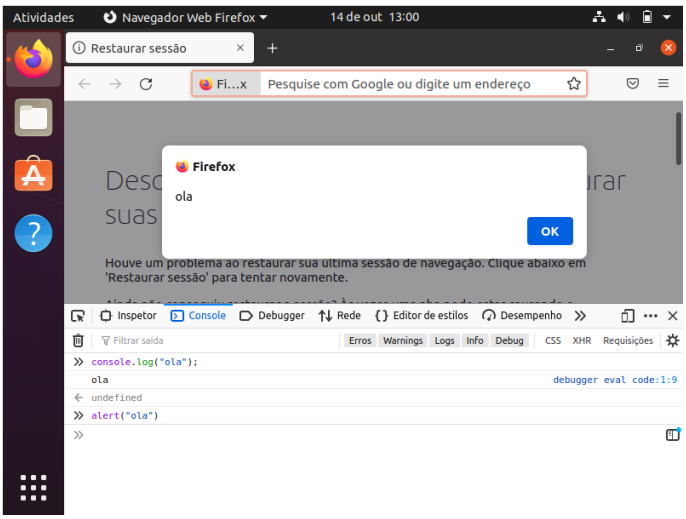
INTRODUÇÃO AO JAVASCRIPT

Iuri de Carvalho Salgado

JavaScript é uma linguagem de programação de tipagem dinâmica e fraca, e é multiparadigma. Um código em JavasScript pode ser executado de diferentes formas e com retornos também variados.

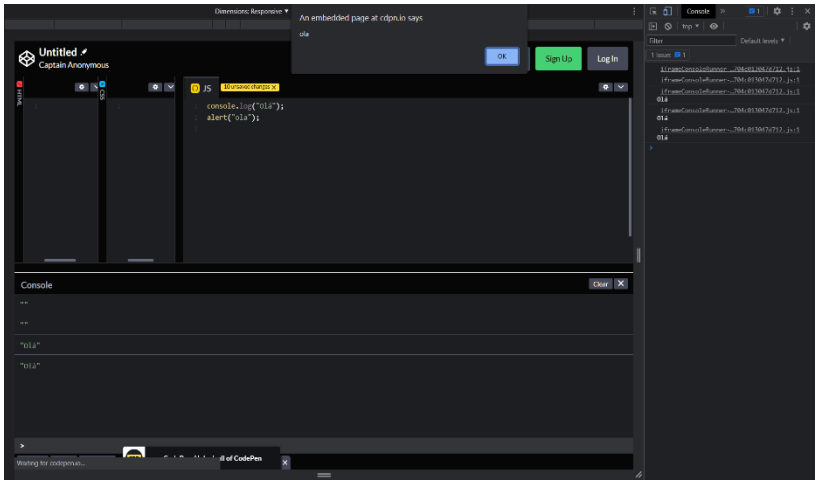
1. Maneiras de excutar o JavaScript:

No navegador.



Utilizando o navegador Firefox como exemplo, nas três barrinhas no canto direito do navegador, vá em mais ferramentas e ferramentas de desenvolvedor. Também é possível ter o mesmo resultado com os atalhos F12 ou ctrl+shif+i. Na aba console você pode digitar as instruções e com um enter obter o resultado. Isso também pode ser feito no Google Chrome.

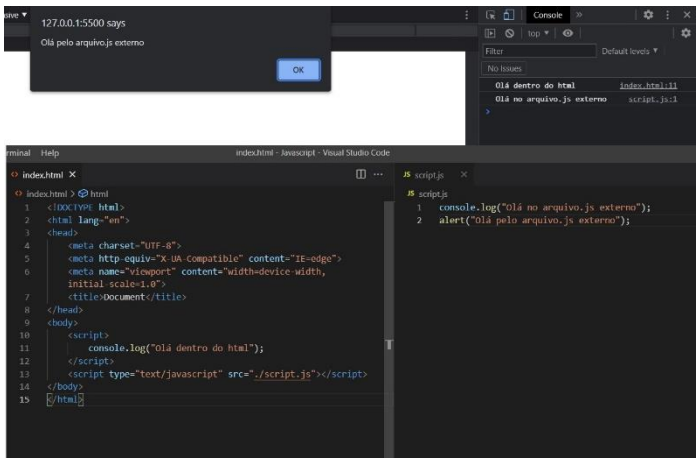
No codepen.io/pen.



