Basic JavaScript Code Snippets

**1. all**

This snippet returns true if the predicate function returns true for all elements in a collection and false otherwise. You can omit the second argument fnif you want to use Booleanas a default value.

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

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

all([1, 2, 3]); // true

**2. allEqual**

This snippet checks whether all elements of the array are equal.

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

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

allEqual([1, 1, 1, 1]); // true

**3. approximatelyEqual**

This snippet checks whether two numbers are approximately equal to each other, with a small difference.

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

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

**4. arrayToCSV**

This snippet converts the elements to strings with comma-separated values.

const arrayToCSV = (arr, delimiter = ',') =>

arr.map(v => v.map(x => `"${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"'```

**5. arrayToHtmlList**

This snippet converts the elements of an array into <li> tags and appends them to the list of the given ID.

const arrayToHtmlList = (arr, listID) =>

(el => (

(el = document.querySelector('#' + listID)),

(el.innerHTML += arr.map(item => `<li>${item}</li>`).join(''))

))();

arrayToHtmlList(['item 1', 'item 2'], 'myListID');

**6. attempt**

This snippet executes a function, returning either the result or the caught error object.

const attempt = (fn, ...args) => {

try {

return fn(...args);

} catch (e) {

return e instanceof Error ? e : new Error(e);

}

};

var elements = attempt(function(selector) {

return document.querySelectorAll(selector);

}, '>\_>');

if (elements instanceof Error) elements = []; // elements = []

**7. average**

This snippet returns the average of two or more numerical values.

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

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

average(1, 2, 3); // 2

**8. averageBy**

This snippet returns the average of an array after initially doing the mapping of each element to a value using a given function.

const averageBy = (arr, fn) =>

arr.map(typeof fn === 'function' ? fn : val => val[fn]).reduce((acc, val) => acc + val, 0) /

arr.length;

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

averageBy([{ n: 4 }, { n: 2 }, { n: 8 }, { n: 6 }], 'n'); // 5

**9. bifurcate**

This snippet splits values into two groups and then puts a truthy element of filterin the first group, and in the second group otherwise.

You can use Array.prototype.reduce()and Array.prototype.push()to add elements to groups based on filter.

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'] ]

**10. bifurcateBy**

This snippet splits values into two groups, based on a predicate function. If the predicate function returns a truthy value, the element will be placed in the first group. Otherwise, it will be placed in the second group.

You can use Array.prototype.reduce()and Array.prototype.push()to add elements to groups, based on the value returned by fnfor each element.

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'] ]

**11. bottomVisible**

This snippet checks whether the bottom of a page is visible.

const bottomVisible = () =>

document.documentElement.clientHeight + window.scrollY >=

(document.documentElement.scrollHeight || document.documentElement.clientHeight);

bottomVisible(); // true

**12. byteSize**

This snippet returns the length of a string in bytes.

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

byteSize('😀'); // 4

byteSize('Hello World'); // 11

**13. capitalize**

This snippet capitalizes the first letter of a string.

const capitalize = ([first, ...rest]) =>

first.toUpperCase() + rest.join('');

capitalize('fooBar'); // 'FooBar'

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

**14. capitalizeEveryWord**

This snippet capitalizes the first letter of every word in a given string.

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

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

**15. castArray**

This snippet converts a non-array value into array.

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

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

castArray([1]); // [1]

**16. compact**

This snippet removes false values from an array.

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

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

// [ 1, 2, 3, 'a', 's', 34 ]

**17. countOccurrences**

This snippet counts the occurrences of a value in an array.

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

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

**18. Create Directory**

This snippet uses existsSync() to check whether a directory exists and then mkdirSync() to create it if it doesn’t.

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

**19. currentURL**

This snippet returns the current URL.

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

currentURL(); // 'https://medium.com/@fatosmorina'

**20. dayOfYear**

This snippet gets the day of the year from a Dateobject.

const dayOfYear = date =>

Math.floor((date - new Date(date.getFullYear(), 0, 0)) / 1000 / 60 / 60 / 24);

dayOfYear(new Date()); // 272

**21. decapitalize**

This snippet turns the first letter of a string into lowercase.

const decapitalize = ([first, ...rest]) =>

first.toLowerCase() + rest.join('')

decapitalize('FooBar'); // 'fooBar'

decapitalize('FooBar'); // 'fooBar'

**22. deepFlatten**

This snippet flattens an array recursively.

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

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

**23. default**

This snippet assigns default values for all properties in an object that are undefined.

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

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

**24. defer**

This snippet delays the execution of a function until the current call stack is cleared.

