Reativa Tecnologia

Exemplos de códigos em Javascript

ARRAY

Verifique se todos os elementos da lista são iguais a um determinado valor

```
const isEqual = (arr, value) => arr.every(item => item === value);

// isEqual(['foo', 'foo'], 'foo') === true
// isEqual(['foo', 'bar'], 'foo') === false
// isEqual(['bar', 'bar'], 'foo') === false
```

Verifique se todos os itens em uma lista são iguais

```
const areEqual = arr => arr.length > 0 && arr.every(item => item ===
arr[0]);

// Or
const areEqual = arr => new Set(arr).size === 1;

// areEqual([1, 2, 3, 4]) === false
// areEqual(['hello', 'hello', 'hello']) === true
```

Verifique se uma lista contém um valor correspondente a alguns critérios

```
const contains = (arr, criteria) => arr.some(v => criteria(v));
```

```
// contains([10, 20, 30], v => v > 25 )  === true
// contains([10, 20, 30], v => v > 100 || v < 15 )  === true
// contains([10, 20, 30], v => v > 100 )  === false
```

Verifique se uma lista não está vazia

```
const isNotEmpty = arr => Array.isArray(arr) && Object.keys(arr).length > 0;

// isNotEmpty([]) === false
// isNotEmpty([1, 2, 3]) === true
```

Verifique se uma lista é subconjunto de outra lista

```
const isSubset = (a, b) => (new Set(b)).size === (new Set(b.concat(a))).size;

// isSubset([1,2], [1,2,3,4]) === true
// isSubset([1,2,5], [1,2,3,4]) === false
// isSubset([6], [1,2,3,4]) === false
```

Verifique se um objeto é uma lista

```
const isArray = obj => Array.isArray(obj);
```

Clonar uma lista

```
// `arr` is an array
const clone = arr => arr.slice(0);

// Or
const clone = arr => [...arr];

// Or
const clone = arr => Array.from(arr);

// Or
const clone = arr => arr.map(x => x);

// Or
const clone = arr => JSON.parse(JSON.stringify(arr));
```

```
// Or
const clone = arr => arr.concat([]);
```

Compare duas listas, independentemente do pedido

```
// `a` and `b` are arrays
const isEqual = (a, b) => JSON.stringify(a.sort()) ===
JSON.stringify(b.sort());

// Or
const isEqual = (a, b) => a.length === b.length && a.every((v) =>
b.includes(v));

// Or
const isEqual = (a, b) => a.length === b.length && (new
Set(a.concat(b)).size === a.length);

// isEqual([1, 2, 3], [1, 2, 3]) === true
// isEqual([1, 2, 3], [1, 3, 2]) === true
// isEqual([1, 2, 3], [1, '2', 3]) === false
```

Compare duas listas

```
// `a` and `b` are arrays
const isEqual = (a, b) => JSON.stringify(a) === JSON.stringify(b);

// Or
const isEqual = (a, b) => a.length === b.length && a.every((v, i) => v ===
 b[i]);

// isEqual([1, 2, 3], [1, 2, 3]) === true
// isEqual([1, 2, 3], [1, '2', 3]) === false
```

Converter uma lista de objetos em um único objeto

```
const toObject = (arr, identifier) => arr.reduce((a, b) => ({ ...a,
[b[identifier]]: b }), {});

/*
toObject(
    [
        { id: '1', name: 'Alpha', gender: 'Male' },
        { id: '2', name: 'Bravo', gender: 'Male' },
```

```
        { id: '3', name: 'Charlie', gender: 'Female' },
    ],
    'id'
)
returns
{
    '1': { id: '1', name: 'Alpha', gender: 'Male' },
    '2': { id: '2', name: 'Bravo', gender: 'Male' },
    '3': { id: '3', name: 'Charlie', gender: 'Female' },
}
*/
```

Converter uma lista de seqüências de caracteres em números

```
const toNumbers = arr => arr.map(Number);

// Or
const toNumbers = arr => arr.map(x => +x);

// toNumbers(['2', '3', '4']) returns [2, 3, 4]
```

Crie uma lista de soma acumulada

```
const accumulate = arr => arr.map((sum => value => sum += value)(0));

// Or
const accumulate = arr => arr.reduce((a, b, i) => i === 0 ? [b] : [...a,
b + a[i - 1]], []);

// Or
const accumulate = arr => arr.reduce((a, b, i) => i === 0 ? [b] : [...a,
b + a[i - 1]], 0);

/*
accumulate([1, 2, 3, 4]) === [1, 3, 6, 10]
// 1             = 1
// 1 + 2         = 3
// 1 + 2 + 3     = 6
// 1 + 2 + 3 + 4 = 10
*/
```

Crie uma lista de números no intervalo especificado

```
const range = (min, max) => [...Array(max - min + 1).keys()].map(i => i +
min);

// Or
const range = (min, max) => Array(max - min + 1).fill(0).map((_, i) =>
min + i);

// Or
const range = (min, max) => Array.from({ length: max - min + 1 }, (_, i)
=> min + i);

// range(5, 10) === [5, 6, 7, 8, 9, 10]
```

Esvaziar uma lista

```
const empty = arr => arr.length = 0;

// Or
arr = [];
```

Encontre o comprimento da string mais longa em uma lista

```
const findLongest = words => Math.max(...(words.map(el => el.length)));

// findLongest(['always','look','on','the','bright','side','of','life'])
=== 6;
```

