# JS Advanced - Exam: 17.03.2019

**Problem 1. SoftUni Courses**

function solve() {

let $inputJSFundamentals = $('input[name="js-fundamentals"]');

let $inputJSAdvanced = $('input[name="js-advanced"]');

let $inputJSApplications = $('input[name="js-applications"]');

let $inputJSWeb = $('input[name="js-web"]');

let $inputRadioOnline = $('input[value="online"]');

let $buttonSignMeUp = $('button');

$buttonSignMeUp.on('click', addCourses);

function addCourses(){

let $ulMyCourses = $('#myCourses div[class="courseBody"] ul');

let $pCost = $('#myCourses div[class="courseFoot"] p');

let isCheckedInputJSFundamentals = $inputJSFundamentals.is(':checked');

let isCheckedInputJSAdvanced = $inputJSAdvanced.is(':checked');

let isCheckedInputJSApplications = $inputJSApplications.is(':checked');

let isCheckedInputJSWeb = $inputJSWeb.is(':checked');

let isCheckedinputRadioOnline = $inputRadioOnline.is(':checked');

// let isCheckedInputJSFundamentals;

// let isCheckedInputJSAdvanced;

// let isCheckedInputJSApplications;

// let isCheckedInputJSWeb;

// $inputJSFundamentals.input(function(){

// if ($(this).is(':checked'))

// isCheckedInputJSFundamentals = true;

// });

// $inputJSAdvanced.change(function(){

// if ($(this).is(':checked'))

// isCheckedInputJSAdvanced = true;

// });

let cost = 0;

if (isCheckedInputJSFundamentals){

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

$li.text('JS-Fundamentals');

$ulMyCourses.append($li);

cost += 170;

}

if (isCheckedInputJSAdvanced){

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

$li.text('JS-Advanced');

$ulMyCourses.append($li);

cost += 180;

}

if (isCheckedInputJSApplications){

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

$li.text('JS-Applications');

$ulMyCourses.append($li);

cost += 190;

}

if (isCheckedInputJSWeb){

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

$li.text('JS-Web');

$ulMyCourses.append($li);

cost += 490;

}

if(isCheckedInputJSFundamentals && isCheckedInputJSAdvanced){

cost -= 180 \* 0.1;

}

if (isCheckedInputJSFundamentals && isCheckedInputJSAdvanced && isCheckedInputJSApplications){

cost -= 31.32;

}

if(isCheckedinputRadioOnline){

cost -= cost \* 0.06;

}

if (isCheckedInputJSFundamentals && isCheckedInputJSAdvanced && isCheckedInputJSApplications && isCheckedInputJSWeb){

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

$li.text('HTML and CSS');

$ulMyCourses.append($li);

}

cost = Math.floor(cost);

$pCost.text(`Cost: ${cost}.00 BGN`);

}

}

//in Judge must be paste without this below:

solve();

|  |
| --- |
| function solve() { |
|  | let $signMeButton = $('button'); |
|  |  |
|  | const courses = { |
|  | 'js-fundamentals': 170, |
|  | 'js-advanced': 180, |
|  | 'js-applications': 190, |
|  | 'js-web': 490, |
|  | }; |
|  |  |
|  | const couseName = { |
|  | 'js-fundamentals': 'JS-Fundamentals', |
|  | 'js-advanced': 'JS-Advanced', |
|  | 'js-applications': 'JS-Applications', |
|  | 'js-web': 'JS-Web', |
|  | } |
|  |  |
|  | $signMeButton.on('click', function () { |
|  | let $courseFoot = $('.courseFoot p'); |
|  | let $checkboxes = $('input[type="checkbox"]:checked'); |
|  | let $isOonline = $('input[value="online"]:checked'); |
|  | let $myCourses = $('#myCourses .courseBody ul'); |
|  |  |
|  |  |
|  | $myCourses.empty(); |
|  |  |
|  | let totalPrice = 0; |
|  | let moduleTotalPrice = 0; |
|  |  |
|  | $checkboxes.toArray().forEach(option => { |
|  | totalPrice += courses[$(option).val()]; |
|  | if(couseName[$(option).val()] !== 'js-web'){ |
|  | moduleTotalPrice += courses[$(option).val()]; |
|  | } |
|  |  |
|  | let $li = $('<li>'); |
|  | let textToAppend = couseName[$(option).val()]; |
|  | $li.text(textToAppend); |
|  |  |
|  |  |
|  | $myCourses.append($li); |
|  |  |
|  | }); |
|  |  |
|  | if($checkboxes.length === 4) { |
|  | let $li = $('<li>HTML and CSS</li>'); |
|  |  |
|  | $myCourses.append($li); |
|  | } |
|  |  |
|  | if($($checkboxes[0]).val() === 'js-fundamentals' && $checkboxes[0].checked |
|  | && $($checkboxes[1]).val() === 'js-advanced' && $checkboxes[1].checked) { |
|  | let discount = 180 \* 0.10; |
|  | totalPrice -= discount; |
|  | moduleTotalPrice -= discount; |
|  | } |
|  |  |
|  | if($($checkboxes[0]).val() === 'js-fundamentals' && $checkboxes[0].checked |
|  | && $($checkboxes[1]).val() === 'js-advanced' && $checkboxes[1].checked |
|  | && $($checkboxes[2]).val() === 'js-applications' && $checkboxes[2].checked) { |
|  | totalPrice -= 31.32; |
|  | } |
|  |  |
|  | if($isOonline.length > 0 && totalPrice > 0) { |
|  | totalPrice -= totalPrice \* 0.06; |
|  | } |
|  |  |
|  | $courseFoot.text(`Cost: ${Math.floor(totalPrice).toFixed(2)} BGN`); |
|  |  |
|  | }) |
|  | } |
|  |  |
|  | solve(); |

function solve() {

$('.courseFoot button').click(function () {

let selectedCources = $('.courseBody ul li input:checked').toArray().map(x => x.value);

let onlineOnsite = $('#educationForm input:checked').val();

if (selectedCources.length !== 0) {

$('#myCourses ul').empty();

let courcePrice = {

"js-fundamentals": 170,

"js-advanced": 180,

"js-applications": 190,

"js-web": 490

};

let realNames = {

"js-fundamentals": "JS-Fundamentals",

"js-advanced": "JS-Advanced",

"js-applications": "JS-Applications",

"js-web": "JS-Web"

};

let discount = 0;

let price = 0;

if (selectedCources.indexOf('js-fundamentals') !== -1 &&

selectedCources.indexOf('js-advanced') !== -1) {

courcePrice["js-advanced"] = 162;

}

if (selectedCources.indexOf('js-fundamentals') !== -1 &&

selectedCources.indexOf('js-advanced') !== -1 &&

selectedCources.indexOf('js-applications') !== -1) {

courcePrice["js-fundamentals"] = 0;

courcePrice["js-advanced"] = 490.68;

courcePrice["js-applications"] = 0;

}

for (const cource of selectedCources) {

price += courcePrice[cource];

$('#myCourses ul').append($(`<li>${realNames[cource]}</li>`));

}

if (selectedCources.indexOf('js-fundamentals') !== -1 &&

selectedCources.indexOf('js-advanced') !== -1 &&

selectedCources.indexOf('js-applications') !== -1 &&

selectedCources.indexOf('js-web') !== -1) {

$('#myCourses ul').append($(`<li>HTML and CSS</li>`));

}

if (onlineOnsite === 'online') {

discount += 6;

}

price = price \* ((100 - discount) / 100);

$('.courseFoot p').text(`Cost: ${Math.floor(price).toFixed(2)} BGN`);

}

});

}

function solve() {

let coursesObj = {

'JS Fundamentals': 170,

'JS Advanced': 180,

'JS Applications': 190,

'JS Web': 490

}

let price = 0;

let check = 0;

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

let inputArr = document.getElementsByTagName('input');

if ($(inputArr[0]).is(':checked') && $(inputArr[1]).is(':checked')

&& $(inputArr[2]).is(':checked') && $(inputArr[3]).is(':checked')){

price = (170 + 180 \* 0.90 + 190) \* 0.94 + 490;

check = 1;

} else if ($(inputArr[0]).is(':checked') && $(inputArr[1]).is(':checked')

