# Solutions-JS Advanced: Exam 18.11.2018

# Problem 1. Form Acceptance

function acceptance() {

let $company = $('input[name="shippingCompany"]');

let $product = $('input[name="productName"]');

let $quantity = $('input[name="productQuantity"]')

let $scrape = $('input[name="productScrape"]');

let $warehouse = $('#warehouse');

if (($company.val() && $product.val()) && (Number($quantity.val()) && Number($scrape.val()))) {

let quantity = Number($quantity.val()) - Number($scrape.val());

if (quantity > 0) {

let $div = $('<div>');

let $p = $(`<p>[${$company.val()}] ${$product.val()} - ${quantity} pieces</p>`);

let $button = $('<button>Out of stock</button>');

$button.on('click', () => $div.remove());

$div.append($p, $button);

$warehouse.append($div);

}

}

$company.val('');

$product.val('');

$quantity.val('');

$scrape.val('');

}

|  |
| --- |
| <!DOCTYPE html> |
|  | <html lang="en"> |
|  |  |
|  | <head> |
|  | <meta charset="UTF-8"> |
|  | <title>Title</title> |
|  | <link type="text/css" rel="stylesheet" href="styles.css"> |
|  | <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script> |
|  | <script src="app.js"></script> |
|  | </head> |
|  |  |
|  | <body> |
|  | <div id="protocol"> |
|  | <section> |
|  | <table id="acceptanceForm"> |
|  | <thead> |
|  | <tr> |
|  | <th colspan="5">Acceptance form</th> |
|  | </tr> |
|  | </thead> |
|  | <tbody> |
|  | <tr id="productInfo"> |
|  | <td>Company</td> |
|  | <td>Product</td> |
|  | <td>Quantity</td> |
|  | <td>Scrape</td> |
|  | <td>Acceptance</td> |
|  | </tr> |
|  | <tr id="fields"> |
|  | <td> |
|  | <input type="text" name="shippingCompany" placeholder="Coca-Cola Bulgaria.." value = 'Coca-Cola Bulgaria'/> |
|  | </td> |
|  | <td><input type="text" name="productName" placeholder="Coca-Cola.." value = 'Coca-Cola'/></td> |
|  | <td><input type="text" name="productQuantity" placeholder="200.." value = '100'/></td> |
|  | <td><input type="text" name="productScrape" placeholder="14.." value = '50'/></td> |
|  | <td> |
|  | <button type="button" id="acceptance" onclick="acceptance()">Add it</button> |
|  | </td> |
|  | </tr> |
|  | </tbody> |
|  | </table> |
|  | <br> |
|  | <hr> |
|  | <div id="warehouse"> |
|  | <h1>Stock available in the warehouse</h1> |
|  | </div> |
|  | </section> |
|  | </div> |
|  | </body> |
|  |  |
|  | </html> |

|  |
| --- |
| function acceptance() { |
|  | let fieldsRow = document.getElementById('fields'); |
|  | let fieldsCol = fieldsRow.children; |
|  |  |
|  | let shippingCompanyInput = fieldsCol[0].children[0]; |
|  | let productNameInput = fieldsCol[1].children[0]; |
|  | let productQuantityInput = fieldsCol[2].children[0]; |
|  | let productScrapeInput = fieldsCol[3].children[0]; |
|  |  |
|  | let shippingCompany = shippingCompanyInput.value; |
|  | let productName = productNameInput.value; |
|  | let productQuantity = +productQuantityInput.value; |
|  | let productScrape = +productScrapeInput.value; |
|  |  |
|  | if (isCorrectInput(shippingCompany, productName, productQuantity, productScrape)) { |
|  |  |
|  | addToWarehouse(shippingCompany, productName, productQuantity, productScrape); |
|  |  |
|  | } |
|  | resetInput(shippingCompanyInput); |
|  | resetInput(productNameInput); |
|  | resetInput(productQuantityInput); |
|  | resetInput(productScrapeInput); |
|  |  |
|  | function addToWarehouse(shippingCompany, productName, productQuantity, productScrape) { |
|  | let divElem = document.createElement('div'); |
|  |  |
|  | let pElem = document.createElement('p'); |
|  | pElem.textContent = `[${shippingCompany}] ${productName} - ${productQuantity - productScrape} pieces`; |
|  | divElem.appendChild(pElem); |
|  |  |
|  | let buttonElem = document.createElement('button'); |
|  | buttonElem.textContent = 'Out of stock'; |
|  | buttonElem.addEventListener('click', (e) => { |
|  | e.target.parentNode.remove(); |
|  | }); |
|  |  |
|  | divElem.appendChild(buttonElem); |
|  |  |
|  | document.getElementById('warehouse').appendChild(divElem); |
|  | } |
|  |  |
|  | function isCorrectInput(shippingCompany, productName, productQuantity, productScrape) { |
|  |  |
|  | let isValid = isNonEmptyString(shippingCompany) && |
|  | isNonEmptyString(productName) && |
|  | isNumber(productQuantity) && |
|  | isNumber(productScrape) && |
|  | productQuantity > productScrape; |
|  |  |
|  | return isValid; |
|  | } |
|  |  |
|  | function isNonEmptyString(value) { |
|  |  |
|  | return typeof value === 'string' && value.length > 0; |
|  | } |
|  |  |
|  | function isNumber(value) { |
|  | return typeof value === 'number'; |
|  | } |
|  |  |
|  | function resetInput(elem) { |
|  | elem.value = ''; |
|  | } |
|  | } |

|  |
| --- |
| function acceptance() { |
|  | addProduct(); |
|  |  |
|  | function addProduct() { |
|  |  |
|  | let company = $("input[name=shippingCompany]").val(); |
|  | let product = $("input[name=productName]").val(); |
|  | let quantity = $("input[name=productQuantity]").val(); |
|  | let scrape = $("input[name=productScrape]").val(); |
|  |  |
|  | $('#fields :input').val(''); // reset input fields |
|  |  |
|  | let emptyInputFields = (/\s\s+/g.test(company)) && (/\s\s+/g.test(product)); |
|  |  |
|  | let productValidations = company !== '' |
|  | && product !== '' |
|  | && quantity !== '' |
|  | && scrape !== '' |
|  | && !isNaN(quantity) |
|  | && !isNaN(scrape) |
|  | && quantity > 0 |
|  | && scrape > 0 |
|  | && quantity > scrape; |
|  |  |
|  | if (productValidations && !emptyInputFields) { |
|  | createNewProduct(company, product, quantity, scrape); |
|  | } |
|  |  |
|  | function createNewProduct(company, product, quantity, scrape) { |
|  | let productContainer = $('<div class="parentDiv"></div>'); |
|  |  |
|  | let outOfStockBtn = $('<button type="button" class="outOfStock">Out of stock</button>'); |
|  | let productStr = `[${company}] ${product} - ${quantity - scrape} pieces`; |
|  |  |
|  | let newProduct = $(`<p>${productStr}</p>`); |
|  |  |
|  | productContainer.append(newProduct); |
|  | productContainer.append(outOfStockBtn); |
|  |  |
|  | $('#warehouse').append(productContainer); |
|  |  |
|  | } |
|  |  |
|  | $('.outOfStock:last-child').click(outOfStock); |
|  |  |
|  | function outOfStock(e) { |
|  | e.preventDefault(); |
|  | $(this).parents('#warehouse div').remove(); |
|  | } |
|  | } |
|  | } |

# Problem 2. SoftUniFy (Unit Testing)

class SoftUniFy {

constructor() {

this.allSongs = {};

}

downloadSong(artist, song, lyrics) {

if (!this.allSongs[artist]) {

this.allSongs[artist] = {rate: 0, votes: 0, songs: []}

}

this.allSongs[artist]['songs'].push(`${song} - ${lyrics}`);

return this;

}

playSong(song) {

let songArtists = Object.keys(this.allSongs).reduce((acc, cur) => {

let songs = this.allSongs[cur]['songs']

.filter((songInfo) => songInfo

.split(/ - /)[0] === song);

if(songs.length > 0){

acc[cur] = songs;

}

return acc;

}, {});

let arr = Object.keys(songArtists);

let output = "";

if(arr.length > 0){

arr.forEach((artist) => {

output += `${artist}:\n`;

output += `${songArtists[artist].join('\n')}\n`;

});

} else {

output = `You have not downloaded a ${song} song yet. Use SoftUniFy's function downloadSong() to change that!`

}

return output;

}

get songsList() {

let songs = Object.values(this.allSongs)

.map((v) => v['songs'])

.reduce((acc, cur) => {

return acc.concat(cur);

}, []);

let output;

if (songs.length > 0) {

output = songs.join('\n');

} else {

output = 'Your song list is empty';

}

return output;

}

rateArtist() {

let artistExist = this.allSongs[arguments[0]];

let output;

if (artistExist) {

if (arguments.length === 2) {

artistExist['rate'] += +arguments[1];

artistExist['votes'] += 1;