Encontre o item máximo de uma lista

```
const max = arr => Math.max(...arr);
```

Encontre o item mínimo de uma lista

```
const min = arr => Math.min(...arr);
```

Achatar uma lista

```
const flat = arr => [].concat.apply([], arr.map(a => Array.isArray(a) ?
flat(a) : a));
// Or
const flat = arr => arr.reduce((a, b) => Array.isArray(b) ? [...a,
```

```
...flat(b)] : [...a, b], []);

// Or
// See the browser compatibility at https://caniuse.com/#feat=array-flat
const flat = arr => arr.flat();

// flat(['cat', ['lion', 'tiger']]) returns ['cat', 'lion', 'tiger']
```

Obter um item aleatório de uma lista

```
const randomItem = arr => arr[(Math.random() * arr.length) | 0];
```

Obter a média de uma lista

```
const average = arr => arr.reduce((a, b) => a + b, 0) / arr.length;
```

Obter a interseção de listas

```
// space: O(n)
// time: O(n)
const getIntersection = (...arr) => [...(arr.flat().reduce((map, v) =>
map.set(v, (map.get(v) || 0) + 1), new Map()))].reduce((acc, [v, count])
=> void (count === arr.length && acc.push(v)) || acc, []);

// Or
// Only support two arrays
const getIntersection = (a, b) => [...new Set(a)].filter(v =>
b.includes(v));

// getIntersection([1, 2, 3], [2, 3, 4, 5]) returns [2, 3]
// getIntersection([1, 2, 3], [2, 3, 4, 5], [1, 3, 5]) returns [3]
```

Obter a soma da lista de números

```
const sum = arr => arr.reduce((a, b) => a + b, 0);
```

Obter os valores exclusivos de uma lista

```
```js
const unique = arr => [...new Set(arr)];

// Or
```
```

```
const unique = arr => arr.filter((el, i, array) => array.indexOf(el) ===
i);

// Or
const unique = arr => arr.reduce((acc, el) => acc.includes(el) ? acc :
[...acc, el], []);
```

Obter união de listas

```
const union = (...arr) => [...new Set(arr.flat())];

// union([1, 2], [2, 3], [3]) returns [1, 2, 3]
```

Mesclar duas listas

```
// Merge but don't remove the duplications
const merge = (a, b) => a.concat(b);
// Or
const merge = (a, b) => [...a, ...b];

// Merge and remove the duplications
const merge = [...new Set(a.concat(b))];
// Or
const merge = [...new Set([...a, ...b])];
```

Particionar uma lista com base em uma condição

```
const partition = (arr, criteria) => arr.reduce((acc, i) =>
(acc[criteria(i) ? 0 : 1].push(i), acc), [[], []]);

// Example
partition([1, 2, 3, 4, 5], n => n % 2);     // returns [[2, 4], [1, 3, 5]]

Remover valores falsos da lista

```js
const removeFalsy = arr => arr.filter(Boolean);

// removeFalsy([0, 'a string', '', NaN, true, 5, undefined, 'another
string', false])
// returns ['a string', true, 5, 'another string']
```

Dividir uma lista em pedaços
```

```
const chunk = (arr, size) => arr.reduce((acc, e, i) => (i % size ?
acc[acc.length - 1].push(e) : acc.push([e]), acc), []);

// Example
chunk([1, 2, 3, 4, 5, 6, 7, 8], 3);     // returns [[1, 2, 3], [4, 5, 6],
[7, 8]]
chunk([1, 2, 3, 4, 5, 6, 7, 8], 4);     // returns [[1, 2, 3, 4], [5, 6,
7, 8]]
```

Descompacte uma lista de listas

```
const unzip = arr => arr.reduce((acc, c) => (c.forEach((v, i) =>
acc[i].push(v)), acc), Array.from({ length: Math.max(...arr.map(a =>
a.length)) }, (_) => []));

// Example
unzip([['a', 1], ['b', 2], ['c', 3], ['d', 4], ['e', 5]]);  // [['a',
'b', 'c', 'd', 'e'], [1, 2, 3, 4, 5]]

/*
    a     1
        b   2
        c 3
        d 4
        e 5
*/
```

Zip várias listas

```
const zip = (...arr) => Array.from({ length: Math.max(...arr.map(a =>
a.length)) }, (_, i) => arr.map(a => a[i]));

// Example
zip(['a', 'b', 'c', 'd', 'e'], [1, 2, 3, 4, 5]);   // [['a', 1], ['b',
2], ['c', 3], ['d', 4], ['e', 5]]

/*
Does it look like a zipper?
        a 1
        b 2
        c   3
        d     4
        e       5
*/
```

DATA HORA

Calcular o número de dias de diferença entre duas datas

```
const diffDays = (date, otherDate) => Math.ceil(Math.abs(date - otherDate)
 / (1000 * 60 * 60 * 24));

// diffDays(new Date('2014-12-19'), new Date('2020-01-01')) === 1839
```

Verifique se a data está entre duas datas

```
// `min`, `max` and `date` are `Date` instances
const isBetween = (date, min, max) => (date.getTime() >= min.getTime() &&
date.getTime() <= max.getTime());
```

Verifique se a data é hoje

```
// `date` is a Date object
const isToday = (date) => date.toISOString().slice(0, 10) === new
Date().toISOString().slice(0, 10);
```