&& $(inputArr[2]).is(':checked')){

price = (170 + 180 \* 0.90 + 190) \* 0.94;

} else if ($(inputArr[0]).is(':checked') && $(inputArr[1]).is(':checked')) {

price = 170 + 180 \* 0.90;

} else {

for (let i = 0; i <= 3; i++) {

if ($(inputArr[i]).is(':checked')) {

let el = $(inputArr[i]).next().text().split(' - ')[0];

price += coursesObj[el]

}

}

}

for (let i = 0; i <= 3; i++) {

let print = document.getElementsByClassName('courseBody')[1].children;

let ul = print[0];

if ($(inputArr[i]).is(':checked')) {

let li = document.createElement('li')

$(li).text($(inputArr[i]).next().text().split('-')[0].replace(' ', '-').trim())

ul.appendChild(li)

}

}

if (check) {

let print = document.getElementsByClassName('courseBody')[1].children;

let ul = print[0];

let li = document.createElement('li')

$(li).text('HTML and CSS')

ul.appendChild(li)

}

if ($(inputArr[5]).is(':checked')) {

price = price \* 0.94

}

if (price) {

let output = Array.from(document.getElementsByClassName('courseFoot'));

output[1].getElementsByTagName('p')[0].textContent = `Cost: ${Math.floor(price)}.00 BGN`

}

})

}

solve();

function solve() {

let prices = {

'js-fundamentals': 170,

'js-advanced': 180,

'js-applications': 190,

'js-web': 490

}

let namesOfCourses = {

'js-fundamentals': 'JS-Fundamentals',

'js-advanced': 'JS-Advanced',

'js-applications': 'JS-Applications',

'js-web': 'JS-Web'

}

let $btn = $('#availableCourses > div.courseFoot > button');

$btn.on('click', signMeUp);

function signMeUp(){

let courses = [];

let $checkedInput = $('input[type="checkbox"]:checked');

if ($checkedInput) {

for (let input of $checkedInput) {

courses.push(input.value);

}

let checkedInputRadio = $('input[type="radio"]:checked').val();

let $coursesUl = $('#myCourses > div.courseBody > ul');

for (let course of courses) {

let $li = $(`<li>${namesOfCourses[course]}</li>`);

$coursesUl.append($li);

}

let cost = 0;

let isOnline = checkedInputRadio === 'online';

let tenPercentDiscountTrue = courses.includes('js-advanced') &&

courses.includes('js-fundamentals');

let moduleDiscountTrue = courses.includes('js-advanced') &&

courses.includes('js-fundamentals') &&

courses.includes('js-applications');

let bonusCourseTrue = courses.includes('js-advanced') &&

courses.includes('js-fundamentals') &&

courses.includes('js-applications') &&

courses.includes('js-web');

console.log(bonusCourseTrue);

if (tenPercentDiscountTrue) {

let oldPrice = prices['js-advanced'];

prices['js-advanced'] = oldPrice \* 0.9;

if (moduleDiscountTrue) {

cost = (prices['js-fundamentals'] + prices['js-advanced'] + prices['js-applications']) \* 0.94;

if (bonusCourseTrue) {

let $li = $('<li>HTML and CSS</li>');

cost += prices['js-web']

$coursesUl.append($li);

}

}else{

cost = 0;

for (let course of courses) {

cost += prices[course];

}

}

}else{

cost = 0;

for (let course of courses) {

cost += prices[course];

}

}

if (isOnline){

cost = cost\*0.94;

}

let $sum = $('#myCourses > div.courseFoot > p');

$sum.text(`Cost: ${Math.floor(cost)}.00 BGN`)

}

}

}

## Problem 2. Film Studio

class FilmStudio {

constructor(studioName) {

this.name = studioName;

this.films = [];

}

casting(actor, role) {

let isTheActorIsUnemployed = true;

let output;

if (this.films.length) {

for (let f of this.films) {

let roles = f.filmRoles.filter((r) => r.role === role);

if (roles.length) {

let filmIndex = this.films.indexOf(f);

let wantedRole = this.films[filmIndex].filmRoles.filter((fR) => fR.role === role)[0];

let roleIndex = this.films[filmIndex].filmRoles.indexOf(wantedRole);

this.films[filmIndex].filmRoles[roleIndex].actor = actor;

isTheActorIsUnemployed = false;

output = `You got the job! Mr. ${actor} you are next ${role} in the ${f.filmName}. Congratz!`;

break;

}

}

if (isTheActorIsUnemployed) {

output = `${actor}, we cannot find a ${role} role...`;

}

} else {

output = `There are no films yet in ${this.name}.`;

}

return output;

}

makeMovie(filmName, roles) {

if (arguments.length === 2) {

let firstArgIsString = typeof arguments[0] === 'string';

let secondArgIsArray = arguments[1] instanceof Array;

if (firstArgIsString && secondArgIsArray) {

let findedFilms = this.films.filter((f) => f.filmName.includes(filmName));

let filmRoles = roles.reduce((acc, cur) => {

let curFilmRole = {

role: cur,

actor: false

};

acc.push(curFilmRole);

return acc;

}, []);

let film = {

filmName,

filmRoles

};

if (findedFilms.length > 0) {

film.filmName += ` ${++findedFilms.length}`;

}

this.films.push(film);

return film;

} else {

throw ('Invalid arguments')

}

} else {

throw ('Invalid arguments count')

}

}

lookForProducer(film) {

let f = this.films.filter((f) => f.filmName === film)[0];

let output;

if (f) {

output = `Film name: ${f.filmName}\n`;

output += 'Cast:\n';

Object.keys(f.filmRoles).forEach((role) => {

output += `${f.filmRoles[role].actor} as ${f.filmRoles[role].role}\n`;

});

} else {

throw new Error(`${film} do not exist yet, but we need the money...`)

}

return output;

}

}

module.exports = {FilmStudio};

let filmStudio = new FilmStudio('Verginia');

filmStudio.makeMovie('The Avengers', ['Iron-Man', 'Thor', 'Hulk', 'Arrow guy']);

console.log(filmStudio.lookForProducer('The Avengers'));

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

let FilmStudio = require('../02FilmStudio').FilmStudio;

//In Judge must be paste without this above

describe('SoftUniFy', function () {

describe('constructor', function () {

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

let filmStudio = new FilmStudio('Verginia');

expect(filmStudio).to.haveOwnProperty('name');

});

it('should have correct property name', function () {

let filmStudio = new FilmStudio('Verginia');

expect(filmStudio.name).to.be.equal('Verginia');

});

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

let filmStudio = new FilmStudio();

expect(filmStudio).to.haveOwnProperty('films');

});

it('should have property films empty array', function () {

let filmStudio = new FilmStudio();

expect(JSON.stringify(filmStudio.films)).to.be.equal('[]');

});

it('should have property films empty array', function () {

let filmStudio = new FilmStudio();

expect(filmStudio.films).to.be.eql([]);

});

it('should have property films empty array', function () {

let filmStudio = new FilmStudio('Verginia');

expect(JSON.stringify(filmStudio.films)).to.be.equal('[]');

});

it('should have property films empty array', function () {

let filmStudio = new FilmStudio('Verginia');

expect(filmStudio.films).to.be.eql([]);

});

});

describe('makeMovie(filmName, roles)', function () {

let filmStudio;

beforeEach(function () {

filmStudio = new FilmStudio('Verginia');

});

it('should throw by invalid arguments count', function () {

expect(() => { filmStudio.makeMovie('Vanilla scay'); }).to.throw('Invalid arguments count');

});

it('should throw by invalid arguments count', function () {

expect(() => { filmStudio.makeMovie('Vanilla scay', ['roles'], 'wrongArgument'); }).to.throw('Invalid arguments count');

});

it('should throw by invalid argument filmName', function () {

expect(() => { filmStudio.makeMovie(9, ['roles']); }).to.throw('Invalid arguments');

expect(() => { filmStudio.makeMovie([], ['roles']); }).to.throw('Invalid arguments');

expect(() => { filmStudio.makeMovie(true, ['roles']); }).to.throw('Invalid arguments');

});

it('should throw by invalid argument roles', function () {

expect(() => { filmStudio.makeMovie('Vanilla scay', 'roles'); }).to.throw('Invalid arguments');

expect(() => { filmStudio.makeMovie('Vanilla scay', { 'roles': 1 }); }).to.throw('Invalid arguments');

expect(() => { filmStudio.makeMovie('Vanilla scay', 1); }).to.throw('Invalid arguments');

