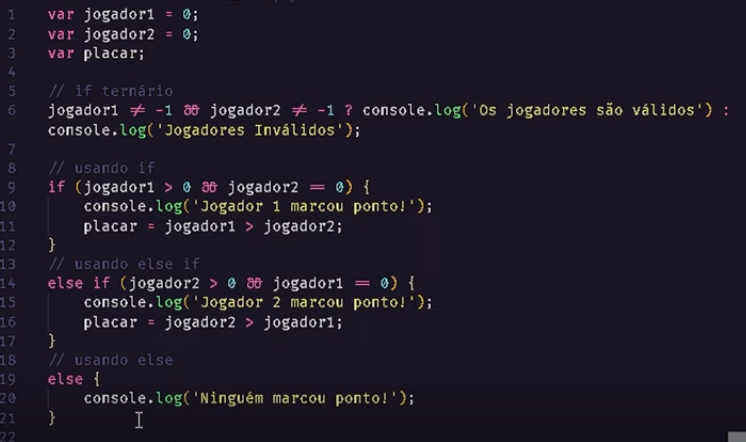
}

If ternário

Podemos também fazer uma verificação em uma única linha usando o “if” ternário:

Ex.: [condição] ? [instrução1] : [instrução2];

jogador1 > 0 ? console.log(‘marcou ponto’) : console.log(‘não marcou ponto”);



// lembre de usar a interrogação “?” e dois pontos “:”

Usando switch/case

O “switch/case” funciona como uma estrutura condicional também;

Ex.: switch (${expressao}) {

case 1:

${instrucao};

break;

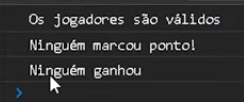
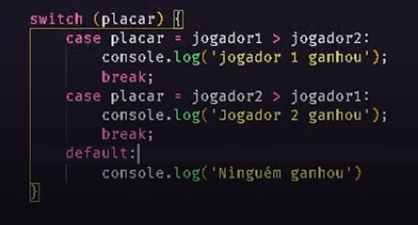
case 2:

${instrução};

break;

}

Default dá um valor padrão caso nenhuma das instruções sejam atendidas:



No exemplo acima os jogadores ficaram com valor zero.

Estruturas de repetição

Laços de repetição

São estruturas condicionais que repetem uma instrução até atingir determinada condição:

• For;

• For/in;

• For/of;

• While;

• Do/while;

For

Funciona como uma repetição de instrução (repete a execução) até que a condição seja falsa:

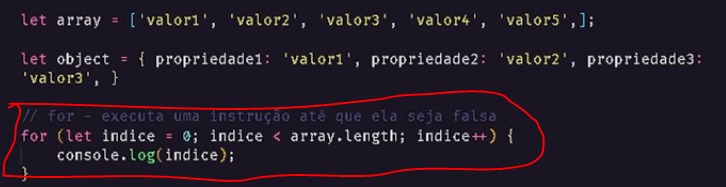
for ([expressaoInicial]; [condicao]; [incremento]) { declaracao }

Ex.: var array = [‘valor1’, ‘valor2’, ‘valor3’, ‘valor4’]

for (let i = 0; i < array.length; i++) {

console.log(i);

}



For/in: imprime o índice, mas não como número como no caso do for, mas como string.

Funciona como uma repetição a partir de uma propriedade:

for ([indice] in [objeto ou array]) { declaracao }

Ex.: var array = [‘valor1’, ‘valor2’, ‘valor3’, ‘valor4’]

for (i in array) {

console.log(i);

}

For/of: imprime o conteúdo do índice e não o índice.

Funciona como uma repetição a partir de um valor:

for ([indice] of [array]) { declaracao }

Ex.: var array = [‘valor1’, ‘valor2’, ‘valor3’, ‘valor4’]

for (i of array) {

console.log(i);

}

Ele vair imprimir, valor1, valor2, valor3, valor4

For/of

O For/of não funciona com objetos pois as propriedades variam, diferentes do índice em um array que sempre serão números inteiros.

Mas e se eu quiser pegar o valor mesmo assim?

Ex.: for (i of object.propriedade) {

console.log(i);

}

// porém cada caractere dentro do valor será Impresso em linhas separadas.

While

Executa uma instrução “enquanto” determinada condição for verdadeira, a verificação é feita antes da execução;

Ex.: var a = 0;

while (a < 10) {

a++;

console.log(a);

}

// como ler: enquanto a variável a for menor que 10 ela vai receber mais um e imprimir no console.