Verifique se um ano é bissexto

```
const isLeapYear = year => (((year % 4 === 0) && (year % 100 !== 0)) ||
(year % 400 === 0));

// Or
// Get the number of days in February
const isLeapYear = year => new Date(year, 1, 29).getDate() === 29;
```

Compare duas datas

```
// `a` and `b` are `Date` instances
const compare = (a, b) => a.getTime() > b.getTime();

// compare(new Date('2020-03-30'), new Date('2020-01-01')) === true
```

Converter uma data para o formato dd mm aaaa

```
// `date` is a `Date` object
const formatYmd = date => date.toISOString().slice(0, 10);

// formatYmd(new Date()) returns `2020-05-06`
```

Converter segundos para o formato hh mm ss

```
// `s` is number of seconds
const formatSeconds = s => new Date(s * 1000).toISOString().substr(11, 8);

// Or
const formatSeconds = s => (new Date(s *
1000)).toUTCString().match(/(\d\d:\d\d:\d\d)/)[0];

// Or
const formatSeconds = s => [parseInt(s / 60 / 60), parseInt(s / 60 % 60),
parseInt(s % 60)].join(':').replace(/\b(\d)\b/g, '0$1');

// Example
formatSeconds(200);     // '00:03:20'
formatSeconds(500);     // '00:08:20'
```

Extrair ano mês dia hora minuto segundo e milissegundo de uma data

```
// `date` is a `Date` object
const extract = date => date.toISOString().split(/[^0-9]/).slice(0, -1);

// `extract` is an array of [year, month, day, hour, minute, second,
millisecond]
```

Formate uma data para a localidade especificada

```
// `date` is a `Date` object
// `locale` is a locale (en-US, pt-BR, for example)
const format = (date, locale) => new
Intl.DateTimeFormat(locale).format(date);

// format(new Date(), 'pt-BR') returns `06/05/2020`
```

Obter o timestamp atual em segundos

```
const ts = () => Math.floor(new Date().getTime() / 1000);
```

Obter o nome do mês de uma data

```
// `date` is a Date object
const getMonthName = date => ['January', 'February', 'March', 'April',
'May', 'June', 'July', 'August', 'September', 'October',' November',
'December'][date.getMonth()];
```

Obter o número de dias em um determinado mês

```
// `month` is zero-based index
const daysInMonth = (month, year) => new Date(year, month, 0).getDate();
```

Obter o dia da semana de uma data

```
// `date` is a Date object
const getWeekday = date => ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday'][date.getDay()];
```

Classificar uma lista de datas

```
// `arr` is an array of `Date` items
const sortDescending = arr => arr.sort((a, b) => a.getTime() >
b.getTime());
const sortAscending = arr => arr.sort((a, b) => a.getTime() <
b.getTime());
```

Validar uma data gregoriana

```
// `m`: the month (zero-based index)
// `d`: the day
// `y`: the year
const isValidDate = (m, d, y) => 0 <= m && m <= 11 && 0 < y && y < 32768
&& 0 < d && d <= (new Date(y, m, 0)).getDate();
```

DOM

Verifique se um elemento é descendente de outro

```
const isDescendant = (child, parent) => parent.contains(child);
```

Verifique se um elemento está focado

```
const hasFocus = ele => (ele === document.activeElement);
```

Verifique se os eventos de toque são suportados

```
const touchSupported = () => ('ontouchstart' in window ||
window.DocumentTouch && document instanceof window.DocumentTouch);
```

Detectar o navegador Internet Explorer

```
const isIE = !!document.documentMode;
```

Detectar o navegador macos

```
const isMacBrowser = /Mac|iPod|iPhone|iPad/.test(navigator.platform);
```

Obter todos os irmãos de um elemento

```
const siblings = ele =>
[].slice.call(ele.parentNode.children).filter((child) => (child !== ele));
```

Obter o texto selecionado

```
const getSelectedText = () => window.getSelection().toString();
```

Volte para a página anterior

```
history.back();

// Or
history.go(-1);
```

Ocultar um elemento

```
// Pick the method that is suitable for your use case
const hide = ele => ele.style.display = 'none';
```

```
// Or
const hide = ele => ele.style.visibility = 'hidden';
```

Inserir um elemento após o outro

```
const insertAfter = (ele, anotherEle) =>
anotherEle.parentNode.insertBefore(ele, anotherEle.nextSibling);

// Or
const insertAfter = (ele, anotherEle) =>
anotherEle.insertAdjacentElement('afterend', ele);
```

Inserir um elemento antes do outro

```
const insertBefore = (ele, anotherEle) =>
anotherEle.parentNode.insertBefore(ele, anotherEle);

// Or
const insertBefore = (ele, anotherEle) =>
anotherEle.insertAdjacentElement('beforebegin', ele);
```

Inserir html fornecido após um elemento

```
const insertHtmlAfter = (html, ele) => ele.insertAdjacentHTML('afterend',
html);
```

Inserir html fornecido antes de um elemento

```
const insertHtmlBefore = (html, ele) =>
ele.insertAdjacentHTML('beforebegin', html);
```

Redirecionar para outra página

```
const goTo = url => location.href = url;
```

Recarregar a página atual

```
const reload = () => location.reload();
```

```
// Or
const reload = () => (location.href = location.href);
```

Substituir um elemento

```
const replace = (ele, newEle) => ele.parentNode.replaceChild(newEle, ele);
```