Neste endereço existe uma interface de programação parecida com uma IDE, logicamente respeitando as proporções. É possível neste link criar códigos html, css e javascript e executa-los também em tempo real. Vale ressaltar que alguns detalhes de exibição são um pouco diferentes e que o console dentro do navegador também funcionará.

Usando o VS code.

O JavaScript pode ser escrito direto no corpo do HTML, dentro das tags <script></script> ou em um arquivo.js externo, sendo referenciado com a tag <script src=”./arquivo.js”> </script>. Normalmente o arquivo.js é chamado no final do corpo do html, melhorando a performance de exibição da página e suas animações.



2. Elaborando um código.

Comentários:

Os comentários de linha são identificados pela dupla de barras, //, já os comentários de bloco ficam entre a sequência: barra e asterisco, asterisco e barra, /**/.

Operadores:

Podem ser de atribuição, de comparação, aritméticos ou lógicos.

Operadores de atribuição		
Operador	Sintaxe	Equivalente
=	a = b	Igual
!=	a != b	Diferente
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	Resto da divisão

Operadores Aritméticos	
Operador	Operação
+	Soma
-	Subtração
/	Divisão
*	Multiplicação
**	Exponenciação
%	Módulo da divisão
++	Incremento
--	Decremento

Operadores Comparadores	
Operador	Função
==	Valor igual
===	Valor e tipo igual
!=	Valor diferente
!==	Val. e tipo diferente
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual

Operadores lógicos		
Operador	Lógica	Como Funciona
&&	AND	O retorno é verdadeiro apenas quando todas as proposições são verdadeiras
||	OR	O retorno é verdadeiro quando ao menos uma proposição for verdadeira
!	NOT	A proposição tem seu valor lógico invertido ou negado
?	IF ternário	É uma condicional de uma linha. condição ? Ação para verdadeira : ação para falsa;

3. Variáveis e estrutura de dados

Regras e boas práticas para nomear as variáveis:

Primeiro, seus identificadores devem começar com letras, underscore (_) ou cifrão ($), não são permitidos apenas números para nomear uma variável e nem palavras reservadas, espaços em branco ou caracteres especiais.

O JavaScript é *case sensitive*, e são boas práticas na identificação das variáveis o uso do SNAKE_UPPER_CASE para declarar constantes, e o camelCase para as demais variáveis. Além disso é importante que o nome faça sentido dentro do contexto do código, ou seja, variáveis de nome, var1, a, num1, etc. não são intuitivas.

SNAKE_UPPER_CASE
camelCase
kebab-case
PascalCase
snake_case

Declação de variáveis:

Elas podem ser declaradas de três formas (*const, let, var*), e diferença entre cada uma é em relação ao seu escopo e a forma como elas podem ser manipuladas.

	var	let	const
Escopo	Global e local	Bloco	Bloco
Redeclarar	Sim	Não	Não
Reatribuir	Sim	Sim	Não
Hoisting	Sim	Não	Não

Hoisting é um conceito de atribuição de valor a uma variável antes mesmo de declara-la e funciona também para funções. A Redeclação é usar o mesmo nome para declarar uma variável em blocos diferentes e a Reatribuição é a alteração do valor atribuido a uma variável

Estrutura de dados:

No JavaScript não é necessário indicar o tipo de uma variável logo na sua atribuição, como ocorre por exemplo em C, e o tipo pode ser modificado posteriormente no código. Por isso a linguagem é chamada de dinâmica e fracamente tipada. Existem então, dois grandes grupos de tipos de variáveis, os tipos primitivos/básicos e não primitivos/compostos/complexos.