}

let currentRate = (+(artistExist['rate'] / artistExist['votes']).toFixed(2));

isNaN(currentRate) ? output = 0 : output = currentRate;

} else {

output = `The ${arguments[0]} is not on your artist list.`

}

return output;

}

}

module.exports = {SoftUniFy};

let expect = require('chai').expect;

let SoftUniFy = require('../02Softunify').SoftUniFy;

//In Judge must be paste without this above

describe('SoftUniFy', function(){

describe('constructor', function(){

it('should have property allSongs', function(){

let softUniFy = new SoftUniFy();

expect(softUniFy).to.haveOwnProperty('allSongs');

});

it('should have property allSongs empty object', function(){

let softUniFy = new SoftUniFy();

expect(JSON.stringify(softUniFy.allSongs)).to.be.equal('{}');

});

it('should have property allSongs empty object', function(){

let softUniFy = new SoftUniFy();

expect(softUniFy.allSongs).to.be.eql({});

});

});

describe('downloadSong(artist, song, lyrics)', function(){

let softUniFy;

beforeEach(function(){

softUniFy = new SoftUniFy();

});

it('should add one song to allSongs in correct format', function(){

expect(softUniFy.downloadSong('Eminem', 'Venom', 'Knock, knock let...')).to.be.eql({

'allSongs': {

'Eminem': {

'rate': 0,

'votes': 0,

'songs': ['Venom - Knock, knock let...']

}

}

});

});

it('should add more songs to allSongs in correct format', function(){

softUniFy.downloadSong('Eminem', 'Venom', 'Knock, knock let...');

softUniFy.downloadSong('Eminem', 'Fenomenal', 'I am fenomenal...');

softUniFy.downloadSong('Dub Ex', 'Light Me On Fire', 'You can call me...');

expect(softUniFy).to.be.eql({

'allSongs': {

'Eminem': {

'rate': 0,

'votes': 0,

'songs': ['Venom - Knock, knock let...', 'Fenomenal - I am fenomenal...']

},

'Dub Ex': {

'rate': 0,

'votes': 0,

'songs': ['Light Me On Fire - You can call me...']

}

}

});

});

});

describe('playSong', function(){

let softUniFy;

beforeEach(function(){

softUniFy = new SoftUniFy();

softUniFy.downloadSong('Eminem', 'Venom', 'Knock, knock let...');

softUniFy.downloadSong('Eminem', 'Fenomenal', 'I am fenomenal...');

softUniFy.downloadSong('Fiki', 'Fenomenal', 'I am fenomenal too...');

softUniFy.downloadSong('Dub Ex', 'Light Me On Fire', 'You can call me...');

});

it('should return message when the song is not present', function(){

softUniFy.playSong('Is this Love')

expect(softUniFy.playSong('Is this Love')).to.be.equal(`You have not downloaded a Is this Love song yet. Use SoftUniFy's function downloadSong() to change that!`);

});

it('should return one song', function(){

expect(softUniFy.playSong('Light Me On Fire')).to.be.equal('Dub Ex:\nLight Me On Fire - You can call me...\n');

});

it('should return more songs', function(){

expect(softUniFy.playSong('Fenomenal')).to.be.equal('Eminem:\nFenomenal - I am fenomenal...\nFiki:\nFenomenal - I am fenomenal too...\n');

});

});

describe('songsList', function(){

it('should return message for empty allSongs', function(){

let softUniFy = new SoftUniFy();

expect(softUniFy.songsList).to.be.equal('Your song list is empty');

});

it('should return the songs from allSongs', function(){

let softUniFy = new SoftUniFy();

softUniFy.downloadSong('Eminem', 'Venom', 'Knock, knock let...');

softUniFy.downloadSong('Eminem', 'Fenomenal', 'I am fenomenal...');

softUniFy.downloadSong('Dub Ex', 'Light Me On Fire', 'You can call me...');

expect(softUniFy.songsList).to.be.equal('Venom - Knock, knock let...\nFenomenal - I am fenomenal...\nLight Me On Fire - You can call me...');

});

});

describe('rateArtist()', function(){

it('should return message for unexisting in allSongs artist', function(){

let softUniFy = new SoftUniFy();

expect(softUniFy.rateArtist('Eminem', 50)).to.be.equal('The Eminem is not on your artist list.');

});

it('should return 0 rate for non ratted in allSongs artist', function(){

let softUniFy = new SoftUniFy();

softUniFy.downloadSong('Eminem', 'Fenomenal', 'I am fenomenal...');

expect(softUniFy.rateArtist('Eminem')).to.be.equal(0);

});

it('should return correct rate for ratted in allSongs artist', function(){

let softUniFy = new SoftUniFy();

softUniFy.downloadSong('Eminem', 'Fenomenal', 'I am fenomenal...');

expect(softUniFy.rateArtist('Eminem', 50)).to.be.equal(50);

});

it('should return correct rate for more times ratted in allSongs artist', function(){

let softUniFy = new SoftUniFy();

softUniFy.downloadSong('Eminem', 'Fenomenal', 'I am fenomenal...');

softUniFy.rateArtist('Eminem', 40);

expect(softUniFy.rateArtist('Eminem', 60)).to.be.equal(50);

});

});

});