Do/while

Executa uma instrução “até que” determinada condição seja falsa, a verificação é feita depois da execução;

Ex.: var a = 0;

do {

a++;

console.log(a);

} while (a < 10)

// como ler: a variável a vai receber mais um e imprimir no console até que seu valor chegue a 10

Para saber mais

Estruturas condicionais –

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/conditionals>

Laços de repetição –

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Loops_and_iteration>

FUNÇÕES E SUAS PARTICULARIDADES

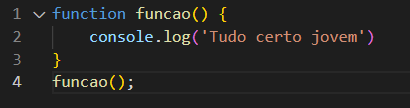
Funções : São blocos de comandos e instruções para a execução de determinadas tarefas:

Ex.: function nomeDaFuncao() {

${instrucao};

}

nomeDaFuncao();



Como declarar?

Geralmente se utiliza a palavra reservada “function” seguida de parênteses “()” e chaves “{}”:

Ex.: function funcao() {

console.log(“mensagem”);

}

funcao();

() – indica que é um objeto do tipo function;

{} – indica que é um bloco de instrução;

Funções com parâmetros

As funções podem receber em sua declaração, parâmetros, que servem como variáveis de estado, onde sua atribuição pode ser feita durante a chamada da função:

Ex.: function nomeDaFuncao(parametro) {

${instrucao};

}

nomeDaFuncao(valorDoParametro);

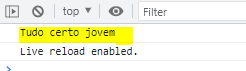
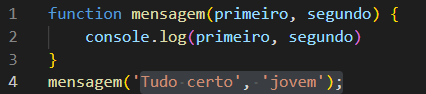
Funções com parâmetros

Ex.: function mensagem(primeiro, segundo) {

console.log(primeiro, segundo);

}

nomeDaFuncao(“Tudo certo”, “jovem!”);



Para saber mais

Funções –

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Functions>

Funções declarativas

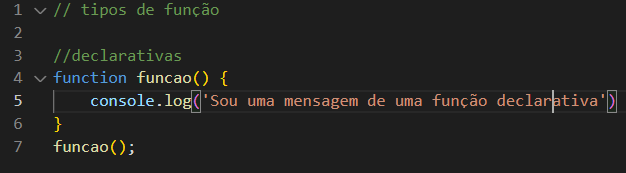
São funções que possuem o uso mais comum, deve ser declarada usando a palavra reservada “function” seguida do nome da função, parênteses “()” e chaves “{}”:

Ex.: function nomeDaFuncao() {

${instrucao};

}

// o nome da função é obrigatório;



Expressões de Funções

São funções atribuídas à expressões. A nomeação das funções por expressão é opcional:

Ex. 1: var funcao = function nomeDaFuncao() {

${instrucao};

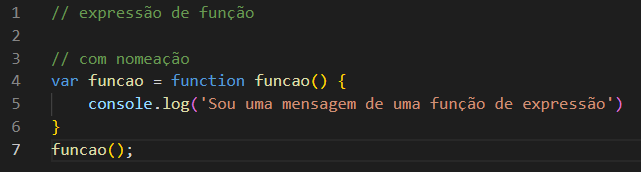
}

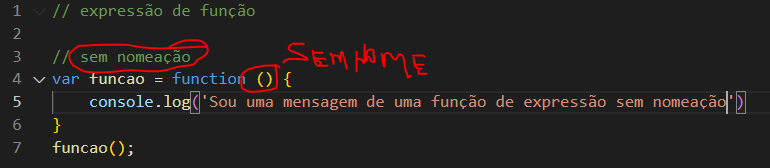
Ex. 2: var funcao = function() {

${instrucao};

};

***Neste exemplo acima, não atribuímos o nome da função, pois o JS já vai atribuir o nome da variável à função***





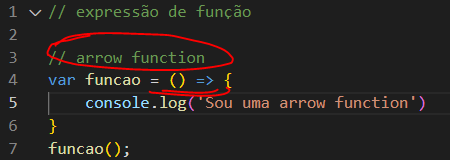
Arrow Function

São funções de expressão de sintaxe curta. Arrow functions sempre serão anônimas, e portanto não podem ser nomeadas. Deve ser declarada com parênteses "()", seguido de "=>" e depois chaves "{}"

Ex. 1: var funcao = () => {

${instrucao};

}



Criando uma calculadora

Para este exercício vamos utilizar alguns métodos nativos do JavaScript:

• Number() – para converter valores em números;

• Prompt() – para registrar entradas de usuário;

• Alert() – para mostrar mensagem ao usuário;

• Template Strings – para usar strings junto com expressões;

Para quebrar linha utiliza-se \n

Para inserir variáveis dentro do alert utiliza-se sinal de crase/ para textos, apóstrofe.

