Sem For

array.slice(inicio, fim?)	Cria uma nova lista
array.map(fn-(item, index))	Aplica a função fn para todos os items e retorna algo
array.filter(fn-(item))	Filtra a array. Fn deve retornar true para o item que devera ser filtrado
array.reduce(fn-(accumulator, current), valorInicial)	Para cada item na lista, a função é aplicada, retornando ao fim o valor total do acummulator
array.forEach(fn(item, index))	Aplica a função fn para cada item do array
array.join(delimitador)	Transforma a array em uma string

Strings

str.split(delimitador)	Retorna uma lista contendo os elementos da string separados pelo demilitador
str.toUpperCase()	Retorna a string com todos os caracteres maiusculos
str.toLowerCase()	Retorna a string com todos os caracteres minusculos

Strings (cont)

`Eu sou uma ${variavelQualquer}`	Interpolação de string
str + str 2	Junta as duas strings
const str = "-ola"	armazena a string ola dentro da constante str
const str = " ola '1' "	armazena a string ola '1' dentro da constante str
const str = ' ola "1" '	armazena a string ola "1" dentro da constante str
\	Caracter de escape

Funções

(parametro1, parametro2) => {}	Cria uma nova arrow function
function(parametro1, parametro2) {}	Cria uma nova função
() => ({key: valor})	cria uma arrow function de uma linha que retorna um objeto
() => "-ola"	Cria uma arrow function que retorna a string ola. Caso a arrow function possua apenas uma linha, não precisa de {}

Classes

class Oi {}	cria uma nova classe Oi
new Oi()	Instancia a classe Oi
constructor{}	Pode-se ser colocado dentro da classe para ter um construtor
Oi.metodo()	chama um novo metodo
class ClassWithPrivateField { #privateField; }	Cria uma classe com um campo privado. Serve para metodos ou atributos. Só precisa colocar # antes

Coisas legais

const newObject = {...oldObject}	Cria um novo objeto igual ao anterior
const newList = [...oldList]	Cria uma lista igual a anterior
const [primeiro, segundo, terceiro] = [1, 2, 3]	Desconstroi a lista em três constantes
const [primeiro, segundo, terceiro] = obj	Desconstroi um obj para as constantes. O Objeto precisa ter as chaves iguais para funcionar!

Encontre o que procura

array.find(fn(item))	Retorna o primeiro item encontrado ou undefined
array.indexOf(fn(item))	Retorna o indice do primeiro item encontrado ou -1
array.includes(item)	Retornar True ou False

Regex no javascript

const regex = /novoRegex/gmi	Cria uma nova expressão regex que pode ser usada posteriormente
str.replace(regex, novoValor)	Substitui o que foi encontrado na string pelo novoValor
regex.test(str)	Retorna verdadeiro caso de um match ou falso caso não
regex.exec(string)	Retorna null em caso de não encontrar nada, retorna uma lista dos matchs e grupos de captura
str.match(regex)	Retorna uma lista com todos os items (Não suporta grupo de captura)
str.matchAll(regex)	Retorna um iterator que pode ser usado com for de todos os matchs(Suporta lista de captura)

Regex no javascript (cont)

[...str.matchAll(regex)]	Lista de todos os matchs (suporta grupo de captura)

Objetos

const obj = {}	Criando um novo objeto
const obj = {chaveUnica: 'valorQualquer'}	Criar objeto com valor inicial
obj.chaveUniva ou obj['chaveUnica']	Acessar o valor de um objeto
delete obj.chaveUnica	Deleta uma chave
Object.keys(obj)	Retornar um array de todas as chaves de um elemento
Object.keys(obj),map(key => obj[key])	Acessando todos os valores de um objeto
Object.values(obj)	Retornar um array de todos os valores de um objeto
Object.assign({}, ...listaDeObjetos)	Cria um novo object mesclando todos os objetos da lista
Object.freeze(obj);	Torna um objeto imutável

Objetos são como dicionários (python). Eles possuem uma chave única (key) e um valor (Qualquer coisa). Podemos cirar um objeto com a seguinte sintaxe:

Listas

array.push(novoItem)	Adiciona um novo item ao array
array.pop()	Remove o último elemento do array
array.shift()	Remove o primeiro elemento do array
array.unshift(item)	Adiciona um item ao inicio do array
array.splice(indexParaRemover, numeroDeItemsParaRemover)	Remove um item de determinada posição de um array
fruits.slice()	Copia o array

Promessas

promise.then(function resolve(res)).catch(function reject(error))	Caso de sucesso, a função dentro do then sera executada. Em de erro, a função dentro do catch sera executada
async function fn(){}	Sugar sintatico que permite a utilização do await
await promise	Faz com que todo o código abaixo do await espere para ser executado. Por baixo dos panos, coloca tudo que esta abaixo dentro de um then.

Promessas (cont)

Promise.all(listaDePromessas)	Executa todas as promises em paralelo e só executa o código abaixo quando todas são executas e bem-sucedidas. Retorna uma lista com o resultado das promessas
Promise.allSettled(listaDePromessas)	Executa todas as promises em paralelo e só executa o código abaixo quando todas são executas, com ou sem erro. Retorna uma lista com o resultado das promessas
new Promise((resolve, reject) => {})	Cria uma nova promesa. Chame resolve e passe o resultado em caso de sucesso. Chame o reject em caso de falha.

Uma promessa é um código assincrono, que sera mandado ao event loop. O resultado só estara disponível depois, mas a aplicação continuara executando o código. Ele é nãp bloqueante