Tipos Primitivos	Não Primitivos
number	objects
string	
boolean	
Outros tipos	arrays
null	
undefined	

O typeof exibe o tipo de uma string. Uma funcionalidade nova na manipulação de strings é o template-string, que permite com o uso de crases a interpolação de strings. Também é possível inserir quebras de linha e espaços com uma sintaxe mais simples e menos verbosa. Ex: *nomeCompleto = `${nome}  ${sobrenome}`;*

Resultado: Nome Sobrenome

Arrays:

É uma variável que contém uma coleção de variáveis, pode ser considerado tambpem um objeto. Abaixo, como declarar uma array e alguns métodos básicos.

let array = [ ]; ou *ler array = new Array();*

array.push(1); //insere um item no final do array

array.unshift(2); //insere no inicio do array

array.pop; //remove o ultimo item do array

array.shift; //remove o primeiro elemento

Objects:

É uma estrutura do tipo chave-valor, ou dicionário de dados. Com as chaves do mesmo tipo e valores de tipos variados.

let object = { }; //cria um objeto

object.name = "Fulano"; //atribui uma chave name com valor Fulano

object.age = 20; //atribui uma chave age com valor 20

Object.values(object); //exibe todos os valores do objeto

Object.keys(object); //exibe todas as chaves do objeto

Empty: É quando algo e declarado sem nenhum valor.

Null: É quando algo é decladro propositalmente como nulo.

Undefined: É quando nada foi declarado.

4. Funções

```
function nome(parametros){
    return;
}
```

Uma função é declarada com a palavra reservada function, precedida de um nome e entre parenteses os parametros que ela recebe. O return para a função e ele pode retornar ou não um valor

Função Anônima:

```
const soma = function (a, b){
    return a + b;
}
```

Nesse caso é possível atribuir o retorno de uma função a uma variável, sem a necessidade da função ter um nome, por isso é chamada de anônima

Função autoinvocavel (IIFE – immediately invoked function expression):

```
(
    function(){
        let name = “Fulano”;
    }
)();
```

Assim que o código for executado a função ja vai estar sendo executada.

Callbacks:

É uma função passada como parametro para outra função.

```
const calc = function(){};
const soma = function(){};
const result = calc(soma, 1, 1); //esse é um callback
```

Spread:

É uma forma de lidar separadamente com elementos de um array. O que era parte de um array e torna um elemento independente.

```
function sum (x, y, z){
    return x + y + z;
}

const numbers = [ 1, 2, 3 ];

console.log(sum(...numbers)); //... é o spread operator
```

Rest:

Combina os elementos em um array. O que era um elemento independente se torna parte de um array.

```
function confereTamanho(...args){
    console.log(args.length);
}
```

Object Destructuring:

Entre chaves podemos filtrar apenas os dados que nos interessam em um objeto.

```
const user ={ id = 10, name = "fulano" };

function userId({id}) { return id; };

userId(user); //vai retornar o id 10 do objeto user
```

5. Laços de repetição

São os básicos como if/else, while, do while e for, comum na maioria das linguagens de programação.

IF/ELSE

```
if ( condição ){
    resultado;
} else if ( condição 2 ){
    resultado2;
} else {
    resultado3;
}
```

SWITCH

Funciona como uma comparação ===, de tipo e valor

```
switch ( op ) {
    case 1:
        return op + 1;
    case 2:
        return op + 2;
    default:
        return 0;
}
```

FOR

É um loop dentro e elementos iteraveis (arrays, strings)

```
for (let i = 0; i < 10; i++){
    console.log(i);
}
```

for ... in é o loop entre propriedades enumeráveis de um objeto

```
for (prop in objeto){
    console.log(prop);
}
```

for ... of é o loop entre estruturas iteráveis (arrays e straings)

```
for (letra of palavra){
        console.log(letra);
}
```