Para saber mais

Funções –

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Functions>

Number –

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

Window.prompt –

<https://developer.mozilla.org/pt-BR/docs/Web/API/Window/prompt>

Para saber mais

Window.alert –

<https://developer.mozilla.org/pt-BR/docs/Web/API/Window/alert>

Template String -

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Template_literals>

SINTAXE E OPERADORES

Operadores: sinais que utilizamos no JS para manipular certos valores ou validar determinada condição. São apresentados por vários sinais:



São os mais utilizados os operadores de:

Aritmética

Atribuição:

= atribui o valor ex: x = y

+= adiciono o valor da variável à outra: x +=y

\*= a variável vale o valor dela multiplicado pela outra: x \*= y

/= a variável tem o valor dela dividido pela outra: x /=y

%= a variável terá o restante dela dividida pela outra: x %=y

Tipos de operadores:

+ Adição

- Subtração

\* Multiplicação

\*\* Exponencial

/ Divisão Real

% Divisão Inteiro (Módulo)

++ Incrementar = ele vai adicionar 1 à variável. Se tenho var = 0, ela valerá 1

-- Decrementar = ele vai subtrair 1 à variável. Se tenho var = 0, ela valerá -1

OPERADORES DE COMPARAÇÃO

== Igual a (Este valor é igual a este, independente do tipo?)

=== Mesmo valor e mesmo tipo (Este valor é igual e do mesmo tipo?)\* + utilizado

!= Diferente (Este valor é diferente do outro, independente do tipo?)

!== Valor e tipo diferentes (Este valor é igual e do mesmo tipo?)

> Maior que

< Menor que

>= Maior ou igual

<= Menor ou igual

OPERADORES DE LÓGICA

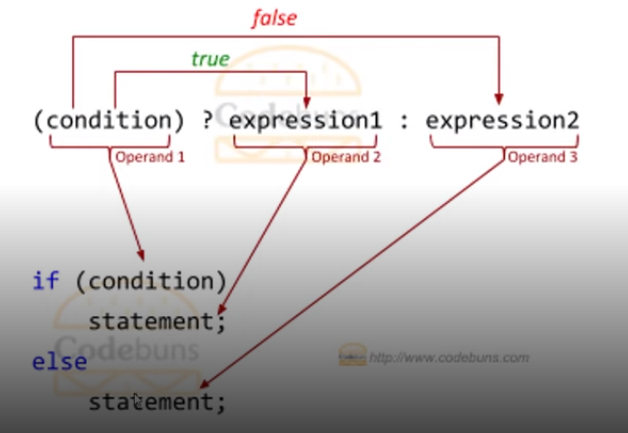
&& “e” lógico (Se esta afirmação e a outra afirmação forem verdadeiras)

|| “ou” lógico (Se esta ou aquela afirmação for verdadeira)

! “não” lógico (Se queremos acessar o dado oposto ao informado. Se mando um true, espero um false e vice-versa)

OPERADOR CONDICIONAL

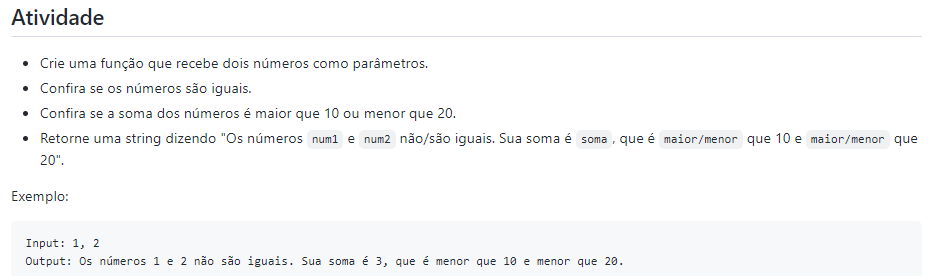
If ternário



Se a condição for verdade, executo a primeira expressão; senão, a segunda

Nem sempre é necessário fazer a operação if + else, podendo ser if + return

Exercício Prático:



function comparaNumeros(num1, num2) {

    // vamos criar as frases:

    const primeiraFrase = criaPrimeiraFrase(num1, num2);

    const segundaFrase = criaSegundaFrase(num1, num2);

