# 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/353/>.

## Delete Town by Name (Simple DOM Interaction)

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

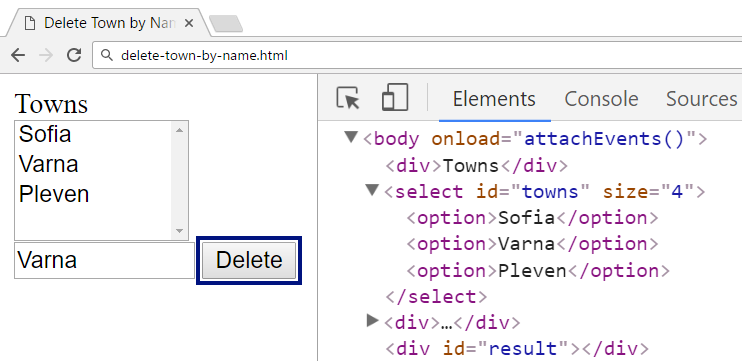
|  |
| --- |
| delete-town-by-name.html |
| <!DOCTYPE **html**> <**html**> <**head**>  <**title**>Delete Town by Name</**title**>  <**style**>**select**, **input** { **width**: 100**px** }</**style**>  <**script src="https://code.jquery.com/jquery-3.1.1.min.js"**></**script**> </**head**> <**body onload="***attachEvents*()**"**> <**div**>Towns</**div**> <**select id="towns" size="4"**>  <**option**>Sofia</**option**>  <**option**>Varna</**option**>  <**option**>Pleven</**option**> </**select**> <**div**>  <**input type="text" id="townName"** />  <**button id="btnDelete"**>Delete</**button**> </**div**> <**div id="result"**></**div**> <**script**>  **function** *attachEvents*() {  *//* ***TODO***  } </**script**> </**body**> </**html**> |

### Your Task

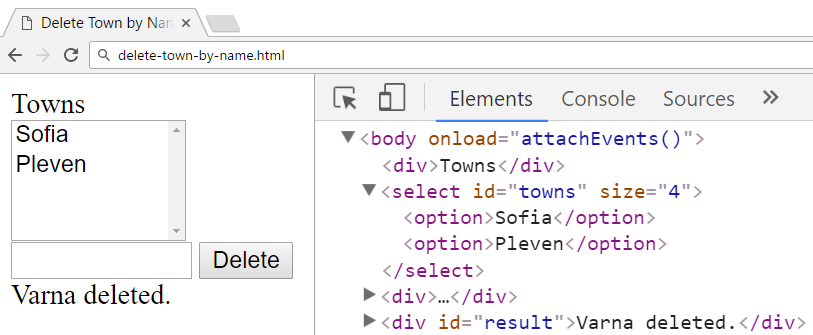
**Write the** **missing JavaScript code** to make the **[Delete]** button work as expected.

* The **[Delete]** button should **delete all towns** from the list **matching** the value in the “townName” text box.
* After a town is successfully deleted, show a message *“<town\_name> deleted.*” in the “result” div.
* When the town is missing, show a message “*<town\_name> not found.*” in the “result” div.
* In all cases clear the value in the “townName” text box.
* Assume there are **no repeating** towns in the list.

