# JS Advanced: Exam Preparation

# Problem 1. Pet Me (DOM Manipulation)

**Link in Judge:** <https://judge.softuni.org/Contests/Practice/Index/2469#0>

**Environment Specifics**

Please, be aware that every JS environment may **behave differently** when executing code. Certain things that work in the browser are not supported in **Node.js**, which is the environment used by **Judge**.

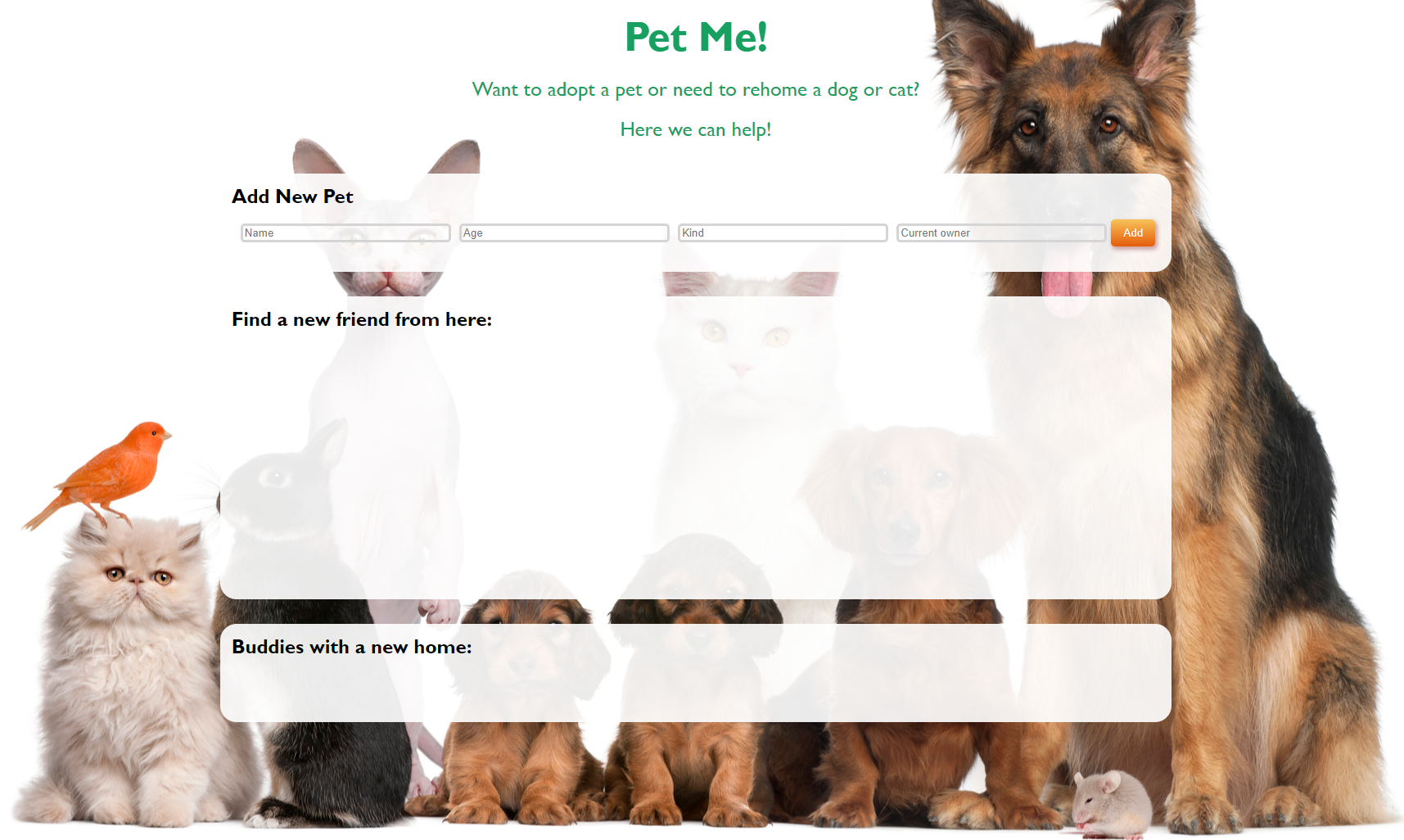
The following actions are **NOT** supported:

* **.forEach()** with **NodeList** (returned by **querySelector()** and **querySelectorAll()**)
* **.forEach()** with **HTMLCollection** (returned by **getElementsByClassName()** and **element.children**)
* Using the **spread-operator** (**...**) to convert a **NodeList** into an array
* **append()** in Judge (use only **appendChild()**)
* **replaceWith()** in Judge
* **replaceAll()** in Judge
* **closest()** in Judge

If you want to perform these operations, you may use **Array.from()** to first convert the collection into an array.