Role para o topo da página

```
const goToTop = () => window.scrollTo(0, 0);
```

Mostrar um elemento

```
const show = ele => ele.style.display = '';
```

Retirar html de um determinado texto

```
const stripHtml = html => (new DOMParser().parseFromString(html,
'text/html')).body.textContent || '';
```

Alternar um elemento

```
const toggle = ele => (ele.style.display = (ele.style.display === 'none')
? 'block' : 'none');
```

FUNÇÃO

Verifique se um valor é uma função

```
const isFunction = v => ['[object Function]', '[object
GeneratorFunction]', '[object AsyncFunction]', '[object
Promise]'].includes(Object.prototype.toString.call(v));

// Example
isFunction(function() {});          // true
isFunction(function*() {});         // true
isFunction(async function() {});    // true
```

Verifique se um valor é uma função geradora

```
const isGeneratorFunction = v => Object.prototype.toString.call(v) ===
'[object GeneratorFunction]';

// Example
isGeneratorFunction(function() {});     // false
isGeneratorFunction(function*() {});    // true
```

Verifique se um valor é uma função assíncrona

```
const isAsyncFunction = v => Object.prototype.toString.call(v) ===
'[object AsyncFunction]';

// Example
isAsyncFunction(function() {});         // false
isAsyncFunction(function*() {});        // false
isAsyncFunction(async function() {});   // true
```

Componha funções da esquerda para a direita

```
// Compose functions from left to right
const pipe = (...fns) => x => fns.reduce((y, f) => f(y), x);

// Example
const lowercase = str => str.toLowerCase();
const capitalize = str => `${str.charAt(0).toUpperCase()}${str.slice(1)}`;
const reverse = str => str.split('').reverse().join('');

const fn = pipe(lowercase, capitalize, reverse);

// We will execute `lowercase`, `capitalize` and `reverse` in order
fn('Hello World') === 'dlrow olleH';
```

Funções de composição

```
// Compose functions from right to left
const compose = (...fns) => x => fns.reduceRight((y, f) => f(y), x);

// Example
const lowercase = str => str.toLowerCase();
const capitalize = str => `${str.charAt(0).toUpperCase()}${str.slice(1)}`;
const reverse = str => str.split('').reverse().join('');

const fn = compose(reverse, capitalize, lowercase);
```

```
// We will execute `lowercase`, `capitalize` and `reverse` in order
fn('Hello World') === 'dlrow olleH';
```

Crie uma função vazia

```
const noop = () => {};

// Or
const noop = Function.prototype;
```

Caril uma função

```
const curry = (fn, ...args) => fn.length <= args.length ? fn(...args) :
curry.bind(null, fn, ...args);

// Example
const sum = (a, b, c) => a + b + c;
curry(sum)(1)(2)(3);    // 6
curry(sum)(1, 2, 3);    // 6
curry(sum, 1)(2, 3);    // 6
curry(sum, 1)(2)(3);    // 6
curry(sum, 1, 2)(3);    // 6
curry(sum, 1, 2, 3);    // 6
```

Atraso na avaliação de uma função

```
// returns a new version of `fn` that returns values as lazy evaluable
const thunkfy = fn => (...args) => () => fn(...args);

// Example
const heavyComputation = x => doStuff(x);
const unnecessarySlow = manyThings.map(heavyComputation)
    .find(result => result.criteria);
const probablyFaster = manyThings.map(thunkfy(heavyComputation))
    .find(thunk => thunk().criteria);
```

Executar uma função uma vez

```
const once = fn => ((ran = false) => () => ran ? fn : (ran = !ran, fn =
fn()))();
```

```
// Example
let n = 0;
const incOnce = once(() => ++n);
incOnce();      // n = 1
incOnce();      // n = 1
incOnce();      // n = 1
```

Inverter os argumentos de uma função

```
// Reverse the order of arguments
const flip = fn => (...args) => fn(...args.reverse());

// For binary functions
const flip = fn => (b, a) => fn(a, b);

// Or for curried functions
const flip = fn => b => a => fn(a)(b);

// Example
const isParent = (parent, child) => parent.children.includes(child);
const isChild = flip(isParent);
```

Função de identidade

```
const identity = x => x;
```

Operador lógico xor

```
// returns `true` if one of the arguments is truthy and the other is falsy

const xor = (a, b) => (a && !b) || (!a && b);

// Or
const xor = (a, b) => !(!a && !b) && !(a && b);

// Or
const xor = (a, b) => Boolean(!a ^ !b);

// Example
// xor(true, true) === false
// xor(false, false) === false
// xor(true, false) === true
// xor(false, true) === true
```

Memorizar uma função

```
const memoize = fn => ((cache = {}) => arg => cache[arg] || (cache[arg] =
fn(arg)))();

// Example
// Calculate Fibonacci numbers
const fibo = memoize(n => n <= 2 ? 1 : fibo(n - 1) + fibo(n - 2));

fibo(1);    // 1
fibo(2);    // 1
fibo(3);    // 2
fibo(4);    // 3
fibo(5);    // 5
fibo(6);    // 8
```

Aplicar parcialmente uma função

```
const partial = (fn, ...a) => (...b) => fn(...a, ...b);

// Example
const sum = (x, y) => x + y;
const inc = partial(sum, 1);
inc(9);     // 10
```