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

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

**25. degreesToRads**

This code snippet can be used to convert a value from degrees to radians.

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

degreesToRads(90.0); // ~1.5708

**26. difference**

This snippet finds the difference between two arrays.

const difference = (a, b) => {

const s = new Set(b);

return a.filter(x => !s.has(x));

};

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

**27. differenceBy**

This method returns the difference between two arrays, after applying a given function to each element of both lists.

const differenceBy = (a, b, fn) => {

const s = new Set(b.map(fn));

return a.filter(x => !s.has(fn(x)));

};

differenceBy([2.1, 1.2], [2.3, 3.4], Math.floor); // [1.2]

differenceBy([{ x: 2 }, { x: 1 }], [{ x: 1 }], v => v.x); // [ { x: 2 } ]

**28. differenceWith**

This snippet removes the values for which the comparator function returns false.

const differenceWith = (arr, val, comp) => 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]

**29. digitize**

This snippet gets a number as input and returns an array of its digits.

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

digitize(431); // [4, 3, 1]

**30. distance**

This snippet returns the distance between two points by calculating the Euclidean distance.

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

distance(1, 1, 2, 3); // 2.23606797749979

**31. Drop Elements**

This snippet returns a new array with n elements removed from the left.

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); // []

**32. dropRight**

This snippet returns a new array with n elements removed from the right.

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); // []

**33. dropRightWhile**

This snippet removes elements from the right side of an array until the passed function returns true.

const dropRightWhile = (arr, func) => {

while (arr.length > 0 && !func(arr[arr.length - 1])) arr = arr.slice(0, -1);

return arr;

};

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

**34. dropWhile**

This snippet removes elements from an array until the passed function returns true.

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]

**35. elementContains**

This snippet checks whether the parent element contains the child.

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

**36. Filter Duplicate Elements**

This snippet removes duplicate values in an array.

const filterNonUnique = arr => arr.filter(i => arr.indexOf(i) === arr.lastIndexOf(i));

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

**37. findKey**

This snippet returns the first key that satisfies a given function.

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 }

},

o => o['active']

); // 'barney'

**38. findLast**

This snippet returns the last element for which a given function returns a truthy value.

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

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

**39. flatten**

This snippet flattens an array up to a specified depth using recursion.

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]

**40. forEachRight**

This snippet executes a function for each element of an array starting from the array’s last element.

const forEachRight = (arr, callback) =>

arr

.slice(0)

.reverse()

.forEach(callback);

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

**41. forOwn**

This snippet iterates on each property of an object and iterates a callback for each one respectively.

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

**42. functionName**

This snippet prints the name of a function into the console.

const functionName = fn => (console.debug(fn.name), fn);

functionName(Math.max); // max (logged in debug channel of console)

**43. Get Time From Date**

This snippet can be used to get the time from a Dateobject as a string.

const getColonTimeFromDate = date => date.toTimeString().slice(0, 8);

getColonTimeFromDate(new Date()); // "08:38:00"

**44. Get Days Between Dates**

This snippet can be used to find the difference in days between two dates.

const getDaysDiffBetweenDates = (dateInitial, dateFinal) =>

(dateFinal - dateInitial) / (1000 \* 3600 \* 24);

getDaysDiffBetweenDates(new Date('2019-01-13'), new Date('2019-01-15')); // 2

**45. getStyle**

This snippet can be used to get the value of a CSS rule for a particular element.

const getStyle = (el, ruleName) => getComputedStyle(el)[ruleName];

getStyle(document.querySelector('p'), 'font-size'); // '16px'

**46. getType**

This snippet can be used to get the type of a value.

const getType = v =>

v === undefined ? 'undefined' : v === null ? 'null' : v.constructor.name.toLowerCase();

getType(new Set([1, 2, 3])); // 'set'

**47. hasClass**

This snippet checks whether an element has a particular class.

const hasClass = (el, className) => el.classList.contains(className);

hasClass(document.querySelector('p.special'), 'special'); // true

**48. head**

This snippet returns the head of a list.

const head = arr => arr[0];

head([1, 2, 3]); // 1

**49. hide**

This snippet can be used to hide all elements specified.

const hide = (...el) => [...el].forEach(e => (e.style.display = 'none'));

hide(document.querySelectorAll('img')); // Hides all <img> elements on the page

**50. httpsRedirect**

This snippet can be used to redirect from HTTP to HTTPS in a particular domain.

