# Solutions-JS Advanced - Retake Exam: 19.12.2018

# Problem 1. X-mas Gifts

//For Judge:

function solution() {

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

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

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

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

if ($toyType.val() && Number($toyPrice.val()) && $toyDescription.val()) {

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

$giftDiv.addClass('gift');

let $img = $('<img src="gift.png"/>');

let $h2 = $('<h2>');

$h2.text($toyType.val());

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

$p.text($toyDescription.val());

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

$button.text(`Buy it for $${$toyPrice.val()}`);

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

$giftDiv.append($img, $h2, $p, $button);

$christmasGiftShop.append($giftDiv);

}

$toyType.val('');

$toyPrice.val('');

$toyDescription.val('');

}

//For local testing:

function solution() {

    let addButton = $('button')[0];//in Judge must be paste without this line

    addButton.addEventListener('click', addToShop);//in Judge must be paste without this line

    function addToShop() {//in Judge must be paste without this line

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

        let $toyType = $('#toyType');

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

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

        if ($toyType.val() && Number($toyPrice.val()) && $toyDescription.val()) {

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

            $giftDiv.addClass('gift');

            let $img = $('<img src="gift.png"/>');

            let $h2 = $('<h2>');

            $h2.text($toyType.val());

            let $p = $('<p>');

            $p.text($toyDescription.val());

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

            $button.text(`Buy it for $${$toyPrice.val()}`);

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

            $giftDiv.append($img, $h2, $p, $button);

            $christmasGiftShop.append($giftDiv);

        }

        $toyType.val('');

        $toyPrice.val('');

        $toyDescription.val('');

    }//in Judge must be paste without this line

}

## Problem 2. Warehouse (Unit Testing)

//Class for testing:

class Warehouse {

get capacity() {

return this.\_capacity;

}

set capacity(givenSpace) {

if (typeof givenSpace === 'number' && givenSpace > 0) {

return this.\_capacity = givenSpace;

} else {

throw `Invalid given warehouse space`;

}

}

constructor(capacity) {

this.capacity = capacity;

this.availableProducts = {'Food': {}, 'Drink': {}};

}

addProduct(type, product, quantity) {

let addedQuantity = ((this.capacity - this.occupiedCapacity()) - quantity);

let output;

if (addedQuantity >= 0) {

if (this.availableProducts[type].hasOwnProperty(product) === false) {

this.availableProducts[type][product] = 0;

}

this.availableProducts[type][product] += quantity;

output = this.availableProducts[type];

} else {

throw `There is not enough space or the warehouse is already full`;

}

return output;

}

orderProducts(type) {

let output;

let sortedKeys = Object.keys(this.availableProducts[type])

.sort((a, b) => this.availableProducts[type][b] - this.availableProducts[type][a]);

let newObj = {};

for (let product of sortedKeys) {

if (newObj.hasOwnProperty(product) === false) {

newObj[product] = 0;

}

newObj[product] += this.availableProducts[type][product];

}

this.availableProducts[type] = newObj;

output = this.availableProducts[type];

return output;

}

occupiedCapacity() {

let output = 0;

let productsCount = Object.keys(this.availableProducts['Food']).length +

Object.keys(this.availableProducts['Drink']).length;

if (productsCount > 0) {

let quantityInStock = 0;

for (let type of Object.keys(this.availableProducts)) {

for (let product of Object.keys(this.availableProducts[type])) {

quantityInStock += this.availableProducts[type][product];

}

}

output = quantityInStock;

}

return output;

}

revision() {

let output = "";

if (this.occupiedCapacity() > 0) {

for (let type of Object.keys(this.availableProducts)) {

output += `Product type - [${type}]\n`;

for (let product of Object.keys(this.availableProducts[type])) {

output += `- ${product} ${this.availableProducts[type][product]}\n`;

}

}

} else {

output = 'The warehouse is empty';

}

return output.trim();

}

scrapeAProduct(product, quantity) {

let type = Object.keys(this.availableProducts).find(t => Object.keys(this.availableProducts[t]).includes(product));

let output;

if (type !== undefined) {

if (quantity <= this.availableProducts[type][product]) {

this.availableProducts[type][product] -= quantity;

} else {

this.availableProducts[type][product] = 0;

}

output = this.availableProducts[type];

} else {

throw `${product} do not exists`;

}

return output;

}

}

module.exports = Warehouse;

//Tests:

// 100/100 in Judge but the second zero test didn't past - local all is OK

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

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

//In Judge must be paste without this above

describe('Warehouse', function () {

describe('constructor', function () {

it('should have correct capacity', function () {

let warehouse = new Warehouse(10);

expect(warehouse.capacity).to.be.equal(10);

});

it('should throw on negative capacity', function () {

expect(() => { new Warehouse(-10); }).to.throw('Invalid given warehouse space');

});

it('should throw on 0 capacity', function () {

expect(() => { new Warehouse(0); }).to.throw('Invalid given warehouse space');

});

it('should throw on string instead number for capacity', function () {

expect(() => { new Warehouse('10'); }).to.throw('Invalid given warehouse space');

});

});

describe('addProduct', function () {

it('should add product correctly', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 9);

expect(Object.keys(warehouse.availableProducts['Food']).length).to.be.equal(1);

});

it('should add products correctly', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 3);

warehouse.addProduct('Food', 'orange', 3);

warehouse.addProduct('Food', 'apple', 4);

expect(Object.keys(warehouse.availableProducts['Food']).length).to.be.equal(3);

});

it('should throw on not enough capacity for one product', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 11);

}).to.throw('There is not enough space or the warehouse is already full');

});

it('should throw on not enough capacity for more products', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 4);

warehouse.addProduct('Food', 'apple', 4);

warehouse.addProduct('Food', 'orange', 3);

}).to.throw('There is not enough space or the warehouse is already full');

});

});

describe('orderProducts', function () {

it('should order products for food with correct values', function () {

let warehouse = new Warehouse(1000);

warehouse.addProduct('Drink', 'tea', 10);

warehouse.addProduct('Food', 'bannana', 10);

warehouse.addProduct('Food', 'apple', 15);

warehouse.addProduct('Food', 'bannana', 30);

warehouse.addProduct('Drink', 'coffee', 15);

warehouse.addProduct('Food', 'orange', 10);

warehouse.addProduct('Drink', 'milk', 30);

let food = warehouse.orderProducts('Food');

let foodProducts = Object.keys(food);

let expectedFood = {

'bannana': 40,

'apple': 15,

'orange': 10

};

let expectedFoodProducts = Object.keys(expectedFood);

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

expect(foodProducts[i]).to.be.equal(expectedFoodProducts[i]);

}

});

it('should order products for drink with correct values', function () {

let warehouse = new Warehouse(1000);

warehouse.addProduct('Drink', 'tea', 10);

warehouse.addProduct('Food', 'bannana', 10);

warehouse.addProduct('Food', 'apple', 15);

warehouse.addProduct('Food', 'bannana', 30);

warehouse.addProduct('Drink', 'coffee', 15);

warehouse.addProduct('Food', 'orange', 10);

warehouse.addProduct('Drink', 'milk', 30);

let drink = warehouse.orderProducts('Drink');

let drinkProducts = Object.keys(drink);

let expectedDrink = {

'milk': 30,

'coffee': 15,

'tea': 10

};

let expectedDrinkProducts = Object.keys(expectedDrink);

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

expect(drinkProducts[i]).to.be.equal(expectedDrinkProducts[i]);

}

});

});

describe('occupiedCapacity', function () {

it('should return 0 capacity for empty space', function () {

let warehouse = new Warehouse(10);

expect(warehouse.occupiedCapacity()).to.be.equal(0);

});

it('should return correct capacity for full space with food', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with food and drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

});

describe('scrapeAProduct', function () {

it('should throw for non existing type', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.scrapeAProduct('orange', 1)

}).to.throw('orange do not exists');

});

it('should give correct result for less than available quantity', function(){

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

expect(JSON.stringify(warehouse.scrapeAProduct('bannana', 5))).to.be.equal('{"bannana":1,"apple":3}');

});

it('should give 0 for more than available quantity', function(){

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

expect(JSON.stringify(warehouse.scrapeAProduct('bannana', 7))).to.be.equal('{"bannana":0,"apple":3}');

});

});

describe('revision', function () {

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

let warehouse = new Warehouse(10);

expect(warehouse.revision()).to.be.equal('The warehouse is empty');

});

it('should return correct value for warehouse with food', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- apple 3\n- orange 1\nProduct type - [Drink]';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

let expectedString = 'Product type - [Food]\nProduct type - [Drink]\n- milk 6\n- coffee 3\n- tea 1';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with food and drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- orange 1\nProduct type - [Drink]\n- tea 3';

expect(warehouse.revision()).to.be.equal(expectedString);

});

});

describe('constructor', function () {

it('should have correct capacity', function () {

let warehouse = new Warehouse(10);

expect(warehouse.capacity).to.be.equal(10);