});

it('should return corect value for one film ', function () {

expect(filmStudio.makeMovie('The Avengers', ['Iron-Man', 'Thor', 'Hulk', 'Arrow guy'])).to.be.eql({

filmName: 'The Avengers',

filmRoles:

[{ role: 'Iron-Man', actor: false },

{ role: 'Thor', actor: false },

{ role: 'Hulk', actor: false },

{ role: 'Arrow guy', actor: false }]

}

);

});

it('should return corect value for one existing film ', function () {

filmStudio.makeMovie('The Avengers', ['Iron-Man', 'Thor', 'Hulk', 'Arrow guy']);

expect(filmStudio.makeMovie('The Avengers', ['Iron-Man', 'Hulk', 'Arrow guy', 'Ant-man'])).to.be.eql({

filmName: 'The Avengers 2',

filmRoles:

[{ role: 'Iron-Man', actor: false },

{ role: 'Hulk', actor: false },

{ role: 'Arrow guy', actor: false },

{ role: 'Ant-man', actor: false }]

}

);

});

it('should return corect value for one new film by existing films', function () {

filmStudio.makeMovie('The Avengers', ['Iron-Man', 'Thor', 'Hulk', 'Arrow guy']);

filmStudio.makeMovie('The Avengers', ['Iron-Man', 'Hulk', 'Arrow guy', 'Ant-man']);

expect(filmStudio.makeMovie('The New Avengers', ['Iron-Man', 'Thor', 'Hulk', 'Arrow guy', 'Black Panther'])).to.be.eql({

filmName: 'The New Avengers',

filmRoles:

[{ role: 'Iron-Man', actor: false },

{ role: 'Thor', actor: false },

{ role: 'Hulk', actor: false },

{ role: 'Arrow guy', actor: false },

{ role: 'Black Panther', actor: false }]

}

);

});

});

describe('casting(actor, role)', function () {

let filmStudio;

beforeEach(function () {

filmStudio = new FilmStudio('Verginia');

});

it('should return notsuccess message for no films', function () {

expect(filmStudio.casting('Leo De Caprio', 'detectiv')).to.be.equal(`There are no films yet in Verginia.`);

});

it('should return notsuccess message for no roles', function () {

filmStudio.makeMovie('The Avengers', ['Iron-Man', 'Thor', 'Hulk', 'Arrow guy']);

expect(filmStudio.casting('Leo De Caprio', 'detectiv')).to.be.equal(`Leo De Caprio, we cannot find a detectiv role...`);

});

it('should return success message for find role', function () {

filmStudio.makeMovie('The Avengers', ['Iron-Man', 'Thor', 'Hulk', 'Arrow guy']);

expect(filmStudio.casting('Leo De Caprio', 'Thor')).to.be.equal(`You got the job! Mr. Leo De Caprio you are next Thor in the The Avengers. Congratz!`);

});

});

describe('lookForProducer(film)', function () {

let filmStudio;

beforeEach(function () {

filmStudio = new FilmStudio('Verginia');

});

it('should throw error for not finded film', function () {

filmStudio.makeMovie('The Avengers', ['Iron-Man', 'Thor', 'Hulk', 'Arrow guy']);

expect(() => { filmStudio.lookForProducer('The Vilage'); }).to.throw(`The Vilage do not exist yet, but we need the money...`);

});

it('should return correct finded film', function () {

filmStudio.makeMovie('The Avengers', ['Iron-Man', 'Thor', 'Hulk', 'Arrow guy']);

expect(filmStudio.lookForProducer('The Avengers')).to.be.equal('Film name: The Avengers\nCast:\nfalse as Iron-Man\nfalse as Thor\nfalse as Hulk\nfalse as Arrow guy\n');

});

});

});

|  |
| --- |
| let FilmStudio = require('../filmStudio'); |
|  | let {assert} = require('chai'); |
|  |  |
|  | describe('Testing FilmStudio', function () { |
|  | describe('Testing constructor', function () { |
|  | it('should create a studi with correct name', function () { |
|  | let studio = new FilmStudio('test'); |
|  |  |
|  | assert.equal(studio.name, 'test'); |
|  | }); |
|  | }); |
|  |  |
|  | describe('Testing makeMovie(filmName, roles)', function () { |
|  | let studio; |
|  | beforeEach(function () { |
|  | studio = new FilmStudio(); |
|  | }) |
|  |  |
|  | it('should create film', function () { |
|  | let movie = studio.makeMovie('test', ['test', 'test2']); |
|  | let result = { filmName: 'test', |
|  | filmRoles: |
|  | [ { role: 'test', actor: false }, |
|  | { role: 'test2', actor: false } ] }; |
|  |  |
|  | assert.deepEqual(movie, result); |
|  | }); |
|  |  |
|  | it('should create sequel', function () { |
|  | studio.makeMovie('test', ['test']); |
|  | let sequel = studio.makeMovie('test', ['test', 'Test2']); |
|  |  |
|  | let result = { filmName: 'test 2', |
|  | filmRoles: |
|  | [ { role: 'test', actor: false }, { role: 'Test2', actor: false } ] }; |
|  |  |
|  | assert.deepEqual(sequel, result); |
|  | }); |
|  |  |
|  | it('should create film with 1 actor', function () { |
|  | let movie = studio.makeMovie('test', ['test']); |
|  | let result = { filmName: 'test', |
|  | filmRoles: |
|  | [ { role: 'test', actor: false } ] }; |
|  |  |
|  | assert.deepEqual(movie, result); |
|  | }); |
|  |  |
|  | it('should throw error if less than 1 parameter is passed', function () { |
|  | assert.throws(function () { |
|  | studio.makeMovie('test'); |
|  | }, 'Invalid arguments count'); |
|  | }); |
|  |  |
|  | it('should throw error if 0 parameters are passed', function () { |
|  | assert.throws(function () { |
|  | studio.makeMovie(); |
|  | }, 'Invalid arguments count'); |
|  | }); |
|  |  |
|  | it('should throw error if more than 2 parameters are passed', function () { |
|  | assert.throws(function () { |
|  | studio.makeMovie('test', ['asd'], 'test'); |
|  | }, 'Invalid arguments count'); |
|  | }); |
|  |  |
|  | it('should throw error if first arg is not a string', function () { |
|  | assert.throws(function () { |
|  | studio.makeMovie(12, ['asd']); |
|  | }, 'Invalid arguments'); |
|  | }); |
|  |  |
|  | it('should throw error if first arg is object', function () { |
|  | assert.throws(function () { |
|  | studio.makeMovie({name: 'test'}, ['asd']); |
|  | }, 'Invalid arguments'); |
|  | }); |
|  |  |
|  | it('should throw error if first arg is func', function () { |
|  | assert.throws(function () { |
|  | studio.makeMovie(() => {}, ['asd']); |
|  | }, 'Invalid arguments'); |
|  | }); |
|  |  |
|  | it('should throw error if first arg is arr', function () { |
|  | assert.throws(function () { |
|  | studio.makeMovie([ 'test'], ['asd']); |
|  | }, 'Invalid arguments'); |
|  | }); |
|  |  |
|  | it('should throw error if second arg is int', function () { |
|  | assert.throws(function () { |
|  | studio.makeMovie('test', 2); |
|  | }, 'Invalid arguments'); |
|  | }); |
|  |  |
|  | it('should throw error if second arg is string', function () { |
|  | assert.throws(function () { |
|  | studio.makeMovie('test', 'testt'); |
|  | }, 'Invalid arguments'); |
|  | }); |
|  |  |
|  | it('should throw error if second arg is {}', function () { |
|  | assert.throws(function () { |
|  | studio.makeMovie('test', {name: 'test'}); |
|  | }, 'Invalid arguments'); |
|  | }); |
|  |  |
|  | it('should throw error if second arg is func', function () { |
|  | assert.throws(function () { |
|  | studio.makeMovie('test', () => {}); |
|  | }, 'Invalid arguments'); |
|  | }); |
|  |  |
|  | it('should create film, assign actor, and actor is employed', function () { |
|  | studio.makeMovie('test', ['test', 'test2']); |
|  | studio.casting('imence', 'test') |
|  |  |
|  | let test = studio.films[0].filmRoles[0].actor; |
|  |  |
|  | assert.deepEqual(test, 'imence'); |
|  | }); |
|  |  |
|  | }); |
|  |  |
|  | describe('Testing casting(actor, role)', function () { |
|  | let stidio; |
|  | beforeEach(function () { |
|  | studio = new FilmStudio('SU'); |
|  | studio.makeMovie('Test movie', ['test 1', 'test 2']); |
|  | }) |
|  |  |
|  | it('should add actor', function () { |
|  | let test = studio.casting('ime', 'test 2'); |
|  | let result = "You got the job! Mr. ime you are next test 2 in the Test movie. Congratz!" |
|  |  |
|  | assert.equal(test, result); |
|  | }); |
|  |  |
|  | it('should add actor if more than 1 movie with same role persist', function () { |
|  | studio.makeMovie('Test movie', ['test 2', "test 3"]); |
|  | let test = studio.casting('ime', 'test 2'); |
|  | let result = "You got the job! Mr. ime you are next test 2 in the Test movie. Congratz!" |
|  |  |
|  | assert.equal(test, result); |
|  | }); |
|  |  |
|  | it('should add actor if more than 1 movie with same role persist', function () { |
|  | studio.makeMovie('Test movie', ['test 2', "test 3"]); |
|  | let test = studio.casting('ime', 'test 2'); |
|  | let result = "You got the job! Mr. ime you are next test 2 in the Test movie. Congratz!" |
|  |  |
|  | assert.equal(test, result); |
|  | }); |
|  |  |
|  | it('should add actor if more than 1 role in the same movie persist', function () { |
|  | studio.makeMovie('Test movie 2', ['test 3', "test 2", "test 3"]); |
|  | studio.casting('ime', 'test 3'); |
|  | test = studio.casting('ime', 'test 2'); |
|  | let result = "You got the job! Mr. ime you are next test 2 in the Test movie. Congratz!" |
|  |  |
|  | assert.equal(test, result); |
|  | }); |
|  |  |
|  | it('should return message if no movies', function () { |
|  | let newStudio = new FilmStudio('Asd'); |
|  |  |
|  | let test = newStudio.casting('ime', 'test 3'); |
|  | let result = "There are no films yet in Asd." |
|  |  |
|  | assert.equal(test, result); |
|  | }); |
|  |  |
|  | it('should return message if no roles found', function () { |
|  |  |
|  | let test = studio.casting('ime', 'test 3'); |
|  | let result = "ime, we cannot find a test 3 role..." |
|  |  |
|  | assert.equal(test, result); |
|  | }); |
|  |  |
|  | }); |
|  |  |
|  | describe('Testing lookForProducer(film)', function () { |
|  | let studio; |
|  | beforeEach(function () { |
|  | studio = new FilmStudio('Asd'); |
|  | studio.makeMovie('Test film', ['actor1', 'actor2']); |
|  | }); |
|  |  |
|  | it('should print correct result', function () { |
|  | let test = studio.lookForProducer('Test film'); |
|  | let result = "Film name: Test film\n" + |
|  | "Cast:\n" + |
|  | "false as actor1\n" + |
|  | "false as actor2\n"; |
|  |  |
|  | assert.equal(test, result); |
|  | }); |
|  |  |
|  | it('should print correct result with actor employed ', function () { |
|  | studio.casting('imence', 'actor1'); |
|  | let test = studio.lookForProducer('Test film'); |
|  | let result = "Film name: Test film\n" + |
|  | "Cast:\n" + |
|  | "imence as actor1\n" + |
|  | "false as actor2\n"; |
|  |  |
|  | assert.equal(test, result); |
|  | }); |
|  |  |
|  | it('should throw error if no movie found', function () { |
|  | assert.throws(function () { |
|  | studio.lookForProducer('tup film'); |
|  | }, 'tup film do not exist yet, but we need the money...'); |
|  | }); |
|  | }); |
|  | }); |