Desenrole uma função

```
// `fn` is a curried function
// `n` is the depth of parameters
const uncurry = (fn, n = 1) => (...args) => (acc => args =>
args.reduce((x, y) => x(y), acc))(fn)(args.slice(0, n));

// Example
const sum = a => b => c => a + b + c;
uncurry(sum, 1)(1)(2)(3);       // 6
uncurry(sum, 2)(1, 2)(3);       // 6
uncurry(sum, 3)(1, 2, 3);       // 6
```

MISC

Verifique se um valor é um número

```
const isNumber = value => !isNaN(parseFloat(value)) && isFinite(value);
```

Verifique se um valor é uma expressão regular

```
const isRegExp = value => Object.prototype.toString.call(value) ===
'[object RegExp]';
```

Verifique se um valor é nulo

```
const isNil = (value) => value == null;
```

Verifique se um objeto é uma promessa

```
const isPromise = obj => !!obj && (typeof obj === 'object' || typeof obj
=== 'function') && typeof obj.then === 'function';
```

Verifique se o código está sendo executado no nó js

```
const isNode = typeof process !== 'undefined' && process.versions != null
&& process.versions.node != null;
```

Verifique se o código está sendo executado no navegador

```
const isBrowser = typeof window === 'object' && typeof document ===
'object';
```

Converter celsius em fahrenheit

```
const celsiusToFahrenheit = celsius => celsius * 9/5 + 32;

const fahrenheitToCelsius = fahrenheit => (fahrenheit - 32) * 5/9;

// celsiusToFahrenheit(15) === 59
// celsiusToFahrenheit(0) === 32
// celsiusToFahrenheit(-20) === -4

// fahrenheitToCelsius(59) === 15
// fahrenheitToCelsius(32) === 0
```

Converter hex para rgb

```
const hexToRgb = hex => hex.replace(/^#?([a-f\d])([a-f\d])([a-f\d])$/i,
(_, r, g, b) => `#${r}${r}${g}${g}${b}${b}`).substring(1).match(/.
{2}/g).map(x => parseInt(x, 16));

// hexToRgb('#00ffff') === [0, 255, 255]
// hexToRgb('#0ff') === [0, 255, 255]
```

Converter cor rgb em hexadecimal

```
const rgbToHex = (red, green, blue) => `#${((1 << 24) + (red << 16) +
(green << 8) + blue).toString(16).slice(1)}`;

// Or
const rgbToHex = (red, green, blue) => `#${[red, green, blue].map(v =>
v.toString(16).padStart(2, '0')).join('')}`;

// rgbToHex(0, 255, 255) === '#00ffff'
```

Detectar o modo escuro

```
const isDarkMode = window.matchMedia && window.matchMedia('(prefers-color-
scheme: dark)').matches;
```

Facilitando funções

```
// Some easing functions
// See https://gist.github.com/gre/1650294 and https://easings.net

const linear = t => t;

const easeInQuad = t => t * t;
const easeOutQuad = t => t * (2-t);
const easeInOutQuad = t => t < .5 ? 2 * t * t : -1 + (4 - 2 * t) * t;

const easeInCubic = t => t * t * t;
const easeOutCubic = t => (--t) * t * t + 1;
const easeInOutCubic = t => t < .5 ? 4 * t * t * t : (t - 1) * (2 * t - 2)
 * (2 * t - 2) + 1;

const easeInQuart = t => t * t * t * t;
```

```
const easeOutQuart = t => 1 - (--t) * t * t * t;
const easeInOutQuart = t => t < .5 ? 8 * t * t * t * t : 1 - 8 * (--t) *
t * t * t;

const easeInQuint = t => t * t * t * t * t;
const easeOutQuint = t => 1 + (--t) * t * t * t * t;
const easeInOutQuint = t => t < .5 ? 16 * t * t * t * t * t : 1 + 16 * (--
t) * t * t * t * t;

const easeInSine = t => 1 + Math.sin(Math.PI / 2 * t - Math.PI / 2);
const easeOutSine = t => Math.sin(Math.PI / 2 * t);
const easeInOutSine = t => (1 + Math.sin(Math.PI * t - Math.PI / 2)) / 2;

const easeInElastic = t => (.04 - .04 / t) * Math.sin(25 * t) + 1;
const easeOutElastic = t => .04 * t / (--t) * Math.sin(25 * t);
const easeInOutElastic = t => (t -= .5) < 0 ? (.02 + .01 / t) *
Math.sin(50 * t) : (.02 - .01 / t) * Math.sin(50 * t) + 1;
```

Emule um lançamento de dados

```
const throwdice = () => ~~(Math.random() * 6) + 1;

// throwdice() === 4
// throwdice() === 1
// throwdice() === 6
```

Codificar um URL

```
// `encodeURIComponent` doesn't encode -_.!~*'()
const encode = url => encodeURIComponent(url).replace(/!/g,
'%21').replace(/~/g, '%7E').replace(/\*/g, '%2A').replace(/'/g,
'%27').replace(/\(/g, '%28').replace(/\)/g, '%29').replace(/%20/g, '+');
```

Gere um booleano aleatório

```
const randomBoolean = () => Math.random() >= 0.5;
```

Gere uma cor hexadecimal aleatória

```
const randomColor = () => `#${Math.random().toString(16).slice(2,
8).padEnd(6, '0')}`;
```

