**Exercise: DOM Manipulations**

Problems for exercises and homework for the ["JavaScript Advanced" course @ SoftUni](https://softuni.bg/courses/js-advanced)". Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1550/Exercise-DOM-Manipulations>.

## Sections

You will receive an **array** of strings. For each string, create a **div** with a **paragraph** with the **string** in it. Each paragraph is initially **hidden (display:none)**. Add a **click** **event** **listener** to **each div** that **displays** the **hidden** paragraph. Finally, you should **append** all divs to the element with an **id** "**content**".

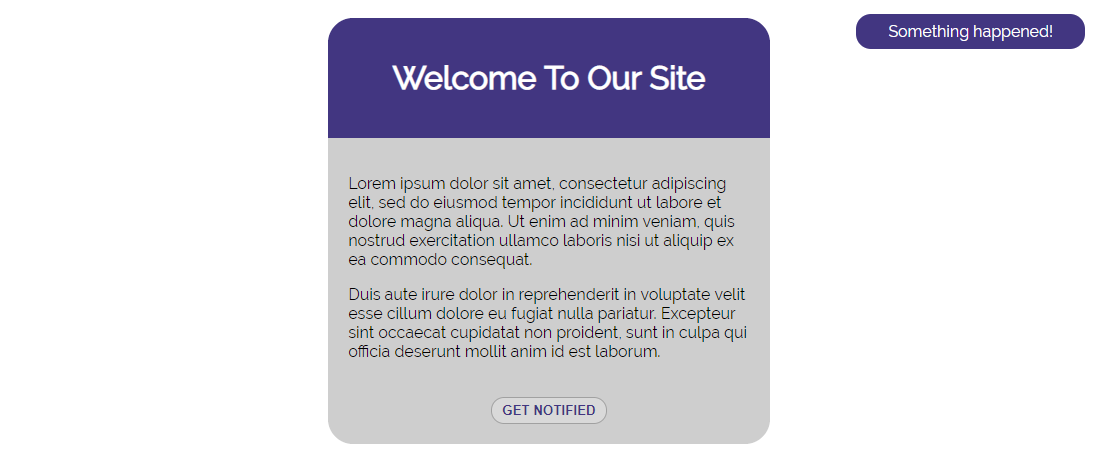
**Example**

 🡪 

## Notification

Write a **function** that receives a string **message** and **displays** it inside a div with an id "**notification**" for 2 seconds. The div is initially **hidden** and when the function is called, it must be **shown**. After 2 seconds, **hide** the div. In the example below, a notification is shown when you **click** the button.

**Example**



When we click the [**GET NOTIFIED**] **button**, a **div** appears in our upper-right corner. It should **disappear** in 2 seconds.

## Time Converter

Create a program that **converts** different time units. Your task is to add a **click** event listener to **all** [**CONVERT**] **buttons**. When a button is **clicked**, read the **corresponding** input field, **convert** the value to the **three other** time units and **display** it in the input fields.

**Example**



One day is equal to 24 hours/1440 minutes/86400 seconds. Whichever button we **click,** the input fields should **change** depending on the added value on the left. (For example, if we write 48 hours and click convert the days, the field value should change to 2).

## Locked Profile

In this problem, you should **create a JS functonality** which **shows** and **hides** the additional

information about users.



When one of the [**Show more**] **buttons** is clicked, the **hiden information** inside the div should

be shown, only if **the profile is not locked**! If the current profile is **locked,** nothing should

happen.



If the **hidden information is displayed** and we **lock** **the profile again**, the [**Hide it**] button

should **not be working**! Otherwise, when the profile is **unlocked** and we click on the **[Hide it]**

button, the new fields must hide again.

## Encode and Decode Messages

In this problem, you should **create a JS functonality** which **encodes and decodes some**

**messages which travel to the network.**



This program should contain **two functionalities**.

The first one is to **encode the given message** and **send it** to the **receiver**.

The second one is to **decode the received message** and **read it (display it)**.