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | class SoftUniFy { | |  | constructor() { | |  | this.allSongs = {}; | |  | } | |  |  | |  | downloadSong(artist, song, lyrics) { | |  | if (!this.allSongs[artist]) { | |  | this.allSongs[artist] = { rate: 0, votes: 0, songs: [] }; | |  | } | |  |  | |  | this.allSongs[artist]['songs'].push(`${song} - ${lyrics}`); | |  |  | |  | return this; | |  | } | |  |  | |  | playSong(song) { | |  | let songArtists = Object.keys(this.allSongs).reduce((acc, cur) => { | |  |  | |  | let songs = this.allSongs[cur]['songs'] | |  | .filter((songInfo) => songInfo | |  | .split(/ - /)[0] === song); | |  |  | |  | if (songs.length > 0) { | |  | acc[cur] = songs; | |  | } | |  |  | |  | return acc; | |  | }, {}); | |  |  | |  | let arr = Object.keys(songArtists); | |  | let output = ""; | |  |  | |  | if (arr.length > 0) { | |  |  | |  | arr.forEach((artist) => { | |  | output += `${artist}:\n`; | |  | output += `${songArtists[artist].join('\n')}\n`; | |  | }); | |  |  | |  | } else { | |  | output = `You have not downloaded a ${song} song yet. Use SoftUniFy's function downloadSong() to change that!`; | |  | } | |  |  | |  | return output; | |  | } | |  |  | |  | get songsList() { | |  | let songs = Object.values(this.allSongs) | |  | .map((v) => v['songs']) | |  | .reduce((acc, cur) => { | |  | return acc.concat(cur); | |  | }, []); | |  |  | |  | let output; | |  |  | |  | if (songs.length > 0) { | |  | output = songs.join('\n'); | |  | } else { | |  | output = 'Your song list is empty'; | |  | } | |  |  | |  | return output; | |  |  | |  | } | |  |  | |  | rateArtist() { | |  | let artistExist = this.allSongs[arguments[0]]; | |  | let output; | |  |  | |  | if (artistExist) { | |  |  | |  | if (arguments.length === 2) { | |  | artistExist['rate'] += +arguments[1]; | |  | artistExist['votes'] += 1; | |  | } | |  |  | |  | let currentRate = (+(artistExist['rate'] / artistExist['votes']).toFixed(2)); | |  | isNaN(currentRate) ? output = 0 : output = currentRate; | |  |  | |  | } else { | |  | output = `The ${arguments[0]} is not on your artist list.`; | |  | } | |  |  | |  | return output; | |  | } | |  | } | |  |  | |  |  | |  | module.exports = SoftUniFy; |   let SoftUniFy = require('./SoftUniFy'); |
|  | let assert = require('chai').assert; |
|  |  |
|  | describe('SoftUniFy', function () { |
|  | describe('constuctor', function () { |
|  | it('have allSongs property', function () { |
|  | let softUniFy = new SoftUniFy(); |
|  |  |
|  | assert.property(softUniFy, 'allSongs'); |
|  | }); |
|  | it('allSongs is initialized as an empty obj', () => { |
|  | let softUniFy = new SoftUniFy(); |
|  |  |
|  | assert.isEmpty(softUniFy.allSongs); |
|  | }); |
|  | }); |
|  |  |
|  | describe('downloadSong', () => { |
|  | let softUniFy; |
|  | beforeEach(() => { |
|  | softUniFy = new SoftUniFy(); |
|  | }); |
|  |  |
|  | it('return itself', () => { |
|  | let artist = 'Artist'; |
|  | let song = 'Song'; |
|  | let lyrics = 'Lirics'; |
|  |  |
|  | let resultObj = softUniFy.downloadSong(artist, song, lyrics); |
|  |  |
|  | assert.instanceOf(resultObj, SoftUniFy); |
|  | }); |
|  | it('add artist to allSongs', () => { |
|  | let artist = 'Artist'; |
|  | let song = 'Song'; |
|  | let lyrics = 'Lirics'; |
|  |  |
|  | softUniFy.downloadSong(artist, song, lyrics); |
|  |  |
|  | assert.isTrue(softUniFy.allSongs.hasOwnProperty(artist)); |
|  | }); |
|  | it('default rate is zero', () => { |
|  | let artist = 'Artist'; |
|  | let song = 'Song'; |
|  | let lyrics = 'Lirics'; |
|  |  |
|  | softUniFy.downloadSong(artist, song, lyrics); |
|  |  |
|  | assert.equal(softUniFy.allSongs[artist]['rate'], 0); |
|  | }); |
|  | it('default votes is zero', () => { |
|  | let artist = 'Artist'; |
|  | let song = 'Song'; |
|  | let lyrics = 'Lirics'; |
|  |  |
|  | softUniFy.downloadSong(artist, song, lyrics); |
|  |  |
|  | assert.equal(softUniFy.allSongs[artist]['votes'], 0); |
|  | }); |
|  | it('song is added', () => { |
|  | let artist = 'Artist'; |
|  | let song = 'Song'; |
|  | let lyrics = 'Lirics'; |
|  |  |
|  | softUniFy.downloadSong(artist, song, lyrics); |
|  |  |
|  | assert.equal(softUniFy.allSongs[artist]['songs'].length, 1); |
|  | }); |
|  | it('song is added with correct format', () => { |
|  | let artist = 'Artist'; |
|  | let song = 'Song'; |
|  | let lyrics = 'Lirics'; |
|  |  |
|  | softUniFy.downloadSong(artist, song, lyrics); |
|  | let songs = softUniFy.allSongs[artist]['songs']; |
|  |  |
|  | assert.isTrue(songs.indexOf(`${song} - ${lyrics}`) !== -1); |
|  | }); |
|  | it('multiple song', () => { |
|  | let artist = 'Artist'; |
|  | let song = 'Song'; |
|  | let lyrics = 'Lirics'; |
|  |  |
|  | softUniFy.downloadSong(artist, song, lyrics); |
|  | softUniFy.downloadSong(artist, song, lyrics); |
|  | softUniFy.downloadSong(artist, song, lyrics); |
|  |  |
|  | let songs = softUniFy.allSongs[artist]['songs']; |
|  |  |
|  | assert.equal(songs.length, 3); |
|  | }); |
|  | }); |
|  |  |
|  | describe('playSong', () => { |
|  | let softUniFy; |
|  | beforeEach(() => { |
|  | softUniFy = new SoftUniFy(); |
|  | }); |
|  |  |
|  | it('with no downloaded song', () => { |
|  | let song = 'SongName'; |
|  | let expected = `You have not downloaded a ${song} song yet. Use SoftUniFy's function downloadSong() to change that!`; |
|  |  |
|  | let result = softUniFy.playSong(song); |
|  |  |
|  | assert.equal(result, expected); |
|  | }); |
|  | it('with no available song', () => { |
|  | let artist = 'ArtistName'; |
|  | let song = 'SongName'; |
|  | let lyrics = 'Lyrics'; |
|  |  |
|  | let songToPlay = 'SongToPlay'; |
|  | let expected = `You have not downloaded a ${songToPlay} song yet. Use SoftUniFy's function downloadSong() to change that!`; |
|  |  |
|  | softUniFy.downloadSong(artist, song, lyrics); |
|  | let result = softUniFy.playSong(songToPlay); |
|  |  |
|  | assert.equal(result, expected); |
|  | }); |
|  | it('with different artists', () => { |
|  | let artist = 'ArtistName'; |
|  | let song = 'SongName'; |
|  | let lyrics = 'Lyrics'; |
|  |  |
|  | for (let index = 0; index < 10; index++) { |
|  | softUniFy.downloadSong(artist + `${index}`, song + `${index}`, lyrics + `${index}`); |
|  | } |
|  |  |
|  |  |
|  | for (let index = 0; index < 10; index++) { |
|  | let expected = `ArtistName${index}:\nSongName${index} - Lyrics${index}\n`; |
|  | let result = softUniFy.playSong(song + `${index}`); |
|  | assert.equal(result, expected); |
|  | } |
|  | }); |
|  | it('with multiple songs from one artist', () => { |
|  | let artist = 'ArtistName'; |
|  | let song = 'SongName'; |
|  | let lyrics = 'Lyrics'; |
|  |  |
|  | for (let index = 0; index < 10; index++) { |
|  | softUniFy.downloadSong(artist, song + `${index}`, lyrics + `${index}`); |
|  | } |
|  |  |
|  |  |
|  | for (let index = 0; index < 10; index++) { |
|  | let expected = `ArtistName:\nSongName${index} - Lyrics${index}\n`; |
|  | let result = softUniFy.playSong(song + `${index}`); |
|  | assert.equal(result, expected); |
|  | } |
|  | }); |
|  | it('with multiple songs from multiple artists', () => { |
|  | let artist = 'ArtistName'; |
|  | let song = 'SongName'; |
|  | let lyrics = 'Lyrics'; |
|  |  |
|  | let expected = ''; |
|  | for (let index = 0; index < 10; index++) { |
|  | softUniFy.downloadSong(artist + `${index}`, song, lyrics + `${index}`); |
|  | expected += `${artist + index}:\n${song} - ${lyrics + index}\n`; |
|  | } |
|  |  |
|  | let result = softUniFy.playSong(song); |
|  | assert.equal(result, expected); |
|  |  |
|  |  |
|  | }); |
|  | }); |
|  |  |
|  | describe('songList', () => { |
|  | let softUniFy; |
|  | beforeEach(() => { |
|  | softUniFy = new SoftUniFy(); |
|  | }); |
|  |  |
|  | it('with empty songList', () => { |
|  | let expected = 'Your song list is empty'; |
|  |  |
|  | let result = softUniFy.songsList; |
|  |  |
|  | assert.equal(result, expected) |
|  | }); |
|  | it('with multiple songs', () => { |
|  | let artist = 'ArtistName'; |
|  | let song = 'SongName'; |
|  | let lyrics = 'Lyrics'; |
|  |  |
|  | let songs = []; |
|  | for (let index = 0; index < 10; index++) { |
|  | softUniFy.downloadSong(artist + `${index}`, song + `${index}`, lyrics + `${index}`); |
|  | songs.push(`${song + index} - ${lyrics + index}`); |
|  | } |
|  |  |
|  | let expected = songs.join('\n'); |
|  | let result = softUniFy.songsList; |
|  |  |
|  | assert.equal(result, expected); |
|  | }); |
|  | }); |
|  |  |
|  |  |
|  | describe('rateArtist', () => { |
|  | let softUniFy; |
|  | beforeEach(() => { |
|  | softUniFy = new SoftUniFy(); |
|  | }); |
|  |  |
|  | it('with no existing artist', () => { |
|  | let artist = 'ArtistName'; |
|  | let expected = `The ${artist} is not on your artist list.`; |
|  |  |
|  | let result = softUniFy.rateArtist(artist); |
|  |  |
|  | assert.equal(result, expected); |
|  | }); |
|  | it('with new added artist', () => { |
|  | let artist = 'ArtistName'; |
|  | let song = 'SongName'; |
|  | let lyrics = 'Lyrics'; |
|  |  |
|  | softUniFy.downloadSong(artist, song, lyrics); |
|  |  |
|  | let expected = '0'; |
|  |  |
|  | let result = softUniFy.rateArtist(artist); |
|  |  |
|  | assert.equal(result, expected); |
|  | }); |
|  | it('with multiple rate', () => { |
|  | let artist = 'ArtistName'; |
|  | let song = 'SongName'; |
|  | let lyrics = 'Lyrics'; |
|  |  |
|  | softUniFy.downloadSong(artist, song, lyrics); |
|  |  |
|  | let rate = 50; |
|  |  |
|  |  |
|  | let expected = rate; |
|  |  |
|  | softUniFy.rateArtist(artist, rate); |
|  | softUniFy.rateArtist(artist, rate); |
|  | softUniFy.rateArtist(artist, rate); |
|  | let result = softUniFy.rateArtist(artist, rate); |
|  |  |
|  | assert.equal(result, expected); |
|  | }); |
|  | }); |
|  | }); |

//90/100

describe('check if methods work properly', **function** () {

it('rateArtist() with data', **function** () {

            let test = **new** SoftUniFy();

            expect(test.playSong('Jaazz')).to.be.equal(`You have not downloaded a Jaazz song yet. Use SoftUniFy's function downloadSong() to change that!`);

           test.downloadSong('Eminem', 'Venom', 'Knock, Knock let the devil **in**...');

           expect(test.rateArtist('Eminem', 50)).to.be.equal(50);

       });

   });

//90/100

describe('check if methods work properly', **function** () {

        let test;

        beforeEach(**function** () {

            test = **new** SoftUniFy();

