Converts a comma-separated values (CSV) string to a 2D array.

- Use `Array.prototype.slice()` and `Array.prototype.indexOf('\n')` to remove the first row (title row) if `omitFirstRow` is `true`.
- Use `String.prototype.split('\n')` to create a string for each row, then `String.prototype.split(delimiter)` to separate the values in each row.
- Omit the second argument, `delimiter`, to use a default delimiter of `','`.
- Omit the third argument, `omitFirstRow`, to include the first row (title row) of the CSV string.

```
const CSVToArray = (data, delimiter = ',', omitFirstRow = false) =>
  data
    .slice(omitFirstRow ? data.indexOf('\n') + 1 : 0)
    .split('\n')
    .map(v => v.split(delimiter));
```

```
CSVToArray('a,b\nc,d'); // [['a', 'b'], ['c', 'd']];
CSVToArray('a;b\nc;d', ';'); // [['a', 'b'], ['c', 'd']];
CSVToArray('col1,col2\na,b\nc,d', ',', true); // [['a', 'b'], ['c', 'd']];
```

title: CSVToJSON

Converts a comma-separated values (CSV) string to a 2D array of objects.
The first row of the string is used as the title row.

- Use `Array.prototype.slice()` and `Array.prototype.indexOf('\n')` and `String.prototype.split(delimiter)` to separate the first row (title row) into values.
- Use `String.prototype.split('\n')` to create a string for each row, then `Array.prototype.map()` and `String.prototype.split(delimiter)` to separate the values in each row.
- Use `Array.prototype.reduce()` to create an object for each row's values, with the keys parsed from the title row.
- Omit the second argument, `delimiter`, to use a default delimiter of `,`.

```js
const CSVToJSON = (data, delimiter = ',') => {
  const titles = data.slice(0, data.indexOf('\n')).split(delimiter);
  return data
    .slice(data.indexOf('\n') + 1)
    .split('\n')
    .map(v => {
      const values = v.split(delimiter);
      return titles.reduce(
        (obj, title, index) => ((obj[title] = values[index]), obj),
        {}
      );
    });
};
```

```js
CSVToJSON('col1,col2\na,b\nc,d');
// [{'col1': 'a', 'col2': 'b'}, {'col1': 'c', 'col2': 'd'}];
CSVToJSON('col1;col2\na;b\nc;d', ';');
// [{'col1': 'a', 'col2': 'b'}, {'col1': 'c', 'col2': 'd'}];
```

title: CSVToArray

Converts a comma-separated values (CSV) string to a 2D array.

- Use `Array.prototype.slice()` and `Array.prototype.indexOf('\n')` to remove the first row (title row) if `omitFirstRow` is `true`.
- Use `String.prototype.split('\n')` to create a string for each row, then `String.prototype.split(delimiter)` to separate the values in each row.
- Omit the second argument, `delimiter`, to use a default delimiter of `','`.
- Omit the third argument, `omitFirstRow`, to include the first row (title row) of the CSV string.

```js
const CSVToArray = (data, delimiter = ',', omitFirstRow = false) =>
  data
    .slice(omitFirstRow ? data.indexOf('\n') + 1 : 0)
    .split('\n')
    .map(v => v.split(delimiter));
```

```js
CSVToArray('a,b\nc,d'); // [['a', 'b'], ['c', 'd']];
CSVToArray('a;b\nc;d', ';'); // [['a', 'b'], ['c', 'd']];
CSVToArray('col1,col2\na,b\nc,d', ',', true); // [['a', 'b'], ['c', 'd']];
```

title: CSVToJSON

Converts a comma-separated values (CSV) string to a 2D array of objects.
The first row of the string is used as the title row.

- Use `Array.prototype.slice()` and `Array.prototype.indexOf('\n')` and `String.prototype.split(delimiter)` to separate the first row (title row) into values.
- Use `String.prototype.split('\n')` to create a string for each row, then `Array.prototype.map()` and `String.prototype.split(delimiter)` to separate the values in each row.
- Use `Array.prototype.reduce()` to create an object for each row's values, with the keys parsed from the title row.
- Omit the second argument, `delimiter`, to use a default delimiter of `,`.

```
const CSVToJSON = (data, delimiter = ',') => {
  const titles = data.slice(0, data.indexOf('\n')).split(delimiter);
  return data
    .slice(data.indexOf('\n') + 1)
    .split('\n')
    .map(v => {
      const values = v.split(delimiter);
      return titles.reduce(
        (obj, title, index) => ((obj[title] = values[index]), obj),
        {}
      );
    });
};
```

```
CSVToJSON('col1,col2\na,b\nc,d');
// [{'col1': 'a', 'col2': 'b'}, {'col1': 'c', 'col2': 'd'}];
CSVToJSON('col1;col2\na;b\nc;d', ';');
// [{'col1': 'a', 'col2': 'b'}, {'col1': 'c', 'col2': 'd'}];
```

title: HSBToRGB

Converts a HSB color tuple to RGB format.

- Use the HSB to RGB conversion formula to convert to the appropriate format.
- The range of the input parameters is H: [0, 360], S: [0, 100], B: [0, 100].
- The range of all output values is [0, 255].

```
const HSBToRGB = (h, s, b) => {
  s /= 100;
  b /= 100;
  const k = (n) => (n + h / 60) % 6;
  const f = (n) => b * (1 - s * Math.max(0, Math.min(k(n), 4 - k(n), 1)));
  return [255 * f(5), 255 * f(3), 255 * f(1)];
};
```

```
HSBToRGB(18, 81, 99); // [252.45, 109.31084999999996, 47.965499999999984]
```

title: HSLToRGB

Converts a HSL color tuple to RGB format.

- Use the HSL to RGB conversion formula to convert to the appropriate format.
- The range of the input parameters is H: [0, 360], S: [0, 100], L: [0, 100].
- The range of all output values is [0, 255].

```
const HSLToRGB = (h, s, l) => {
  s /= 100;
  l /= 100;
  const k = n => (n + h / 30) % 12;
  const a = s * Math.min(l, 1 - l);
  const f = n =>
    l - a * Math.max(-1, Math.min(k(n) - 3, Math.min(9 - k(n), 1)));
  return [255 * f(0), 255 * f(8), 255 * f(4)];
};
```

```
HSLToRGB(13, 100, 11); // [56.1, 12.155, 0]
```

title: JSONToFile

Writes a JSON object to a file.

- Use `fs.writeFileSync()`, template literals and `JSON.stringify()` to write a `json` object to a `.json` file.

```js
const fs = require('fs');

const JSONToFile = (obj, filename) =>
  fs.writeFileSync(`${filename}.json`, JSON.stringify(obj, null, 2));


JSONToFile({ test: 'is passed' }, 'testJsonFile');
// writes the object to 'testJsonFile.json'
```

title: JSONtoCSV

Converts an array of objects to a comma-separated values (CSV) string that contains only the `columns` specified.

- Use `Array.prototype.join(delimiter)` to combine all the names in `columns` to create the first row.
- Use `Array.prototype.map()` and `Array.prototype.reduce()` to create a row for each object. Substitute non-existent values with empty strings and only mapping values in `columns` .
- Use `Array.prototype.join('\n')` to combine all rows into a string.
- Omit the third argument, `delimiter` , to use a default delimiter of `','` .

```js
const JSONtoCSV = (arr, columns, delimiter = ',') =>
  [
    columns.join(delimiter),
    ...arr.map(obj =>
      columns.reduce(
        (acc, key) =>
          `${acc}${!acc.length ? '' : delimiter}"${!obj[key] ? '' : obj[key]}"`,
        ''
      )
    ),
  ].join('\n');


JSONtoCSV(
  [{ a: 1, b: 2 }, { a: 3, b: 4, c: 5 }, { a: 6 }, { b: 7 }],
  ['a', 'b']
); // 'a,b\n"1","2"\n"3","4"\n"6",""\n"","7"'
JSONtoCSV(
  [{ a: 1, b: 2 }, { a: 3, b: 4, c: 5 }, { a: 6 }, { b: 7 }],
  ['a', 'b'],
  ';'
); // 'a;b\n"1";"2"\n"3";"4"\n"6";""\n"";"7"'
```

title: RGBToHSB

Converts a RGB color tuple to HSB format.

- Use the RGB to HSB conversion formula to convert to the appropriate format.
- The range of all input parameters is [0, 255].
- The range of the resulting values is H: [0, 360], S: [0, 100], B: [0, 100].

```
const RGBToHSB = (r, g, b) => {
  r /= 255;
  g /= 255;
  b /= 255;
  const v = Math.max(r, g, b),
    n = v - Math.min(r, g, b);
  const h =
    n === 0 ? 0 : n && v === r ? (g - b) / n : v === g ? 2 + (b - r) / n : 4 + (r - g) / n;
  return [60 * (h < 0 ? h + 6 : h), v && (n / v) * 100, v * 100];
};
```

```
RGBToHSB(252, 111, 48);
// [18.529411764705856, 80.95238095238095, 98.82352941176471]
```

title: RGBToHSL

Converts a RGB color tuple to HSL format.

- Use the RGB to HSL conversion formula to convert to the appropriate format.
- The range of all input parameters is [0, 255].
- The range of the resulting values is H: [0, 360], S: [0, 100], L: [0, 100].

```
const RGBToHSL = (r, g, b) => {
  r /= 255;
  g /= 255;
  b /= 255;
  const l = Math.max(r, g, b);
  const s = l - Math.min(r, g, b);
  const h = s
    ? l === r
      ? (g - b) / s
      : l === g
      ? 2 + (b - r) / s
      : 4 + (r - g) / s
    : 0;
  return [
    60 * h < 0 ? 60 * h + 360 : 60 * h,
    100 * (s ? (l <= 0.5 ? s / (2 * l - s) : s / (2 - (2 * l - s))) : 0),
    (100 * (2 * l - s)) / 2,
  ];
};
```

```
RGBToHSL(45, 23, 11); // [21.17647, 60.71428, 10.98039]
```

title: RGBToHex

Converts the values of RGB components to a hexadecimal color code.

- Convert given RGB parameters to hexadecimal string using bitwise left-shift operator (`<<`) and `Number.prototype.toString(16)` .
- Use `String.prototype.padStart(6, '0')` to get a 6-digit hexadecimal value.

```
const RGBToHex = (r, g, b) =>
  ((r << 16) + (g << 8) + b).toString(16).padStart(6, '0');
```

```
RGBToHex(255, 165, 1); // 'ffa501'
```

title: URLJoin

Joins all given URL segments together, then normalizes the resulting URL.

- Use `String.prototype.join('/')` to combine URL segments.
- Use a series of `String.prototype.replace()` calls with various regexps to normalize the resulting URL (remove double slashes, add proper slashes for protocol, remove slashes before parameters, combine parameters with `'&'` and normalize first parameter delimiter).

```
const URLJoin = (...args) =>
  args
    .join('/')
    .replace(/[\/]+/g, '/')
    .replace(/^(.+):\//, '$1://')
    .replace(/^file:/, 'file:/')
    .replace(/\/(\?|&|#[^!])/g, '$1')
    .replace(/\?/g, '&')
    .replace('&', '?');
```

```
URLJoin('http://www.google.com', 'a', '/b/cd', '?foo=123', '?bar=foo');
// 'http://www.google.com/a/b/cd?foo=123&bar=foo'
```

title: UUIDGeneratorBrowser

Generates a UUID in a browser.

- Use `Crypto.getRandomValues()` to generate a UUID, compliant with RFC4122 version 4.
- Use `Number.prototype.toString(16)` to convert it to a proper UUID.

```
const UUIDGeneratorBrowser = () =>
  ([1e7] + -1e3 + -4e3 + -8e3 + -1e11).replace(/[018]/g, c =>
    (
      c ^
      (crypto.getRandomValues(new Uint8Array(1))[0] & (15 >> (c / 4)))
    ).toString(16)
  );
```

```
UUIDGeneratorBrowser(); // '7982fcfe-5721-4632-bede-6000885be57d'
```

title: UUIDGeneratorNode

Generates a UUID in Node.JS.

- Use `crypto.randomBytes()` to generate a UUID, compliant with RFC4122 version 4.
- Use `Number.prototype.toString(16)` to convert it to a proper UUID.

```
const crypto = require('crypto');

const UUIDGeneratorNode = () =>
  ([1e7] + -1e3 + -4e3 + -8e3 + -1e11).replace(/[018]/g, c =>
    (c ^ (crypto.randomBytes(1)[0] & (15 >> (c / 4)))).toString(16)
  );


UUIDGeneratorNode(); // '79c7c136-60ee-40a2-beb2-856f1feabefc'
```

title: accumulate

Creates an array of partial sums.

- Use `Array.prototype.reduce()` , initialized with an empty array accumulator to iterate over `nums` .
- Use `Array.prototype.slice(-1)` , the spread operator (`...`) and the unary `+` operator to add each value to the accumulator array containing the previous sums.

```
const accumulate = (...nums) =>
  nums.reduce((acc, n) => [...acc, n + +acc.slice(-1)], []);


accumulate(1, 2, 3, 4); // [1, 3, 6, 10]
accumulate(...[1, 2, 3, 4]); // [1, 3, 6, 10]
```

title: addClass

Adds a class to an HTML element.

- Use `Element.classList` and `DOMTokenList.add()` to add the specified class to the element.

```
const addClass = (el, className) => el.classList.add(className);
```

```js
addClass(document.querySelector('p'), 'special');
// The paragraph will now have the 'special' class
```

title: addDaysToDate

Calculates the date of `n` days from the given date, returning its string representation.

- Use `new Date()` to create a date object from the first argument.
- Use `Date.prototype.getDate()` and `Date.prototype.setDate()` to add `n` days to the given date.
- Use `Date.prototype.toISOString()` to return a string in `yyyy-mm-dd` format.

```js
const addDaysToDate = (date, n) => {
  const d = new Date(date);
  d.setDate(d.getDate() + n);
  return d.toISOString().split('T')[0];
};
```

```js
addDaysToDate('2020-10-15', 10); // '2020-10-25'
addDaysToDate('2020-10-15', -10); // '2020-10-05'
```

title: addEventListenerAll

Attaches an event listener to all the provided targets.

- Use `Array.prototype.forEach()` and `EventTarget.addEventListener()` to attach the provided `listener` for the given event `type` to all `targets` .

```js
const addEventListenerAll = (targets, type, listener, options, useCapture) => {
  targets.forEach(target =>
    target.addEventListener(type, listener, options, useCapture)
  );
};
```

```js
addEventListenerAll(document.querySelectorAll('a'), 'click', () =>
  console.log('Clicked a link')
);
// Logs 'Clicked a link' whenever any anchor element is clicked
```

title: addMinutesToDate

Calculates the date of `n` minutes from the given date, returning its string representation.

- Use `new Date()` to create a date object from the first argument.
- Use `Date.prototype.getTime()` and `Date.prototype.setTime()` to add `n` minutes to the given date.
- Use `Date.prototype.toISOString()`, `String.prototype.split()` and `String.prototype.replace()` to return a string in `yyyy-mm-dd HH:MM:SS` format.

```
const addMinutesToDate = (date, n) => {
  const d = new Date(date);
  d.setTime(d.getTime() + n * 60000);
  return d.toISOString().split('.')[0].replace('T',' ');
};
```

```
addMinutesToDate('2020-10-19 12:00:00', 10); // '2020-10-19 12:10:00'
addMinutesToDate('2020-10-19', -10); // '2020-10-18 23:50:00'
```

title: addMultipleListeners

Adds multiple event listeners with the same handler to an element.

- Use `Array.prototype.forEach()` and `EventTarget.addEventListener()` to add multiple event listeners with an assigned callback function to an element.

```
const addMultipleListeners = (el, types, listener, options, useCapture) => {
  types.forEach(type =>
    el.addEventListener(type, listener, options, useCapture)
  );
};
```

```
addMultipleListeners(
  document.querySelector('.my-element'),
  ['click', 'mousedown'],
  () => { console.log('hello!') }
);
```

title: addStyles

Adds the provided styles to the given element.

- Use `Object.assign()` and `ElementCSSInlineStyle.style` to merge the provided `styles` object into the style of the given element.

```
const addStyles = (el, styles) => Object.assign(el.style, styles);
```

```
addStyles(document.getElementById('my-element'), {
  background: 'red',
  color: '#ffff00',
  fontSize: '3rem'
});
```

title: addWeekDays

Calculates the date after adding the given number of business days.

- Use `Array.from()` to construct an array with `length` equal to the `count` of business days to be added.
- Use `Array.prototype.reduce()` to iterate over the array, starting from `startDate` and incrementing, using `Date.prototype.getDate()` and `Date.prototype.setDate()` .
- If the current `date` is on a weekend, update it again by adding either one day or two days to make it a weekday.
- **NOTE:** Does not take official holidays into account.

```
const addWeekDays = (startDate, count) =>
  Array.from({ length: count }).reduce(date => {
    date = new Date(date.setDate(date.getDate() + 1));
    if (date.getDay() % 6 === 0)
      date = new Date(date.setDate(date.getDate() + (date.getDay() / 6 + 1)));
    return date;
  }, startDate);
```

```
addWeekDays(new Date('Oct 09, 2020'), 5); // 'Oct 16, 2020'
addWeekDays(new Date('Oct 12, 2020'), 5); // 'Oct 19, 2020'
```

title: all

Checks if the provided predicate function returns `true` for all elements in a collection.

- Use `Array.prototype.every()` to test if all elements in the collection return `true` based on `fn` .
- Omit the second argument, `fn` , to use `Boolean` as a default.

```
const all = (arr, fn = Boolean) => arr.every(fn);


all([4, 2, 3], x => x > 1); // true
all([1, 2, 3]); // true
```

title: allEqual

Checks if all elements in an array are equal.

- Use `Array.prototype.every()` to check if all the elements of the array are the same as the first one.
- Elements in the array are compared using the strict comparison operator, which does not account for `NaN` self-inequality.

```
const allEqual = arr => arr.every(val => val === arr[0]);


allEqual([1, 2, 3, 4, 5, 6]); // false
allEqual([1, 1, 1, 1]); // true
```

title: allEqualBy

Checks if all elements in an array are equal, based on the provided mapping function.

- Apply `fn` to the first element of `arr` .
- Use `Array.prototype.every()` to check if `fn` returns the same value for all elements in the array as it did for the first one.

- Elements in the array are compared using the strict comparison operator, which does not account for `NaN` self-inequality.

```
const allEqualBy = (arr, fn) => {
  const eql = fn(arr[0]);
  return arr.every(val => fn(val) === eql);
};
```

```
allEqualBy([1.1, 1.2, 1.3], Math.round); // true
allEqualBy([1.1, 1.3, 1.6], Math.round); // false
```

title: allUnique

Checks if all elements in an array are unique.

- Create a new `Set` from the mapped values to keep only unique occurrences.
- Use `Array.prototype.length` and `Set.prototype.size` to compare the length of the unique values to the original array.

```
const allUnique = arr => arr.length === new Set(arr).size;
```

```
allUnique([1, 2, 3, 4]); // true
allUnique([1, 1, 2, 3]); // false
```

title: allUniqueBy

Checks if all elements in an array are unique, based on the provided mapping function.

- Use `Array.prototype.map()` to apply `fn` to all elements in `arr` .
- Create a new `Set` from the mapped values to keep only unique occurrences.
- Use `Array.prototype.length` and `Set.prototype.size` to compare the length of the unique mapped values to the original array.

```
const allUniqueBy = (arr, fn) => arr.length === new Set(arr.map(fn)).size;
```

```
allUniqueBy([1.2, 2.4, 2.9], Math.round); // true
allUniqueBy([1.2, 2.3, 2.4], Math.round); // false
```

title: and
unlisted: true

Checks if both arguments are `true` .

- Use the logical and (`&&`) operator on the two given values.

```
const and = (a, b) => a && b;
```

```
and(true, true); // true
and(true, false); // false
and(false, false); // false
```

title: any

Checks if the provided predicate function returns `true` for at least one element in a collection.

- Use `Array.prototype.some()` to test if any elements in the collection return `true` based on `fn` .
- Omit the second argument, `fn` , to use `Boolean` as a default.

```
const any = (arr, fn = Boolean) => arr.some(fn);
```

```
any([0, 1, 2, 0], x => x >= 2); // true
any([0, 0, 1, 0]); // true
```

title: aperture

Creates an array of `n` -tuples of consecutive elements.

- Use `Array.prototype.slice()` and `Array.prototype.map()` to create an array of appropriate length.
- Populate the array with `n` -tuples of consecutive elements from `arr` .
- If `n` is greater than the length of `arr` , return an empty array.

```
const aperture = (n, arr) =>
  n > arr.length
    ? []
    : arr.slice(n - 1).map((v, i) => arr.slice(i, i + n));
```

```
aperture(2, [1, 2, 3, 4]); // [[1, 2], [2, 3], [3, 4]]
aperture(3, [1, 2, 3, 4]); // [[1, 2, 3], [2, 3, 4]]
aperture(5, [1, 2, 3, 4]); // []
```

title: approximatelyEqual

Checks if two numbers are approximately equal to each other.

- Use `Math.abs()` to compare the absolute difference of the two values to `epsilon` .
- Omit the third argument, `epsilon` , to use a default value of `0.001` .

```
const approximatelyEqual = (v1, v2, epsilon = 0.001) =>
  Math.abs(v1 - v2) < epsilon;
```

```
approximatelyEqual(Math.PI / 2.0, 1.5708); // true
```

title: arithmeticProgression

Creates an array of numbers in the arithmetic progression, starting with the given positive integer and up to the specified limit.

- Use `Array.from()` to create an array of the desired length, `lim/n` . Use a map function to fill it with the desired values in the given range.

```js
const arithmeticProgression  = (n, lim) =>
  Array.from({ length: Math.ceil(lim / n) }, (_, i) => (i + 1) * n );


arithmeticProgression(5, 25); // [5, 10, 15, 20, 25]
```

title: arrayToCSV

Converts a 2D array to a comma-separated values (CSV) string.

- Use `Array.prototype.map()` and `Array.prototype.join(delimiter)` to combine individual 1D arrays (rows) into strings.
- Use `Array.prototype.join('\n')` to combine all rows into a CSV string, separating each row with a newline.
- Omit the second argument, `delimiter` , to use a default delimiter of `,` .

```js
const arrayToCSV = (arr, delimiter = ',') =>
  arr
    .map(v =>
      v.map(x => (isNaN(x) ? `"${x.replace(/"/g, '""')}"` : x)).join(delimiter)
    )
    .join('\n');


arrayToCSV([['a', 'b'], ['c', 'd']]); // '"a","b"\n"c","d"'
arrayToCSV([['a', 'b'], ['c', 'd']], ';'); // '"a";"b"\n"c";"d"'
arrayToCSV([['a', '"b" great'], ['c', 3.1415]]);
// '"a","""b"" great"\n"c",3.1415'
```

title: arrayToHTMLList

Converts the given array elements into `<li>` tags and appends them to the list of the given id.

- Use `Array.prototype.map()` and `Document.querySelector()` to create a list of html tags.

```
const arrayToHTMLList = (arr, listID) =>
  document.querySelector(`#${listID}`).innerHTML += arr
    .map(item => `<li>${item}</li>`)
    .join('');


arrayToHTMLList(['item 1', 'item 2'], 'myListID');
```

title: ary

Creates a function that accepts up to `n` arguments, ignoring any additional arguments.

- Call the provided function, `fn` , with up to `n` arguments, using `Array.prototype.slice(0, n)` and the spread operator (`...`).

```
const ary = (fn, n) => (...args) => fn(...args.slice(0, n));


const firstTwoMax = ary(Math.max, 2);
[[2, 6, 'a'], [6, 4, 8], [10]].map(x => firstTwoMax(...x)); // [6, 6, 10]
```

title: assertValidKeys

Validates all keys in an object match the given `keys` .

- Use `Object.keys()` to get the keys of the given object, `obj` .
- Use `Array.prototype.every()` and `Array.prototype.includes()` to validate that each key in the object is specified in the `keys` array.

```
const assertValidKeys = (obj, keys) =>
  Object.keys(obj).every(key => keys.includes(key));


assertValidKeys({ id: 10, name: 'apple' }, ['id', 'name']); // true
assertValidKeys({ id: 10, name: 'apple' }, ['id', 'type']); // false
```

title: atob

Decodes a string of data which has been encoded using base-64 encoding.

- Create a `Buffer` for the given string with base-64 encoding and use `Buffer.toString('binary')` to return the decoded string.

```
const atob = str => Buffer.from(str, 'base64').toString('binary');
```

```
atob('Zm9vYmFy'); // 'foobar'
```

title: attempt

Attempts to invoke a function with the provided arguments, returning either the result or the caught error object.

- Use a `try... catch` block to return either the result of the function or an appropriate error.
- If the caught object is not an `Error` , use it to create a new `Error` .

```
const attempt = (fn, ...args) => {
  try {
    return fn(...args);
  } catch (e) {
    return e instanceof Error ? e : new Error(e);
  }
};
```

```
let elements = attempt(function(selector) {
  return document.querySelectorAll(selector);
}, '>_>');
if (elements instanceof Error) elements = []; // elements = []
```

title: average

Calculates the average of two or more numbers.

- Use `Array.prototype.reduce()` to add each value to an accumulator, initialized with a value of `0`.
- Divide the resulting array by its length.

```js
const average = (...nums) =>
  nums.reduce((acc, val) => acc + val, 0) / nums.length;
```

```js
average(...[1, 2, 3]); // 2
average(1, 2, 3); // 2
```

title: averageBy

Calculates the average of an array, after mapping each element to a value using the provided function.

- Use `Array.prototype.map()` to map each element to the value returned by `fn`.
- Use `Array.prototype.reduce()` to add each value to an accumulator, initialized with a value of `0`.
- Divide the resulting array by its length.

```js
const averageBy = (arr, fn) =>
  arr
    .map(typeof fn === 'function' ? fn : val => val[fn])
    .reduce((acc, val) => acc + val, 0) / arr.length;
```

```js
averageBy([{ n: 4 }, { n: 2 }, { n: 8 }, { n: 6 }], o => o.n); // 5
averageBy([{ n: 4 }, { n: 2 }, { n: 8 }, { n: 6 }], 'n'); // 5
```

title: bifurcate

Splits values into two groups, based on the result of the given `filter` array.

- Use `Array.prototype.reduce()` and `Array.prototype.push()` to add elements to groups, based on `filter`.
- If `filter` has a truthy value for any element, add it to the first group, otherwise add it to the second group.

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

```
bifurcate(['beep', 'boop', 'foo', 'bar'], [true, true, false, true]);
// [ ['beep', 'boop', 'bar'], ['foo'] ]
```

title: bifurcateBy

Splits values into two groups, based on the result of the given filtering function.

- Use `Array.prototype.reduce()` and `Array.prototype.push()` to add elements to groups, based on the value returned by `fn` for each element.
- If `fn` returns a truthy value for any element, add it to the first group, otherwise add it to the second group.

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

```
bifurcateBy(['beep', 'boop', 'foo', 'bar'], x => x[0] === 'b');
// [ ['beep', 'boop', 'bar'], ['foo'] ]
```

title: binary

Creates a function that accepts up to two arguments, ignoring any additional arguments.

- Call the provided function, `fn` , with the first two arguments given.

```
const binary = fn => (a, b) => fn(a, b);
```

```
['2', '1', '0'].map(binary(Math.max)); // [2, 1, 2]
```

title: binarySearch

Finds the index of a given element in a sorted array using the binary search algorithm.

- Declare the left and right search boundaries, `l` and `r`, initialized to `0` and the `length` of the array respectively.
- Use a `while` loop to repeatedly narrow down the search subarray, using `Math.floor()` to cut it in half.
- Return the index of the element if found, otherwise return `-1`.
- **Note:** Does not account for duplicate values in the array.

```
const binarySearch = (arr, item) => {
  let l = 0,
    r = arr.length - 1;
  while (l <= r) {
    const mid = Math.floor((l + r) / 2);
    const guess = arr[mid];
    if (guess === item) return mid;
    if (guess > item) r = mid - 1;
    else l = mid + 1;
  }
  return -1;
};
```

```
binarySearch([1, 2, 3, 4, 5], 1); // 0
binarySearch([1, 2, 3, 4, 5], 5); // 4
binarySearch([1, 2, 3, 4, 5], 6); // -1
```

title: bind

Creates a function that invokes `fn` with a given context, optionally prepending any additional supplied parameters to the arguments.

- Return a `function` that uses `Function.prototype.apply()` to apply the given `context` to `fn`.
- Use the spread operator (`...`) to prepend any additional supplied parameters to the arguments.

```
const bind = (fn, context, ...boundArgs) => (...args) =>
  fn.apply(context, [...boundArgs, ...args]);
```

```
function greet(greeting, punctuation) {
  return greeting + ' ' + this.user + punctuation;
}
const freddy = { user: 'fred' };
const freddyBound = bind(greet, freddy);
console.log(freddyBound('hi', '!')); // 'hi fred!'
```

title: bindAll

Binds methods of an object to the object itself, overwriting the existing method.

- Use `Array.prototype.forEach()` to iterate over the given `fns` .
- Return a function for each one, using `Function.prototype.apply()` to apply the given context (`obj`) to `fn` .

```
const bindAll = (obj, ...fns) =>
  fns.forEach(
    fn => (
      (f = obj[fn]),
      (obj[fn] = function() {
        return f.apply(obj);
      })
    )
  );
```

```
let view = {
  label: 'docs',
  click: function() {
    console.log('clicked ' + this.label);
  }
};
bindAll(view, 'click');
document.body.addEventListener('click', view.click);
// Log 'clicked docs' when clicked.
```

title: bindKey

Creates a function that invokes the method at a given key of an object, optionally prepending any additional supplied parameters to the arguments.

- Return a `function` that uses `Function.prototype.apply()` to bind `context[fn]` to `context` .
- Use the spread operator (`...`) to prepend any additional supplied parameters to the arguments.

```
const bindKey = (context, fn, ...boundArgs) => (...args) =>
  context[fn].apply(context, [...boundArgs, ...args]);
```

```
const freddy = {
  user: 'fred',
  greet: function(greeting, punctuation) {
    return greeting + ' ' + this.user + punctuation;
  }
};
const freddyBound = bindKey(freddy, 'greet');
console.log(freddyBound('hi', '!')); // 'hi fred!'
```

title: binomialCoefficient

Calculates the number of ways to choose `k` items from `n` items without repetition and without order.

- Use `Number.isNaN()` to check if any of the two values is `NaN` .
- Check if `k` is less than `0` , greater than or equal to `n` , equal to `1` or `n - 1` and return the appropriate result.
- Check if `n - k` is less than `k` and switch their values accordingly.
- Loop from `2` through `k` and calculate the binomial coefficient.
- Use `Math.round()` to account for rounding errors in the calculation.

```
const binomialCoefficient = (n, k) => {
  if (Number.isNaN(n) || Number.isNaN(k)) return NaN;
  if (k < 0 || k > n) return 0;
  if (k === 0 || k === n) return 1;
  if (k === 1 || k === n - 1) return n;
  if (n - k < k) k = n - k;
  let res = n;
  for (let j = 2; j <= k; j++) res *= (n - j + 1) / j;
  return Math.round(res);
};
```

```
binomialCoefficient(8, 2); // 28
```

title: both
unlisted: true

Checks if both of the given functions return `true` for a given set of arguments.

- Use the logical and (`&&`) operator on the result of calling the two functions with the supplied `args` .

```
const both = (f, g) => (...args) => f(...args) && g(...args);
```

```
const isEven = num => num % 2 === 0;
const isPositive = num => num > 0;
const isPositiveEven = both(isEven, isPositive);
isPositiveEven(4); // true
isPositiveEven(-2); // false
```

title: HSBToRGB

Converts a HSB color tuple to RGB format.

- Use the HSB to RGB conversion formula to convert to the appropriate format.
- The range of the input parameters is H: [0, 360], S: [0, 100], B: [0, 100].
- The range of all output values is [0, 255].

```
const HSBToRGB = (h, s, b) => {
  s /= 100;
  b /= 100;
  const k = (n) => (n + h / 60) % 6;
  const f = (n) => b * (1 - s * Math.max(0, Math.min(k(n), 4 - k(n), 1)));
  return [255 * f(5), 255 * f(3), 255 * f(1)];
};
```

```
HSBToRGB(18, 81, 99); // [252.45, 109.31084999999996, 47.965499999999984]
```

title: HSLToRGB

Converts a HSL color tuple to RGB format.

- Use the HSL to RGB conversion formula to convert to the appropriate format.
- The range of the input parameters is H: [0, 360], S: [0, 100], L: [0, 100].
- The range of all output values is [0, 255].

```
const HSLToRGB = (h, s, l) => {
  s /= 100;
  l /= 100;
  const k = n => (n + h / 30) % 12;
  const a = s * Math.min(l, 1 - l);
  const f = n =>
    l - a * Math.max(-1, Math.min(k(n) - 3, Math.min(9 - k(n), 1)));
  return [255 * f(0), 255 * f(8), 255 * f(4)];
};
```

```
HSLToRGB(13, 100, 11); // [56.1, 12.155, 0]
```

title: JSONToFile

Writes a JSON object to a file.

- Use `fs.writeFileSync()` , template literals and `JSON.stringify()` to write a `json` object to a `.json` file.

```
const fs = require('fs');

const JSONToFile = (obj, filename) =>
  fs.writeFileSync(`${filename}.json`, JSON.stringify(obj, null, 2));
```

```
JSONToFile({ test: 'is passed' }, 'testJsonFile');
// writes the object to 'testJsonFile.json'
```

title: JSONtoCSV

Converts an array of objects to a comma-separated values (CSV) string that contains only the `columns` specified.

- Use `Array.prototype.join(delimiter)` to combine all the names in `columns` to create the first row.
- Use `Array.prototype.map()` and `Array.prototype.reduce()` to create a row for each object. Substitute non-existent values with empty strings and only mapping values in `columns`.
- Use `Array.prototype.join('\n')` to combine all rows into a string.
- Omit the third argument, `delimiter`, to use a default delimiter of `','`.

```
const JSONtoCSV = (arr, columns, delimiter = ',') =>
  [
    columns.join(delimiter),
    ...arr.map(obj =>
      columns.reduce(
        (acc, key) =>
          `${acc}${!acc.length ? '' : delimiter}"${!obj[key] ? '' : obj[key]}"`,
        ''
      )
    ),
  ].join('\n');


JSONtoCSV(
  [{ a: 1, b: 2 }, { a: 3, b: 4, c: 5 }, { a: 6 }, { b: 7 }],
  ['a', 'b']
); // 'a,b\n"1","2"\n"3","4"\n"6",""\n"","7"'
JSONtoCSV(
  [{ a: 1, b: 2 }, { a: 3, b: 4, c: 5 }, { a: 6 }, { b: 7 }],
  ['a', 'b'],
  ';'
); // 'a;b\n"1";"2"\n"3";"4"\n"6";""\n"";"7"'
```

title: RGBToHSB

Converts a RGB color tuple to HSB format.

- Use the RGB to HSB conversion formula to convert to the appropriate format.
- The range of all input parameters is [0, 255].
- The range of the resulting values is H: [0, 360], S: [0, 100], B: [0, 100].

```js
const RGBToHSB = (r, g, b) => {
  r /= 255;
  g /= 255;
  b /= 255;
  const v = Math.max(r, g, b),
    n = v - Math.min(r, g, b);
  const h =
    n === 0 ? 0 : n && v === r ? (g - b) / n : v === g ? 2 + (b - r) / n : 4 + (r - g) / n;
  return [60 * (h < 0 ? h + 6 : h), v && (n / v) * 100, v * 100];
};


RGBToHSB(252, 111, 48);
// [18.529411764705856, 80.95238095238095, 98.82352941176471]
```

title: RGBToHSL

Converts a RGB color tuple to HSL format.

- Use the RGB to HSL conversion formula to convert to the appropriate format.
- The range of all input parameters is [0, 255].
- The range of the resulting values is H: [0, 360], S: [0, 100], L: [0, 100].

```js
const RGBToHSL = (r, g, b) => {
  r /= 255;
  g /= 255;
  b /= 255;
  const l = Math.max(r, g, b);
  const s = l - Math.min(r, g, b);
  const h = s
    ? l === r
      ? (g - b) / s
      : l === g
      ? 2 + (b - r) / s
      : 4 + (r - g) / s
    : 0;
  return [
    60 * h < 0 ? 60 * h + 360 : 60 * h,
    100 * (s ? (l <= 0.5 ? s / (2 * l - s) : s / (2 - (2 * l - s))) : 0),
    (100 * (2 * l - s)) / 2,
  ];
};
```

```
RGBToHSL(45, 23, 11); // [21.17647, 60.71428, 10.98039]
```

title: RGBToHex

Converts the values of RGB components to a hexadecimal color code.

- Convert given RGB parameters to hexadecimal string using bitwise left-shift operator (`<<`) and `Number.prototype.toString(16)` .
- Use `String.prototype.padStart(6, '0')` to get a 6-digit hexadecimal value.

```
const RGBToHex = (r, g, b) =>
  ((r << 16) + (g << 8) + b).toString(16).padStart(6, '0');
```

```
RGBToHex(255, 165, 1); // 'ffa501'
```

title: URLJoin

Joins all given URL segments together, then normalizes the resulting URL.

- Use `String.prototype.join('/')` to combine URL segments.
- Use a series of `String.prototype.replace()` calls with various regexps to normalize the resulting URL (remove double slashes, add proper slashes for protocol, remove slashes before parameters, combine parameters with `'&'` and normalize first parameter delimiter).

```
const URLJoin = (...args) =>
  args
    .join('/')
    .replace(/[\/]+/g, '/')
    .replace(/^(.+):\//, '$1://')
    .replace(/^file:/, 'file:/')
    .replace(/\/(\?|&|#[^!])/g, '$1')
    .replace(/\?/g, '&')
    .replace('&', '?');
```

```
URLJoin('http://www.google.com', 'a', '/b/cd', '?foo=123', '?bar=foo');
// 'http://www.google.com/a/b/cd?foo=123&bar=foo'
```

title: UUIDGeneratorBrowser

Generates a UUID in a browser.

- Use `Crypto.getRandomValues()` to generate a UUID, compliant with RFC4122 version 4.
- Use `Number.prototype.toString(16)` to convert it to a proper UUID.

```
const UUIDGeneratorBrowser = () =>
  ([1e7] + -1e3 + -4e3 + -8e3 + -1e11).replace(/[018]/g, c =>
    (
      c ^
      (crypto.getRandomValues(new Uint8Array(1))[0] & (15 >> (c / 4)))
    ).toString(16)
  );
```

```
UUIDGeneratorBrowser(); // '7982fcfe-5721-4632-bede-6000885be57d'
```

title: UUIDGeneratorNode

Generates a UUID in Node.JS.

- Use `crypto.randomBytes()` to generate a UUID, compliant with RFC4122 version 4.
- Use `Number.prototype.toString(16)` to convert it to a proper UUID.

```
const crypto = require('crypto');

const UUIDGeneratorNode = () =>
  ([1e7] + -1e3 + -4e3 + -8e3 + -1e11).replace(/[018]/g, c =>
    (c ^ (crypto.randomBytes(1)[0] & (15 >> (c / 4)))).toString(16)
  );
```

```
UUIDGeneratorNode(); // '79c7c136-60ee-40a2-beb2-856f1feabefc'
```

title: accumulate

Creates an array of partial sums.

- Use `Array.prototype.reduce()` , initialized with an empty array accumulator to iterate over `nums` .
- Use `Array.prototype.slice(-1)` , the spread operator (`...`) and the unary `+` operator to add each value to the accumulator array containing the previous sums.

```
const accumulate = (...nums) =>
  nums.reduce((acc, n) => [...acc, n + +acc.slice(-1)], []);


accumulate(1, 2, 3, 4); // [1, 3, 6, 10]
accumulate(...[1, 2, 3, 4]); // [1, 3, 6, 10]
```

title: addClass

Adds a class to an HTML element.

- Use `Element.classList` and `DOMTokenList.add()` to add the specified class to the element.

```
const addClass = (el, className) => el.classList.add(className);


addClass(document.querySelector('p'), 'special');
// The paragraph will now have the 'special' class
```

title: addDaysToDate

Calculates the date of `n` days from the given date, returning its string representation.

- Use `new Date()` to create a date object from the first argument.
- Use `Date.prototype.getDate()` and `Date.prototype.setDate()` to add `n` days to the given date.
- Use `Date.prototype.toISOString()` to return a string in `yyyy-mm-dd` format.

```
const addDaysToDate = (date, n) => {
  const d = new Date(date);
  d.setDate(d.getDate() + n);
  return d.toISOString().split('T')[0];
};
```

```js
addDaysToDate('2020-10-15', 10); // '2020-10-25'
addDaysToDate('2020-10-15', -10); // '2020-10-05'
```

title: addEventListenerAll

Attaches an event listener to all the provided targets.

- Use `Array.prototype.forEach()` and `EventTarget.addEventListener()` to attach the provided `listener` for the given event `type` to all `targets` .

```js
const addEventListenerAll = (targets, type, listener, options, useCapture) => {
  targets.forEach(target =>
    target.addEventListener(type, listener, options, useCapture)
  );
};
```

```js
addEventListenerAll(document.querySelectorAll('a'), 'click', () =>
  console.log('Clicked a link')
);
// Logs 'Clicked a link' whenever any anchor element is clicked
```

title: addMinutesToDate

Calculates the date of `n` minutes from the given date, returning its string representation.

- Use `new Date()` to create a date object from the first argument.
- Use `Date.prototype.getTime()` and `Date.prototype.setTime()` to add `n` minutes to the given date.
- Use `Date.prototype.toISOString()` , `String.prototype.split()` and `String.prototype.replace()` to return a string in `yyyy-mm-dd HH:MM:SS` format.

```js
const addMinutesToDate = (date, n) => {
  const d = new Date(date);
  d.setTime(d.getTime() + n * 60000);
  return d.toISOString().split('.')[0].replace('T',' ');
};
```

```
addMinutesToDate('2020-10-19 12:00:00', 10); // '2020-10-19 12:10:00'
addMinutesToDate('2020-10-19', -10); // '2020-10-18 23:50:00'
```

title: addMultipleListeners

Adds multiple event listeners with the same handler to an element.

- Use `Array.prototype.forEach()` and `EventTarget.addEventListener()` to add multiple event listeners with an assigned callback function to an element.

```
const addMultipleListeners = (el, types, listener, options, useCapture) => {
  types.forEach(type =>
    el.addEventListener(type, listener, options, useCapture)
  );
};


addMultipleListeners(
  document.querySelector('.my-element'),
  ['click', 'mousedown'],
  () => { console.log('hello!') }
);
```

title: addStyles

Adds the provided styles to the given element.