WHILE

Executa instruções enquanto a condição é verdadeira

```
while ( 1 = 1 ){
        console.log("laço infinito");
}
```

DO WHILE

Executa instruções ao menos uma vez enquanto a condição é verdadeira

```
do {
        console.log("executa uma vez");
} while ( 1 > 10 )
```

6. This

Sempre que uma função está dentro de um objeto é chamada de método. A palavra reservada this é uma referência de contexto.

```
const pessoa = {
        firstName = "Fulano",
        lastName = "Silva",
        fullName = function(){
                return this.firstName + " " + this.lastName;
        },
};
```

fullName neste caso é um metodo dentro do objeto pessoa, sempre que se usa this dentro de um método, ele vai se referir ao objeto pai deste método. O valor pode mudar de acordo com o lugar no código onde foi chamada.

Contexto	Referência
Em um objeto (método)	Próprio objeto dono do método
Sozinho	Objeto global (em navegadores, window)
Função	Objeto Global
Evento	Elemento ue recebeu o evento (um botão por exemplo com evento onclick)

Manipulando o valor de this:

CALL

```
const pessoa = {
    nome = "Fulano",
};

const pet = {
    nome = "Joãozinho",
};
function getNome(){
    console.log(this.nome);
}
getNome.call(pessoa); //nesse caso a referência vai ser o objeto pessoa
getNome.call(pet); //e aqui a referência muda para o objeto pet
```

APPLY

É semelhante ao call porém quando se precisa usar argumentos no apply, eles são enviados dentro de um array e no call não é necessário. Ex:

```
soma.call(objeto, 1, 5); //os paramentos são enviados logo depois da referência

soma.apply(objeto, [1, 5]); //os parmetros dentro de um array
```

BIND

Clona a estrutura da função onde é chamada e aplica o valor do objeto passado como parâmetro.

```
const retornaNomes = function () {
    return this.nome;     };
let fulano = retornaNomes.bind({ nome: 'Fulano" });
fulano();
```

7. Arrow Functions

const helloWorld = () => { return “Hello World”; }

Uma arrow function é equivalente a:

```
const helloWorld = function (){
        return “Hello World”;
}
```

Caso exista apenas uma linha, pode dispensar as chaves do return:

const soma = (a, b) => a + b;

Caso exista apenas um parâmetro, pode dispensar os parênteses:

const soma = a => a;

Arrow functions não permitem hoisting

8. Coleção chaveada MAP

Uma coleção de arrays no formato chave-valor, podendo ser iterado num loop for...of

const myMap = new Map();

Metodos:

SET – *myMap.set(‘apple’, ‘fruit’);*

GET – myMap.get(‘apple’);

DELETE – myMap.delete(‘apple’);

Diferença para um objeto:

Chaves de qualquer tipo,

Possuem a propriedade length,

Faceis de iterar,

Utilizado quando o valor das chaves é desconhecido,

Valores tem o mesmo tipo.

9. Coleção chaveada SET

São estruturas que armazenam apenas valores únicos, que não se repetem nunca.

Métodos:

ADD – mySet.add(1);

HAS – mySet.has(2);

DELETE – mySet.delete(1);

Diferença entre Set e Array:

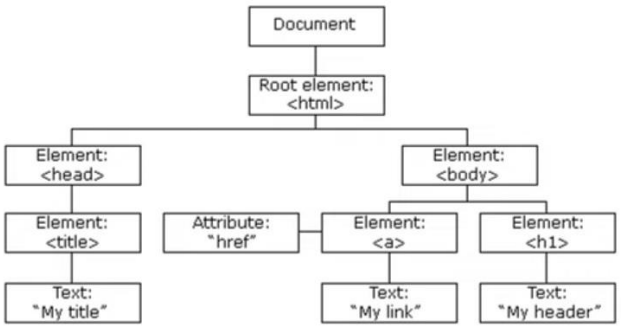
Possui valores únicos,

No lugar do length, consulta-se o número de registros com a propriedade size,

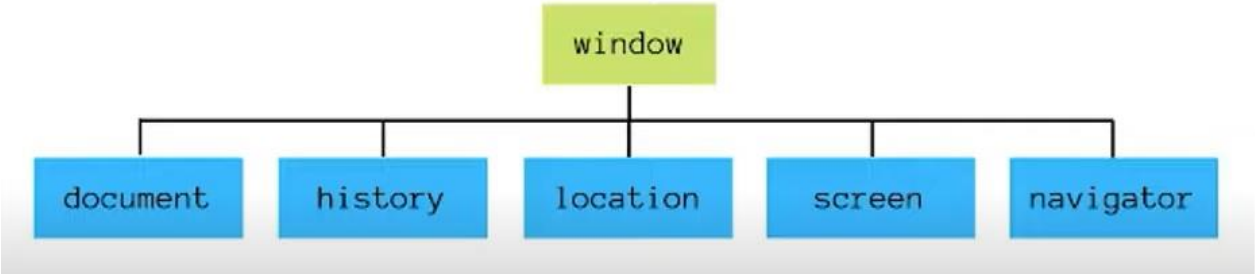
Não possui os métodos map, filter, reduce. Etc.

10. Document Object Model – DOM

O DOM html é um padrão de como acessar e modificar os elementos html de uma página. É a arvore de dominio do HTML.



O BOM é o Browser Object Model, ele é tudo que esta dentro do objeto window, que é um nível acima do DOM, sendo a arvore de dominio do browser



Estrutura da página HTML:

Tags: como o <body> e <h1>

Ids: Um dos atributos das Tags, o ID é um identificador único e não pode repetir

Classes: Atributo identificados não único

Manipulando o DOM:

Alguns Métodos:

document.getElementById('titulo'); //Retorna um único elemento que tem aquele id informado.

document.getElementsByTagName('li'); //Retorna um array com todos as ocorrências de uma tag informada.

document.getElementsByClassName('textos'); //Retorna um array com todas as ocorrências de uma classe informada.

document.querySelectorAll('.primeira-classe #primeiro-id'); //Retorna uma busca personalizada informada, com ponto para classe, sustenido para id, tag ou todos combinados.

document.createElement("div"); //Para criar novos elementos, por exemplo uma div.

document.removeChild("elemento"); //Para remover um elemento.

document.appendChild("elemento"); //Para adicionar um elemento.

document.replaceChild("new, old"); //Para substituir um elemento.

Trabalhando os estilos com um arquivo css já existente:

const meuElemento = document.getElementsById("meu-elemento");

meuElemento.classlist.add("novo-estilo"): //adiciona um novo estilo de um CSS

meuElemento.classlist.remove("classe"): //remove a classe do elemento

meuElemento.classlist.toggle("outro-estilo"): //adiciona se não existir ou remove se já existr um outro estilo de um CSS

Inserindo Estilos:

document.getElementsByTagName("p").style.color = "blue"; //Adiciona um estilo acessando diretamente o css do elemento.

Event Listener:

Diretamente no JavaScript, cria um evento que vai ser acionado no momento em que o usuário realizar uma determinada ação como mouseover, mouseout, click, dbclick, change, load...

const botao = document.getElementsById(“btn”);

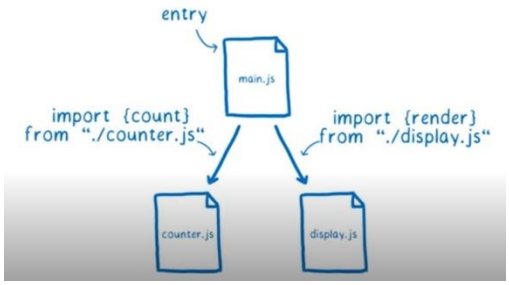
botao.addEventListener(“dbclick”, outraFuncao);

Eventos dentro do html:

Especifica a função a ser executada diretamente no html.