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>

**51. indexOfAll**

This snippet can be used to get all indexes of a value in an array, which returns an empty array, in case this value is not included in it.

const indexOfAll = (arr, val) => arr.reduce((acc, el, i) => (el === val ? [...acc, i] : acc), []);

indexOfAll([1, 2, 3, 1, 2, 3], 1); // [0,3]

indexOfAll([1, 2, 3], 4); // []

**52. initial**

This snippet returns all elements of an array except the last one.

const initial = arr => arr.slice(0, -1);

initial([1, 2, 3]); // [1,2]const initial = arr => arr.slice(0, -1);

initial([1, 2, 3]); // [1,2]

**53. insertAfter**

This snippet can be used to insert an HTML string after the end of a particular element.

const insertAfter = (el, htmlString) => el.insertAdjacentHTML('afterend', htmlString);

insertAfter(document.getElementById('myId'), '<p>after</p>'); // <div id="myId">...</div> <p>after</p>

**54. insertBefore**

This snippet can be used to insert an HTML string before a particular element.

const insertBefore = (el, htmlString) => el.insertAdjacentHTML('beforebegin', htmlString);

insertBefore(document.getElementById('myId'), '<p>before</p>'); // <p>before</p> <div id="myId">...</div>

**55. intersection**

This snippet can be used to get an array with elements that are included in two other arrays.

const intersection = (a, b) => {

const s = new Set(b);

return a.filter(x => s.has(x));

};

intersection([1, 2, 3], [4, 3, 2]); // [2, 3]

**56. intersectionBy**

This snippet can be used to return a list of elements that exist in both arrays, after a particular function has been executed to each element of both arrays.

const intersectionBy = (a, b, fn) => {

const s = new Set(b.map(fn));

return a.filter(x => s.has(fn(x)));

};

intersectionBy([2.1, 1.2], [2.3, 3.4], Math.floor); // [2.1]

**57. intersectionWith**

This snippet can be used to return a list of elements that exist in both arrays by using a comparator function.

const intersectionWith = (a, b, comp) => a.filter(x => b.findIndex(y => comp(x, y)) !== -1);

intersectionWith([1, 1.2, 1.5, 3, 0], [1.9, 3, 0, 3.9], (a, b) => Math.round(a) === Math.round(b));

// [1.5, 3, 0]

**58. is**

This snippet can be used to check if a value is of a particular type.

const is = (type, val) => ![, null].includes(val) && val.constructor === type;

is(Array, [1]); // true

is(ArrayBuffer, new ArrayBuffer()); // true

is(Map, new Map()); // true

is(RegExp, /./g); // true

is(Set, new Set()); // true

is(WeakMap, new WeakMap()); // true

is(WeakSet, new WeakSet()); // true

is(String, ''); // true

is(String, new String('')); // true

is(Number, 1); // true

is(Number, new Number(1)); // true

is(Boolean, true); // true

is(Boolean, new Boolean(true)); // true

**59. isAfterDate**

This snippet can be used to check whether a date is after another date.

const isAfterDate = (dateA, dateB) => dateA > dateB;

isAfterDate(new Date(2010, 10, 21), new Date(2010, 10, 20));

// true

**60. isAnagram**

This snippet can be used to check whether a particular string is an anagram with another string.

const isAnagram = (str1, str2) => {

const normalize = str =>

str

.toLowerCase()

.replace(/[^a-z0-9]/gi, '')

.split('')

.sort()

.join('');

return normalize(str1) === normalize(str2);

};

isAnagram('iceman', 'cinema'); // true

61. isArrayLike

This snippet can be used to check if a provided argument is iterable like an array.

const isArrayLike = obj => obj != null && typeof obj[Symbol.iterator] === 'function';

isArrayLike(document.querySelectorAll('.className')); // true

isArrayLike('abc'); // true

isArrayLike(null); // false

**62. isBeforeDate**

This snippet can be used to check whether a date is before another date.

const isBeforeDate = (dateA, dateB) => dateA < dateB;

isBeforeDate(new Date(2010, 10, 20), new Date(2010, 10, 21)); // true

**63. isBoolean**

This snippet can be used to check whether an argument is a boolean.

const isBoolean = val => typeof val === 'boolean';

isBoolean(null); // false

isBoolean(false); // true

64. isBrowser

This snippet can be used to determine whether the current runtime environment is a browser. This is helpful for avoiding errors when running front-end modules on the server (Node).

const isBrowser = () => ![typeof window, typeof document].includes('undefined');

isBrowser(); // true (browser)

isBrowser(); // false (Node)

**65. isBrowserTabFocused**

This snippet can be used to determine whether the browser tab is focused.

const isBrowserTabFocused = () => !document.hidden;

isBrowserTabFocused(); // true

**66. isLowerCase**

This snippet can be used to determine whether a string is lower case.