- Use `Object.assign()` and `ElementCSSInlineStyle.style` to merge the provided `styles` object into the style of the given element.

```
const addStyles = (el, styles) => Object.assign(el.style, styles);


addStyles(document.getElementById('my-element'), {
  background: 'red',
  color: '#ffff00',
  fontSize: '3rem'
});
```

title: addWeekDays

Calculates the date after adding the given number of business days.

- Use `Array.from()` to construct an array with `length` equal to the `count` of business days to be added.
- Use `Array.prototype.reduce()` to iterate over the array, starting from `startDate` and incrementing, using `Date.prototype.getDate()` and `Date.prototype.setDate()` .
- If the current `date` is on a weekend, update it again by adding either one day or two days to make it a weekday.
- **NOTE:** Does not take official holidays into account.

```
const addWeekDays = (startDate, count) =>
  Array.from({ length: count }).reduce(date => {
    date = new Date(date.setDate(date.getDate() + 1));
    if (date.getDay() % 6 === 0)
      date = new Date(date.setDate(date.getDate() + (date.getDay() / 6 + 1)));
    return date;
  }, startDate);
```

```
addWeekDays(new Date('Oct 09, 2020'), 5); // 'Oct 16, 2020'
addWeekDays(new Date('Oct 12, 2020'), 5); // 'Oct 19, 2020'
```

title: all

Checks if the provided predicate function returns `true` for all elements in a collection.

- Use `Array.prototype.every()` to test if all elements in the collection return `true` based on `fn` .
- Omit the second argument, `fn` , to use `Boolean` as a default.

```
const all = (arr, fn = Boolean) => arr.every(fn);
```

```
all([4, 2, 3], x => x > 1); // true
all([1, 2, 3]); // true
```

title: allEqual

Checks if all elements in an array are equal.

- Use `Array.prototype.every()` to check if all the elements of the array are the same as the first one.
- Elements in the array are compared using the strict comparison operator, which does not account for `NaN` self-inequality.

```js
const allEqual = arr => arr.every(val => val === arr[0]);
```

```js
allEqual([1, 2, 3, 4, 5, 6]); // false
allEqual([1, 1, 1, 1]); // true
```

title: allEqualBy

Checks if all elements in an array are equal, based on the provided mapping function.

- Apply `fn` to the first element of `arr` .
- Use `Array.prototype.every()` to check if `fn` returns the same value for all elements in the array as it did for the first one.
- Elements in the array are compared using the strict comparison operator, which does not account for `NaN` self-inequality.

```js
const allEqualBy = (arr, fn) => {
  const eql = fn(arr[0]);
  return arr.every(val => fn(val) === eql);
};
```

```js
allEqualBy([1.1, 1.2, 1.3], Math.round); // true
allEqualBy([1.1, 1.3, 1.6], Math.round); // false
```

title: allUnique

Checks if all elements in an array are unique.

- Create a new `Set` from the mapped values to keep only unique occurrences.
- Use `Array.prototype.length` and `Set.prototype.size` to compare the length of the unique values to the original array.

```
const allUnique = arr => arr.length === new Set(arr).size;
```

```
allUnique([1, 2, 3, 4]); // true
allUnique([1, 1, 2, 3]); // false
```

title: allUniqueBy

Checks if all elements in an array are unique, based on the provided mapping function.

- Use `Array.prototype.map()` to apply `fn` to all elements in `arr` .
- Create a new `Set` from the mapped values to keep only unique occurrences.
- Use `Array.prototype.length` and `Set.prototype.size` to compare the length of the unique mapped values to the original array.

```
const allUniqueBy = (arr, fn) => arr.length === new Set(arr.map(fn)).size;
```

```
allUniqueBy([1.2, 2.4, 2.9], Math.round); // true
allUniqueBy([1.2, 2.3, 2.4], Math.round); // false
```

title: and
unlisted: true

Checks if both arguments are `true` .

- Use the logical and (`&&`) operator on the two given values.

```
const and = (a, b) => a && b;
```

```
and(true, true); // true
and(true, false); // false
and(false, false); // false
```

title: any

Checks if the provided predicate function returns `true` for at least one element in a collection.

- Use `Array.prototype.some()` to test if any elements in the collection return `true` based on `fn` .
- Omit the second argument, `fn` , to use `Boolean` as a default.

```
const any = (arr, fn = Boolean) => arr.some(fn);
```

```
any([0, 1, 2, 0], x => x >= 2); // true
any([0, 0, 1, 0]); // true
```

title: aperture

Creates an array of `n` -tuples of consecutive elements.

- Use `Array.prototype.slice()` and `Array.prototype.map()` to create an array of appropriate length.
- Populate the array with `n` -tuples of consecutive elements from `arr` .
- If `n` is greater than the length of `arr` , return an empty array.

```
const aperture = (n, arr) =>
  n > arr.length
    ? []
    : arr.slice(n - 1).map((v, i) => arr.slice(i, i + n));
```

```
aperture(2, [1, 2, 3, 4]); // [[1, 2], [2, 3], [3, 4]]
aperture(3, [1, 2, 3, 4]); // [[1, 2, 3], [2, 3, 4]]
aperture(5, [1, 2, 3, 4]); // []
```

title: approximatelyEqual

Checks if two numbers are approximately equal to each other.

- Use `Math.abs()` to compare the absolute difference of the two values to `epsilon`.
- Omit the third argument, `epsilon`, to use a default value of `0.001`.

```
const approximatelyEqual = (v1, v2, epsilon = 0.001) =>
  Math.abs(v1 - v2) < epsilon;
```

```
approximatelyEqual(Math.PI / 2.0, 1.5708); // true
```

title: arithmeticProgression

Creates an array of numbers in the arithmetic progression, starting with the given positive integer and up to the specified limit.

- Use `Array.from()` to create an array of the desired length, `lim/n`. Use a map function to fill it with the desired values in the given range.

```
const arithmeticProgression  = (n, lim) =>
  Array.from({ length: Math.ceil(lim / n) }, (_, i) => (i + 1) * n );
```

```
arithmeticProgression(5, 25); // [5, 10, 15, 20, 25]
```

title: arrayToCSV

Converts a 2D array to a comma-separated values (CSV) string.

- Use `Array.prototype.map()` and `Array.prototype.join(delimiter)` to combine individual 1D arrays (rows) into strings.
- Use `Array.prototype.join('\n')` to combine all rows into a CSV string, separating each row with a newline.
- Omit the second argument, `delimiter`, to use a default delimiter of `,`.

```
const arrayToCSV = (arr, delimiter = ',') =>
  arr
    .map(v =>
      v.map(x => (isNaN(x) ? `"${x.replace(/"/g, '""')}"` : x)).join(delimiter)
    )
    .join('\n');
```

```
arrayToCSV([['a', 'b'], ['c', 'd']]); // '"a","b"\n"c","d"'
arrayToCSV([['a', 'b'], ['c', 'd']], ';'); // '"a";"b"\n"c";"d"'
arrayToCSV([['a', '"b" great'], ['c', 3.1415]]);
// '"a","""b"" great"\n"c",3.1415'
```

title: arrayToHTMLList

Converts the given array elements into `<li>` tags and appends them to the list of the given id.

- Use `Array.prototype.map()` and `Document.querySelector()` to create a list of html tags.

```
const arrayToHTMLList = (arr, listID) =>
  document.querySelector(`#${listID}`).innerHTML += arr
    .map(item => `<li>${item}</li>`)
    .join('');
```

```
arrayToHTMLList(['item 1', 'item 2'], 'myListID');
```

title: ary

Creates a function that accepts up to `n` arguments, ignoring any additional arguments.

- Call the provided function, `fn` , with up to `n` arguments, using `Array.prototype.slice(0, n)` and the spread operator (`...`).

```
const ary = (fn, n) => (...args) => fn(...args.slice(0, n));
```

```
const firstTwoMax = ary(Math.max, 2);
[[2, 6, 'a'], [6, 4, 8], [10]].map(x => firstTwoMax(...x)); // [6, 6, 10]
```

title: assertValidKeys

Validates all keys in an object match the given `keys`.

- Use `Object.keys()` to get the keys of the given object, `obj`.
- Use `Array.prototype.every()` and `Array.prototype.includes()` to validate that each key in the object is specified in the `keys` array.

```
const assertValidKeys = (obj, keys) =>
  Object.keys(obj).every(key => keys.includes(key));
```

```
assertValidKeys({ id: 10, name: 'apple' }, ['id', 'name']); // true
assertValidKeys({ id: 10, name: 'apple' }, ['id', 'type']); // false
```

title: atob

Decodes a string of data which has been encoded using base-64 encoding.

- Create a `Buffer` for the given string with base-64 encoding and use `Buffer.toString('binary')` to return the decoded string.

```
const atob = str => Buffer.from(str, 'base64').toString('binary');
```

```
atob('Zm9vYmFy'); // 'foobar'
```

title: attempt

Attempts to invoke a function with the provided arguments, returning either the result or the caught error object.

- Use a `try... catch` block to return either the result of the function or an appropriate error.
- If the caught object is not an `Error`, use it to create a new `Error`.

```
const attempt = (fn, ...args) => {
  try {
    return fn(...args);
  } catch (e) {
    return e instanceof Error ? e : new Error(e);
  }
};
```

```
let elements = attempt(function(selector) {
  return document.querySelectorAll(selector);
}, '>_>');
if (elements instanceof Error) elements = []; // elements = []
```

title: average

Calculates the average of two or more numbers.

- Use `Array.prototype.reduce()` to add each value to an accumulator, initialized with a value of `0`.
- Divide the resulting array by its length.

```
const average = (...nums) =>
  nums.reduce((acc, val) => acc + val, 0) / nums.length;
```

```
average(...[1, 2, 3]); // 2
average(1, 2, 3); // 2
```

title: averageBy

Calculates the average of an array, after mapping each element to a value using the provided function.

- Use `Array.prototype.map()` to map each element to the value returned by `fn`.
- Use `Array.prototype.reduce()` to add each value to an accumulator, initialized with a value of `0`.
- Divide the resulting array by its length.

```js
const averageBy = (arr, fn) =>
  arr
    .map(typeof fn === 'function' ? fn : val => val[fn])
    .reduce((acc, val) => acc + val, 0) / arr.length;
```

```js
averageBy([{ n: 4 }, { n: 2 }, { n: 8 }, { n: 6 }], o => o.n); // 5
averageBy([{ n: 4 }, { n: 2 }, { n: 8 }, { n: 6 }], 'n'); // 5
```

title: bifurcate

Splits values into two groups, based on the result of the given `filter` array.

- Use `Array.prototype.reduce()` and `Array.prototype.push()` to add elements to groups, based on `filter`.
- If `filter` has a truthy value for any element, add it to the first group, otherwise add it to the second group.

```js
const bifurcate = (arr, filter) =>
  arr.reduce((acc, val, i) => (acc[filter[i] ? 0 : 1].push(val), acc), [
    [],
    [],
  ]);
```

```js
bifurcate(['beep', 'boop', 'foo', 'bar'], [true, true, false, true]);
// [ ['beep', 'boop', 'bar'], ['foo'] ]
```

title: bifurcateBy

Splits values into two groups, based on the result of the given filtering function.

- Use `Array.prototype.reduce()` and `Array.prototype.push()` to add elements to groups, based on the value returned by `fn` for each element.
- If `fn` returns a truthy value for any element, add it to the first group, otherwise add it to the second group.

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


bifurcateBy(['beep', 'boop', 'foo', 'bar'], x => x[0] === 'b');
// [ ['beep', 'boop', 'bar'], ['foo'] ]
```

title: binary

Creates a function that accepts up to two arguments, ignoring any additional arguments.

- Call the provided function, `fn` , with the first two arguments given.

```
const binary = fn => (a, b) => fn(a, b);


['2', '1', '0'].map(binary(Math.max)); // [2, 1, 2]
```

title: binarySearch

Finds the index of a given element in a sorted array using the binary search algorithm.

- Declare the left and right search boundaries, `l` and `r` , initialized to `0` and the `length` of the array respectively.
- Use a `while` loop to repeatedly narrow down the search subarray, using `Math.floor()` to cut it in half.
- Return the index of the element if found, otherwise return `-1` .
- **Note:** Does not account for duplicate values in the array.

```
const binarySearch = (arr, item) => {
  let l = 0,
    r = arr.length - 1;
  while (l <= r) {
    const mid = Math.floor((l + r) / 2);
    const guess = arr[mid];
    if (guess === item) return mid;
    if (guess > item) r = mid - 1;
    else l = mid + 1;
  }
  return -1;
};


binarySearch([1, 2, 3, 4, 5], 1); // 0
binarySearch([1, 2, 3, 4, 5], 5); // 4
binarySearch([1, 2, 3, 4, 5], 6); // -1
```

title: bind

Creates a function that invokes `fn` with a given context, optionally prepending any additional supplied parameters to the arguments.

- Return a `function` that uses `Function.prototype.apply()` to apply the given `context` to `fn` .
- Use the spread operator (`...`) to prepend any additional supplied parameters to the arguments.

```
const bind = (fn, context, ...boundArgs) => (...args) =>
  fn.apply(context, [...boundArgs, ...args]);


function greet(greeting, punctuation) {
  return greeting + ' ' + this.user + punctuation;
}
const freddy = { user: 'fred' };
const freddyBound = bind(greet, freddy);
console.log(freddyBound('hi', '!')); // 'hi fred!'
```

title: bindAll

Binds methods of an object to the object itself, overwriting the existing method.

- Use `Array.prototype.forEach()` to iterate over the given `fns` .
- Return a function for each one, using `Function.prototype.apply()` to apply the given context (`obj`) to `fn` .

```
const bindAll = (obj, ...fns) =>
  fns.forEach(
    fn => (
      (f = obj[fn]),
      (obj[fn] = function() {
        return f.apply(obj);
      })
    )
  );
```

```
let view = {
  label: 'docs',
  click: function() {
    console.log('clicked ' + this.label);
  }
};
bindAll(view, 'click');
document.body.addEventListener('click', view.click);
// Log 'clicked docs' when clicked.
```

title: bindKey

Creates a function that invokes the method at a given key of an object, optionally prepending any additional supplied parameters to the arguments.

- Return a `function` that uses `Function.prototype.apply()` to bind `context[fn]` to `context` .
- Use the spread operator (`...`) to prepend any additional supplied parameters to the arguments.

```
const bindKey = (context, fn, ...boundArgs) => (...args) =>
  context[fn].apply(context, [...boundArgs, ...args]);
```

```
const freddy = {
  user: 'fred',
  greet: function(greeting, punctuation) {
    return greeting + ' ' + this.user + punctuation;
  }
};
const freddyBound = bindKey(freddy, 'greet');
console.log(freddyBound('hi', '!')); // 'hi fred!'
```

title: binomialCoefficient

Calculates the number of ways to choose `k` items from `n` items without repetition and without order.

- Use `Number.isNaN()` to check if any of the two values is `NaN` .
- Check if `k` is less than `0` , greater than or equal to `n` , equal to `1` or `n - 1` and return the appropriate result.
- Check if `n - k` is less than `k` and switch their values accordingly.
- Loop from `2` through `k` and calculate the binomial coefficient.
- Use `Math.round()` to account for rounding errors in the calculation.

```
const binomialCoefficient = (n, k) => {
  if (Number.isNaN(n) || Number.isNaN(k)) return NaN;
  if (k < 0 || k > n) return 0;
  if (k === 0 || k === n) return 1;
  if (k === 1 || k === n - 1) return n;
  if (n - k < k) k = n - k;
  let res = n;
  for (let j = 2; j <= k; j++) res *= (n - j + 1) / j;
  return Math.round(res);
};
```

```
binomialCoefficient(8, 2); // 28
```

title: both
unlisted: true

Checks if both of the given functions return `true` for a given set of arguments.

- Use the logical and (`&&`) operator on the result of calling the two functions with the supplied `args` .

```
const both = (f, g) => (...args) => f(...args) && g(...args);


const isEven = num => num % 2 === 0;
const isPositive = num => num > 0;
const isPositiveEven = both(isEven, isPositive);
isPositiveEven(4); // true
isPositiveEven(-2); // false
```

Widgets

chrome-extension://gppongmhjkpfnbhagpmjfkannfbllamg/images/icons/Moat.svg

chrome-extension://gppongmhjkpfnbhagpmjfkannfbllamg/images/icons/Facebook.svg

Facebook](https://www.wappalyzer.com/technologies/widgets/facebook/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

AddThis](https://www.wappalyzer.com/technologies/widgets/addthis/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

Analytics

[

Moat](https://www.wappalyzer.com/technologies/analytics/moat/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

Google Analytics](https://www.wappalyzer.com/technologies/analytics/google-analytics/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

Google Ads Conversion Tracking](https://www.wappalyzer.com/technologies/analytics/google-ads-conversion-tracking/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

JavaScript frameworks

[

React](https://www.wappalyzer.com/technologies/javascript-frameworks/react/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

Gatsby2.25.4](https://www.wappalyzer.com/technologies/javascript-frameworks/gatsby/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

Font scripts

[

Google Font API](https://www.wappalyzer.com/technologies/font-scripts/google-font-api/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

Miscellaneous

[

webpack](https://www.wappalyzer.com/technologies/miscellaneous/webpack/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

Prism](https://www.wappalyzer.com/technologies/miscellaneous/prism/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

CDN

[

Unpkg](https://www.wappalyzer.com/technologies/cdn/unpkg/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

jsDelivr](https://www.wappalyzer.com/technologies/cdn/jsdelivr/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

jQuery CDN](https://www.wappalyzer.com/technologies/cdn/jquery-cdn/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

Netlify](https://www.wappalyzer.com/technologies/cdn/netlify/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

Marketing automation

[

MailChimp](https://www.wappalyzer.com/technologies/marketing-automation/mailchimp/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

Advertising

[

Google AdSense](https://www.wappalyzer.com/technologies/advertising/google-adsense/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

Tag managers

[

Google Tag Manager](https://www.wappalyzer.com/technologies/tag-managers/google-tag-manager/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

Live chat

[

Smartsupp1](https://www.wappalyzer.com/technologies/live-chat/smartsupp/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

LiveChat](https://www.wappalyzer.com/technologies/live-chat/livechat/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

Static site generators

[

Gatsby2.25.4](https://www.wappalyzer.com/technologies/static-site-generator/gatsby/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

JavaScript libraries

[

Lodash4.17.11](https://www.wappalyzer.com/technologies/javascript-libraries/lodash/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

Dojo1](https://www.wappalyzer.com/technologies/javascript-libraries/dojo/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

core-js3.10.2](https://www.wappalyzer.com/technologies/javascript-libraries/core-js/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

[

jQuery3.1.1](https://www.wappalyzer.com/technologies/javascript-libraries/jquery/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

PaaS

[

Netlify](https://www.wappalyzer.com/technologies/paas/netlify/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

UI frameworks

[

Bootstrap5.1.1](https://www.wappalyzer.com/technologies/ui-frameworks/bootstrap/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

Authentication

[

Facebook Login](https://www.wappalyzer.com/technologies/authentication/facebook-login/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

Email

[

MailChimp](https://www.wappalyzer.com/technologies/email/mailchimp/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

Retargeting

[

Google Remarketing Tag](https://www.wappalyzer.com/technologies/retargeting/google-remarketing-tag/?utm_source=popup&utm_medium=extension&utm_campaign=wappalyzer)

title: btoa

Creates a base-64 encoded ASCII string from a String object in which each character in the string is treated as a byte of binary data.

- Create a `Buffer` for the given string with binary encoding and use `Buffer.toString('base64')` to return the encoded string.

```
const btoa = str => Buffer.from(str, 'binary').toString('base64');
```

```
btoa('foobar'); // 'Zm9vYmFy'
```

title: bubbleSort

Sorts an array of numbers, using the bubble sort algorithm.

- Declare a variable, `swapped` , that indicates if any values were swapped during the current iteration.
- Use the spread operator (`...`) to clone the original array, `arr` .
- Use a `for` loop to iterate over the elements of the cloned array, terminating before the last element.
- Use a nested `for` loop to iterate over the segment of the array between `0` and `i` , swapping any adjacent out of order elements and setting `swapped` to `true` .
- If `swapped` is `false` after an iteration, no more changes are needed, so the cloned array is returned.

```js
const bubbleSort = arr => {
  let swapped = false;
  const a = [...arr];
  for (let i = 1; i < a.length; i++) {
    swapped = false;
    for (let j = 0; j < a.length - i; j++) {
      if (a[j + 1] < a[j]) {
        [a[j], a[j + 1]] = [a[j + 1], a[j]];
        swapped = true;
      }
    }
    if (!swapped) return a;
  }
  return a;
};


bubbleSort([2, 1, 4, 3]); // [1, 2, 3, 4]
```

title: bucketSort

Sorts an array of numbers, using the bucket sort algorithm.

- Use `Math.min()`, `Math.max()` and the spread operator (`...`) to find the minimum and maximum values of the given array.
- Use `Array.from()` and `Math.floor()` to create the appropriate number of `buckets` (empty arrays).
- Use `Array.prototype.forEach()` to populate each bucket with the appropriate elements from the array.
- Use `Array.prototype.reduce()` , the spread operator (`...`) and `Array.prototype.sort()` to sort each bucket and append it to the result.

```js
const bucketSort = (arr, size = 5) => {
  const min = Math.min(...arr);
  const max = Math.max(...arr);
  const buckets = Array.from(
    { length: Math.floor((max - min) / size) + 1 },
    () => []
  );
  arr.forEach(val => {
    buckets[Math.floor((val - min) / size)].push(val);
  });
  return buckets.reduce((acc, b) => [...acc, ...b.sort((a, b) => a - b)], []);
};


bucketSort([6, 3, 4, 1]); // [1, 3, 4, 6]
```

title: byteSize

Returns the length of a string in bytes.

- Convert a given string to a `Blob` Object.
- Use `Blob.size` to get the length of the string in bytes.

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


byteSize('😀'); // 4
byteSize('Hello World'); // 11
```

title: caesarCipher

Encrypts or decrypts a given string using the Caesar cipher.

- Use the modulo (`%`) operator and the ternary operator (`?`) to calculate the correct encryption/decryption key.
- Use the spread operator (`...`) and `Array.prototype.map()` to iterate over the letters of the given string.
- Use `String.prototype.charCodeAt()` and `String.fromCharCode()` to convert each letter appropriately, ignoring special characters, spaces etc.

- Use `Array.prototype.join()` to combine all the letters into a string.
- Pass `true` to the last parameter, `decrypt` , to decrypt an encrypted string.

```js
const caesarCipher = (str, shift, decrypt = false) => {
  const s = decrypt ? (26 - shift) % 26 : shift;
  const n = s > 0 ? s : 26 + (s % 26);
  return [...str]
    .map((l, i) => {
      const c = str.charCodeAt(i);
      if (c >= 65 && c <= 90)
        return String.fromCharCode(((c - 65 + n) % 26) + 65);
      if (c >= 97 && c <= 122)
        return String.fromCharCode(((c - 97 + n) % 26) + 97);
      return l;
    })
    .join('');
};
```

```js
caesarCipher('Hello World!', -3); // 'Ebiil Tloia!'
caesarCipher('Ebiil Tloia!', 23, true); // 'Hello World!'
```

title: call

Given a key and a set of arguments, call them when given a context.

- Use a closure to call `key` with `args` for the given `context` .

```js
const call = (key, ...args) => context => context[key](...args);
```

```js
Promise.resolve([1, 2, 3])
  .then(call('map', x => 2 * x))
  .then(console.log); // [ 2, 4, 6 ]
const map = call.bind(null, 'map');
Promise.resolve([1, 2, 3])
  .then(map(x => 2 * x))
  .then(console.log); // [ 2, 4, 6 ]
```

title: capitalize

Capitalizes the first letter of a string.

- Use array destructuring and `String.prototype.toUpperCase()` to capitalize the first letter of the string.
- Use `Array.prototype.join('')` to combine the capitalized `first` with the `...rest` of the characters.
- Omit the `lowerRest` argument to keep the rest of the string intact, or set it to `true` to convert to lowercase.

```
const capitalize = ([first, ...rest], lowerRest = false) =>
  first.toUpperCase() +
  (lowerRest ? rest.join('').toLowerCase() : rest.join(''));
```

```
capitalize('fooBar'); // 'FooBar'
capitalize('fooBar', true); // 'Foobar'
```

title: capitalizeEveryWord

Capitalizes the first letter of every word in a string.

- Use `String.prototype.replace()` to match the first character of each word and `String.prototype.toUpperCase()` to capitalize it.

```
const capitalizeEveryWord = str =>
  str.replace(/\b[a-z]/g, char => char.toUpperCase());
```

```
capitalizeEveryWord('hello world!'); // 'Hello World!'
```

title: cartesianProduct

Calculates the cartesian product of two arrays.

- Use `Array.prototype.reduce()` , `Array.prototype.map()` and the spread operator (`...`) to generate all possible element pairs from the two arrays.

```
const cartesianProduct = (a, b) =>
  a.reduce((p, x) => [...p, ...b.map(y => [x, y])], []);
```

```
cartesianProduct(['x', 'y'], [1, 2]);
// [['x', 1], ['x', 2], ['y', 1], ['y', 2]]
```

title: castArray

Casts the provided value as an array if it's not one.

- Use `Array.prototype.isArray()` to determine if `val` is an array and return it as-is or encapsulated in an array accordingly.

```
const castArray = val => (Array.isArray(val) ? val : [val]);
```

```
castArray('foo'); // ['foo']
castArray([1]); // [1]
```

title: celsiusToFahrenheit unlisted: true

Converts Celsius to Fahrenheit.

- Follow the conversion formula `F = 1.8 * C + 32`.

```
const celsiusToFahrenheit = degrees => 1.8 * degrees + 32;
```

```
celsiusToFahrenheit(33); // 91.4
```

title: chainAsync

Chains asynchronous functions.

- Loop through an array of functions containing asynchronous events, calling `next` when each asynchronous event has completed.

```
const chainAsync = fns => {
  let curr = 0;
  const last = fns[fns.length - 1];
  const next = () => {
    const fn = fns[curr++];
    fn === last ? fn() : fn(next);
  };
  next();
};


chainAsync([
  next => {
    console.log('0 seconds');
    setTimeout(next, 1000);
  },
  next => {
    console.log('1 second');
    setTimeout(next, 1000);
  },
  () => {
    console.log('2 second');
  }
]);
```

title: changeLightness

Changes the lightness value of an `hsl()` color string.

- Use `String.prototype.match()` to get an array of 3 strings with the numeric values.
- Use `Array.prototype.map()` in combination with `Number` to convert them into an array of numeric values.
- Make sure the lightness is within the valid range (between `0` and `100`), using `Math.max()` and `Math.min()`.
- Use a template literal to create a new `hsl()` string with the updated value.

```js
const changeLightness = (delta, hslStr) => {
  const [hue, saturation, lightness] = hslStr.match(/\d+/g).map(Number);

  const newLightness = Math.max(
    0,
    Math.min(100, lightness + parseFloat(delta))
  );

  return `hsl(${hue}, ${saturation}%, ${newLightness}%)`;
};


changeLightness(10, 'hsl(330, 50%, 50%)'); // 'hsl(330, 50%, 60%)'
changeLightness(-10, 'hsl(330, 50%, 50%)'); // 'hsl(330, 50%, 40%)'
```

title: checkProp

Creates a function that will invoke a predicate function for the specified property on a given object.

- Return a curried function, that will invoke `predicate` for the specified `prop` on `obj` and return a boolean.

```js
const checkProp = (predicate, prop) => obj => !!predicate(obj[prop]);


const lengthIs4 = checkProp(l => l === 4, 'length');
lengthIs4([]); // false
lengthIs4([1, 2, 3, 4]); // true
lengthIs4(new Set([1, 2, 3, 4])); // false (Set uses Size, not length)

const session = { user: {} };
const validUserSession = checkProp(u => u.active && !u.disabled, 'user');

validUserSession(session); // false

session.user.active = true;
validUserSession(session); // true

const noLength = checkProp(l => l === undefined, 'length');
noLength([]); // false
noLength({}); // true
noLength(new Set()); // true
```

title: chunk

Chunks an array into smaller arrays of a specified size.

- Use `Array.from()` to create a new array, that fits the number of chunks that will be produced.
- Use `Array.prototype.slice()` to map each element of the new array to a chunk the length of `size` .
- If the original array can't be split evenly, the final chunk will contain the remaining elements.

```
const chunk = (arr, size) =>
  Array.from({ length: Math.ceil(arr.length / size) }, (v, i) =>
    arr.slice(i * size, i * size + size)
  );
```

```
chunk([1, 2, 3, 4, 5], 2); // [[1, 2], [3, 4], [5]]
```

title: chunkIntoN

Chunks an array into `n` smaller arrays.

- Use `Math.ceil()` and `Array.prototype.length` to get the size of each chunk.
- Use `Array.from()` to create a new array of size `n` .
- Use `Array.prototype.slice()` to map each element of the new array to a chunk the length of `size` .
- If the original array can't be split evenly, the final chunk will contain the remaining elements.

```
const chunkIntoN = (arr, n) => {
  const size = Math.ceil(arr.length / n);
  return Array.from({ length: n }, (v, i) =>
    arr.slice(i * size, i * size + size)
  );
}
```

```
chunkIntoN([1, 2, 3, 4, 5, 6, 7], 4); // [[1, 2], [3, 4], [5, 6], [7]]
```

title: chunkify

Chunks an iterable into smaller arrays of a specified size.

- Use a `for...of` loop over the given iterable, using `Array.prototype.push()` to add each new value to the current `chunk` .
- Use `Array.prototype.length` to check if the current `chunk` is of the desired `size` and `yield` the value if it is.
- Finally, use `Array.prototype.length` to check the final `chunk` and `yield` it if it's non-empty.

```
const chunkify = function* (itr, size) {
  let chunk = [];
  for (const v of itr) {
    chunk.push(v);
    if (chunk.length === size) {
      yield chunk;
      chunk = [];
    }
  }
  if (chunk.length) yield chunk;
};
```

```
const x = new Set([1, 2, 1, 3, 4, 1, 2, 5]);
[...chunkify(x, 2)]; // [[1, 2], [3, 4], [5]]
```

title: clampNumber

Clamps `num` within the inclusive range specified by the boundary values `a` and `b` .

- If `num` falls within the range, return `num` .
- Otherwise, return the nearest number in the range.

```
const clampNumber = (num, a, b) =>
  Math.max(Math.min(num, Math.max(a, b)), Math.min(a, b));
```

```
clampNumber(2, 3, 5); // 3
clampNumber(1, -1, -5); // -1
```

title: cloneRegExp

Clones a regular expression.

- Use `new RegExp()` , `RegExp.prototype.source` and `RegExp.prototype.flags` to clone the given regular expression.

```
const cloneRegExp = regExp => new RegExp(regExp.source, regExp.flags);
```

```
const regExp = /lorem ipsum/gi;
const regExp2 = cloneRegExp(regExp); // regExp !== regExp2
```

title: coalesce

Returns the first defined, non-null argument.

- Use `Array.prototype.find()` and `Array.prototype.includes()` to find the first value that is not equal to `undefined` or `null` .

```
const coalesce = (...args) => args.find(v => ![undefined, null].includes(v));
```

```
coalesce(null, undefined, '', NaN, 'Waldo'); // ''
```

title: coalesceFactory

Customizes a coalesce function that returns the first argument which is true based on the given validator.

- Use `Array.prototype.find()` to return the first argument that returns `true` from the provided argument validation function, `valid` .

```
const coalesceFactory = valid => (...args) => args.find(valid);
const customCoalesce = coalesceFactory(
  v => ![null, undefined, '', NaN].includes(v)
);
customCoalesce(undefined, null, NaN, '', 'Waldo'); // 'Waldo'
```

title: collectInto

Changes a function that accepts an array into a variadic function.

- Given a function, return a closure that collects all inputs into an array-accepting function.

```
const collectInto = fn => (...args) => fn(args);
```

```
const Pall = collectInto(Promise.all.bind(Promise));
let p1 = Promise.resolve(1);
let p2 = Promise.resolve(2);
let p3 = new Promise(resolve => setTimeout(resolve, 2000, 3));
Pall(p1, p2, p3).then(console.log); // [1, 2, 3] (after about 2 seconds)
```

title: colorize

Adds special characters to text to print in color in the console (combined with `console.log()`).

- Use template literals and special characters to add the appropriate color code to the string output.
- For background colors, add a special character that resets the background color at the end of the string.

```
const colorize = (...args) => ({
  black: `\x1b[30m${args.join(' ')}`,
  red: `\x1b[31m${args.join(' ')}`,
  green: `\x1b[32m${args.join(' ')}`,
  yellow: `\x1b[33m${args.join(' ')}`,
  blue: `\x1b[34m${args.join(' ')}`,
  magenta: `\x1b[35m${args.join(' ')}`,
  cyan: `\x1b[36m${args.join(' ')}`,
  white: `\x1b[37m${args.join(' ')}`,
  bgBlack: `\x1b[40m${args.join(' ')}\x1b[0m`,
  bgRed: `\x1b[41m${args.join(' ')}\x1b[0m`,
  bgGreen: `\x1b[42m${args.join(' ')}\x1b[0m`,
  bgYellow: `\x1b[43m${args.join(' ')}\x1b[0m`,
  bgBlue: `\x1b[44m${args.join(' ')}\x1b[0m`,
  bgMagenta: `\x1b[45m${args.join(' ')}\x1b[0m`,
  bgCyan: `\x1b[46m${args.join(' ')}\x1b[0m`,
  bgWhite: `\x1b[47m${args.join(' ')}\x1b[0m`
});
```

```
console.log(colorize('foo').red); // 'foo' (red letters)
console.log(colorize('foo', 'bar').bgBlue); // 'foo bar' (blue background)
console.log(colorize(colorize('foo').yellow, colorize('foo').green).bgWhite);
// 'foo bar' (first word in yellow letters, second word in green letters, white background for b
```

title: combine

Combines two arrays of objects, using the specified key to match objects.

- Use `Array.prototype.reduce()` with an object accumulator to combine all objects in both arrays based on the given `prop` .
- Use `Object.values()` to convert the resulting object to an array and return it.

```
const combine = (a, b, prop) =>
  Object.values(
    [...a, ...b].reduce((acc, v) => {
      if (v[prop])
        acc[v[prop]] = acc[v[prop]]
          ? { ...acc[v[prop]], ...v }
          : { ...v };
      return acc;
    }, {})
  );
```

```
const x = [
  { id: 1, name: 'John' },
  { id: 2, name: 'Maria' }
];
const y = [
  { id: 1, age: 28 },
  { id: 3, age: 26 },
  { age: 3}
];
combine(x, y, 'id');
// [
//   { id: 1, name: 'John', age: 28 },
//   { id: 2, name: 'Maria' },
//   { id: 3, age: 26 }
// ]
```

title: compact

Removes falsy values from an array.

- Use `Array.prototype.filter()` to filter out falsy values (`false` , `null` , `0` , `""` , `undefined` , and `NaN`).

```
const compact = arr => arr.filter(Boolean);
```

```
compact([0, 1, false, 2, '', 3, 'a', 'e' * 23, NaN, 's', 34]);
// [ 1, 2, 3, 'a', 's', 34 ]
```

title: compactObject

Deeply removes all falsy values from an object or array.

- Use recursion.
- Initialize the iterable data, using `Array.isArray()` , `Array.prototype.filter()` and `Boolean` for arrays in order to avoid sparse arrays.
- Use `Object.keys()` and `Array.prototype.reduce()` to iterate over each key with an appropriate initial value.

- Use `Boolean` to determine the truthiness of each key's value and add it to the accumulator if it's truthy.
- Use `typeof` to determine if a given value is an `object` and call the function again to deeply compact it.

```
const compactObject = val => {
  const data = Array.isArray(val) ? val.filter(Boolean) : val;
  return Object.keys(data).reduce(
    (acc, key) => {
      const value = data[key];
      if (Boolean(value))
        acc[key] = typeof value === 'object' ? compactObject(value) : value;
      return acc;
    },
    Array.isArray(val) ? [] : {}
  );
};
```

```
const obj = {
  a: null,
  b: false,
  c: true,
  d: 0,
  e: 1,
  f: '',
  g: 'a',
  h: [null, false, '', true, 1, 'a'],
  i: { j: 0, k: false, l: 'a' }
};
compactObject(obj);
// { c: true, e: 1, g: 'a', h: [ true, 1, 'a' ], i: { l: 'a' } }
```

title: compactWhitespace

Compacts whitespaces in a string.

- Use `String.prototype.replace()` with a regular expression to replace all occurrences of 2 or more whitespace characters with a single space.

```
const compactWhitespace = str => str.replace(/\s{2,}/g, ' ');
```

```
compactWhitespace('Lorem    Ipsum'); // 'Lorem Ipsum'
compactWhitespace('Lorem \n Ipsum'); // 'Lorem Ipsum'
```

title: complement

Returns a function that is the logical complement of the given function, `fn` .

- Use the logical not (`!`) operator on the result of calling `fn` with any supplied `args` .

```
const complement = fn => (...args) => !fn(...args);
```

```
const isEven = num => num % 2 === 0;
const isOdd = complement(isEven);
isOdd(2); // false
isOdd(3); // true
```

title: compose

Performs right-to-left function composition.

- Use `Array.prototype.reduce()` to perform right-to-left function composition.
- The last (rightmost) function can accept one or more arguments; the remaining functions must be unary.

```
const compose = (...fns) =>
  fns.reduce((f, g) => (...args) => f(g(...args)));
```

```
const add5 = x => x + 5;
const multiply = (x, y) => x * y;
const multiplyAndAdd5 = compose(
  add5,
  multiply
);
multiplyAndAdd5(5, 2); // 15
```

title: composeRight

Performs left-to-right function composition.

- Use `Array.prototype.reduce()` to perform left-to-right function composition.
- The first (leftmost) function can accept one or more arguments; the remaining functions must be unary.

```
const composeRight = (...fns) =>
  fns.reduce((f, g) => (...args) => g(f(...args)));


const add = (x, y) => x + y;
const square = x => x * x;
const addAndSquare = composeRight(add, square);
addAndSquare(1, 2); // 9
```

title: containsWhitespace

Checks if the given string contains any whitespace characters.

- Use `RegExp.prototype.test()` with an appropriate regular expression to check if the given string contains any whitespace characters.

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


containsWhitespace('lorem'); // false
containsWhitespace('lorem ipsum'); // true
```

title: converge

Accepts a converging function and a list of branching functions and returns a function that applies each branching function to the arguments and the results of the branching functions are passed as arguments to the converging function.

- Use `Array.prototype.map()` and `Function.prototype.apply()` to apply each function to the given arguments.
- Use the spread operator (`...`) to call `converger` with the results of all other functions.

```js
const converge = (converger, fns) => (...args) =>
  converger(...fns.map(fn => fn.apply(null, args)));
```

```js
const average = converge((a, b) => a / b, [
  arr => arr.reduce((a, v) => a + v, 0),
  arr => arr.length
]);
average([1, 2, 3, 4, 5, 6, 7]); // 4
```

title: copySign

Returns the absolute value of the first number, but the sign of the second.

- Use `Math.sign()` to check if the two numbers have the same sign.
- Return `x` if they do, `-x` otherwise.

```js
const copySign = (x, y) => Math.sign(x) === Math.sign(y) ? x : -x;
```

```js
copySign(2, 3); // 2
copySign(2, -3); // -2
copySign(-2, 3); // 2
copySign(-2, -3); // -2
```

title: copyToClipboard

Copies a string to the clipboard.
Only works as a result of user action (i.e. inside a `click` event listener).

- Create a new `<textarea>` element, fill it with the supplied data and add it to the HTML document.
- Use `Selection.getRangeAt()` to store the selected range (if any).
- Use `Document.execCommand('copy')` to copy to the clipboard.
- Remove the `<textarea>` element from the HTML document.

- Finally, use `Selection().addRange()` to recover the original selected range (if any).
- **Note:** You can use the new asynchronous Clipboard API to implement the same functionality. It's experimental but should be used in the future instead of this snippet. Find out more about it here.

```
const copyToClipboard = str => {
  const el = document.createElement('textarea');
  el.value = str;
  el.setAttribute('readonly', '');
  el.style.position = 'absolute';
  el.style.left = '-9999px';
  document.body.appendChild(el);
  const selected =
    document.getSelection().rangeCount > 0
      ? document.getSelection().getRangeAt(0)
      : false;
  el.select();
  document.execCommand('copy');
  document.body.removeChild(el);
  if (selected) {
    document.getSelection().removeAllRanges();
    document.getSelection().addRange(selected);
  }
};
```

```
copyToClipboard('Lorem ipsum'); // 'Lorem ipsum' copied to clipboard.
```

title: countBy

Groups the elements of an array based on the given function and returns the count of elements in each group.

- Use `Array.prototype.map()` to map the values of an array to a function or property name.
- Use `Array.prototype.reduce()` to create an object, where the keys are produced from the mapped results.

```
const countBy = (arr, fn) =>
  arr.map(typeof fn === 'function' ? fn : val => val[fn]).reduce((acc, val) => {
    acc[val] = (acc[val] || 0) + 1;
    return acc;
  }, {});
```

```
countBy([6.1, 4.2, 6.3], Math.floor); // {4: 1, 6: 2}
countBy(['one', 'two', 'three'], 'length'); // {3: 2, 5: 1}
countBy([{ count: 5 }, { count: 10 }, { count: 5 }], x => x.count)
// {5: 2, 10: 1}
```

title: countOccurrences

Counts the occurrences of a value in an array.

- Use `Array.prototype.reduce()` to increment a counter each time the specific value is encountered inside the array.

```
const countOccurrences = (arr, val) =>
  arr.reduce((a, v) => (v === val ? a + 1 : a), 0);
```

```
countOccurrences([1, 1, 2, 1, 2, 3], 1); // 3
```

title: countSubstrings

Counts the occurrences of a substring in a given string.

- Use `Array.prototype.indexOf()` to look for `searchValue` in `str` .
- Increment a counter if the value is found and update the index, `i` .
- Use a `while` loop that will return as soon as the value returned from `Array.prototype.indexOf()` is `-1` .

```
const countSubstrings = (str, searchValue) => {
  let count = 0,
    i = 0;
  while (true) {
    const r = str.indexOf(searchValue, i);
    if (r !== -1) [count, i] = [count + 1, r + 1];
    else return count;
  }
};
```

```
countSubstrings('tiktok tok tok tik tok tik', 'tik'); // 3
countSubstrings('tutut tut tut', 'tut'); // 4
```

title: countWeekDaysBetween

Counts the weekdays between two dates.

- Use `Array.from()` to construct an array with `length` equal to the number of days between `startDate` and `endDate` .
- Use `Array.prototype.reduce()` to iterate over the array, checking if each date is a weekday and incrementing `count` .
- Update `startDate` with the next day each loop using `Date.prototype.getDate()` and `Date.prototype.setDate()` to advance it by one day.
- **NOTE:** Does not take official holidays into account.