When the [**Encode and send it**] **button** is clicked, you should get the given message from the first textarea. When you get the current message, you should encode it as follows:

* **Change** the **ASCII CODE** on **every single character** in that message when you **add 1** to the current **ASCII NUMBER**, that represent the current character in that message
* **Clear** the **sender** **textarea** and **append** the encoded message to the **receiver textarea**



After clicking [**Encode and send it**] **button** the result should be:



After that, when the [**Decode and read it**] **button** is clicked. You need to get the **encoded message** from **the receiver textarea** and do the **opposite logic** from encoding:

* **Subtract 1** from the current **ASCII NUMBER**, that represents the current character in that message
* Replace the **encoded message** with the already **decoded message** in the receiver textrea, to make it readable



## Table – Search Engine

Write a function that **searches** in a **table** by given input.



When the "**Search**" **button** is **clicked**, go through all cells in the table except for the first row (Student name, Student email and Student course) and check if the given input has a match (check for both **full words** and **single letters**).

If any of the rows contain the submitted string, add a select class to that row. Note that more than one row may contain the given string.

Оtherwise, if there is no match, **nothing should happen**.

**Note:** After every search ("Search" button is clicked), **clear** **the input field** and **remove** **all already selected classes** (if any) from the previous search, in order for the **new search** to contain only the **new result**.

For instance, if we try to find **eva:**



The result should be:



If we try to find all students who have email addresses in **softuni** domain, the expected result should be:



## Furniture

You will be given some furniture as an **array of objects**. Each object will have a name, a price and a decoration factor.

When the **"Generate" button is clicked**, add a **new row to the table** for each piece of furniture with image**,** name**,** price and decoration factor(code example below).

When the **"**Buy**"** button is clicked, get all **checkboxes that are marked** and show in the result textbox the **names** of the piece of furniture that **were checked**, separated by a **comma** and **single** **space** (**", "**) in the following format: **"Bought furniture: {furniture1} {furniture2}…"**.

On the next line, print the total price in format: **"Total price: {totalPrice}"** (formatted to the second decimal point). Finally, print the average decoration factor in the format: **"Average decoration factor: {decFactor}"**

### Input Example

**[{"name": "Sofa", "img": "https://res.cloudinary.com/maisonsdumonde/image/upload/q\_auto,f\_auto/w\_200/img/grey-3-seater-sofa-bed-200-13-0-175521\_9.jpg", "price": 150, "decFactor": 1.2}]**