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

isLowerCase('abc'); // true

isLowerCase('a3@$'); // true

isLowerCase('Ab4'); // false

67. isNil

This snippet can be used to check whether a value is null or undefined.

const isNil = val => val === undefined || val === null;

isNil(null); // true

isNil(undefined); // true

**68. isNull**

This snippet can be used to check whether a value is null.

const isNull = val => val === null;

isNull(null); // true

**69. isNumber**

This snippet can be used to check whether a provided value is a number.

const isNumber = val => typeof val === 'number';

isNumber('1'); // false

isNumber(1); // true

**70. isObject**

This snippet can be used to check whether a provided value is an object. It uses the Object constructor to create an object wrapper for the given value.

If it is already an object, then an object type that corresponds to the given value will be returned. Otherwise, a new object will be returned.

const isObject = obj => obj === Object(obj);

isObject([1, 2, 3, 4]); // true

isObject([]); // true

isObject(['Hello!']); // true

isObject({ a: 1 }); // true

isObject({}); // true

isObject(true); // false

**71. isObjectLike**

This snippet can be used to check if a value is not null and that its typeof is “object”.

const isObjectLike = val => val !== null && typeof val === 'object';

isObjectLike({}); // true

isObjectLike([1, 2, 3]); // true

isObjectLike(x => x); // false

isObjectLike(null); // false

**72. isPlainObject**

This snippet checks whether a value is an object created by the Object constructor.

const isPlainObject = val => !!val && typeof val === 'object' && val.constructor === Object;

isPlainObject({ a: 1 }); // true

isPlainObject(new Map()); // false

**73. isPromiseLike**

This snippet checks whether an object looks like a Promise.

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

**74. isSameDate**

This snippet can be used to check whether two dates are equal.

const isSameDate = (dateA, dateB) => dateA.toISOString() === dateB.toISOString();

isSameDate(new Date(2010, 10, 20), new Date(2010, 10, 20)); // true

**75. isString**

This snippet can be used to check whether an argument is a string.

const isString = val => typeof val === 'string';

isString('10'); // true

**76. isSymbol**

This snippet can be used to check whether an argument is a symbol.

const isSymbol = val => typeof val === 'symbol';

isSymbol(Symbol('x')); // true

**77. isUndefined**

This snippet can be used to check whether a value is undefined.

const isUndefined = val => val === undefined;

isUndefined(undefined); // true

**78. isUpperCase**

This snippet can be used to check whether a string is upper case.

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

isUpperCase('ABC'); // true

isLowerCase('A3@$'); // true

isLowerCase('aB4'); // false

**79. isValidJSON**

This snippet can be used to check whether a 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

**80. last**

This snippet returns the last element of an array.

const last = arr => arr[arr.length - 1];

last([1, 2, 3]); // 3

**81. matches**

This snippet compares two objects to determine if the first one contains the same property values as the second one.

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

**82. maxDate**

This snippet can be used to get the latest date.

const maxDate = (...dates) => new Date(Math.max.apply(null, ...dates));

const array = [

new Date(2017, 4, 13),

new Date(2018, 2, 12),

new Date(2016, 0, 10),

new Date(2016, 0, 9)

];

maxDate(array); // 2018-03-11T22:00:00.000Z

**83. maxN**

This snippet returns the n largest elements from a list. If nis greater than or equal to the list’s length, then it will return the original list (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]

**84. minDate**

This snippet can be used to get the earliest date.

const minDate = (...dates) => new Date(Math.min.apply(null, ...dates));

const array = [

new Date(2017, 4, 13),

new Date(2018, 2, 12),

new Date(2016, 0, 10),

new Date(2016, 0, 9)

];

minDate(array); // 2016-01-08T22:00:00.000Z

**85. minN**

This snippet returns the n smallest elements from a list. If nis greater than or equal to the list’s length, then it will return the original list (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]

**86. negate**

This snippet can be used to apply the not operator (!) to a predicate function with its arguments.

const negate = func => (...args) => !func(...args);

[1, 2, 3, 4, 5, 6].filter(negate(n => n % 2 === 0)); // [ 1, 3, 5 ]

**87. nodeListToArray**

This snippet can be used to convert a nodeList to an array.

const nodeListToArray = nodeList => [...nodeList];

nodeListToArray(document.childNodes); // [ <!DOCTYPE html>, html ]

**88. pad**

This snippet can be used to pad a string on both sides with a specified character if it is shorter than the specified length.