```
const countWeekDaysBetween = (startDate, endDate) =>
  Array
    .from({ length: (endDate - startDate) / (1000 * 3600 * 24) })
    .reduce(count => {
      if (startDate.getDay() % 6 !== 0) count++;
      startDate = new Date(startDate.setDate(startDate.getDate() + 1));
      return count;
    }, 0);
```

```
countWeekDaysBetween(new Date('Oct 05, 2020'), new Date('Oct 06, 2020')); // 1
countWeekDaysBetween(new Date('Oct 05, 2020'), new Date('Oct 14, 2020')); // 7
```

title: counter

Creates a counter with the specified range, step and duration for the specified selector.

- Check if `step` has the proper sign and change it accordingly.
- Use `setInterval()` in combination with `Math.abs()` and `Math.floor()` to calculate the time between each new text draw.
- Use `Document.querySelector()` , `Element.innerHTML` to update the value of the selected element.
- Omit the fourth argument, `step` , to use a default step of `1` .

- Omit the fifth argument, `duration`, to use a default duration of `2000` ms.

```
const counter = (selector, start, end, step = 1, duration = 2000) => {
  let current = start,
    _step = (end - start) * step < 0 ? -step : step,
    timer = setInterval(() => {
      current += _step;
      document.querySelector(selector).innerHTML = current;
      if (current >= end) document.querySelector(selector).innerHTML = end;
      if (current >= end) clearInterval(timer);
    }, Math.abs(Math.floor(duration / (end - start))));
  return timer;
};
```

```
counter('#my-id', 1, 1000, 5, 2000);
// Creates a 2-second timer for the element with id="my-id"
```

title: createDirIfNotExists

Creates a directory, if it does not exist.

- Use `fs.existsSync()` to check if the directory exists, `fs.mkdirSync()` to create it.

```
const fs = require('fs');

const createDirIfNotExists = dir => (!fs.existsSync(dir) ? fs.mkdirSync(dir) : undefined);
```

```
createDirIfNotExists('test');
// creates the directory 'test', if it doesn't exist
```

title: createElement

Creates an element from a string (without appending it to the document).
If the given string contains multiple elements, only the first one will be returned.

- Use `Document.createElement()` to create a new element.
- Use `Element.innerHTML` to set its inner HTML to the string supplied as the argument.

- Use `ParentNode.firstElementChild` to return the element version of the string.

```
const createElement = str => {
  const el = document.createElement('div');
  el.innerHTML = str;
  return el.firstElementChild;
};


const el = createElement(
  `<div class="container">
    <p>Hello!</p>
  </div>`
);
console.log(el.className); // 'container'
```

title: createEventHub

Creates a pub/sub (publish–subscribe) event hub with `emit` , `on` , and `off` methods.

- Use `Object.create(null)` to create an empty `hub` object that does not inherit properties from `Object.prototype` .
- For `emit` , resolve the array of handlers based on the `event` argument and then run each one with `Array.prototype.forEach()` by passing in the data as an argument.
- For `on` , create an array for the event if it does not yet exist, then use `Array.prototype.push()` to add the handler
- to the array.
- For `off` , use `Array.prototype.findIndex()` to find the index of the handler in the event array and remove it using `Array.prototype.splice()` .

```
const createEventHub = () => ({
  hub: Object.create(null),
  emit(event, data) {
    (this.hub[event] || []).forEach(handler => handler(data));
  },
  on(event, handler) {
    if (!this.hub[event]) this.hub[event] = [];
    this.hub[event].push(handler);
  },
  off(event, handler) {
    const i = (this.hub[event] || []).findIndex(h => h === handler);
    if (i > -1) this.hub[event].splice(i, 1);
    if (this.hub[event].length === 0) delete this.hub[event];
  }
});
```

```
const handler = data => console.log(data);
const hub = createEventHub();
let increment = 0;

// Subscribe: listen for different types of events
hub.on('message', handler);
hub.on('message', () => console.log('Message event fired'));
hub.on('increment', () => increment++);

// Publish: emit events to invoke all handlers subscribed to them, passing the data to them as a
hub.emit('message', 'hello world'); // logs 'hello world' and 'Message event fired'
hub.emit('message', { hello: 'world' }); // logs the object and 'Message event fired'
hub.emit('increment'); // `increment` variable is now 1

// Unsubscribe: stop a specific handler from listening to the 'message' event
hub.off('message', handler);
```

title: currentURL

Returns the current URL.

- Use `Window.location.href` to get the current URL.

```
const currentURL = () => window.location.href;
```

```
currentURL(); // 'https://www.google.com/'
```

title: curry

Curries a function.

- Use recursion.
- If the number of provided arguments (`args`) is sufficient, call the passed function `fn` .
- Otherwise, use `Function.prototype.bind()` to return a curried function `fn` that expects the rest of the arguments.
- If you want to curry a function that accepts a variable number of arguments (a variadic function, e.g. `Math.min()`), you can optionally pass the number of arguments to the second parameter `arity` .

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

```
curry(Math.pow)(2)(10); // 1024
curry(Math.min, 3)(10)(50)(2); // 2
```

title: cycleGenerator

Creates a generator, looping over the given array indefinitely.

- Use a non-terminating `while` loop, that will `yield` a value every time `Generator.prototype.next()` is called.
- Use the module operator (`%`) with `Array.prototype.length` to get the next value's index and increment the counter after each `yield` statement.

```
const cycleGenerator = function* (arr) {
  let i = 0;
  while (true) {
    yield arr[i % arr.length];
    i++;
  }
};
```

```js
const binaryCycle = cycleGenerator([0, 1]);
binaryCycle.next(); // { value: 0, done: false }
binaryCycle.next(); // { value: 1, done: false }
binaryCycle.next(); // { value: 0, done: false }
binaryCycle.next(); // { value: 1, done: false }
```

title: dateRangeGenerator

Creates a generator, that generates all dates in the given range using the given step.

- Use a `while` loop to iterate from `start` to `end`, using `yield` to return each date in the range, using the `Date` constructor.
- Use `Date.prototype.getDate()` and `Date.prototype.setDate()` to increment by `step` days after returning each subsequent value.
- Omit the third argument, `step`, to use a default value of `1`.

```js
const dateRangeGenerator = function* (start, end, step = 1) {
  let d = start;
  while (d < end) {
    yield new Date(d);
    d.setDate(d.getDate() + step);
  }
};
```

```js
[...dateRangeGenerator(new Date('2021-06-01'), new Date('2021-06-04'))];
// [ 2021-06-01, 2021-06-02, 2021-06-03 ]
```

title: dayName

Gets the name of the weekday from a `Date` object.

- Use `Date.prototype.toLocaleDateString()` with the `{ weekday: 'long' }` option to retrieve the weekday.
- Use the optional second argument to get a language-specific name or omit it to use the default locale.

```
const dayName = (date, locale) =>
  date.toLocaleDateString(locale, { weekday: 'long' });


dayName(new Date()); // 'Saturday'
dayName(new Date('09/23/2020'), 'de-DE'); // 'Samstag'
```

title: dayOfYear

Gets the day of the year (number in the range 1-366) from a `Date` object.

- Use `new Date()` and `Date.prototype.getFullYear()` to get the first day of the year as a `Date` object.
- Subtract the first day of the year from `date` and divide with the milliseconds in each day to get the result.
- Use `Math.floor()` to appropriately round the resulting day count to an integer.

```
const dayOfYear = date =>
  Math.floor((date - new Date(date.getFullYear(), 0, 0)) / 1000 / 60 / 60 / 24);


dayOfYear(new Date()); // 272
```

title: daysAgo

Calculates the date of `n` days ago from today as a string representation.

- Use `new Date()` to get the current date, `Math.abs()` and `Date.prototype.getDate()` to update the date accordingly and set to the result using `Date.prototype.setDate()` .
- Use `Date.prototype.toISOString()` to return a string in `yyyy-mm-dd` format.

```
const daysAgo = n => {
  let d = new Date();
  d.setDate(d.getDate() - Math.abs(n));
  return d.toISOString().split('T')[0];
};
```

```
daysAgo(20); // 2020-09-16 (if current date is 2020-10-06)
```

title: daysFromNow

Calculates the date of `n` days from today as a string representation.

- Use `new Date()` to get the current date, `Math.abs()` and `Date.prototype.getDate()` to update the date accordingly and set to the result using `Date.prototype.setDate()`.
- Use `Date.prototype.toISOString()` to return a string in `yyyy-mm-dd` format.

```
const daysFromNow = n => {
  let d = new Date();
  d.setDate(d.getDate() + Math.abs(n));
  return d.toISOString().split('T')[0];
};
```

```
daysFromNow(5); // 2020-10-13 (if current date is 2020-10-08)
```

title: daysInMonth

Gets the number of days in the given `month` of the specified `year`.

- Use the `new Date()` constructor to create a date from the given `year` and `month`.
- Set the days parameter to `0` to get the last day of the previous month, as months are zero-indexed.
- Use `Date.prototype.getDate()` to return the number of days in the given `month`.

```
const daysInMonth = (year, month) => new Date(year, month, 0).getDate();
```

```
daysInMonth(2020, 12)); // 31
daysInMonth(2024, 2)); // 29
```

title: debounce

Creates a debounced function that delays invoking the provided function until at least `ms` milliseconds have elapsed since its last invocation.

- Each time the debounced function is invoked, clear the current pending timeout with `clearTimeout()`. Use `setTimeout()` to create a new timeout that delays invoking the function until at least `ms` milliseconds have elapsed.
- Use `Function.prototype.apply()` to apply the `this` context to the function and provide the necessary arguments.
- Omit the second argument, `ms`, to set the timeout at a default of `0` ms.

```
const debounce = (fn, ms = 0) => {
  let timeoutId;
  return function(...args) {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => fn.apply(this, args), ms);
  };
};
```

```
window.addEventListener(
  'resize',
  debounce(() => {
    console.log(window.innerWidth);
    console.log(window.innerHeight);
  }, 250)
); // Will log the window dimensions at most every 250ms
```

title: debouncePromise

Creates a debounced function that returns a promise, but delays invoking the provided function until at least `ms` milliseconds have elapsed since the last time it was invoked.
All promises returned during this time will return the same data.

- Each time the debounced function is invoked, clear the current pending timeout with `clearTimeout()` and use `setTimeout()` to create a new timeout that delays invoking the function until at least `ms` milliseconds has elapsed.
- Use `Function.prototype.apply()` to apply the `this` context to the function and provide the necessary arguments.

- Create a new `Promise` and add its `resolve` and `reject` callbacks to the `pending` promises stack.
- When `setTimeout` is called, copy the current stack (as it can change between the provided function call and its resolution), clear it and call the provided function.
- When the provided function resolves/rejects, resolve/reject all promises in the stack (copied when the function was called) with the returned data.
- Omit the second argument, `ms`, to set the timeout at a default of `0` ms.

```js
const debouncePromise = (fn, ms = 0) => {
  let timeoutId;
  const pending = [];
  return (...args) =>
    new Promise((res, rej) => {
      clearTimeout(timeoutId);
      timeoutId = setTimeout(() => {
        const currentPending = [...pending];
        pending.length = 0;
        Promise.resolve(fn.apply(this, args)).then(
          data => {
            currentPending.forEach(({ resolve }) => resolve(data));
          },
          error => {
            currentPending.forEach(({ reject }) => reject(error));
          }
        );
      }, ms);
      pending.push({ resolve: res, reject: rej });
    });
};
```

```js
const fn = arg => new Promise(resolve => {
  setTimeout(resolve, 1000, ['resolved', arg]);
});
const debounced = debouncePromise(fn, 200);
debounced('foo').then(console.log);
debounced('bar').then(console.log);
// Will log ['resolved', 'bar'] both times
```

title: decapitalize

Decapitalizes the first letter of a string.

- Use array destructuring and `String.prototype.toLowerCase()` to decapitalize first letter, `...rest` to get array of characters after first letter and then `Array.prototype.join('')` to make it a string again.
- Omit the `upperRest` argument to keep the rest of the string intact, or set it to `true` to convert to uppercase.

```js
const decapitalize = ([first, ...rest], upperRest = false) =>
  first.toLowerCase() +
  (upperRest ? rest.join('').toUpperCase() : rest.join(''));


decapitalize('FooBar'); // 'fooBar'
decapitalize('FooBar', true); // 'fOOBAR'
```

title: deepClone

Creates a deep clone of an object.
Clones primitives, arrays and objects, excluding class instances.

- Use recursion.
- Check if the passed object is `null` and, if so, return `null`.
- Use `Object.assign()` and an empty object (`{}`) to create a shallow clone of the original.
- Use `Object.keys()` and `Array.prototype.forEach()` to determine which key-value pairs need to be deep cloned.
- If the object is an `Array`, set the `clone`'s `length` to that of the original and use `Array.from(clone)` to create a clone.

```js
const deepClone = obj => {
  if (obj === null) return null;
  let clone = Object.assign({}, obj);
  Object.keys(clone).forEach(
    key =>
      (clone[key] =
        typeof obj[key] === 'object' ? deepClone(obj[key]) : obj[key])
  );
  if (Array.isArray(obj)) {
    clone.length = obj.length;
    return Array.from(clone);
  }
  return clone;
};
```

```
const a = { foo: 'bar', obj: { a: 1, b: 2 } };
const b = deepClone(a); // a !== b, a.obj !== b.obj
```

title: deepFlatten

Deep flattens an array.

- Use recursion.
- Use `Array.prototype.concat()` with an empty array (`[]`) and the spread operator (`...`) to flatten an array.
- Recursively flatten each element that is an array.

```
const deepFlatten = arr =>
  [].concat(...arr.map(v => (Array.isArray(v) ? deepFlatten(v) : v)));
```

```
deepFlatten([1, [2], [[3], 4], 5]); // [1, 2, 3, 4, 5]
```

title: deepFreeze

Deep freezes an object.

- Use `Object.keys()` to get all the properties of the passed object, `Array.prototype.forEach()` to iterate over them.
- Call `Object.freeze(obj)` recursively on all properties, applying `deepFreeze()` as necessary.
- Finally, use `Object.freeze()` to freeze the given object.

```
const deepFreeze = obj => {
  Object.keys(obj).forEach(prop => {
    if (typeof obj[prop] === 'object') deepFreeze(obj[prop]);
  });
  return Object.freeze(obj);
};
```

```
'use strict';

const val = deepFreeze([1, [2, 3]]);

val[0] = 3; // not allowed
val[1][0] = 4; // not allowed as well
```

title: deepGet

Gets the target value in a nested JSON object, based on the `keys` array.

- Compare the keys you want in the nested JSON object as an `Array` .
- Use `Array.prototype.reduce()` to get the values in the nested JSON object one by one.
- If the key exists in the object, return the target value, otherwise return `null` .

```
const deepGet = (obj, keys) =>
  keys.reduce(
    (xs, x) => (xs && xs[x] !== null && xs[x] !== undefined ? xs[x] : null),
    obj
  );
```

```
let index = 2;
const data = {
  foo: {
    foz: [1, 2, 3],
    bar: {
      baz: ['a', 'b', 'c']
    }
  }
};
deepGet(data, ['foo', 'foz', index]); // get 3
deepGet(data, ['foo', 'bar', 'baz', 8, 'foz']); // null
```

title: deepMapKeys

Deep maps an object's keys.

- Creates an object with the same values as the provided object and keys generated by running the provided function for each key.

- Use `Object.keys(obj)` to iterate over the object's keys.
- Use `Array.prototype.reduce()` to create a new object with the same values and mapped keys using `fn` .

```
const deepMapKeys = (obj, fn) =>
  Array.isArray(obj)
    ? obj.map(val => deepMapKeys(val, fn))
    : typeof obj === 'object'
    ? Object.keys(obj).reduce((acc, current) => {
        const key = fn(current);
        const val = obj[current];
        acc[key] =
          val !== null && typeof val === 'object' ? deepMapKeys(val, fn) : val;
        return acc;
      }, {})
    : obj;


const obj = {
  foo: '1',
  nested: {
    child: {
      withArray: [
        {
          grandChild: ['hello']
        }
      ]
    }
  }
};
const upperKeysObj = deepMapKeys(obj, key => key.toUpperCase());
/*
{
  "FOO":"1",
  "NESTED":{
    "CHILD":{
      "WITHARRAY":[
        {
          "GRANDCHILD":[ 'hello' ]
        }
      ]
    }
  }
}
*/
```

title: deepMerge

Deeply merges two objects, using a function to handle keys present in both.

- Use `Object.keys()` to get the keys of both objects, create a `Set` from them and use the spread operator (`...`) to create an array of all the unique keys.
- Use `Array.prototype.reduce()` to add each unique key to the object, using `fn` to combine the values of the two given objects.

```
const deepMerge = (a, b, fn) =>
  [...new Set([...Object.keys(a), ...Object.keys(b)])].reduce(
    (acc, key) => ({ ...acc, [key]: fn(key, a[key], b[key]) }),
    {}
  );
```

```
deepMerge(
  { a: true, b: { c: [1, 2, 3] } },
  { a: false, b: { d: [1, 2, 3] } },
  (key, a, b) => (key === 'a' ? a && b : Object.assign({}, a, b))
);
// { a: false, b: { c: [ 1, 2, 3 ], d: [ 1, 2, 3 ] } }
```

title: defaults

Assigns default values for all properties in an object that are `undefined` .

- Use `Object.assign()` to create a new empty object and copy the original one to maintain key order.
- Use `Array.prototype.reverse()` and the spread operator (`...`) to combine the default values from left to right.
- Finally, use `obj` again to overwrite properties that originally had a value.

```
const defaults = (obj, ...defs) =>
  Object.assign({}, obj, ...defs.reverse(), obj);
```

```
defaults({ a: 1 }, { b: 2 }, { b: 6 }, { a: 3 }); // { a: 1, b: 2 }
```

title: defer

Defers invoking a function until the current call stack has cleared.

- Use `setTimeout()` with a timeout of `1` ms to add a new event to the event queue and allow the rendering engine to complete its work.
- Use the spread (`...`) operator to supply the function with an arbitrary number of arguments.

```
const defer = (fn, ...args) => setTimeout(fn, 1, ...args);


// Example A:
defer(console.log, 'a'), console.log('b'); // logs 'b' then 'a'

// Example B:
document.querySelector('#someElement').innerHTML = 'Hello';
longRunningFunction();
// Browser will not update the HTML until this has finished
defer(longRunningFunction);
// Browser will update the HTML then run the function
```

title: degreesToRads

Converts an angle from degrees to radians.

- Use `Math.PI` and the degree to radian formula to convert the angle from degrees to radians.

```
const degreesToRads = deg => (deg * Math.PI) / 180.0;


degreesToRads(90.0); // ~1.5708
```

title: delay

Invokes the provided function after `ms` milliseconds.

- Use `setTimeout()` to delay execution of `fn` .
- Use the spread (`...`) operator to supply the function with an arbitrary number of arguments.

```
const delay = (fn, ms, ...args) => setTimeout(fn, ms, ...args);


delay(
  function(text) {
    console.log(text);
  },
  1000,
  'later'
); // Logs 'later' after one second.
```

title: detectDeviceType

Detects whether the page is being viewed on a mobile device or a desktop.

- Use a regular expression to test the `navigator.userAgent` property to figure out if the device is a mobile device or a desktop.

```
const detectDeviceType = () =>
  /Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i.test(
    navigator.userAgent
  )
    ? 'Mobile'
    : 'Desktop';


detectDeviceType(); // 'Mobile' or 'Desktop'
```

title: detectLanguage

Detects the preferred language of the current user.

- Use `NavigationLanguage.language` or the first `NavigationLanguage.languages` if available, otherwise return `defaultLang` .
- Omit the second argument, `defaultLang` , to use `'en-US'` as the default language code.

```
const detectLanguage = (defaultLang = 'en-US') =>
  navigator.language ||
  (Array.isArray(navigator.languages) && navigator.languages[0]) ||
  defaultLang;
```

```
detectLanguage(); // 'nl-NL'
```

title: difference

Calculates the difference between two arrays, without filtering duplicate values.

- Create a `Set` from `b` to get the unique values in `b` .
- Use `Array.prototype.filter()` on `a` to only keep values not contained in `b` , using `Set.prototype.has()` .

```
const difference = (a, b) => {
  const s = new Set(b);
  return a.filter(x => !s.has(x));
};
```

```
difference([1, 2, 3, 3], [1, 2, 4]); // [3, 3]
```

title: differenceBy

Returns the difference between two arrays, after applying the provided function to each array element of both.

- Create a `Set` by applying `fn` to each element in `b` .
- Use `Array.prototype.map()` to apply `fn` to each element in `a` .
- Use `Array.prototype.filter()` in combination with `fn` on `a` to only keep values not contained in `b` , using `Set.prototype.has()` .

```
const differenceBy = (a, b, fn) => {
  const s = new Set(b.map(fn));
  return a.map(fn).filter(el => !s.has(el));
};
```

```
differenceBy([2.1, 1.2], [2.3, 3.4], Math.floor); // [1]
differenceBy([{ x: 2 }, { x: 1 }], [{ x: 1 }], v => v.x); // [2]
```

title: differenceWith

Filters out all values from an array for which the comparator function does not return `true` .

- Use `Array.prototype.filter()` and `Array.prototype.findIndex()` to find the appropriate values.
- Omit the last argument, `comp` , to use a default strict equality comparator.

```
const differenceWith = (arr, val, comp = (a, b) => a === b) =>
  arr.filter(a => val.findIndex(b => comp(a, b)) === -1);


differenceWith(
  [1, 1.2, 1.5, 3, 0],
  [1.9, 3, 0],
  (a, b) => Math.round(a) === Math.round(b)
); // [1, 1.2]
differenceWith([1, 1.2, 1.3], [1, 1.3, 1.5]); // [1.2]
```

title: dig

Gets the target value in a nested JSON object, based on the given key.

- Use the `in` operator to check if `target` exists in `obj` .
- If found, return the value of `obj[target]` .
- Otherwise use `Object.values(obj)` and `Array.prototype.reduce()` to recursively call `dig` on each nested object until the first matching key/value pair is found.

```
const dig = (obj, target) =>
  target in obj
    ? obj[target]
    : Object.values(obj).reduce((acc, val) => {
        if (acc !== undefined) return acc;
        if (typeof val === 'object') return dig(val, target);
      }, undefined);
```

```
const data = {
  level1: {
    level2: {
      level3: 'some data'
    }
  }
};
dig(data, 'level3'); // 'some data'
dig(data, 'level4'); // undefined
```

title: digitize

Converts a number to an array of digits, removing its sign if necessary.

- Use `Math.abs()` to strip the number's sign.
- Convert the number to a string, using the spread operator (`...`) to build an array.
- Use `Array.prototype.map()` and `parseInt()` to transform each value to an integer.

```
const digitize = n => [...`${Math.abs(n)}`].map(i => parseInt(i));
```

```
digitize(123); // [1, 2, 3]
digitize(-123); // [1, 2, 3]
```

title: distance

Calculates the distance between two points.

- Use `Math.hypot()` to calculate the Euclidean distance between two points.

```
const distance = (x0, y0, x1, y1) => Math.hypot(x1 - x0, y1 - y0);
```

```
distance(1, 1, 2, 3); // ~2.2361
```

title: divmod

Returns an array consisting of the quotient and remainder of the given numbers.

- Use `Math.floor()` to get the quotient of the division `x / y`.
- Use the modulo operator (`%`) to get the remainder of the division `x / y`.

```
const divmod = (x, y) => [Math.floor(x / y), x % y];
```

```
divmod(8, 3); // [2, 2]
divmod(3, 8); // [0, 3]
divmod(5, 5); // [1, 0]
```

title: drop

Creates a new array with `n` elements removed from the left.

- Use `Array.prototype.slice()` to remove the specified number of elements from the left.
- Omit the last argument, `n`, to use a default value of `1`.

```
const drop = (arr, n = 1) => arr.slice(n);
```

```
drop([1, 2, 3]); // [2, 3]
drop([1, 2, 3], 2); // [3]
drop([1, 2, 3], 42); // []
```

title: dropRight

Creates a new array with `n` elements removed from the right.

- Use `Array.prototype.slice()` to remove the specified number of elements from the right.
- Omit the last argument, `n`, to use a default value of `1`.

```
const dropRight = (arr, n = 1) => arr.slice(0, -n);
```

```
dropRight([1, 2, 3]); // [1, 2]
dropRight([1, 2, 3], 2); // [1]
dropRight([1, 2, 3], 42); // []
```

title: dropRightWhile

Removes elements from the end of an array until the passed function returns `true` .
Returns the remaining elements in the array.

- Loop through the array, using `Array.prototype.slice()` to drop the last element of the array until the value returned from `func` is `true` .
- Return the remaining elements.

```
const dropRightWhile = (arr, func) => {
  let rightIndex = arr.length;
  while (rightIndex-- && !func(arr[rightIndex]));
  return arr.slice(0, rightIndex + 1);
};
```

```
dropRightWhile([1, 2, 3, 4], n => n < 3); // [1, 2]
```

title: dropWhile

Removes elements in an array until the passed function returns `true` .
Returns the remaining elements in the array.

- Loop through the array, using `Array.prototype.slice()` to drop the first element of the array until the value returned from `func` is `true` .
- Return the remaining elements.

```
const dropWhile = (arr, func) => {
  while (arr.length > 0 && !func(arr[0])) arr = arr.slice(1);
  return arr;
};
```

```
dropWhile([1, 2, 3, 4], n => n >= 3); // [3, 4]
```

title: either

Checks if at least one function returns `true` for a given set of arguments.

- Use the logical or (`||`) operator on the result of calling the two functions with the supplied `args` .

```
const either = (f, g) => (...args) => f(...args) || g(...args);
```

```
const isEven = num => num % 2 === 0;
const isPositive = num => num > 0;
const isPositiveOrEven = either(isPositive, isEven);
isPositiveOrEven(4); // true
isPositiveOrEven(3); // true
```

title: elementContains

Checks if the `parent` element contains the `child` element.

- Check that `parent` is not the same element as `child` .
- Use `Node.contains()` to check if the `parent` element contains the `child` element.

```
const elementContains = (parent, child) =>
  parent !== child && parent.contains(child);
```

```
elementContains(
  document.querySelector('head'),
  document.querySelector('title')
);
// true
elementContains(document.querySelector('body'), document.querySelector('body'));
// false
```

title: elementIsFocused

Checks if the given element is focused.

- Use `Document.activeElement` to determine if the given element is focused.

```
const elementIsFocused = el => (el === document.activeElement);


elementIsFocused(el); // true if the element is focused
```

title: elementIsVisibleInViewport

Checks if the element specified is visible in the viewport.

- Use `Element.getBoundingClientRect()` and the `Window.inner(Width|Height)` values to determine if a given element is visible in the viewport.
- Omit the second argument to determine if the element is entirely visible, or specify `true` to determine if it is partially visible.

```
const elementIsVisibleInViewport = (el, partiallyVisible = false) => {
  const { top, left, bottom, right } = el.getBoundingClientRect();
  const { innerHeight, innerWidth } = window;
  return partiallyVisible
    ? ((top > 0 && top < innerHeight) ||
        (bottom > 0 && bottom < innerHeight)) &&
        ((left > 0 && left < innerWidth) || (right > 0 && right < innerWidth))
    : top >= 0 && left >= 0 && bottom <= innerHeight && right <= innerWidth;
};


// e.g. 100x100 viewport and a 10x10px element at position {top: -1, left: 0, bottom: 9, right:
elementIsVisibleInViewport(el); // false - (not fully visible)
elementIsVisibleInViewport(el, true); // true - (partially visible)
```

title: equals

Performs a deep comparison between two values to determine if they are equivalent.

- Check if the two values are identical.
- Check if both values are `Date` objects with the same time, using `Date.prototype.getTime()` .
- Check if both values are non-object values with an equivalent value (strict comparison).

- Check if only one value is `null` or `undefined` or if their prototypes differ.
- If none of the above conditions are met, use `Object.keys()` to check if both values have the same number of keys.
- Use `Array.prototype.every()` to check if every key in `a` exists in `b` and if they are equivalent by calling `equals()` recursively.

```
const equals = (a, b) => {
  if (a === b) return true;

  if (a instanceof Date && b instanceof Date)
    return a.getTime() === b.getTime();

  if (!a || !b || (typeof a !== 'object' && typeof b !== 'object'))
    return a === b;

  if (a.prototype !== b.prototype) return false;

  const keys = Object.keys(a);
  if (keys.length !== Object.keys(b).length) return false;

  return keys.every(k => equals(a[k], b[k]));
};


equals(
  { a: [2, { e: 3 }], b: [4], c: 'foo' },
  { a: [2, { e: 3 }], b: [4], c: 'foo' }
); // true
equals([1, 2, 3], { 0: 1, 1: 2, 2: 3 }); // true
```

title: escapeHTML

Escapes a string for use in HTML.

- Use `String.prototype.replace()` with a regexp that matches the characters that need to be escaped.
- Use the callback function to replace each character instance with its associated escaped character using a dictionary object.

```
const escapeHTML = str =>
  str.replace(
    /[&<>'"]/g,
    tag =>
      ({
        '&': '&amp;',
        '<': '&lt;',
        '>': '&gt;',
        "'": '&#39;',
        '"': '&quot;'
      }[tag] || tag)
  );
```

```
escapeHTML('<a href="#">Me & you</a>');
// '&lt;a href=&quot;#&quot;&gt;Me &amp; you&lt;/a&gt;'
```

title: escapeRegExp

Escapes a string to use in a regular expression.

- Use `String.prototype.replace()` to escape special characters.

```
const escapeRegExp = str => str.replace(/[.*+?^${}()|[\]\\]/g, '\\$&');
```

```
escapeRegExp('(test)'); // \\(test\\)
```

title: euclideanDistance

Calculates the distance between two points in any number of dimensions.

- Use `Object.keys()` and `Array.prototype.map()` to map each coordinate to its difference between the two points.
- Use `Math.hypot()` to calculate the Euclidean distance between the two points.

```
const euclideanDistance = (a, b) =>
  Math.hypot(...Object.keys(a).map(k => b[k] - a[k]));
```

```
euclideanDistance([1, 1], [2, 3]); // ~2.2361
euclideanDistance([1, 1, 1], [2, 3, 2]); // ~2.4495
```

title: everyNth

Returns every `nth` element in an array.

- Use `Array.prototype.filter()` to create a new array that contains every `nth` element of a given array.

```
const everyNth = (arr, nth) => arr.filter((e, i) => i % nth === nth - 1);
```

```
everyNth([1, 2, 3, 4, 5, 6], 2); // [ 2, 4, 6 ]
```

title: expandTabs

Convert tabs to spaces, where each tab corresponds to `count` spaces.

- Use `String.prototype.replace()` with a regular expression and `String.prototype.repeat()` to replace each tab character with `count` spaces.

```
const expandTabs = (str, count) => str.replace(/\t/g, ' '.repeat(count));
```

```
expandTabs('\t\tlorem', 3); // '      lorem'
```

title: extendHex

Extends a 3-digit color code to a 6-digit color code.

- Use `Array.prototype.map()` , `String.prototype.split()` and `Array.prototype.join()` to join the mapped array for converting a 3-digit RGB notated hexadecimal color-code to the 6-digit form.
- `Array.prototype.slice()` is used to remove `#` from string start since it's added once.

```
const extendHex = shortHex =>
  '#' +
  shortHex
    .slice(shortHex.startsWith('#') ? 1 : 0)
    .split('')
    .map(x => x + x)
    .join('');


extendHex('#03f'); // '#0033ff'
extendHex('05a'); // '#0055aa'
```

title: factorial

Calculates the factorial of a number.

- Use recursion.
- If `n` is less than or equal to `1`, return `1`.
- Otherwise, return the product of `n` and the factorial of `n - 1`.
- Throw a `TypeError` if `n` is a negative number.

```
const factorial = n =>
  n < 0
    ? (() => {
        throw new TypeError('Negative numbers are not allowed!');
      })()
    : n <= 1
    ? 1
    : n * factorial(n - 1);


factorial(6); // 720
```

title: fahrenheitToCelsius
unlisted: true

Converts Fahrenheit to Celsius.

- Follow the conversion formula `C = (F - 32) * 5/9`.

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


fahrenheitToCelsius(32); // 0
```

title: fibonacci

Generates an array, containing the Fibonacci sequence, up until the nth term.

- Use `Array.from()` to create an empty array of the specific length, initializing the first two values (`0` and `1`).
- Use `Array.prototype.reduce()` and `Array.prototype.concat()` to add values into the array, using the sum of the last two values, except for the first two.

```
const fibonacci = n =>
  Array.from({ length: n }).reduce(
    (acc, val, i) => acc.concat(i > 1 ? acc[i - 1] + acc[i - 2] : i),
    []
  );


fibonacci(6); // [0, 1, 1, 2, 3, 5]
```

title: filterNonUnique

Creates an array with the non-unique values filtered out.

- Use `new Set()` and the spread operator (`...`) to create an array of the unique values in `arr` .
- Use `Array.prototype.filter()` to create an array containing only the unique values.

```
const filterNonUnique = arr =>
  [...new Set(arr)].filter(i => arr.indexOf(i) === arr.lastIndexOf(i));


filterNonUnique([1, 2, 2, 3, 4, 4, 5]); // [1, 3, 5]
```

title: filterNonUniqueBy

Creates an array with the non-unique values filtered out, based on a provided comparator function.

- Use `Array.prototype.filter()` and `Array.prototype.every()` to create an array containing only the unique values, based on the comparator function, `fn` .
- The comparator function takes four arguments: the values of the two elements being compared and their indexes.

```
const filterNonUniqueBy = (arr, fn) =>
  arr.filter((v, i) => arr.every((x, j) => (i === j) === fn(v, x, i, j)));
```

```
filterNonUniqueBy(
  [
    { id: 0, value: 'a' },
    { id: 1, value: 'b' },
    { id: 2, value: 'c' },
    { id: 1, value: 'd' },
    { id: 0, value: 'e' }
  ],
  (a, b) => a.id === b.id
); // [ { id: 2, value: 'c' } ]
```

title: filterUnique

Creates an array with the unique values filtered out.

- Use `new Set()` and the spread operator (`...`) to create an array of the unique values in `arr` .
- Use `Array.prototype.filter()` to create an array containing only the non-unique values.

```
const filterUnique = arr =>
  [...new Set(arr)].filter(i => arr.indexOf(i) !== arr.lastIndexOf(i));
```

```
filterUnique([1, 2, 2, 3, 4, 4, 5]); // [2, 4]
```

title: filterUniqueBy

Creates an array with the unique values filtered out, based on a provided comparator function.

- Use `Array.prototype.filter()` and `Array.prototype.every()` to create an array containing only the non-unique values, based on the comparator function, `fn` .
- The comparator function takes four arguments: the values of the two elements being compared and their indexes.

```
const filterUniqueBy = (arr, fn) =>
  arr.filter((v, i) => arr.some((x, j) => (i !== j) === fn(v, x, i, j)));
```

```
filterUniqueBy(
  [
    { id: 0, value: 'a' },
    { id: 1, value: 'b' },
    { id: 2, value: 'c' },
    { id: 3, value: 'd' },
    { id: 0, value: 'e' }
  ],
  (a, b) => a.id == b.id
); // [ { id: 0, value: 'a' }, { id: 0, value: 'e' } ]
```

title: findClosestAnchor

Finds the anchor node closest to the given `node` , if any.

- Use a `for` loop and `Node.parentNode` to traverse the node tree upwards from the given `node` .
- Use `Node.nodeName` and `String.prototype.toLowerCase()` to check if any given node is an anchor (`'a'`).
- If no matching node is found, return `null` .

```
const findClosestAnchor = node => {
  for (let n = node; n.parentNode; n = n.parentNode)
    if (n.nodeName.toLowerCase() === 'a') return n;
  return null;
};
```

```
findClosestAnchor(document.querySelector('a > span')); // a
```

title: findClosestMatchingNode

Finds the closest matching node starting at the given `node` .

- Use a `for` loop and `Node.parentNode` to traverse the node tree upwards from the given `node` .
- Use `Element.matches()` to check if any given element node matches the provided `selector` .
- If no matching node is found, return `null` .

```
const findClosestMatchingNode = (node, selector) => {
  for (let n = node; n.parentNode; n = n.parentNode)
    if (n.matches && n.matches(selector)) return n;
  return null;
};
```

```
findClosestMatchingNode(document.querySelector('span'), 'body'); // body
```

title: findFirstN

Finds the first `n` elements for which the provided function returns a truthy value.

- Use a `for..in` loop to execute the provided `matcher` for each element of `arr` .
- Use `Array.prototype.push()` to append elements to the results array and return them if its `length` is equal to `n` .

```
const findFirstN = (arr, matcher, n = 1) => {
  let res = [];
  for (let i in arr) {
    const el = arr[i];
    const match = matcher(el, i, arr);
    if (match) res.push(el);
    if (res.length === n) return res;
  }
  return res;
};
```

```
findFirstN([1, 2, 4, 6], n => n % 2 === 0, 2); // [2, 4]
findFirstN([1, 2, 4, 6], n => n % 2 === 0, 5); // [2, 4, 6]
```

title: findKey

Finds the first key that satisfies the provided testing function.
Otherwise `undefined` is returned.

- Use `Object.keys(obj)` to get all the properties of the object, `Array.prototype.find()` to test each key-value pair using `fn` .
- The callback receives three arguments - the value, the key and the object.

```
const findKey = (obj, fn) =>
  Object.keys(obj).find(key => fn(obj[key], key, obj));
```

```
findKey(
  {
    barney: { age: 36, active: true },
    fred: { age: 40, active: false },
    pebbles: { age: 1, active: true }
  },
  x => x['active']
); // 'barney'
```

title: findKeys

Finds all the keys in the provided object that match the given value.

- Use `Object.keys(obj)` to get all the properties of the object.
- Use `Array.prototype.filter()` to test each key-value pair and return all keys that are equal to the given value.

```
const findKeys = (obj, val) =>
  Object.keys(obj).filter(key => obj[key] === val);
```

```
const ages = {
  Leo: 20,
  Zoey: 21,
  Jane: 20,
};
findKeys(ages, 20); // [ 'Leo', 'Jane' ]
```

title: findLast

Finds the last element for which the provided function returns a truthy value.

- Use `Array.prototype.filter()` to remove elements for which `fn` returns falsy values.
- Use `Array.prototype.pop()` to get the last element in the filtered array.

```
const findLast = (arr, fn) => arr.filter(fn).pop();
```

```
findLast([1, 2, 3, 4], n => n % 2 === 1); // 3
```

title: findLastIndex

Finds the index of the last element for which the provided function returns a truthy value.

- Use `Array.prototype.map()` to map each element to an array with its index and value.
- Use `Array.prototype.filter()` to remove elements for which `fn` returns falsy values
- Use `Array.prototype.pop()` to get the last element in the filtered array.
- Return `-1` if there are no matching elements.

```
const findLastIndex = (arr, fn) =>
  (arr
    .map((val, i) => [i, val])
    .filter(([i, val]) => fn(val, i, arr))
    .pop() || [-1])[0];
```

```
findLastIndex([1, 2, 3, 4], n => n % 2 === 1); // 2 (index of the value 3)
findLastIndex([1, 2, 3, 4], n => n === 5); // -1 (default value when not found)
```

title: findLastKey

Finds the last key that satisfies the provided testing function.
Otherwise `undefined` is returned.

- Use `Object.keys(obj)` to get all the properties of the object.

- Use `Array.prototype.reverse()` to reverse the order and `Array.prototype.find()` to test the provided function for each key-value pair.
- The callback receives three arguments - the value, the key and the object.

```
const findLastKey = (obj, fn) =>
  Object.keys(obj)
    .reverse()
    .find(key => fn(obj[key], key, obj));


findLastKey(
  {
    barney: { age: 36, active: true },
    fred: { age: 40, active: false },
    pebbles: { age: 1, active: true }
  },
  x => x['active']
); // 'pebbles'
```

title: findLastN

Finds the last `n` elements for which the provided function returns a truthy value.

- Use a `for` loop to execute the provided `matcher` for each element of `arr`.
- Use `Array.prototype.unshift()` to prepend elements to the results array and return them if its `length` is equal to `n`.

```
const findLastN = (arr, matcher, n = 1) => {
  let res = [];
  for (let i = arr.length - 1; i >= 0; i--) {
    const el = arr[i];
    const match = matcher(el, i, arr);
    if (match) res.unshift(el);
    if (res.length === n) return res;
  }
  return res;
};


findLastN([1, 2, 4, 6], n => n % 2 === 0, 2); // [4, 6]
findLastN([1, 2, 4, 6], n => n % 2 === 0, 5); // [2, 4, 6]
```

title: flatten

Flattens an array up to the specified depth.

- Use recursion, decrementing `depth` by `1` for each level of depth.
- Use `Array.prototype.reduce()` and `Array.prototype.concat()` to merge elements or arrays.
- Base case, for `depth` equal to `1` stops recursion.
- Omit the second argument, `depth` , to flatten only to a depth of `1` (single flatten).

```
const flatten = (arr, depth = 1) =>
  arr.reduce(
    (a, v) =>
      a.concat(depth > 1 && Array.isArray(v) ? flatten(v, depth - 1) : v),
    []
  );
```

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

title: flattenObject

Flattens an object with the paths for keys.

- Use recursion.
- Use `Object.keys(obj)` combined with `Array.prototype.reduce()` to convert every leaf node to a flattened path node.
- If the value of a key is an object, the function calls itself with the appropriate `prefix` to create the path using `Object.assign()` .
- Otherwise, it adds the appropriate prefixed key-value pair to the accumulator object.
- You should always omit the second argument, `prefix` , unless you want every key to have a prefix.

```js
const flattenObject = (obj, prefix = '') =>
  Object.keys(obj).reduce((acc, k) => {
    const pre = prefix.length ? `${prefix}.` : '';
    if (
      typeof obj[k] === 'object' &&
      obj[k] !== null &&
      Object.keys(obj[k]).length > 0
    )
      Object.assign(acc, flattenObject(obj[k], pre + k));
    else acc[pre + k] = obj[k];
    return acc;
  }, {});
```

```js
flattenObject({ a: { b: { c: 1 } }, d: 1 }); // { 'a.b.c': 1, d: 1 }
```

title: flip

Takes a function as an argument, then makes the first argument the last.

