# JS Advanced: Exam Preparation

Problems for exam preparation for the [“JavaScript Advanced” course @ SoftUni](https://softuni.bg/courses/javascript-advanced). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/355/>.

## Add / Remove Towns (Simple DOM Interaction)

You are given the following **HTML code**:

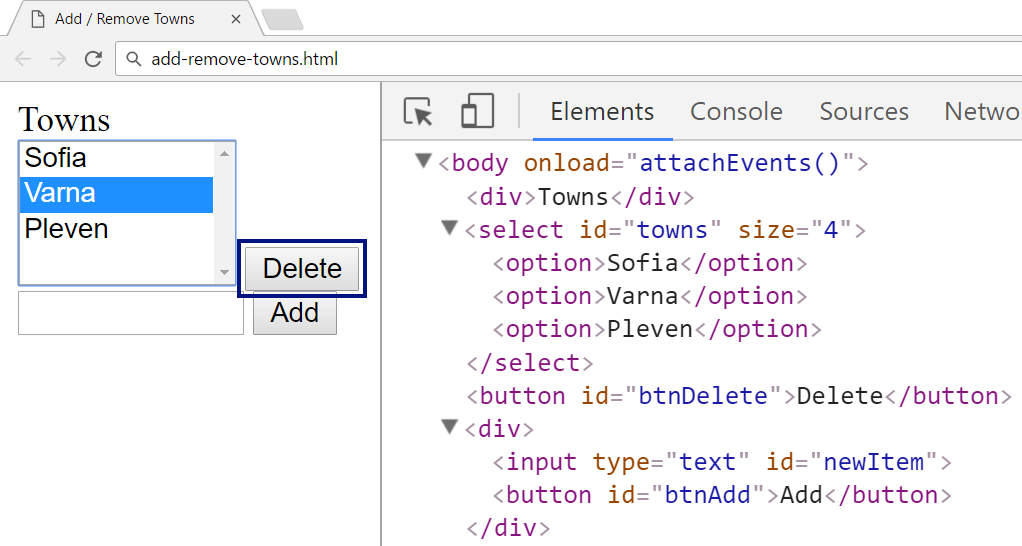
|  |
| --- |
| add-remove-towns.html |
| *<!-- JS Advanced Exam @ SoftUni - 13-Nov-2016 -->* <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Add / Remove Towns</**title**>  <**style**>**select**, **input** { **width**: 100**px** }</**style**>  <**script src="https://code.jquery.com/jquery-3.1.1.min.js"**></**script**> </**head**>  <**body onload="***attachEvents*()**"**>  <**div**>Towns</**div**> <**select id="towns" size="4"**>  <**option**>Sofia</**option**>  <**option**>Varna</**option**>  <**option**>Pleven</**option**> </**select**>  <**button id="btnDelete"**>Delete</**button**>  <**div**>  <**input type="text" id="newItem"** />  <**button id="btnAdd"**>Add</**button**> </**div**>  <**script**>  **function** *attachEvents*() {  *//* ***TODO:*** …  } </**script**> </**body**> </**html**> |

### Your Task

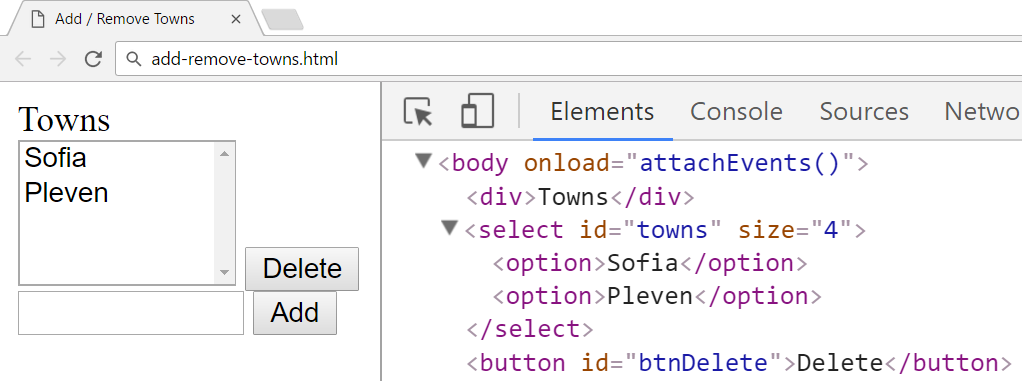
**Write the** **missing JavaScript code** to make the **[Add]** and **[Delete]** buttons work as expected.

