Домашнє завдання №28_1

Виконати домашнє завдання №27_1 повторно(без використання RxJS) за допомогою мови JavaScript для платформи NodeJS(застосовуючи тільки рушій цієї платформи).

** код домашнього завдання №27_1 та домашнього завдання №28_1 має відрізнятися тільки кількома останніми стрічками, які в першому випадку показують використання реактивної парадигми програмування, а в другому випадку – подійно-орієнтованої*

** **коментар:** це завдання аналогічне №27_1, №27_2 та №27_3 і є повністю тотожне до завдань №28_2 та №28_3; таким чином можна порівняти різні засоби програмування; далі наводиться приклад повністю виконаного завдання; для компіляції і запуску можна використати https://repl.it/languages/nodejs .*

Вибір варіанту

Варіант завдання відповідає варіанту домашнього завдання №27_1

Приклад коду

Наведений зразок коду реалізовує завдання для 5-ти максимально допустимих спроб введення ключа ліцензії.

Максимальна кількість спроб для введення ключа ліцензії	5
Оголошення в коді	`const ATTEMPTS_COUNT = 5`

Для коректного виконання коду за допомогою https://repl.it/languages/nodejs віртуальну консоль з правого боку краще трохи розширити перед початком виконання коду, а **у процесі виконання розмір консолі не змінювати**.

Лістинг

```
const ATTEMPTS_COUNT = 5
var attemptsDownCount = ATTEMPTS_COUNT;

const GROUPS_DIGITS_COUNT = 5;
const GROUP_DIGITS_SIZE = 5;

const PRODUCT_KEY_PART1 = [
0xF, 0xF, 0xF, 0xF, 0xF,
0xD, 0xD, 0xD, 0xD, 0xD,
0x8, 0x8, 0x8, 0x8, 0x8,
0xB, 0xB, 0xB, 0xB, 0xB,
0xF, 0xF, 0xF, 0xF, 0xF
];

const PRODUCT_KEY_PART2 = [
0xE, 0xE, 0xE, 0xE, 0xE,
0xF, 0xF, 0xF, 0xF, 0xF,
0xB, 0xB, 0xB, 0xB, 0xB,
0xF, 0xF, 0xF, 0xF, 0xF,
0xA, 0xA, 0xA, 0xA, 0xA
];

const DIGITS_COUNT = GROUPS_DIGITS_COUNT * GROUP_DIGITS_SIZE;

var outOfEdgeIndex = 0;
```

```
var currIndex = 0;
var data = new Array(DIGITS_COUNT).fill(0); // var data = [];

function integerDiv(a, b){
    return (a - a % b) / b;
}

function checkProductKey(productKey){
    for(var index = 0; index < DIGITS_COUNT; ++index){
        if(productKey[index] ^ PRODUCT_KEY_PART1[index] ^ PRODUCT_KEY_PART2[index]){
            return false;
        }
    }

    return true
}

function toDigitPosition(currIndex){
    let positionAddon = integerDiv(currIndex, GROUP_DIGITS_SIZE);
    positionAddon && positionAddon >= GROUPS_DIGITS_COUNT ? --positionAddon : 0;
    process.stdout.cursorTo(currIndex + positionAddon);
}

function printProductKey(productKey, outOfEdgeIndex){
    for(var index = 0; index < DIGITS_COUNT && index < outOfEdgeIndex; ++index){
        process.stdout.write(productKey[index].toString(16) );
    }
}

function printFormattedProductKey(productKey, outOfEdgeIndex){
    for(var index = 0; index < DIGITS_COUNT && index < outOfEdgeIndex; ++index){
        process.stdout.write(productKey[index].toString(16) );
        if(!((index + 1) % GROUP_DIGITS_SIZE) && (index + 1) < DIGITS_COUNT){
            process.stdout.write( '-' );
        }
    }
}

function inputHandler(ch, key) {
    if(!attemptsDownCount){
      return;
    }
    if ( key && key.name == 'return' ) {
        if (checkProductKey(data) ) {
            process.stdout.write("\nThe product key is correct\n\n");
            printProductKey(data, outOfEdgeIndex)
            process.stdout.write(' (COMPLETE)');
            process.stdout.write('\nFor exit press Ctrl + C\n');
            attemptsDownCount = 0;
        }
        else{
            process.stdout.write("\nThe product key is not correct\n");
            process.stdout.write("\nYou have " + --attemptsDownCount + " attempts to try");
            if(attemptsDownCount){
                process.stdout.write("\nPlease, enter the product key:\n");
                printFormattedProductKey(data, outOfEdgeIndex);
                toDigitPosition(currIndex);
            }
            else{
                process.stdout.write("\nThe product key is not entered\n");
                process.stdout.write("For exit press Ctrl + C\n");
            }
        }
    }

    if (key && key.name == 'backspace') {
        if(currIndex){
            --currIndex;
            toDigitPosition(currIndex);
            data[currIndex] = 0;
            process.stdout.write( '0' );
            toDigitPosition(currIndex);
        }
    }
    else if (key && key.name == 'delete') {
        toDigitPosition(currIndex);
        data[currIndex] = 0;
        process.stdout.write( '0' );
        toDigitPosition(currIndex);
    }
    else if (key && key.name == 'left') {
```

```
        if(currIndex){
            toDigitPosition(--currIndex); // got to 1.5
        }
    }
    else if (key && key.name == 'right') {
        if(currIndex < outOfEdgeIndex){
            toDigitPosition(++currIndex);
        }
    }

    ch == ' ' || ch == '\t' ? ch = '0' : 0;

    var hexDigitRegularExpression = /^[0-9A-Fa-f]\b/; // /[0-9A-Fa-f]/g

    if (ch && hexDigitRegularExpression.test(ch) && currIndex < DIGITS_COUNT) {
        data[currIndex] = ch.toUpperCase();
        process.stdout.write( data[currIndex] );
        if(outOfEdgeIndex <= currIndex){
            outOfEdgeIndex = currIndex + 1;
        }
        if(currIndex + 1 < DIGITS_COUNT) {
            ++currIndex;
            if (currIndex != DIGITS_COUNT && !(currIndex % 5)) {
                process.stdout.write( '-' );
            }
        }
        if(currIndex + 1 == DIGITS_COUNT){
            toDigitPosition(currIndex);
        }
    }
}

console.clear();
var keypress = require('keypress');
keypress(process.stdin);

process.stdin.setRawMode(true); // without press enter
process.stdin.setEncoding( 'utf8' );

// Resume stdin in the parent process.
// Node application close all by itself if be an error or process.exit().
process.stdin.resume();

process.stdin.on( 'keypress', (ch, key) => {
    if ( key && key.ctrl && key.name == 'c' ) { // ctrl-c ( return from program )
        process.exit();
    }
});

process.stdin.on( 'keypress', inputHandler);

console.clear();
if(attemptsDownCount){
    process.stdout.write('Please, enter the product key:\n');
}
process.stdout.close; // state out fix
```