- Use argument destructuring and a closure with variadic arguments.
- Splice the first argument, using the spread operator (`...`), to make it the last before applying the rest.

```js
const flip = fn => (first, ...rest) => fn(...rest, first);
```

```js
let a = { name: 'John Smith' };
let b = {};
const mergeFrom = flip(Object.assign);
let mergePerson = mergeFrom.bind(null, a);
mergePerson(b); // == b
b = {};
Object.assign(b, a); // == b
```

title: forEachRight

Executes a provided function once for each array element, starting from the array's last element.

- Use `Array.prototype.slice()` to clone the given array and `Array.prototype.reverse()` to reverse it.
- Use `Array.prototype.forEach()` to iterate over the reversed array.

```
const forEachRight = (arr, callback) =>
  arr
    .slice()
    .reverse()
    .forEach(callback);


forEachRight([1, 2, 3, 4], val => console.log(val)); // '4', '3', '2', '1'
```

title: forOwn

Iterates over all own properties of an object, running a callback for each one.

- Use `Object.keys(obj)` to get all the properties of the object.
- Use `Array.prototype.forEach()` to run the provided function for each key-value pair.
- The callback receives three arguments - the value, the key and the object.

```
const forOwn = (obj, fn) =>
  Object.keys(obj).forEach(key => fn(obj[key], key, obj));


forOwn({ foo: 'bar', a: 1 }, v => console.log(v)); // 'bar', 1
```

title: forOwnRight

Iterates over all own properties of an object in reverse, running a callback for each one.

- Use `Object.keys(obj)` to get all the properties of the object, `Array.prototype.reverse()` to reverse their order.
- Use `Array.prototype.forEach()` to run the provided function for each key-value pair.
- The callback receives three arguments - the value, the key and the object.

```
const forOwnRight = (obj, fn) =>
  Object.keys(obj)
    .reverse()
    .forEach(key => fn(obj[key], key, obj));


forOwnRight({ foo: 'bar', a: 1 }, v => console.log(v)); // 1, 'bar'
```

title: formToObject

Encodes a set of form elements as an `object`.

- Use the `FormData` constructor to convert the HTML `form` to `FormData` and `Array.from()` to convert to an array.
- Collect the object from the array using `Array.prototype.reduce()`.

```
const formToObject = form =>
  Array.from(new FormData(form)).reduce(
    (acc, [key, value]) => ({
      ...acc,
      [key]: value
    }),
    {}
  );


formToObject(document.querySelector('#form'));
// { email: 'test@email.com', name: 'Test Name' }
```

title: formatDuration

Returns the human-readable format of the given number of milliseconds.

- Divide `ms` with the appropriate values to obtain the appropriate values for `day`, `hour`, `minute`, `second` and `millisecond`.
- Use `Object.entries()` with `Array.prototype.filter()` to keep only non-zero values.
- Use `Array.prototype.map()` to create the string for each value, pluralizing appropriately.
- Use `String.prototype.join(', ')` to combine the values into a string.

```
const formatDuration = ms => {
  if (ms < 0) ms = -ms;
  const time = {
    day: Math.floor(ms / 86400000),
    hour: Math.floor(ms / 3600000) % 24,
    minute: Math.floor(ms / 60000) % 60,
    second: Math.floor(ms / 1000) % 60,
    millisecond: Math.floor(ms) % 1000
  };
  return Object.entries(time)
    .filter(val => val[1] !== 0)
    .map(([key, val]) => `${val} ${key}${val !== 1 ? 's' : ''}`)
    .join(', ');
};
```

```
formatDuration(1001); // '1 second, 1 millisecond'
formatDuration(34325055574);
// '397 days, 6 hours, 44 minutes, 15 seconds, 574 milliseconds'
```

title: formatNumber

Formats a number using the local number format order.

- Use `Number.prototype.toLocaleString()` to convert a number to using the local number format separators.

```
const formatNumber = num => num.toLocaleString();
```

```
formatNumber(123456); // '123,456' in `en-US`
formatNumber(15675436903); // '15.675.436.903' in `de-DE`
```

title: formatSeconds

Returns the ISO format of the given number of seconds.

- Divide `s` with the appropriate values to obtain the appropriate values for `hour` , `minute` and `second` .
- Store the `sign` in a variable to prepend it to the result.

- Use `Array.prototype.map()` in combination with `Math.floor()` and `String.prototype.padStart()` to stringify and format each segment.
- Use `String.prototype.join(':')` to combine the values into a string.

```js
const formatSeconds = s => {
  const [hour, minute, second, sign] =
    s > 0
      ? [s / 3600, (s / 60) % 60, s % 60, '']
      : [-s / 3600, (-s / 60) % 60, -s % 60, '-'];

  return (
    sign +
    [hour, minute, second]
      .map(v => `${Math.floor(v)}`.padStart(2, '0'))
      .join(':')
  );
};
```

```js
formatSeconds(200); // '00:03:20'
formatSeconds(-200); // '-00:03:20'
formatSeconds(99999); // '27:46:39'
```

title: frequencies

Creates an object with the unique values of an array as keys and their frequencies as the values.

- Use `Array.prototype.reduce()` to map unique values to an object's keys, adding to existing keys every time the same value is encountered.

```js
const frequencies = arr =>
  arr.reduce((a, v) => {
    a[v] = a[v] ? a[v] + 1 : 1;
    return a;
  }, {});
```

```js
frequencies(['a', 'b', 'a', 'c', 'a', 'a', 'b']); // { a: 4, b: 2, c: 1 }
frequencies([...'ball']); // { b: 1, a: 1, l: 2 }
```

title: fromCamelCase

Converts a string from camelcase.

- Use `String.prototype.replace()` to break the string into words and add a `separator` between them.
- Omit the second argument to use a default `separator` of `_`.

```js
const fromCamelCase = (str, separator = '_') =>
  str
    .replace(/([a-z\d])([A-Z])/g, '$1' + separator + '$2')
    .replace(/([A-Z]+)([A-Z][a-z\d]+)/g, '$1' + separator + '$2')
    .toLowerCase();
```

```js
fromCamelCase('someDatabaseFieldName', ' '); // 'some database field name'
fromCamelCase('someLabelThatNeedsToBeDecamelized', '-');
// 'some-label-that-needs-to-be-decamelized'
fromCamelCase('someJavascriptProperty', '_'); // 'some_javascript_property'
fromCamelCase('JSONToCSV', '.'); // 'json.to.csv'
```

title: fromTimestamp

Creates a `Date` object from a Unix timestamp.

- Convert the timestamp to milliseconds by multiplying with `1000`.
- Use `new Date()` to create a new `Date` object.

```js
const fromTimestamp = timestamp => new Date(timestamp * 1000);
```

```js
fromTimestamp(1602162242); // 2020-10-08T13:04:02.000Z
```

title: frozenSet

Creates a frozen `Set` object.

- Use the `new Set()` constructor to create a new `Set` object from `iterable`.

- Set the `add`, `delete` and `clear` methods of the newly created object to `undefined`, so that they cannot be used, practically freezing the object.

```
const frozenSet = iterable => {
  const s = new Set(iterable);
  s.add = undefined;
  s.delete = undefined;
  s.clear = undefined;
  return s;
};
```

```
frozenSet([1, 2, 3, 1, 2]);
// Set { 1, 2, 3, add: undefined, delete: undefined, clear: undefined }
```

title: fullscreen

Opens or closes an element in fullscreen mode.

- Use `Document.querySelector()` and `Element.requestFullscreen()` to open the given element in fullscreen.
- Use `Document.exitFullscreen()` to exit fullscreen mode.
- Omit the second argument, `el`, to use `body` as the default element.
- Omit the first element, `mode`, to open the element in fullscreen mode by default.

```
const fullscreen = (mode = true, el = 'body') =>
  mode
    ? document.querySelector(el).requestFullscreen()
    : document.exitFullscreen();
```

```
fullscreen(); // Opens `body` in fullscreen mode
fullscreen(false); // Exits fullscreen mode
```

title: functionName

Logs the name of a function.

- Use `console.debug()` and the `name` property of the passed function to log the function's name to the `debug` channel of the console.
- Return the given function `fn` .

```
const functionName = fn => (console.debug(fn.name), fn);


let m = functionName(Math.max)(5, 6);
// max (logged in debug channel of console)
// m = 6
```

title: functions

Gets an array of function property names from own (and optionally inherited) enumerable properties of an object.

- Use `Object.keys(obj)` to iterate over the object's own properties.
- If `inherited` is `true` , use `Object.getPrototypeOf(obj)` to also get the object's inherited properties.
- Use `Array.prototype.filter()` to keep only those properties that are functions.
- Omit the second argument, `inherited` , to not include inherited properties by default.

```
const functions = (obj, inherited = false) =>
  (inherited
    ? [...Object.keys(obj), ...Object.keys(Object.getPrototypeOf(obj))]
    : Object.keys(obj)
  ).filter(key => typeof obj[key] === 'function');


function Foo() {
  this.a = () => 1;
  this.b = () => 2;
}
Foo.prototype.c = () => 3;
functions(new Foo()); // ['a', 'b']
functions(new Foo(), true); // ['a', 'b', 'c']
```

title: gcd

Calculates the greatest common divisor between two or more numbers/arrays.

- The inner `_gcd` function uses recursion.
- Base case is when `y` equals `0`. In this case, return `x`.
- Otherwise, return the GCD of `y` and the remainder of the division `x/y`.

```
const gcd = (...arr) => {
  const _gcd = (x, y) => (!y ? x : gcd(y, x % y));
  return [...arr].reduce((a, b) => _gcd(a, b));
};
```

```
gcd(8, 36); // 4
gcd(...[12, 8, 32]); // 4
```

title: generateItems

Generates an array with the given amount of items, using the given function.

- Use `Array.from()` to create an empty array of the specific length, calling `fn` with the index of each newly created element.
- The callback takes one argument - the index of each element.

```
const generateItems = (n, fn) => Array.from({ length: n }, (_, i) => fn(i));
```

```
generateItems(10, Math.random);
// [0.21, 0.08, 0.40, 0.96, 0.96, 0.24, 0.19, 0.96, 0.42, 0.70]
```

title: generatorToArray

Converts the output of a generator function to an array.

- Use the spread operator (`...`) to convert the output of the generator function to an array.

```
const generatorToArray = gen => [...gen];
```

```js
const s = new Set([1, 2, 1, 3, 1, 4]);
generatorToArray(s.entries()); // [[ 1, 1 ], [ 2, 2 ], [ 3, 3 ], [ 4, 4 ]]
```

title: geometricProgression

Initializes an array containing the numbers in the specified range where `start` and `end` are inclusive and the ratio between two terms is `step` .
Returns an error if `step` equals `1` .

- Use `Array.from()` , `Math.log()` and `Math.floor()` to create an array of the desired length, `Array.prototype.map()` to fill with the desired values in a range.
- Omit the second argument, `start` , to use a default value of `1` .
- Omit the third argument, `step` , to use a default value of `2` .

```js
const geometricProgression = (end, start = 1, step = 2) =>
  Array.from({
    length: Math.floor(Math.log(end / start) / Math.log(step)) + 1,
  }).map((_, i) => start * step ** i);
```

```js
geometricProgression(256); // [1, 2, 4, 8, 16, 32, 64, 128, 256]
geometricProgression(256, 3); // [3, 6, 12, 24, 48, 96, 192]
geometricProgression(256, 1, 4); // [1, 4, 16, 64, 256]
```

title: get

Retrieves a set of properties indicated by the given selectors from an object.

- Use `Array.prototype.map()` for each selector, `String.prototype.replace()` to replace square brackets with dots.
- Use `String.prototype.split('.')` to split each selector.
- Use `Array.prototype.filter()` to remove empty values and `Array.prototype.reduce()` to get the value indicated by each selector.

```
const get = (from, ...selectors) =>
  [...selectors].map(s =>
    s
      .replace(/\[([^\[\]]*)\]/g, '.$1.')
      .split('.')
      .filter(t => t !== '')
      .reduce((prev, cur) => prev && prev[cur], from)
  );


const obj = {
  selector: { to: { val: 'val to select' } },
  target: [1, 2, { a: 'test' }],
};
get(obj, 'selector.to.val', 'target[0]', 'target[2].a');
// ['val to select', 1, 'test']
```

title: getAncestors

Returns all the ancestors of an element from the document root to the given element.

- Use `Node.parentNode` and a `while` loop to move up the ancestor tree of the element.
- Use `Array.prototype.unshift()` to add each new ancestor to the start of the array.

```
const getAncestors = el => {
  let ancestors = [];
  while (el) {
    ancestors.unshift(el);
    el = el.parentNode;
  }
  return ancestors;
};


getAncestors(document.querySelector('nav'));
// [document, html, body, header, nav]
```

title: getBaseURL

Gets the current URL without any parameters or fragment identifiers.

- Use `String.prototype.replace()` with an appropriate regular expression to remove everything after either `'?'` or `'#'` , if found.

```
const getBaseURL = url => url.replace(/[?#].*$/, '');


getBaseURL('http://url.com/page?name=Adam&surname=Smith');
// 'http://url.com/page'
```

title: getColonTimeFromDate

Returns a string of the form `HH:MM:SS` from a `Date` object.

- Use `Date.prototype.toTimeString()` and `String.prototype.slice()` to get the `HH:MM:SS` part of a given `Date` object.

```
const getColonTimeFromDate = date => date.toTimeString().slice(0, 8);


getColonTimeFromDate(new Date()); // '08:38:00'
```

title: getDaysDiffBetweenDates

Calculates the difference (in days) between two dates.

- Subtract the two `Date` objects and divide by the number of milliseconds in a day to get the difference (in days) between them.

```
const getDaysDiffBetweenDates = (dateInitial, dateFinal) =>
  (dateFinal - dateInitial) / (1000 * 3600 * 24);


getDaysDiffBetweenDates(new Date('2017-12-13'), new Date('2017-12-22')); // 9
```

title: getElementsBiggerThanViewport

Returns an array of HTML elements whose width is larger than that of the viewport's.

- Use `HTMLElement.offsetWidth` to get the width of the `document`.
- Use `Array.prototype.filter()` on the result of `Document.querySelectorAll()` to check the width of all elements in the document.

```
const getElementsBiggerThanViewport = () => {
  const docWidth = document.documentElement.offsetWidth;
  return [...document.querySelectorAll('*')].filter(
    el => el.offsetWidth > docWidth
  );
};


getElementsBiggerThanViewport(); // <div id="ultra-wide-item" />
```

title: getHoursDiffBetweenDates

Calculates the difference (in hours) between two dates.

- Subtract the two `Date` objects and divide by the number of milliseconds in an hour to get the difference (in hours) between them.

```
const getHoursDiffBetweenDates = (dateInitial, dateFinal) =>
  (dateFinal - dateInitial) / (1000 * 3600);


getHoursDiffBetweenDates(
  new Date('2021-04-24 10:25:00'),
  new Date('2021-04-25 10:25:00')
); // 24
```

title: getImages

Fetches all images from within an element and puts them into an array.

- Use `Element.getElementsByTagName()` to get all `<img>` elements inside the provided element.
- Use `Array.prototype.map()` to map every `src` attribute of each `<img>` element.

- If `includeDuplicates` is `false`, create a new `Set` to eliminate duplicates and return it after spreading into an array.
- Omit the second argument, `includeDuplicates`, to discard duplicates by default.

```
const getImages = (el, includeDuplicates = false) => {
  const images = [...el.getElementsByTagName('img')].map(img =>
    img.getAttribute('src')
  );
  return includeDuplicates ? images : [...new Set(images)];
};
```

```
getImages(document, true); // ['image1.jpg', 'image2.png', 'image1.png', '...']
getImages(document, false); // ['image1.jpg', 'image2.png', '...']
```

title: getMeridiemSuffixOfInteger

Converts an integer to a suffixed string, adding `am` or `pm` based on its value.

- Use the modulo operator (`%`) and conditional checks to transform an integer to a stringified 12-hour format with meridiem suffix.

```
const getMeridiemSuffixOfInteger = num =>
  num === 0 || num === 24
    ? 12 + 'am'
    : num === 12
    ? 12 + 'pm'
    : num < 12
    ? (num % 12) + 'am'
    : (num % 12) + 'pm';
```

```
getMeridiemSuffixOfInteger(0); // '12am'
getMeridiemSuffixOfInteger(11); // '11am'
getMeridiemSuffixOfInteger(13); // '1pm'
getMeridiemSuffixOfInteger(25); // '1pm'
```

title: getMinutesDiffBetweenDates

Calculates the difference (in minutes) between two dates.

- Subtract the two `Date` objects and divide by the number of milliseconds in a minute to get the difference (in minutes) between them.

```
const getMinutesDiffBetweenDates = (dateInitial, dateFinal) =>
  (dateFinal - dateInitial) / (1000 * 60);


getMinutesDiffBetweenDates(
  new Date('2021-04-24 01:00:15'),
  new Date('2021-04-24 02:00:15')
); // 60
```

title: getMonthsDiffBetweenDates

Calculates the difference (in months) between two dates.

- Use `Date.prototype.getFullYear()` and `Date.prototype.getMonth()` to calculate the difference (in months) between two `Date` objects.

```
const getMonthsDiffBetweenDates = (dateInitial, dateFinal) =>
  Math.max(
    (dateFinal.getFullYear() - dateInitial.getFullYear()) * 12 +
      dateFinal.getMonth() -
      dateInitial.getMonth(),
    0
  );


getMonthsDiffBetweenDates(new Date('2017-12-13'), new Date('2018-04-29')); // 4
```

title: getParentsUntil

Finds all the ancestors of an element up until the element matched by the specified selector.

- Use `Node.parentNode` and a `while` loop to move up the ancestor tree of the element.
- Use `Array.prototype.unshift()` to add each new ancestor to the start of the array.
- Use `Element.matches()` to check if the current element matches the specified `selector` .

```
const getParentsUntil = (el, selector) => {
  let parents = [],
    _el = el.parentNode;
  while (_el && typeof _el.matches === 'function') {
    parents.unshift(_el);
    if (_el.matches(selector)) return parents;
    else _el = _el.parentNode;
  }
  return [];
};


getParentsUntil(document.querySelector('#home-link'), 'header');
// [header, nav, ul, li]
```

title: getProtocol

Gets the protocol being used on the current page.

- Use `Window.location.protocol` to get the protocol (`http:` or `https:`) of the current page.

```
const getProtocol = () => window.location.protocol;


getProtocol(); // 'https:'
```

title: getScrollPosition

Returns the scroll position of the current page.

- Use `Window.pageXOffset` and `Window.pageYOffset` if they are defined, otherwise `Element.scrollLeft` and `Element.scrollTop` .
- Omit the single argument, `el` , to use a default value of `window` .

```
const getScrollPosition = (el = window) => ({
  x: el.pageXOffset !== undefined ? el.pageXOffset : el.scrollLeft,
  y: el.pageYOffset !== undefined ? el.pageYOffset : el.scrollTop
});
```

```
getScrollPosition(); // {x: 0, y: 200}
```

title: getSecondsDiffBetweenDates

Calculates the difference (in seconds) between two dates.

- Subtract the two `Date` objects and divide by the number of milliseconds in a second to get the difference (in seconds) between them.

```
const getSecondsDiffBetweenDates = (dateInitial, dateFinal) =>
  (dateFinal - dateInitial) / 1000;
```

```
getSecondsDiffBetweenDates(
  new Date('2020-12-24 00:00:15'),
  new Date('2020-12-24 00:00:17')
); // 2
```

title: getSelectedText

Gets the currently selected text.

- Use `Window.getSelection()` and `Selection.toString()` to get the currently selected text.

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

```
getSelectedText(); // 'Lorem ipsum'
```

title: getSiblings

Returns an array containing all the siblings of the given element.

- Use `Node.parentNode` and `Node.childNodes` to get a `NodeList` of all the elements contained in the element's parent.

- Use the spread operator (`...`) and `Array.prototype.filter()` to convert to an array and remove the given element from it.

```
const getSiblings = el =>
  [...el.parentNode.childNodes].filter(node => node !== el);


getSiblings(document.querySelector('head')); // ['body']
```

title: getStyle

Retrieves the value of a CSS rule for the specified element.

- Use `Window.getComputedStyle()` to get the value of the CSS rule for the specified element.

```
const getStyle = (el, ruleName) => getComputedStyle(el)[ruleName];


getStyle(document.querySelector('p'), 'font-size'); // '16px'
```

title: getTimestamp

Gets the Unix timestamp from a `Date` object.

- Use `Date.prototype.getTime()` to get the timestamp in milliseconds and divide by `1000` to get the timestamp in seconds.
- Use `Math.floor()` to appropriately round the resulting timestamp to an integer.
- Omit the argument, `date` , to use the current date.

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


getTimestamp(); // 1602162242
```

title: getType

Returns the native type of a value.

- Return `'undefined'` or `'null'` if the value is `undefined` or `null`.
- Otherwise, use `Object.prototype.constructor.name` to get the name of the constructor.

```
const getType = v =>
  (v === undefined ? 'undefined' : v === null ? 'null' : v.constructor.name);


getType(new Set([1, 2, 3])); // 'Set'
```

title: getURLParameters

Creates an object containing the parameters of the current URL.

- Use `String.prototype.match()` with an appropriate regular expression to get all key-value pairs.
- Use `Array.prototype.reduce()` to map and combine them into a single object.
- Pass `location.search` as the argument to apply to the current `url`.

```
const getURLParameters = url =>
  (url.match(/([^?=&]+)(=([^&]*))/g) || []).reduce(
    (a, v) => (
      (a[v.slice(0, v.indexOf('='))] = v.slice(v.indexOf('=') + 1)), a
    ),
    {}
  );


getURLParameters('google.com'); // {}
getURLParameters('http://url.com/page?name=Adam&surname=Smith');
// {name: 'Adam', surname: 'Smith'}
```

title: getVerticalOffset

Finds the distance from a given element to the top of the document.

- Use a `while` loop and `HTMLElement.offsetParent` to move up the offset parents of the given element.

- Add `HTMLElement.offsetTop` for each element and return the result.

```
const getVerticalOffset = el => {
  let offset = el.offsetTop,
    _el = el;
  while (_el.offsetParent) {
    _el = _el.offsetParent;
    offset += _el.offsetTop;
  }
  return offset;
};


getVerticalOffset('.my-element'); // 120
```

title: groupBy

Groups the elements of an array based on the given function.

- Use `Array.prototype.map()` to map the values of the array to a function or property name.
- Use `Array.prototype.reduce()` to create an object, where the keys are produced from the mapped results.

```
const groupBy = (arr, fn) =>
  arr
    .map(typeof fn === 'function' ? fn : val => val[fn])
    .reduce((acc, val, i) => {
      acc[val] = (acc[val] || []).concat(arr[i]);
      return acc;
    }, {});


groupBy([6.1, 4.2, 6.3], Math.floor); // {4: [4.2], 6: [6.1, 6.3]}
groupBy(['one', 'two', 'three'], 'length'); // {3: ['one', 'two'], 5: ['three']}
```

title: hammingDistance

Calculates the Hamming distance between two values.

- Use the XOR operator (`^`) to find the bit difference between the two numbers.

- Convert to a binary string using `Number.prototype.toString(2)` .
- Count and return the number of `1` s in the string, using `String.prototype.match(/1/g)` .

```
const hammingDistance = (num1, num2) =>
  ((num1 ^ num2).toString(2).match(/1/g) || '').length;


hammingDistance(2, 3); // 1
```

title: hasClass

Checks if the given element has the specified class.

- Use `Element.classList` and `DOMTokenList.contains()` to check if the element has the specified class.

```
const hasClass = (el, className) => el.classList.contains(className);


hasClass(document.querySelector('p.special'), 'special'); // true
```

title: hasDuplicates

Checks if there are duplicate values in a flat array.

- Use `Set()` to get the unique values in the array.
- Use `Set.prototype.size` and `Array.prototype.length` to check if the count of the unique values is the same as elements in the original array.

```
const hasDuplicates = arr => new Set(arr).size !== arr.length;


hasDuplicates([0, 1, 1, 2]); // true
hasDuplicates([0, 1, 2, 3]); // false
```

title: hasFlags

Checks if the current process's arguments contain the specified flags.

- Use `Array.prototype.every()` and `Array.prototype.includes()` to check if `process.argv` contains all the specified flags.
- Use a regular expression to test if the specified flags are prefixed with `-` or `--` and prefix them accordingly.

```
const hasFlags = (...flags) =>
  flags.every(flag =>
    process.argv.includes(/^-{1,2}/.test(flag) ? flag : '--' + flag)
  );
```

```
// node myScript.js -s --test --cool=true
hasFlags('-s'); // true
hasFlags('--test', 'cool=true', '-s'); // true
hasFlags('special'); // false
```

title: hasKey

Checks if the target value exists in a JSON object.

- Check if `keys` is non-empty and use `Array.prototype.every()` to sequentially check its keys to internal depth of the object, `obj` .
- Use `Object.prototype.hasOwnProperty()` to check if `obj` does not have the current key or is not an object, stop propagation and return `false` .
- Otherwise assign the key's value to `obj` to use on the next iteration.
- Return `false` beforehand if given key list is empty.

```
const hasKey = (obj, keys) => {
  return (
    keys.length > 0 &&
    keys.every(key => {
      if (typeof obj !== 'object' || !obj.hasOwnProperty(key)) return false;
      obj = obj[key];
      return true;
    })
  );
};
```

```
let obj = {
  a: 1,
  b: { c: 4 },
  'b.d': 5
};
hasKey(obj, ['a']); // true
hasKey(obj, ['b']); // true
hasKey(obj, ['b', 'c']); // true
hasKey(obj, ['b.d']); // true
hasKey(obj, ['d']); // false
hasKey(obj, ['c']); // false
hasKey(obj, ['b', 'f']); // false
```

title: hasMany

Checks if an array has more than one value matching the given function.

- Use `Array.prototype.filter()` in combination with `fn` to find all matching array elements.
- Use `Array.prototype.length` to check if more than one element match `fn` .

```
const hasMany = (arr, fn) => arr.filter(fn).length > 1;
```

```
hasMany([1, 3], x => x % 2); // true
hasMany([1, 2], x => x % 2); // false
```

title: hasOne

Checks if an array has only one value matching the given function.

- Use `Array.prototype.filter()` in combination with `fn` to find all matching array elements.
- Use `Array.prototype.length` to check if only one element matches `fn` .

```
const hasOne = (arr, fn) => arr.filter(fn).length === 1;
```

```
hasOne([1, 2], x => x % 2); // true
hasOne([1, 3], x => x % 2); // false
```

title: hashBrowser

Creates a hash for a value using the SHA-256 algorithm.
Returns a promise.

- Use the SubtleCrypto API to create a hash for the given value.
- Create a new `TextEncoder` and use it to encode `val`. Pass its value to `SubtleCrypto.digest()` to generate a digest of the given data.
- Use `DataView.prototype.getUint32()` to read data from the resolved `ArrayBuffer`.
- Convert the data to it hexadecimal representation using `Number.prototype.toString(16)`. Add the data to an array using `Array.prototype.push()`.
- Finally, use `Array.prototype.join()` to combine values in the array of `hexes` into a string.

```
const hashBrowser = val =>
  crypto.subtle
    .digest('SHA-256', new TextEncoder('utf-8').encode(val))
    .then(h => {
      let hexes = [],
        view = new DataView(h);
      for (let i = 0; i < view.byteLength; i += 4)
        hexes.push(('00000000' + view.getUint32(i).toString(16)).slice(-8));
      return hexes.join('');
    });


hashBrowser(
  JSON.stringify({ a: 'a', b: [1, 2, 3, 4], foo: { c: 'bar' } })
).then(console.log);
// '04aa106279f5977f59f9067fa9712afc4aedc6f5862a8defc34552d8c7206393'
```

title: hashNode

Creates a hash for a value using the SHA-256 algorithm.
Returns a promise.

- Use `crypto.createHash()` to create a `Hash` object with the appropriate algorithm.
- Use `hash.update()` to add the data from `val` to the `Hash`, `hash.digest()` to calculate the digest of the data.

- Use `setTimeout()` to prevent blocking on a long operation. Return a `Promise` to give it a familiar interface.

```
const crypto = require('crypto');

const hashNode = val =>
  new Promise(resolve =>
    setTimeout(
      () => resolve(crypto.createHash('sha256').update(val).digest('hex')),
      0
    )
  );
```

```
hashNode(JSON.stringify({ a: 'a', b: [1, 2, 3, 4], foo: { c: 'bar' } })).then(
  console.log
);
// '04aa106279f5977f59f9067fa9712afc4aedc6f5862a8defc34552d8c7206393'
```

title: haveSameContents

Checks if two arrays contain the same elements regardless of order.

- Use a `for...of` loop over a `Set` created from the values of both arrays.
- Use `Array.prototype.filter()` to compare the amount of occurrences of each distinct value in both arrays.
- Return `false` if the counts do not match for any element, `true` otherwise.

```
const haveSameContents = (a, b) => {
  for (const v of new Set([...a, ...b]))
    if (a.filter(e => e === v).length !== b.filter(e => e === v).length)
      return false;
  return true;
};
```

```
haveSameContents([1, 2, 4], [2, 4, 1]); // true
```

title: head

Returns the head of an array.

- Check if `arr` is truthy and has a `length` property.
- Use `arr[0]` if possible to return the first element, otherwise return `undefined`.

```
const head = arr => (arr && arr.length ? arr[0] : undefined);
```

```
head([1, 2, 3]); // 1
head([]); // undefined
head(null); // undefined
head(undefined); // undefined
```

title: heapsort

Sorts an array of numbers, using the heapsort algorithm.

- Use recursion.
- Use the spread operator (`...`) to clone the original array, `arr`.
- Use closures to declare a variable, `l`, and a function `heapify`.
- Use a `for` loop and `Math.floor()` in combination with `heapify` to create a max heap from the array.
- Use a `for` loop to repeatedly narrow down the considered range, using `heapify` and swapping values as necessary in order to sort the cloned array.

```
const heapsort = arr => {
  const a = [...arr];
  let l = a.length;

  const heapify = (a, i) => {
    const left = 2 * i + 1;
    const right = 2 * i + 2;
    let max = i;
    if (left < l && a[left] > a[max]) max = left;
    if (right < l && a[right] > a[max]) max = right;
    if (max !== i) {
      [a[max], a[i]] = [a[i], a[max]];
      heapify(a, max);
    }
  };

  for (let i = Math.floor(l / 2); i >= 0; i -= 1) heapify(a, i);
  for (i = a.length - 1; i > 0; i--) {
    [a[0], a[i]] = [a[i], a[0]];
    l--;
    heapify(a, 0);
  }
  return a;
};
```

```
heapsort([6, 3, 4, 1]); // [1, 3, 4, 6]
```

title: hexToRGB

Converts a color code to an `rgb()` or `rgba()` string if alpha value is provided.

- Use bitwise right-shift operator and mask bits with `&` (and) operator to convert a hexadecimal color code (with or without prefixed with `#`) to a string with the RGB values.
- If it's 3-digit color code, first convert to 6-digit version.
- If an alpha value is provided alongside 6-digit hex, give `rgba()` string in return.

```
const hexToRGB = hex => {
  let alpha = false,
    h = hex.slice(hex.startsWith('#') ? 1 : 0);
  if (h.length === 3) h = [...h].map(x => x + x).join('');
  else if (h.length === 8) alpha = true;
  h = parseInt(h, 16);
  return (
    'rgb' +
    (alpha ? 'a' : '') +
    '(' +
    (h >>> (alpha ? 24 : 16)) +
    ', ' +
    ((h & (alpha ? 0x00ff0000 : 0x00ff00)) >>> (alpha ? 16 : 8)) +
    ', ' +
    ((h & (alpha ? 0x0000ff00 : 0x0000ff)) >>> (alpha ? 8 : 0)) +
    (alpha ? `, ${h & 0x000000ff}` : '') +
    ')'
  );
};
```

```
hexToRGB('#27ae60ff'); // 'rgba(39, 174, 96, 255)'
hexToRGB('27ae60'); // 'rgb(39, 174, 96)'
hexToRGB('#fff'); // 'rgb(255, 255, 255)'
```

title: hide

Hides all the elements specified.

- Use the spread operator (`...`) and `Array.prototype.forEach()` to apply `display: none` to each element specified.

```
const hide = (...el) => [...el].forEach(e => (e.style.display = 'none'));
```

```
hide(...document.querySelectorAll('img')); // Hides all <img> elements on the page
```

title: httpDelete

Makes a `DELETE` request to the passed URL.

- Use the `XMLHttpRequest` web API to make a `DELETE` request to the given `url`.
- Handle the `onload` event, by running the provided `callback` function.
- Handle the `onerror` event, by running the provided `err` function.
- Omit the third argument, `err` to log the request to the console's error stream by default.

```
const httpDelete = (url, callback, err = console.error) => {
  const request = new XMLHttpRequest();
  request.open('DELETE', url, true);
  request.onload = () => callback(request);
  request.onerror = () => err(request);
  request.send();
};
```

```
httpDelete('https://jsonplaceholder.typicode.com/posts/1', request => {
  console.log(request.responseText);
}); // Logs: {}
```

title: httpGet

Makes a `GET` request to the passed URL.

- Use the `XMLHttpRequest` web API to make a `GET` request to the given `url`.
- Handle the `onload` event, by calling the given `callback` the `responseText`.
- Handle the `onerror` event, by running the provided `err` function.
- Omit the third argument, `err`, to log errors to the console's `error` stream by default.

```
const httpGet = (url, callback, err = console.error) => {
  const request = new XMLHttpRequest();
  request.open('GET', url, true);
  request.onload = () => callback(request.responseText);
  request.onerror = () => err(request);
  request.send();
};
```

```
httpGet(
  'https://jsonplaceholder.typicode.com/posts/1',
  console.log
); /*
Logs: {
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit mol
}
*/
```

title: httpPost

Makes a `POST` request to the passed URL.

- Use the `XMLHttpRequest` web API to make a `POST` request to the given `url` .
- Set the value of an `HTTP` request header with `setRequestHeader` method.
- Handle the `onload` event, by calling the given `callback` the `responseText` .
- Handle the `onerror` event, by running the provided `err` function.
- Omit the fourth argument, `err` , to log errors to the console's `error` stream by default.

```
const httpPost = (url, data, callback, err = console.error) => {
  const request = new XMLHttpRequest();
  request.open('POST', url, true);
  request.setRequestHeader('Content-type', 'application/json; charset=utf-8');
  request.onload = () => callback(request.responseText);
  request.onerror = () => err(request);
  request.send(data);
};
```

```js
const newPost = {
  userId: 1,
  id: 1337,
  title: 'Foo',
  body: 'bar bar bar'
};
const data = JSON.stringify(newPost);
httpPost(
  'https://jsonplaceholder.typicode.com/posts',
  data,
  console.log
); /*
Logs: {
  "userId": 1,
  "id": 1337,
  "title": "Foo",
  "body": "bar bar bar"
}
*/
httpPost(
  'https://jsonplaceholder.typicode.com/posts',
  null, // does not send a body
  console.log
); /*
Logs: {
  "id": 101
}
*/
```

title: httpPut

Makes a `PUT` request to the passed URL.

- Use `XMLHttpRequest` web api to make a `PUT` request to the given `url` .
- Set the value of an `HTTP` request header with `setRequestHeader` method.
- Handle the `onload` event, by running the provided `callback` function.
- Handle the `onerror` event, by running the provided `err` function.
- Omit the last argument, `err` to log the request to the console's error stream by default.

```
const httpPut = (url, data, callback, err = console.error) => {
  const request = new XMLHttpRequest();
  request.open('PUT', url, true);
  request.setRequestHeader('Content-type', 'application/json; charset=utf-8');
  request.onload = () => callback(request);
  request.onerror = () => err(request);
  request.send(data);
};


const password = 'fooBaz';
const data = JSON.stringify({
  id: 1,
  title: 'foo',
  body: 'bar',
  userId: 1
});
httpPut('https://jsonplaceholder.typicode.com/posts/1', data, request => {
  console.log(request.responseText);
}); /*
Logs: {
  id: 1,
  title: 'foo',
  body: 'bar',
  userId: 1
}
*/
```

title: httpsRedirect

Redirects the page to HTTPS if it's currently in HTTP.

- Use `location.protocol` to get the protocol currently being used.
- If it's not HTTPS, use `location.replace()` to replace the existing page with the HTTPS version of the page.
- Use `location.href` to get the full address, split it with `String.prototype.split()` and remove the protocol part of the URL.
- Note that pressing the back button doesn't take it back to the HTTP page as its replaced in the history.

```
const httpsRedirect = () => {
  if (location.protocol !== 'https:')
    location.replace('https://' + location.href.split('//')[1]);
};


httpsRedirect();
// If you are on http://mydomain.com, you are redirected to https://mydomain.com
```

title: hz
unlisted: true

Measures the number of times a function is executed per second (`hz` / `hertz`).

- Use `performance.now()` to get the difference in milliseconds before and after the iteration loop to calculate the time elapsed executing the function `iterations` times.
- Return the number of cycles per second by converting milliseconds to seconds and dividing it by the time elapsed.
- Omit the second argument, `iterations` , to use the default of 100 iterations.

```
const hz = (fn, iterations = 100) => {
  const before = performance.now();
  for (let i = 0; i < iterations; i++) fn();
  return (1000 * iterations) / (performance.now() - before);
};


const numbers = Array(10000).fill().map((_, i) => i);

const sumReduce = () => numbers.reduce((acc, n) => acc + n, 0);
const sumForLoop = () => {
  let sum = 0;
  for (let i = 0; i < numbers.length; i++) sum += numbers[i];
  return sum;
};

Math.round(hz(sumReduce)); // 572
Math.round(hz(sumForLoop)); // 4784
```

title: inRange

Checks if the given number falls within the given range.

- Use arithmetic comparison to check if the given number is in the specified range.
- If the second argument, `end`, is not specified, the range is considered to be from `0` to `start`.

```
const inRange = (n, start, end = null) => {
  if (end && start > end) [end, start] = [start, end];
  return end == null ? n >= 0 && n < start : n >= start && n < end;
};
```

```
inRange(3, 2, 5); // true
inRange(3, 4); // true
inRange(2, 3, 5); // false
inRange(3, 2); // false
```

title: includesAll

Checks if all the elements in `values` are included in `arr`.

- Use `Array.prototype.every()` and `Array.prototype.includes()` to check if all elements of `values` are included in `arr`.

```
const includesAll = (arr, values) => values.every(v => arr.includes(v));
```

```
includesAll([1, 2, 3, 4], [1, 4]); // true
includesAll([1, 2, 3, 4], [1, 5]); // false
```

title: includesAny

Checks if at least one element of `values` is included in `arr`.

- Use `Array.prototype.some()` and `Array.prototype.includes()` to check if at least one element of `values` is included in `arr`.

```
const includesAny = (arr, values) => values.some(v => arr.includes(v));
```

```
includesAny([1, 2, 3, 4], [2, 9]); // true
includesAny([1, 2, 3, 4], [8, 9]); // false
```

title: indentString

Indents each line in the provided string.

- Use `String.prototype.replace()` and a regular expression to add the character specified by `indent` `count` times at the start of each line.
- Omit the third argument, `indent` , to use a default indentation character of `' '` .

```
const indentString = (str, count, indent = ' ') =>
  str.replace(/^/gm, indent.repeat(count));
```

```
indentString('Lorem\nIpsum', 2); // '  Lorem\n  Ipsum'
indentString('Lorem\nIpsum', 2, '_'); // '__Lorem\n__Ipsum'
```

title: indexBy

Creates an object from an array, using a function to map each value to a key.

- Use `Array.prototype.reduce()` to create an object from `arr` .
- Apply `fn` to each value of `arr` to produce a key and add the key-value pair to the object.

```
const indexBy = (arr, fn) =>
  arr.reduce((obj, v, i) => {
    obj[fn(v, i, arr)] = v;
    return obj;
  }, {});
```

```
indexBy([
  { id: 10, name: 'apple' },
  { id: 20, name: 'orange' }
], x => x.id);
// { '10': { id: 10, name: 'apple' }, '20': { id: 20, name: 'orange' } }
```

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������
���������������������������

���---

title: isBeforeDate

Checks if a date is before another date.

- Use the less than operator (`<`) to check if the first date comes before the second one.

```js
const isBeforeDate = (dateA, dateB) => dateA < dateB;


isBeforeDate(new Date(2010, 10, 20), new Date(2010, 10, 21)); // true
```

title: isBetweenDates

Checks if a date is between two other dates.

- Use the greater than (`>`) and less than (`<`) operators to check if `date` is between `dateStart` and `dateEnd` .

```js
const isBetweenDates = (dateStart, dateEnd, date) =>
  date > dateStart && date < dateEnd;


isBetweenDates(
  new Date(2010, 11, 20),
  new Date(2010, 11, 30),
  new Date(2010, 11, 19)
); // false
isBetweenDates(
  new Date(2010, 11, 20),
  new Date(2010, 11, 30),
  new Date(2010, 11, 25)
); // true
```

title: isBoolean

Checks if the given argument is a native boolean element.

- Use `typeof` to check if a value is classified as a boolean primitive.

```
const isBoolean = val => typeof val === 'boolean';
```

```
isBoolean(null); // false
isBoolean(false); // true
```

title: isBrowser

Determines if the current runtime environment is a browser so that front-end modules can run on the server (Node) without throwing errors.

- Use `Array.prototype.includes()` on the `typeof` values of both `window` and `document` (globals usually only available in a browser environment unless they were explicitly defined), which will return `true` if one of them is `undefined`.
- `typeof` allows globals to be checked for existence without throwing a `ReferenceError`.
- If both of them are not `undefined`, then the current environment is assumed to be a browser.

```
const isBrowser = () => ![typeof window, typeof document].includes('undefined');
```

```
isBrowser(); // true (browser)
isBrowser(); // false (Node)
```

title: isBrowserTabFocused

Checks if the browser tab of the page is focused.

- Use the `Document.hidden` property, introduced by the Page Visibility API to check if the browser tab of the page is visible or hidden.

```
const isBrowserTabFocused = () => !document.hidden;


isBrowserTabFocused(); // true
```

title: isContainedIn

Checks if the elements of the first array are contained in the second one regardless of order.

- Use a `for...of` loop over a `Set` created from the first array.
- Use `Array.prototype.some()` to check if all distinct values are contained in the second array.
- Use `Array.prototype.filter()` to compare the number of occurrences of each distinct value in both arrays.
- Return `false` if the count of any element is greater in the first array than the second one, `true` otherwise.