### Use the given skeleton to solve this problem.

### Note: You have NO permission to change directly the given HTML *(index.html file)*.



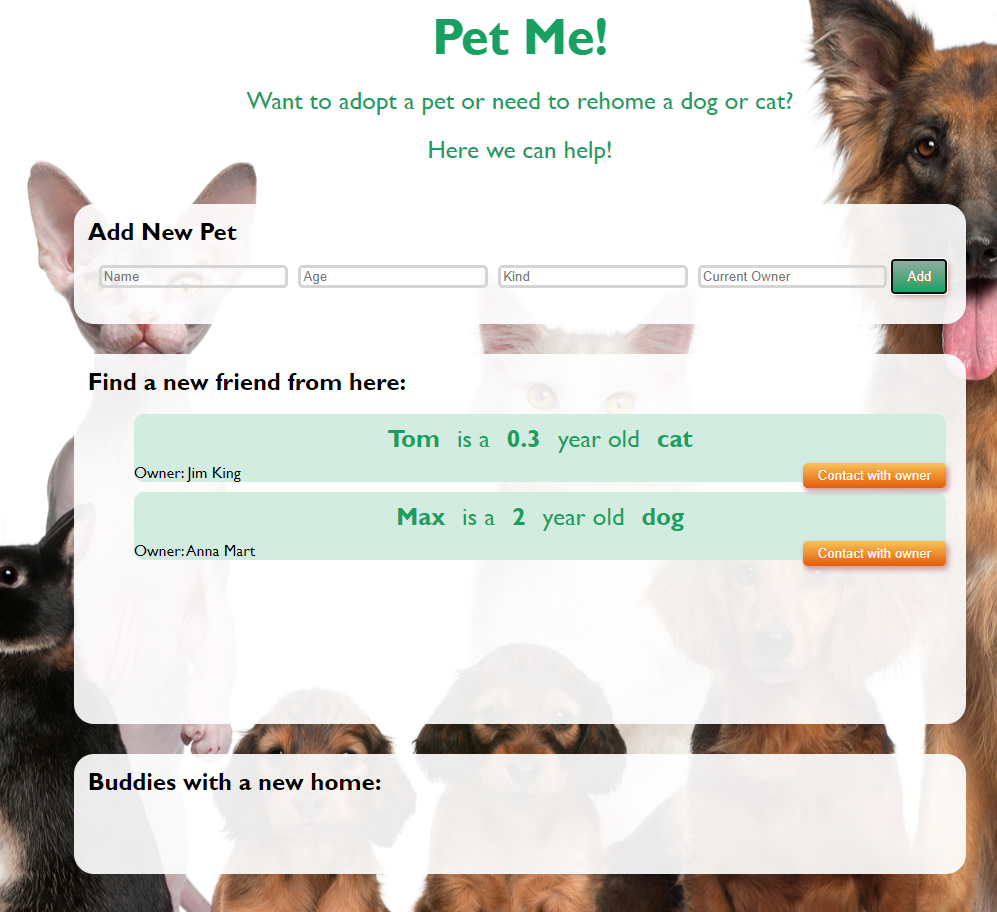
### Your Task

Write the missing JavaScript code to make the **Pet Me** application work as expected.

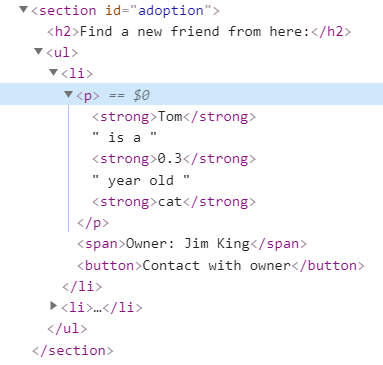
Each new registered pet must have a **Name**, **Age**, **Kind** and **Current Owner**.