const pad = (str, length, char = ' ') =>

str.padStart((str.length + length) / 2, char).padEnd(length, char);

pad('cat', 8); // ' cat '

pad(String(42), 6, '0'); // '004200'

pad('foobar', 3); // 'foobar'

89. radsToDegrees

This snippet can be used to convert an angle from radians to degrees.

const radsToDegrees = rad => (rad \* 180.0) / Math.PI;

radsToDegrees(Math.PI / 2); // 90

**90. Random Hexadecimal Color Code**

This snippet can be used to generate a random hexadecimal color code.

const randomHexColorCode = () => {

let n = (Math.random() \* 0xfffff \* 1000000).toString(16);

return '#' + n.slice(0, 6);

};

randomHexColorCode(); // "#e34155"

**91. randomIntArrayInRange**

This snippet can be used to generate an array with n random integers in a specified range.

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 ]

**92. randomIntegerInRange**

This snippet can be used to generate a random integer in a specified range.

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

randomIntegerInRange(0, 5); // 3

**93. randomNumberInRange**

This snippet can be used to return a random number in a specified range.

const randomNumberInRange = (min, max) => Math.random() \* (max - min) + min;

randomNumberInRange(2, 10); // 6.0211363285087005

**94. readFileLines**

This snippet can be used to read a file by getting an array of lines from a file.

const fs = require('fs');

const readFileLines = filename =>

fs

.readFileSync(filename)

.toString('UTF8')

.split('\n');

let arr = readFileLines('test.txt');

console.log(arr); // ['line1', 'line2', 'line3']

**95. Redirect to a URL**

This snippet can be used to do a redirect to a specified URL.

const redirect = (url, asLink = true) =>

asLink ? (window.location.href = url) : window.location.replace(url);

redirect('https://google.com');

**96. reverse**

This snippet can be used to reverse a string.

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

reverseString('foobar'); // 'raboof'

**97. round**

This snippet can be used to round a number to a specified number of digits.

const round = (n, decimals = 0) => Number(`${Math.round(`${n}e${decimals}`)}e-${decimals}`);

round(1.005, 2); // 1.01

**98. runPromisesInSeries**

This snippet can be used to run an array of promises in series.

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

**99. sample**

This snippet can be used to get a random number from an array.

const sample = arr => arr[Math.floor(Math.random() \* arr.length)];

sample([3, 7, 9, 11]); // 9

**100. sampleSize**

This snippet can be used to get nrandom elements from unique positions from an array up to the size of the array. Elements in the array are shuffled using the Fisher-Yates algorithm.

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]

**101. scrollToTop**

This snippet can be used to do a smooth scroll to the top of the current page.

const scrollToTop = () => {

const c = document.documentElement.scrollTop || document.body.scrollTop;

if (c > 0) {

window.requestAnimationFrame(scrollToTop);

window.scrollTo(0, c - c / 8);

}

};

scrollToTop();

**102. serializeCookie**

This snippet can be used to serialize a cookie name-value pair into a Set-Cookie header string.

const serializeCookie = (name, val) => `${encodeURIComponent(name)}=${encodeURIComponent(val)}`;

serializeCookie('foo', 'bar'); // 'foo=bar'

**103. setStyle**

This snippet can be used to set the value of a CSS rule for a particular element.

const setStyle = (el, ruleName, val) => (el.style[ruleName] = val);

setStyle(document.querySelector('p'), 'font-size', '20px');

// The first <p> element on the page will have a font-size of 20px

**104. shallowClone**

This snippet can be used to create a shallow clone of an object.

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

const a = { x: true, y: 1 };

const b = shallowClone(a); // a !== b

**105. show**

This snippet can be used to show all the elements specified.

const show = (...el) => [...el].forEach(e => (e.style.display = ''));

show(...document.querySelectorAll('img')); // Shows all <img> elements on the page

**106. shuffle**

This snippet can be used to order the elements of an array randomly using the Fisher-Yates algorithm.

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]

**107. similarity**

This snippet can be used to return an array of elements that appear in two arrays.

https://medium.com/better-programming/127-helpful-javascript-snippets-you-can-learn-in-30-seconds-or-less-part-6-of-6-862a6403d334

**108. sleep**

This snippet can be used to delay the execution of an asynchronous function by putting it into sleep.

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.');

}

**109. smoothScroll**

This snippet can be used to smoothly scroll the element on which it is called into the visible area of the browser window.