```
const isContainedIn = (a, b) => {
  for (const v of new Set(a)) {
    if (
      !b.some(e => e === v) ||
      a.filter(e => e === v).length > b.filter(e => e === v).length
    )
      return false;
  }
  return true;
};


isContainedIn([1, 4], [2, 4, 1]); // true
```

title: isDateValid

Checks if a valid date object can be created from the given values.

- Use the spread operator (`...`) to pass the array of arguments to the `Date` constructor.
- Use `Date.prototype.valueOf()` and `Number.isNaN()` to check if a valid `Date` object can be created from the given values.

```js
const isDateValid = (...val) => !Number.isNaN(new Date(...val).valueOf());
```

```js
isDateValid('December 17, 1995 03:24:00'); // true
isDateValid('1995-12-17T03:24:00'); // true
isDateValid('1995-12-17 T03:24:00'); // false
isDateValid('Duck'); // false
isDateValid(1995, 11, 17); // true
isDateValid(1995, 11, 17, 'Duck'); // false
isDateValid({}); // false
```

title: isDeepFrozen

Checks if an object is deeply frozen.

- Use recursion.
- Use `Object.isFrozen()` on the given object.
- Use `Object.keys()` , `Array.prototype.every()` to check that all keys are either deeply frozen objects or non-object values.

```js
const isDeepFrozen = obj =>
  Object.isFrozen(obj) &&
  Object.keys(obj).every(
    prop => typeof obj[prop] !== 'object' || isDeepFrozen(obj[prop])
  );
```

```js
const x = Object.freeze({ a: 1 });
const y = Object.freeze({ b: { c: 2 } });
isDeepFrozen(x); // true
isDeepFrozen(y); // false
```

title: isDisjoint

Checks if the two iterables are disjointed (have no common values).

- Use the `new Set()` constructor to create a new `Set` object from each iterable.
- Use `Array.prototype.every()` and `Set.prototype.has()` to check that the two iterables have no common values.

```
const isDisjoint = (a, b) => {
  const sA = new Set(a), sB = new Set(b);
  return [...sA].every(v => !sB.has(v));
};
```

```
isDisjoint(new Set([1, 2]), new Set([3, 4])); // true
isDisjoint(new Set([1, 2]), new Set([1, 3])); // false
```

title: isDivisible

Checks if the first numeric argument is divisible by the second one.

- Use the modulo operator (`%`) to check if the remainder is equal to `0` .

```
const isDivisible = (dividend, divisor) => dividend % divisor === 0;
```

```
isDivisible(6, 3); // true
```

title: isDuplexStream

Checks if the given argument is a duplex (readable and writable) stream.

- Check if the value is different from `null` .
- Use `typeof` to check if a value is of type `object` and the `pipe` property is of type `function` .
- Additionally check if the `typeof` the `_read` , `_write` and `_readableState` , `_writableState` properties are `function` and `object` respectively.

```
const isDuplexStream = val =>
  val !== null &&
  typeof val === 'object' &&
  typeof val.pipe === 'function' &&
  typeof val._read === 'function' &&
  typeof val._readableState === 'object' &&
  typeof val._write === 'function' &&
  typeof val._writableState === 'object';
```

```js
const Stream = require('stream');

isDuplexStream(new Stream.Duplex()); // true
```

title: isEmpty

Checks if the a value is an empty object/collection, has no enumerable properties or is any type that is not considered a collection.

- Check if the provided value is `null` or if its `length` is equal to `0`.

```js
const isEmpty = val => val == null || !(Object.keys(val) || val).length;
```

```js
isEmpty([]); // true
isEmpty({}); // true
isEmpty(''); // true
isEmpty([1, 2]); // false
isEmpty({ a: 1, b: 2 }); // false
isEmpty('text'); // false
isEmpty(123); // true - type is not considered a collection
isEmpty(true); // true - type is not considered a collection
```

title: isEven

Checks if the given number is even.

- Checks whether a number is odd or even using the modulo (`%`) operator.
- Returns `true` if the number is even, `false` if the number is odd.

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

```js
isEven(3); // false
```

title: isFunction

Checks if the given argument is a function.

- Use `typeof` to check if a value is classified as a function primitive.

```
const isFunction = val => typeof val === 'function';


isFunction('x'); // false
isFunction(x => x); // true
```

title: isGeneratorFunction

Checks if the given argument is a generator function.

- Use `Object.prototype.toString()` and `Function.prototype.call()` and check if the result is `'[object GeneratorFunction]'` .

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


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

title: isISOString

Checks if the given string is valid in the simplified extended ISO format (ISO 8601).

- Use `new Date()` to create a date object from the given string.
- Use `Date.prototype.valueOf()` and `Number.isNaN()` to check if the produced date object is valid.
- Use `Date.prototype.toISOString()` to compare the ISO formatted string representation of the date with the original string.

```
const isISOString = val => {
  const d = new Date(val);
  return !Number.isNaN(d.valueOf()) && d.toISOString() === val;
};
```

```
isISOString('2020-10-12T10:10:10.000Z'); // true
isISOString('2020-10-12'); // false
```

title: isLeapYear

Checks if the given `year` is a leap year.

- Use `new Date()`, setting the date to February 29th of the given `year`.
- Use `Date.prototype.getMonth()` to check if the month is equal to `1`.

```
const isLeapYear = year => new Date(year, 1, 29).getMonth() === 1;
```

```
isLeapYear(2019); // false
isLeapYear(2020); // true
```

title: isLocalStorageEnabled

Checks if `localStorage` is enabled.

- Use a `try...catch` block to return `true` if all operations complete successfully, `false` otherwise.
- Use `Storage.setItem()` and `Storage.removeItem()` to test storing and deleting a value in `window.localStorage`.

```
const isLocalStorageEnabled = () => {
  try {
    const key = `__storage__test`;
    window.localStorage.setItem(key, null);
    window.localStorage.removeItem(key);
    return true;
  } catch (e) {
    return false;
  }
};


isLocalStorageEnabled(); // true, if localStorage is accessible
```

title: isLowerCase

Checks if a string is lower case.

- Convert the given string to lower case, using `String.prototype.toLowerCase()` and compare it to the original.

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


isLowerCase('abc'); // true
isLowerCase('a3@$'); // true
isLowerCase('Ab4'); // false
```

title: isNegativeZero

Checks if the given value is equal to negative zero (`-0`).

- Check whether a passed value is equal to `0` and if `1` divided by the value equals `-Infinity` .

```
const isNegativeZero = val => val === 0 && 1 / val === -Infinity;


isNegativeZero(-0); // true
isNegativeZero(0); // false
```

title: isNil

Checks if the specified value is `null` or `undefined`.

- Use the strict equality operator to check if the value of `val` is equal to `null` or `undefined`.

```
const isNil = val => val === undefined || val === null;
```

```
isNil(null); // true
isNil(undefined); // true
isNil(''); // false
```

title: isNode

Determines if the current runtime environment is Node.js.

- Use the `process` global object that provides information about the current Node.js process.
- Check if `process`, `process.versions` and `process.versions.node` are defined.

```
const isNode = () =>
  typeof process !== 'undefined' &&
  !!process.versions &&
  !!process.versions.node;
```

```
isNode(); // true (Node)
isNode(); // false (browser)
```

title: isNull

Checks if the specified value is `null`.

- Use the strict equality operator to check if the value of `val` is equal to `null`.

```
const isNull = val => val === null;
```

```js
isNull(null); // true
```

title: isNumber

Checks if the given argument is a number.

- Use `typeof` to check if a value is classified as a number primitive.
- To safeguard against `NaN` , check if `val === val` (as `NaN` has a `typeof` equal to `number` and is the only value not equal to itself).

```js
const isNumber = val => typeof val === 'number' && val === val;
```

```js
isNumber(1); // true
isNumber('1'); // false
isNumber(NaN); // false
```

title: isObject

Checks if the passed value is an object or not.

- Uses the `Object` constructor to create an object wrapper for the given value.
- If the value is `null` or `undefined` , create and return an empty object.
- Otherwise, return an object of a type that corresponds to the given value.

```js
const isObject = obj => obj === Object(obj);
```

```js
isObject([1, 2, 3, 4]); // true
isObject([]); // true
isObject(['Hello!']); // true
isObject({ a: 1 }); // true
isObject({}); // true
isObject(true); // false
```

title: isObjectLike

Checks if a value is object-like.

- Check if the provided value is not `null` and its `typeof` is equal to `'object'` .

```
const isObjectLike = val => val !== null && typeof val === 'object';
```

```
isObjectLike({}); // true
isObjectLike([1, 2, 3]); // true
isObjectLike(x => x); // false
isObjectLike(null); // false
```

title: isOdd

Checks if the given number is odd.

- Check whether a number is odd or even using the modulo (`%`) operator.
- Return `true` if the number is odd, `false` if the number is even.

```
const isOdd = num => num % 2 === 1;
```

```
isOdd(3); // true
```

title: isPlainObject

Checks if the provided value is an object created by the Object constructor.

- Check if the provided value is truthy.
- Use `typeof` to check if it is an object and `Object.prototype.constructor` to make sure the constructor is equal to `Object` .

```
const isPlainObject = val =>
  !!val && typeof val === 'object' && val.constructor === Object;
```

```
isPlainObject({ a: 1 }); // true
isPlainObject(new Map()); // false
```

title: isPowerOfTen

Checks if the given number is a power of `10`.

- Use `Math.log10()` and the modulo operator (`%`) to determine if `n` is a power of `10`.

```
const isPowerOfTen = n => Math.log10(n) % 1 === 0;
```

```
isPowerOfTen(1); // true
isPowerOfTen(10); // true
isPowerOfTen(20); // false
```

title: isPowerOfTwo

Checks if the given number is a power of `2`.

- Use the bitwise binary AND operator (`&`) to determine if `n` is a power of `2`.
- Additionally, check that `n` is not falsy.

```
const isPowerOfTwo = n => !!n && (n & (n - 1)) == 0;
```

```
isPowerOfTwo(0); // false
isPowerOfTwo(1); // true
isPowerOfTwo(8); // true
```

title: isPrime

Checks if the provided integer is a prime number.

- Check numbers from `2` to the square root of the given number.

- Return `false` if any of them divides the given number, else return `true` , unless the number is less than `2` .

```
const isPrime = num => {
  const boundary = Math.floor(Math.sqrt(num));
  for (let i = 2; i <= boundary; i++) if (num % i === 0) return false;
  return num >= 2;
};


isPrime(11); // true
```

title: isPrimitive

Checks if the passed value is primitive or not.

- Create an object from `val` and compare it with `val` to determine if the passed value is primitive (i.e. not equal to the created object).

```
const isPrimitive = val => Object(val) !== val;


isPrimitive(null); // true
isPrimitive(undefined); // true
isPrimitive(50); // true
isPrimitive('Hello!'); // true
isPrimitive(false); // true
isPrimitive(Symbol()); // true
isPrimitive([]); // false
isPrimitive({}); // false
```

title: isPromiseLike

Checks if an object looks like a `Promise` .

- Check if the object is not `null` , its `typeof` matches either `object` or `function` and if it has a `.then` property, which is also a `function` .

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


isPromiseLike({
  then: function() {
    return '';
  }
}); // true
isPromiseLike(null); // false
isPromiseLike({}); // false
```

title: isReadableStream

Checks if the given argument is a readable stream.

- Check if the value is different from `null` .
- Use `typeof` to check if the value is of type `object` and the `pipe` property is of type `function` .
- Additionally check if the `typeof` the `_read` and `_readableState` properties are `function` and `object` respectively.

```
const isReadableStream = val =>
  val !== null &&
  typeof val === 'object' &&
  typeof val.pipe === 'function' &&
  typeof val._read === 'function' &&
  typeof val._readableState === 'object';


const fs = require('fs');

isReadableStream(fs.createReadStream('test.txt')); // true
```

title: isSameDate

Checks if a date is the same as another date.

- Use `Date.prototype.toISOString()` and strict equality checking (`===`) to check if the first date is the same as the second one.

```
const isSameDate = (dateA, dateB) =>
  dateA.toISOString() === dateB.toISOString();


isSameDate(new Date(2010, 10, 20), new Date(2010, 10, 20)); // true
```

title: isSameOrigin

Checks if two URLs are on the same origin.

- Use `URL.protocol` and `URL.host` to check if both URLs have the same protocol and host.

```
const isSameOrigin = (origin, destination) =>
  origin.protocol === destination.protocol && origin.host === destination.host;


const origin = new URL('https://www.30secondsofcode.org/about');
const destination = new URL('https://www.30secondsofcode.org/contact');
isSameOrigin(origin, destination); // true
const other = new URL('https://developer.mozilla.org);
isSameOrigin(origin, other); // false
```

title: isSessionStorageEnabled

Checks if `sessionStorage` is enabled.

- Use a `try...catch` block to return `true` if all operations complete successfully, `false` otherwise.
- Use `Storage.setItem()` and `Storage.removeItem()` to test storing and deleting a value in `window.sessionStorage` .

```
const isSessionStorageEnabled = () => {
  try {
    const key = `__storage__test`;
    window.sessionStorage.setItem(key, null);
    window.sessionStorage.removeItem(key);
    return true;
  } catch (e) {
    return false;
  }
};
```

```
isSessionStorageEnabled(); // true, if sessionStorage is accessible
```

title: isSorted

Checks if a numeric array is sorted.

- Calculate the ordering `direction` for the first pair of adjacent array elements.
- Return `0` if the given array is empty, only has one element or the `direction` changes for any pair of adjacent array elements.
- Use `Math.sign()` to covert the final value of `direction` to `-1` (descending order) or `1` (ascending order).

```
const isSorted = arr => {
  if (arr.length <= 1) return 0;
  const direction = arr[1] - arr[0];
  for (let i = 2; i < arr.length; i++) {
    if ((arr[i] - arr[i - 1]) * direction < 0) return 0;
  }
  return Math.sign(direction);
};
```

```
isSorted([0, 1, 2, 2]); // 1
isSorted([4, 3, 2]); // -1
isSorted([4, 3, 5]); // 0
isSorted([4]); // 0
```

title: isStream

Checks if the given argument is a stream.

- Check if the value is different from `null`.
- Use `typeof` to check if the value is of type `object` and the `pipe` property is of type `function`.

```
const isStream = val =>
  val !== null && typeof val === 'object' && typeof val.pipe === 'function';
```

```
const fs = require('fs');

isStream(fs.createReadStream('test.txt')); // true
```

title: isString

Checks if the given argument is a string.
Only works for string primitives.

- Use `typeof` to check if a value is classified as a string primitive.

```
const isString = val => typeof val === 'string';
```

```
isString('10'); // true
```

title: isSymbol

Checks if the given argument is a symbol.

- Use `typeof` to check if a value is classified as a symbol primitive.

```
const isSymbol = val => typeof val === 'symbol';
```

```
isSymbol(Symbol('x')); // true
```

title: isTravisCI
unlisted: true

Checks if the current environment is Travis CI.

- Check if the current environment has the `TRAVIS` and `CI` environment variables (reference).

```js
const isTravisCI = () => 'TRAVIS' in process.env && 'CI' in process.env;
```

```js
isTravisCI(); // true (if code is running on Travis CI)
```

title: isUndefined

Checks if the specified value is `undefined`.

- Use the strict equality operator to check if `val` is equal to `undefined`.

```js
const isUndefined = val => val === undefined;
```

```js
isUndefined(undefined); // true
```

title: isUpperCase

Checks if a string is upper case.

- Convert the given string to upper case, using `String.prototype.toUpperCase()` and compare it to the original.

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

```js
isUpperCase('ABC'); // true
isUpperCase('A3@$'); // true
isUpperCase('aB4'); // false
```

title: isValidJSON

Checks if the provided string is a valid JSON.

- Use `JSON.parse()` and a `try... catch` block to check if the provided string is a valid JSON.

```
const isValidJSON = str => {
  try {
    JSON.parse(str);
    return true;
  } catch (e) {
    return false;
  }
};
```

```
isValidJSON('{"name":"Adam","age":20}'); // true
isValidJSON('{"name":"Adam",age:"20"}'); // false
isValidJSON(null); // true
```

title: isWeekday

Checks if the given date is a weekday.

- Use `Date.prototype.getDay()` to check weekday by using a modulo operator (`%`).
- Omit the argument, `d` , to use the current date as default.

```
const isWeekday = (d = new Date()) => d.getDay() % 6 !== 0;
```

```
isWeekday(); // true (if current date is 2019-07-19)
```

title: isWeekend

Checks if the given date is a weekend.

- Use `Date.prototype.getDay()` to check weekend by using a modulo operator (`%`).

- Omit the argument, `d`, to use the current date as default.

```
const isWeekend = (d = new Date()) => d.getDay() % 6 === 0;
```

```
isWeekend(); // 2018-10-19 (if current date is 2018-10-18)
```

title: isWritableStream

Checks if the given argument is a writable stream.

- Check if the value is different from `null`.
- Use `typeof` to check if the value is of type `object` and the `pipe` property is of type `function`.
- Additionally check if the `typeof` the `_write` and `_writableState` properties are `function` and `object` respectively.

```
const isWritableStream = val =>
  val !== null &&
  typeof val === 'object' &&
  typeof val.pipe === 'function' &&
  typeof val._write === 'function' &&
  typeof val._writableState === 'object';
```

```
const fs = require('fs');

isWritableStream(fs.createWriteStream('test.txt')); // true
```

title: join

Joins all elements of an array into a string and returns this string.
Uses a separator and an end separator.

- Use `Array.prototype.reduce()` to combine elements into a string.
- Omit the second argument, `separator`, to use a default separator of `','`.
- Omit the third argument, `end`, to use the same value as `separator` by default.

```
const join = (arr, separator = ',', end = separator) =>
  arr.reduce(
    (acc, val, i) =>
      i === arr.length - 2
        ? acc + val + end
        : i === arr.length - 1
          ? acc + val
          : acc + val + separator,
    ''
  );


join(['pen', 'pineapple', 'apple', 'pen'],',','&'); // 'pen,pineapple,apple&pen'
join(['pen', 'pineapple', 'apple', 'pen'], ','); // 'pen,pineapple,apple,pen'
join(['pen', 'pineapple', 'apple', 'pen']); // 'pen,pineapple,apple,pen'
```

title: juxt

Takes several functions as argument and returns a function that is the juxtaposition of those functions.

- Use `Array.prototype.map()` to return a `fn` that can take a variable number of `args` .
- When `fn` is called, return an array containing the result of applying each `fn` to the `args` .

```
const juxt = (...fns) => (...args) => [...fns].map(fn => [...args].map(fn));


juxt(
  x => x + 1,
  x => x - 1,
  x => x * 10
)(1, 2, 3); // [[2, 3, 4], [0, 1, 2], [10, 20, 30]]
juxt(
  s => s.length,
  s => s.split(' ').join('-')
)('30 seconds of code'); // [[18], ['30-seconds-of-code']]
```

title: kMeans

Groups the given data into `k` clusters, using the k-means clustering algorithm.

- Use `Array.from()` and `Array.prototype.slice()` to initialize appropriate variables for the cluster `centroids`, `distances` and `classes`.
- Use a `while` loop to repeat the assignment and update steps as long as there are changes in the previous iteration, as indicated by `itr`.
- Calculate the euclidean distance between each data point and centroid using `Math.hypot()`, `Object.keys()` and `Array.prototype.map()`.
- Use `Array.prototype.indexOf()` and `Math.min()` to find the closest centroid.
- Use `Array.from()` and `Array.prototype.reduce()`, as well as `parseFloat()` and `Number.prototype.toFixed()` to calculate the new centroids.

```
const kMeans = (data, k = 1) => {
  const centroids = data.slice(0, k);
  const distances = Array.from({ length: data.length }, () =>
    Array.from({ length: k }, () => 0)
  );
  const classes = Array.from({ length: data.length }, () => -1);
  let itr = true;

  while (itr) {
    itr = false;

    for (let d in data) {
      for (let c = 0; c < k; c++) {
        distances[d][c] = Math.hypot(
          ...Object.keys(data[0]).map(key => data[d][key] - centroids[c][key])
        );
      }
      const m = distances[d].indexOf(Math.min(...distances[d]));
      if (classes[d] !== m) itr = true;
      classes[d] = m;
    }

    for (let c = 0; c < k; c++) {
      centroids[c] = Array.from({ length: data[0].length }, () => 0);
      const size = data.reduce((acc, _, d) => {
        if (classes[d] === c) {
          acc++;
          for (let i in data[0]) centroids[c][i] += data[d][i];
        }
        return acc;
      }, 0);
      for (let i in data[0]) {
        centroids[c][i] = parseFloat(Number(centroids[c][i] / size).toFixed(2));
      }
    }
  }

  return classes;
};
```

```
kMeans([[0, 0], [0, 1], [1, 3], [2, 0]], 2); // [0, 1, 1, 0]
```

title: kNearestNeighbors

Classifies a data point relative to a labelled data set, using the k-nearest neighbors algorithm.

- Use `Array.prototype.map()` to map the `data` to objects. Each object contains the euclidean distance of the element from `point`, calculated using `Math.hypot()`, `Object.keys()` and its `label`.
- Use `Array.prototype.sort()` and `Array.prototype.slice()` to get the `k` nearest neighbors of `point`.
- Use `Array.prototype.reduce()` in combination with `Object.keys()` and `Array.prototype.indexOf()` to find the most frequent `label` among them.

```
const kNearestNeighbors = (data, labels, point, k = 3) => {
  const kNearest = data
    .map((el, i) => ({
      dist: Math.hypot(...Object.keys(el).map(key => point[key] - el[key])),
      label: labels[i]
    }))
    .sort((a, b) => a.dist - b.dist)
    .slice(0, k);

  return kNearest.reduce(
    (acc, { label }, i) => {
      acc.classCounts[label] =
        Object.keys(acc.classCounts).indexOf(label) !== -1
          ? acc.classCounts[label] + 1
          : 1;
      if (acc.classCounts[label] > acc.topClassCount) {
        acc.topClassCount = acc.classCounts[label];
        acc.topClass = label;
      }
      return acc;
    },
    {
      classCounts: {},
      topClass: kNearest[0].label,
      topClassCount: 0
    }
  ).topClass;
};


const data = [[0, 0], [0, 1], [1, 3], [2, 0]];
const labels = [0, 1, 1, 0];

kNearestNeighbors(data, labels, [1, 2], 2); // 1
kNearestNeighbors(data, labels, [1, 0], 2); // 0
```

title: kmToMiles
unlisted: true

Converts kilometers to miles.

- Follow the conversion formula `mi = km * 0.621371` .

```
const kmToMiles = km => km * 0.621371;
```

```
kmToMiles(8.1) // 5.0331051
```

title: last

Returns the last element in an array.

- Check if `arr` is truthy and has a `length` property.
- Use `Array.prototype.length - 1` to compute the index of the last element of the given array and return it, otherwise return `undefined` .

```
const last = arr => (arr && arr.length ? arr[arr.length - 1] : undefined);
```

```
last([1, 2, 3]); // 3
last([]); // undefined
last(null); // undefined
last(undefined); // undefined
```

title: lastDateOfMonth

Returns the string representation of the last date in the given date's month.

- Use `Date.prototype.getFullYear()` , `Date.prototype.getMonth()` to get the current year and month from the given date.
- Use the `new Date()` constructor to create a new date with the given year and month incremented by `1` , and the day set to `0` (last day of previous month).

- Omit the argument, `date` , to use the current date by default.

```
const lastDateOfMonth = (date = new Date()) => {
  let d = new Date(date.getFullYear(), date.getMonth() + 1, 0);
  return d.toISOString().split('T')[0];
};


lastDateOfMonth(new Date('2015-08-11')); // '2015-08-30'
```

title: lcm

Calculates the least common multiple of two or more numbers.

- Use the greatest common divisor (GCD) formula and the fact that `lcm(x, y) = x * y / gcd(x, y)` to determine the least common multiple.
- The GCD formula uses recursion.

```
const lcm = (...arr) => {
  const gcd = (x, y) => (!y ? x : gcd(y, x % y));
  const _lcm = (x, y) => (x * y) / gcd(x, y);
  return [...arr].reduce((a, b) => _lcm(a, b));
};


lcm(12, 7); // 84
lcm(...[1, 3, 4, 5]); // 60
```

title: levenshteinDistance

Calculates the difference between two strings, using the Levenshtein distance algorithm.

- If either of the two strings has a `length` of zero, return the `length` of the other one.
- Use a `for` loop to iterate over the letters of the target string and a nested `for` loop to iterate over the letters of the source string.
- Calculate the cost of substituting the letters corresponding to `i - 1` and `j - 1` in the target and source respectively (`0` if they are the same, `1` otherwise).

- Use `Math.min()` to populate each element in the 2D array with the minimum of the cell above incremented by one, the cell to the left incremented by one or the cell to the top left incremented by the previously calculated cost.
- Return the last element of the last row of the produced array.

```
const levenshteinDistance = (s, t) => {
  if (!s.length) return t.length;
  if (!t.length) return s.length;
  const arr = [];
  for (let i = 0; i <= t.length; i++) {
    arr[i] = [i];
    for (let j = 1; j <= s.length; j++) {
      arr[i][j] =
        i === 0
          ? j
          : Math.min(
              arr[i - 1][j] + 1,
              arr[i][j - 1] + 1,
              arr[i - 1][j - 1] + (s[j - 1] === t[i - 1] ? 0 : 1)
            );
    }
  }
  return arr[t.length][s.length];
};
```

```
levenshteinDistance('duck', 'dark'); // 2
```

title: linearSearch

Finds the first index of a given element in an array using the linear search algorithm.

- Use a `for...in` loop to iterate over the indexes of the given array.
- Check if the element in the corresponding index is equal to `item`.
- If the element is found, return the index, using the unary `+` operator to convert it from a string to a number.
- If the element is not found after iterating over the whole array, return `-1`.

```
const linearSearch = (arr, item) => {
  for (const i in arr) {
    if (arr[i] === item) return +i;
  }
  return -1;
};
```

```
linearSearch([2, 9, 9], 9); // 1
linearSearch([2, 9, 9], 7); // -1
```

title: listenOnce

Adds an event listener to an element that will only run the callback the first time the event is triggered.

- Use `EventTarget.addEventListener()` to add an event listener to an element.
- Use `{ once: true }` as options to only run the given callback once.

```
const listenOnce = (el, evt, fn) =>
  el.addEventListener(evt, fn, { once: true });
```

```
listenOnce(
  document.getElementById('my-id'),
  'click',
  () => console.log('Hello world')
); // 'Hello world' will only be logged on the first click
```

title: logBase

Calculates the logarithm of the given number in the given base.

- Use `Math.log()` to get the logarithm from the value and the base and divide them.

```
const logBase = (n, base) => Math.log(n) / Math.log(base);
```

```
logBase(10, 10); // 1
logBase(100, 10); // 2
```

title: longestItem

Takes any number of iterable objects or objects with a `length` property and returns the longest one.

- Use `Array.prototype.reduce()`, comparing the length of objects to find the longest one.
- If multiple objects have the same length, the first one will be returned.
- Returns `undefined` if no arguments are provided.

```
const longestItem = (...vals) =>
  vals.reduce((a, x) => (x.length > a.length ? x : a));


longestItem('this', 'is', 'a', 'testcase'); // 'testcase'
longestItem(...['a', 'ab', 'abc']); // 'abc'
longestItem(...['a', 'ab', 'abc'], 'abcd'); // 'abcd'
longestItem([1, 2, 3], [1, 2], [1, 2, 3, 4, 5]); // [1, 2, 3, 4, 5]
longestItem([1, 2, 3], 'foobar'); // 'foobar'
```

title: lowercaseKeys

Creates a new object from the specified object, where all the keys are in lowercase.

- Use `Object.keys()` and `Array.prototype.reduce()` to create a new object from the specified object.
- Convert each key in the original object to lowercase, using `String.prototype.toLowerCase()`.

```
const lowercaseKeys = obj =>
  Object.keys(obj).reduce((acc, key) => {
    acc[key.toLowerCase()] = obj[key];
    return acc;
  }, {});


const myObj = { Name: 'Adam', sUrnAME: 'Smith' };
const myObjLower = lowercaseKeys(myObj); // {name: 'Adam', surname: 'Smith'};
```

title: luhnCheck

Implements the Luhn Algorithm used to validate a variety of identification numbers, such as credit card numbers, IMEI numbers, National Provider Identifier numbers etc.

- Use `String.prototype.split('')`, `Array.prototype.reverse()` and `Array.prototype.map()` in combination with `parseInt()` to obtain an array of digits.
- Use `Array.prototype.splice(0, 1)` to obtain the last digit.
- Use `Array.prototype.reduce()` to implement the Luhn Algorithm.
- Return `true` if `sum` is divisible by `10`, `false` otherwise.

```
const luhnCheck = num => {
  let arr = (num + '')
    .split('')
    .reverse()
    .map(x => parseInt(x));
  let lastDigit = arr.splice(0, 1)[0];
  let sum = arr.reduce(
    (acc, val, i) => (i % 2 !== 0 ? acc + val : acc + ((val *= 2) > 9 ? val - 9 : val)),
    0
  );
  sum += lastDigit;
  return sum % 10 === 0;
};
```

```
luhnCheck('4485275742308327'); // true
luhnCheck(6011329933655299); //  true
luhnCheck(123456789); // false
```

title: mapConsecutive

Maps each block of `n` consencutive elements using the given function, `fn`.

- Use `Array.prototype.slice()` to get `arr` with `n` elements removed from the left.
- Use `Array.prototype.map()` and `Array.prototype.slice()` to apply `fn` to each block of `n` consecutive elements in `arr`.

```
const mapConsecutive = (arr, n, fn) =>
  arr.slice(n - 1).map((v, i) => fn(arr.slice(i, i + n)));
```

```
mapConsecutive([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 3, x => x.join('-'));
// ['1-2-3', '2-3-4', '3-4-5', '4-5-6', '5-6-7', '6-7-8', '7-8-9', '8-9-10'];
```

title: mapKeys

Maps the keys of an object using the provided function, generating a new object.

- Use `Object.keys()` to iterate over the object's keys.
- Use `Array.prototype.reduce()` to create a new object with the same values and mapped keys using `fn`.

```
const mapKeys = (obj, fn) =>
  Object.keys(obj).reduce((acc, k) => {
    acc[fn(obj[k], k, obj)] = obj[k];
    return acc;
  }, {});
```

```
mapKeys({ a: 1, b: 2 }, (val, key) => key + val); // { a1: 1, b2: 2 }
```

title: mapNumRange

Maps a number from one range to another range.

- Return `num` mapped between `outMin`-`outMax` from `inMin`-`inMax`.

```
const mapNumRange = (num, inMin, inMax, outMin, outMax) =>
  ((num - inMin) * (outMax - outMin)) / (inMax - inMin) + outMin;
```

```
mapNumRange(5, 0, 10, 0, 100); // 50
```

title: mapObject

Maps the values of an array to an object using a function.

- Use `Array.prototype.reduce()` to apply `fn` to each element in `arr` and combine the results into an object.
- Use `el` as the key for each property and the result of `fn` as the value.

```
const mapObject = (arr, fn) =>
  arr.reduce((acc, el, i) => {
    acc[el] = fn(el, i, arr);
    return acc;
  }, {});


mapObject([1, 2, 3], a => a * a); // { 1: 1, 2: 4, 3: 9 }
```

title: mapString

Creates a new string with the results of calling a provided function on every character in the given string.

- Use `String.prototype.split('')` and `Array.prototype.map()` to call the provided function, `fn` , for each character in `str` .
- Use `Array.prototype.join('')` to recombine the array of characters into a string.
- The callback function, `fn` , takes three arguments (the current character, the index of the current character and the string `mapString` was called upon).

```
const mapString = (str, fn) =>
  str
    .split('')
    .map((c, i) => fn(c, i, str))
    .join('');


mapString('lorem ipsum', c => c.toUpperCase()); // 'LOREM IPSUM'
```

title: mapValues

Maps the values of an object using the provided function, generating a new object with the same keys.

- Use `Object.keys()` to iterate over the object's keys.

- Use `Array.prototype.reduce()` to create a new object with the same keys and mapped values using `fn`.

```
const mapValues = (obj, fn) =>
  Object.keys(obj).reduce((acc, k) => {
    acc[k] = fn(obj[k], k, obj);
    return acc;
  }, {});


const users = {
  fred: { user: 'fred', age: 40 },
  pebbles: { user: 'pebbles', age: 1 }
};
mapValues(users, u => u.age); // { fred: 40, pebbles: 1 }
```

title: mask

Replaces all but the last `num` of characters with the specified mask character.

- Use `String.prototype.slice()` to grab the portion of the characters that will remain unmasked.
- Use `String.padStart()` to fill the beginning of the string with the `mask` character up to the original length.
- If `num` is negative, the unmasked characters will be at the start of the string.
- Omit the second argument, `num`, to keep a default of `4` characters unmasked.
- Omit the third argument, `mask`, to use a default character of `'*'` for the mask.

```
const mask = (cc, num = 4, mask = '*') =>
  `${cc}`.slice(-num).padStart(`${cc}`.length, mask);


mask(1234567890); // '******7890'
mask(1234567890, 3); // '*******890'
mask(1234567890, -4, '$'); // '$$$$567890'
```

title: matches

Compares two objects to determine if the first one contains equivalent property values to the second one.

- Use `Object.keys()` to get all the keys of the second object.
- Use `Array.prototype.every()` , `Object.prototype.hasOwnProperty()` and strict comparison to determine if all keys exist in the first object and have the same values.

```
const matches = (obj, source) =>
  Object.keys(source).every(
    key => obj.hasOwnProperty(key) && obj[key] === source[key]
  );
```

```
matches({ age: 25, hair: 'long', beard: true }, { hair: 'long', beard: true });
// true
matches({ hair: 'long', beard: true }, { age: 25, hair: 'long', beard: true });
// false
```

title: matchesWith

Compares two objects to determine if the first one contains equivalent property values to the second one, based on a provided function.

- Use `Object.keys()` to get all the keys of the second object.
- Use `Array.prototype.every()` , `Object.prototype.hasOwnProperty()` and the provided function to determine if all keys exist in the first object and have equivalent values.
- If no function is provided, the values will be compared using the equality operator.

```
const matchesWith = (obj, source, fn) =>
  Object.keys(source).every(key =>
    obj.hasOwnProperty(key) && fn
      ? fn(obj[key], source[key], key, obj, source)
      : obj[key] == source[key]
  );
```

```
const isGreeting = val => /^h(?:i|ello)$/.test(val);
matchesWith(
  { greeting: 'hello' },
  { greeting: 'hi' },
  (oV, sV) => isGreeting(oV) && isGreeting(sV)
); // true
```

title: maxBy

Returns the maximum value of an array, after mapping each element to a value using the provided function.

- Use `Array.prototype.map()` to map each element to the value returned by `fn`.
- Use `Math.max()` to get the maximum value.

```
const maxBy = (arr, fn) =>
  Math.max(...arr.map(typeof fn === 'function' ? fn : val => val[fn]));
```

```
maxBy([{ n: 4 }, { n: 2 }, { n: 8 }, { n: 6 }], x => x.n); // 8
maxBy([{ n: 4 }, { n: 2 }, { n: 8 }, { n: 6 }], 'n'); // 8
```

title: maxDate

Returns the maximum of the given dates.

- Use the ES6 spread syntax with `Math.max()` to find the maximum date value.
- Use `new Date()` to convert it to a `Date` object.

```
const maxDate = (...dates) => new Date(Math.max(...dates));
```

```
const dates = [
  new Date(2017, 4, 13),
  new Date(2018, 2, 12),
  new Date(2016, 0, 10),
  new Date(2016, 0, 9)
];
maxDate(...dates); // 2018-03-11T22:00:00.000Z
```

title: maxN

Returns the `n` maximum elements from the provided array.

- Use `Array.prototype.sort()` combined with the spread operator (`...`) to create a shallow clone of the array and sort it in descending order.
- Use `Array.prototype.slice()` to get the specified number of elements.
- Omit the second argument, `n` , to get a one-element array.
- If `n` is greater than or equal to the provided array's length, then return the original array (sorted in descending order).

```
const maxN = (arr, n = 1) => [...arr].sort((a, b) => b - a).slice(0, n);
```

```
maxN([1, 2, 3]); // [3]
maxN([1, 2, 3], 2); // [3, 2]
```

title: median

Calculates the median of an array of numbers.

- Find the middle of the array, use `Array.prototype.sort()` to sort the values.
- Return the number at the midpoint if `Array.prototype.length` is odd, otherwise the average of the two middle numbers.

```
const median = arr => {
  const mid = Math.floor(arr.length / 2),
    nums = [...arr].sort((a, b) => a - b);
  return arr.length % 2 !== 0 ? nums[mid] : (nums[mid - 1] + nums[mid]) / 2;
};
```

```
median([5, 6, 50, 1, -5]); // 5
```

title: memoize

Returns the memoized (cached) function.

- Create an empty cache by instantiating a new `Map` object.
- Return a function which takes a single argument to be supplied to the memoized function by first checking if the function's output for that specific input value is already cached, or store and return

it if not.
- The `function` keyword must be used in order to allow the memoized function to have its `this` context changed if necessary.
- Allow access to the `cache` by setting it as a property on the returned function.

```
const memoize = fn => {
  const cache = new Map();
  const cached = function (val) {
    return cache.has(val)
      ? cache.get(val)
      : cache.set(val, fn.call(this, val)) && cache.get(val);
  };
  cached.cache = cache;
  return cached;
};
```

```
// See the `anagrams` snippet.
const anagramsCached = memoize(anagrams);
anagramsCached('javascript'); // takes a long time
anagramsCached('javascript'); // returns virtually instantly since it's cached
console.log(anagramsCached.cache); // The cached anagrams map
```

title: merge

Creates a new object from the combination of two or more objects.

- Use `Array.prototype.reduce()` combined with `Object.keys()` to iterate over all objects and keys.
- Use `Object.prototype.hasOwnProperty()` and `Array.prototype.concat()` to append values for keys existing in multiple objects.

```
const merge = (...objs) =>
  [...objs].reduce(
    (acc, obj) =>
      Object.keys(obj).reduce((a, k) => {
        acc[k] = acc.hasOwnProperty(k)
          ? [].concat(acc[k]).concat(obj[k])
          : obj[k];
        return acc;
      }, {}),
    {}
  );
```

```
const object = {
  a: [{ x: 2 }, { y: 4 }],
  b: 1
};
const other = {
  a: { z: 3 },
  b: [2, 3],
  c: 'foo'
};
merge(object, other);
// { a: [ { x: 2 }, { y: 4 }, { z: 3 } ], b: [ 1, 2, 3 ], c: 'foo' }
```

title: mergeSort

Sorts an array of numbers, using the merge sort algorithm.

- Use recursion.
- If the `length` of the array is less than `2`, return the array.
- Use `Math.floor()` to calculate the middle point of the array.
- Use `Array.prototype.slice()` to slice the array in two and recursively call `mergeSort()` on the created subarrays.
- Finally, use `Array.from()` and `Array.prototype.shift()` to combine the two sorted subarrays into one.

```
const mergeSort = arr => {
  if (arr.length < 2) return arr;
  const mid = Math.floor(arr.length / 2);
  const l = mergeSort(arr.slice(0, mid));
  const r = mergeSort(arr.slice(mid, arr.length));
  return Array.from({ length: l.length + r.length }, () => {
    if (!l.length) return r.shift();
    else if (!r.length) return l.shift();
    else return l[0] > r[0] ? r.shift() : l.shift();
  });
};
```

```
mergeSort([5, 1, 4, 2, 3]); // [1, 2, 3, 4, 5]
```

title: mergeSortedArrays

Merges two sorted arrays into one.

- Use the spread operator (`...`) to clone both of the given arrays.
- Use `Array.from()` to create an array of the appropriate length based on the given arrays.
- Use `Array.prototype.shift()` to populate the newly created array from the removed elements of the cloned arrays.

```
const mergeSortedArrays = (a, b) => {
  const _a = [...a],
    _b = [...b];
  return Array.from({ length: _a.length + _b.length }, () => {
    if (!_a.length) return _b.shift();
    else if (!_b.length) return _a.shift();
    else return _a[0] > _b[0] ? _b.shift() : _a.shift();
  });
};
```

```
mergeSortedArrays([1, 4, 5], [2, 3, 6]); // [1, 2, 3, 4, 5, 6]
```

title: midpoint

Calculates the midpoint between two pairs of (x,y) points.

- Destructure the array to get `x1` , `y1` , `x2` and `y2` .
- Calculate the midpoint for each dimension by dividing the sum of the two endpoints by `2` .

```
const midpoint = ([x1, y1], [x2, y2]) => [(x1 + x2) / 2, (y1 + y2) / 2];
```

```
midpoint([2, 2], [4, 4]); // [3, 3]
midpoint([4, 4], [6, 6]); // [5, 5]
midpoint([1, 3], [2, 4]); // [1.5, 3.5]
```

title: milesToKm
unlisted: true

Converts miles to kilometers.

- Follow the conversion formula `km = mi * 1.609344` .

```
const milesToKm = miles => miles * 1.609344;


milesToKm(5); // ~8.04672
```

title: minBy

Returns the minimum value of an array, after mapping each element to a value using the provided function.

- Use `Array.prototype.map()` to map each element to the value returned by `fn` .
- Use `Math.min()` to get the minimum value.

```
const minBy = (arr, fn) =>
  Math.min(...arr.map(typeof fn === 'function' ? fn : val => val[fn]));


minBy([{ n: 4 }, { n: 2 }, { n: 8 }, { n: 6 }], x => x.n); // 2
minBy([{ n: 4 }, { n: 2 }, { n: 8 }, { n: 6 }], 'n'); // 2
```

title: minDate

Returns the minimum of the given dates.

- Use the ES6 spread syntax with `Math.min()` to find the minimum date value.
- Use `new Date()` to convert it to a `Date` object.

```
const minDate = (...dates) => new Date(Math.min(...dates));
const dates = [
  new Date(2017, 4, 13),
  new Date(2018, 2, 12),
  new Date(2016, 0, 10),
  new Date(2016, 0, 9)
];
minDate(...dates); // 2016-01-08T22:00:00.000Z
```

title: minN

Returns the `n` minimum elements from the provided array.

- Use `Array.prototype.sort()` combined with the spread operator (`...`) to create a shallow clone of the array and sort it in ascending order.
- Use `Array.prototype.slice()` to get the specified number of elements.
- Omit the second argument, `n` , to get a one-element array.
- If `n` is greater than or equal to the provided array's length, then return the original array (sorted in ascending order).