    // vamos concatenar a frase 1 com a frase 2:

    return `${primeiraFrase} ${segundaFrase}`;

}

// aí vamos a função "Primeira Frase":

function criaPrimeiraFrase(num1, num2) {

    // aí vamos colocar a análise utilizando uma string auxiliar (let) permitindo mudar se os valores delas se forem ou não iguais:

    let saoIguais = ''; // se elas forem iguais o valor será 0

    if(num1 !== num2) {

        saoIguais = 'não'; // se os valores não forem iguais vai retornar o não

    }

    // Então aqui no return, sendo os números iguais, o texto mantem-se; se diferentes, será acrescida a variável não estabelecida no {somosIguais}

    return `Os números ${num1} e ${num2} ${saoIguais} são iguais.`

};

// vamos declarar a função da segunda frase:

function criaSegundaFrase(num1, num2) {

    const soma = num1 + num2;

    // vamos criar as strings para cada comparativo:

    let resultado10 = 'menor';

    let resultado20 = 'menor';

    const compara10 = soma > 10; // essa variável é bolleana retornando true ou false

    const compara20 = soma > 20; // essa variável é bolleana retornando true ou false

    if(compara10) {

        resultado10 = 'maior'; // se resultado for maior que 10 ele muda para maior

   }

   if(compara20) {

    resultado20 = 'maior'; // se resultado for maior que 20 ele muda para maior

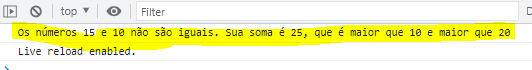
}

// retornando a segunda frase usando crase para inserir as variáveis:

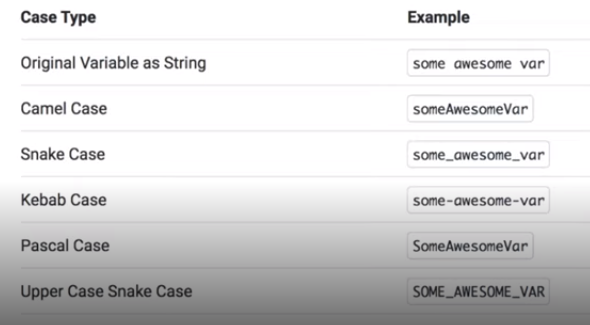
return `Sua soma é ${soma}, que é ${resultado10} que 10 e ${resultado20} que 20`;

}

console.log(comparaNumeros(15, 10));



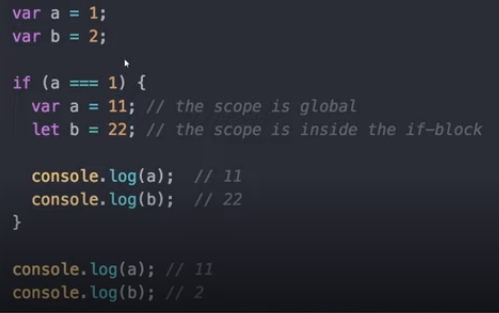
ATRIBUIÇÃO DE VALORES



Camel Case: primeira palavra da variável inicia minúscula e demais maiúsculas: primSegTerQua, por exemplo

Upper Case Snake Case: todas as palavras em maiúsculo separados por underline

Let: utilizado para definir variável dentro de um bloco. É o ideal atualmente.



Neste exemplo, quando se chama o console log dentro do bloco { }, ele vai atribuir à variável b o valor inicial estabelecido por let, de 22; mas ao executar fora do bloco, trará o valor de 2 estabelecido pelo var.

Var (global) e Let (escopo de bloco) têm escopos diferentes.

Hoisting: atribuir valor a uma variável antes de declará-la. Funciona em var e function. O JS script levará a declaração para cima, mesmo sendo declarada no final do código.

Redeclaração: só permitido no var

Reatribuição: o nome da variável, sem a palavra reservada na frente, permitido para o let e var. Exemplo: var lastName = “de Paula”, para retribuir digito apenas lastName = “da Silva”

Todas utilizando camelCase

Exercício:

// vou declarar as variáveis e atribuir valor

var firstName = "João";

var lastName = "Souza";

// agora abrirei um novo escopo:

/\* Se a variável firstName for do mesmo tipo e valor

eu a mudarei, mas não o lastName, pois está

como let e não var o novo valor atribuído: \*/

if (firstName === "João") {

    var firstName = "Pedro";

    let lastName = "Silva";

// a variável last name ficou com cor diferente, pois não teve utilização

}

console.log(firstName, lastName)



CONSTANTES

São declaradas em SNAKE\_UPPER\_CASE

Tem escopo de bloco

Não fazem hoisting (atribuição de valor antes da declaração)

Não pode reatribuir valor a uma constante.