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

**110. sortCharactersInString**

This snippet can be used to alphabetically sort the characters in a string.

const sortCharactersInString = str => [...str].sort((a, b) => a.localeCompare(b)).join('');

sortCharactersInString('cabbage'); // 'aabbceg'

**111. splitLines**

This snippet can be used to split a multi-line string into an array of lines.

const splitLines = str => str.split(/\r?\n/);

splitLines('This\nis a\nmultiline\nstring.\n'); // ['This', 'is a', 'multiline', 'string.' , '']

**112. stripHTMLTags**

This snippet can be used 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'

**113. sum**

This snippet can be used to find the sum of two or more numbers or arrays.

const sum = (...arr) => [...arr].reduce((acc, val) => acc + val, 0);

sum(1, 2, 3, 4); // 10

sum(...[1, 2, 3, 4]); // 10

**114. tail**

This snippet can be used to get an array with all the elements of an array except for the first one. If the array has only one element, then that an array with that element will be returned instead.

const tail = arr => (arr.length > 1 ? arr.slice(1) : arr);

tail([1, 2, 3]); // [2,3]

tail([1]); // [1]

**115. take**

This snippet can be used to get an array with nelements removed from the beginning.

const take = (arr, n = 1) => arr.slice(0, n);

take([1, 2, 3], 5); // [1, 2, 3]

take([1, 2, 3], 0); // []

**116. takeRight**

This snippet can be used to get an array with nelements removed 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]

117. timeTaken

This snippet can be used to find out the time it takes to execute a function.

const timeTaken = callback => {

console.time('timeTaken');

const r = callback();

console.timeEnd('timeTaken');

return r;

};

timeTaken(() => Math.pow(2, 10)); // 1024, (logged): timeTaken: 0.02099609375ms

**118. times**

This snippet can be used to iterate over a callback ntimes.

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

**119. toCurrency**

This snippet can be used to format a number like a currency.

const toCurrency = (n, curr, LanguageFormat = undefined) =>

Intl.NumberFormat(LanguageFormat, { style: 'currency', currency: curr }).format(n);

toCurrency(123456.789, 'EUR'); // €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

**120. toDecimalMark**

This snippet uses the toLocaleString()function to convert float-point arithmetic to the decimal mark form by using a number to make a comma-separated string.

const toDecimalMark = num => num.toLocaleString('en-US');

toDecimalMark(12305030388.9087); // "12,305,030,388.909"

**121. toggleClass**

This snippet can be used to toggle a class for an element.

const toggleClass = (el, className) => el.classList.toggle(className);

toggleClass(document.querySelector('p.special'), 'special');

// The paragraph will not have the 'special' class anymore

**122. tomorrow**

This snippet can be used to get a string representation of tomorrow’s date.

const tomorrow = () => {

let t = new Date();

t.setDate(t.getDate() + 1);

return t.toISOString().split('T')[0];

};

tomorrow(); // 2019-09-08 (if current date is 2018-09-08)

**123. unfold**

This snippet can be used to build an array using an iterator function and an initial seed value.

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]

**124. union**

This snippet can be used to find the union of two arrays, resulting in an array that has elements that come from both arrays but that do not repeat.

const union = (a, b) => Array.from(new Set([...a, ...b]));

union([1, 2, 3], [4, 3, 2]); // [1,2,3,4]

**125. uniqueElements**

This snippet uses ES6 Setandthe…restoperator to get every element only once.

const uniqueElements = arr => [...new Set(arr)];

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

**126. validateNumber**

This snippet can be used to check whether a value is a number.

const validateNumber = n => !isNaN(parseFloat(n)) && isFinite(n) && Number(n) == n;

validateNumber('10'); // true

**127. words**

This snippet converts a string into an array of words.

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"]

**128 Check how long an operation takes**

* The performance.now() method returns a DOMHighResTimeStamp, measured in milliseconds.
* performance.now() is relative to page load and more precise in orders of magnitude.
* Use cases include benchmarking and other cases where a high-resolution time is required
* such as media (gaming, audio, video, //etc.)

var startTime = performance.now();

doSomething();

const endTime = performance.now();

console.log("this doSomething took " + (endTime - startTime) + " milliseconds.");

129 Sorting an Array

This snippet is very useful when you want to sort an array quickly without writing a long sorting function, by using JavaScipt built-in sort method.

const months = ['March', 'Jan', 'Feb', 'Dec'];

const numbers = [5, 7, 1, 0, 13, 10]

months.sort();

numbers.sort();

console.log(months); //["Dec", "Feb", "Jan", "March"]

console.log(numbers); //[0, 1, 10, 13, 5, 7]

130 Count Occurrences

This useful code snippet will count the occurrences of each element of the array. This comes in handy when you are required to know how many times each item is present in the array.