```
const minN = (arr, n = 1) => [...arr].sort((a, b) => a - b).slice(0, n);
```

```
minN([1, 2, 3]); // [1]
minN([1, 2, 3], 2); // [1, 2]
```

title: mostFrequent

Returns the most frequent element in an array.

- Use `Array.prototype.reduce()` to map unique values to an object's keys, adding to existing keys every time the same value is encountered.
- Use `Object.entries()` on the result in combination with `Array.prototype.reduce()` to get the most frequent value in the array.

```
const mostFrequent = arr =>
  Object.entries(
    arr.reduce((a, v) => {
      a[v] = a[v] ? a[v] + 1 : 1;
      return a;
    }, {})
  ).reduce((a, v) => (v[1] >= a[1] ? v : a), [null, 0])[0];


mostFrequent(['a', 'b', 'a', 'c', 'a', 'a', 'b']); // 'a'
```

title: mostPerformant

Returns the index of the function in an array of functions which executed the fastest.

- Use `Array.prototype.map()` to generate an array where each value is the total time taken to execute the function after `iterations` times.
- Use the difference in `performance.now()` values before and after to get the total time in milliseconds to a high degree of accuracy.
- Use `Math.min()` to find the minimum execution time, and return the index of that shortest time which corresponds to the index of the most performant function.
- Omit the second argument, `iterations` , to use a default of `10000` iterations.
- The more iterations, the more reliable the result but the longer it will take.

```
const mostPerformant = (fns, iterations = 10000) => {
  const times = fns.map(fn => {
    const before = performance.now();
    for (let i = 0; i < iterations; i++) fn();
    return performance.now() - before;
  });
  return times.indexOf(Math.min(...times));
};
```

```
mostPerformant([
  () => {
    // Loops through the entire array before returning `false`
    [1, 2, 3, 4, 5, 6, 7, 8, 9, '10'].every(el => typeof el === 'number');
  },
  () => {
    // Only needs to reach index `1` before returning `false`
    [1, '2', 3, 4, 5, 6, 7, 8, 9, 10].every(el => typeof el === 'number');
  }
]); // 1
```

title: negate

Negates a predicate function.

- Take a predicate function and apply the not operator (`!`) to it with its arguments.

```
const negate = func => (...args) => !func(...args);
```

```
[1, 2, 3, 4, 5, 6].filter(negate(n => n % 2 === 0)); // [ 1, 3, 5 ]
```

title: nest

Nests recursively objects linked to one another in a flat array.

- Use recursion.
- Use `Array.prototype.filter()` to filter the items where the `id` matches the `link` .
- Use `Array.prototype.map()` to map each item to a new object that has a `children` property which recursively nests the items based on which ones are children of the current item.
- Omit the second argument, `id` , to default to `null` which indicates the object is not linked to another one (i.e. it is a top level object).
- Omit the third argument, `link` , to use `'parent_id'` as the default property which links the object to another one by its `id` .

```
const nest = (items, id = null, link = 'parent_id') =>
  items
    .filter(item => item[link] === id)
    .map(item => ({ ...item, children: nest(items, item.id, link) }));
```

```
const comments = [
  { id: 1, parent_id: null },
  { id: 2, parent_id: 1 },
  { id: 3, parent_id: 1 },
  { id: 4, parent_id: 2 },
  { id: 5, parent_id: 4 }
];
const nestedComments = nest(comments);
// [{ id: 1, parent_id: null, children: [...] }]
```

title: nodeListToArray

Converts a `NodeList` to an array.

- Use spread operator (`...`) inside new array to convert a `NodeList` to an array.

```
const nodeListToArray = nodeList => [...nodeList];
```

```
nodeListToArray(document.childNodes); // [ <!DOCTYPE html>, html ]
```

title: none

Checks if the provided predicate function returns `false` for all elements in a collection.

- Use `Array.prototype.some()` to test if any elements in the collection return `true` based on `fn` .
- Omit the second argument, `fn` , to use `Boolean` as a default.

```
const none = (arr, fn = Boolean) => !arr.some(fn);
```

```
none([0, 1, 3, 0], x => x == 2); // true
none([0, 0, 0]); // true
```

title: nor
unlisted: true

Checks if none of the arguments are `true`.

- Use the logical not (`!`) operator to return the inverse of the logical or (`||`) of the two given values.

```
const nor = (a, b) => !(a||b);
```

```
nor(true, true); // false
nor(true, false); // false
nor(false, false); // true
```

title: normalizeLineEndings

Normalizes line endings in a string.

- Use `String.prototype.replace()` and a regular expression to match and replace line endings with the `normalized` version.
- Omit the second argument, `normalized`, to use the default value of `'\r\n'`.

```
const normalizeLineEndings = (str, normalized = '\r\n') =>
  str.replace(/\r?\n/g, normalized);
```

```
normalizeLineEndings('This\r\nis a\nmultiline\nstring.\r\n');
// 'This\r\nis a\r\nmultiline\r\nstring.\r\n'
normalizeLineEndings('This\r\nis a\nmultiline\nstring.\r\n', '\n');
// 'This\nis a\nmultiline\nstring.\n'
```

title: not
unlisted: true

Returns the logical inverse of the given value.

- Use the logical not (`!`) operator to return the inverse of the given value.

```
const not = a => !a;


not(true); // false
not(false); // true
```

title: nthArg

Creates a function that gets the argument at index `n` .

- Use `Array.prototype.slice()` to get the desired argument at index `n` .
- If `n` is negative, the nth argument from the end is returned.

```
const nthArg = n => (...args) => args.slice(n)[0];


const third = nthArg(2);
third(1, 2, 3); // 3
third(1, 2); // undefined
const last = nthArg(-1);
last(1, 2, 3, 4, 5); // 5
```

title: nthElement

Returns the nth element of an array.

- Use `Array.prototype.slice()` to get an array containing the nth element at the first place.
- If the index is out of bounds, return `undefined` .
- Omit the second argument, `n` , to get the first element of the array.

```
const nthElement = (arr, n = 0) =>
  (n === -1 ? arr.slice(n) : arr.slice(n, n + 1))[0];
```

```js
nthElement(['a', 'b', 'c'], 1); // 'b'
nthElement(['a', 'b', 'b'], -3); // 'a'
```

title: nthRoot

Calculates the nth root of a given number.

- Use `Math.pow()` to calculate `x` to the power of `1/n` which is equal to the nth root of `x` .

```js
const nthRoot = (x, n) => Math.pow(x, 1 / n);
```

```js
nthRoot(32, 5); // 2
```

title: objectFromPairs

Creates an object from the given key-value pairs.

- Use `Array.prototype.reduce()` to create and combine key-value pairs.

```js
const objectFromPairs = arr =>
  arr.reduce((a, [key, val]) => ((a[key] = val), a), {});
```

```js
objectFromPairs([['a', 1], ['b', 2]]); // {a: 1, b: 2}
```

title: objectToEntries

Creates an array of key-value pair arrays from an object.

- Use `Object.keys()` and `Array.prototype.map()` to iterate over the object's keys and produce an array with key-value pairs.

```js
const objectToEntries = obj => Object.keys(obj).map(k => [k, obj[k]]);
```

```js
objectToEntries({ a: 1, b: 2 }); // [ ['a', 1], ['b', 2] ]
```

title: objectToPairs

Creates an array of key-value pair arrays from an object.

- Use `Object.entries()` to get an array of key-value pair arrays from the given object.

```js
const objectToPairs = obj => Object.entries(obj);
```

```js
objectToPairs({ a: 1, b: 2 }); // [ ['a', 1], ['b', 2] ]
```

title: objectToQueryString

Generates a query string from the key-value pairs of the given object.

- Use `Array.prototype.reduce()` on `Object.entries(queryParameters)` to create the query string.
- Determine the `symbol` to be either `?` or `&` based on the length of `queryString` .
- Concatenate `val` to `queryString` only if it's a string.
- Return the `queryString` or an empty string when the `queryParameters` are falsy.

```js
const objectToQueryString = queryParameters => {
  return queryParameters
    ? Object.entries(queryParameters).reduce(
        (queryString, [key, val], index) => {
          const symbol = queryString.length === 0 ? '?' : '&';
          queryString +=
            typeof val === 'string' ? `${symbol}${key}=${val}` : '';
          return queryString;
        },
        ''
      )
    : '';
};
```

```js
objectToQueryString({ page: '1', size: '2kg', key: undefined });
// '?page=1&size=2kg'
```

title: observeMutations

Creates a new `MutationObserver` and runs the provided callback for each mutation on the specified element.

- Use a `MutationObserver` to observe mutations on the given element.
- Use `Array.prototype.forEach()` to run the callback for each mutation that is observed.
- Omit the third argument, `options`, to use the default options (all `true`).

```
const observeMutations = (element, callback, options) => {
  const observer = new MutationObserver(mutations =>
    mutations.forEach(m => callback(m))
  );
  observer.observe(
    element,
    Object.assign(
      {
        childList: true,
        attributes: true,
        attributeOldValue: true,
        characterData: true,
        characterDataOldValue: true,
        subtree: true,
      },
      options
    )
  );
  return observer;
};
```

```
const obs = observeMutations(document, console.log);
// Logs all mutations that happen on the page
obs.disconnect();
// Disconnects the observer and stops logging mutations on the page
```

title: off

Removes an event listener from an element.

- Use `EventTarget.removeEventListener()` to remove an event listener from an element.

- Omit the fourth argument `opts` to use `false` or specify it based on the options used when the event listener was added.

```
const off = (el, evt, fn, opts = false) =>
  el.removeEventListener(evt, fn, opts);
```

```
const fn = () => console.log('!');
document.body.addEventListener('click', fn);
off(document.body, 'click', fn); // no longer logs '!' upon clicking on the page
```

title: offset

Moves the specified amount of elements to the end of the array.

- Use `Array.prototype.slice()` twice to get the elements after the specified index and the elements before that.
- Use the spread operator (`...`) to combine the two into one array.
- If `offset` is negative, the elements will be moved from end to start.

```
const offset = (arr, offset) => [...arr.slice(offset), ...arr.slice(0, offset)];
```

```
offset([1, 2, 3, 4, 5], 2); // [3, 4, 5, 1, 2]
offset([1, 2, 3, 4, 5], -2); // [4, 5, 1, 2, 3]
```

title: omit

Omits the key-value pairs corresponding to the given keys from an object.

- Use `Object.keys()` , `Array.prototype.filter()` and `Array.prototype.includes()` to remove the provided keys.
- Use `Array.prototype.reduce()` to convert the filtered keys back to an object with the corresponding key-value pairs.

```
const omit = (obj, arr) =>
  Object.keys(obj)
    .filter(k => !arr.includes(k))
    .reduce((acc, key) => ((acc[key] = obj[key]), acc), {});


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

title: omitBy

Omits the key-value pairs corresponding to the keys of the object for which the given function returns falsy.

- Use `Object.keys()` and `Array.prototype.filter()` to remove the keys for which `fn` returns a truthy value.
- Use `Array.prototype.reduce()` to convert the filtered keys back to an object with the corresponding key-value pairs.
- The callback function is invoked with two arguments: (value, key).

```
const omitBy = (obj, fn) =>
  Object.keys(obj)
    .filter(k => !fn(obj[k], k))
    .reduce((acc, key) => ((acc[key] = obj[key]), acc), {});


omitBy({ a: 1, b: '2', c: 3 }, x => typeof x === 'number'); // { b: '2' }
```

title: on

Adds an event listener to an element with the ability to use event delegation.

- Use `EventTarget.addEventListener()` to add an event listener to an element.
- If there is a `target` property supplied to the options object, ensure the event target matches the target specified and then invoke the callback by supplying the correct `this` context.
- Omit `opts` to default to non-delegation behavior and event bubbling.
- Returns a reference to the custom delegator function, in order to be possible to use with `off` .

```
const on = (el, evt, fn, opts = {}) => {
  const delegatorFn = e =>
    e.target.matches(opts.target) && fn.call(e.target, e);
  el.addEventListener(
    evt,
    opts.target ? delegatorFn : fn,
    opts.options || false
  );
  if (opts.target) return delegatorFn;
};
```

```
const fn = () => console.log('!');
on(document.body, 'click', fn); // logs '!' upon clicking the body
on(document.body, 'click', fn, { target: 'p' });
// logs '!' upon clicking a `p` element child of the body
on(document.body, 'click', fn, { options: true });
// use capturing instead of bubbling
```

title: onClickOutside

Runs the callback whenever the user clicks outside of the specified element.

- Use `EventTarget.addEventListener()` to listen for `'click'` events.
- Use `Node.contains()` to check if `Event.target` is a descendant of `element` and run `callback` if not.

```
const onClickOutside = (element, callback) => {
  document.addEventListener('click', e => {
    if (!element.contains(e.target)) callback();
  });
};
```

```
onClickOutside('#my-element', () => console.log('Hello'));
// Will log 'Hello' whenever the user clicks outside of #my-element
```

title: onScrollStop

Runs the callback whenever the user has stopped scrolling.

- Use `EventTarget.addEventListener()` to listen for the `'scroll'` event.
- Use `setTimeout()` to wait `150` ms until calling the given `callback`.
- Use `clearTimeout()` to clear the timeout if a new `'scroll'` event is fired in under `150` ms.

```
const onScrollStop = callback => {
  let isScrolling;
  window.addEventListener(
    'scroll',
    e => {
      clearTimeout(isScrolling);
      isScrolling = setTimeout(() => {
        callback();
      }, 150);
    },
    false
  );
};


onScrollStop(() => {
  console.log('The user has stopped scrolling');
});
```

title: onUserInputChange

Runs the callback whenever the user input type changes (`mouse` or `touch`).

- Use two event listeners.
- Assume `mouse` input initially and bind a `'touchstart'` event listener to the document.
- On `'touchstart'` , add a `'mousemove'` event listener to listen for two consecutive `'mousemove'` events firing within 20ms, using `performance.now()` .
- Run the callback with the input type as an argument in either of these situations.

```
const onUserInputChange = callback => {
  let type = 'mouse',
    lastTime = 0;
  const mousemoveHandler = () => {
    const now = performance.now();
    if (now - lastTime < 20)
      (type = 'mouse'),
        callback(type),
        document.removeEventListener('mousemove', mousemoveHandler);
    lastTime = now;
  };
  document.addEventListener('touchstart', () => {
    if (type === 'touch') return;
    (type = 'touch'),
      callback(type),
      document.addEventListener('mousemove', mousemoveHandler);
  });
};


onUserInputChange(type => {
  console.log('The user is now using', type, 'as an input method.');
});
```

title: once

Ensures a function is called only once.

- Utilizing a closure, use a flag, `called` , and set it to `true` once the function is called for the first time, preventing it from being called again.
- In order to allow the function to have its `this` context changed (such as in an event listener), the `function` keyword must be used, and the supplied function must have the context applied.
- Allow the function to be supplied with an arbitrary number of arguments using the rest/spread (`...`) operator.

```
const once = fn => {
  let called = false;
  return function(...args) {
    if (called) return;
    called = true;
    return fn.apply(this, args);
  };
};
```

```
const startApp = function(event) {
  console.log(this, event); // document.body, MouseEvent
};
document.body.addEventListener('click', once(startApp));
// only runs `startApp` once upon click
```

title: or
unlisted: true

Checks if at least one of the arguments is `true` .

- Use the logical or (`||`) operator on the two given values.

```
const or = (a, b) => a || b;
```

```
or(true, true); // true
or(true, false); // true
or(false, false); // false
```

title: orderBy

Sorts an array of objects, ordered by properties and orders.

- Uses `Array.prototype.sort()` , `Array.prototype.reduce()` on the `props` array with a default value of `0` .
- Use array destructuring to swap the properties position depending on the order supplied.
- If no `orders` array is supplied, sort by `'asc'` by default.

```
const orderBy = (arr, props, orders) =>
  [...arr].sort((a, b) =>
    props.reduce((acc, prop, i) => {
      if (acc === 0) {
        const [p1, p2] =
          orders && orders[i] === 'desc'
            ? [b[prop], a[prop]]
            : [a[prop], b[prop]];
        acc = p1 > p2 ? 1 : p1 < p2 ? -1 : 0;
      }
      return acc;
    }, 0)
  );


const users = [
  { name: 'fred', age: 48 },
  { name: 'barney', age: 36 },
  { name: 'fred', age: 40 },
];
orderBy(users, ['name', 'age'], ['asc', 'desc']);
// [{name: 'barney', age: 36}, {name: 'fred', age: 48}, {name: 'fred', age: 40}]
orderBy(users, ['name', 'age']);
// [{name: 'barney', age: 36}, {name: 'fred', age: 40}, {name: 'fred', age: 48}]
```

title: orderWith

Sorts an array of objects, ordered by a property, based on the array of orders provided.

- Use `Array.prototype.reduce()` to create an object from the `order` array with the values as keys and their original index as the value.
- Use `Array.prototype.sort()` to sort the given array, skipping elements for which `prop` is empty or not in the `order` array.

```js
const orderWith = (arr, prop, order) => {
  const orderValues = order.reduce((acc, v, i) => {
    acc[v] = i;
    return acc;
  }, {});
  return [...arr].sort((a, b) => {
    if (orderValues[a[prop]] === undefined) return 1;
    if (orderValues[b[prop]] === undefined) return -1;
    return orderValues[a[prop]] - orderValues[b[prop]];
  });
};


const users = [
  { name: 'fred', language: 'Javascript' },
  { name: 'barney', language: 'TypeScript' },
  { name: 'frannie', language: 'Javascript' },
  { name: 'anna', language: 'Java' },
  { name: 'jimmy' },
  { name: 'nicky', language: 'Python' },
];
orderWith(users, 'language', ['Javascript', 'TypeScript', 'Java']);
/*
[
  { name: 'fred', language: 'Javascript' },
  { name: 'frannie', language: 'Javascript' },
  { name: 'barney', language: 'TypeScript' },
  { name: 'anna', language: 'Java' },
  { name: 'jimmy' },
  { name: 'nicky', language: 'Python' }
]
*/
```

title: over

Creates a function that invokes each provided function with the arguments it receives and returns the results.

- Use `Array.prototype.map()` and `Function.prototype.apply()` to apply each function to the given arguments.

```js
const over = (...fns) => (...args) => fns.map(fn => fn.apply(null, args));
const minMax = over(Math.min, Math.max);
minMax(1, 2, 3, 4, 5); // [1, 5]
```

title: overArgs

Creates a function that invokes the provided function with its arguments transformed.

- Use `Array.prototype.map()` to apply `transforms` to `args` in combination with the spread operator (`...`) to pass the transformed arguments to `fn` .

```js
const overArgs = (fn, transforms) =>
  (...args) => fn(...args.map((val, i) => transforms[i](val)));
```

```js
const square = n => n * n;
const double = n => n * 2;
const fn = overArgs((x, y) => [x, y], [square, double]);
fn(9, 3); // [81, 6]
```

title: pad

Pads a string on both sides with the specified character, if it's shorter than the specified `length` .

- Use `String.prototype.padStart()` and `String.prototype.padEnd()` to pad both sides of the given string.
- Omit the third argument, `char` , to use the whitespace character as the default padding character.

```js
const pad = (str, length, char = ' ') =>
  str.padStart((str.length + length) / 2, char).padEnd(length, char);
```

```js
pad('cat', 8); // '  cat   '
pad(String(42), 6, '0'); // '004200'
pad('foobar', 3); // 'foobar'
```

title: padNumber

Pads a given number to the specified length.

- Use `String.prototype.padStart()` to pad the number to specified length, after converting it to a string.

```
const padNumber = (n, l) => `${n}`.padStart(l, '0');


padNumber(1234, 6); // '001234'
```

title: palindrome

Checks if the given string is a palindrome.

- Normalize the string to `String.prototype.toLowerCase()` and use `String.prototype.replace()` to remove non-alphanumeric characters from it.
- Use the spread operator (`...`) to split the normalized string into individual characters.
- Use `Array.prototype.reverse()` , `String.prototype.join('')` and compare the result to the normalized string.

```
const palindrome = str => {
  const s = str.toLowerCase().replace(/[\W_]/g, '');
  return s === [...s].reverse().join('');
};


palindrome('taco cat'); // true
```

title: parseCookie

Parses an HTTP Cookie header string, returning an object of all cookie name-value pairs.

- Use `String.prototype.split(';')` to separate key-value pairs from each other.
- Use `Array.prototype.map()` and `String.prototype.split('=')` to separate keys from values in each pair.

- Use `Array.prototype.reduce()` and `decodeURIComponent()` to create an object with all key-value pairs.

```
const parseCookie = str =>
  str
    .split(';')
    .map(v => v.split('='))
    .reduce((acc, v) => {
      acc[decodeURIComponent(v[0].trim())] = decodeURIComponent(v[1].trim());
      return acc;
    }, {});


parseCookie('foo=bar; equation=E%3Dmc%5E2');
// { foo: 'bar', equation: 'E=mc^2' }
```

title: partial

Creates a function that invokes `fn` with `partials` prepended to the arguments it receives.

- Use the spread operator (`...`) to prepend `partials` to the list of arguments of `fn` .

```
const partial = (fn, ...partials) => (...args) => fn(...partials, ...args);


const greet = (greeting, name) => greeting + ' ' + name + '!';
const greetHello = partial(greet, 'Hello');
greetHello('John'); // 'Hello John!'
```

title: partialRight

Creates a function that invokes `fn` with `partials` appended to the arguments it receives.

- Use the spread operator (`...`) to append `partials` to the list of arguments of `fn` .

```
const partialRight = (fn, ...partials) => (...args) => fn(...args, ...partials);
const greet = (greeting, name) => greeting + ' ' + name + '!';
const greetJohn = partialRight(greet, 'John');
greetJohn('Hello'); // 'Hello John!'
```

title: partition

Groups the elements into two arrays, depending on the provided function's truthiness for each element.

- Use `Array.prototype.reduce()` to create an array of two arrays.
- Use `Array.prototype.push()` to add elements for which `fn` returns `true` to the first array and elements for which `fn` returns `false` to the second one.

```
const partition = (arr, fn) =>
  arr.reduce(
    (acc, val, i, arr) => {
      acc[fn(val, i, arr) ? 0 : 1].push(val);
      return acc;
    },
    [[], []]
  );
```

```
const users = [
  { user: 'barney', age: 36, active: false },
  { user: 'fred', age: 40, active: true },
];
partition(users, o => o.active);
// [
//   [{ user: 'fred', age: 40, active: true }],
//   [{ user: 'barney', age: 36, active: false }]
// ]
```

title: partitionBy

Applies `fn` to each value in `arr`, splitting it each time the provided function returns a new value.

- Use `Array.prototype.reduce()` with an accumulator object that will hold the resulting array and the last value returned from `fn`.

- Use `Array.prototype.push()` to add each value in `arr` to the appropriate partition in the accumulator array.

```
const partitionBy = (arr, fn) =>
  arr.reduce(
    ({ res, last }, v, i, a) => {
      const next = fn(v, i, a);
      if (next !== last) res.push([v]);
      else res[res.length - 1].push(v);
      return { res, last: next };
    },
    { res: [] }
  ).res;


const numbers = [1, 1, 3, 3, 4, 5, 5, 5];
partitionBy(numbers, n => n % 2 === 0); // [[1, 1, 3, 3], [4], [5, 5, 5]]
partitionBy(numbers, n => n); // [[1, 1], [3, 3], [4], [5, 5, 5]]
```

title: percentile

Calculates the percentage of numbers in the given array that are less or equal to the given value.

- Use `Array.prototype.reduce()` to calculate how many numbers are below the value and how many are the same value and apply the percentile formula.

```
const percentile = (arr, val) =>
  (100 *
    arr.reduce(
      (acc, v) => acc + (v < val ? 1 : 0) + (v === val ? 0.5 : 0),
      0
    )) /
  arr.length;


percentile([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 6); // 55
```

title: permutations

Generates all permutations of an array's elements (contains duplicates).

- Use recursion.
- For each element in the given array, create all the partial permutations for the rest of its elements.
- Use `Array.prototype.map()` to combine the element with each partial permutation, then `Array.prototype.reduce()` to combine all permutations in one array.
- Base cases are for `Array.prototype.length` equal to `2` or `1`.
- ⚠ **WARNING**: This function's execution time increases exponentially with each array element. Anything more than 8 to 10 entries may cause your browser to hang as it tries to solve all the different combinations.

```
const permutations = arr => {
  if (arr.length <= 2) return arr.length === 2 ? [arr, [arr[1], arr[0]]] : arr;
  return arr.reduce(
    (acc, item, i) =>
      acc.concat(
        permutations([...arr.slice(0, i), ...arr.slice(i + 1)]).map(val => [
          item,
          ...val,
        ])
      ),
    []
  );
};
```

```
permutations([1, 33, 5]);
// [ [1, 33, 5], [1, 5, 33], [33, 1, 5], [33, 5, 1], [5, 1, 33], [5, 33, 1] ]
```

title: pick

Picks the key-value pairs corresponding to the given keys from an object.

- Use `Array.prototype.reduce()` to convert the filtered/picked keys back to an object with the corresponding key-value pairs if the key exists in the object.

```
const pick = (obj, arr) =>
  arr.reduce((acc, curr) => (curr in obj && (acc[curr] = obj[curr]), acc), {});
```

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

title: pickBy

Creates an object composed of the properties the given function returns truthy for.

- Use `Object.keys(obj)` and `Array.prototype.filter()` to remove the keys for which `fn` returns a falsy value.
- Use `Array.prototype.reduce()` to convert the filtered keys back to an object with the corresponding key-value pairs.
- The callback function is invoked with two arguments: (value, key).

```
const pickBy = (obj, fn) =>
  Object.keys(obj)
    .filter(k => fn(obj[k], k))
    .reduce((acc, key) => ((acc[key] = obj[key]), acc), {});
```

```
pickBy({ a: 1, b: '2', c: 3 }, x => typeof x === 'number');
// { 'a': 1, 'c': 3 }
```

title: pipeAsyncFunctions

Performs left-to-right function composition for asynchronous functions.

- Use `Array.prototype.reduce()` and the spread operator (`...`) to perform function composition using `Promise.prototype.then()` .
- The functions can return a combination of normal values, `Promise` s or be `async` , returning through `await` .
- All functions must accept a single argument.

```
const pipeAsyncFunctions = (...fns) =>
  arg => fns.reduce((p, f) => p.then(f), Promise.resolve(arg));
```

```js
const sum = pipeAsyncFunctions(
  x => x + 1,
  x => new Promise(resolve => setTimeout(() => resolve(x + 2), 1000)),
  x => x + 3,
  async x => (await x) + 4
);
(async() => {
  console.log(await sum(5)); // 15 (after one second)
})();
```

title: pipeFunctions

Performs left-to-right function composition.

- Use `Array.prototype.reduce()` with the spread operator (`...`) to perform left-to-right function composition.
- The first (leftmost) function can accept one or more arguments; the remaining functions must be unary.

```js
const pipeFunctions = (...fns) =>
  fns.reduce((f, g) => (...args) => g(f(...args)));
```

```js
const add5 = x => x + 5;
const multiply = (x, y) => x * y;
const multiplyAndAdd5 = pipeFunctions(multiply, add5);
multiplyAndAdd5(5, 2); // 15
```

title: pluck

Converts an array of objects into an array of values corresponding to the specified `key` .

- Use `Array.prototype.map()` to map the array of objects to the value of `key` for each one.

```js
const pluck = (arr, key) => arr.map(i => i[key]);
const simpsons = [
  { name: 'lisa', age: 8 },
  { name: 'homer', age: 36 },
  { name: 'marge', age: 34 },
  { name: 'bart', age: 10 }
];
pluck(simpsons, 'age'); // [8, 36, 34, 10]
```

title: pluralize

Returns the singular or plural form of the word based on the input number, using an optional dictionary if supplied.

- Use a closure to define a function that pluralizes the given `word` based on the value of `num` .
- If `num` is either `-1` or `1` , return the singular form of the word.
- If `num` is any other number, return the `plural` form.
- Omit the third argument, `plural` , to use the default of the singular word + `s` , or supply a custom pluralized `word` when necessary.
- If the first argument is an `object` , return a function which can use the supplied dictionary to resolve the correct plural form of the word.

```
const pluralize = (val, word, plural = word + 's') => {
  const _pluralize = (num, word, plural = word + 's') =>
    [1, -1].includes(Number(num)) ? word : plural;
  if (typeof val === 'object')
    return (num, word) => _pluralize(num, word, val[word]);
  return _pluralize(val, word, plural);
};
```

```
pluralize(0, 'apple'); // 'apples'
pluralize(1, 'apple'); // 'apple'
pluralize(2, 'apple'); // 'apples'
pluralize(2, 'person', 'people'); // 'people'

const PLURALS = {
  person: 'people',
  radius: 'radii'
};
const autoPluralize = pluralize(PLURALS);
autoPluralize(2, 'person'); // 'people'
```

title: powerset

Returns the powerset of a given array of numbers.

- Use `Array.prototype.reduce()` combined with `Array.prototype.map()` to iterate over elements and combine into an array containing all combinations.

```
const powerset = arr =>
  arr.reduce((a, v) => a.concat(a.map(r => r.concat(v))), [[]]);


powerset([1, 2]); // [[], [1], [2], [1, 2]]
```

title: prefersDarkColorScheme

Checks if the user color scheme preference is `dark` .

- Use `Window.matchMedia()` with the appropriate media query to check the user color scheme preference.

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


prefersDarkColorScheme(); // true
```

title: prefersLightColorScheme

Checks if the user color scheme preference is `light` .

- Use `Window.matchMedia()` with the appropriate media query to check the user color scheme preference.

```
const prefersLightColorScheme = () =>
  window &&
  window.matchMedia &&
  window.matchMedia('(prefers-color-scheme: light)').matches;


prefersLightColorScheme(); // true
```

title: prefix

Prefixes a CSS property based on the current browser.

- Use `Array.prototype.findIndex()` on an array of vendor prefix strings to test if `Document.body` has one of them defined in its `CSSStyleDeclaration` object, otherwise return `null` .
- Use `String.prototype.charAt()` and `String.prototype.toUpperCase()` to capitalize the property, which will be appended to the vendor prefix string.

```
const prefix = prop => {
  const capitalizedProp = prop.charAt(0).toUpperCase() + prop.slice(1);
  const prefixes = ['', 'webkit', 'moz', 'ms', 'o'];
  const i = prefixes.findIndex(
    prefix =>
      typeof document.body.style[prefix ? prefix + capitalizedProp : prop] !==
      'undefined'
  );
  return i !== -1 ? (i === 0 ? prop : prefixes[i] + capitalizedProp) : null;
};


prefix('appearance');
// 'appearance' on a supported browser, otherwise 'webkitAppearance', 'mozAppearance', 'msAppear
```

title: prettyBytes

Converts a number in bytes to a human-readable string.

- Use an array dictionary of units to be accessed based on the exponent.
- Use `Number.prototype.toPrecision()` to truncate the number to a certain number of digits.

- Return the prettified string by building it up, taking into account the supplied options and whether it is negative or not.
- Omit the second argument, `precision` , to use a default precision of `3` digits.
- Omit the third argument, `addSpace` , to add space between the number and unit by default.

```
const prettyBytes = (num, precision = 3, addSpace = true) => {
  const UNITS = ['B', 'KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'];
  if (Math.abs(num) < 1) return num + (addSpace ? ' ' : '') + UNITS[0];
  const exponent = Math.min(
    Math.floor(Math.log10(num < 0 ? -num : num) / 3),
    UNITS.length - 1
  );
  const n = Number(
    ((num < 0 ? -num : num) / 1000 ** exponent).toPrecision(precision)
  );
  return (num < 0 ? '-' : '') + n + (addSpace ? ' ' : '') + UNITS[exponent];
};
```

```
prettyBytes(1000); // '1 KB'
prettyBytes(-27145424323.5821, 5); // '-27.145 GB'
prettyBytes(123456789, 3, false); // '123MB'
```

title: primeFactors

Finds the prime factors of a given number using the trial division algorithm.

- Use a `while` loop to iterate over all possible prime factors, starting with `2` .
- If the current factor, `f` , exactly divides `n` , add `f` to the factors array and divide `n` by `f` . Otherwise, increment `f` by one.

```js
const primeFactors = n => {
  let a = [],
    f = 2;
  while (n > 1) {
    if (n % f === 0) {
      a.push(f);
      n /= f;
    } else {
      f++;
    }
  }
  return a;
};
```

```js
primeFactors(147); // [3, 7, 7]
```

title: primes

Generates primes up to a given number, using the Sieve of Eratosthenes.

- Generate an array from `2` to the given number.
- Use `Array.prototype.filter()` to filter out the values divisible by any number from `2` to the square root of the provided number.

```js
const primes = num => {
  let arr = Array.from({ length: num - 1 }).map((x, i) => i + 2),
    sqroot = Math.floor(Math.sqrt(num)),
    numsTillSqroot = Array.from({ length: sqroot - 1 }).map((x, i) => i + 2);
  numsTillSqroot.forEach(x => (arr = arr.filter(y => y % x !== 0 || y === x)));
  return arr;
};
```

```js
primes(10); // [2, 3, 5, 7]
```

title: prod

Calculates the product of two or more numbers/arrays.

- Use `Array.prototype.reduce()` to multiply each value with an accumulator, initialized with a value of `1`.

```
const prod = (...arr) => [...arr].reduce((acc, val) => acc * val, 1);


prod(1, 2, 3, 4); // 24
prod(...[1, 2, 3, 4]); // 24
```

title: promisify

Converts an asynchronous function to return a promise.

- Use currying to return a function returning a `Promise` that calls the original function.
- Use the rest operator (`...`) to pass in all the parameters.
- **Note:** In Node 8+, you can use `util.promisify`.

```
const promisify = func => (...args) =>
  new Promise((resolve, reject) =>
    func(...args, (err, result) => (err ? reject(err) : resolve(result)))
  );


const delay = promisify((d, cb) => setTimeout(cb, d));
delay(2000).then(() => console.log('Hi!')); // Promise resolves after 2s
```

title: pull

Mutates the original array to filter out the values specified.

- Use `Array.prototype.filter()` and `Array.prototype.includes()` to pull out the values that are not needed.
- Set `Array.prototype.length` to mutate the passed in an array by resetting its length to `0`.
- Use `Array.prototype.push()` to re-populate it with only the pulled values.

```
const pull = (arr, ...args) => {
  let argState = Array.isArray(args[0]) ? args[0] : args;
  let pulled = arr.filter(v => !argState.includes(v));
  arr.length = 0;
  pulled.forEach(v => arr.push(v));
};
```

```
let myArray = ['a', 'b', 'c', 'a', 'b', 'c'];
pull(myArray, 'a', 'c'); // myArray = [ 'b', 'b' ]
```

title: pullAtIndex

Mutates the original array to filter out the values at the specified indexes.
Returns the removed elements.

- Use `Array.prototype.filter()` and `Array.prototype.includes()` to pull out the values that are not needed.
- Set `Array.prototype.length` to mutate the passed in an array by resetting its length to `0`.
- Use `Array.prototype.push()` to re-populate it with only the pulled values.
- Use `Array.prototype.push()` to keep track of pulled values.

```
const pullAtIndex = (arr, pullArr) => {
  let removed = [];
  let pulled = arr
    .map((v, i) => (pullArr.includes(i) ? removed.push(v) : v))
    .filter((v, i) => !pullArr.includes(i));
  arr.length = 0;
  pulled.forEach(v => arr.push(v));
  return removed;
};
```

```
let myArray = ['a', 'b', 'c', 'd'];
let pulled = pullAtIndex(myArray, [1, 3]);
// myArray = [ 'a', 'c' ] , pulled = [ 'b', 'd' ]
```

title: pullAtValue

Mutates the original array to filter out the values specified.
Returns the removed elements.

- Use `Array.prototype.filter()` and `Array.prototype.includes()` to pull out the values that are not needed.
- Set `Array.prototype.length` to mutate the passed in an array by resetting its length to `0`.
- Use `Array.prototype.push()` to re-populate it with only the pulled values.
- Use `Array.prototype.push()` to keep track of pulled values.

```
const pullAtValue = (arr, pullArr) => {
  let removed = [],
    pushToRemove = arr.forEach((v, i) =>
      pullArr.includes(v) ? removed.push(v) : v
    ),
    mutateTo = arr.filter((v, i) => !pullArr.includes(v));
  arr.length = 0;
  mutateTo.forEach(v => arr.push(v));
  return removed;
};
```

```
let myArray = ['a', 'b', 'c', 'd'];
let pulled = pullAtValue(myArray, ['b', 'd']);
// myArray = [ 'a', 'c' ] , pulled = [ 'b', 'd' ]
```

title: pullBy

Mutates the original array to filter out the values specified, based on a given iterator function.

- Check if the last argument provided is a function.
- Use `Array.prototype.map()` to apply the iterator function `fn` to all array elements.
- Use `Array.prototype.filter()` and `Array.prototype.includes()` to pull out the values that are not needed.
- Set `Array.prototype.length` to mutate the passed in an array by resetting its length to `0`.
- Use `Array.prototype.push()` to re-populate it with only the pulled values.

```
const pullBy = (arr, ...args) => {
  const length = args.length;
  let fn = length > 1 ? args[length - 1] : undefined;
  fn = typeof fn == 'function' ? (args.pop(), fn) : undefined;
  let argState = (Array.isArray(args[0]) ? args[0] : args).map(val => fn(val));
  let pulled = arr.filter((v, i) => !argState.includes(fn(v)));
  arr.length = 0;
  pulled.forEach(v => arr.push(v));
};
```

```
var myArray = [{ x: 1 }, { x: 2 }, { x: 3 }, { x: 1 }];
pullBy(myArray, [{ x: 1 }, { x: 3 }], o => o.x); // myArray = [{ x: 2 }]
```

title: quarterOfYear

Returns the quarter and year to which the supplied date belongs to.

- Use `Date.prototype.getMonth()` to get the current month in the range (0, 11), add `1` to map it to the range (1, 12).
- Use `Math.ceil()` and divide the month by `3` to get the current quarter.
- Use `Date.prototype.getFullYear()` to get the year from the given `date` .
- Omit the argument, `date` , to use the current date by default.

```
const quarterOfYear = (date = new Date()) => [
  Math.ceil((date.getMonth() + 1) / 3),
  date.getFullYear()
];
```

```
quarterOfYear(new Date('07/10/2018')); // [ 3, 2018 ]
quarterOfYear(); // [ 4, 2020 ]
```

title: queryStringToObject

Generates an object from the given query string or URL.

- Use `String.prototype.split()` to get the params from the given `url` .

- Use `new URLSearchParams()` to create an appropriate object and convert it to an array of key-value pairs using the spread operator (`...`).
- Use `Array.prototype.reduce()` to convert the array of key-value pairs into an object.

```
const queryStringToObject = url =>
  [...new URLSearchParams(url.split('?')[1])].reduce(
    (a, [k, v]) => ((a[k] = v), a),
    {}
  );


queryStringToObject('https://google.com?page=1&count=10');
// {page: '1', count: '10'}
```

title: quickSort

Sorts an array of numbers, using the quicksort algorithm.

- Use recursion.
- Use the spread operator (`...`) to clone the original array, `arr` .
- If the `length` of the array is less than `2` , return the cloned array.
- Use `Math.floor()` to calculate the index of the pivot element.
- Use `Array.prototype.reduce()` and `Array.prototype.push()` to split the array into two subarrays. The first one contains elements smaller than or equal to `pivot` and the second on elements greather than it. Destructure the result into two arrays.
- Recursively call `quickSort()` on the created subarrays.

```
const quickSort = arr => {
  const a = [...arr];
  if (a.length < 2) return a;
  const pivotIndex = Math.floor(arr.length / 2);
  const pivot = a[pivotIndex];
  const [lo, hi] = a.reduce(
    (acc, val, i) => {
      if (val < pivot || (val === pivot && i != pivotIndex)) {
        acc[0].push(val);
      } else if (val > pivot) {
        acc[1].push(val);
      }
      return acc;
    },
    [[], []]
  );
  return [...quickSort(lo), pivot, ...quickSort(hi)];
};


quickSort([1, 6, 1, 5, 3, 2, 1, 4]); // [1, 1, 1, 2, 3, 4, 5, 6]
```

title: radsToDegrees

Converts an angle from radians to degrees.

- Use `Math.PI` and the radian to degree formula to convert the angle from radians to degrees.

```
const radsToDegrees = rad => (rad * 180.0) / Math.PI;


radsToDegrees(Math.PI / 2); // 90
```

title: randomAlphaNumeric

Generates a random string with the specified length.

- Use `Array.from()` to create a new array with the specified `length` .
- Use `Math.random()` generate a random floating-point number, `Number.prototype.toString(36)` to convert it to an alphanumeric string.

- Use `String.prototype.slice(2)` to remove the integral part and decimal point from each generated number.
- Use `Array.prototype.some()` to repeat this process as many times as required, up to `length` , as it produces a variable-length string each time.
- Finally, use `String.prototype.slice()` to trim down the generated string if it's longer than the given `length` .

```
const randomAlphaNumeric = length => {
  let s = '';
  Array.from({ length }).some(() => {
    s += Math.random().toString(36).slice(2);
    return s.length >= length;
  });
  return s.slice(0, length);
};


randomAlphaNumeric(5); // '0afad'
```

title: randomBoolean

Generates a random boolean value.

- Use `Math.random()` to generate a random number and check if it is greater than or equal to `0.5` .

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


randomBoolean(); // true
```

title: randomHexColorCode

Generates a random hexadecimal color code.

- Use `Math.random()` to generate a random 24-bit (6 * 4bits) hexadecimal number.
- Use bit shifting and then convert it to an hexadecimal string using `Number.prototype.toString(16)` .

```
const randomHexColorCode = () => {
  let n = (Math.random() * 0xfffff * 1000000).toString(16);
  return '#' + n.slice(0, 6);
};
```

```
randomHexColorCode(); // '#e34155'
```

title: randomIntArrayInRange

Generates an array of `n` random integers in the specified range.

- Use `Array.from()` to create an empty array of the specific length.
- Use `Math.random()` to generate random numbers and map them to the desired range, using `Math.floor()` to make them integers.

```
const randomIntArrayInRange = (min, max, n = 1) =>
  Array.from(
    { length: n },
    () => Math.floor(Math.random() * (max - min + 1)) + min
  );
```

```
randomIntArrayInRange(12, 35, 10); // [ 34, 14, 27, 17, 30, 27, 20, 26, 21, 14 ]
```

title: randomIntegerInRange

Generates a random integer in the specified range.

- Use `Math.random()` to generate a random number and map it to the desired range.
- Use `Math.floor()` to make it an integer.

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

```
randomIntegerInRange(0, 5); // 2
```

title: randomNumberInRange

Generates a random number in the specified range.

- Use `Math.random()` to generate a random value, map it to the desired range using multiplication.