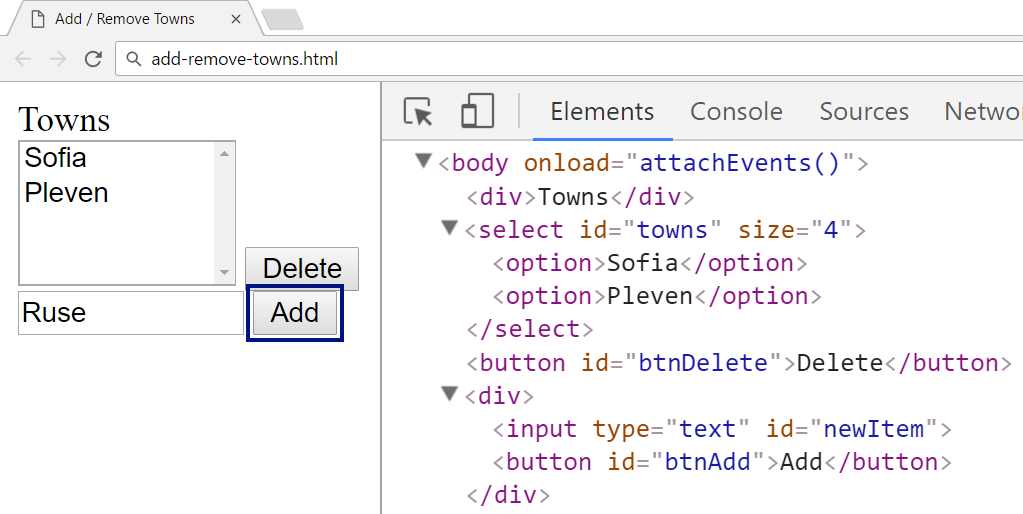
* The **[Add]** button should append the text from the text box as a new item at the end of the list box and clear the input text box after that. If the textbox is empty, it should do nothing.
* The **[Delete]** button should **delete selected item** from the list. If no item is selected, it should do nothing.
  + When an item is selected it has the property selected.

### Examples

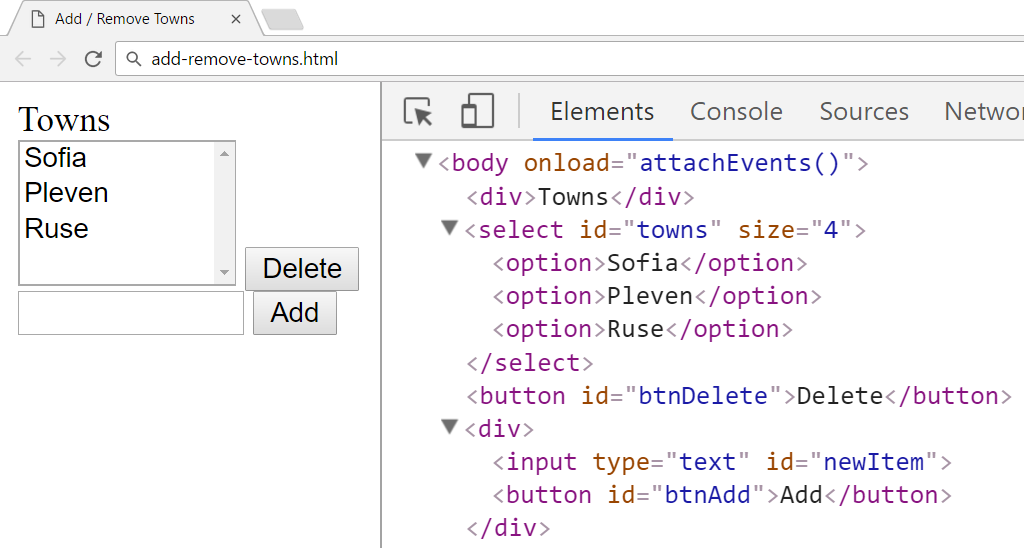


↓





↓



### Submission

Submit your attachEvents function as “**JavaScript code (DOM unit tests)**”.