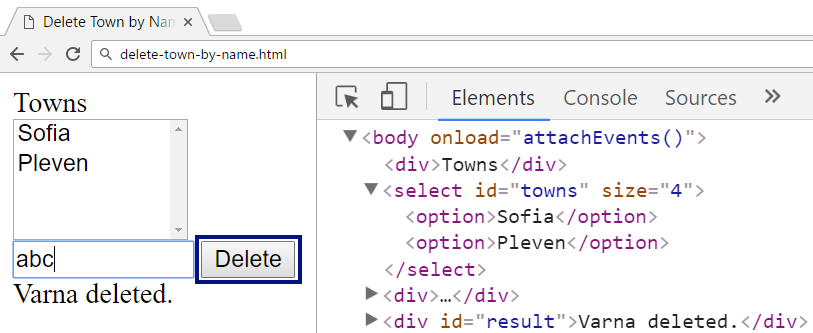
### Examples



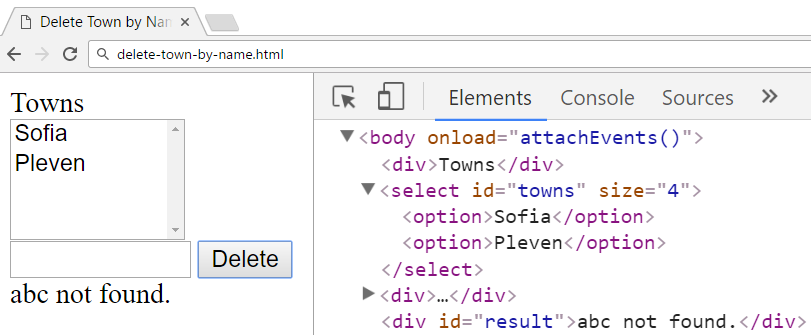
↓



↓



↓



### Submission

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

## Add / Delete in List (Unit Testing)

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

|  |
| --- |
| add-delete-in-list.js |
| **let *list*** = (**function**(){  **let** data = [];  **return** {  add: **function**(item) {  data.push(item);  },  delete: **function**(index) {  **if** (Number.isInteger(index) && index >= 0 && index < data.**length**) {  **return** data.splice(index, 1)[0];  } **else** {  **return undefined**;  }  },  toString: **function**() {  **return** data.join(**", "**);  }  }; })(); |

### Functionality

The above code implements 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.
* delete(index) – **deletes** the item at given index. **Returns** the deleted item. If the index is **invalid**, do not delete anything and returns undefined.
* 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 |
| **console**.log(**`list = [**${***list***}**]`**); ***list***.add(1); **console**.log(**`list = [**${***list***}**]`**); ***list***.add(**"two"**); ***list***.add(3); **console**.log(**`list = [**${***list***}**]`**); **console**.log(**"deleted: "** + ***list***.delete(1)); **console**.log(**`list = [**${***list***}**]`**); **console**.log(**"cannot delete: "** + ***list***.delete(-1)); **console**.log(**`list = [**${***list***}**]`**); | list = []  list = [1]  list = [1, two, 3]  deleted: two  list = [1, 3]  cannot delete: undefined  list = [1, 3] |

### Your Task

Using **Mocha** and **Chai** write **JS unit tests** to test the entire functionality of the list object. You should have at least **7 test 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)**”.

## Training Course (Simple Class)

Write a **JavaScript class** TrainingCourse to hold training course: a course has a **title**, a **trainer** and ordered list of **topics**.

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

Each topic holds **title** and **date**. Topics are **ordered by date** (ascending). Implement the following features:

* **Constructor** – creates a training course by **title** and **trainer** (strings).
* Method addTopic(title, date) – adds a topic to the course. The topic consists of topic **title** (string) and topic **date** (JavaScript Date object). Topics in the course internally should be **ordered by date**.
* Accessor property firstTopic – returns the **earliest** (by date) topic from the course – a **JS object** holding **title** (string) and **date** (JS Date object).
* Accessor property lastTopic – returns the **latest** (by date) topic from the course – a **JS object** holding **title** (string) and **date** (JS Date object).
* Method toString() – returns a string representing the course and its topics in the following format:

|  |
| --- |
| Course "*<title>*" by *<trainer>*  \* *<topic 1>* - <*date 1>*  \* *<topic 2>* - <*date 2>*  \* … |

* Assume there are no topics on the same date.
* Print all **dates** in their **default format** (no additional date / time formatting is required)

### Examples

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

|  |
| --- |
| Sample code usage |
| **let *js*** = **new TrainingCourse**(**"JS Intro"**, **"Svetlin Nakov"**); **console**.log(**"First topic: "** + **JSON**.stringify(***js***.firstTopic)); **console**.log(**"Last topic: "** + **JSON**.stringify(***js***.lastTopic)); **console**.log(**""** + ***js***);  ***js***.addTopic(**"Maps"**, **new** Date(2016, 9, 6, 18, 0)); ***js***.addTopic(**"JS Overview"**, **new** Date(2016, 8, 27, 18, 0)); ***js***.addTopic(**"Program Logic"**, **new** Date(2016, 8, 29, 18, 0)); ***js***.addTopic(**"Arrays"**, **new** Date(2016, 9, 3, 18, 0)); **console**.log(**"First topic: "** + **JSON**.stringify(***js***.firstTopic)); **console**.log(**"Last topic: "** + **JSON**.stringify(***js***.lastTopic)); **console**.log(**""** + ***js***);  **let *php*** = **new** TrainingCourse(**"PHP Intro"**, **"Ivan Yonkov"**)  .addTopic(**"Strings"**, **new** Date(2017, 2, 16, 18, 0))  .addTopic(**"PHP First Steps"**, **new** Date(2017, 2, 12, 18, 0))  .addTopic(**"Arrays"**, **new** Date(2017, 2, 14, 18, 0)); **console**.log(**""** + ***php***); |