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


randomNumberInRange(2, 10); // 6.0211363285087005
```

title: rangeGenerator

Creates a generator, that generates all values in the given range using the given step.

- Use a `while` loop to iterate from `start` to `end`, using `yield` to return each value and then incrementing by `step`.
- Omit the third argument, `step`, to use a default value of `1`.

```
const rangeGenerator = function* (start, end, step = 1) {
  let i = start;
  while (i < end) {
    yield i;
    i += step;
  }
};


for (let i of rangeGenerator(6, 10)) console.log(i);
// Logs 6, 7, 8, 9
```

title: readFileLines

Returns an array of lines from the specified file.

- Use `fs.readFileSync()` to create a `Buffer` from a file.
- Convert buffer to string using `buf.toString(encoding)` function.
- Use `String.prototype.split(\n)` to create an array of lines from the contents of the file.

```
const fs = require('fs');

const readFileLines = filename =>
  fs
    .readFileSync(filename)
    .toString('UTF8')
    .split('\n');


/*
contents of test.txt :
  line1
  line2
  line3
  ___________________________
*/
let arr = readFileLines('test.txt');
console.log(arr); // ['line1', 'line2', 'line3']
```

title: rearg

Creates a function that invokes the provided function with its arguments arranged according to the specified indexes.

- Use `Array.prototype.map()` to reorder arguments based on `indexes` .
- Use the spread operator (`...`) to pass the transformed arguments to `fn` .

```
const rearg = (fn, indexes) => (...args) => fn(...indexes.map(i => args[i]));


var rearged = rearg(
  function(a, b, c) {
    return [a, b, c];
  },
  [2, 0, 1]
);
rearged('b', 'c', 'a'); // ['a', 'b', 'c']
```

title: recordAnimationFrames

Invokes the provided callback on each animation frame.

- Use recursion.
- Provided that `running` is `true`, continue invoking `Window.requestAnimationFrame()` which invokes the provided callback.
- Return an object with two methods `start` and `stop` to allow manual control of the recording.
- Omit the second argument, `autoStart`, to implicitly call `start` when the function is invoked.

```
const recordAnimationFrames = (callback, autoStart = true) => {
  let running = false,
    raf;
  const stop = () => {
    if (!running) return;
    running = false;
    cancelAnimationFrame(raf);
  };
  const start = () => {
    if (running) return;
    running = true;
    run();
  };
  const run = () => {
    raf = requestAnimationFrame(() => {
      callback();
      if (running) run();
    });
  };
  if (autoStart) start();
  return { start, stop };
};
```

```
const cb = () => console.log('Animation frame fired');
const recorder = recordAnimationFrames(cb);
// logs 'Animation frame fired' on each animation frame
recorder.stop(); // stops logging
recorder.start(); // starts again
const recorder2 = recordAnimationFrames(cb, false);
// `start` needs to be explicitly called to begin recording frames
```

title: redirect

Redirects to a specified URL.

- Use `Window.location.href` or `Window.location.replace()` to redirect to `url` .
- Pass a second argument to simulate a link click (`true` - default) or an HTTP redirect (`false`).

```
const redirect = (url, asLink = true) =>
  asLink ? (window.location.href = url) : window.location.replace(url);


redirect('https://google.com');
```

title: reduceSuccessive

Applies a function against an accumulator and each element in the array (from left to right), returning an array of successively reduced values.

- Use `Array.prototype.reduce()` to apply the given function to the given array, storing each new result.

```
const reduceSuccessive = (arr, fn, acc) =>
  arr.reduce(
    (res, val, i, arr) => (res.push(fn(res.slice(-1)[0], val, i, arr)), res),
    [acc]
  );


reduceSuccessive([1, 2, 3, 4, 5, 6], (acc, val) => acc + val, 0);
// [0, 1, 3, 6, 10, 15, 21]
```

title: reduceWhich

Returns the minimum/maximum value of an array, after applying the provided function to set the comparing rule.

- Use `Array.prototype.reduce()` in combination with the `comparator` function to get the appropriate element in the array.
- Omit the second argument, `comparator` , to use the default one that returns the minimum element in the array.

```js
const reduceWhich = (arr, comparator = (a, b) => a - b) =>
  arr.reduce((a, b) => (comparator(a, b) >= 0 ? b : a));


reduceWhich([1, 3, 2]); // 1
reduceWhich([1, 3, 2], (a, b) => b - a); // 3
reduceWhich(
  [
    { name: 'Tom', age: 12 },
    { name: 'Jack', age: 18 },
    { name: 'Lucy', age: 9 }
  ],
  (a, b) => a.age - b.age
); // {name: 'Lucy', age: 9}
```

title: reducedFilter

Filters an array of objects based on a condition while also filtering out unspecified keys.

- Use `Array.prototype.filter()` to filter the array based on the predicate `fn` so that it returns the objects for which the condition returned a truthy value.
- On the filtered array, use `Array.prototype.map()` to return the new object.
- Use `Array.prototype.reduce()` to filter out the keys which were not supplied as the `keys` argument.

```js
const reducedFilter = (data, keys, fn) =>
  data.filter(fn).map(el =>
    keys.reduce((acc, key) => {
      acc[key] = el[key];
      return acc;
    }, {})
  );
```

```js
const data = [
  {
    id: 1,
    name: 'john',
    age: 24
  },
  {
    id: 2,
    name: 'mike',
    age: 50
  }
];
reducedFilter(data, ['id', 'name'], item => item.age > 24);
// [{ id: 2, name: 'mike'}]
```

title: reject

Filters an array's values based on a predicate function, returning only values for which the predicate function returns `false`.

- Use `Array.prototype.filter()` in combination with the predicate function, `pred`, to return only the values for which it returns `false`.

```js
const reject = (pred, array) => array.filter((...args) => !pred(...args));
```

```js
reject(x => x % 2 === 0, [1, 2, 3, 4, 5]); // [1, 3, 5]
reject(word => word.length > 4, ['Apple', 'Pear', 'Kiwi', 'Banana']);
// ['Pear', 'Kiwi']
```

title: remove

Mutates an array by removing elements for which the given function returns `false`.

- Use `Array.prototype.filter()` to find array elements that return truthy values.
- Use `Array.prototype.reduce()` to remove elements using `Array.prototype.splice()`.
- The callback function is invoked with three arguments (value, index, array).

```js
const remove = (arr, func) =>
  Array.isArray(arr)
    ? arr.filter(func).reduce((acc, val) => {
      arr.splice(arr.indexOf(val), 1);
      return acc.concat(val);
    }, [])
    : [];


remove([1, 2, 3, 4], n => n % 2 === 0); // [2, 4]
```

title: removeAccents

Removes accents from strings.

- Use `String.prototype.normalize()` to convert the string to a normalized Unicode format.
- Use `String.prototype.replace()` to replace diacritical marks in the given Unicode range by empty strings.

```js
const removeAccents = str =>
  str.normalize('NFD').replace(/[\u0300-\u036f]/g, '');


removeAccents('Antoine de Saint-Exupéry'); // 'Antoine de Saint-Exupery'
```

title: removeClass

Removes a class from an HTML element.

- Use `Element.classList` and `DOMTokenList.remove()` to remove the specified class from the element.

```js
const removeClass = (el, className) => el.classList.remove(className);


removeClass(document.querySelector('p.special'), 'special');
// The paragraph will not have the 'special' class anymore
```

title: removeElement

Removes an element from the DOM.

- Use `Element.parentNode` to get the given element's parent node.
- Use `Element.removeChild()` to remove the given element from its parent node.

```js
const removeElement = el => el.parentNode.removeChild(el);


removeElement(document.querySelector('#my-element'));
// Removes #my-element from the DOM
```

title: removeEventListenerAll

Detaches an event listener from all the provided targets.

- Use `Array.prototype.forEach()` and `EventTarget.removeEventListener()` to detach the provided `listener` for the given event `type` from all `targets` .

```js
const removeEventListenerAll = (
  targets,
  type,
  listener,
  options,
  useCapture
) => {
  targets.forEach(target =>
    target.removeEventListener(type, listener, options, useCapture)
  );
};


const linkListener = () => console.log('Clicked a link');
document.querySelector('a').addEventListener('click', linkListener);
removeEventListenerAll(document.querySelectorAll('a'), 'click', linkListener);
```

title: removeNonASCII

Removes non-printable ASCII characters.

- Use `String.prototype.replace()` with a regular expression to remove non-printable ASCII characters.

```
const removeNonASCII = str => str.replace(/[^\x20-\x7E]/g, '');
```

```
removeNonASCII('äÄçÇéÉêlorem-ipsumöÖÐþúÚ'); // 'lorem-ipsum'
```

title: removeWhitespace

Returns a string with whitespaces removed.

- Use `String.prototype.replace()` with a regular expression to replace all occurrences of whitespace characters with an empty string.

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

```
removeWhitespace('Lorem ipsum.\n Dolor sit amet. ');
// 'Loremipsum.Dolorsitamet.'
```

title: renameKeys

Replaces the names of multiple object keys with the values provided.

- Use `Object.keys()` in combination with `Array.prototype.reduce()` and the spread operator (`...`) to get the object's keys and rename them according to `keysMap` .

```
const renameKeys = (keysMap, obj) =>
  Object.keys(obj).reduce(
    (acc, key) => ({
      ...acc,
      ...{ [keysMap[key] || key]: obj[key] }
    }),
    {}
  );
```

```
const obj = { name: 'Bobo', job: 'Front-End Master', shoeSize: 100 };
renameKeys({ name: 'firstName', job: 'passion' }, obj);
// { firstName: 'Bobo', passion: 'Front-End Master', shoeSize: 100 }
```

title: renderElement

Renders the given DOM tree in the specified DOM element.

- Destructure the first argument into `type` and `props`. Use `type` to determine if the given element is a text element.
- Based on the element's `type`, use either `Document.createTextNode()` or `Document.createElement()` to create the DOM element.
- Use `Object.keys()` to add attributes to the DOM element and set event listeners, as necessary.
- Use recursion to render `props.children`, if any.
- Finally, use `Node.appendChild()` to append the DOM element to the specified `container`.

```
const renderElement = ({ type, props = {} }, container) => {
  const isTextElement = !type;
  const element = isTextElement
    ? document.createTextNode('')
    : document.createElement(type);

  const isListener = p => p.startsWith('on');
  const isAttribute = p => !isListener(p) && p !== 'children';

  Object.keys(props).forEach(p => {
    if (isAttribute(p)) element[p] = props[p];
    if (!isTextElement && isListener(p))
      element.addEventListener(p.toLowerCase().slice(2), props[p]);
  });

  if (!isTextElement && props.children && props.children.length)
    props.children.forEach(childElement =>
      renderElement(childElement, element)
    );

  container.appendChild(element);
};
```

```
const myElement = {
  type: 'button',
  props: {
    type: 'button',
    className: 'btn',
    onClick: () => alert('Clicked'),
    children: [{ props: { nodeValue: 'Click me' } }]
  }
};

renderElement(myElement, document.body);
```

title: repeatGenerator

Creates a generator, repeating the given value indefinitely.

- Use a non-terminating `while` loop, that will `yield` a value every time `Generator.prototype.next()` is called.
- Use the return value of the `yield` statement to update the returned value if the passed value is not `undefined` .

```
const repeatGenerator = function* (val) {
  let v = val;
  while (true) {
    let newV = yield v;
    if (newV !== undefined) v = newV;
  }
};


const repeater = repeatGenerator(5);
repeater.next(); // { value: 5, done: false }
repeater.next(); // { value: 5, done: false }
repeater.next(4); // { value: 4, done: false }
repeater.next(); // { value: 4, done: false }
```

title: replaceLast

Replaces the last occurence of a pattern in a string.

- Use `typeof` to determine if `pattern` is a string or a regular expression.
- If the `pattern` is a string, use it as the `match`.
- Otherwise, use the `RegeExp` constructor to create a new regular expression using the `RegExp.source` of the `pattern` and adding the `'g'` flag to it. Use `String.prototype.match()` and `Array.prototype.slice()` to get the last match, if any.
- Use `String.prototype.lastIndexOf()` to find the last occurence of the match in the string.
- If a match is found, use `String.prototype.slice()` and a template literal to replace the matching substring with the given `replacement`.
- If no match is found, return the original string.

```
const replaceLast = (str, pattern, replacement) => {
  const match =
    typeof pattern === 'string'
      ? pattern
      : (str.match(new RegExp(pattern.source, 'g')) || []).slice(-1)[0];
  if (!match) return str;
  const last = str.lastIndexOf(match);
  return last !== -1
    ? `${str.slice(0, last)}${replacement}${str.slice(last + match.length)}`
    : str;
};
```

```
replaceLast('abcabdef', 'ab', 'gg'); // 'abcggdef'
replaceLast('abcabdef', /ab/, 'gg'); // 'abcggdef'
replaceLast('abcabdef', 'ad', 'gg'); // 'abcabdef'
replaceLast('abcabdef', /ad/, 'gg'); // 'abcabdef'
```

title: requireUncached

Loads a module after removing it from the cache (if exists).

- Use `delete` to remove the module from the cache (if exists).
- Use `require()` to load the module again.

```
const requireUncached = module => {
  delete require.cache[require.resolve(module)];
  return require(module);
};
```

```
const fs = requireUncached('fs'); // 'fs' will be loaded fresh every time
```

title: reverseNumber

Reverses a number.

- Use `Object.prototype.toString()` to convert `n` to a string.
- Use `String.prototype.split('')`, `Array.prototype.reverse()` and `String.prototype.join('')` to get the reversed value of `n` as a string.
- Use `parseFloat()` to convert the string to a number and `Math.sign()` to preserve its sign.

```
const reverseNumber = n =>
  parseFloat(`${n}`.split('').reverse().join('')) * Math.sign(n);
```

```
reverseNumber(981); // 189
reverseNumber(-500); // -5
reverseNumber(73.6); // 6.37
reverseNumber(-5.23); // -32.5
```

title: reverseString

Reverses a string.

- Use the spread operator (`...`) and `Array.prototype.reverse()` to reverse the order of the characters in the string.
- Combine characters to get a string using `String.prototype.join('')`.

```
const reverseString = str => [...str].reverse().join('');
```

```
reverseString('foobar'); // 'raboof'
```

title: round

Rounds a number to a specified amount of digits.

- Use `Math.round()` and template literals to round the number to the specified number of digits.

- Omit the second argument, `decimals` , to round to an integer.

```
const round = (n, decimals = 0) =>
  Number(`${Math.round(`${n}e${decimals}`)}e-${decimals}`);


round(1.005, 2); // 1.01
```

title: runAsync

Runs a function in a separate thread by using a Web Worker, allowing long running functions to not block the UI.

- Create a `new Worker()` using a `Blob` object URL, the contents of which should be the stringified version of the supplied function.
- Immediately post the return value of calling the function back.
- Return a `new Promise()` , listening for `onmessage` and `onerror` events and resolving the data posted back from the worker, or throwing an error.

```
const runAsync = fn => {
  const worker = new Worker(
    URL.createObjectURL(new Blob([`postMessage((${fn})());`]), {
      type: 'application/javascript; charset=utf-8'
    })
  );
  return new Promise((res, rej) => {
    worker.onmessage = ({ data }) => {
      res(data), worker.terminate();
    };
    worker.onerror = err => {
      rej(err), worker.terminate();
    };
  });
};
```

```
const longRunningFunction = () => {
  let result = 0;
  for (let i = 0; i < 1000; i++)
    for (let j = 0; j < 700; j++)
      for (let k = 0; k < 300; k++) result = result + i + j + k;

  return result;
};
/*
  NOTE: Since the function is running in a different context, closures are not supported.
  The function supplied to `runAsync` gets stringified, so everything becomes literal.
  All variables and functions must be defined inside.
*/
runAsync(longRunningFunction).then(console.log); // 209685000000
runAsync(() => 10 ** 3).then(console.log); // 1000
let outsideVariable = 50;
runAsync(() => typeof outsideVariable).then(console.log); // 'undefined'
```

title: runPromisesInSeries

Runs an array of promises in series.

- Use `Array.prototype.reduce()` to create a promise chain, where each promise returns the next promise when resolved.

```
const runPromisesInSeries = ps =>
  ps.reduce((p, next) => p.then(next), Promise.resolve());
```

```
const delay = d => new Promise(r => setTimeout(r, d));
runPromisesInSeries([() => delay(1000), () => delay(2000)]);
// Executes each promise sequentially, taking a total of 3 seconds to complete
```

title: sample

Gets a random element from an array.

- Use `Math.random()` to generate a random number.
- Multiply it by `Array.prototype.length` and round it off to the nearest whole number using `Math.floor()` .

- This method also works with strings.

```
const sample = arr => arr[Math.floor(Math.random() * arr.length)];


sample([3, 7, 9, 11]); // 9
```

title: sampleSize

Gets `n` random elements at unique keys from an array up to the size of the array.

- Shuffle the array using the Fisher-Yates algorithm.
- Use `Array.prototype.slice()` to get the first `n` elements.
- Omit the second argument, `n` , to get only one element at random from the array.

```
const sampleSize = ([...arr], n = 1) => {
  let m = arr.length;
  while (m) {
    const i = Math.floor(Math.random() * m--);
    [arr[m], arr[i]] = [arr[i], arr[m]];
  }
  return arr.slice(0, n);
};


sampleSize([1, 2, 3], 2); // [3, 1]
sampleSize([1, 2, 3], 4); // [2, 3, 1]
```

title: scrollToTop

Smooth-scrolls to the top of the page.

- Get distance from top using `Document.documentElement` or `Document.body` and `Element.scrollTop` .
- Scroll by a fraction of the distance from the top.
- Use `Window.requestAnimationFrame()` to animate the scrolling.

```
const scrollToTop = () => {
  const c = document.documentElement.scrollTop || document.body.scrollTop;
  if (c > 0) {
    window.requestAnimationFrame(scrollToTop);
    window.scrollTo(0, c - c / 8);
  }
};


scrollToTop(); // Smooth-scrolls to the top of the page
```

title: sdbm

Hashes the input string into a whole number.

- Use `String.prototype.split('')` and `Array.prototype.reduce()` to create a hash of the input string, utilizing bit shifting.

```
const sdbm = str => {
  let arr = str.split('');
  return arr.reduce(
    (hashCode, currentVal) =>
      (hashCode =
        currentVal.charCodeAt(0) +
        (hashCode << 6) +
        (hashCode << 16) -
        hashCode),
    0
  );
};


sdbm('name'); // -3521204949
```

title: selectionSort

Sorts an array of numbers, using the selection sort algorithm.

- Use the spread operator (`...`) to clone the original array, `arr` .
- Use a `for` loop to iterate over elements in the array.

- Use `Array.prototype.slice()` and `Array.prototype.reduce()` to find the index of the minimum element in the subarray to the right of the current index. Perform a swap, if necessary.

```js
const selectionSort = arr => {
  const a = [...arr];
  for (let i = 0; i < a.length; i++) {
    const min = a
      .slice(i + 1)
      .reduce((acc, val, j) => (val < a[acc] ? j + i + 1 : acc), i);
    if (min !== i) [a[i], a[min]] = [a[min], a[i]];
  }
  return a;
};


selectionSort([5, 1, 4, 2, 3]); // [1, 2, 3, 4, 5]
```

title: serializeCookie

Serializes a cookie name-value pair into a Set-Cookie header string.

- Use template literals and `encodeURIComponent()` to create the appropriate string.

```js
const serializeCookie = (name, val) =>
  `${encodeURIComponent(name)}=${encodeURIComponent(val)}`;


serializeCookie('foo', 'bar'); // 'foo=bar'
```

title: serializeForm

Encodes a set of form elements as a query string.

- Use the `FormData` constructor to convert the HTML `form` to `FormData` .
- Use `Array.from()` to convert to an array, passing a map function as the second argument.
- Use `Array.prototype.map()` and `encodeURIComponent()` to encode each field's value.
- Use `Array.prototype.join()` with appropriate arguments to produce an appropriate query string.

```
const serializeForm = form =>
  Array.from(new FormData(form), field =>
    field.map(encodeURIComponent).join('=')
  ).join('&');


serializeForm(document.querySelector('#form'));
// email=test%40email.com&name=Test%20Name
```

title: setStyle

Sets the value of a CSS rule for the specified HTML element.

- Use `ElementCSSInlineStyle.style` to set the value of the CSS `rule` for the specified element to `val` .

```
const setStyle = (el, rule, val) => (el.style[rule] = val);


setStyle(document.querySelector('p'), 'font-size', '20px');
// The first <p> element on the page will have a font-size of 20px
```

title: shallowClone

Creates a shallow clone of an object.

- Use `Object.assign()` and an empty object (`{}`) to create a shallow clone of the original.

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


const a = { x: true, y: 1 };
const b = shallowClone(a); // a !== b
```

title: shank

Has the same functionality as `Array.prototype.splice()` , but returning a new array instead of mutating the original array.

- Use `Array.prototype.slice()` and `Array.prototype.concat()` to get an array with the new contents after removing existing elements and/or adding new elements.
- Omit the second argument, `index` , to start at `0` .
- Omit the third argument, `delCount` , to remove `0` elements.
- Omit the fourth argument, `elements` , in order to not add any new elements.

```
const shank = (arr, index = 0, delCount = 0, ...elements) =>
  arr
    .slice(0, index)
    .concat(elements)
    .concat(arr.slice(index + delCount));
```

```
const names = ['alpha', 'bravo', 'charlie'];
const namesAndDelta = shank(names, 1, 0, 'delta');
// [ 'alpha', 'delta', 'bravo', 'charlie' ]
const namesNoBravo = shank(names, 1, 1); // [ 'alpha', 'charlie' ]
console.log(names); // ['alpha', 'bravo', 'charlie']
```

title: show

Shows all the elements specified.

- Use the spread operator (`...`) and `Array.prototype.forEach()` to clear the `display` property for each element specified.

```
const show = (...el) => [...el].forEach(e => (e.style.display = ''));
```

```
show(...document.querySelectorAll('img'));
// Shows all <img> elements on the page
```

title: shuffle

Randomizes the order of the values of an array, returning a new array.

- Use the Fisher-Yates algorithm to reorder the elements of the array.

```
const shuffle = ([...arr]) => {
  let m = arr.length;
  while (m) {
    const i = Math.floor(Math.random() * m--);
    [arr[m], arr[i]] = [arr[i], arr[m]];
  }
  return arr;
};
```

```
const foo = [1, 2, 3];
shuffle(foo); // [2, 3, 1], foo = [1, 2, 3]
```

title: similarity

Returns an array of elements that appear in both arrays.

- Use `Array.prototype.includes()` to determine values that are not part of `values` .
- Use `Array.prototype.filter()` to remove them.

```
const similarity = (arr, values) => arr.filter(v => values.includes(v));
```

```
similarity([1, 2, 3], [1, 2, 4]); // [1, 2]
```

title: size

Gets the size of an array, object or string.

- Get type of `val` (`array` , `object` or `string`).
- Use `Array.prototype.length` property for arrays.
- Use `length` or `size` value if available or number of keys for objects.
- Use `size` of a `Blob` object created from `val` for strings.
- Split strings into array of characters with `split('')` and return its length.

```
const size = val =>
  Array.isArray(val)
    ? val.length
    : val && typeof val === 'object'
      ? val.size || val.length || Object.keys(val).length
      : typeof val === 'string'
        ? new Blob([val]).size
        : 0;


size([1, 2, 3, 4, 5]); // 5
size('size'); // 4
size({ one: 1, two: 2, three: 3 }); // 3
```

title: sleep

Delays the execution of an asynchronous function.

- Delay executing part of an `async` function, by putting it to sleep, returning a `new Promise()` .

```
const sleep = ms => new Promise(resolve => setTimeout(resolve, ms));


async function sleepyWork() {
  console.log("I'm going to sleep for 1 second.");
  await sleep(1000);
  console.log('I woke up after 1 second.');
}
```

title: slugify

Converts a string to a URL-friendly slug.

- Use `String.prototype.toLowerCase()` and `String.prototype.trim()` to normalize the string.
- Use `String.prototype.replace()` to replace spaces, dashes and underscores with `-` and remove special characters.

```js
const slugify = str =>
  str
    .toLowerCase()
    .trim()
    .replace(/[^\w\s-]/g, '')
    .replace(/[\s_-]+/g, '-')
    .replace(/^-+|-+$/g, '');

slugify('Hello World!'); // 'hello-world'
```

title: smoothScroll

Smoothly scrolls the element on which it's called into the visible area of the browser window.

- Use `Element.scrollIntoView()` to scroll the element.
- Use `{ behavior: 'smooth' }` to scroll smoothly.

```js
const smoothScroll = element =>
  document.querySelector(element).scrollIntoView({
    behavior: 'smooth'
  });

smoothScroll('#fooBar'); // scrolls smoothly to the element with the id fooBar
smoothScroll('.fooBar');
// scrolls smoothly to the first element with a class of fooBar
```

title: sortCharactersInString

Alphabetically sorts the characters in a string.

- Use the spread operator (`...`), `Array.prototype.sort()` and `String.prototype.localeCompare()` to sort the characters in `str` .
- Recombine using `String.prototype.join('')` .

```js
const sortCharactersInString = str =>
  [...str].sort((a, b) => a.localeCompare(b)).join('');
```

```
sortCharactersInString('cabbage'); // 'aabbceg'
```

title: sortedIndex

Finds the lowest index at which a value should be inserted into an array in order to maintain its sorting order.

- Loosely check if the array is sorted in descending order.
- Use `Array.prototype.findIndex()` to find the appropriate index where the element should be inserted.

```
const sortedIndex = (arr, n) => {
  const isDescending = arr[0] > arr[arr.length - 1];
  const index = arr.findIndex(el => (isDescending ? n >= el : n <= el));
  return index === -1 ? arr.length : index;
};
```

```
sortedIndex([5, 3, 2, 1], 4); // 1
sortedIndex([30, 50], 40); // 1
```

title: sortedIndexBy

Finds the lowest index at which a value should be inserted into an array in order to maintain its sorting order, based on the provided iterator function.

- Loosely check if the array is sorted in descending order.
- Use `Array.prototype.findIndex()` to find the appropriate index where the element should be inserted, based on the iterator function `fn`.

```
const sortedIndexBy = (arr, n, fn) => {
  const isDescending = fn(arr[0]) > fn(arr[arr.length - 1]);
  const val = fn(n);
  const index = arr.findIndex(el =>
    isDescending ? val >= fn(el) : val <= fn(el)
  );
  return index === -1 ? arr.length : index;
};
```

```
sortedIndexBy([{ x: 4 }, { x: 5 }], { x: 4 }, o => o.x); // 0
```

title: sortedLastIndex

Finds the highest index at which a value should be inserted into an array in order to maintain its sort order.

- Loosely check if the array is sorted in descending order.
- Use `Array.prototype.reverse()` and `Array.prototype.findIndex()` to find the appropriate last index where the element should be inserted.

```
const sortedLastIndex = (arr, n) => {
  const isDescending = arr[0] > arr[arr.length - 1];
  const index = arr
    .reverse()
    .findIndex(el => (isDescending ? n <= el : n >= el));
  return index === -1 ? 0 : arr.length - index;
};
```

```
sortedLastIndex([10, 20, 30, 30, 40], 30); // 4
```

title: sortedLastIndexBy

Finds the highest index at which a value should be inserted into an array in order to maintain its sort order, based on a provided iterator function.

- Loosely check if the array is sorted in descending order.
- Use `Array.prototype.map()` to apply the iterator function to all elements of the array.
- Use `Array.prototype.reverse()` and `Array.prototype.findIndex()` to find the appropriate last index where the element should be inserted, based on the provided iterator function.

```
const sortedLastIndexBy = (arr, n, fn) => {
  const isDescending = fn(arr[0]) > fn(arr[arr.length - 1]);
  const val = fn(n);
  const index = arr
    .map(fn)
    .reverse()
    .findIndex(el => (isDescending ? val <= el : val >= el));
  return index === -1 ? 0 : arr.length - index;
};
```

```
sortedLastIndexBy([{ x: 4 }, { x: 5 }], { x: 4 }, o => o.x); // 1
```

title: splitLines

Splits a multiline string into an array of lines.

- Use `String.prototype.split()` and a regular expression to match line breaks and create an array.

```
const splitLines = str => str.split(/\r?\n/);
```

```
splitLines('This\nis a\nmultiline\nstring.\n');
// ['This', 'is a', 'multiline', 'string.' , '']
```

title: spreadOver

Takes a variadic function and returns a function that accepts an array of arguments.

- Use a closure and the spread operator (`...`) to map the array of arguments to the inputs of the function.

```
const spreadOver = fn => argsArr => fn(...argsArr);
```

```
const arrayMax = spreadOver(Math.max);
arrayMax([1, 2, 3]); // 3
```

title: stableSort

Performs stable sorting of an array, preserving the initial indexes of items when their values are the same.

- Use `Array.prototype.map()` to pair each element of the input array with its corresponding index.
- Use `Array.prototype.sort()` and a `compare` function to sort the list, preserving their initial order if the items compared are equal.
- Use `Array.prototype.map()` to convert back to the initial array items.
- Does not mutate the original array, but returns a new array instead.

```
const stableSort = (arr, compare) =>
  arr
    .map((item, index) => ({ item, index }))
    .sort((a, b) => compare(a.item, b.item) || a.index - b.index)
    .map(({ item }) => item);
```

```
const arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const stable = stableSort(arr, () => 0); // [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

title: standardDeviation

Calculates the standard deviation of an array of numbers.

- Use `Array.prototype.reduce()` to calculate the mean, variance and the sum of the variance of the values and determine the standard deviation.
- Omit the second argument, `usePopulation`, to get the sample standard deviation or set it to `true` to get the population standard deviation.

```
const standardDeviation = (arr, usePopulation = false) => {
  const mean = arr.reduce((acc, val) => acc + val, 0) / arr.length;
  return Math.sqrt(
    arr
      .reduce((acc, val) => acc.concat((val - mean) ** 2), [])
      .reduce((acc, val) => acc + val, 0) /
      (arr.length - (usePopulation ? 0 : 1))
  );
};
```

```
standardDeviation([10, 2, 38, 23, 38, 23, 21]); // 13.284434142114991 (sample)
standardDeviation([10, 2, 38, 23, 38, 23, 21], true);
// 12.29899614287479 (population)
```

title: stringPermutations

Generates all permutations of a string (contains duplicates).

- Use recursion.
- For each letter in the given string, create all the partial permutations for the rest of its letters.
- Use `Array.prototype.map()` to combine the letter with each partial permutation.
- Use `Array.prototype.reduce()` to combine all permutations in one array.
- Base cases are for `String.prototype.length` equal to `2` or `1` .
- ⚠ **WARNING**: The execution time increases exponentially with each character. Anything more than 8 to 10 characters will cause your environment to hang as it tries to solve all the different combinations.

```
const stringPermutations = str => {
  if (str.length <= 2) return str.length === 2 ? [str, str[1] + str[0]] : [str];
  return str
    .split('')
    .reduce(
      (acc, letter, i) =>
        acc.concat(
          stringPermutations(str.slice(0, i) + str.slice(i + 1)).map(
            val => letter + val
          )
        ),
      []
    );
};
```

```
stringPermutations('abc'); // ['abc', 'acb', 'bac', 'bca', 'cab', 'cba']
```

title: stringifyCircularJSON

Serializes a JSON object containing circular references into a JSON format.

- Create a `new WeakSet()` to store and check seen values, using `WeakSet.prototype.add()` and `WeakSet.prototype.has()` .
- Use `JSON.stringify()` with a custom replacer function that omits values already in `seen` , adding new values as necessary.
- ⚠ **NOTICE:** This function finds and removes circular references, which causes circular data loss in the serialized JSON.

```
const stringifyCircularJSON = obj => {
  const seen = new WeakSet();
  return JSON.stringify(obj, (k, v) => {
    if (v !== null && typeof v === 'object') {
      if (seen.has(v)) return;
      seen.add(v);
    }
    return v;
  });
};


const obj = { n: 42 };
obj.obj = obj;
stringifyCircularJSON(obj); // '{"n": 42}'
```

title: stripHTMLTags

Removes HTML/XML tags from string.

- Use a regular expression to remove HTML/XML tags from a string.

```
const stripHTMLTags = str => str.replace(/<[^>]*>/g, '');


stripHTMLTags('<p><em>lorem</em> <strong>ipsum</strong></p>'); // 'lorem ipsum'
```

title: subSet

Checks if the first iterable is a subset of the second one, excluding duplicate values.

- Use the `new Set()` constructor to create a new `Set` object from each iterable.

- Use `Array.prototype.every()` and `Set.prototype.has()` to check that each value in the first iterable is contained in the second one.

```
const subSet = (a, b) => {
  const sA = new Set(a), sB = new Set(b);
  return [...sA].every(v => sB.has(v));
};
```

```
subSet(new Set([1, 2]), new Set([1, 2, 3, 4])); // true
subSet(new Set([1, 5]), new Set([1, 2, 3, 4])); // false
```

title: sum

Calculates the sum of two or more numbers/arrays.

- Use `Array.prototype.reduce()` to add each value to an accumulator, initialized with a value of `0` .

```
const sum = (...arr) => [...arr].reduce((acc, val) => acc + val, 0);
```

```
sum(1, 2, 3, 4); // 10
sum(...[1, 2, 3, 4]); // 10
```

title: sumBy

Calculates the sum of an array, after mapping each element to a value using the provided function.

- Use `Array.prototype.map()` to map each element to the value returned by `fn` .
- Use `Array.prototype.reduce()` to add each value to an accumulator, initialized with a value of `0` .

```
const sumBy = (arr, fn) =>
  arr
    .map(typeof fn === 'function' ? fn : val => val[fn])
    .reduce((acc, val) => acc + val, 0);
```

```
sumBy([{ n: 4 }, { n: 2 }, { n: 8 }, { n: 6 }], x => x.n); // 20
sumBy([{ n: 4 }, { n: 2 }, { n: 8 }, { n: 6 }], 'n'); // 20
```

title: sumN

Sums all the numbers between `1` and `n`.

- Use the formula `(n * (n + 1)) / 2` to get the sum of all the numbers between 1 and `n`.

```
const sumN = n => (n * (n + 1)) / 2;


sumN(100); // 5050
```

title: sumPower

Calculates the sum of the powers of all the numbers from `start` to `end` (both inclusive).

- Use `Array.prototype.fill()` to create an array of all the numbers in the target range.
- Use `Array.prototype.map()` and the exponent operator (`**`) to raise them to `power` and `Array.prototype.reduce()` to add them together.
- Omit the second argument, `power`, to use a default power of `2`.
- Omit the third argument, `start`, to use a default starting value of `1`.

```
const sumPower = (end, power = 2, start = 1) =>
  Array(end + 1 - start)
    .fill(0)
    .map((x, i) => (i + start) ** power)
    .reduce((a, b) => a + b, 0);


sumPower(10); // 385
sumPower(10, 3); // 3025
sumPower(10, 3, 5); // 2925
```

title: superSet

Checks if the first iterable is a superset of the second one, excluding duplicate values.

- Use the `new Set()` constructor to create a new `Set` object from each iterable.
- Use `Array.prototype.every()` and `Set.prototype.has()` to check that each value in the second iterable is contained in the first one.

```
const superSet = (a, b) => {
  const sA = new Set(a), sB = new Set(b);
  return [...sB].every(v => sA.has(v));
};
```

```
superSet(new Set([1, 2, 3, 4]), new Set([1, 2])); // true
superSet(new Set([1, 2, 3, 4]), new Set([1, 5])); // false
```

title: supportsTouchEvents

Checks if touch events are supported.

- Check if `'ontouchstart'` exists in `window` .

```
const supportsTouchEvents = () =>
  window && 'ontouchstart' in window;
```

```
supportsTouchEvents(); // true
```

title: swapCase

Creates a string with uppercase characters converted to lowercase and vice versa.

- Use the spread operator (`...`) to convert `str` into an array of characters.
- Use `String.prototype.toLowerCase()` and `String.prototype.toUpperCase()` to convert lowercase characters to uppercase and vice versa.
- Use `Array.prototype.map()` to apply the transformation to each character, `Array.prototype.join()` to combine back into a string.
- Note that it is not necessarily true that `swapCase(swapCase(str)) === str` .

```
const swapCase = str =>
  [...str]
    .map(c => (c === c.toLowerCase() ? c.toUpperCase() : c.toLowerCase()))
    .join('');


swapCase('Hello world!'); // 'hELLO WORLD!'
```

title: symbolizeKeys

Creates a new object, converting each key to a `Symbol`.

- Use `Object.keys()` to get the keys of `obj`.
- Use `Array.prototype.reduce()` and `Symbol()` to create a new object where each key is converted to a `Symbol`.

```
const symbolizeKeys = obj =>
  Object.keys(obj).reduce(
    (acc, key) => ({ ...acc, [Symbol(key)]: obj[key] }),
    {}
  );


symbolizeKeys({ id: 10, name: 'apple' });
// { [Symbol(id)]: 10, [Symbol(name)]: 'apple' }
```

title: symmetricDifference

Returns the symmetric difference between two arrays, without filtering out duplicate values.

- Create a `new Set()` from each array to get the unique values of each one.
- Use `Array.prototype.filter()` on each of them to only keep values not contained in the other.

```
const symmetricDifference = (a, b) => {
  const sA = new Set(a),
    sB = new Set(b);
  return [...a.filter(x => !sB.has(x)), ...b.filter(x => !sA.has(x))];
};
```

```
symmetricDifference([1, 2, 3], [1, 2, 4]); // [3, 4]
symmetricDifference([1, 2, 2], [1, 3, 1]); // [2, 2, 3]
```

title: symmetricDifferenceBy

Returns the symmetric difference between two arrays, after applying the provided function to each array element of both.

- Create a `new Set()` from each array to get the unique values of each one after applying `fn` to them.
- Use `Array.prototype.filter()` on each of them to only keep values not contained in the other.

```
const symmetricDifferenceBy = (a, b, fn) => {
  const sA = new Set(a.map(v => fn(v))),
    sB = new Set(b.map(v => fn(v)));
  return [...a.filter(x => !sB.has(fn(x))), ...b.filter(x => !sA.has(fn(x)))];
};
```

```
symmetricDifferenceBy([2.1, 1.2], [2.3, 3.4], Math.floor); // [ 1.2, 3.4 ]
symmetricDifferenceBy(
  [{ id: 1 }, { id: 2 }, { id: 3 }],
  [{ id: 1 }, { id: 2 }, { id: 4 }],
  i => i.id
);
// [{ id: 3 }, { id: 4 }]
```

title: symmetricDifferenceWith

Returns the symmetric difference between two arrays, using a provided function as a comparator.

- Use `Array.prototype.filter()` and `Array.prototype.findIndex()` to find the appropriate values.

```
const symmetricDifferenceWith = (arr, val, comp) => [
  ...arr.filter(a => val.findIndex(b => comp(a, b)) === -1),
  ...val.filter(a => arr.findIndex(b => comp(a, b)) === -1)
];
```

```js
symmetricDifferenceWith(
  [1, 1.2, 1.5, 3, 0],
  [1.9, 3, 0, 3.9],
  (a, b) => Math.round(a) === Math.round(b)
); // [1, 1.2, 3.9]
```

title: tail

Returns all elements in an array except for the first one.

- Return `Array.prototype.slice(1)` if `Array.prototype.length` is more than `1`, otherwise, return the whole array.

```js
const tail = arr => (arr.length > 1 ? arr.slice(1) : arr);
```

```js
tail([1, 2, 3]); // [2, 3]
tail([1]); // [1]
```

title: take

Creates an array with `n` elements removed from the beginning.

- Use `Array.prototype.slice()` to create a slice of the array with `n` elements taken from the beginning.

```js
const take = (arr, n = 1) => arr.slice(0, n);
```

```js
take([1, 2, 3], 5); // [1, 2, 3]
take([1, 2, 3], 0); // []
```

title: takeRight

Creates an array with `n` elements removed from the end.

- Use `Array.prototype.slice()` to create a slice of the array with `n` elements taken from the end.

```
const takeRight = (arr, n = 1) => arr.slice(arr.length - n, arr.length);


takeRight([1, 2, 3], 2); // [ 2, 3 ]
takeRight([1, 2, 3]); // [3]
```

title: takeRightUntil

Removes elements from the end of an array until the passed function returns `true` .
Returns the removed elements.

- Create a reversed copy of the array, using the spread operator (`...`) and `Array.prototype.reverse()` .
- Loop through the reversed copy, using a `for...of` loop over `Array.prototype.entries()` until the returned value from the function is truthy.
- Return the removed elements, using `Array.prototype.slice()` .
- The callback function, `fn` , accepts a single argument which is the value of the element.

```
const takeRightUntil = (arr, fn) => {
  for (const [i, val] of [...arr].reverse().entries())
    if (fn(val)) return i === 0 ? [] : arr.slice(-i);
  return arr;
};


takeRightUntil([1, 2, 3, 4], n => n < 3); // [3, 4]
```

title: takeRightWhile

Removes elements from the end of an array until the passed function returns `false` .
Returns the removed elements.

- Create a reversed copy of the array, using the spread operator (`...`) and `Array.prototype.reverse()` .

- Loop through the reversed copy, using a `for...of` loop over `Array.prototype.entries()` until the returned value from the function is falsy.
- Return the removed elements, using `Array.prototype.slice()` .
- The callback function, `fn` , accepts a single argument which is the value of the element.

```
const takeRightWhile = (arr, fn) => {
  for (const [i, val] of [...arr].reverse().entries())
    if (!fn(val)) return i === 0 ? [] : arr.slice(-i);
  return arr;
};
```

```
takeRightWhile([1, 2, 3, 4], n => n >= 3); // [3, 4]
```

title: takeUntil

Removes elements in an array until the passed function returns `true` .
Returns the removed elements.

- Loop through the array, using a `for...of` loop over `Array.prototype.entries()` until the returned value from the function is truthy.
- Return the removed elements, using `Array.prototype.slice()` .
- The callback function, `fn` , accepts a single argument which is the value of the element.

```
const takeUntil = (arr, fn) => {
  for (const [i, val] of arr.entries()) if (fn(val)) return arr.slice(0, i);
  return arr;
};
```

```
takeUntil([1, 2, 3, 4], n => n >= 3); // [1, 2]
```

title: takeWhile

Removes elements in an array until the passed function returns `false` .
Returns the removed elements.

- Loop through the array, using a `for...of` loop over `Array.prototype.entries()` until the returned value from the function is falsy.
- Return the removed elements, using `Array.prototype.slice()` .
- The callback function, `fn` , accepts a single argument which is the value of the element.

