JS Advanced: Exam Preparation 1

Link to contest: https://judge.softuni.org/Contests/3425/JS-Advanced-Retake-Exam

Problem 1. Forum posts

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:

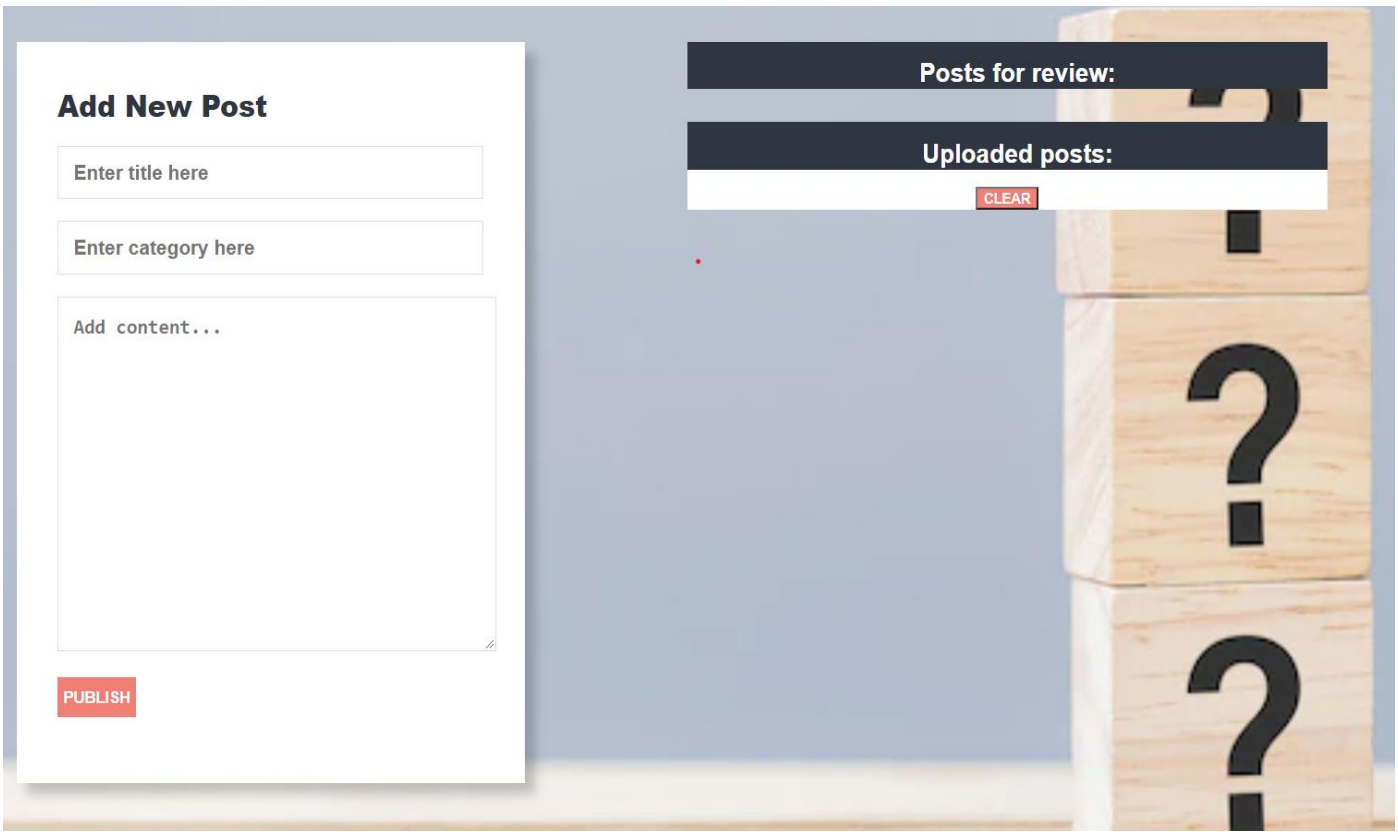
- `.forEach()` with **NodeList** (returned by `querySelector()` and `querySelectorAll()`)
- `.forEach()` with **HTMLCollection** (returned by `getElementsByClassName()` and `element.children`)
- Using the **spread-operator** (`...`) to convert a **NodeList** into an array
- `append()` in Judge (use only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

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

Use the provided skeleton to solve this problem.

Note: You **can't** and you have no permission to **change** directly the given HTML code (index.html file).