});

it('should throw on negative capacity', function () {

expect(() => { new Warehouse(-10); }).to.throw('Invalid given warehouse space');

});

it('should throw on 0 capacity', function () {

expect(() => { new Warehouse(0); }).to.throw('Invalid given warehouse space');

});

it('should throw on string instead number for capacity', function () {

expect(() => { new Warehouse('10'); }).to.throw('Invalid given warehouse space');

});

});

describe('addProduct', function () {

it('should add product correctly', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 9);

expect(Object.keys(warehouse.availableProducts['Food']).length).to.be.equal(1);

});

it('should add products correctly', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 3);

warehouse.addProduct('Food', 'orange', 3);

warehouse.addProduct('Food', 'apple', 4);

expect(Object.keys(warehouse.availableProducts['Food']).length).to.be.equal(3);

});

it('should throw on not enough capacity for one product', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 11);

}).to.throw('There is not enough space or the warehouse is already full');

});

it('should throw on not enough capacity for more products', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 4);

warehouse.addProduct('Food', 'apple', 4);

warehouse.addProduct('Food', 'orange', 3);

}).to.throw('There is not enough space or the warehouse is already full');

});

});

describe('orderProducts', function () {

it('should order products for food with correct values', function () {

let warehouse = new Warehouse(1000);

warehouse.addProduct('Drink', 'tea', 10);

warehouse.addProduct('Food', 'bannana', 10);

warehouse.addProduct('Food', 'apple', 15);

warehouse.addProduct('Food', 'bannana', 30);

warehouse.addProduct('Drink', 'coffee', 15);

warehouse.addProduct('Food', 'orange', 10);

warehouse.addProduct('Drink', 'milk', 30);

let food = warehouse.orderProducts('Food');

let foodProducts = Object.keys(food);

let expectedFood = {

'bannana': 40,

'apple': 15,

'orange': 10

};

let expectedFoodProducts = Object.keys(expectedFood);

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

expect(foodProducts[i]).to.be.equal(expectedFoodProducts[i]);

}

});

});

describe('occupiedCapacity', function () {

let warehouse;

beforeEach(function(){

warehouse = new Warehouse(10);

});

it('should return 0 capacity for empty space', function () {

expect(warehouse.occupiedCapacity()).to.be.equal(0);

});

it('should return correct capacity for full space with food', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with drink', function () {

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with food and drink', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

});

describe('scrapeAProduct', function () {

let warehouse;

beforeEach(function() {

warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

});

it('should throw for non existing type', function () {

expect(() => {

warehouse.scrapeAProduct('orange', 1)

}).to.throw('orange do not exists');

});

it('should give correct result for less than available quantity', function(){

expect(JSON.stringify(warehouse.scrapeAProduct('banana', 3))).to.be.equal('{"banana":1,"apple":1}');

});

it('should give 0 for more than available quantity', function(){

expect(JSON.stringify(warehouse.scrapeAProduct('banana', 5))).to.be.equal('{"banana":0,"apple":1}');

});

});

describe('revision', function () {

let warehouse;

beforeEach(function(){

warehouse = new Warehouse(10);

});

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

expect(warehouse.revision()).to.be.equal('The warehouse is empty');

});

it('should return correct value for warehouse with food', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- apple 3\n- orange 1\nProduct type - [Drink]';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with drink', function () {

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

let expectedString = 'Product type - [Food]\nProduct type - [Drink]\n- milk 6\n- coffee 3\n- tea 1';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with food and drink', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- orange 1\nProduct type - [Drink]\n- tea 3';

expect(warehouse.revision()).to.be.equal(expectedString);

});

});

});

## Problem 3. Kitchen

class Kitchen{

constructor(budget){

this.budget = budget;

this.menu = {};

this.productsInStock = {};

this.actionsHistory = [];

}

loadProducts(products){

for(let product of products){

let [productName, productQuantity, productPrice] = product.split(' ');

productQuantity = Number(productQuantity);

productPrice = Number(productPrice);

if(this.budget >= productPrice){

if(!this.productsInStock.hasOwnProperty(productName)){

this.productsInStock[productName] = 0;

}

this.productsInStock[productName] += productQuantity;

this.budget -= productPrice;

this.actionsHistory.push(`Successfully loaded ${productQuantity} ${productName}`);

}

else {

this.actionsHistory.push(`There was not enough money to load ${productQuantity} ${productName}`);

}

}

//return this.actionsHistory.join('\n').trim() + '\n';