        });

        it('playSong with no matches', **function** () {

            let result = test.downloadSong('Eminem', 'Venom', 'Knock, Knock let the devil in...');

            expect(result.playSong('Jaazz')).to.be.equal(`You have not downloaded a Jaazz song yet. Use SoftUniFy's function downloadSong() to change that!`);

       });

   });

|  |
| --- |
| const assert = require('chai').assert; |
|  | const SoftUniFy = require('./softunify'); |
|  |  |
|  | describe('SoftUniFy', function () { |
|  | let softunify; |
|  | beforeEach(() => { |
|  | softunify = new SoftUniFy(); |
|  | }); |
|  |  |
|  | it('should have property', function () { |
|  | assert.property(softunify, 'allSongs'); |
|  | }); |
|  |  |
|  | it('should return correct object', function () { |
|  | let artist = 'artist'; |
|  | let song = 'song'; |
|  | let lyrics = 'lyrics'; |
|  |  |
|  | let expectedResult = { |
|  | 'allSongs': { |
|  | artist: |
|  | { |
|  | rate: 0, |
|  | votes: 0, |
|  | songs: [`${song} - ${lyrics}`] |
|  | } |
|  | } |
|  | }; |
|  |  |
|  | assert.deepEqual(softunify.downloadSong(artist, song, lyrics), expectedResult); |
|  | }); |
|  |  |
|  | it('should return incorrect result', function () { |
|  | softunify.downloadSong('artist', 'song', 'lyrics'); |
|  |  |
|  | let result = softunify.playSong('test'); |
|  | let expectedResult = `You have not downloaded a test song yet. Use SoftUniFy's function downloadSong() to change that!`; |
|  |  |
|  | assert.equal(result, expectedResult); |
|  | }); |
|  |  |
|  | it('should rate artist', function () { |
|  | softunify.downloadSong('artist', 'song', 'lyrics'); |
|  |  |
|  | let result = softunify.rateArtist('test'); |
|  | let expectedResult = `The test is not on your artist list.`; |
|  |  |
|  | assert.equal(result, expectedResult); |
|  | }); |
|  | }); |

|  |
| --- |
| let SoftUniFy = require('../app'); |
|  | let expect = require('chai').expect; |
|  | let assert = require('chai').assert; |
|  |  |
|  | describe("Test class functionality", function() { |
|  | describe("Create instance", function() { |
|  | it("Create instance of the class", function() { |
|  | let expected = new SoftUniFy(); |
|  | expected = expected.allSongs; |
|  |  |
|  | expect(expected).to.be.eql({}); |
|  | }); |
|  | }); |
|  |  |
|  | describe("Download songs", function() { |
|  | it("Add one song", function() { |
|  | let expected = new SoftUniFy(); |
|  | expected = expected.downloadSong("pesho","trepni","trepni trepni trepni"); |
|  |  |
|  | expect(expected).to.be.eql({ |
|  | "allSongs": { |
|  | "pesho": { |
|  | "rate": 0, |
|  | "songs": ["trepni - trepni trepni trepni"], |
|  | "votes": 0 |
|  | } |
|  | } |
|  | }); |
|  | }); |
|  |  |
|  | it("Add two songs", function() { |
|  | let expected = new SoftUniFy(); |
|  | expected = expected.downloadSong("pesho","trepni","trepni trepni trepni"); |
|  | expected = expected.downloadSong("gosho","krisko","losh ili dobur dvete mi lica obichash"); |
|  |  |
|  | expect(expected).to.be.eql({ |
|  | "allSongs": { |
|  | "pesho": { |
|  | "rate": 0, |
|  | "songs": ["trepni - trepni trepni trepni"], |
|  | "votes": 0 |
|  | }, |
|  | "gosho": { |
|  | "rate": 0, |
|  | "songs": ["krisko - losh ili dobur dvete mi lica obichash"], |
|  | "votes": 0 |
|  | } |
|  | } |
|  | }); |
|  | }); |
|  | }); |
|  |  |
|  | describe("Play song", function() { |
|  | it("Play existing song", function() { |
|  | let expected = new SoftUniFy(); |
|  | expected = expected.downloadSong("pesho","trepni","trepni trepni trepni"); |
|  | expected = expected.playSong("trepni"); |
|  |  |
|  | expect(expected).to.be.eql("pesho:\ntrepni - trepni trepni trepni\n"); |
|  | }); |
|  |  |
|  | it("Play non-existing song", function() { |
|  | let expected = new SoftUniFy(); |
|  | expected = expected.downloadSong("pesho","trepni","trepni trepni trepni"); |
|  | expected = expected.playSong("ideal petroff"); |
|  |  |
|  | expect(expected).to.be.eql("You have not downloaded a ideal petroff song yet. Use SoftUniFy's function downloadSong() to change that!"); |
|  | }); |
|  | }); |
|  |  |
|  | describe("Song list", function() { |
|  | it("Get all songs in the list", function() { |
|  | let expected = new SoftUniFy(); |
|  | expected = expected.downloadSong("pesho","trepni","trepni trepni trepni"); |
|  | expected = expected.downloadSong("krisko","ideal petroff","seks buren, stres nulev"); |
|  | expected = expected.songsList; |
|  |  |
|  | expect(expected).to.be.eql("trepni - trepni trepni trepni\nideal petroff - seks buren, stres nulev"); |
|  | }); |
|  |  |
|  | it("Get all songs in empty list", function() { |
|  | let expected = new SoftUniFy(); |
|  | expected = expected.songsList; |
|  |  |
|  | expect(expected).to.be.eql("Your song list is empty"); |
|  | }); |
|  | }); |
|  |  |
|  | describe("Rate artist", function() { |
|  | it("Rate two artists", function() { |
|  | let expected = new SoftUniFy(); |
|  | expected = expected.downloadSong("pesho","trepni","trepni trepni trepni"); |
|  | expected = expected.downloadSong("krisko","ideal petroff","seks buren, stres nulev"); |
|  | expected = expected.rateArtist("pesho"); |
|  |  |
|  | expect(expected).to.be.eql(0); |
|  | }); |
|  |  |
|  | }); |
|  | }); |

## Problem 3. Vacation

class Vacation {

constructor(organizer, destination, budget) {

this.organizer = organizer;

this.destination = destination;

this.budget = budget;

this.kids = {};

}

get numberOfChildren(){

let kidsCount = 0;

for(let grade in this.kids){

kidsCount += this.kids[grade].length;

}

return kidsCount;q

}

registerChild(name, grade, budget) {

if (budget < this.budget) {

return `${name}'s money is not enough to go on vacation to ${this.destination}.`;

}

if (!this.kids.hasOwnProperty(grade)) {

this.kids[grade] = [];

}

for (let kid of this.kids[grade]) {

let kidName = kid.split('-')[0];

if (kidName === name) {

return `${name} is already in the list for this ${this.destination} vacation.`;

}

}

this.kids[grade].push(`${name}-${budget}`);

return this.kids[grade];

}

removeChild(name, grade) {

if(!this.kids.hasOwnProperty(grade)){

return `We couldn't find ${name} in ${grade} grade.`;

}

let filteredKids = this.kids[grade].filter(k => k.split('-')[0] !== name);

if (filteredKids.length === this.kids[grade].length) {

return `We couldn't find ${name} in ${grade} grade.`;

}

else {

this.kids[grade] = filteredKids;

return this.kids[grade];

}

}

toString(){

if (this.numberOfChildren === 0){

return `No children are enrolled for the trip and the organization of ${this.organizer} falls out...`;

}

let kidsList = `${this.organizer} will take ${this.numberOfChildren} children on trip to ${this.destination}\n`;

Object.keys(this.kids).sort((a, b) => a - b).forEach(grade => {

kidsList += `Grade: ${grade}\n`;

for (let i = 0; i < this.kids[grade].length; i++) {

kidsList += (i + 1) + '. ' + this.kids[grade][i] + '\n';

}

});

return kidsList;

//return kidsList.trim();

//return kidsList.splice(-1);

}

}

// let vacation = new Vacation('Mr Pesho', 'San diego', 2000);

// console.log(vacation.registerChild('Gosho', 5, 2000));

// console.log(vacation.registerChild('Lilly', 6, 2100));

// console.log(vacation.registerChild('Pesho', 6, 2400));

// console.log(vacation.registerChild('Gosho', 5, 2000));

// console.log(vacation.registerChild('Tanya', 5, 6000));

// console.log(vacation.registerChild('Mitko', 10, 1590));

// let vacation = new Vacation('Mr Pesho', 'San diego', 2000);

// vacation.registerChild('Gosho', 5, 2000);

// vacation.registerChild('Lilly', 6, 2100);

// console.log(vacation.removeChild('Gosho', 9));

// vacation.registerChild('Pesho', 6, 2400);

// vacation.registerChild('Gosho', 5, 2000);