Gere um uuid aleatório

```
const uuid = (a) => a ? (a ^ Math.random() * 16 >> a / 4).toString(16) :
([1e7] + -1e3 + -4e3 + -8e3 + -1e11).replace(/[018]/g, uuid);
```

Obter o valor de um parâmetro de um URL

```
const getParam = (url, param) => new URLSearchParams(new
URL(url).search).get(param);

// getParam('http://domain.com?message=hello', 'message') === 'hello'
```

Executar promessas em sequência

```
// `promises` is an array of `Promise`
const run = promises => promises.reduce((p, c) => p.then(rp => c.then(rc
=> [...rp, rc])), Promise.resolve([]));

/*
run(promises).then((results) => {
    // results is an array of promise results in the same order
});
*/
```

Troque duas variáveis

```
[a, b] = [b, a];
```

Aguarde um pouco de tempo

```
const wait = async (milliseconds) => new Promise((resolve) =>
setTimeout(resolve, milliseconds));
```

NÚMERO

Adicionar um sufixo ordinal a um número

```
// `n` is a position number
const addOrdinal = n => `${n}${['st', 'nd', 'rd'][((n + 90) % 100 - 10) %
10 - 1] || 'th'}`;
```

```
// Or
const addOrdinal = n => `${n}${[, 'st', 'nd', 'rd'][/1?.$/.exec(n)] ||
'th'}`;

// Or
const addOrdinal = n => `${n}${[, 'st', 'nd', 'rd'][n % 100 >> 3^1 && n %
10] || 'th'}`;

// Or
const addOrdinal = n => `${n}${{one: 'st', two: 'nd', few: 'rd', other:
'th'}[new Intl.PluralRules('en', { type: 'ordinal' }).select(n)]}`;

// addOrdinal(1) === '1st'
// addOrdinal(2) === '2nd'
// addOrdinal(3) === '3rd'
// addOrdinal(11) === '11th'
// addOrdinal(12) === '13th'
// addOrdinal(13) === '13th'
```

Calcular números de fibonacci

```
const fibo = (n, memo = {}) => memo[n] || (n <= 2 ? 1 : (memo[n] = fibo(n
- 1, memo) + fibo(n - 2, memo)));

// Examples
fibo(1);    // 1
fibo(2);    // 1
fibo(3);    // 2
fibo(4);    // 3
fibo(5);    // 5
fibo(6);    // 8
```

Calcular a média de argumentos

```
const average = (...args) => args.reduce((a, b) => a + b) / args.length;

// average(1, 2, 3, 4) === 2.5
```

Calcular a divisão dos argumentos

```
const division = (...args) => args.reduce((a, b) => a / b);
```

```
// division(1, 2, 3, 4) === 0.04166666666666666
```

Calcular o índice de modificação da coleção

```
const mod = (a, b) => ((a % b) + b) % b;

// mod(-1, 5) === 4
// mod(3, 5) === 3
// mod(6, 5) === 1
```

Calcular o restante da divisão de argumentos

```
const remainder = (...args) => args.reduce((a, b) => a % b);

// remainder(1, 2, 3, 4) === 1
```

Calcular a soma dos argumentos

```js
const sum = (...args) => args.reduce((a, b) => a + b);

// sum(1, 2, 3, 4) === 10
```

Verifique se um número inteiro é um número primo

```js
const isPrime = num => (num > 1) && Array(Math.floor(Math.sqrt(num)) -
1).fill(0).map((_, i) => i + 2).every(i => num % i !== 0);
```

Verifique se um número é uma potência de 2

```js
const isPowerOfTwo = number => (number & (number - 1)) === 0;

// isPowerOfTwo(256) === true
// isPowerOfTwo(129) === false
```

Verifique se um número é par

```js
const isEven = number => number % 2 === 0;

// isEven(1) === false
// isEven(2) === true
```

Verifique se um número é negativo

```
const isNegative = number => Math.sign(number) === -1;

// isNegative(-3) === true
// isNegative(8) === false
```

Verifique se um número é ímpar

```
const isOdd = number => number % 2 !== 0;

// isOdd(1) === true
// isOdd(2) === false
```

Verifique se um número é positivo

```
const isPositive = number => Math.sign(number) === 1;

// isPositive(3) === true
// isPositive(-8) === false
```

Prenda um número entre dois valores

```
const clamp = (val, min = 0, max = 1) => Math.max(min, Math.min(max,
val));

// clamp(199, 10, 25) === 25;
```

Calcular o maior divisor comum entre dois números

```
const gcd = (a, b) => b === 0 ? a : gcd(b, a % b);

// gcd(10, 15) === 5
```

Converter uma sequência em número

```
const toNumber = str => +str;

// toNumber('42') === 42
```

Gere um número de ponto flutuante aleatório em um determinado intervalo

```
const randomFloat = (min, max) => Math.random() * (max - min) + min;
```

Gere um número inteiro aleatório em determinado intervalo

```
const randomInteger = (min, max) => Math.floor(Math.random() * (max - min
+ 1)) + min;
```

Multiplicar argumentos

```
const mul = (...args) => args.reduce((a, b) => a * b);

// mul(1, 2, 3, 4) === 24
```

Prefixar um número inteiro com zeros

```
const prefixWithZeros = (number, length) => (number / Math.pow(10,
length)).toFixed(length).substr(2);

// Or
const prefixWithZeros = (number, length) =>
`${Array(length).join('0')}${number}`.slice(-length);

// Or
const prefixWithZeros = (number, length) =>
String(number).padStart(length, '0');

// prefixWithZeros(42, 5) === '00042'
```