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

console.log(Occurrences([1, 4, 5, 5, 1, 3], 1)) // 2

console.log(Occurrences([2, 5, 5, 6, 7, 6, 6], 6)) // 3

131 Reverse a String

I will show you the easy way to reverse a String in JavaScript using the below snippet code.

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

console.log(ReverseString("JavaScript")) // tpircSavaJ

132 Minimum Number of Array

Every day we had to tackle one problem where we need find the smallest in the big or small array. To do this in an easy and fast way without using any loop, I had a show code example below.

const minimumNum = (arr, n = 1) => [...arr].sort((a, b) => a - b).slice(0, n);

console.log(minimumNum([4, 3, 2, 1, 0])) // 1

console.log(minimumNum([12, 34, 12, 5, 7])) //

133 Largest Number of Array

This snippet is helpful to find the largest number value in the array in a short time without using extra loops.

const LargestNum = (arr, n = 1) => [...arr].sort((a, b) => b - a).slice(0, n);

LargestNum([12, 34, 78]) // 78

LargestNum([1, 4, 5, 3, 2]) // 5

LargestNum([2, 0, 3]) // 3

134 Convert to Array

This snippet will convert different data types in JavaScript to Array format. This comes in handy when you need to convert a string into an array to apply different functions to it.

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

console.log(ConvertToArray("JavaScript")) // [JavaScript]

console.log(ConvertToArray(1000)) // [1000]

console.log(ConvertToArray(true)) // [true]

135 Byte Size

You always need to calculate the size of any string or integer number.

const byteSize1 = str => new Blob([str]).size;

const byteSize2 = int => new Blob([int]).size;

byteSize1("Coding") // 6

byteSize2(1234) // 4

byteSize2('12344') // 5

136 Remove Duplicates in List

This snippet code from the title you will know that we will go to remove duplicates from the list.

const RemoveDupli = arr => [...new Set(arr)];

console.log(RemoveDupli([2, 2, 2, 5, 1, 0, 2])) // [2,5,1,0]

console.log(RemoveDupli([9, 6, 7, 8, 8, 6])) // [9,6,7,8]

137 Error Handling

Sometimes the error we face comes unusually and hard to detect them because they are sudden error. If you are working on an API sometimes you will 404 error response in that case you can use Error Handling statements of JavaScript.

try {

try\_statements

}

catch (exception\_var) {

catch\_statements

}

finally {

finally\_statements

}

138 Merge Arrays

This snippet code will help you to merge two arrays into one array. This is an easy and fast way to do this without using iterating each and placing them in a new array.

const lang1 = ["JavaScript", "Python", "C++"];

const lang2 = ["C#", "Dart", "Java"];

const merge = [...lang1, ...lang2];

console.log(merge); // [""JavaScript", "Python", "C++", "C#", "Dart", "Java"]

139 Empty An Array

This snippet code will helpful when you need to empty an array in a short time. Check the below code to know how to do it.

var array = ["Haider", "Jeff", "Jessica", "Tadashi"];

array.length = 0; //making size zero

console.log(array) // [] array is now empty

140 Shorten Array

Do you know there is an easy way to shorten your array, This snippet code will show you how to do that by an example code.

var items = [4, 5, 7, 8, 2, 1, 3, 6];

items.length = 3

console.log(items) //[4, 5, 7]

141 DataTypes Conversion

This snippet code simply converts the String into integers and integers into String by using built-in JavaScript methods.

// converting string to number

let string2 = "903"

let num = Number(string2)

// converting number to string

let num = 49

let string1 = num.toString();

142 Replace Characters in String

This snippet code is using a regex to replace specific characters in a string in a short time.

var string = "JavaScript";

console.log(string.replace(/Script/, "code")); //Javacode

143 Trim Whitespaces in a String

This snippet code will remove whitespaces from a string using the regex method in JavaScript. This comes in handy when you get raw data from the internet and they had tabs and spaces in it.

var str = " a b c d e f g hi j ";

var newStr = str.replace(/\s+/g, '');

console.log(newStr)

143 Union

Union (a ∪ b): create a set that contains the elements of both set a and set b.

let a = new Set([1,2,3]);

let b = new Set([4,3,2]);

let union = new Set([...a, ...b]);

// {1,2,3,4}

The pattern is always the same:

* Convert one or both sets to arrays.
* Perform the operation.
* Convert the result back to a set.

As explained in [1], the spread operator (...) inserts the elements of something iterable (like a set) into an array. Therefore, [...a, ...b] means that a and b are converted to arrays and concatenated. It is equivalent to [...a].concat([...b]).