## Problem 3. Hotel

class Hotel {

constructor(name, capacity) {

this.name = name;

this.capacity = capacity;

this.bookings = [];

this.currentBookingNumber = 1;

this.rooms = {

single: Math.floor(this.capacity / 2),

double: Math.floor(this.capacity \* 0.3),

maisonette: Math.floor(this.capacity \* 0.2),

};

}

get roomsPricing() {

return {

single: 50,

double: 90,

maisonette: 135

};

}

get servicesPricing() {

return {

food: 10,

drink: 15,

housekeeping: 25

};

}

rentARoom(name, type, nights) {

if (this.rooms[type] <= 0) {

let outputMessage = `No ${type} rooms available!`;

let availableRooms = Object.keys(this.rooms);

availableRooms.forEach(room => {

if (room !== type && this.rooms[room] > 0) {

outputMessage += ` Available ${room} rooms: ${this.rooms[room]}.`;

}

});

return outputMessage;

}

let roomBooking = {

name,

type,

nights,

number: this.currentBookingNumber

};

let bookingNumber = this.currentBookingNumber;

this.bookings.push(roomBooking);

this.currentBookingNumber++;

this.capacity--;

this.rooms[type]--;

return `Enjoy your time here Mr./Mrs. ${name}. Your booking is ${bookingNumber}.`;

}

roomService(currentBookingNumber, serviceType) {

let roomBooking = null;

for (let booking of this.bookings) {

if (booking.number === currentBookingNumber) {

roomBooking = booking;

break;

}

}

//let roomBooking = this.bookings.find(b => b.number === currentBookingNumber);//one test in Judge doesn't pass

if (roomBooking === null) {

return `The booking ${currentBookingNumber} is invalid.`;

}

if (!this.servicesPricing.hasOwnProperty(serviceType)) {

return `We do not offer ${serviceType} service.`;

}

if (!roomBooking.hasOwnProperty('services')) {

roomBooking.services = [];

}

roomBooking.services.push(serviceType);

return `Mr./Mrs. ${roomBooking.name}, Your order for ${serviceType} service has been successful.`;

}

checkOut(currentBookingNumber) {

let roomBooking = null;

let index = 0;

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

if (this.bookings[i].number === currentBookingNumber){

roomBooking = this.bookings[i];

index = i;

break;

}

}

if (roomBooking === null) {

return `The booking ${currentBookingNumber} is invalid.`;

}

let totalMoney = roomBooking.nights \* this.roomsPricing[roomBooking.type];

let outputMessage = `We hope you enjoyed your time here, Mr./Mrs. ${roomBooking.name}. The total amount of money you have to pay is ${totalMoney} BGN.`;

if (roomBooking.hasOwnProperty('services')) {

let totalServiceMoney = 0;

for (let serviceType of roomBooking.services) {

totalServiceMoney += this.servicesPricing[serviceType];

}

outputMessage = `We hope you enjoyed your time here, Mr./Mrs. ${roomBooking.name}. The total amount of money you have to pay is ${totalMoney + totalServiceMoney} BGN. You have used additional room services, costing ${totalServiceMoney} BGN.`;

}

this.capacity++;

this.rooms[roomBooking.type]++;

this.currentBookingNumber--;

this.bookings.splice(index, 1);

return outputMessage;

}

report(){

if(this.bookings.length > 0){

let outputMessage = `${this.name.toUpperCase()} DATABASE:\n` + ('-').repeat(20) + '\n';

let reportBookingsData = [];

for(let roomBooking of this.bookings){

let oneBookingData;

if(!roomBooking.hasOwnProperty('services')){

oneBookingData = `bookingNumber - ${roomBooking.number}\nclientName - ${roomBooking.name}\nroomType - ${roomBooking.type}\nnights - ${roomBooking.nights}`;

}

else {

oneBookingData = `bookingNumber - ${roomBooking.number}\nclientName - ${roomBooking.name}\nroomType - ${roomBooking.type}\nnights - ${roomBooking.nights}\nservices: ${roomBooking.services.join(', ')}`;

}

reportBookingsData.push(oneBookingData);

}

outputMessage += reportBookingsData.join(`\n${('-').repeat(10)}\n`);

return outputMessage;

}

else {

return `${this.name.toUpperCase()} DATABASE:\n` + ('-').repeat(20) + '\nThere are currently no bookings.';