// console.log(vacation.removeChild('Lilly', 6));

// console.log(vacation.registerChild('Tanya', 5, 6000))

// let vacation = new Vacation('Miss Elizabeth', 'Dubai', 2000);

// vacation.registerChild('Gosho', 5, 3000);

// vacation.registerChild('Lilly', 6, 1500);

// vacation.registerChild('Pesho', 7, 4000);

// vacation.registerChild('Tanya', 5, 5000);

// vacation.registerChild('Mitko', 10, 5500);

// console.log(vacation.toString());

|  |
| --- |
| class Vacation { |
|  | constructor(organizer, destination, budget) { |
|  | this.organizer = organizer; |
|  | this.destination = destination; |
|  | this.budget = budget; |
|  | this.kids = {}; |
|  | } |
|  | get numberOfChildren() { |
|  | let count = 0; |
|  | for (const key in this.kids) { |
|  | if (this.kids.hasOwnProperty(key)) { |
|  | count += this.kids[key].length; |
|  | } |
|  | } |
|  |  |
|  | return count; |
|  | } |
|  |  |
|  | registerChild(name, grade, budget) { |
|  | if (budget < this.budget) { |
|  | return `${name}'s money is not enough to go on vacation to ${this.destination}.`; |
|  | } |
|  |  |
|  | if (!this.kids.hasOwnProperty(grade)) { |
|  | this.kids[grade] = []; |
|  | } |
|  |  |
|  | if (this.kids[grade].join(' ').includes(name)) { |
|  | return `${name} is already in the list for this ${this.destination} vacation.`; |
|  | } |
|  |  |
|  | this.kids[grade].push(`${name}-${budget}`); |
|  |  |
|  | return this.kids[grade]; |
|  | } |
|  |  |
|  | removeChild(name, grade) { |
|  | if (!Object.keys(this.kids).join(' ').includes(grade) || !this.kids[grade].join(' ').includes(name)) { |
|  | return `We couldn't find ${name} in ${grade} grade.`; |
|  | } |
|  |  |
|  | let indexOfKid = -1; |
|  | for (let index = 0; index < this.kids[grade].length; index++) { |
|  | let elem = this.kids[grade][index]; |
|  | if (elem.indexOf(name) !== -1) { |
|  | indexOfKid = index; |
|  | break; |
|  | } |
|  | } |
|  | this.kids[grade].splice(indexOfKid, 1); |
|  |  |
|  | return this.kids[grade]; |
|  | } |
|  |  |
|  | toString() { |
|  | let grades = Object.keys(this.kids); |
|  | if (grades.length === 0) { |
|  | return `No children are enrolled for the trip and the organization of ${this.organizer} falls out...`; |
|  | } |
|  |  |
|  | grades.sort((a, b) => +a - +b); |
|  |  |
|  | let result = `${this.organizer} will take ${this.numberOfChildren} children on trip to ${this.destination}\n`; |
|  |  |
|  | for (const grade of grades) { |
|  | result += `Grade: ${grade}\n`; |
|  |  |
|  | let kidNumber = 1; |
|  | for (const kid of this.kids[grade]) { |
|  | result += `${kidNumber}. ${kid}\n`; |
|  | } |
|  | result += '\n'; |
|  | } |
|  |  |
|  | return result.trim(); |
|  | } |
|  | } |

class Vacation {

constructor(organizer, destination, budget) {

this.organizer = organizer;

this.destination = destination;

this.budget = budget;

this.kids = {};

}

registerChild(name, grade, budget) {

if (budget >= this.budget) {

if (this.kids.hasOwnProperty(grade)) {

let isKidAlreadyInThisGrade = this.kids[grade]

.map(x => x.split('-')[0])

.filter(x => x === name);

if (isKidAlreadyInThisGrade.length > 0) {

return `${name} is already in the list for this ${this.destination} vacation.`

} else {

this.kids[grade].push(`${name}-${budget}`);

}

} else {

this.kids[grade] = [`${name}-${budget}`];

}

return this.kids[grade];

} else {

return `${name}'s money is not enough to go on vacation to ${this.destination}.`;

}

}

removeChild(name, grade) {

if (this.kids.hasOwnProperty(grade)) {

let kidsInThisGrade = this.kids[grade];

for (let kid of kidsInThisGrade) {

let kidName = kid.split('-')[0];

let index = kidsInThisGrade.indexOf(kid);

if (kidName === name) {

this.kids[grade].splice(index, 1);

return this.kids[grade];

}

}

}

return `We couldn't find ${name} in ${grade} grade.`

}

toString() {

let kidsOnVacation = [];

for (let [grade, kids] of Object.entries(this.kids)) {

if(kids.length > 0){

kids.map(x => kidsOnVacation.push(x));

}

}

if (kidsOnVacation.length === 0) {

return `No children are enrolled for the trip and the organization of ${this.organizer} falls out...`

} else {

let result = `${this.organizer} will take ${kidsOnVacation.length} children on trip to ${this.destination}\n`;

let sortedByGrade = Object.entries(this.kids)

.sort((a, b) => a[0].localeCompare(b[0]));

for(let [grade, kids] of sortedByGrade){

result += `Grade: ${grade}\n`;

for(let i = 0; i < kids.length; i++){

result += `${i + 1}. ${kids[i]}\n`;

}

}

return result;

}

}

get numberOfChildren() {

let numberOfKids = 0;

for(let [grade, kids] of Object.entries(this.kids)){

numberOfKids += kids.length

}

return numberOfKids;

}

}

|  |
| --- |
|  |
| class Vacation { |
|  | constructor(organizer, destination, budget) { |
|  | this.organizer = organizer; |
|  | this.destination = destination; |
|  | this.budget = budget; |
|  | this.kids = {}; |
|  | } |
|  |  |
|  | get numberOfChildren() { |
|  | let count = 0; |
|  | for (let grade of Object.keys(this.kids)) { |
|  | count += this.kids[grade].length; |
|  | } |
|  |  |
|  | return count; |
|  | } |
|  |  |
|  | registerChild(name, grade, budget) { |
|  | if (budget < this.budget) { |
|  | return `${name}'s money is not enough to go on vacation to ${this.destination}.`; |
|  | } |
|  |  |
|  | if (!this.kids.hasOwnProperty(grade)) { |
|  | this.kids[grade] = []; |
|  | } |
|  |  |
|  | for (let kid of this.kids[grade]) { |
|  | if (kid.startsWith(name)) { |
|  | return `${name} is already in the list for this ${this.destination} vacation.`; |
|  | } |
|  | } |
|  |  |
|  | this.kids[grade].push(`${name}-${budget}`); |
|  | return this.kids[grade]; |
|  | } |
|  |  |
|  | removeChild(name, grade) { |
|  | if (!this.kids[grade]) { |
|  | return `We couldn't find ${name} in ${grade} grade.`; |
|  | } |
|  |  |
|  | let index = -1; |
|  | for (let i = 0; i < this.kids[grade].length; i++) { |
|  | if (this.kids[grade][i].startsWith(name)) { |
|  | index = i; |
|  | } |
|  | } |
|  |  |
|  | if (index === -1) { |
|  | return `We couldn't find ${name} in ${grade} grade.`; |
|  | } else { |
|  | this.kids[grade].splice(index, 1); |
|  | return grade; |
|  | } |
|  | } |
|  |  |
|  | toString() { |
|  | let result = ''; |
|  | if (this.numberOfChildren === 0) { |
|  | result += `No children are enrolled for the trip and the organization of ${this.organizer} falls out...\n`; |
|  | } else { |
|  | result += `${this.organizer} will take ${this.numberOfChildren} children on trip to ${this.destination}\n`; |
|  | Object.keys(this.kids).sort((a, b) => function (a, b) { |
|  | return this.kids[a] - this.kids[b]; |
|  | }); |
|  |  |
|  | for (let grade of Object.keys(this.kids)) { |
|  | result += `Grade: ${grade}\n`; |
|  | let count = 1; |
|  | for (let kid of this.kids[grade]) { |
|  | result += `${count}. ${kid}\n`; |
|  | count++; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | return result.trim(); |
|  | } |
|  | } |
|  |  |
|  | // let vacation = new Vacation('Mr Pesho', 'San diego', 2000); |
|  | // console.log(vacation.registerChild('Gosho', 5, 2000)); |
|  | // console.log(vacation.registerChild('Lilly', 6, 2100)); |
|  | // console.log(vacation.registerChild('Pesho', 6, 2400)); |
|  | // console.log(vacation.registerChild('Gosho', 5, 2000)); |
|  | // console.log(vacation.registerChild('Tanya', 5, 6000)); |
|  | // console.log(vacation.registerChild('Mitko', 10, 1590)); |
|  | // console.log(vacation.numberOfChildren()); |
|  |  |
|  | // [ 'Gosho-2000' ] |
|  | // [ 'Lilly-2100' ] |
|  | // [ 'Lilly-2100', 'Pesho-2400' ] |
|  | // Gosho is already in the list for this San diego vacation. |
|  | // [ 'Gosho-2000', 'Tanya-6000' ] |
|  | // Mitko's money is not enough to go on vacation to San diego. |
|  |  |
|  | let vacation = new Vacation('Mr Pesho', 'San diego', 2000); |
|  | vacation.registerChild('Gosho', 5, 2000); |
|  | vacation.registerChild('Lilly', 6, 2100); |
|  | console.log(vacation.removeChild('Gosho', 9)); |
|  | vacation.registerChild('Pesho', 6, 2400); |
|  | vacation.registerChild('Gosho', 5, 2000); |
|  | console.log(vacation.removeChild('Lilly', 6)); |
|  | console.log(vacation.registerChild('Tanya', 5, 6000)) |
|  |  |
|  | // We couldn't find Gosho in 9 grade. |
|  | // [ 'Pesho-2400' ] |
|  | // [ 'Gosho-2000', 'Tanya-6000' ] |
|  |  |
|  | // let vacation = new Vacation('Miss Elizabeth', 'Dubai', 2000); |
|  | // vacation.registerChild('Gosho', 5, 3000); |
|  | // vacation.registerChild('Lilly', 6, 1500); |
|  | // vacation.registerChild('Pesho', 7, 4000); |
|  | // vacation.registerChild('Tanya', 5, 5000); |
|  | // vacation.registerChild('Mitko', 10, 5500); |
|  | // console.log(vacation.toString()); |
|  |  |
|  | // Miss Elizabeth will take 4 children on trip to Dubai |
|  | // Grade: 5 |
|  | // 1. Gosho-3000 |
|  | // 2. Tanya-5000 |
|  | // Grade: 7 |
|  | // 1. Pesho-4000 |
|  | // Grade: 10 |
|  | // 1. Mitko-5500 |
|  |  |