When you click the **[Add]** button and **only** if all **inputs** are **filled** and the age is a **number**, then a new **list item** should be **added** to the section with id "**adoption**". Don't forget to **clear the inputs** when you add a new pet.

#### Already added pets

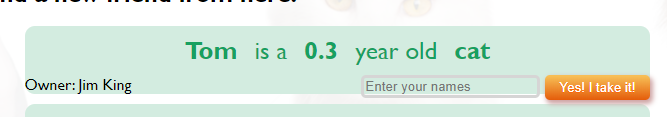


The new item should have the **following structure**:

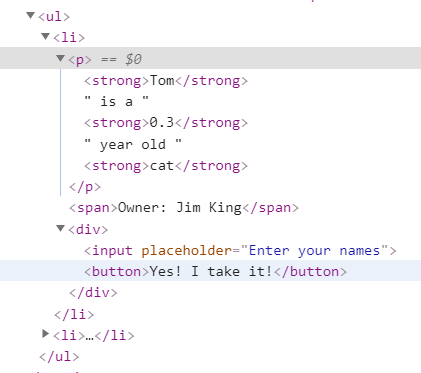


You should create a **li** element that contains a **paragraph** with the name, age, and kind of the new pet in the following format: **`{name} is a {years} year old {kind}`,** where the **name, years** and **kind** are in **strong** elements inside the paragraph. After that, we have a **span** element with **`Owner: {owner name}`**, and a button **[Contact with owner]**.

When you click the **[Contact with owner]** button an input field appears and the button changes to **[** **Yes! I take it! ]** like this:



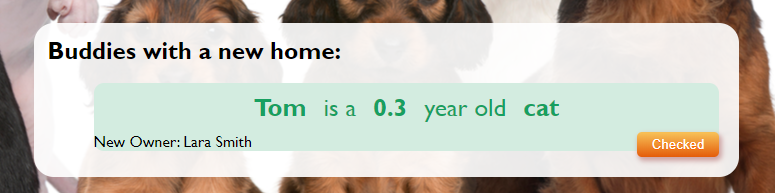
The new elements are into a **div** element and structure is changed like this:



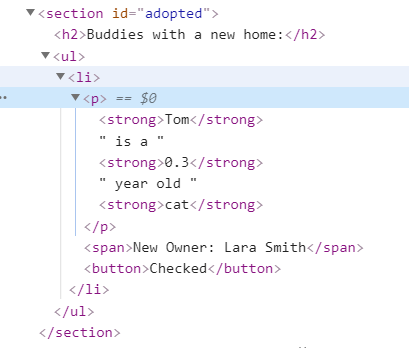
#### Moving pets into the new home section

When you click the button **[Yes! I take it!]**, **if there is entered the name** you should **move the current list item** to the **adopted section.**





Here we have changed the **owner name** with the new one. And the button is **[Checked]**. We have the next HTML structure:



And in the end button **[Checked]** must **delete** the current list item.

# Problem 2. Movie

**Link in Judge:** <https://judge.softuni.org/Contests/Practice/Index/2466#2>

class Movie {  
 *// TODO: implement this class...*  
}

Your Task

Write a class Movie, which implements the following functionality:

Functionality

#### constructor ( movieName, ticketPrice )

Receives 2 parameters at initialization of the class - movieName and ticketPrice.

Should have these **3** properties:

* **movieName** – property of type **string**;
* **ticketPrice** – property of type **number,** may come as a **string**;
* **screenings**– initially an empty **array**;

**Hint:** Here you can add more properties to help you finish the task.

#### newScreening(date, hall, description)

The date, hall and description are of type string.

* Check if there is already entered screening with the same date and hall and throw an Error:

`Sorry, {hall} hall is not available on {date}`

* Otherwise this function should add the screening to the screenings array and return:

`New screening of {movieName} is added.`

#### endScreening(date, hall, soldTickets)

* Check if the **screening array** has a screening with the given date and hall, if **NOT** **throw an Error**:

`Sorry, there is no such screening for {movieName} movie.`

* Otherwise calculate the current screening profit of sold tickets, add the profit to the total profit for the movie, also add the sold tickets count to the total sold movie tickets, delete the screening from the screenings array and return:

`{movieName} movie screening on {date} in {hall} hall has ended. Screening profit: {currentProfit}`

#### toString()

In the end is the toString() method where we return the full information of the movie.