}

}

}

//in Judge must be paste without this below

// let hotel = new Hotel('HotUni', 10);

// console.log(hotel.rentARoom('Peter', 'single', 4));

// console.log(hotel.rentARoom('Robert', 'double', 4));

// console.log(hotel.rentARoom('Geroge', 'maisonette', 6));

// let hotel = new Hotel('HotUni', 10);

// hotel.rentARoom('Peter', 'single', 4);

// hotel.rentARoom('Robert', 'double', 4);

// hotel.rentARoom('Geroge', 'maisonette', 6);

// console.log(hotel.roomService(3, 'housekeeping'));

// console.log(hotel.roomService(3, 'drink'));

// console.log(hotel.roomService(2, 'room'));

// let hotel = new Hotel('HotUni', 10);

// hotel.rentARoom('Peter', 'single', 4);

// hotel.rentARoom('Robert', 'double', 4);

// hotel.rentARoom('Geroge', 'maisonette', 6);

// hotel.roomService(3, 'housekeeping');

// hotel.roomService(3, 'drink');

// hotel.roomService(2, 'room');

// console.log(hotel.report());

class Hotel {

constructor(name, capacity) {

this.name = name;

this.capacity = capacity;

this.bookings = [];

this.currentBookingNumber = 1;

this.roomsPricing = {

single: 50,

double: 90,

maisonette: 135

};

this.servicesPricing = {

food: 10,

drink: 15,

housekeeping: 25

};

if (this.capacity % 2 !== 0) {

this.capacity -= 1;

}

this.avaivables = {

single: Math.floor(this.capacity \* 0.5),

double: Math.floor(this.capacity \* 0.3),

maisonette: Math.floor(this.capacity \* 0.2)

};

}

rentARoom(clientName, roomType, nights) {

if (this.avaivables[roomType] <= 0) {

let output = [];

output.push(`No ${roomType} rooms available!`);

let keys = Object.keys(this.avaivables).filter(x => this.avaivables[x] > 0);

for (const room of keys) {

output.push(`Available ${room} rooms: ${this.avaivables[room]}.`);

}

return output.join(' ');

}

let output = `Enjoy your time here Mr./Mrs. ${clientName}. Your booking is ${this.currentBookingNumber}.`;

let obj = {

clientName,

roomType,

nights,

currentBookingNumber: this.currentBookingNumber

}

this.bookings.push(obj);

this.currentBookingNumber += 1;

this.avaivables[roomType] -= 1;

return output;

}

roomService(currentBookingNumber, serviceType) {

let currentRoom = this.bookings.filter(x => x.currentBookingNumber === currentBookingNumber);

if (currentRoom.length === 0) {

return `The booking ${currentBookingNumber} is invalid.`;

}

if (!this.servicesPricing.hasOwnProperty(serviceType)) {

return `We do not offer ${serviceType} service.`;

}

if (!currentRoom[0].hasOwnProperty('services')) {

currentRoom[0]['services'] = [];

}

currentRoom[0]['services'].push(serviceType);

return `Mr./Mrs. ${currentRoom[0]['clientName']}, Your order for ${serviceType} service has been successful.`;

}

checkOut(currentBookingNumber) {

let currentRoom = this.bookings.filter(x => x.currentBookingNumber === currentBookingNumber)[0];

if (!currentRoom) {

return `The booking ${currentBookingNumber} is invalid.`;

}

let totalMoney = 0;

let roomType = currentRoom['roomType'];

this.avaivables[roomType] += 1;

this.bookings = this.bookings.filter(x => x.currentBookingNumber !== currentBookingNumber);

totalMoney = this.roomsPricing[roomType] \* currentRoom['nights'];

if (!currentRoom['services']) {

return `We hope you enjoyed your time here, Mr./Mrs. ${currentRoom['clientName']}. The total amount of money you have to pay is ${totalMoney} BGN.`

}

let totalServiceMoney = 0;

for (const service of currentRoom['services']) {

totalServiceMoney += this.servicesPricing[service];

}

return `We hope you enjoyed your time here, Mr./Mrs. ${currentRoom['clientName']}. The total amount of money you have to pay is ${totalMoney + totalServiceMoney} BGN. You have used additional room services, costing ${totalServiceMoney} BGN.`;

}

report() {

let output = [];

output.push(`${this.name.toUpperCase()} DATABASE:`);

output.push(`--------------------`);

if (this.bookings.length === 0) {

output.push(`There are currently no bookings.`);

return output.join('\n');

}

let midOutput = [];

for (const room of this.bookings) {

let curr = [];

curr.push(`bookingNumber - ${room['currentBookingNumber']}`);

curr.push(`clientName - ${room['clientName']}`);

curr.push(`roomType - ${room['roomType']}`);

curr.push(`nights - ${room['nights']}`);

if (room['services']) {

let server = [];

for (const serv of room['services']) {

server.push(`${serv}`);

}

curr.push('services: ' + server.join(', '));

}

midOutput.push(curr.join('\n'));

}

output.push(midOutput.join(`\n${'-'.repeat(10)}\n`))

return output.join('\n').trim();

}

}

class Hotel {

constructor(name, capacity) {

this.name = name;

this.capacity = capacity;

this.bookings = [];

this.currentBookingNumber = 1;

this.roomsPricing = {

single: 50,

double: 90,

maisonette: 135

};

this.servicesPricing = {

food: 10,

drink: 15,

housekeeping: 25

};

if (this.capacity % 2 !== 0) {

this.capacity -= 1;

}

this.avaivables = {

single: Math.floor(this.capacity \* 0.5),

double: Math.floor(this.capacity \* 0.3),

maisonette: Math.floor(this.capacity \* 0.2)

};

}

rentARoom(clientName, roomType, nights) {

if (this.avaivables[roomType] <= 0) {

let output = [];

output.push(`No ${roomType} rooms available!`);

let keys = Object.keys(this.avaivables).filter(x => this.avaivables[x] > 0);

for (const room of keys) {

output.push(`Available ${room} rooms: ${this.avaivables[room]}.`);

}

return output.join(' ');

}

let output = `Enjoy your time here Mr./Mrs. ${clientName}. Your booking is ${this.currentBookingNumber}.`;

let obj = {

clientName,

roomType,

nights,

currentBookingNumber: this.currentBookingNumber

}

this.bookings.push(obj);

this.currentBookingNumber += 1;

this.avaivables[roomType] -= 1;

return output;

}

roomService(currentBookingNumber, serviceType) {

let currentRoom = this.bookings.filter(x => x.currentBookingNumber === currentBookingNumber);

if (currentRoom.length === 0) {

return `The booking ${currentBookingNumber} is invalid.`;

}

if (!this.servicesPricing.hasOwnProperty(serviceType)) {

return `We do not offer ${serviceType} service.`;

}

if (!currentRoom[0].hasOwnProperty('services')) {

currentRoom[0]['services'] = [];

}

currentRoom[0]['services'].push(serviceType);

return `Mr./Mrs. ${currentRoom[0]['clientName']}, Your order for ${serviceType} service has been successful.`;

}

checkOut(currentBookingNumber) {

let currentRoom = this.bookings.filter(x => x.currentBookingNumber === currentBookingNumber)[0];

if (!currentRoom) {

return `The booking ${currentBookingNumber} is invalid.`;

}

let totalMoney = 0;

let roomType = currentRoom['roomType'];

this.avaivables[roomType] += 1;

this.bookings = this.bookings.filter(x => x.currentBookingNumber !== currentBookingNumber);

totalMoney = this.roomsPricing[roomType] \* currentRoom['nights'];

if (!currentRoom['services']) {

return `We hope you enjoyed your time here, Mr./Mrs. ${currentRoom['clientName']}. The total amount of money you have to pay is ${totalMoney} BGN.`

}

let totalServiceMoney = 0;

for (const service of currentRoom['services']) {

totalServiceMoney += this.servicesPricing[service];

}

return `We hope you enjoyed your time here, Mr./Mrs. ${currentRoom['clientName']}. The total amount of money you have to pay is ${totalMoney + totalServiceMoney} BGN. You have used additional room services, costing ${totalServiceMoney} BGN.`;

}

report() {

let output = [];

output.push(`${this.name.toUpperCase()} DATABASE:`);

output.push(`--------------------`);

if (this.bookings.length === 0) {

output.push(`There are currently no bookings.`);

return output.join('\n');