Computing the union of two Sets a and b means creating a Set that contains the elements of both a and b.

const a = new Set([1,2,3]);

const b = new Set([4,3,2]);

// Use spreading to concatenate two iterables

const union = new Set([...a, ...b]);

assert.deepEqual(Array.from(union), [1, 2, 3, 4]);

144 Intersection

Intersection (a ∩ b): create a set that contains those elements of set a that are also in set b.

let a = new Set([1,2,3]);

let b = new Set([4,3,2]);

let intersection = new Set(

[...a].filter(x => b.has(x)));

// {2,3}

Steps: Convert a to an array, filter the elements, convert the result to a set.

Computing the intersection of two Sets a and b means creating a Set that contains those elements of a that are also in b.

const a = new Set([1,2,3]);

const b = new Set([4,3,2]);

const intersection = new Set(

Array.from(a).filter(x => b.has(x))

);

assert.deepEqual(

Array.from(intersection), [2, 3]

);

145 Difference

Difference (a \ b): create a set that contains those elements of set a that are not in set b. This operation is also sometimes called minus (-).

let a = new Set([1,2,3]);

let b = new Set([4,3,2]);

let difference = new Set(

[...a].filter(x => !b.has(x)));

// {1}

Difference (a \ b)

Computing the difference between two Sets a and b means creating a Set that contains those elements of a that are not in b. This operation is also sometimes called minus (−).

const a = new Set([1,2,3]);

const b = new Set([4,3,2]);

const difference = new Set(

Array.from(a).filter(x => !b.has(x))

);

assert.deepEqual(

Array.from(difference), [1]

);

146 Join two arrays and remove duplicate elements using Javascript

**(1)** We are using the spread operator and Set() function in the above code example. The spread operator that we are using to merge two or multiple arrays using the below syntax.

[...arr1, ...arr2, ...arr3]

The Set() function is used to remove the duplicates from an array. We can use the below code for that.

[...new Set(Array)]

More methods that we will be using to perform the same task can be found below.

const arr1 = [10, 20, 30, 40, 50];

const arr2 = [40, 50, 60, 70, 80];

// concatenate the above arrays using spread operator

const join\_arr = [...arr1, ...arr2];

console.log(join\_arr);

// remove duplicate from above array using Set() function

const result = [...new Set(join\_arr)];

console.log(result);

//output

//[10,20,30,40,50,40,50,60,70,80]

//[10,20,30,40,50,60,70,80]

(2)

const arr1 = [10,20,30,40,50]

const arr2 = [40, 50, 60, 70, 80];

const result = [...new Set([...arr1, ...arr2])];

console.log(result);

//output

//[10,20,30,40,50,60,70,80]

(3) Use Set() and concat() function to merge and remove duplicates from arrays

The concat() function in Javascript is used to merge two or multiple arrays. We will be using it to join our arrays and then removing the repeating values from the array using the Set() function.

**concat() function syntax**

array1.concat(array2)

If you are using ES6 you can also use concat() function as below

[].concat(...[array1, array2, array3])

const arr1 = [10, 20, 30, 40, 50];

const arr2 = [40, 50, 60, 70, 80];

const result = [...new Set([].concat(...[arr1, arr2]))];

console.log(result);

//output

//[10,20,30,40,50,60,70,80]

(4) Using concat() and filter() functions

We have already learned the concat() function working process. Now we will use the filter() function along with it to merge and remove duplicates from two arrays in Javascript.

const arr1 = [10, 20, 30, 40];

const arr2 = [50, 40, 90, 20, 10]

var result = arr1.concat(arr2.filter((item) => arr1.indexOf(item) == -1));

console.log(result);

//Output

[10,20,30,40,50,90]

(5) Using concat() and For loop to merge and remove duplicates from an Array

As we know from the previous code examples we can use the concat() function to merge two or multiple arrays into one. We will use Javascript For Loop to remove the duplicate elements from the array merged using the concat() function.

const names1 = ["Rick", "James", "Nora", "Clark", "Rohit"];

const names2 = ["Carol", "Handricks", "Clark", "Stephen", "James"];

// merge the above arrays

const names = names1.concat(names2);

// use for loop to remove duplicate items

for (var i = 0; i < names.length; ++i) {

for (var j = i + 1; j < names.length; ++j) {

if (names[i] === names[j])

names.splice(j--, 1);

}

}

console.log(names);

//Output

//["Rick","James","Nora","Clark","Rohit","Carol","Handricks","Stephen"]