* At the first line:

`{movieName} full information:`

* On the second two lines comes the collected profit, fixed to 0 and count of all sold tickets:

`Total profit: {profit}$

Sold Tickets: {soldTickets}`

* If there are screenings into the **screening array**, add on new line:

**"Remaining film screenings:"**

Sort screenings **alphabetically** by hall name and add a message for each of them on new line:

**`{hall} - {date} - {desrtiption}`**

* If there are no screenings into screenings array add this line to the returned message**:**

**"No more screenings!"**

Examples

|  |
| --- |
| Sample code usage |
| let m = new Movie('Wonder Woman 1984', '10.00');  console.log(m.newScreening('October 2, 2020', 'IMAX 3D', `3D`));  console.log(m.newScreening('October 3, 2020', 'Main', `regular`));  console.log(m.newScreening('October 4, 2020', 'IMAX 3D', `3D`));  console.log(m.endScreening('October 2, 2020', 'IMAX 3D', 150));  console.log(m.endScreening('October 3, 2020', 'Main', 78));  console.log(m.toString());  m.newScreening('October 4, 2020', '235', `regular`);  m.newScreening('October 5, 2020', 'Main', `regular`);  m.newScreening('October 3, 2020', '235', `regular`);  m.newScreening('October 4, 2020', 'Main', `regular`);  console.log(m.toString()); |
| Corresponding output |
| **New screening of Wonder Woman 1984 is added.**  **New screening of Wonder Woman 1984 is added.**  **New screening of Wonder Woman 1984 is added.**  **Wonder Woman 1984 movie screening on October 2, 2020 in IMAX 3D hall has ended. Screening profit: 1500**  **Wonder Woman 1984 movie screening on October 3, 2020 in Main hall has ended. Screening profit: 780**  **Wonder Woman 1984 full information:**  **Total profit: 2280$**  **Sold Tickets: 228**  **Remaining film screenings:**  **IMAX 3D - October 4, 2020 - 3D**    **Wonder Woman 1984 full information:**  **Total profit: 2280$**  **Sold Tickets: 228**  **Remaining film screenings:**  **235 - October 4, 2020 - regular**  **235 - October 3, 2020 - regular**  **IMAX 3D - October 4, 2020 - 3D**  **Main - October 5, 2020 - regular**  Main - October 4, 2020 - regular |

# Problem 3. Test Numbers

**Link in Judge:** <https://judge.softuni.org/Contests/Practice/Index/3008#2>

### Your Task

Using **Mocha** and **Chai** write **JS Unit Tests** to test a variable named **testNumbers**, which represents an object. You may use the following code as a template:

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

The object that should have the following functionality:

* sumNumber(num1, num2) - A function that accepts two parameters:
  + check if parameters are numbers
  + numbers can be positive and negative
  + if parameters aren't number, function return undefined
  + the function returns the sum of the given numbers, rounded to second number after decimal point
* numberChecker(input) - A function that accepts a single parameter:
  + the function parses the input to number, and validates it
  + if the input is a number, the function checks if it is even. If so the function returns the string: **"The number is even!"**
  + otherwise the function returns: **"The number is odd!"**
  + If the input is not a number the function throws an error – **"The input is not a number!"**
* averageSumArray(arr) - A function that accept single parameter (array):
  + the function iterates through each element in the array, calculates the sum, and returns the average sum
  + The array will be always valid, there is no need to test the input arrays.

### JS Code

To ease you in the process, you are provided with an implementation which meets all of the specification requirements for the **testNumbers** object:

|  |
| --- |
| testNumbers.js |
| const testNumbers = {      sumNumbers: function (num1, num2) {  let sum = 0;  **if** (**typeof**(num1) !== **'number'** || **typeof**(num2) !== **'number'**) {  **return undefined**;  } **else** {  **sum = (num1 + num2).toFixed(2);**           return sum  }      },      numberChecker: function (input) {          input = Number(input);    if (isNaN(input)) {              throw new Error('The input is not a number!');          }          if (input % 2 === 0) {              return 'The number is even!';          } else {              return 'The number is odd!';  }      },      averageSumArray: function (arr) {          let arraySum = 0;          for (let I = 0; I < arr.length; i++) {              arraySum += arr[i]          }          return arraySum / arr.length      }  }; |

### Submission

Submit your tests inside a describe() statement, as shown above.

*GOOD LUCK! ☺*