}

let midOutput = [];

for (const room of this.bookings) {

let curr = [];

curr.push(`bookingNumber - ${room['currentBookingNumber']}`);

curr.push(`clientName - ${room['clientName']}`);

curr.push(`roomType - ${room['roomType']}`);

curr.push(`nights - ${room['nights']}`);

if (room['services']) {

let server = [];

for (const serv of room['services']) {

server.push(`${serv}`);

}

curr.push('services: ' + server.join(', '));

}

midOutput.push(curr.join('\n'));

}

output.push(midOutput.join('\n----------\n'))

return output.join('\n').trim();

}

}

class Hotel {

constructor(name, capacity) {

this.name = name;

this.capacity = capacity % 2 == 0 ? capacity : capacity - 1;

this.bookings = [];

this.currentBookingNumber = 1;

this.single = Math.floor(this.capacity \* 0.5);

this.double = Math.floor(this.capacity \* 0.3);

this.maisonette = Math.floor(this.capacity \* 0.2);

}

get roomsPricing() {

return {

single: 50,

double: 90,

maisonette: 135

};

}

get servicesPricing() {

return {

food: 10,

drink: 15,

housekeeping: 25

};

}

rentARoom(client, room, nights) {

if (nights < 1) {

return;

}

let error = 'No ' + room + ' rooms available!';

switch (room) {

case 'single':

if (this.single <= 0) {

if (this.double > 0) {

error += ' Available double rooms: ' + this.double + '.';

}

if (this.maisonette > 0) {

error += ' Available maisonette rooms: ' + this.maisonette + '.';

}

return error;

}

break;

case 'double':

if (this.double <= 0) {

if (this.single > 0) {

error += ' Available single rooms: ' + this.single + '.';

}

if (this.maisonette > 0) {

error += ' Available maisonette rooms: ' + this.maisonette + '.';

}

return error;

}

break;

case 'maisonette':

if (this.maisonette <= 0) {

if (this.single > 0) {

error += ' Available single rooms: ' + this.single + '.';

}

if (this.double > 0) {

error += ' Available double rooms: ' + this.double + '.';

}

return error;

}

break;

default:

return;

}

var booking = {

clientName: client,

roomType: room,

nights: nights,

roomNumber: this.currentBookingNumber,

services: []

};

//this.currentBookingNumber++;

switch (room) {

case 'single':

this.single--;

break;

case 'double':

this.double--;

break;

case 'maisonette':

this.maisonette--;

break;

default:

break;

}

this.bookings.push(booking);

return 'Enjoy your time here Mr./Mrs. ' + client + '. Your booking is ' + this.currentBookingNumber++ + '.';

}

roomService(bookingNumber, service) {

if (bookingNumber >= this.currentBookingNumber || bookingNumber < 1) {

return 'The booking ' + bookingNumber + ' is invalid.';

}

if (service !== 'food' && service !== 'drink' && service !== 'housekeeping') {

return 'We do not offer ' + service + ' service.';

}

var booking = this.bookings[bookingNumber - 1];

booking.services.push(service);

return 'Mr./Mrs. ' + booking['clientName'] + ', Your order for ' + service + ' service has been successful.';

}

checkOut(bookingNumber) {

var roomsPricing = {

single: 50,

double: 90,

maisonette: 135

};

var servicesPricing = {

food: 10,

drink: 15,

housekeeping: 25

};

if (bookingNumber >= this.currentBookingNumber || bookingNumber < 1) {

return 'The booking ' + bookingNumber + ' is invalid.';

}

var booking = this.bookings[bookingNumber - 1];

switch (booking['roomType']) {

case 'single':

this.single++;

break;

case 'double':

this.double++;

break;

case 'maisonette':

this.maisonette++;

break;

default:

break;

}

this.bookings.splice(bookingNumber - 1, 1);

if (booking.services.length > 0) {

let serviceMoney = 0;

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

serviceMoney += servicesPricing[booking.services[i]];

}

let money = (roomsPricing[booking.roomType] \* booking.nights) + serviceMoney;

return 'We hope you enjoyed your time here, Mr./Mrs. ' + booking['clientName'] + '. The total amount of money you have to pay is ' + money + ' BGN. You have used additional room services, costing ' + serviceMoney + ' BGN.';

} else {

let money = roomsPricing[booking.roomType] \* booking.nights;

return 'We hope you enjoyed your time here, Mr./Mrs. ' + booking['clientName'] + '. The total amount of money you have to pay is ' + money + ' BGN.';

}

}

report() {

if (this.bookings.length <= 0) {

return this.name.toUpperCase() + ' DATABASE:\n--------------------\nThere are currently no bookings.';

}

var str = this.name.toUpperCase() + ' DATABASE:\n--------------------\n';

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

str += 'bookingNumber - ' + this.bookings[i].roomNumber + '\nclientName - ' + this.bookings[i].clientName + '\nroomType - ' + this.bookings[i].roomType + '\nnights - ' + this.bookings[i].nights;

if (this.bookings[i].services.length > 0) {

str += '\nservices: ' + this.bookings[i].services.join(', ');

}

if (i < this.bookings.length - 1) {

str += '\n----------\n';

}

}

return str;

}

}

class Hotel {

constructor(name, capacity) {

this.name = name;

this.capacity = capacity;

this.bookings = [];

this.currentBookingNumber = 1;

this.rooms = {

single: Math.floor(this.capacity / 2),

double: Math.floor(this.capacity \* 0.3),

maisonette: Math.floor(this.capacity \* 0.2),

};

}

get roomsPricing() {

return {

single: 50,

double: 90,

maisonette: 135,

};

}

get servicesPricing() {

return {

food: 10,

drink: 15,

housekeeping: 25,

};

}

rentARoom(name, type, nights) {

if (this.rooms[type] <= 0) {

let str = `No ${type} rooms available!`;

Object.keys(this.rooms).forEach(room => {

if (room !== type && this.rooms[room] > 0) {

str += ` Available ${room} rooms: ${this.rooms[room]}.`;

}

});

return str;

}

this.bookings.push({

name,

type,

nights,

number: this.currentBookingNumber,

});

this.currentBookingNumber++;

this.rooms[type]--;

return `Enjoy your time here Mr./Mrs. ${name}. Your booking is ${this.currentBookingNumber - 1}.`;

}

roomService(number, type) {

const providedServices = this.servicesPricing;

let client = null;

for (const booking of this.bookings) {

if (booking.number === number) {

client = booking;

break;

}

}

if (client === null) {

return `The booking ${number} is invalid.`;

}

if (providedServices[type] === undefined) {

return `We do not offer ${type} service.`;

}

if (client.services === undefined) {

client.services = [];

}

client.services.push(type);

return `Mr./Mrs. ${client.name}, Your order for ${type} service has been successful.`;

}

checkOut(number) {

const providedServices = this.servicesPricing;

const roomPrices = this.roomsPricing;

let index = 0;

let client = null;

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

if (this.bookings[i].number === number) {

client = this.bookings[i];

index = i;

break;

}

}

if (client === null) {

return `The booking ${number} is invalid.`;

}

const totalMoney = roomPrices[client.type] \* client.nights;

let totalServiceMoney;

if (client.services !== undefined) {

totalServiceMoney = client.services.reduce((acc, curr) => {

return acc += providedServices[curr];

}, 0);

// client.services.forEach(service => {

// totalServiceMoney += providedServices[service];

// });

}

this.rooms[client.type]++;

this.currentBookingNumber--;

this.bookings.splice(index, 1);

if (client.services === undefined) {

return `We hope you enjoyed your time here, Mr./Mrs. ${client.name}. The total amount of money you have to pay is ${totalMoney} BGN.`;

}

return `We hope you enjoyed your time here, Mr./Mrs. ${client.name}. The total amount of money you have to pay is ${totalMoney + totalServiceMoney} BGN. You have used additional room services, costing ${totalServiceMoney} BGN.`;

}

report() {

let str = this.name.toUpperCase() + ' DATABASE:\n';

str += '-'.repeat(20) + '\n';

if (this.bookings.length === 0) {

str += 'There are currently no bookings.';

return str;