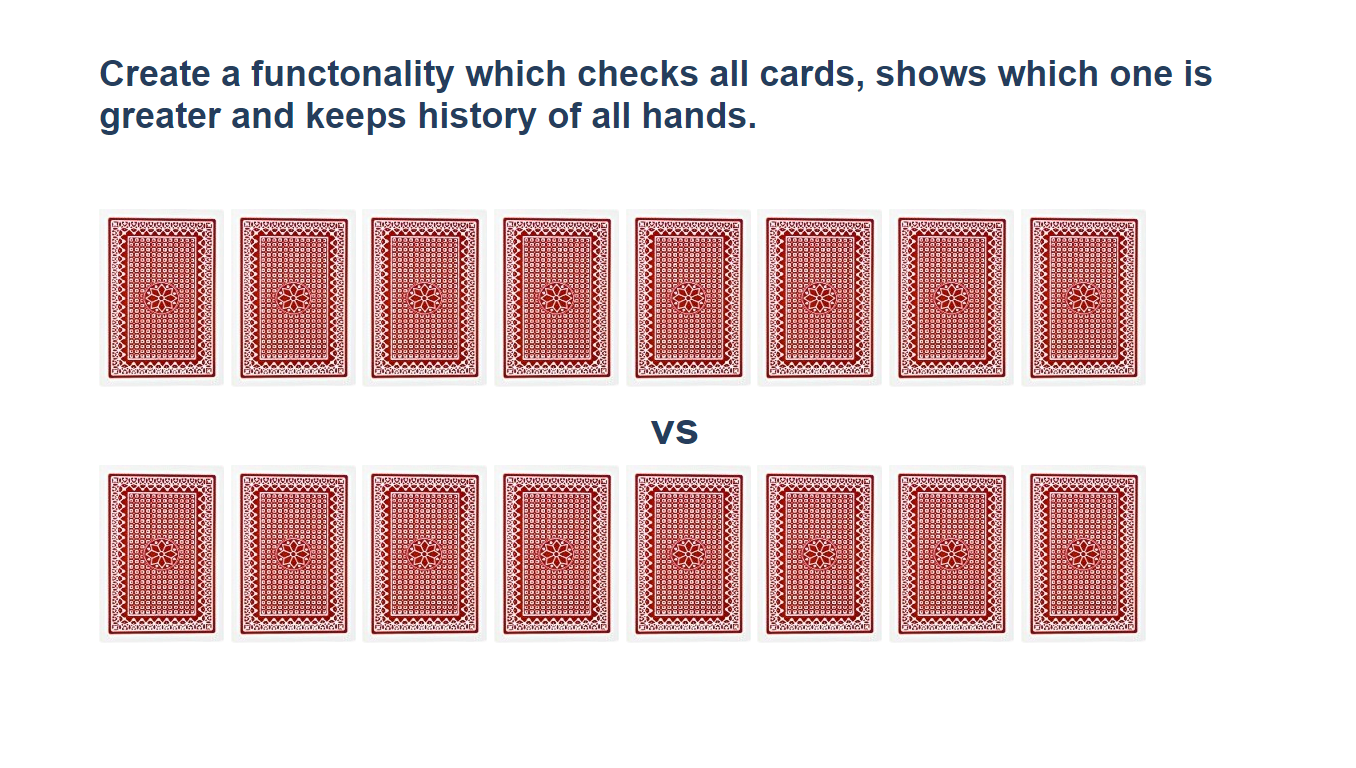
### Examples

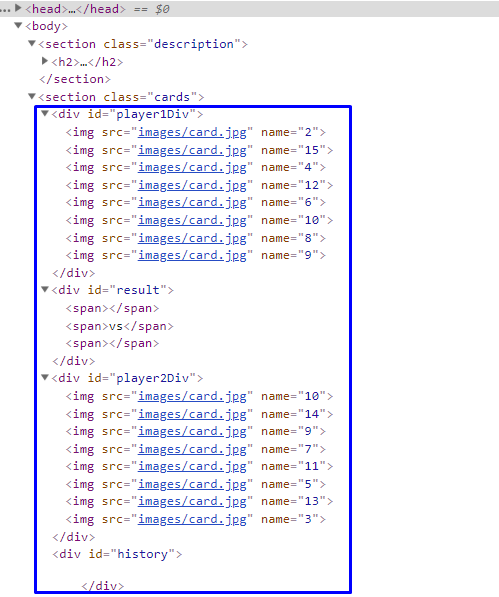




## Cards

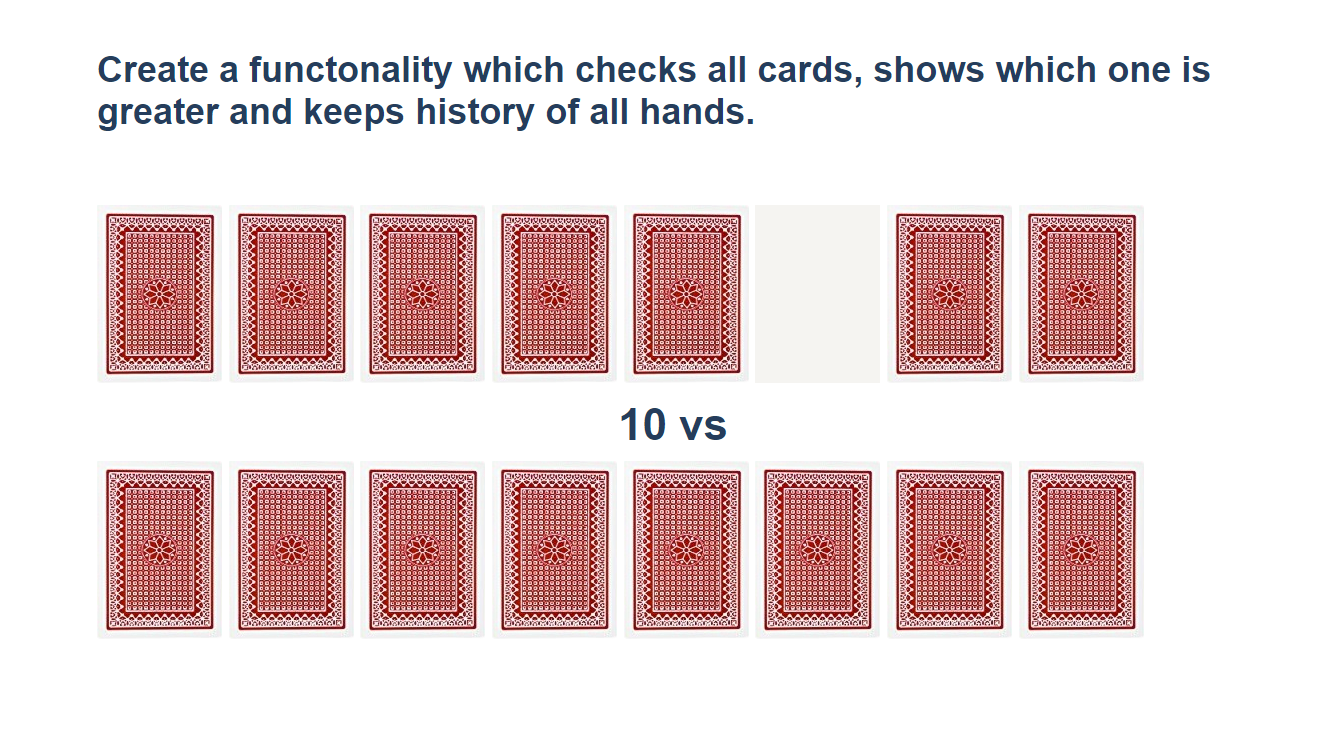
Write **a function** which **checks cards**, shows which one **is greater** and **keeps history of all hands**.