|  |
| --- |
| class Vacation { |
|  | constructor(organizer, destination, budget) { |
|  | this.organizer = organizer; |
|  | this.destination = destination; |
|  | this.budget = budget; |
|  | this.kids = {}; |
|  | } |
|  |  |
|  | registerChild(name, grade, budget) { |
|  | if (budget < this.budget) { |
|  | return `${name}'s money is not enough to go on vacation to ${this.destination}.` |
|  | } else { |
|  | if (!this.kids[grade]) { |
|  | this.kids[grade] = []; //`${name}-${budget}` |
|  | } |
|  |  |
|  | let kidValues = this.kids[grade]; |
|  | if (kidValues.filter(o => o['name'] === name).length === 0) { |
|  | kidValues.push({name: name, budget: budget}); |
|  | } else { |
|  | return `${name} is already in the list for this ${this.destination} vacation.`; |
|  | } |
|  |  |
|  | let result = []; |
|  |  |
|  | for (let i = 0; i < this.kids[grade].length; i++) { |
|  | let currentKid = this.kids[grade][i]; |
|  | result.push(`${currentKid.name}-${currentKid.budget}`); |
|  | } |
|  |  |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | removeChild(name, grade) { |
|  | let currentGrade = this.kids[grade]; |
|  |  |
|  | if (currentGrade && currentGrade.filter(o => o.name === name).length > 0) { |
|  |  |
|  | let index = this.kids[grade].findIndex(x => x.name === name); |
|  |  |
|  | if (index !== -1) { |
|  | this.kids[grade].splice(index, 1); |
|  | } |
|  |  |
|  | let result = []; |
|  |  |
|  | for (let i = 0; i < this.kids[grade].length; i++) { |
|  | let currentKid = this.kids[grade][i]; |
|  | result.push(`${currentKid.name}-${currentKid.budget}`); |
|  | } |
|  |  |
|  | return result; |
|  |  |
|  | } else { |
|  | return `We couldn't find ${name} in ${grade} grade.`; |
|  | } |
|  | } |
|  |  |
|  | numberOfChildren() { |
|  | let count = 0; |
|  |  |
|  | for (let i = 0; i < Object.keys(this.kids).length; i++) { |
|  | let currentGrade = (Object.keys(this.kids))[i]; |
|  |  |
|  | for (let kid of Object.values((this.kids[currentGrade]))) { |
|  | count++; |
|  | } |
|  | } |
|  |  |
|  | return count; |
|  | } |
|  |  |
|  | set newBudget(newBudget) { |
|  | this.budget = newBudget; |
|  |  |
|  | for (let i = 0; i < Object.values(this.kids).length; i++) { |
|  | let currentGrade = (Object.values(this.kids)[i]); |
|  |  |
|  | for (let grade of Object.values(currentGrade)) { |
|  |  |
|  | for (const kid of grade) { |
|  | if (kid.budget < newBudget) { |
|  | let index = grade.indexOf(kid); |
|  | grade.splice(index, 1); |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | toString() { |
|  | // sort in ascending order |
|  | let result = ''; |
|  |  |
|  | let ordered = Object.values((this.kids)).sort(); |
|  |  |
|  | result += `${this.organizer} will take ${this.numberOfChildren()} children on trip to ${this.destination}\n`; |
|  |  |
|  |  |
|  | for (let i = 0; i < Object.keys(this.kids).length; i++) { |
|  | let currentGrade = (Object.keys(this.kids))[i]; |
|  | result += `Grade: ${currentGrade}\n`; |
|  |  |
|  | let count = 0; |
|  |  |
|  | for (let kid of Object.values((this.kids[currentGrade]))) { |
|  | count++; |
|  | result += `${count}. ${kid.name}-${kid.budget}\n`; |
|  | } |
|  | } |
|  |  |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | let vacation = new Vacation('Miss Elizabeth', 'Dubai', 2000); |
|  |  |
|  | vacation.registerChild('Gosho', 5, 3000); |
|  | vacation.registerChild('Lilly', 6, 1500); |
|  | vacation.registerChild('Pesho', 7, 4000); |
|  | vacation.registerChild('Tanya', 5, 5000); |
|  | vacation.registerChild('Mitko', 10, 5500); |
|  |  |
|  | console.log(vacation.toString()); |

# Problem 4. Real Estate Agency

function realEstateAgency() {

$('button[name="regOffer"]').on('click', regOffer);

$('button[name="findOffer"]').on('click', findOffer);

let $message = $('#message');

let $building = $('#building');

function regOffer() {

let $rentInput = $('input[name="apartmentRent"]');

let $apartmentInput = $('input[name="apartmentType"]');

let $commissionInput = $('input[name="agencyCommission"]');

let areValidRentAndCommission = (Number($rentInput.val()) && Number($commissionInput.val())) && (Number($commissionInput.val()) >= 0 && Number($commissionInput.val()) <= 100) && (Number($rentInput.val()) > 0);

let isValidApartment = ($apartmentInput.val()) && ($apartmentInput.val().indexOf(':') === -1);

if (areValidRentAndCommission && isValidApartment) {

let $div = createHTMLElement('div', '', 'apartment');

let $rent = createHTMLElement('p', `Rent: ${$rentInput.val()}`);

let $type = createHTMLElement('p', `Type: ${$apartmentInput.val()}`);

let $commission = createHTMLElement('p', `Commission: ${$commissionInput.val()}`);

$div.append($rent, $type, $commission);

$building.append($div);

$message.text('Your offer was created successfully.');

}

else {

$message.text('Your offer registration went wrong, try again.');

}

$rentInput.val('');

$apartmentInput.val('');

$commissionInput.val('');

}

function findOffer() {

let $familyBudget = $('input[name="familyBudget"]');

let $familyApartmentType = $('input[name="familyApartmentType"]');

let $familyName = $('input[name="familyName"]');

let budget = Number($familyBudget.val());

let isBudgetValid = !isNaN(budget) && budget > 0;

let areFamilyNameAndApartmentTypeValid = $familyApartmentType.val() && $familyName.val();

if (isBudgetValid && areFamilyNameAndApartmentTypeValid) {

let $totalAgencyProfit = $('#roof h1');

let isHomeless = true;

for (let apartment of Array.from($('.apartment'))) {

let $type = $(apartment).children().eq(1);

let type = $type.text().split(': ')[1];

if ($familyApartmentType.val() === type) {

let $rent = $(apartment).children().eq(0);

let rent = $rent.text().split(': ')[1];

let $commission = $(apartment).children().eq(2);

let commission = $commission.text().split(': ')[1];

let commissionAmount = Number(rent) \* Number(commission) / 100;

let neededMoney = Number(rent) + commissionAmount;

if (budget >= neededMoney) {

$rent.text(`${$familyName.val()}`);

$type.text('live here now');

$commission.remove();

let $button = createHTMLElement('button', 'MoveOut');

$button.on('click', function() {

$(apartment).remove();

$message.text(`They had found cockroaches in ${$familyName.val()}\'s apartment`);

});

$(apartment).append($button);

$(apartment).css('border', '2px solid red');

$message.text('Enjoy your new home! :))');

isHomeless = false;

let currentAgencyCommission = Number($totalAgencyProfit.text().split(' ')[2]);

let updatedAgencyCommission = currentAgencyCommission + commissionAmount \* 2;

$totalAgencyProfit.text(`Agency profit: ${updatedAgencyCommission} lv.`);

break;

}

}

}

if (isHomeless) {

$message.text('We were unable to find you a home, so sorry :(');

}

}

else {

$message.text('We were unable to find you a home, so sorry :(');

        }

        //By condition, the imputfields must be cleared after evry click on the FindOffer button, but the last test in Judge did not past when this is done!

