# 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/352/>.

## Summary

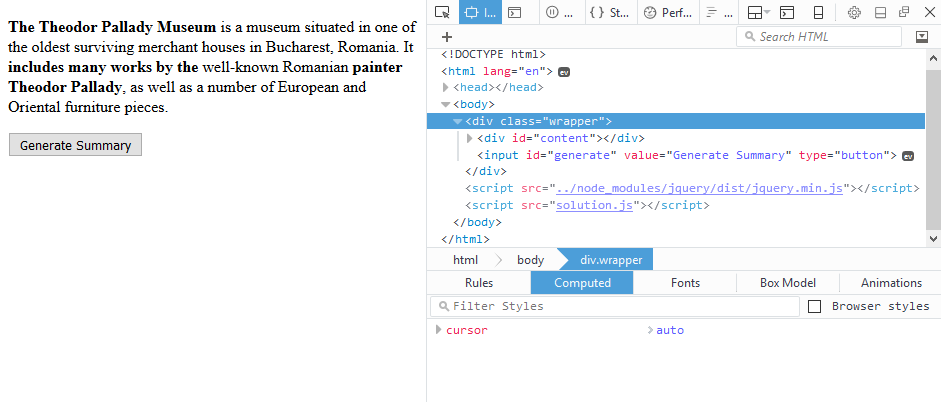
You will be given an HTML document, containing an article and a button. Some parts of the text will be highlighted with a <strong> tag. Write a JS function that receives a **selector** to the **button** and attaches an **event** to it. The event **executes a function** that finds all **highlighted** parts of the text and places them in a **summary after** the original **article**. The article will **always** be inside a div with **ID** **'content'**.

The **summary** should be inside a div with **ID** '**summary'**, appended after the original article (at the end of its parent). The first element inside the div is a heading (<h2>) with the text **'Summary'**. The second element is a paragraph (<p>), inside which are all the extracted parts from the original article. Note that you **only** need the text **without** the <strong> tags that originally surround it. See the example for more details.

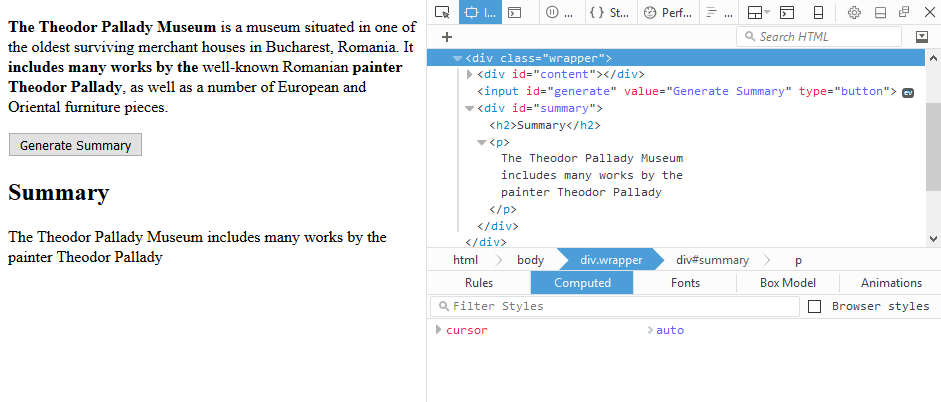
### Examples

|  |
| --- |
| Sample Input |
| <**div class="holder"**>  <**input type="button" id="generate" value="Generate Summary"**/>  <**div id="content"**>  <**p**><**strong**>Important text. </**strong**>Not too important. <**strong**>Also useful info.</**strong**></**p**>  </**div**> </**div**> |
| Sample Output |
| <**div class="holder"**>  <**input type="button" id="generate" value="Generate Summary"**/>  <**div id="content"**>  …  </**div**>  <**div id="summary"**>  <**h2**>Summary</**h2**>  <**p**>Important text. Also useful info.</**p**>  </**div**>  </**div**> |

|  |
| --- |
| Sample Input |
| <**div class="wrapper"**>  <**div id="content"**>  <**p**><**strong**>The Theodor Pallady Museum </**strong**>is a museum situated in one of the oldest surviving merchant houses in Bucharest, Romania. It <**strong**>includes many works by the </**strong**>well-known Romanian <**strong**>painter Theodor Pallady</**strong**>, as well as a number of European and Oriental furniture pieces.</**p**>  </**div**>  <**input type="button" id="generate" value="Generate Summary"**/> </**div**> |
| Sample Output |
| <**div class="wrapper"**>  <**div id="content"**>  …  </**div**>  <**input type="button" id="generate" value="Generate Summary"**/>  <**div id="summary"**>  <**h2**>Summary</**h2**>  <**p**>The Theodor Pallady Museum includes many works by the painter Theodor Pallady</**p**>  </**div**>  </**div**> |



↓



### Note

Submit your code using the **DOM unit tests** Judge strategy.

