EPAM Suzhou JavaScript Coding Test

1. Write a function that accepts an array of strings. Return the longest or shortest string. Please consider:
   * How to specify which one (longest or shortest) to return?
   * Extensibility: how to specify the length of string to return?

function getFeature(array) {

    if (array.length <= 0) return null;

    let map = [];

    let count = 0;

    for (const index in array) {

        map.push({

            index: count++,

            item: array[index]

        });

    }

    map = map.sort((a, b) => a.item.length - b.item.length);

    return {

        getLongest: () => {

            const longest = map[map.length - 1];

            return longest.item;

        },

        getShortest: () => {

            const shortest = map[0];

            return shortest.item;

        },

        getSpecifiedLengthString: (length) => {

            const i = map.filter(c => c.item.length === length);

            return i ? i : null

        }

    }

}

1. Write a function that calculates the square root of a given number. Format the result with 2 decimals.

function getSquareRoot(n) {

    const number = parseInt(Math.sqrt(n) \* 100);

    return number / 100;

}

1. Write a function that accepts an object and adds logging to a key when the key is assigned new value. For example:

let a = {foo: ‘bar’};

/\*

… Implementation of the **logger** function

\*/

logger(a, ‘bar’);

a.bar = ‘baz’;

Expected console output:

*Replacing ‘bar’ with new value ‘baz’ to ‘foo’*

Please consider:

* 1. How to add logging to all enumerable keys when *key* is not passed to the *logger*?

let a = { foo: 'bar' };

function logger(obj, value) {

    Object.defineProperty(obj, value, {

        set: function (v) {

            for (const i in obj) {

                if (obj[i] === value) {

                    obj[i] = v;

                }

            }

        }

    });

}

logger(a, 'bar');

a.bar = 'baz';

console.log(a);

1. Write a function that accepts one argument as the number of seconds to wait and returns a Promise that will resolve after given seconds. Please consider robustness and testability.

function do (seconds) {

    return new Promise((resolve) => {

        setTimeout(resolve, seconds);

    });

}

1. Have 10 files on the disk, 1.json to 10.json. Every file contains the json object string, i.e. { “name”: “EPAM”, timestamp: 1616339429}. Write code to read the files, and merge all of them as a json array, sort the array by timestamp.

const fs = require('fs');

const files = ['1.json', '2.json', '3.json', '4.json', '5.json', '6.json', '7.json', '8.json', '9.json', '10.json'];

let result = [];

for (const i in files) {

    const line = fs.readFileSync(files[i], 'utf-8');

    const json = JSON.parse(line);

    result.push(json);

}

result = result.sort((a, b) => a.timestamp - b.timestamp);

1. Design a ‘cacheWrapper’ function that accepts any function regardless of the arguments it takes and caches the result of the function. Please consider supporting TTL (time to live).

function cacheWrapper(fn) {

    const data = [];

    data.push({

        value: fn(),

        timestamp: (new Date()).getTime()

    });

    globalThis.data = data;

}