        // $familyBudget.val('');

// $familyApartmentType.val('');

// $familyName.val('');

}

function createHTMLElement(type, text, className) {

let $element = $(`<${type}>${text}</${type}>`);

if (className) {

$element.addClass(className);

}

return $element;

}

}

|  |
| --- |
| <!DOCTYPE html> |
|  | <html lang="en"> | |
|  |  | |
|  | <head> | |
|  | <meta charset="UTF-8"> | |
|  | <title>Title</title> | |
|  | <link type="text/css" rel="stylesheet" href="styles.css"> | |
|  | <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script> | |
|  | <script src="app.js"></script> | |
|  | </head> | |
|  |  | |
|  | <body onload="realEstateAgency()"> | |
|  | <section> | |
|  | <div id="roof"> | |
|  | <h1>Agency profit: 0 lv.</h1> | |
|  | </div> | |
|  | <div id="building"></div> | |
|  | <div id="offerManager"> | |
|  | <div id="regOffer"> | |
|  | <h2>Reg offer</h2> | |
|  | <input type="text" name="apartmentRent" placeholder="Rent price..." value = "700"> | |
|  | <input type="text" name="apartmentType" placeholder="Apartment type..." value ="two room apartment"> | |
|  | <input type="text" name="agencyCommission" placeholder="Commission rate..." value ="50"> | |
|  | <button name="regOffer">Reg offer</button> | |
|  | </div> | |
|  | <div id="findOffer"> | |
|  | <h2>Find offer</h2> | |
|  | <input type="text" name="familyBudget" placeholder="Family budget...." value = "1100"> | |
|  | <input type="text" name="familyApartmentType" placeholder="Apartment type..." value = "two room apartment"> | |
|  | <input type="text" name="familyName" placeholder="Family name..." value="Peshovi"> | |
|  | <button name="findOffer">Find offer</button> | |
|  | </div> | |
|  | <div id="notifications"> | |
|  | <p id="message"></p> | |
|  | </div> | |
|  | </div> | |
|  | </section> | |
|  | </body> | |
|  |  | |
|  | </html> | |
| function realEstateAgency() { | |
|  | | // Find Offer | |
|  | | let findOfferButton = document.querySelector('#findOffer button'); | |
|  | | findOfferButton.addEventListener('click', () => { | |
|  | | findOffer(); | |
|  | | }); | |
|  | |  | |
|  | | function findOffer() { | |
|  | | let inputs = document.querySelectorAll('#findOffer input'); | |
|  | |  | |
|  | | let familyBudgetElem = inputs[0]; | |
|  | | let familyApartmentTypeElem = inputs[1]; | |
|  | | let familyNameElem = inputs[2]; | |
|  | |  | |
|  | | let familyBudget = +familyBudgetElem.value; | |
|  | | let familyApartmentType = familyApartmentTypeElem.value; | |
|  | | let familyName = familyNameElem.value; | |
|  | |  | |
|  | | if (!isFindOfferValid(familyBudget, familyApartmentType, familyName) || | |
|  | | !tryFindApartment(familyBudget, familyApartmentType, familyName)) { | |
|  | | printMessage('We were unable to find you a home, so sorry :('); | |
|  | |  | |
|  | | } else { | |
|  | | printMessage('Enjoy your new home! :))'); | |
|  | | } | |
|  | |  | |
|  | | clearInputElements(inputs); | |
|  | | } | |
|  | |  | |
|  | | function isFindOfferValid(familyBudget, familyApartmentType, familyName) { | |
|  | | let isValid = isPositiveNumber(familyBudget) && | |
|  | | isValidString(familyApartmentType) && | |
|  | | isValidString(familyName); | |
|  | |  | |
|  | | return isValid; | |
|  | | } | |
|  | |  | |
|  | | function tryFindApartment(familyBudget, familyApartmentType, familyName) { | |
|  | | let buildingElem = document.getElementById('building'); | |
|  | | let index = indexOfApartment(buildingElem, familyApartmentType, familyBudget); | |
|  | |  | |
|  | | if (index < 0) { | |
|  | | return false; | |
|  | | } | |
|  | | let apartment = buildingElem.children[index]; | |
|  | |  | |
|  | | changeAgencyProfit(apartment, familyName); | |
|  | |  | |
|  | | changeApartment(apartment, familyName); | |
|  | | return true; | |
|  | | } | |
|  | |  | |
|  | | function changeAgencyProfit(apartment, familyName) { | |
|  | | let rent = +apartment.children[0].textContent.split(': ')[1]; | |
|  | | let commission = +apartment.children[2].textContent.split(': ')[1]; | |
|  | |  | |
|  | | let newProfit = (rent \* (commission / 100)) \* 2; | |
|  | |  | |
|  | | let roofElem = document.getElementById('roof'); | |
|  | | let profitElem = roofElem.children[0]; | |
|  | |  | |
|  | | let profitTokens = profitElem.textContent.split(' '); | |
|  | | profitTokens[2] = '' + (+profitTokens[2] + newProfit); | |
|  | |  | |
|  | | profitElem.textContent = profitTokens.join(' '); | |
|  | | } | |
|  | |  | |
|  | | function changeApartment(apartment, familyName) { | |
|  | | let rentElem = apartment.children[0]; | |
|  | | let typeElem = apartment.children[1]; | |
|  | | let commissionElem = apartment.children[2]; | |
|  | | commissionElem.remove(); | |
|  | |  | |
|  | | rentElem.textContent = familyName; | |
|  | | typeElem.textContent = 'live here now'; | |
|  | |  | |
|  | | let moveOutButton = document.createElement('button'); | |
|  | | moveOutButton.textContent = 'MoveOut'; | |
|  | | moveOutButton.addEventListener('click', (e) => { | |
|  | | let elemToRemove = e.target.parentNode; | |
|  | | let familyName = elemToRemove.children[0].textContent; | |
|  | | elemToRemove.remove(); | |
|  | |  | |
|  | | printMessage(`They had found cockroaches in ${familyName}\'s apartment`); | |
|  | | }); | |
|  | | apartment.appendChild(moveOutButton); | |
|  | | } | |
|  | |  | |
|  | | function indexOfApartment(building, familyApartmentType, familyBudget) { | |
|  | | let apartments = building.children; | |
|  | |  | |
|  | | for (let index = 0; index < apartments.length; index++) { | |
|  | | let apartment = apartments[index]; | |
|  | | let rentElem = apartment.children[0]; | |
|  | | let typeElem = apartment.children[1]; | |
|  | | let commissionElem = apartment.children[2]; | |
|  | |  | |
|  | | if (isCorrectType(familyApartmentType, typeElem.textContent) && | |
|  | | isEnoughBudget(familyBudget, rentElem.textContent, commissionElem.textContent)) { | |
|  | | return index; | |
|  | | } | |
|  | | } | |
|  | |  | |
|  | | return -1; | |
|  | | } | |
|  | |  | |
|  | | function isEnoughBudget(familyBudget, rentText, commissionText) { | |
|  | | let rent = +rentText.split(': ')[1]; | |
|  | | let commission = +commissionText.split(': ')[1]; | |
|  | |  | |
|  | | let apartmentCost = rent + (rent \* (commission / 100)); | |
|  | |  | |
|  | |  | |
|  | | return familyBudget >= apartmentCost; | |
|  | | } | |
|  | |  | |
|  | | function isCorrectType(familyApartmentType, apartmenTypeText) { | |
|  | | if (apartmenTypeText === 'live here now') { | |
|  | | return false; | |
|  | | } | |
|  | |  | |
|  | | console.log(apartmenTypeText); | |
|  | |  | |
|  | | let apartmenType = apartmenTypeText.split(':')[1].trim(); | |
|  | |  | |
|  | | return familyApartmentType === apartmenType; | |
|  | | } | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | | // Reg Offer part | |
|  | | let regOfferButton = document.querySelector('#regOffer button'); | |
|  | | regOfferButton.addEventListener('click', () => { | |
|  | | regOffer(); | |
|  | | }); | |
|  | |  | |
|  | | function regOffer() { | |
|  | | let inputs = document.querySelectorAll('#regOffer input'); | |
|  | |  | |
|  | | let rentPriceElem = inputs[0]; | |
|  | | let apartmentTypeElem = inputs[1]; | |
|  | | let commissionRateElem = inputs[2]; | |
|  | |  | |
|  | | let rentPrice = +rentPriceElem.value; | |
|  | | let apartmentType = apartmentTypeElem.value; | |
|  | | let commissionRate = +commissionRateElem.value; | |
|  | |  | |
|  | | if (!isRegOfferValid(rentPrice, apartmentType, commissionRate)) { | |
|  | | printMessage('Your offer registration went wrong, try again.'); | |
|  | | } else { | |
|  | | addBuildingElement(rentPrice, apartmentType, commissionRate); | |
|  | | printMessage('Your offer was created successfully.'); | |
|  | | } | |
|  | |  | |
|  | | clearInputElements(inputs); | |
|  | | } | |
|  | |  | |
|  | | function addBuildingElement(rentPrice, apartmentType, commissionRate) { | |
|  | | let buildingElem = document.getElementById('building'); | |
|  | |  | |
|  | | let apartmentDiv = document.createElement('div'); | |
|  | | apartmentDiv.classList.add('apartment'); | |
|  | |  | |
|  | | let rentP = document.createElement('p'); | |
|  | | rentP.textContent = `Rent: ${rentPrice}`; | |
|  | | apartmentDiv.appendChild(rentP); | |
|  | |  | |
|  | | let typeP = document.createElement('p'); | |
|  | | typeP.textContent = `Type: ${apartmentType}`; | |
|  | | apartmentDiv.appendChild(typeP); | |
|  | |  | |
|  | | let commissionP = document.createElement('p'); | |
|  | | commissionP.textContent = `Commission: ${commissionRate}`; | |
|  | | apartmentDiv.appendChild(commissionP); | |
|  | |  | |
|  | | buildingElem.appendChild(apartmentDiv); | |
|  | | } | |
|  | |  | |
|  | | function isRegOfferValid(rentPrice, apartmentType, commissionRate) { | |
|  | | let isValid = isPositiveNumber(rentPrice) && | |
|  | | isValidCommissionRate(commissionRate) && | |
|  | | isValidApartmentType(apartmentType); | |
|  | |  | |
|  | | return isValid; | |
|  | | } | |
|  | |  | |
|  | | function isValidApartmentType(apartmentType) { | |
|  | | return isValidString(apartmentType) && !apartmentType.includes(':'); | |
|  | | } | |
|  | |  | |
|  | | function isValidCommissionRate(commissionRate) { | |
|  | | return isNumber(commissionRate) && (commissionRate >= 0 && commissionRate <= 100); | |
|  | | } | |
|  | |  | |
|  | | function isValidString(value) { | |
|  | | return typeof value === 'string' && value.length > 0; | |
|  | | } | |
|  | |  | |
|  | | function isPositiveNumber(rentPrice) { | |
|  | | return isNumber(rentPrice) && rentPrice > 0; | |
|  | | } | |
|  | |  | |
|  | | function isNumber(value) { | |
|  | | return typeof value === 'number'; | |
|  | | } | |
|  | |  | |
|  | | function clearInputElements(inputs) { | |
|  | | for (const input of inputs) { | |
|  | | input.value = ''; | |
|  | | } | |
|  | | } | |
|  | |  | |
|  | | function printMessage(message) { | |
|  | | let messageElem = document.getElementById('message'); | |
|  | | messageElem.textContent = message; | |
|  | | } | |
|  | | } | |