<h1 onclick=”mudaTexto(this)”>Clique aqui</h1>

11. Módulos JavaScript



Módulos são arquivos JavaScript que tem capacidade de importar ou exportar informações de outros arquivos do mesmo tipo. Proporciona a organização do código, compartilhamento de variáveis em escopos diferentes, explicitar as dependências dos arquivos. Módulos sempre estão em Strict mode e podem ser utilizadas as extensões .js ou .mjs

Exportar:

Named Exports:

export function minhaFuncao(){} ou *export {minhaFuncao}*

Default Exports:

Só pode haver um por arquivo e será o retorno padrão do arquivo

export default minhaFuncao;

Importar:

Named Exports:

import {funcao, variavel, classe} from ‘./arquivo.js’

Default Exports:

import valorDefault from ‘./arquivo.js’

Alias:

import { arquivo as Apelido } from ‘./arquivo.js’

Importar tudo:

*import * as INFOS from ‘./arquivo.js’*

Vinculando ao HTML:

<script type=”module” src=”./main.js></script>.

Para fazer teste localmente, será necessário estar rodando um servidor, pode ser usado o live server do vscode

12. Tratamento de erros

ECMAScript Error:

Erro que ocorre em tempo de execução, composto por uma mensagem, nome, linha e call stack.

DOMException:

Erros relacionados a um código rodando numa pagina web (DOM). Referentes a estrutura da árvore de elementos.

Return x Throw:

O return trara o retorno com uma string normal e retorna apenas a mensagem. O throw trata o retorno como um erro exibindo a menságem.

return “string inválida”; x *throw “string inválida”;*

Try ... catch:

Dentro do bloco try é verificado um pedaço de código e se houver algum erro ele será capturado na função catch.

```
try{
    verificaEntrada(string).
} catch(e) {
    throw e }
```

Finally:

Instrução que vai ser chamada independente de ocorrência de erro ou não. Não é obrigatório.

```
try{
        verificaEntrada(string).
}
catch(e) {
        throw e
}
finally {
        console.log("A string enviada foi: " + string);
}
```

Objeto Error:

Forma de declarar um novo erro, instanciando o erro e criando um nome para o objeto

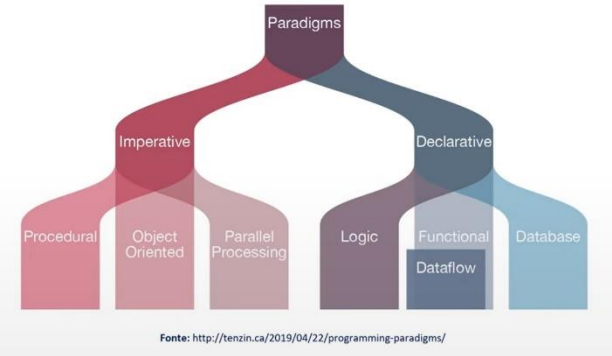
```
const MeuErro = new Error('Menságem Inválida');

MeuErro.name = 'invalidMessage';

throw MeuErro;
```

13. Orientação a Objetos

Paradigmas:



Existem dois paradigmas, o imperativo com foco em como os problemas serão resolvidos, e o declarativo com foco no que será feito como por exemplo um banco de dados em SQL! Na orientação a objetos que fazem parte do paradigma imperativo os programas são objetos que possuem uma série de propriedades.

Pilares da Orientação a Objetos:

Herança:

Objetos filhos herdam propriedades e métodos do objeto pai, e possui também suas proprias caracteristicas.

Polimorfismo:

Objetos podem herdar a mesma classe pai, mas se comportarem de forma diferentes quando invocados seus métodos

Encapsulamento:

Cada classe tem propriedades e métodos independentes do restante do código.

Abstração:

Consiste em simplicar um problema mais complexo. Abstração na verdade é generalizar alguma coisa.

Protótipos:

São o esqueleto de todos os objetos. Todos os objetos js herdam propriedades e métodos de um prototype. O objeto Object.prototype está no topo dessa cadeia.