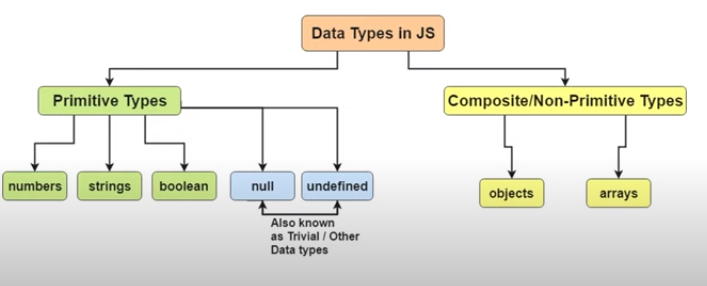
Não pode ser redeclarada.



ESTRUTURAS DE DADOS

Existentes em todas as linguagens de programação, mas no JS são de tipagem dinâmica, ou seja, antes de declarar o valor não é necessário especificar o tipo dele.

Typeof no console mostra o tipo de dado



Tipos primitivos não têm dados guardados neles e são escritos em letras minúsculas.

Não primitivos são objetos e arrays

STRINGS: são comumente utilizadas para textos e são declarados entre aspas (só texto) ou crases (quando compostos por textos e variáveis).

Var + concat (vai concatenar variáveis):

let nome = “João”

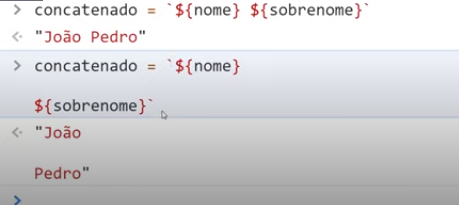
let sobrenome = “Pedro”

nome.concat(sobrenome)

Resultado = JoãoPedro

Para incluir o espaço, executo concatenado = nome + “ “ + sobrenome

Utilizando crase fica mais fácil a manipulação da variável:



String iniciado com maiúscula é igual objeto

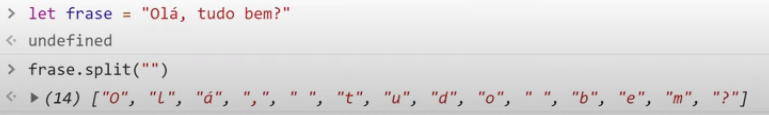
\_proto\_: prototype mostra todos os métodos possíveis de utilização na string

lenght mostra tamanho do item (quantidade de caracteres)

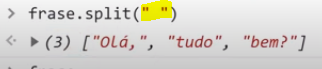
\n quebra de linha

Para printar “ dentro de uma string utiliza \ antes dela

Split separa os argumentos da string



Split com espaço retorna as palavras

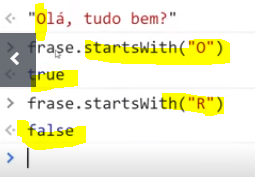


Includes procura uma palavra na string;



Retorna um resultado booleano

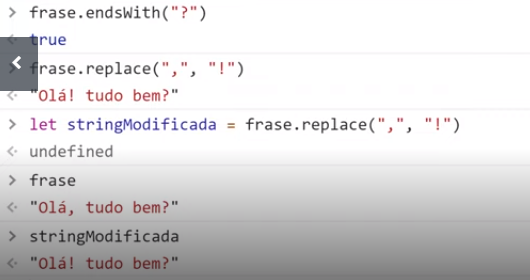
startsWith responde se a começa com a letra informada:



O mesmo funcionamento tem endsWith para verificar com o que termina

replace: modifica algum item na impressão da string, mas não muda a string.