```
const takeWhile = (arr, fn) => {
  for (const [i, val] of arr.entries()) if (!fn(val)) return arr.slice(0, i);
  return arr;
};
```

```
takeWhile([1, 2, 3, 4], n => n < 3); // [1, 2]
```

title: throttle

Creates a throttled function that only invokes the provided function at most once per every `wait` milliseconds

- Use `setTimeout()` and `clearTimeout()` to throttle the given method, `fn` .
- Use `Function.prototype.apply()` to apply the `this` context to the function and provide the necessary `arguments` .
- Use `Date.now()` to keep track of the last time the throttled function was invoked.
- Use a variable, `inThrottle` , to prevent a race condition between the first execution of `fn` and the next loop.
- Omit the second argument, `wait` , to set the timeout at a default of `0` ms.

```
const throttle = (fn, wait) => {
  let inThrottle, lastFn, lastTime;
  return function() {
    const context = this,
      args = arguments;
    if (!inThrottle) {
      fn.apply(context, args);
      lastTime = Date.now();
      inThrottle = true;
    } else {
      clearTimeout(lastFn);
      lastFn = setTimeout(function() {
        if (Date.now() - lastTime >= wait) {
          fn.apply(context, args);
          lastTime = Date.now();
        }
      }, Math.max(wait - (Date.now() - lastTime), 0));
    }
  };
};


window.addEventListener(
  'resize',
  throttle(function(evt) {
    console.log(window.innerWidth);
    console.log(window.innerHeight);
  }, 250)
); // Will log the window dimensions at most every 250ms
```

title: timeTaken

Measures the time it takes for a function to execute.

- Use `Console.time()` and `Console.timeEnd()` to measure the difference between the start and end times to determine how long the callback took to execute.

```
const timeTaken = callback => {
  console.time('timeTaken');
  const r = callback();
  console.timeEnd('timeTaken');
  return r;
};
```

```
timeTaken(() => Math.pow(2, 10)); // 1024, (logged): timeTaken: 0.02099609375ms
```

title: times

Iterates over a callback `n` times

- Use `Function.prototype.call()` to call `fn` `n` times or until it returns `false` .
- Omit the last argument, `context` , to use an `undefined` object (or the global object in non-strict mode).

```
const times = (n, fn, context = undefined) => {
  let i = 0;
  while (fn.call(context, i) !== false && ++i < n) {}
};
```

```
var output = '';
times(5, i => (output += i));
console.log(output); // 01234
```

title: toCamelCase

Converts a string to camelcase.

- Use `String.prototype.match()` to break the string into words using an appropriate regexp.
- Use `Array.prototype.map()` , `Array.prototype.slice()` , `Array.prototype.join()` , `String.prototype.toLowerCase()` and `String.prototype.toUpperCase()` to combine them, capitalizing the first letter of each one.

```
const toCamelCase = str => {
  const s =
    str &&
    str
      .match(
        /[A-Z]{2,}(?=[A-Z][a-z]+[0-9]*|\b)|[A-Z]?[a-z]+[0-9]*|[A-Z]|[0-9]+/g
      )
      .map(x => x.slice(0, 1).toUpperCase() + x.slice(1).toLowerCase())
      .join('');
  return s.slice(0, 1).toLowerCase() + s.slice(1);
};
```

```
toCamelCase('some_database_field_name'); // 'someDatabaseFieldName'
toCamelCase('Some label that needs to be camelized');
// 'someLabelThatNeedsToBeCamelized'
toCamelCase('some-javascript-property'); // 'someJavascriptProperty'
toCamelCase('some-mixed_string with spaces_underscores-and-hyphens');
// 'someMixedStringWithSpacesUnderscoresAndHyphens'
```

title: toCharArray

Converts a string to an array of characters.

- Use the spread operator (`...`) to convert the string into an array of characters.

```
const toCharArray = s => [...s];
```

```
toCharArray('hello'); // ['h', 'e', 'l', 'l', 'o']
```

title: toCurrency

Takes a number and returns it in the specified currency formatting.

- Use `Intl.NumberFormat` to enable country / currency sensitive formatting.

```
const toCurrency = (n, curr, LanguageFormat = undefined) =>
  Intl.NumberFormat(LanguageFormat, {
    style: 'currency',
    currency: curr,
  }).format(n);


toCurrency(123456.789, 'EUR');
// &euro;123,456.79  | currency: Euro | currencyLangFormat: Local
toCurrency(123456.789, 'USD', 'en-us');
// $123,456.79  | currency: US Dollar | currencyLangFormat: English (United States)
toCurrency(123456.789, 'USD', 'fa');
// ۱۲۳,۴۵۶,۷۹ $ | currency: US Dollar | currencyLangFormat: Farsi
toCurrency(322342436423.2435, 'JPY');
// ¥322,342,436,423 | currency: Japanese Yen | currencyLangFormat: Local
toCurrency(322342436423.2435, 'JPY', 'fi');
// 322 342 436 423 ¥ | currency: Japanese Yen | currencyLangFormat: Finnish
```

title: toDecimalMark

Converts a number to a decimal mark formatted string.

- Use `Number.prototype.toLocaleString()` to convert the number to decimal mark format.

```
const toDecimalMark = num => num.toLocaleString('en-US');


toDecimalMark(12305030388.9087); // '12,305,030,388.909'
```

title: toHSLArray

Converts an `hsl()` color string to an array of values.

- Use `String.prototype.match()` to get an array of 3 string with the numeric values.
- Use `Array.prototype.map()` in combination with `Number` to convert them into an array of numeric values.

```
const toHSLArray = hslStr => hslStr.match(/\d+/g).map(Number);
```

```
toHSLArray('hsl(50, 10%, 10%)'); // [50, 10, 10]
```

title: toHSLObject

Converts an `hsl()` color string to an object with the values of each color.

- Use `String.prototype.match()` to get an array of 3 string with the numeric values.
- Use `Array.prototype.map()` in combination with `Number` to convert them into an array of numeric values.
- Use array destructuring to store the values into named variables and create an appropriate object from them.

```
const toHSLObject = hslStr => {
  const [hue, saturation, lightness] = hslStr.match(/\d+/g).map(Number);
  return { hue, saturation, lightness };
};
```

```
toHSLObject('hsl(50, 10%, 10%)'); // { hue: 50, saturation: 10, lightness: 10 }
```

title: toHash

Reduces a given array-like into a value hash (keyed data store).

- Given an iterable object or array-like structure, call `Array.prototype.reduce.call()` on the provided object to step over it and return an `Object` , keyed by the reference value.

```
const toHash = (object, key) =>
  Array.prototype.reduce.call(
    object,
    (acc, data, index) => ((acc[!key ? index : data[key]] = data), acc),
    {}
  );
```

```js
toHash([4, 3, 2, 1]); // { 0: 4, 1: 3, 2: 2, 3: 1 }
toHash([{ a: 'label' }], 'a'); // { label: { a: 'label' } }
// A more in depth example:
let users = [
  { id: 1, first: 'Jon' },
  { id: 2, first: 'Joe' },
  { id: 3, first: 'Moe' },
];
let managers = [{ manager: 1, employees: [2, 3] }];
// We use function here because we want a bindable reference,
// but a closure referencing the hash would work, too.
managers.forEach(
  manager =>
    (manager.employees = manager.employees.map(function(id) {
      return this[id];
    }, toHash(users, 'id')))
);
managers;
// [ {manager:1, employees: [ {id: 2, first: 'Joe'}, {id: 3, first: 'Moe'} ] } ]
```

title: toISOStringWithTimezone

Converts a date to extended ISO format (ISO 8601), including timezone offset.

- Use `Date.prototype.getTimezoneOffset()` to get the timezone offset and reverse it. Store its sign in `diff`.
- Define a helper function, `pad`, that normalizes any passed number to an integer using `Math.floor()` and `Math.abs()` and pads it to `2` digits, using `String.prototype.padStart()`.
- Use `pad()` and the built-in methods in the `Date` prototype to build the ISO 8601 string with timezone offset.

```js
const toISOStringWithTimezone = date => {
  const tzOffset = -date.getTimezoneOffset();
  const diff = tzOffset >= 0 ? '+' : '-';
  const pad = n => `${Math.floor(Math.abs(n))}`.padStart(2, '0');
  return date.getFullYear() +
    '-' + pad(date.getMonth() + 1) +
    '-' + pad(date.getDate()) +
    'T' + pad(date.getHours()) +
    ':' + pad(date.getMinutes()) +
    ':' + pad(date.getSeconds()) +
    diff + pad(tzOffset / 60) +
    ':' + pad(tzOffset % 60);
};
```

```js
toISOStringWithTimezone(new Date()); // '2020-10-06T20:43:33-04:00'
```

title: toKebabCase

Converts a string to kebab case.

- Use `String.prototype.match()` to break the string into words using an appropriate regexp.
- Use `Array.prototype.map()` , `Array.prototype.join()` and `String.prototype.toLowerCase()` to combine them, adding `-` as a separator.

```js
const toKebabCase = str =>
  str &&
  str
    .match(/[A-Z]{2,}(?=[A-Z][a-z]+[0-9]*|\b)|[A-Z]?[a-z]+[0-9]*|[A-Z]|[0-9]+/g)
    .map(x => x.toLowerCase())
    .join('-');
```

```js
toKebabCase('camelCase'); // 'camel-case'
toKebabCase('some text'); // 'some-text'
toKebabCase('some-mixed_string With spaces_underscores-and-hyphens');
// 'some-mixed-string-with-spaces-underscores-and-hyphens'
toKebabCase('AllThe-small Things'); // 'all-the-small-things'
toKebabCase('IAmEditingSomeXMLAndHTML');
// 'i-am-editing-some-xml-and-html'
```

title: toOrdinalSuffix

Takes a number and returns it as a string with the correct ordinal indicator suffix.

- Use the modulo operator (`%`) to find values of single and tens digits.
- Find which ordinal pattern digits match.
- If digit is found in teens pattern, use teens ordinal.

```
const toOrdinalSuffix = num => {
  const int = parseInt(num),
    digits = [int % 10, int % 100],
    ordinals = ['st', 'nd', 'rd', 'th'],
    oPattern = [1, 2, 3, 4],
    tPattern = [11, 12, 13, 14, 15, 16, 17, 18, 19];
  return oPattern.includes(digits[0]) && !tPattern.includes(digits[1])
    ? int + ordinals[digits[0] - 1]
    : int + ordinals[3];
};
```

```
toOrdinalSuffix('123'); // '123rd'
```

title: toPairs

Creates an array of key-value pair arrays from an object or other iterable.

- Check if `Symbol.iterator` is defined and, if so, use `Array.prototype.entries()` to get an iterator for the given iterable.
- Use `Array.from()` to convert the result to an array of key-value pair arrays.
- If `Symbol.iterator` is not defined for `obj` , use `Object.entries()` instead.

```
const toPairs = obj =>
  obj[Symbol.iterator] instanceof Function && obj.entries instanceof Function
    ? Array.from(obj.entries())
    : Object.entries(obj);
```

```
toPairs({ a: 1, b: 2 }); // [['a', 1], ['b', 2]]
toPairs([2, 4, 8]); // [[0, 2], [1, 4], [2, 8]]
toPairs('shy'); // [['0', 's'], ['1', 'h'], ['2', 'y']]
toPairs(new Set(['a', 'b', 'c', 'a'])); // [['a', 'a'], ['b', 'b'], ['c', 'c']]
```

title: toPascalCase

Converts a string to pascal case.

- Use `String.prototype.match()` to break the string into words using an appropriate regexp.
- Use `Array.prototype.map()` , `Array.prototype.slice()` , `Array.prototype.join()` , `String.prototype.toUpperCase()` and `String.prototype.toLowerCase()` to combine them, capitalizing the first letter of each word and lowercasing the rest.

```
const toPascalCase = str =>
  str
    .match(/[A-Z]{2,}(?=[A-Z][a-z]+[0-9]*|\b)|[A-Z]?[a-z]+[0-9]*|[A-Z]|[0-9]+/g)
    .map(x => x.charAt(0).toUpperCase() + x.slice(1).toLowerCase())
    .join('');
```

```
toPascalCase('some_database_field_name'); // 'SomeDatabaseFieldName'
toPascalCase('Some label that needs to be pascalized');
// 'SomeLabelThatNeedsToBePascalized'
toPascalCase('some-javascript-property'); // 'SomeJavascriptProperty'
toPascalCase('some-mixed_string with spaces_underscores-and-hyphens');
// 'SomeMixedStringWithSpacesUnderscoresAndHyphens'
```

title: toRGBArray

Converts an `rgb()` color string to an array of values.

- Use `String.prototype.match()` to get an array of 3 string with the numeric values.
- Use `Array.prototype.map()` in combination with `Number` to convert them into an array of numeric values.

```
const toRGBArray = rgbStr => rgbStr.match(/\d+/g).map(Number);
```

```
toRGBArray('rgb(255, 12, 0)'); // [255, 12, 0]
```

title: toRGBObject

Converts an `rgb()` color string to an object with the values of each color.

- Use `String.prototype.match()` to get an array of 3 string with the numeric values.
- Use `Array.prototype.map()` in combination with `Number` to convert them into an array of numeric values.
- Use array destructuring to store the values into named variables and create an appropriate object from them.

```
const toRGBObject = rgbStr => {
  const [red, green, blue] = rgbStr.match(/\d+/g).map(Number);
  return { red, green, blue };
};
```

```
toRGBObject('rgb(255, 12, 0)'); // {red: 255, green: 12, blue: 0}
```

title: toRomanNumeral

Converts an integer to its roman numeral representation.
Accepts value between `1` and `3999` (both inclusive).

- Create a lookup table containing 2-value arrays in the form of (roman value, integer).
- Use `Array.prototype.reduce()` to loop over the values in `lookup` and repeatedly divide `num` by the value.
- Use `String.prototype.repeat()` to add the roman numeral representation to the accumulator.

```js
const toRomanNumeral = num => {
  const lookup = [
    ['M', 1000],
    ['CM', 900],
    ['D', 500],
    ['CD', 400],
    ['C', 100],
    ['XC', 90],
    ['L', 50],
    ['XL', 40],
    ['X', 10],
    ['IX', 9],
    ['V', 5],
    ['IV', 4],
    ['I', 1],
  ];
  return lookup.reduce((acc, [k, v]) => {
    acc += k.repeat(Math.floor(num / v));
    num = num % v;
    return acc;
  }, '');
};
```

```js
toRomanNumeral(3); // 'III'
toRomanNumeral(11); // 'XI'
toRomanNumeral(1998); // 'MCMXCVIII'
```

title: toSafeInteger

Converts a value to a safe integer.

- Use `Math.max()` and `Math.min()` to find the closest safe value.
- Use `Math.round()` to convert to an integer.

```js
const toSafeInteger = num =>
  Math.round(
    Math.max(Math.min(num, Number.MAX_SAFE_INTEGER), Number.MIN_SAFE_INTEGER)
  );
```

```js
toSafeInteger('3.2'); // 3
toSafeInteger(Infinity); // 9007199254740991
```

title: toSnakeCase

Converts a string to snake case.

- Use `String.prototype.match()` to break the string into words using an appropriate regexp.
- Use `Array.prototype.map()` , `Array.prototype.slice()` , `Array.prototype.join()` and `String.prototype.toLowerCase()` to combine them, adding `_` as a separator.

```
const toSnakeCase = str =>
  str &&
  str
    .match(/[A-Z]{2,}(?=[A-Z][a-z]+[0-9]*|\b)|[A-Z]?[a-z]+[0-9]*|[A-Z]|[0-9]+/g)
    .map(x => x.toLowerCase())
    .join('_');
```

```
toSnakeCase('camelCase'); // 'camel_case'
toSnakeCase('some text'); // 'some_text'
toSnakeCase('some-mixed_string With spaces_underscores-and-hyphens');
// 'some_mixed_string_with_spaces_underscores_and_hyphens'
toSnakeCase('AllThe-small Things'); // 'all_the_small_things'
toSnakeCase('IAmEditingSomeXMLAndHTML');
// 'i_am_editing_some_xml_and_html'
```

title: toTitleCase

Converts a string to title case.

- Use `String.prototype.match()` to break the string into words using an appropriate regexp.
- Use `Array.prototype.map()` , `Array.prototype.slice()` , `Array.prototype.join()` and `String.prototype.toUpperCase()` to combine them, capitalizing the first letter of each word and adding a whitespace between them.

```
const toTitleCase = str =>
  str
    .match(/[A-Z]{2,}(?=[A-Z][a-z]+[0-9]*|\b)|[A-Z]?[a-z]+[0-9]*|[A-Z]|[0-9]+/g)
    .map(x => x.charAt(0).toUpperCase() + x.slice(1))
    .join(' ');
```

```
toTitleCase('some_database_field_name'); // 'Some Database Field Name'
toTitleCase('Some label that needs to be title-cased');
// 'Some Label That Needs To Be Title Cased'
toTitleCase('some-package-name'); // 'Some Package Name'
toTitleCase('some-mixed_string with spaces_underscores-and-hyphens');
// 'Some Mixed String With Spaces Underscores And Hyphens'
```

title: toggleClass

Toggles a class for an HTML element.

- Use `Element.classList` and `DOMTokenList.toggle()` to toggle the specified class for the element.

```
const toggleClass = (el, className) => el.classList.toggle(className);
```

```
toggleClass(document.querySelector('p.special'), 'special');
// The paragraph will not have the 'special' class anymore
```

title: tomorrow

Results in a string representation of tomorrow's date.

- Use `new Date()` to get the current date.
- Increment it by one using `Date.prototype.getDate()` and set the value to the result using `Date.prototype.setDate()` .
- Use `Date.prototype.toISOString()` to return a string in `yyyy-mm-dd` format.

```
const tomorrow = () => {
  let d = new Date();
  d.setDate(d.getDate() + 1);
  return d.toISOString().split('T')[0];
};
```

```
tomorrow(); // 2018-10-19 (if current date is 2018-10-18)
```

title: transform

Applies a function against an accumulator and each key in the object (from left to right).

- Use `Object.keys()` to iterate over each key in the object.
- Use `Array.prototype.reduce()` to apply the specified function against the given accumulator.

```
const transform = (obj, fn, acc) =>
  Object.keys(obj).reduce((a, k) => fn(a, obj[k], k, obj), acc);
```

```
transform(
  { a: 1, b: 2, c: 1 },
  (r, v, k) => {
    (r[v] || (r[v] = [])).push(k);
    return r;
  },
  {}
); // { '1': ['a', 'c'], '2': ['b'] }
```

title: triggerEvent

Triggers a specific event on a given element, optionally passing custom data.

- Use `new CustomEvent()` to create an event from the specified `eventType` and details.
- Use `EventTarget.dispatchEvent()` to trigger the newly created event on the given element.
- Omit the third argument, `detail`, if you do not want to pass custom data to the triggered event.

```
const triggerEvent = (el, eventType, detail) =>
  el.dispatchEvent(new CustomEvent(eventType, { detail }));
```

```
triggerEvent(document.getElementById('myId'), 'click');
triggerEvent(document.getElementById('myId'), 'click', { username: 'bob' });
```

title: truncateString

Truncates a string up to a specified length.

- Determine if `String.prototype.length` is greater than `num`.
- Return the string truncated to the desired length, with `'...'` appended to the end or the original string.

```
const truncateString = (str, num) =>
  str.length > num ? str.slice(0, num > 3 ? num - 3 : num) + '...' : str;
```

```
truncateString('boomerang', 7); // 'boom...'
```

title: truncateStringAtWhitespace

Truncates a string up to specified length, respecting whitespace when possible.

- Determine if `String.prototype.length` is greater or equal to `lim`. If not, return it as-is.
- Use `String.prototype.slice()` and `String.prototype.lastIndexOf()` to find the index of the last space below the desired `lim`.
- Use `String.prototype.slice()` to appropriately truncate `str` based on `lastSpace`, respecting whitespace if possible and appending `ending` at the end.
- Omit the third argument, `ending`, to use the default ending of `'...'`.

```
const truncateStringAtWhitespace = (str, lim, ending = '...') => {
  if (str.length <= lim) return str;
  const lastSpace = str.slice(0, lim - ending.length + 1).lastIndexOf(' ');
  return str.slice(0, lastSpace > 0 ? lastSpace : lim - ending.length) + ending;
};
```

```
truncateStringAtWhitespace('short', 10); // 'short'
truncateStringAtWhitespace('not so short', 10); // 'not so...'
truncateStringAtWhitespace('trying a thing', 10); // 'trying...'
truncateStringAtWhitespace('javascripting', 10); // 'javascr...'
```

title: truthCheckCollection

Checks if the predicate function is truthy for all elements of a collection.

- Use `Array.prototype.every()` to check if each passed object has the specified property and if it returns a truthy value.

```
const truthCheckCollection = (collection, pre) =>
  collection.every(obj => obj[pre]);


truthCheckCollection(
  [
    { user: 'Tinky-Winky', sex: 'male' },
    { user: 'Dipsy', sex: 'male' },
  ],
  'sex'
); // true
```

title: unary

Creates a function that accepts up to one argument, ignoring any additional arguments.

- Call the provided function, `fn`, with just the first argument supplied.

```
const unary = fn => val => fn(val);


['6', '8', '10'].map(unary(parseInt)); // [6, 8, 10]
```

title: uncurry

Uncurries a function up to depth `n`.

- Return a variadic function.
- Use `Array.prototype.reduce()` on the provided arguments to call each subsequent curry level of the function.
- If the `length` of the provided arguments is less than `n` throw an error.
- Otherwise, call `fn` with the proper amount of arguments, using `Array.prototype.slice(0, n)`.
- Omit the second argument, `n`, to uncurry up to depth `1`.

```
const uncurry = (fn, n = 1) => (...args) => {
  const next = acc => args => args.reduce((x, y) => x(y), acc);
  if (n > args.length) throw new RangeError('Arguments too few!');
  return next(fn)(args.slice(0, n));
};
```

```
const add = x => y => z => x + y + z;
const uncurriedAdd = uncurry(add, 3);
uncurriedAdd(1, 2, 3); // 6
```

title: unescapeHTML

Unescapes escaped HTML characters.

- Use `String.prototype.replace()` with a regexp that matches the characters that need to be unescaped.
- Use the function's callback to replace each escaped character instance with its associated unescaped character using a dictionary (object).

```
const unescapeHTML = str =>
  str.replace(
    /&amp;|&lt;|&gt;|&#39;|&quot;/g,
    tag =>
      ({
        '&amp;': '&',
        '&lt;': '<',
        '&gt;': '>',
        '&#39;': "'",
        '&quot;': '"'
      }[tag] || tag)
  );
```

```
unescapeHTML('&lt;a href=&quot;#&quot;&gt;Me &amp; you&lt;/a&gt;');
// '<a href="#">Me & you</a>'
```

title: unflattenObject

Unflatten an object with the paths for keys.

- Use nested `Array.prototype.reduce()` to convert the flat path to a leaf node.
- Use `String.prototype.split('.')` to split each key with a dot delimiter and `Array.prototype.reduce()` to add objects against the keys.
- If the current accumulator already contains a value against a particular key, return its value as the next accumulator.
- Otherwise, add the appropriate key-value pair to the accumulator object and return the value as the accumulator.

```js
const unflattenObject = obj =>
  Object.keys(obj).reduce((res, k) => {
    k.split('.').reduce(
      (acc, e, i, keys) =>
        acc[e] ||
        (acc[e] = isNaN(Number(keys[i + 1]))
          ? keys.length - 1 === i
            ? obj[k]
            : {}
          : []),
      res
    );
    return res;
  }, {});
```

```js
unflattenObject({ 'a.b.c': 1, d: 1 }); // { a: { b: { c: 1 } }, d: 1 }
unflattenObject({ 'a.b': 1, 'a.c': 2, d: 3 }); // { a: { b: 1, c: 2 }, d: 3 }
unflattenObject({ 'a.b.0': 8, d: 3 }); // { a: { b: [ 8 ] }, d: 3 }
```

title: unfold

Builds an array, using an iterator function and an initial seed value.

- Use a `while` loop and `Array.prototype.push()` to call the function repeatedly until it returns `false` .
- The iterator function accepts one argument (`seed`) and must always return an array with two elements ([`value` , `nextSeed`]) or `false` to terminate.

```
const unfold = (fn, seed) => {
  let result = [],
    val = [null, seed];
  while ((val = fn(val[1]))) result.push(val[0]);
  return result;
};
```

```
var f = n => (n > 50 ? false : [-n, n + 10]);
unfold(f, 10); // [-10, -20, -30, -40, -50]
```

title: union

Returns every element that exists in any of the two arrays at least once.

- Create a `new Set()` with all values of `a` and `b` and convert it to an array.

```
const union = (a, b) => Array.from(new Set([...a, ...b]));
```

```
union([1, 2, 3], [4, 3, 2]); // [1, 2, 3, 4]
```

title: unionBy

Returns every element that exists in any of the two arrays at least once, after applying the provided function to each array element of both.

- Create a `new Set()` by applying all `fn` to all values of `a` .
- Create a `new Set()` from `a` and all elements in `b` whose value, after applying `fn` does not match a value in the previously created set.
- Return the last set converted to an array.

```
const unionBy = (a, b, fn) => {
  const s = new Set(a.map(fn));
  return Array.from(new Set([...a, ...b.filter(x => !s.has(fn(x)))]));
};
```

```
unionBy([2.1], [1.2, 2.3], Math.floor); // [2.1, 1.2]
unionBy([{ id: 1 }, { id: 2 }], [{ id: 2 }, { id: 3 }], x => x.id)
// [{ id: 1 }, { id: 2 }, { id: 3 }]
```

title: unionWith

Returns every element that exists in any of the two arrays at least once, using a provided comparator function.

- Create a `new Set()` with all values of `a` and values in `b` for which the comparator finds no matches in `a`, using `Array.prototype.findIndex()`.

```
const unionWith = (a, b, comp) =>
  Array.from(
    new Set([...a, ...b.filter(x => a.findIndex(y => comp(x, y)) === -1)])
  );
```

```
unionWith(
  [1, 1.2, 1.5, 3, 0],
  [1.9, 3, 0, 3.9],
  (a, b) => Math.round(a) === Math.round(b)
);
// [1, 1.2, 1.5, 3, 0, 3.9]
```

title: uniqueElements

Finds all unique values in an array.

- Create a `new Set()` from the given array to discard duplicated values.
- Use the spread operator (`...`) to convert it back to an array.

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

```
uniqueElements([1, 2, 2, 3, 4, 4, 5]); // [1, 2, 3, 4, 5]
```

title: uniqueElementsBy

Finds all unique values of an array, based on a provided comparator function.

- Use `Array.prototype.reduce()` and `Array.prototype.some()` to create an array containing only the first unique occurrence of each value, based on the comparator function, `fn` .
- The comparator function takes two arguments: the values of the two elements being compared.

```js
const uniqueElementsBy = (arr, fn) =>
  arr.reduce((acc, v) => {
    if (!acc.some(x => fn(v, x))) acc.push(v);
    return acc;
  }, []);
```

```js
uniqueElementsBy(
  [
    { id: 0, value: 'a' },
    { id: 1, value: 'b' },
    { id: 2, value: 'c' },
    { id: 1, value: 'd' },
    { id: 0, value: 'e' }
  ],
  (a, b) => a.id == b.id
); // [ { id: 0, value: 'a' }, { id: 1, value: 'b' }, { id: 2, value: 'c' } ]
```

title: uniqueElementsByRight

Finds all unique values of an array, based on a provided comparator function, starting from the right.

- Use `Array.prototype.reduceRight()` and `Array.prototype.some()` to create an array containing only the last unique occurrence of each value, based on the comparator function, `fn` .
- The comparator function takes two arguments: the values of the two elements being compared.

```js
const uniqueElementsByRight = (arr, fn) =>
  arr.reduceRight((acc, v) => {
    if (!acc.some(x => fn(v, x))) acc.push(v);
    return acc;
  }, []);
```

```
uniqueElementsByRight(
  [
    { id: 0, value: 'a' },
    { id: 1, value: 'b' },
    { id: 2, value: 'c' },
    { id: 1, value: 'd' },
    { id: 0, value: 'e' }
  ],
  (a, b) => a.id == b.id
); // [ { id: 0, value: 'e' }, { id: 1, value: 'd' }, { id: 2, value: 'c' } ]
```

title: uniqueSymmetricDifference

Returns the unique symmetric difference between two arrays, not containing duplicate values from either array.

- Use `Array.prototype.filter()` and `Array.prototype.includes()` on each array to remove values contained in the other.
- Create a `new Set()` from the results, removing duplicate values.

```
const uniqueSymmetricDifference = (a, b) => [
  ...new Set([
    ...a.filter(v => !b.includes(v)),
    ...b.filter(v => !a.includes(v)),
  ]),
];
```

```
uniqueSymmetricDifference([1, 2, 3], [1, 2, 4]); // [3, 4]
uniqueSymmetricDifference([1, 2, 2], [1, 3, 1]); // [2, 3]
```

title: untildify

Converts a tilde path to an absolute path.

- Use `String.prototype.replace()` with a regular expression and `os.homedir()` to replace the `~` in the start of the path with the home directory.

```
const untildify = str =>
  str.replace(/^~($|\/|\\)/, `${require('os').homedir()}$1`);


untildify('~/node'); // '/Users/aUser/node'
```

title: unzip

Creates an array of arrays, ungrouping the elements in an array produced by zip.

- Use `Math.max()` , `Function.prototype.apply()` to get the longest subarray in the array, `Array.prototype.map()` to make each element an array.
- Use `Array.prototype.reduce()` and `Array.prototype.forEach()` to map grouped values to individual arrays.

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


unzip([['a', 1, true], ['b', 2, false]]); // [['a', 'b'], [1, 2], [true, false]]
unzip([['a', 1, true], ['b', 2]]); // [['a', 'b'], [1, 2], [true]]
```

title: unzipWith

Creates an array of elements, ungrouping the elements in an array produced by zip and applying the provided function.

- Use `Math.max()` , `Function.prototype.apply()` to get the longest subarray in the array, `Array.prototype.map()` to make each element an array.
- Use `Array.prototype.reduce()` and `Array.prototype.forEach()` to map grouped values to individual arrays.
- Use `Array.prototype.map()` and the spread operator (`...`) to apply `fn` to each individual group of elements.

```
const unzipWith = (arr, fn) =>
  arr
    .reduce(
      (acc, val) => (val.forEach((v, i) => acc[i].push(v)), acc),
      Array.from({
        length: Math.max(...arr.map(x => x.length))
      }).map(x => [])
    )
    .map(val => fn(...val));


unzipWith(
  [
    [1, 10, 100],
    [2, 20, 200],
  ],
  (...args) => args.reduce((acc, v) => acc + v, 0)
);
// [3, 30, 300]
```

title: validateNumber

Checks if the given value is a number.

- Use `parseFloat()` to try to convert `n` to a number.
- Use `!Number.isNaN()` to check if `num` is a number.
- Use `Number.isFinite()` to check if `num` is finite.
- Use `Number()` and the loose equality operator (`==`) to check if the coercion holds.

```
const validateNumber = n => {
  const num = parseFloat(n);
  return !Number.isNaN(num) && Number.isFinite(num) && Number(n) == n;
}


validateNumber('10'); // true
validateNumber('a'); // false
```

title: vectorAngle

Calculates the angle (theta) between two vectors.

- Use `Array.prototype.reduce()` , `Math.pow()` and `Math.sqrt()` to calculate the magnitude of each vector and the scalar product of the two vectors.
- Use `Math.acos()` to calculate the arccosine and get the theta value.

```
const vectorAngle = (x, y) => {
  let mX = Math.sqrt(x.reduce((acc, n) => acc + Math.pow(n, 2), 0));
  let mY = Math.sqrt(y.reduce((acc, n) => acc + Math.pow(n, 2), 0));
  return Math.acos(x.reduce((acc, n, i) => acc + n * y[i], 0) / (mX * mY));
};
```

```
vectorAngle([3, 4], [4, 3]); // 0.283794109208328
```

title: vectorDistance

Calculates the distance between two vectors.

- Use `Array.prototype.reduce()` , `Math.pow()` and `Math.sqrt()` to calculate the Euclidean distance between two vectors.

```
const vectorDistance = (x, y) =>
  Math.sqrt(x.reduce((acc, val, i) => acc + Math.pow(val - y[i], 2), 0));
```

```
vectorDistance([10, 0, 5], [20, 0, 10]); // 11.180339887498949
```

title: walkThrough

Creates a generator, that walks through all the keys of a given object.

- Use recursion.
- Define a generator function, `walk` , that takes an object and an array of keys.
- Use a `for...of` loop and `Object.keys()` to iterate over the keys of the object.
- Use `typeof` to check if each value in the given object is itself an object.
- If so, use the `yield*` expression to recursively delegate to the same generator function, `walk` , appending the current `key` to the array of keys. Otherwise, `yield` an array of keys representing

  the current path and the value of the given `key`.
- Use the `yield*` expression to delegate to the `walk` generator function.

```
const walkThrough = function* (obj) {
  const walk = function* (x, previous = []) {
    for (let key of Object.keys(x)) {
      if (typeof x[key] === 'object') yield* walk(x[key], [...previous, key]);
      else yield [[...previous, key], x[key]];
    }
  };
  yield* walk(obj);
};


const obj = {
  a: 10,
  b: 20,
  c: {
    d: 10,
    e: 20,
    f: [30, 40]
  },
  g: [
    {
      h: 10,
      i: 20
    },
    {
      j: 30
    },
    40
  ]
};
[...walkThrough(obj)];
/*
[
  [['a'], 10],
  [['b'], 20],
  [['c', 'd'], 10],
  [['c', 'e'], 20],
  [['c', 'f', '0'], 30],
  [['c', 'f', '1'], 40],
  [['g', '0', 'h'], 10],
  [['g', '0', 'i'], 20],
  [['g', '1', 'j'], 30],
  [['g', '2'], 40]
]
*/
```

title: weekOfYear

Returns the zero-indexed week of the year that a date corresponds to.

- Use `new Date()` and `Date.prototype.getFullYear()` to get the first day of the year as a `Date` object.
- Use `Date.prototype.setDate()` , `Date.prototype.getDate()` and `Date.prototype.getDay()` along with the modulo (`%`) operator to get the first Monday of the year.
- Subtract the first Monday of the year from the given `date` and divide with the number of milliseconds in a week.
- Use `Math.round()` to get the zero-indexed week of the year corresponding to the given `date` .
- `-0` is returned if the given `date` is before the first Monday of the year.

```
const weekOfYear = date => {
  const startOfYear = new Date(date.getFullYear(), 0, 1);
  startOfYear.setDate(startOfYear.getDate() + (startOfYear.getDay() % 7));
  return Math.round((date - startOfYear) / (7 * 24 * 3600 * 1000));
};
```

```
weekOfYear(new Date('2021-06-18')); // 23
```

title: weightedAverage

Calculates the weighted average of two or more numbers.

- Use `Array.prototype.reduce()` to create the weighted sum of the values and the sum of the weights.
- Divide them with each other to get the weighted average.

```js
const weightedAverage = (nums, weights) => {
  const [sum, weightSum] = weights.reduce(
    (acc, w, i) => {
      acc[0] = acc[0] + nums[i] * w;
      acc[1] = acc[1] + w;
      return acc;
    },
    [0, 0]
  );
  return sum / weightSum;
};
```

```js
weightedAverage([1, 2, 3], [0.6, 0.2, 0.3]); // 1.72727
```

title: weightedSample

Gets a random element from an array, using the provided `weights` as the probabilities for each element.

- Use `Array.prototype.reduce()` to create an array of partial sums for each value in `weights` .
- Use `Math.random()` to generate a random number and `Array.prototype.findIndex()` to find the correct index based on the array previously produced.
- Finally, return the element of `arr` with the produced index.

```js
const weightedSample = (arr, weights) => {
  let roll = Math.random();
  return arr[
    weights
      .reduce(
        (acc, w, i) => (i === 0 ? [w] : [...acc, acc[acc.length - 1] + w]),
        []
      )
      .findIndex((v, i, s) => roll >= (i === 0 ? 0 : s[i - 1]) && roll < v)
  ];
};
```

```js
weightedSample([3, 7, 9, 11], [0.1, 0.2, 0.6, 0.1]); // 9
```

title: when

Returns a function that takes one argument and runs a callback if it's truthy or returns it if falsy.

- Return a function expecting a single value, `x`, that returns the appropriate value based on `pred`.

```
const when = (pred, whenTrue) => x => (pred(x) ? whenTrue(x) : x);
```

```
const doubleEvenNumbers = when(x => x % 2 === 0, x => x * 2);
doubleEvenNumbers(2); // 4
doubleEvenNumbers(1); // 1
```

title: without

Filters out the elements of an array that have one of the specified values.

- Use `Array.prototype.includes()` to find values to exclude.
- Use `Array.prototype.filter()` to create an array excluding them.

```
const without = (arr, ...args) => arr.filter(v => !args.includes(v));
```

```
without([2, 1, 2, 3], 1, 2); // [3]
```

title: wordWrap

Wraps a string to a given number of characters using a string break character.

- Use `String.prototype.replace()` and a regular expression to insert a given break character at the nearest whitespace of `max` characters.
- Omit the third argument, `br`, to use the default value of `'\n'`.

```
const wordWrap = (str, max, br = '\n') => str.replace(
  new RegExp(`(?![^\\n]{1,${max}}$)([^\\n]{1,${max}})\\s`, 'g'), '$1' + br
);
```

```
wordWrap(
  'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce tempus.',
  32
);
// 'Lorem ipsum dolor sit amet,\nconsectetur adipiscing elit.\nFusce tempus.'
wordWrap(
  'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce tempus.',
  32,
  '\r\n'
);
// 'Lorem ipsum dolor sit amet,\r\nconsectetur adipiscing elit.\r\nFusce tempus.'
```

title: words

Converts a given string into an array of words.

- Use `String.prototype.split()` with a supplied `pattern` (defaults to non-alpha as a regexp) to convert to an array of strings.
- Use `Array.prototype.filter()` to remove any empty strings.
- Omit the second argument, `pattern` , to use the default regexp.

```
const words = (str, pattern = /[^a-zA-Z-]+/) =>
  str.split(pattern).filter(Boolean);
```

```
words('I love javaScript!!'); // ['I', 'love', 'javaScript']
words('python, javaScript & coffee'); // ['python', 'javaScript', 'coffee']
```

title: xProd

Creates a new array out of the two supplied by creating each possible pair from the arrays.

- Use `Array.prototype.reduce()` , `Array.prototype.map()` and `Array.prototype.concat()` to produce every possible pair from the elements of the two arrays.

```
const xProd = (a, b) =>
  a.reduce((acc, x) => acc.concat(b.map(y => [x, y])), []);
```

```
xProd([1, 2], ['a', 'b']); // [[1, 'a'], [1, 'b'], [2, 'a'], [2, 'b']]
```

title: xor
unlisted: true

Checks if only one of the arguments is `true` .

- Use the logical or (`||`), and (`&&`) and not (`!`) operators on the two given values to create the logical xor.

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

```
xor(true, true); // false
xor(true, false); // true
xor(false, true); // true
xor(false, false); // false
```

title: yesNo
unlisted: true

Returns `true` if the string is `y` / `yes` or `false` if the string is `n` / `no` .

- Use `RegExp.prototype.test()` to check if the string evaluates to `y/yes` or `n/no` .
- Omit the second argument, `def` to set the default answer as `no` .

```
const yesNo = (val, def = false) =>
  /^(y|yes)$/i.test(val) ? true : /^(n|no)$/i.test(val) ? false : def;
```

```
yesNo('Y'); // true
yesNo('yes'); // true
yesNo('No'); // false
yesNo('Foo', true); // true
```

title: yesterday

Results in a string representation of yesterday's date.

- Use `new Date()` to get the current date.
- Decrement it by one using `Date.prototype.getDate()` and set the value to the result using `Date.prototype.setDate()` .
- Use `Date.prototype.toISOString()` to return a string in `yyyy-mm-dd` format.

```
const yesterday = () => {
  let d = new Date();
  d.setDate(d.getDate() - 1);
  return d.toISOString().split('T')[0];
};
```

```
yesterday(); // 2018-10-17 (if current date is 2018-10-18)
```

title: zip

Creates an array of elements, grouped based on their position in the original arrays.

- Use `Math.max()` , `Function.prototype.apply()` to get the longest array in the arguments.
- Create an array with that length as return value and use `Array.from()` with a mapping function to create an array of grouped elements.
- If lengths of the argument arrays vary, `undefined` is used where no value could be found.

```
const zip = (...arrays) => {
  const maxLength = Math.max(...arrays.map(x => x.length));
  return Array.from({ length: maxLength }).map((_, i) => {
    return Array.from({ length: arrays.length }, (_, k) => arrays[k][i]);
  });
};
```

```
zip(['a', 'b'], [1, 2], [true, false]); // [['a', 1, true], ['b', 2, false]]
zip(['a'], [1, 2], [true, false]); // [['a', 1, true], [undefined, 2, false]]
```

title: zipObject

Associates properties to values, given array of valid property identifiers and an array of values.

- Use `Array.prototype.reduce()` to build an object from the two arrays.
- If the length of `props` is longer than `values` , remaining keys will be `undefined` .
- If the length of `values` is longer than `props` , remaining values will be ignored.

```
const zipObject = (props, values) =>
  props.reduce((obj, prop, index) => ((obj[prop] = values[index]), obj), {});
```

```
zipObject(['a', 'b', 'c'], [1, 2]); // {a: 1, b: 2, c: undefined}
zipObject(['a', 'b'], [1, 2, 3]); // {a: 1, b: 2}
```

title: zipWith

Creates an array of elements, grouped based on the position in the original arrays and using a function to specify how grouped values should be combined.

- Check if the last argument provided is a function.
- Use `Math.max()` to get the longest array in the arguments.
- Use `Array.from()` to create an array with appropriate length and a mapping function to create array of grouped elements.
- If lengths of the argument arrays vary, `undefined` is used where no value could be found.
- The function is invoked with the elements of each group.

```
const zipWith = (...array) => {
  const fn =
    typeof array[array.length - 1] === 'function' ? array.pop() : undefined;
  return Array.from({ length: Math.max(...array.map(a => a.length)) }, (_, i) =>
    fn ? fn(...array.map(a => a[i])) : array.map(a => a[i])
  );
};
```

```
zipWith([1, 2], [10, 20], [100, 200], (a, b, c) => a + b + c); // [111, 222]
zipWith(
  [1, 2, 3],
  [10, 20],
  [100, 200],
  (a, b, c) =>
    (a != null ? a : 'a') + (b != null ? b : 'b') + (c != null ? c : 'c')
); // [111, 222, '3bc']
```