## Add / Swap / Shift Left / Right in List (Unit Testing)

You are given the following **JavaScript code**:

|  |
| --- |
| list-add-swap-shift-left-right.js |
| **function** *createList*() {  **let** data = [];  **return** {  add: **function** (item) {  data.push(item)  },  shiftLeft: **function** () {  **if** (data.**length** > 1) {  **let** first = data.shift();  data.push(first);  }  },  shiftRight: **function** () {  **if** (data.**length** > 1) {  **let** last = data.pop();  data.unshift(last);  }  },  swap: **function** (index1, index2) {  **if** (!***Number***.isInteger(index1) || index1 < 0 || index1 >= data.**length** ||  !***Number***.isInteger(index2) || index2 < 0 || index2 >= data.**length** ||  index1 === index2) {  **return false**;  }  **let** temp = data[index1];  data[index1] = data[index2];  data[index2] = temp;  **return true**;  },  toString: **function** () {  **return** data.join(**", "**);  }  }; } |

### Functionality

The above code creates a **list** data structure that holds items (of any type). It supports the following operations:

* add(item) – **appends** given item to the end of the list.
* shiftLeft() – shifts all elements **one position left** and the first elements comes last (with **rotation**).
* shiftRight() – shifts all elements **one position right** and the last elements comes first (with **rotation**).
* swap(index1, index2) – swaps the items at the specified indexes and returns **true**. If any of the two indexes **does not exist** or they are **equal** the collection stays **unchanged** and the method returns **false**.
* toString() – returns the string representations of the **list items**, separated by “, “.

### Examples

This is an example how this code is **intended to be used**:

|  |  |  |
| --- | --- | --- |
| Sample code usage |  | Corresponding output |
| **let *list*** = *createList*(); ***list***.add(1); ***list***.add(**"two"**); ***list***.add(3); **console**.log(**`list = [**${***list***}**]`**); ***list***.shiftLeft(); **console**.log(**"shifted left <--"**); **console**.log(**`list = [**${***list***}**]`**); ***list***.add([**"four"**]); **console**.log(**`list = [**${***list***}**]`**); ***list***.shiftRight(); **console**.log(**"shifted right -->"**); **console**.log(**`list = [**${***list***}**]`**); **console**.log(**`Swaping [0] and [3]:** ${***list***.swap(0,3)}**`**); **console**.log(**`list = [**${***list***}**]`**); **console**.log(**`Swaping [1] and [1]:** ${***list***.swap(1,1)}**`**); **console**.log(**`list = [**${***list***}**]`**); | list = [1, two, 3]  shifted left <--  list = [two, 3, 1]  list = [two, 3, 1, four]  shifted right -->  list = [four, two, 3, 1]  Swaping [0] and [3]: true  list = [1, two, 3, four]  Swaping [1] and [1]: false  list = [1, two, 3, four] |

### Your Task

Using **Mocha** and **Chai** write **JS unit tests** to test the entire functionality of the list object. Your code will only be provided the createList function, how you test the list is entirely up to you - whether you create a new list before each test or share the same list between tests.

You should have at least **6 test cases**, make sure you cover all **edge cases**. You may use the following code as a template:

|  |
| --- |
| describe(**"TODO …"**, **function**() {  it(**"TODO …"**, **function**() {  *//* ***TODO: …*** });  *//* ***TODO: …*** }); |

### Submission

Submit **only your tests** as “**JavaScript code (Unit Tests with Sinon and Mocha)**”.

## Mail Box (Simple Class)

Write a **JavaScript class** MailBox to hold a list of mail messages (subject + text).

|  |
| --- |
| **class** MailBox {  *//* ***TODO: implement this class*** } |

Each **mail message** holds **subject** and **text**. Implement the following features:

* **Constructor** – creates an empty mail box.
* Method addMessage(subject, text) – adds a mail message to the mail box. Both **subject** and **text** are strings. Returns the mailbox itself to allow chaining.
* Accessor property messageCount – returns thetotal number of messages in the mail box.
* Method deleteAllMessages() – clears the mail box (deletes all messages).
* Method findBySubject(substr) – returns all mail messages from the mail box that hold the specified substr string in their subject. Order the results by their order of adding to the mail box. Return the results as array of objects {subject, text}. If no messages are matched, return an empty array.
* Method toString() – returns the text representation of the mail box in the following format:
  + Empty mail box:

|  |
| --- |
| \* (empty mailbox) |

* + Non-empty mail box:

|  |
| --- |
| \* [subject1] Text1  \* [subject2] Text2  \* [subject3] Text3  … |