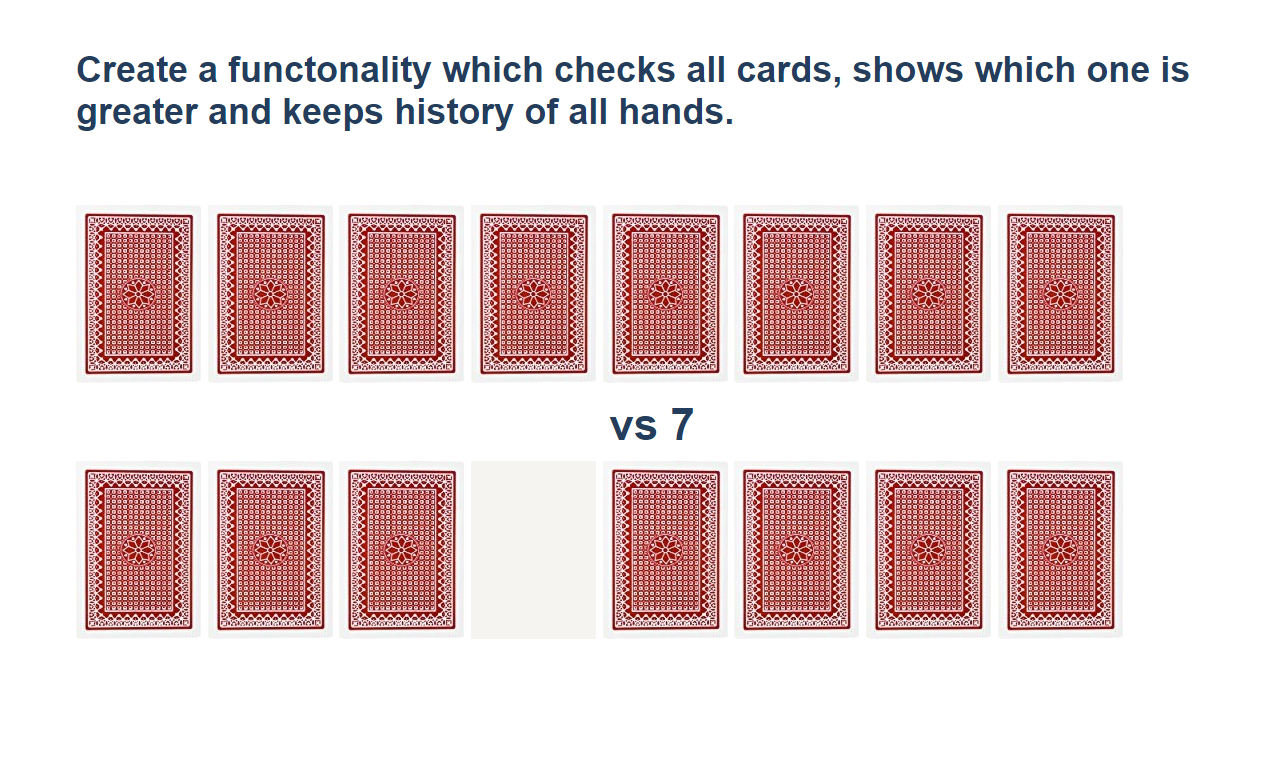




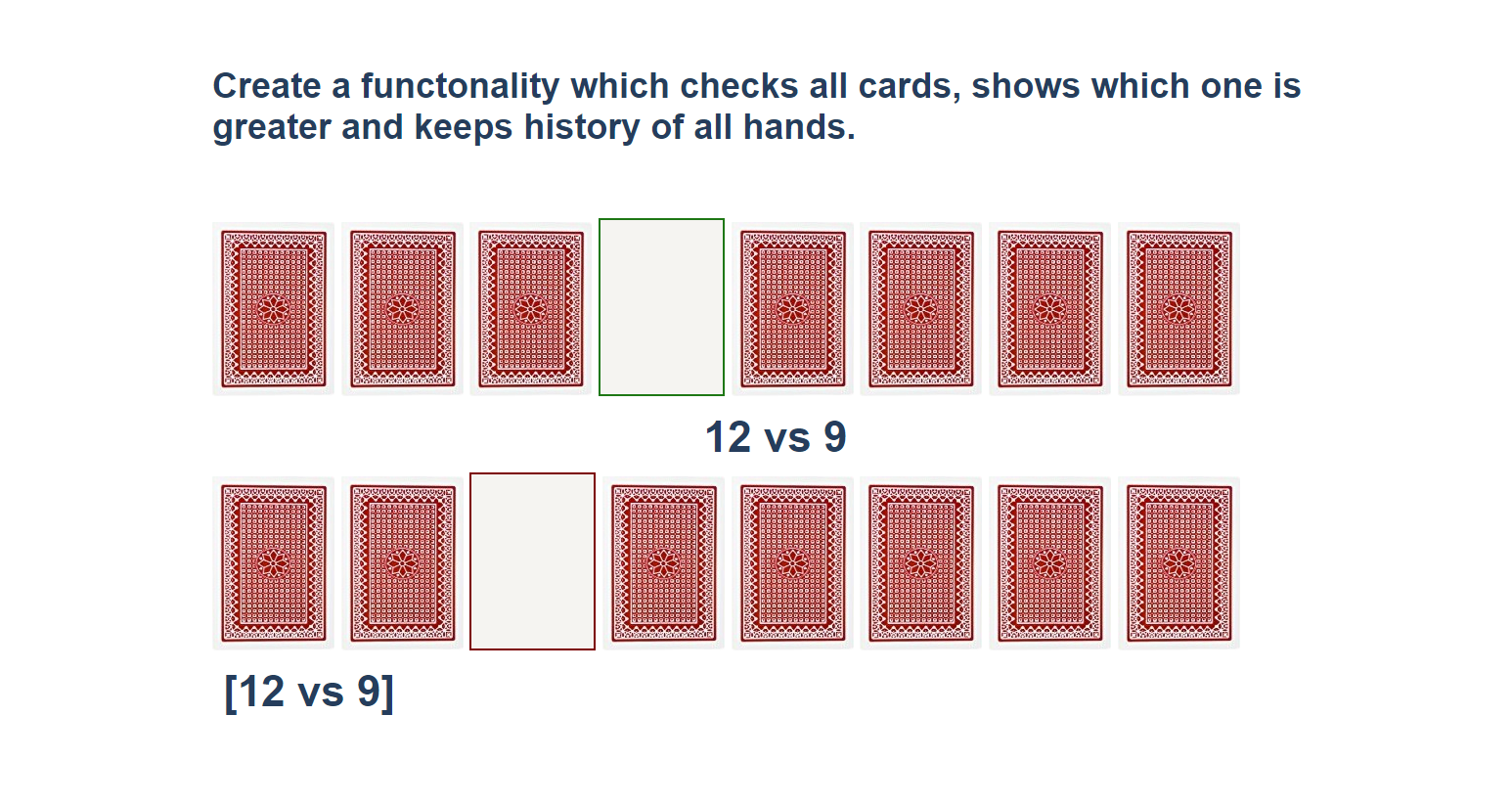
**Firstly, add** **click events** to **all cards**. When one of the cards is clicked, the current background card must be changed with the "whiteCard.jpg" picture (it is given in the skeleton) and the **card name should be appended** to one of the spanelementsinthedivwithid**="**result"**.**