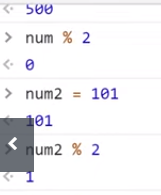
Para modificar a string, preciso atribuir um novo endereço de memória, criando uma nova:



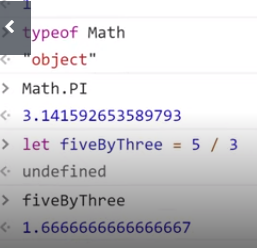
NUMBERS

Inteiros ou decimais

Para identificar se o número é par aplique num % (volta o resto) 2. Se o resultado for zero é que o número é par, se resultar 1 é impar



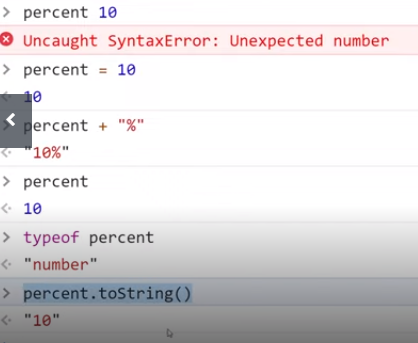
Math (objeto) mostra várias funções matemáticas



Math.floor arredonda para baixo e math.cell para cima



Não posso declarar uma variável percentual com %. Tem que ser através do percent =

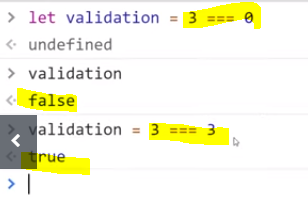


toString converte number em string

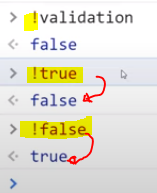
BOOLEANS

Ou retorna true ou false

Validation para validar comparação:



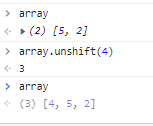
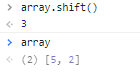
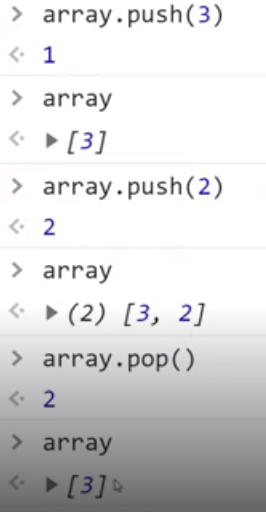
! retorna valor ao contrário



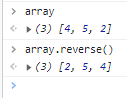
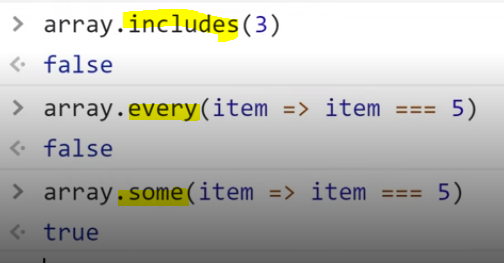
ARRAYS (VETORES)

São listas iteráveis; permite fazer operações em cada um dos elementos

Array.push() envio elemento para o array e array.pop() exclui o último elemento, array.shift() exclui o primeiro elemento. Array.unshift adiciona no começo da lista.

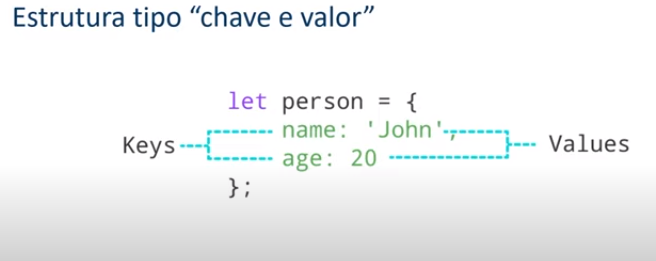


Array.includes() verifica se o item existe no array. Array.every busca saber se todos itens são iguais e array.some busca saber se um item específico existe no array. Array.reverse inverte os elementos.