## Sorted List

Write Mocha tests to verify the functionality of the following JS code:

|  |
| --- |
| sorted-list.js |
| **class** SortedList {  constructor() {  **this**.**list** = [];  }   add(element) {  **this**.**list**.push(element);  **this**.sort();  }   remove(index) {  **this**.vrfyRange(index);  **this**.**list**.splice(index, 1);  }   get(index) {  **this**.vrfyRange(index);  **return this**.**list**[index];  }   vrfyRange(index) {  **if** (**this**.**list**.**length** == 0) **throw new** Error(**"Collection is empty."**);  **if** (index < 0 || index >= **this**.**list**.**length**) **throw new** Error(**"Index was outside the bounds of the collection."**);  }   sort() {  **this**.**list**.sort((a, b) => a - b);  }   **get** size() {  **return this**.**list**.**length**;  } } |

Your tests will be supplied with a **class** named SortedList. It needs to meet the following requirements:

* Maintains a collection of **numeric** elements
* Has an add(element) function, which adds a new element to the collection
* Has a remove(index) function, which removes the element at position index
* Has a get(index) function, which return the value of the element at position index
* Keeps the elements of the collection **sorted** in **ascending** order at **all times**
* Throws an **error** if the functions get() and remove() are supplied with an **invalid index** (negative or outside the collection) or if the collection is **empty**
* Has a **size()** property **getter**, which returns the number of elements inside the collection

### Note

Submit your tests inside a describe() statement, using the **Unit Tests with Sinon and Mocha** Judge strategy.

## Storm Watcher

Write a JS class that represents a meteorological station reading. Each reading has an **id**, **temperature**, **humidity**, **pressure** and **windSpeed** properties which are all numbers. The **ID** is auto assigned and **autoincremented** sequentially for each instance, while the rest of the properties are set trough the **constructor**.

In addition, the class must include a toString() method that **returns** a formatted string with a summary of the information kept inside the record and a **weather status**. The status is either **'Not stormy'** or **'Stormy'**, depending on the readings. For the weather to be **stormy**, **all** of these conditions must be met:

* **temperature** bellow 20
* **pressure** bellow 700 **OR** above 900
* **windSpeed** above 25

For **any other** conditions, the weather is **not stormy**. See the examples for more formatting details. Note each property is on a new line.

### Input / Output

Only valid data will be passed to the constructor. The **output** is expected as a **string**, **returned** by the toString() method of your class.

Depending on how you structure your code, submit just the **class** definition **as is**, **or** wrapped in an **IIFE** that **returns** the **class** definition.

### Examples

|  |  |
| --- | --- |
| Sample Input | Output |
| let record1 = new Record(32, 66, 760, 12);  console.log(record1.toString()); | Reading ID: 0  Temperature: 32\*C  Relative Humidity: 66%  Pressure: 760hpa  Wind Speed: 12m/s  Weather: Not stormy |
| let record2 = new Record(10, 40, 680, 30);  console.log(record2.toString()); | Reading ID: 1  Temperature: 10\*C  Relative Humidity: 40%  Pressure: 680hpa  Wind Speed: 30m/s  Weather: Stormy |

## Title Bar

Write a JS **class** that generates an HTML title bar for a webpage. You will be supplied a title and a list of links for a navigation menu. Compose the markup for the title bar, attach the needed events and when requested, append the element to the document.

A title bar is composed of title text, a button and a menu list of navigation links. Clicking the button toggles the visibility of the menu. See the sample HTML for more details.

The constructor of your class needs to take one string argument – the title of the webpage. The class needs to have the following functions:

* addLink(href, name) – takes two string arguments and adds a **link** to the **menu list**
* appendTo(selector) – takes one string argument, **generates HTML** element and adds it to the passed in **selector**

The HTML shown below has the following elements:

* <span> with class "**title**" which holds the **title** passed in to the constructor
* **<a>** with class "**button**" which **toggles** the visibility of the **menu**; you need to attach a **click event** to this element
* <div> with class "**drawer**" which contains the **navigation links**; it’s display css property needs to be **toggled** between block and none via the button listed above
* <a>’s with class "**menu-link**" for each of the **navigation links**

The generated HTML needs to match this **structure** exactly.

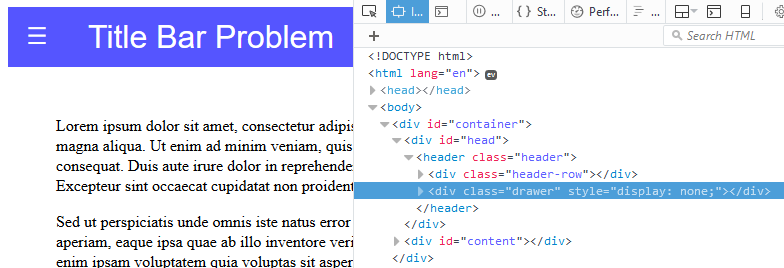
|  |
| --- |
| titlebar.html |
| <**header class="header"**>  <**div class="header-row"**>  <**a class="button"**>**&#9776;**</**a**>  <**span class="title"**>Title</**span**>  </**div**>  <**div class="drawer"**>  <**nav class="menu"**>  <**a class="menu-link" href="href"**>Link name</**a**>  <**a class="menu-link" href="href"**>Link name</**a**>  <**a class="menu-link" href="href"**>Link name</**a**>  <**a class="menu-link" href="href"**>Link name</**a**>  </**nav**>  </**div**> </**header**> |

### Examples

The title bar from the example can be generated using the following code:

|  |
| --- |
| Sample JavaScript |
| **let *header*** = **new** TitleBar(**'Title Bar Problem'**); ***header***.addLink(**'/'**, **'Home'**); ***header***.addLink(**'about'**, **'About'**); ***header***.addLink(**'results'**, **'Results'**); ***header***.addLink(**'faq'**, **'FAQ'**); ***header***.appendTo(**'#container'**); |

Use the provided HTML skeleton as an example and for testing. Note there are no events in the example – the menu will always be visible. jQuery is automatcally available in the Judge, but if you want to use it locally, you’ll have to add it to the HTML yourself.



↓