}

this.bookings.forEach((client, i) => {

str += `bookingNumber - ${client.number}\n`;

str += `clientName - ${client.name}\n`;

str += `roomType - ${client.type}\n`;

str += `nights - ${client.nights}\n`;

if (client.services !== undefined) {

str += `services: ${client.services.join(', ')}\n`;

}

if (i !== this.bookings.length - 1) {

str += '-'.repeat(10) + '\n';

}

});

return str.trim();

}

}

const hotel = new Hotel('HotUni', 10);

hotel.rentARoom('Peter', 'single', 4);

hotel.rentARoom('Robert', 'double', 4);

hotel.rentARoom('Geroge', 'maisonette', 6);

console.log(hotel.roomService(3, 'housekeeping'));

console.log(hotel.roomService(3, 'drink'));

console.log(hotel.roomService(2, 'room'));

console.log(hotel.checkOut(3));

## Problem 4. Heroes

function solve() {

let buttonRebuildAKingdom = document.getElementsByTagName('button')[0];

let buttonJoin = document.getElementsByTagName('button')[1];

let buttonAttack = document.getElementsByTagName('button')[2];

buttonRebuildAKingdom.addEventListener('click', rebuildAKingdom);

buttonJoin.addEventListener('click', joinKingdom);

buttonAttack.addEventListener('click', war);

function rebuildAKingdom() {

let inputKingdom = buttonRebuildAKingdom.parentElement.children[0];

let inputKing = buttonRebuildAKingdom.parentElement.children[1];

let kingdom = inputKingdom.value;

let king = inputKing.value;

if (!king instanceof String || king.length < 2) {

inputKing.value = '';

return;

}

let kingdoms = ['castle', 'dungeon', 'fortress', 'inferno', 'necropolis', 'rampart', 'stronghold', 'tower', 'conflux'];

if (!kingdoms.includes(kingdom.toLowerCase())) {

inputKingdom.value = '';

return;

}

let h1 = document.createElement('h1');

h1.innerHTML = kingdom.toUpperCase();

let divCastle = document.createElement('div');

divCastle.classList.add(kingdom.toLowerCase());

let h2 = document.createElement('h2');

h2.innerHTML = king.toUpperCase();

let fieldset = document.createElement('fieldset');

let legend = document.createElement('legend');

legend.innerHTML = 'Army';

let pTanks = document.createElement('p');

pTanks.innerHTML = 'TANKS - 0';

let pFighters = document.createElement('p');

pFighters.innerHTML = 'FIGHTERS - 0';

let pMages = document.createElement('p');

pMages.innerHTML = 'MAGES - 0';

let divArmyOutput = document.createElement('div');

divArmyOutput.classList.add('armyOutput');

fieldset.appendChild(legend);

fieldset.appendChild(pTanks);

fieldset.appendChild(pFighters);

fieldset.appendChild(pMages);

fieldset.appendChild(divArmyOutput);

if (document.getElementById(kingdom.toLowerCase()).style.display == 'none') {

document.getElementById(kingdom.toLowerCase()).appendChild(h1);

document.getElementById(kingdom.toLowerCase()).appendChild(divCastle);

document.getElementById(kingdom.toLowerCase()).appendChild(h2);

document.getElementById(kingdom.toLowerCase()).appendChild(fieldset);

document.getElementById(kingdom.toLowerCase()).style.display = 'inline-block';

}

}

function joinKingdom() {

if (!Array.from(document.getElementsByName('characterType')).some(rb => rb.checked)) {

return;

}

let inputCharacter = buttonJoin.parentElement.children[0];

let inputKingdom = buttonJoin.parentElement.children[1];

let character = inputCharacter.value;

let kingdom = inputKingdom.value;

if (!character instanceof String || character.length < 2) {

inputCharacter.value = '';

return;

}

let kingdoms = ['castle', 'dungeon', 'fortress', 'inferno', 'necropolis', 'rampart', 'stronghold', 'tower', 'conflux'];

if (!kingdoms.includes(kingdom.toLowerCase())) {

inputKingdom.value = '';

return;

}

if (document.getElementById(kingdom.toLowerCase()).style.display === 'none') {

inputKingdom.value = '';

return;

}

document.getElementById(kingdom.toLowerCase()).children[3].children[4].innerHTML += character + ' ';

let checkedRadioButton = Array.from(document.getElementsByName('characterType')).filter(rb => rb.checked)[0].value;

switch (checkedRadioButton) {

case 'tank':

let tankText = document.getElementById(kingdom.toLowerCase()).children[3].children[1].innerHTML;

let tankCount = tankText.split(' - ')[1];

document.getElementById(kingdom.toLowerCase()).children[3].children[1].innerHTML = 'TANKS - ' + (Number(tankCount) + 1);

break;

case 'fighter':

let fighterText = document.getElementById(kingdom.toLowerCase()).children[3].children[2].innerHTML;

let fighterCount = fighterText.split(' - ')[1];

document.getElementById(kingdom.toLowerCase()).children[3].children[2].innerHTML = 'FIGHTERS - ' + (Number(fighterCount) + 1);

break;

case 'mage':

let mageText = document.getElementById(kingdom.toLowerCase()).children[3].children[3].innerHTML;

let mageCount = mageText.split(' - ')[1];

document.getElementById(kingdom.toLowerCase()).children[3].children[3].innerHTML = 'MAGES - ' + (Number(mageCount) + 1);

break;

default:

break;

}

}

function war() {

let inputAttacker = buttonAttack.parentElement.children[1];

let inputDefender = buttonAttack.parentElement.children[2];

let attackerName = inputAttacker.value;

let defenderName = inputDefender.value;

let kingdoms = ['castle', 'dungeon', 'fortress', 'inferno', 'necropolis', 'rampart', 'stronghold', 'tower', 'conflux'];

if (!kingdoms.includes(attackerName.toLowerCase()) || !kingdoms.includes(defenderName.toLowerCase())) {

inputAttacker.value = '';

inputDefender.value = '';

return;

}

if (document.getElementById(attackerName.toLowerCase()).style.display === 'none' || document.getElementById(defenderName.toLowerCase()).style.display === 'none'){

inputAttacker.value = '';

inputDefender.value = '';

return;

}

let attacker = document.getElementById(attackerName.toLowerCase());

let defender = document.getElementById(defenderName.toLowerCase());

let attackerPoints = 0;

attackerPoints += Number(attacker.children[3].children[1].innerHTML.split(' - ')[1]) \* 20;

attackerPoints += Number(attacker.children[3].children[2].innerHTML.split(' - ')[1]) \* 50;

attackerPoints += Number(attacker.children[3].children[3].innerHTML.split(' - ')[1]) \* 70;

let defenderPoints = 0;

defenderPoints += Number(defender.children[3].children[1].innerHTML.split(' - ')[1]) \* 80;

defenderPoints += Number(defender.children[3].children[2].innerHTML.split(' - ')[1]) \* 50;

defenderPoints += Number(defender.children[3].children[3].innerHTML.split(' - ')[1]) \* 30;

if (attackerPoints > defenderPoints) {

defender.children[2].innerHTML = attacker.children[2].innerHTML;

}

}

}

//in Judje must be paste without this below

solve();

function solve() {

var btn1 = document.getElementsByTagName('button')[0];

var btn2 = document.getElementsByTagName('button')[1];

var btn3 = document.getElementsByTagName('button')[2];