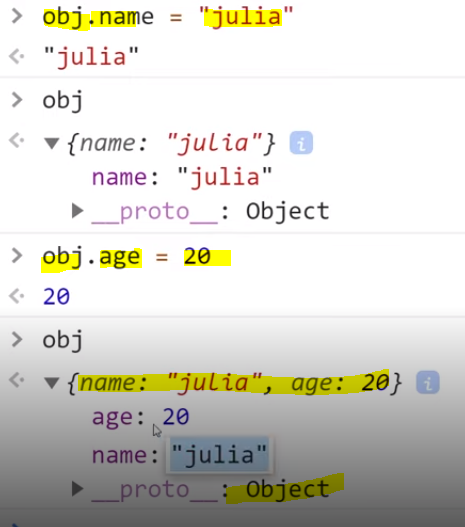


OBJETOS

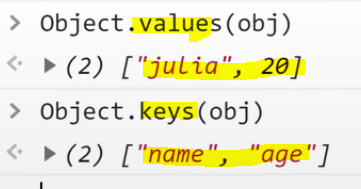
Variável + nome = {name + valor}



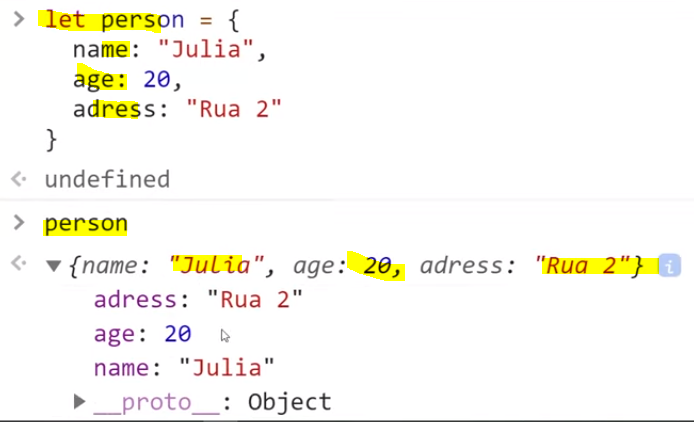
Obj+chave = valor da chave



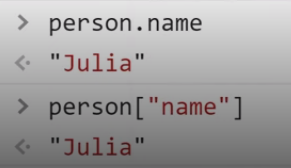
Object.values() mostra os valores do objeto. Object.keys() mostra as chaves do objeto



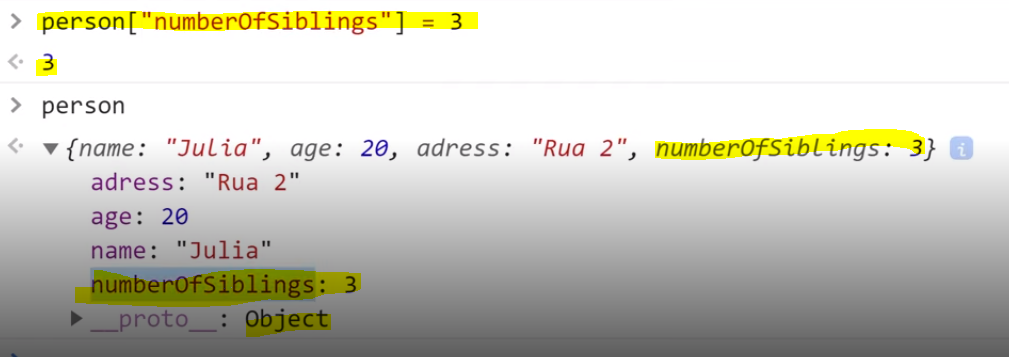
Posso declarar as chaves todas de uma vez, dentro de uma variável:



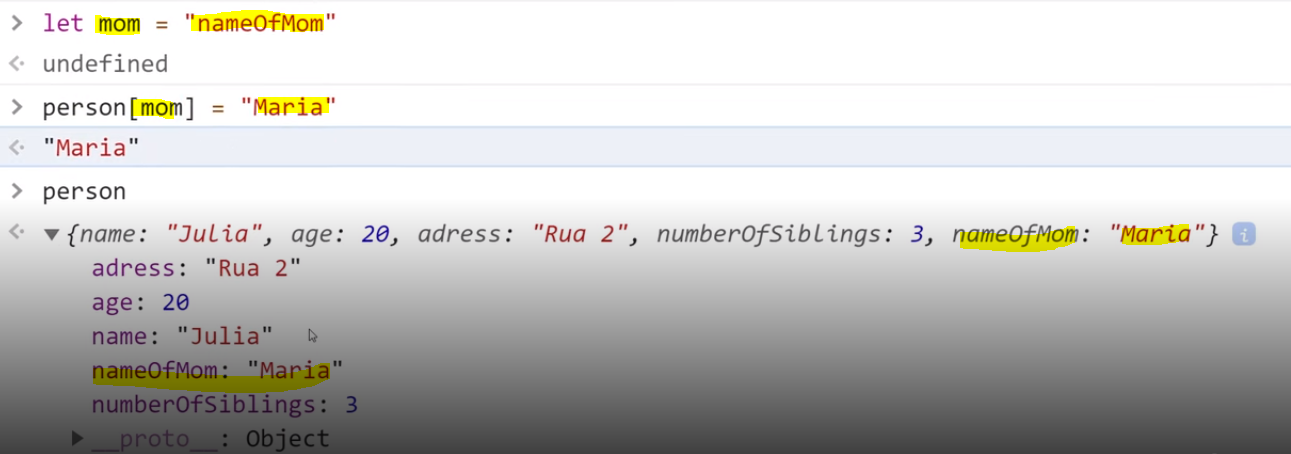
Para retornar informações:



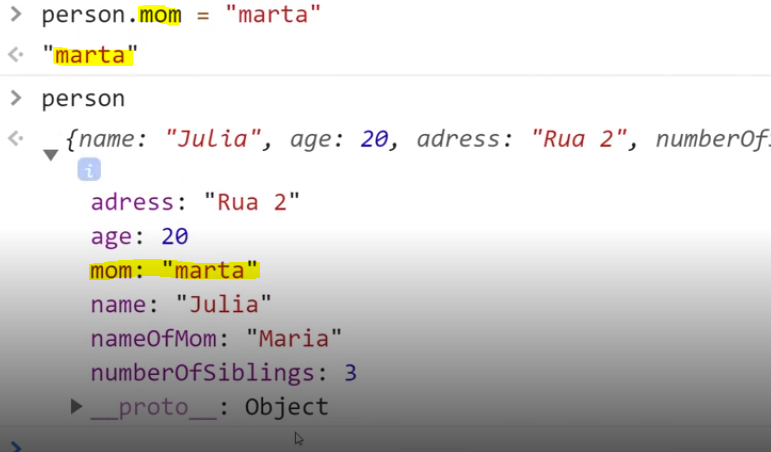
Posso ainda adicionar informação dentro do objeto. Por exemplo, vou adicionar o número de irmãos da person:



Para colocar uma chave que tem nome de outra variável dentro do meu código, eu pego a variável e entre colchetes coloco o nome da nova variável e dá valor a ela



Se eu não colocar dentro do colchete, a informação não vai para dentro da variável, mas ele acrescenta no objeto



EMPTY: vazio, declarou a variável mas, se era número colocou zero, se era string deixou “ “, se era array só incializou com os [ ], se é um objeto só inicializou com { }. Ou seja, não há nenhum valor dentro.

UNDEFINED:

NULL: quando propositalmente você quer que aquele valor não exista, não foi inicializado.

Na ilustração abaixo talvez fique mais claro:

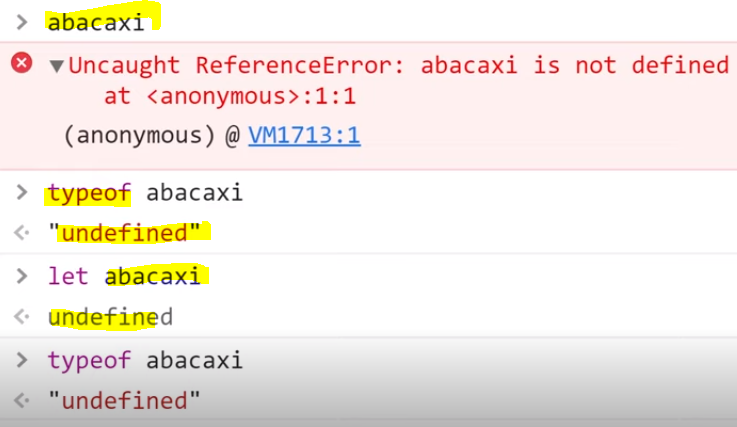
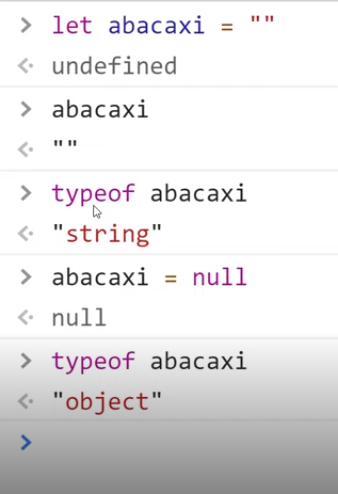


No 0 tem a plaquinha declarando cookies, a gente inicializou a caixinha, mas não há nada dentro.

No null tem a plaquinha declarando cookies, mas não inicializou a caixinha de propósito. Quando coloco valor null ele vira objeto

No undefined não foi nem declarado. Ele é indefinido. Não existe.

Exemplo na prática:

falsyValues que são tratados como falso mas não são iguais ao false do Booleano.