### Examples

This is an example how the MailBox class is **intended to be used**:

|  |
| --- |
| Sample code usage |
| **let *mb*** = **new *MailBox***(); **console**.log(**"Msg count: "** + ***mb***.messageCount); **console**.log(**'Messages:\n'** + ***mb***); ***mb***.addMessage(**"meeting"**, **"Let's meet at 17/11"**); ***mb***.addMessage(**"beer"**, **"Wanna drink beer tomorrow?"**); ***mb***.addMessage(**"question"**, **"How to solve this problem?"**); ***mb***.addMessage(**"Sofia next week"**, **"I am in Sofia next week."**); **console**.log(**"Msg count: "** + ***mb***.messageCount); **console**.log(**'Messages:\n'** + ***mb***); **console**.log(**"Messages holding 'rakiya': "** +  **JSON**.stringify(***mb***.findBySubject(**'rakiya'**))); **console**.log(**"Messages holding 'ee': "** +  **JSON**.stringify(***mb***.findBySubject(**'ee'**)));  ***mb***.deleteAllMessages(); **console**.log(**"Msg count: "** + ***mb***.messageCount); **console**.log(**'Messages:\n'** + ***mb***);  **console**.log(**"New mailbox:\n"** +  **new *MailBox***()  .addMessage(**"Subj 1"**, **"Msg 1"**)  .addMessage(**"Subj 2"**, **"Msg 2"**)  .addMessage(**"Subj 3"**, **"Msg 3"**)  .toString()); |

|  |
| --- |
| Corresponding output |
| Msg count: 0  Messages:  \* (empty mailbox)  Msg count: 4  Messages:  \* [meeting] Let's meet at 17/11  \* [beer] Wanna drink beer tomorrow?  \* [question] How to solve this problem?  \* [Sofia next week] I am in Sofia next week.  Messages holding 'rakiya': []  Messages holding 'ee': [{"subject":"meeting","text":"Let's meet at 17/11"},{"subject":"beer","text":"Wanna drink beer tomorrow?"},{"subject":"Sofia next week","text":"I am in Sofia next week."}]  Msg count: 0  Messages:  \* (empty mailbox)  New mailbox:  \* [Subj 1] Msg 1  \* [Subj 2] Msg 2  \* [Subj 3] Msg 3 |

### Submission

Submit your class MailBox as “**JavaScript code (Mocha unit tests)** ”.

## Cards (Object Interacting with DOM)

You are given the following **HTML code** (with CSS styles), intended to draw a deck of cards:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**title**>Cards</**title**>  <**style**>  **div**.**card** {  **display**: **inline-block**; **padding**: 10**px**; **margin**: 10**px**;  **width**: 50**px**; **height**: 80**px**; **background**: **#EEE**;  **border**: 3**px solid #DDD**; **border-radius**: 5**px**;  **font-size**: 18**pt**; **text-align**: **center**; **line-height**: 75**px**;  }  </**style**>  <**script src="https://code.jquery.com/jquery-3.1.1.min.js"**></**script**> </**head**> <**body**> <**div id="main"**></**div**> <**script**>  **function** *cardDeckBuilder*(selector) {  *//* ***TODO: return the card builder***  } </**script**> <**script**>  $(**function**() {  let builder = *cardDeckBuilder*(**"#main"**);  builder.addCard(**"10"**, **"D"**);  builder.addCard(**"K"**, **"S"**);  builder.addCard(**"Q"**, **"H"**);  builder.addCard(**"4"**, **"C"**);  }); </**script**> </**body**> </**html**> |

The function cardDeckBuilder(selector) takes as input a DOM selector (string) and returns an object holding a function addCard(face, suit). It appends the specified card into the DOM as a **DIV** element:



* The card **face** is one of the following: **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **J**, **Q**, **K**, **A**.
* The card **suit** is one of the following: **C** (clubs – ♣), **D** (diamonds – ♦), **H** (hearts – ♥), **S** (spades – ♠).

Implement a “**reverse cards**” functionality: when any of the cards is **clicked**, their order in the DOM should be **reversed** (reordered in the DOM tree in reverse order – from the last to the first).

Your task is to write the body of the cardDeckBuilder(selector) function to get the above page working as expected (see the screenshots). Do not modify the HTML code and CSS styles, just **write the missing JS function**.

### Constraint

* The card faces and suits will always be valid.

### Examples

When the missing JS function cardDeckBuilder(selector) is implemented correctly, the page should look as follows (after the page loading is completed):



When any of the cards (<div class="card">…</div>) is clicked, all the cards should be **reversed**:



### Hints

You may use the following **Unicode characters** for the card suites:

|  |  |  |  |
| --- | --- | --- | --- |
| \u2663 🡪 ♣ | \u2666 🡪 ♦ | \u2665 🡪 ♥ | \u2660 🡪 ♠ |

### Submission

Submit your cardDeckBuilder(selector) function as “**JavaScript code (DOM unit tests)**”.