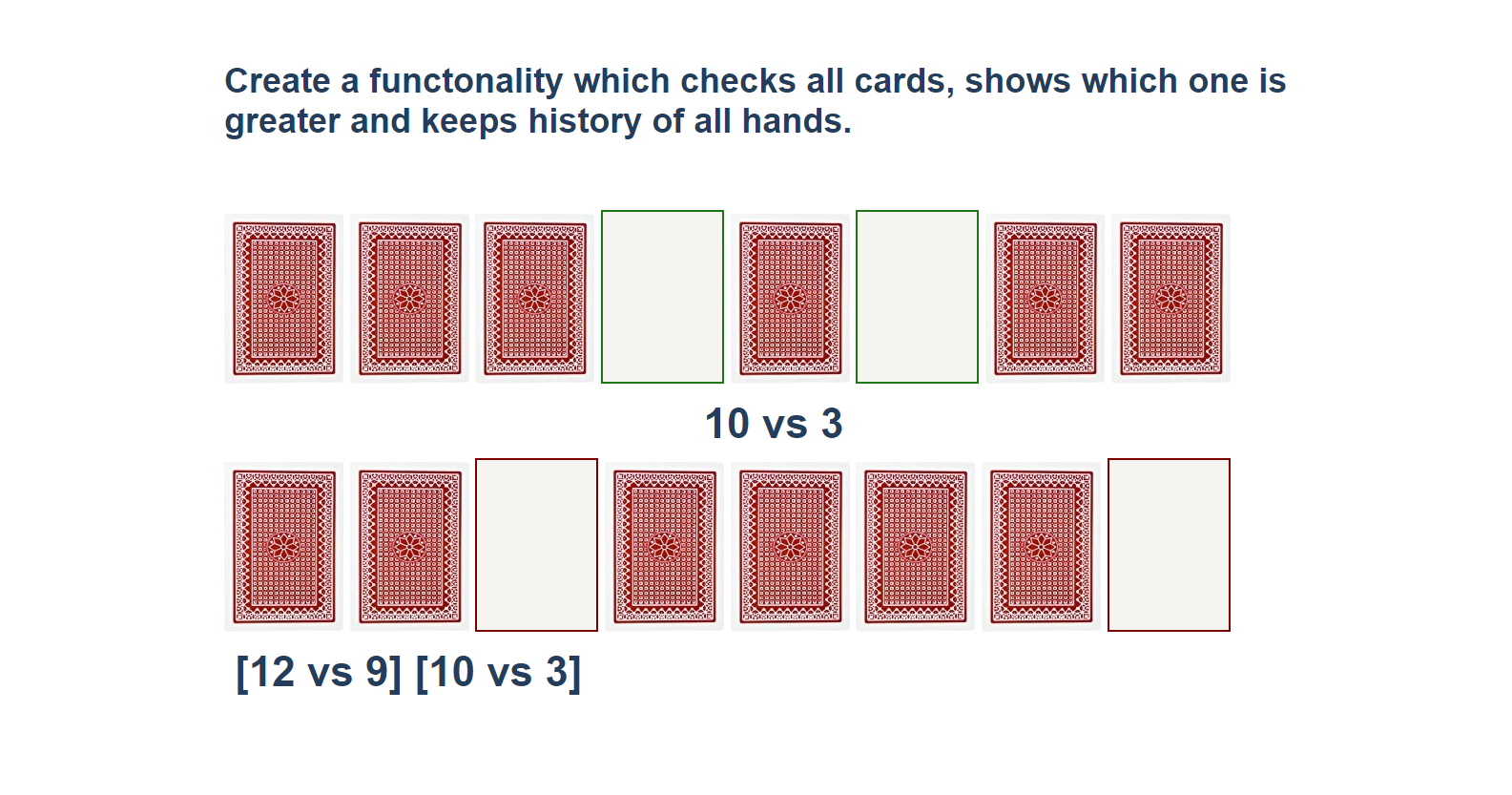
If a card **from the top side** is **clicked**, **append** the **card name** to the **left** span (first empty span), otherwise **append** the card **name** to the **right** span (second/last span).