Classes:

Classes não existem nativamente no JavaScript. Elas são Syntatic Sugar, uma sintaxe feita para facilitar a escrita. O que acontece é que utilizamos sempre objetos e estes tem protótipos.

14. Assincronicidade

Algo que não ocorre ou não se efetiva ao mesmo tempo. Por padrão o JavaScript roda de maneira síncrona.

Promisses:

Objeto de processamento assincrono. Inicialmente, seu valor é desconhecido. Ela pode, então, ser resolvida ou rejeitada. Pssui 3 estados, pending, fullfilled ou rejected.

```
async function resolvePromise(){

const myPromise = new Promise(( resolve, reject ) => {
        window.setTimeout(() => {
                resolve(console.log('Resolvida!'));
        }, 2000);
});

const resolved = await myPromise
        .then((result) =>  result + ' Passando pelo then')
        .then((result) => result + ' e agora acabou!')
        .catch((err) => console.log(err.message));

return resolved;

}
```

Async/await:

Funções assíncronas precisam dessas duas palavras chave. Quando se resolve uma promise, é preciso dizer que a função vai ser assincrona para poder utilizar o await. Ele irá parar o código esperando a promise ser resolvida.

15. O que são APIs

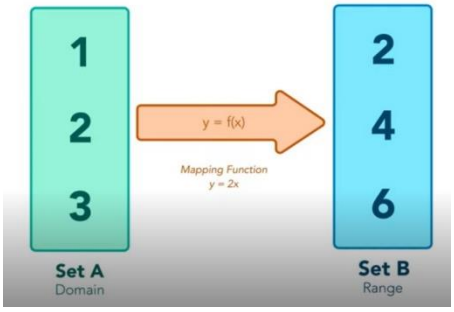
Application Programming Interface. É uma forma de intermediar os resultados do back-end com o que é apresentado no front-end. É possivel acessar esse resultado por uma URL. É muito comum que APIs retornem seus dados no formato .json (javascript object notation), portanto precisamos tratar esses dados em formato de objeto quando os recebemos.

Método fetch:

```
fetch(url, opções)
        .then(response => response.json())
        .then(json => console.log(json))
```

Ele retorna uma Promise, isso significa que precisa do uso do await

16. Método MAP



Usando a referência matemática de conjuntos, é quando um conjunto A passa por uma operação e gera um conjunto B. Lidando como um array, o metodo map não sobrescreve o array original, ele cria um novo array e realiza as operações em ordem.

array.map(callback, thisArg)

Callback: função a ser executada em cada elemento.

thisArg: é opcional: valor de this dentro da função callback

MAP x forEach:

const array = [1, 2, 3, 4, 5];

*array.map((item) => item*2); //retorno de um novo array [2, 4, 6, 8, 10]*

*array.forEach((item) => item*2); //o retorno será undefined*

A diferença vai ser o valor retornado, para usar o forEach seria preciso criar um novo array para armazenar os dados decebidos.

17. FILTER

É como colocar o array dentro de um filtro e apenas os itens que correspondam a uma determinada condição vão ser retornados num novo array.

Array.filter(callback, thisArg)

Ex.

const frutas = [‘maça fuji’, ‘maça verde’, ‘laranja’, ‘uva’];

frutas.filter((fruta) => fruta.includes(‘maçã’)); //Retorna [‘maçã fuji’, ‘maça verde’]

18. REDUCE

Não retorna outro array, ele vai executar uma função em todos os elementos do array e retornar um valor único de qualquer tipo.

array.reduce(callbackFn, initialValue);

Callback é a função a ser executada a partir do acumulador
initiaValue (opcional) é o valor sobre o qual o retorno final irá atuar

```
const callbackFn = function(acumulator, currentValue, index, array){
        //do something
}
```

array.reduce(callbackFn, initialValue)

Acumulator/PrevValue: acumulador de todas as chamadas de callbackFn
currentValue: elemento sendo acessado pela função