//console.log(btn1);

btn1.addEventListener('click', function () {

var king = btn1.parentElement.children[1];

var kingdom = btn1.parentElement.children[0];

if (king.value.length >= 2) {

if (kingdom.value.toUpperCase() == 'CASTLE' ||

kingdom.value.toUpperCase() == 'DUNGEON' ||

kingdom.value.toUpperCase() == 'FORTRESS' ||

kingdom.value.toUpperCase() == 'INFERNO' ||

kingdom.value.toUpperCase() == 'NECROPOLIS' ||

kingdom.value.toUpperCase() == 'RAMPART' ||

kingdom.value.toUpperCase() == 'STRONGHOLD' ||

kingdom.value.toUpperCase() == 'TOWER' ||

kingdom.value.toUpperCase() == 'CONFLUX') {

var h1 = document.createElement('h1');

h1.innerHTML = kingdom.value.toUpperCase();

var div = document.createElement('div');

div.classList.add(kingdom.value.toLowerCase());

var h2 = document.createElement('h2');

h2.innerHTML = king.value.toUpperCase();

var fieldset = document.createElement('fieldset');

var legend = document.createElement('legend');

legend.innerHTML = 'Army';

var p1 = document.createElement('p');

var p2 = document.createElement('p');

var p3 = document.createElement('p');

p1.innerHTML = 'TANKS - 0';

p2.innerHTML = 'FIGHTERS - 0';

p3.innerHTML = 'MAGES - 0';

var divF = document.createElement('div');

divF.classList.add('armyOutput');

fieldset.appendChild(legend);

fieldset.appendChild(p1);

fieldset.appendChild(p2);

fieldset.appendChild(p3);

fieldset.appendChild(divF);

if (document.getElementById(kingdom.value.toLowerCase()).style.display == 'none') {

document.getElementById(kingdom.value.toLowerCase()).appendChild(h1);

document.getElementById(kingdom.value.toLowerCase()).appendChild(div);

document.getElementById(kingdom.value.toLowerCase()).appendChild(h2);

document.getElementById(kingdom.value.toLowerCase()).appendChild(fieldset);

document.getElementById(kingdom.value.toLowerCase()).style.display = 'inline-block';

}

} else {

kingdom.value = '';

}

} else {

king.value = '';

}

});

btn2.addEventListener('click', function () {

//console.log(Array.from(document.getElementsByName('characterType')).filter(x => x.checked)[0].value);

if (Array.from(document.getElementsByName('characterType')).some(x => x.checked)) {

if (btn2.parentElement.children[0].value.length >= 2) {

if (btn2.parentElement.children[1].value.toUpperCase() == 'CASTLE' ||

btn2.parentElement.children[1].value.toUpperCase() == 'DUNGEON' ||

btn2.parentElement.children[1].value.toUpperCase() == 'FORTRESS' ||

btn2.parentElement.children[1].value.toUpperCase() == 'INFERNO' ||

btn2.parentElement.children[1].value.toUpperCase() == 'NECROPOLIS' ||

btn2.parentElement.children[1].value.toUpperCase() == 'RAMPART' ||

btn2.parentElement.children[1].value.toUpperCase() == 'STRONGHOLD' ||

btn2.parentElement.children[1].value.toUpperCase() == 'TOWER' ||

btn2.parentElement.children[1].value.toUpperCase() == 'CONFLUX') {

if (document.getElementById(btn2.parentElement.children[1].value.toLowerCase()).style.display !== 'none') {

document.getElementById(btn2.parentElement.children[1].value.toLowerCase()).children[3].children[4].innerHTML += btn2.parentElement.children[0].value + ' ';

switch (Array.from(document.getElementsByName('characterType')).filter(x => x.checked)[0].value) {

case 'fighter':

var t = document.getElementById(btn2.parentElement.children[1].value.toLowerCase()).children[3].children[2].innerHTML;

var n = t.split(' - ')[1];

document.getElementById(btn2.parentElement.children[1].value.toLowerCase()).children[3].children[2].innerHTML = 'FIGHTERS - ' + (Number(n) + 1);

break;

case 'mage':

var t1 = document.getElementById(btn2.parentElement.children[1].value.toLowerCase()).children[3].children[3].innerHTML;

var n1 = t1.split(' - ')[1];

document.getElementById(btn2.parentElement.children[1].value.toLowerCase()).children[3].children[3].innerHTML = 'MAGES - ' + (Number(n1) + 1);

break;

case 'tank':

var t2 = document.getElementById(btn2.parentElement.children[1].value.toLowerCase()).children[3].children[1].innerHTML;

var n2 = t2.split(' - ')[1];

document.getElementById(btn2.parentElement.children[1].value.toLowerCase()).children[3].children[1].innerHTML = 'TANKS - ' + (Number(n2) + 1);

break;

default:

break;

}

} else {

btn2.parentElement.children[1].value = '';

}

} else {

btn2.parentElement.children[1].value = '';

}

} else {

btn2.parentElement.children[0].value = '';

}

}

});

btn3.addEventListener('click', function () {

//console.log(document.getElementById(btn3.parentElement.children[1].value.toLowerCase()).children[3].children[3].innerHTML.split(' - ')[1]);

if (btn3.parentElement.children[1].value.toUpperCase() == 'CASTLE' ||

btn3.parentElement.children[1].value.toUpperCase() == 'DUNGEON' ||

btn3.parentElement.children[1].value.toUpperCase() == 'FORTRESS' ||

btn3.parentElement.children[1].value.toUpperCase() == 'INFERNO' ||

btn3.parentElement.children[1].value.toUpperCase() == 'NECROPOLIS' ||

btn3.parentElement.children[1].value.toUpperCase() == 'RAMPART' ||

btn3.parentElement.children[1].value.toUpperCase() == 'STRONGHOLD' ||

btn3.parentElement.children[1].value.toUpperCase() == 'TOWER' ||

btn3.parentElement.children[1].value.toUpperCase() == 'CONFLUX') {

if (btn3.parentElement.children[2].value.toUpperCase() == 'CASTLE' ||

btn3.parentElement.children[2].value.toUpperCase() == 'DUNGEON' ||

btn3.parentElement.children[2].value.toUpperCase() == 'FORTRESS' ||

btn3.parentElement.children[2].value.toUpperCase() == 'INFERNO' ||

btn3.parentElement.children[2].value.toUpperCase() == 'NECROPOLIS' ||

btn3.parentElement.children[2].value.toUpperCase() == 'RAMPART' ||

btn3.parentElement.children[2].value.toUpperCase() == 'STRONGHOLD' ||

btn3.parentElement.children[2].value.toUpperCase() == 'TOWER' ||

btn3.parentElement.children[2].value.toUpperCase() == 'CONFLUX') {

if (document.getElementById(btn3.parentElement.children[1].value.toLowerCase()).style.display !== 'none') {

if (document.getElementById(btn3.parentElement.children[2].value.toLowerCase()).style.display !== 'none') {

var attacker = document.getElementById(btn3.parentElement.children[1].value.toLowerCase());

var defender = document.getElementById(btn3.parentElement.children[2].value.toLowerCase());

var aA = Number(attacker.children[3].children[3].innerHTML.split(' - ')[1]) \* 70;

aA += Number(attacker.children[3].children[2].innerHTML.split(' - ')[1]) \* 50;

aA += Number(attacker.children[3].children[1].innerHTML.split(' - ')[1]) \* 20;

var aD = Number(attacker.children[3].children[3].innerHTML.split(' - ')[1]) \* 30;

aD += Number(attacker.children[3].children[2].innerHTML.split(' - ')[1]) \* 50;

aD += Number(attacker.children[3].children[1].innerHTML.split(' - ')[1]) \* 80;

var dA = Number(defender.children[3].children[3].innerHTML.split(' - ')[1]) \* 70;

dA += Number(defender.children[3].children[2].innerHTML.split(' - ')[1]) \* 50;

dA += Number(defender.children[3].children[1].innerHTML.split(' - ')[1]) \* 20;

var dD = Number(defender.children[3].children[3].innerHTML.split(' - ')[1]) \* 30;

dD += Number(defender.children[3].children[2].innerHTML.split(' - ')[1]) \* 50;

dD += Number(defender.children[3].children[1].innerHTML.split(' - ')[1]) \* 80;

if (aA > dD) {

if (defender.children[2].innerHTML !== attacker.children[2].innerHTML) {

defender.children[2].innerHTML = attacker.children[2].innerHTML;

//defender.children[3].children[4].innerHTML = '';

//defender.children[3].children[3].innerHTML = 'MAGES - 0';

//defender.children[3].children[2].innerHTML = 'FIGHTERS - 0';

//defender.children[3].children[1].innerHTML = 'TANKS - 0';

}

}

} else {

btn3.parentElement.children[2].value = '';

btn3.parentElement.children[1].value = '';

}

} else {

btn3.parentElement.children[1].value = '';

btn3.parentElement.children[2].value = '';

}

} else {

btn3.parentElement.children[1].value = '';

btn3.parentElement.children[2].value = '';

}

} else {

btn3.parentElement.children[1].value = '';

btn3.parentElement.children[2].value = '';

}

});

}

|  |  |
| --- | --- |
| //solve(); |  |
|  |  |