|  |
| --- |
| Corresponding output |
| First topic: undefined  Last topic: undefined  Course "JS Intro" by Svetlin Nakov  First topic: {"title":"JS Overview","date":"2016-09-27T15:00:00.000Z"}  Last topic: {"title":"Maps","date":"2016-10-06T15:00:00.000Z"}  Course "JS Intro" by Svetlin Nakov  \* JS Overview - Tue Sep 27 2016 18:00:00 GMT+0300 (FLE Daylight Time)  \* Program Logic - Thu Sep 29 2016 18:00:00 GMT+0300 (FLE Daylight Time)  \* Arrays - Mon Oct 03 2016 18:00:00 GMT+0300 (FLE Daylight Time)  \* Maps - Thu Oct 06 2016 18:00:00 GMT+0300 (FLE Daylight Time)  Course "PHP Intro" by Ivan Yonkov  \* PHP First Steps - Sun Mar 12 2017 18:00:00 GMT+0200 (FLE Standard Time)  \* Arrays - Tue Mar 14 2017 18:00:00 GMT+0200 (FLE Standard Time)  \* Strings - Thu Mar 16 2017 18:00:00 GMT+0200 (FLE Standard Time) |

### Submission

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

## Table Builder (Object Interacting with DOM)

Write a **JavaScript function** to return a **table builder** object. It should take a **DOM selector** as **input** parameter and return as an **output** an object holding two functions: createTable(columnNames) and fillData(dataRows).

|  |
| --- |
| **function** *tableBuilder*(selector) {  *//* ***TODO: return { … }*** } |

The createTable(columnNames) **function** replaces the content in the target DOM element with a **new HTML table** holding a row with table headings given in the columnNames, and an additional column “**Action**”.

The fillData(dataRows) **function** puts in the table inside the target DOM element table rows holding the provided dataRows (array of arrays of strings). Add a **[Delete]** button in the last column. Clicking the **[Delete]** button should **delete** the entire table row.

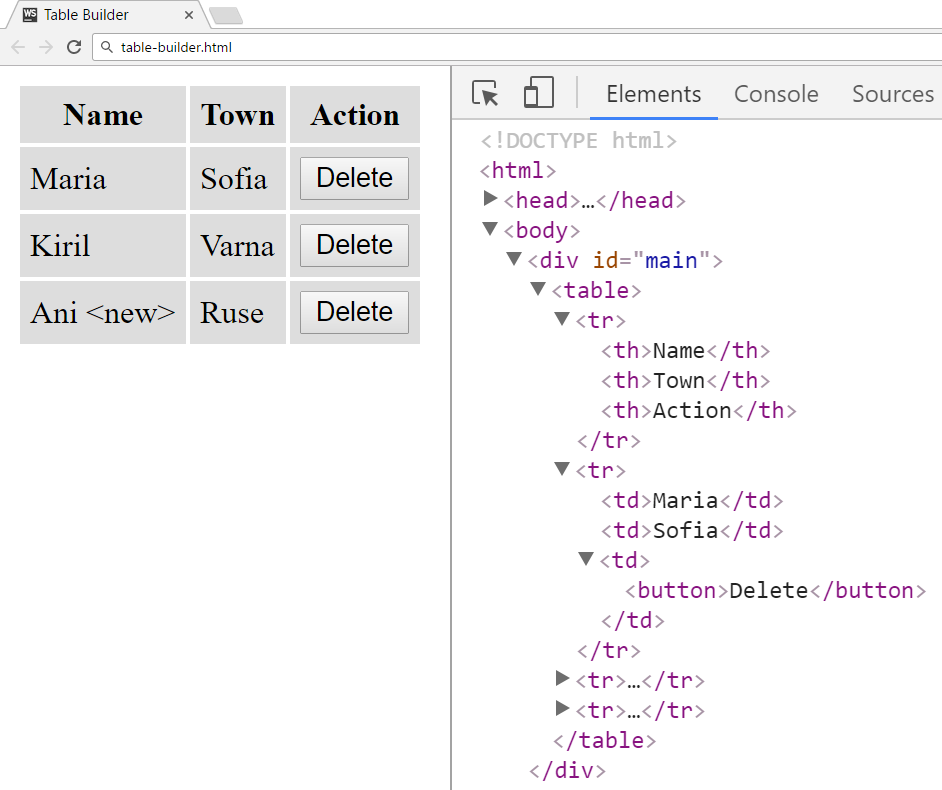
Note that the table column names and table data cells may hold **special characters** that should be displayed as text.