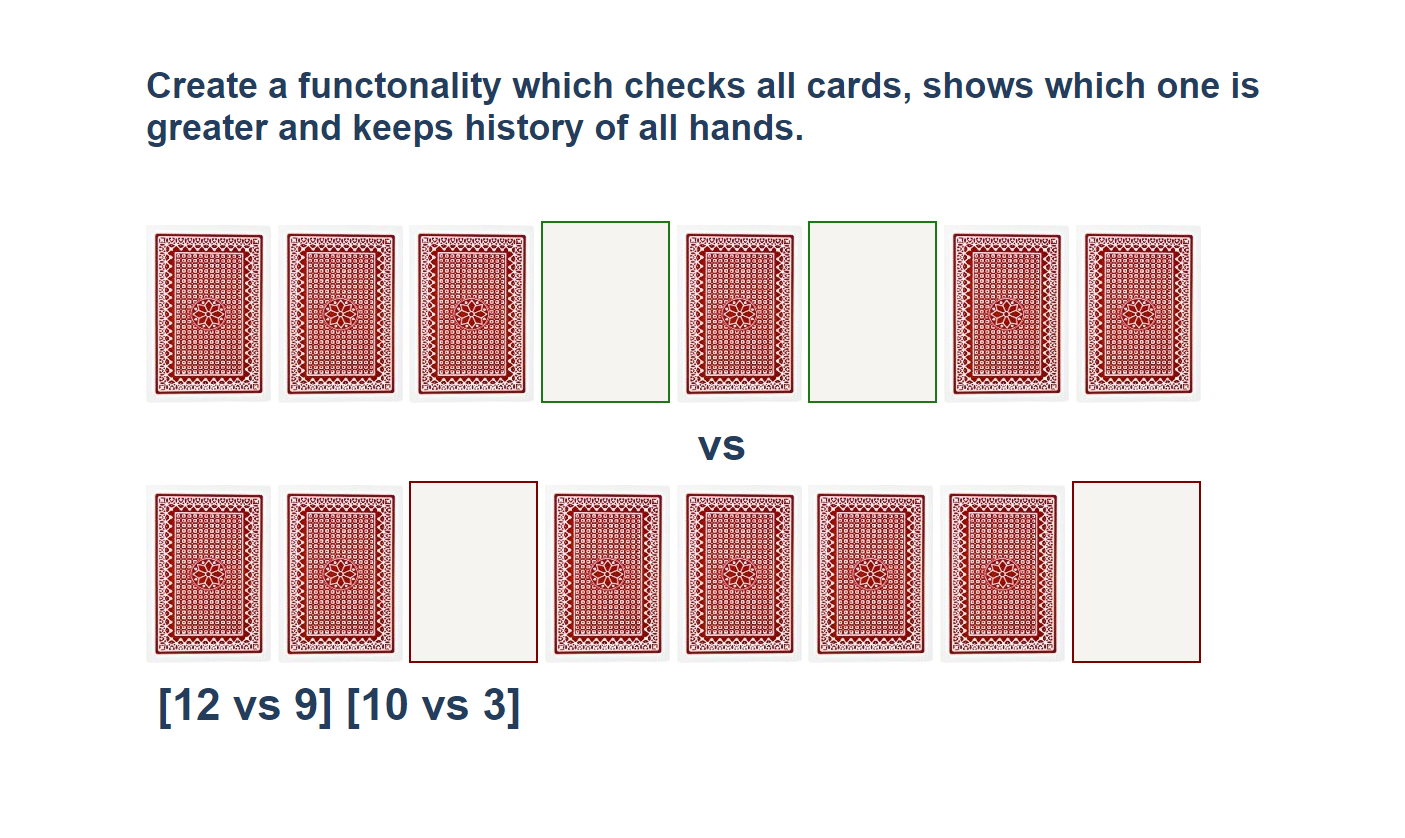


When **cards** from **both sides** **are selected,** **check** which one is **greater**. The greater card should have border "2px solid green" and the lower card - "2px solid red**"**.





You should **clear** the **span elements** which **hold the current card names** when both are selected, and the winner is selected. **After every hand,** push the current card names in the history div in the following format:



## \* Distance Converter

Your task is to convert from **one** distance unit to **another** by adding a **click** event listener to a button. When it is clicked, **read** the value from the input field and **get** the **selected** option from the **input** and **output** units drop downs. Then **calculate** and **display** the converted value in the **disabled** output field.

**Example**



**Hints**

* Multiply the incoming distance by the following conversion rates to convert to meter
* Divide to convert from meters to the required output unit
* To see which option is selected, read the properties of its parent: **value** gives you the value of the selected option (as displayed in the HTML), **selectedIndex** gives you the 0-based index of the selected option. For example, if miles are selected, **inputUnits.value** is "**mi**", **inputUnits.selectedIndex** is **4**. Option text is irrelevant
* Use the following table information to do that:

|  |  |
| --- | --- |
| **1 km** | **1000 m** |
| **1 m** | **1 m** |
| **1 cm** | **0.01 m** |
| **1 mm** | **0.001 m** |
| **1 mi** | **1609.34 m** |
| **1 yrd** | **0.9144 m** |
| **1 ft** | **0.3048 m** |
| **1 in** | **0.0254 m** |

## \* Sudomu

Write a function that implements **SUDOMU** (**Sudoku inside the DOM**).



The rules are simple and they are **the same** as the **typical sudoku game** (for more information,

click [here](https://sudoku.com/how-to-play/sudoku-rules-for-complete%20beginners/))

If the table is filled with the **right numbers**, and the "Quick Check" button is **clicked**, the

expected result should be:



The table borer should be changed to: "2px solid green**".** The text content of the paragraph

inside the div with an **id** "check" must be "You solve it! Congratulations!"

The text color of that div must be **green.**

Otherwise, when the filled table **does not solve** **the sudomu,** the result should be:



The table border should be changed to: "2px solid red".

The text content of the paragraphinside the div with an **id** "check" must be:

"NOP! You are not done yet…"

The text color of thatdiv must be **red!**

The"Clear"button **clears the whole** **SUDOMU (removes all numbers)** and the **paragraph**

**which contains the messages. It also removes the table border.**