Subtrair argumentos

```
const subtract = (...args) => args.reduce((a, b) => a - b);

// subtract(1, 2, 3, 4) === -8
```

OBJETO

Verifique se um valor é um objeto simples

```
const isPlainObject = v => (!!v && typeof v === 'object' && (v.__proto__
=== null || v.__proto__ === Object.prototype));

// isPlainObject(null) === false
// isPlainObject('hello world') === false
// isPlainObject([]) === false
// isPlainObject(Object.create(null)) === false
// isPlainObject(function() {}) === false

// isPlainObject({}) === true
// isPlainObject({ a: '1', b: '2' }) === true
```

Verifique se um valor é um objeto

```
const isObject = v => (v !== null && typeof v === 'object');

// isObject(null) === false
// isObject('hello world') === false

// isObject({}) === true
// isObject([]) === true
```

Verifique se um objeto está vazio

```
const isEmpty = obj => Object.keys(obj).length === 0 && obj.constructor
=== Object;

// Or
const isEmpty = obj => JSON.stringify(obj) === '{}';
```

Crie um mapa vazio que não tenha propriedades

```
// `map` doesn't have any properties
const map = Object.create(null);

// The following `map` has `__proto__` property
// const map = {};
```

Obter o valor no caminho especificado de um objeto

```
const getValue = (path, obj) => path.split('.').reduce((acc, c) => acc &&
acc[c], obj);
```

```
// Example
getValue('a.b', { a: { b: 'Hello World' } });   // 'Hello World';
```

Inverter chaves e valores de um objeto

```
const invert = obj => Object.keys(obj).reduce((res, k) =>
Object.assign(res, {[obj[k]]: k}), {});

// Example
invert({ a: '1', b: '2', c: '3' });     // { 1: 'a', 2: 'b', 3: 'c' }
```

Omita um subconjunto de propriedades de um objeto

```
const omit = (obj, keys) => Object.keys(obj).filter(k =>
!keys.includes(k)).reduce((res, k) => Object.assign(res, {[k]: obj[k]}),
{});

// Example
omit({a: '1', b: '2', c: '3'}, ['a', 'b']);     // returns { c: '3' }
```

Escolha um subconjunto de propriedades de um objeto

```
const pick = (obj, keys) => Object.keys(obj).filter(k =>
keys.includes(k)).reduce((res, k) => Object.assign(res, {[k]: obj[k]}),
{});

// Example
pick({ a: '1', b: '2', c: '3' }, ['a', 'b']);   // returns { a: '1', b:
'2' }
```

Raso copiar um objeto

```
const shallowCopy = obj => Object.assign({}, obj);
```

CORDA

Colocar em maiúscula uma sequência

```
const capitalize = str => `${str.charAt(0).toUpperCase()}${str.slice(1)}`;
```

```
// capitalize('hello world') === 'Hello world'
```

Verifique se um caminho é relativo

```
const isRelative = path => !/^([a-z]+:)?[\\/]/i.test(path);

// Examples
isRelative('/foo/bar/baz');         // false
isRelative('C:\\foo\\bar\\baz');    // false
isRelative('foo/bar/baz.txt');      // true
isRelative('foo.md');               // true
```

Verifique se uma sequência contém caracteres minúsculos

```
const containsLowerCase = str => str !== str.toUpperCase();

// Examples
containsLowerCase('Hello World');   // true
containsLowerCase('HELLO WORLD');   // false
```

Verifique se uma sequência contém apenas dígitos

```
const isNumeric = str => !/[^0-9]/.test(str);

// isNumeric(2) === true
// isNumeric('23') === true
// isNumeric('00123') === true

// isNumeric('1.23') === false
// isNumeric('-Infinity') === false
// isNumeric('Infinity') === false
// isNumeric('NaN') === false
```

Verifique se uma sequência contém caracteres maiúsculos

```
const containsUpperCase = str => str !== str.toLowerCase();

// Examples
containsUpperCase('Hello World');   // true
containsUpperCase('hello world');   // false
```

Verifique se uma string contém espaço em branco

```
const containsWhitespace = str => str => /\s/.test(str);

// containsWhitespace('hello world') === true
```

Verifique se uma sequência é minúscula

```
const isLowerCase = str => str === str.toLowerCase();
```

Verifique se uma string está em maiúsculas

```
const isUpperCase = str => str === str.toUpperCase();
```

Verifique se um URL é absoluto

```
const isAbsoluteUrl = url => /^[a-z][a-z0-9+.-]*:/.test(url);

// Example
isAbsoluteUrl('https://1loc.dev');          // true
isAbsoluteUrl('https://1loc.dev/foo/bar');  // true
isAbsoluteUrl('1loc.dev');                  // false
isAbsoluteUrl('//1loc.dev');                // false
```

Verifique se duas strings são anagramas

```
const areAnagram = (str1, str2) =>
str1.toLowerCase().split('').sort().join('') ===
str2.toLowerCase().split('').sort().join('');

// areAnagram('listen', 'silent') === true
// areAnagram('they see', 'the eyes') === true
```