### Examples

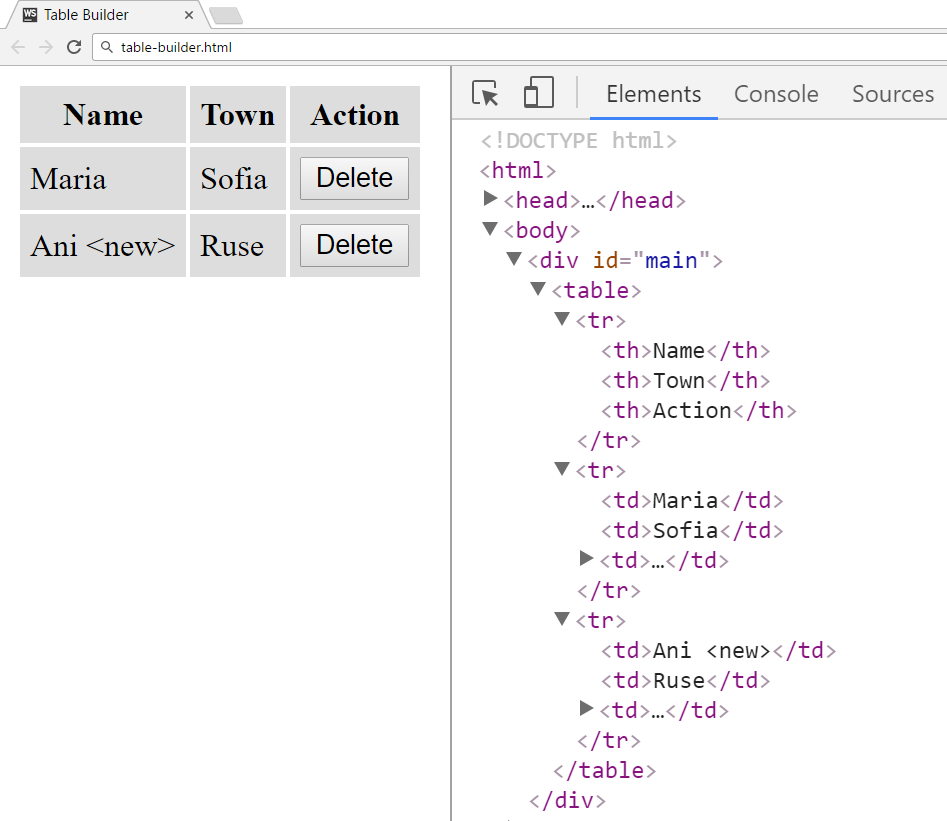
This is an example how the table builder function is **intended to be used**. Assume we have the following **HTML code**:

|  |
| --- |
| table-builder.html |
| <!DOCTYPE **html**> <**html**> <**head**>  <**title**>Table Builder</**title**>  <**style**>**td**,**th** { **background**:**#DDD**; **padding**:5**px** }</**style**>  <**script src="https://code.jquery.com/jquery-3.1.1.min.js"**></**script**> </**head**> <**body**> <**div id="main"**></**div**> <**script**>  **function** *tableBuilder*(selector) {  *//* ***TODO: return { … }*** } </**script**> <**script**>  $(**function**() {  let builder = *tableBuilder*(**"#main"**);  builder.createTable([**'Name'**, **'Town'**]);  builder.fillData([  [**'Maria'**, **'Sofia'**],  [**'Kiril'**, **'Varna'**],  [**'Ani <new>'**, **'Ruse'**],  ]);  }); </**script**> </**body**> </**html**> |

This is how the **HTML** and **DOM** should look like after the above code is loaded in a Web browser:



Clicking the **[Delete]** button at the row “**Kiril / Varna**” should delete this row:



### Submission

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