//return this.actionsHistory.join('\n') + '\n';

return this.actionsHistory.join('\n');

}

addToMenu(meal, neededProducts, price){

if(!this.menu.hasOwnProperty(meal)){

this.menu[meal] = {meal, products: neededProducts, price};

let mealsCount = Object.keys(this.menu).length;

return `Great idea! Now with the ${meal} we have ${mealsCount} meals in the menu, other ideas?`;

}

else {

//return `The ${meal} is already in our menu, try something different.`;//For Exercise in Judge

return `${meal} is already in our menu, try something different.`;//For Exam in Judge

}

}

showTheMenu(){

if(Object.keys(this.menu).length > 0){

let menuList = Object.values(this.menu).map(m => `${m.meal} - $ ${m.price}`);

//return menuList.join('\n').trim() + '\n';

return menuList.join('\n') + '\n';

}

else {

return 'Our menu is not ready yet, please come later...';

}

}

makeTheOrder(meal){

if(!this.menu.hasOwnProperty(meal)){

return `There is not ${meal} yet in our menu, do you want to order something else?`

}

let neededProducts = this.menu[meal].products;

for(let neededProduct of neededProducts){

let [productName, productQuantity] = neededProduct.split(/\s+/);

productQuantity = Number(productQuantity);

if(!this.productsInStock.hasOwnProperty(productName) || this.productsInStock[productName] < productQuantity){

return `For the time being, we cannot complete your order (${meal}), we are very sorry...`;

}

}

neededProducts.forEach(neededProduct => {

let [productName, productQuantity] = neededProduct.split(/\s+/);

productQuantity = Number(productQuantity);

this.productsInStock[productName] -= productQuantity;

});

let price = this.menu[meal].price;

this.budget += price;

return `Your order (${meal}) will be completed in the next 30 minutes and will cost you ${price}.`

}

}

let kitchen = new Kitchen (1000);

console.log(kitchen.loadProducts(['Banana 10 5', 'Banana 20 10', 'Strawberries 50 30', 'Yogurt 10 10', 'Yogurt 500 1500', 'Honey 5 50']));

console.log(kitchen.addToMenu('frozenYogurt', ['Yogurt 1', 'Honey 1', 'Banana 1', 'Strawberries 10'], 9.99));

console.log(kitchen.addToMenu('Pizza', ['Flour 0.5', 'Oil 0.2', 'Yeast 0.5', 'Salt 0.1', 'Sugar 0.1', 'Tomato sauce 0.5', 'Pepperoni 1', 'Cheese 1.5'], 15.55));

console.log(kitchen.showTheMenu());

# Problem 4. Darts

function dart() {

const maxScore = 100;

// const colorMapping = {

// firstLayer: 'Green',

// secondLayer: 'Yellow',

// thirdLayer: 'Orange',

// fourthLayer: 'Red',

// fifthLayer: 'Purple',

// sixsthLayer: 'Blue'

// };

const colorMapping = {

firstLayer: 0,

secondLayer: 1,

thirdLayer: 2,

fourthLayer: 3,

fifthLayer: 4,

sixthLayer: 5

};

$('#playBoard').on('click', 'div', onPlayBoardClick);

function onPlayBoardClick(event) {

event.stopPropagation();

        let points = getScore(event.target.id);

selectPlayer(points);

}

function getScore(id) {

return +$('#scoreBoard')

.find('tbody tr')

.eq(colorMapping[id])

.children()

.eq(1)

.text()

.split(' ')[0];

}

let isHome = true;

function selectPlayer(score) {

let selector = '';

isHome ? selector = '#Home' : selector = '#Away';

let $pointsElement = $(selector).children().eq(0);

$pointsElement.text((i, t) => Number(t) + score);

if(isHome){

$('#turns').children().eq(0).text('Turn on Away');

$('#turns').children().eq(1).text('Next is Home');

}

else{

$('#turns').children().eq(0).text('Turn on Home');

$('#turns').children().eq(1).text('Next is Away');

}

let currentPoints = Number($pointsElement.text());

if (currentPoints >= maxScore){

if(isHome){

$('#Home').children().eq(1).css({

backgroundColor: 'green'

});

$('#Away').children().eq(1).css({

backgroundColor: 'red'

});

}

else {

$('#Away').children().eq(1).css({

backgroundColor: 'green'

});

$('#Home').children().eq(1).css({

backgroundColor: 'red'

});

}

$('#playBoard').off('click');

}

isHome = !isHome;

}

}