|  |
| --- |
| function realEstateAgency() { |
|  | $('#regOffer button[name=regOffer]').click(registerOffer); |
|  |  |
|  | $('#findOffer button[name=findOffer]').click(findOffer); |
|  |  |
|  | $('.moveOut').click(moveOutFamily); |
|  |  |
|  | function findOffer() { |
|  | // e.preventDefault(); |
|  |  |
|  | let familyBudget = $('input[name=familyBudget]').val(); |
|  | let familyApartmentType = $('input[name=familyApartmentType]').val(); |
|  | let familyName = $('input[name=familyName]').val(); |
|  |  |
|  | let offerValidation = !isNaN(familyBudget) |
|  | && familyBudget > 0 |
|  | && familyApartmentType !== '' |
|  | && familyApartmentType !== ' ' |
|  | && familyName !== '' |
|  | && familyName !== ' '; |
|  |  |
|  | if (offerValidation) { |
|  | if (searchAnOffer(familyApartmentType, familyBudget, familyName)) { |
|  | printMessage('Enjoy your new home! :)))'); |
|  | } else { |
|  | printMessage('We were unable to find you a home, so sorry :('); |
|  | } |
|  |  |
|  | } else { |
|  | printMessage('Your offer registration went wrong, try again.'); |
|  | } |
|  | } |
|  |  |
|  | function searchAnOffer(familyApartmentType, familyBudget, familyName) { |
|  | let findApartment = false; |
|  |  |
|  | // Foreach all apartments |
|  | let allApartments = $('.apartment'); |
|  |  |
|  | for (const currentApartment of allApartments) { |
|  | let rent = ($(currentApartment).children('p:nth-child(1)')).text(); |
|  | let type = ($(currentApartment).children('p:nth-child(2)')).text(); |
|  | let commission = ($(currentApartment).children('p:nth-child(3)')).text(); |
|  |  |
|  | let rentInMoney = +(rent.replace('Rent: ', '')); |
|  | let commissionInMoney = +(commission.replace('Commission: ', '')); |
|  | let enoughRent = familyBudget >= (rentInMoney + (rentInMoney \* commissionInMoney) / 100); |
|  |  |
|  | if (type === `Type: ${familyApartmentType}` && enoughRent) { |
|  | increaseAgencyProfit(rentInMoney); |
|  | moveTheFamily(familyName, currentApartment); |
|  | $('.moveOut').click(moveOutFamily); |
|  |  |
|  | findApartment = true; |
|  | } |
|  | } |
|  |  |
|  | return findApartment; |
|  | } |
|  |  |
|  | function moveTheFamily(familyName, currentApartment) { |
|  | $(currentApartment).children('p:nth-child(1)').text(familyName); |
|  | $(currentApartment).children('p:nth-child(2)').text('live here now'); |
|  | $(currentApartment).children('p:nth-child(3)').remove(); |
|  |  |
|  | $(currentApartment).append($('<button class="moveOut">MoveOut</button>')); |
|  | $(currentApartment).css('border', '2px solid red'); |
|  | } |
|  |  |
|  | function increaseAgencyProfit(rentInMoney) { |
|  | // <h1>Agency profit: 0 lv.</h1> |
|  | let currentMoney = +($('#roof h1') |
|  | .text() |
|  | .replace('Agency profit: ', '') |
|  | .replace(' lv.', '')); |
|  |  |
|  | currentMoney += rentInMoney; |
|  |  |
|  | $('#roof h1').text(`Agency profit: ${currentMoney} lv.`) |
|  | } |
|  |  |
|  | function registerOffer(e) { |
|  | e.preventDefault(); |
|  |  |
|  | let apartmentRent = $('input[name=apartmentRent]').val(); |
|  | let apartmentType = $('input[name=apartmentType]').val(); |
|  | let agencyCommission = $('input[name=agencyCommission]').val(); |
|  |  |
|  | let offerValidation = !isNaN(apartmentRent) |
|  | && !isNaN(agencyCommission) |
|  | && apartmentRent > 0 |
|  | && agencyCommission >= 0 |
|  | && agencyCommission <= 100 |
|  | && apartmentType !== '' |
|  | && apartmentType !== ' ' |
|  | && !apartmentType.includes(':'); |
|  |  |
|  | // Clear all input fields |
|  | $('#offerManager input').val(''); |
|  |  |
|  | if (offerValidation) { |
|  | postNewOffer(apartmentRent, apartmentType, agencyCommission); |
|  | printMessage('Your offer was created successfully.'); |
|  | } else { |
|  | printMessage('Your offer registration went wrong, try again.'); |
|  | } |
|  | } |
|  |  |
|  | function printMessage(message) { |
|  | $('#message').text(message); |
|  | } |
|  |  |
|  | function postNewOffer(apartmentRent, apartmentType, agencyCommission) { |
|  | let apartment = $('<div class="apartment"></div>') |
|  |  |
|  | let rent = $('<p></p>'); |
|  | rent.text(`Rent: ${apartmentRent}`); |
|  | let type = $('<p></p>'); |
|  | type.text(`Type: ${apartmentType}`); |
|  | let commission = $('<p></p>'); |
|  | commission.text(`Commission: ${agencyCommission}`); |
|  |  |
|  | apartment.append(rent); |
|  | apartment.append(type); |
|  | apartment.append(commission); |
|  |  |
|  | $('#building').append(apartment); |
|  | } |
|  |  |
|  | function moveOutFamily() { |
|  | let selectedFamily = $(this).parent(); |
|  | let familyName = $(selectedFamily).children('p:nth-child(1)').text(); |
|  |  |
|  | selectedFamily.remove(); |
|  | printMessage(`They had found cockroaches in ${familyName}\'s apartment`); |
|  | } |
|  | } |