Converter uma string em caixa de camelo

```
const toCamelCase = str => str.trim().replace(/[-_\s]+(.)?/g, (_, c) => c
? c.toUpperCase() : '');

// Examples
toCamelCase('background-color');            // backgroundColor
toCamelCase('-webkit-scrollbar-thumb');     // WebkitScrollbarThumb
toCamelCase('_hello_world');                // HelloWorld
toCamelCase('hello_world');                 // helloWorld
```

Converta uma string em caso pascal

```
const toPascalCase = str => (str.match(/[a-zA-Z0-9]+/g) || []).map(w =>
`${w.charAt(0).toUpperCase()}${w.slice(1)}`).join('');

// Examples
toPascalCase('hello world');    // 'HelloWorld'
toPascalCase('hello.world');    // 'HelloWorld'
toPascalCase('foo_bar-baz');    // FooBarBaz
```

Converta uma string em url slug

```
const slugify = string => string.toLowerCase().replace(/\s+/g, '-
').replace(/[^\w-]+/g, '');

// slugify('Chapter One: Once upon a time...') === 'chapter-one-once-upon-
a-time'
```

Converta um caminho de arquivo do Windows para o caminho unix

```
const toUnixPath = path => path.replace(/[\\/]+/g, '/').replace(/^([a-zA-
Z]+:|\.\/)/, '');

// Examples
toUnixPath('./foo/bar/baz');        // foo/bar/baz
toUnixPath('C:\\foo\\bar\\baz');    // /foo/bar/baz
```

Converter estojo de camelo em estojo de kebab e vice-versa

```
const kebabToCamel = str => str.replace(/-./g, m => m.toUpperCase()[1]);

// kebabToCamel('background-color') === 'backgroundColor';

const camelToKebab = str => str.replace(/([a-z0-9])([A-Z])/g,
'$1-$2').toLowerCase();

// camelToKebab('backgroundColor') === 'background-color';
```

Converter estojo de cobra em estojo de camelo

```
const snakeToCamel = str => str.toLowerCase().replace(/(_\w)/g, m =>
m.toUpperCase().substr(1));

// snakeToCamel('HELLO_world') === 'helloWorld'
```

Gere uma sequência aleatória a partir de caracteres especificados

```
const generateString = (length, chars) => Array(length).fill('').map((v)
=> chars[Math.floor(Math.random() * chars.length)]).join('');

// generateString(10,
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ')
```

Gere uma sequência aleatória com determinado comprimento

```
const generateString = length => Array(length).fill('').map((v) =>
Math.random().toString(36).charAt(2)).join('');
```

Obter a extensão do arquivo de um nome de arquivo

```
const ext = fileName => fileName.split('.').pop();
```

Obter o nome do arquivo de um URL

```
const fileName = url => url.substring(url.lastIndexOf('/') + 1);

// fileName('http://domain.com/path/to/document.pdf') === 'document.pdf'
```

Obter o comprimento de uma string em bytes

```
const bytes = str => new Blob([str]).size;

// Examples
bytes('hello world');       // 11
bytes('🎉');                 // 4
```

Torne o primeiro caractere de uma string em minúsculas

```
const lowercaseFirst = str =>
`${str.charAt(0).toLowerCase()}${str.slice(1)}`;
```

```
// lowercaseFirst('Hello World') === 'hello World'
```

Normalizar barras do caminho do arquivo

```
const normalizePath = path => path.replace(/[\\/]+/g, '/');

// Example
normalizePath('\\foo\\bar\\baz\\');          // /foo/bar/baz/
normalizePath('.//foo//bar///////baz/');     // ./foo/bar/baz/
```

Remover espaços de uma sequência

```
const removeSpaces = str => str.replace(/\s/g, '');

// removeSpaces('hel lo wor ld') === 'helloworld'
```

Repita uma string

```
const repeat = (str, numberOfTimes) => str.repeat(numberOfTimes);

// Or
const repeat = (str, numberOfTimes) => Array(numberOfTimes).join(str);
```

Substitua todas as quebras de linha por elementos br

```
const nl2br = str => str.replace(new RegExp('\r?\n', 'g'), '<br>');

// In React
str.split('\n').map((item, index) => <React.Fragment key={index}>{item}
<br /></React.Fragment>)
```

Substitua vários espaços por um único espaço

```
// Replace spaces, tabs and new line characters
const replaceSpaces = str => str.replace(/\s\s+/g, ' ');

// Only replace spaces
const replaceOnlySpaces = str => str.replace(/  +/g, ' ');
```

```
// replaceSpaces('this\n   is     \ta    \rmessage') === 'this is a
message'
```

Inverter uma string

```
const reverse = str => str.split('').reverse().join('');

// Or
const reverse = str => [...str].reverse().join('');

// Or
const reverse = str => str.split('').reduce((rev, char)=> `${char}${rev}`,
 '');

// Or
const reverse = str => (str === '') ? '' :
`${reverse(str.substr(1))}${str.charAt(0)}`;

// reverse('hello world') === 'dlrow olleh'
```

Maiúscula o primeiro caractere de cada palavra em uma sequência

```
const uppercaseWords = str => str.split(' ').map(w =>
`${w.charAt(0).toUpperCase()}${w.slice(1)}`).join(' ');

// Or
const uppercaseWords = str => str.replace(/^(.)|\s+(.)/g, c =>
c.toUpperCase());

// uppercaseWords('hello world') === 'Hello World'
```

Baseado e adaptado do excelente 1loc, veja o repositório original