Write the missing JavaScript code to make the **Furniture Store** work as expected:



Your Task

Write the missing JavaScript code to make the **Forum** work as expected:

All fields **(title, category,** and **content)** are **filled with the correct input**

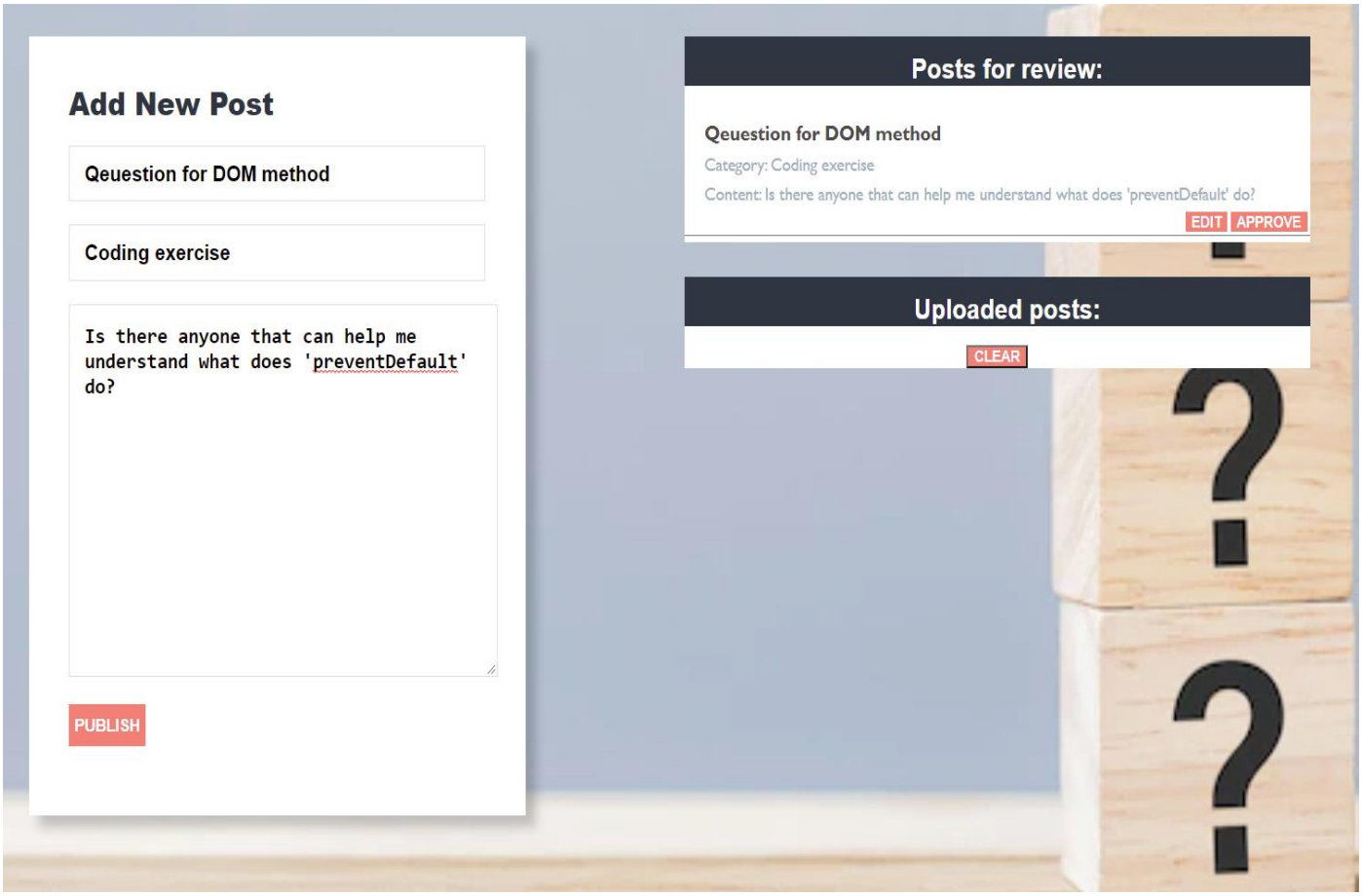
- **Title, category,** and **content** are **non-empty strings**. If any of them are empty, the program should not do anything.

1. Getting the information from the form

When you click the **["Publish"]** button, the information from the input fields must be added to the **ul** with the **id "review-list"** and **then clear input fields**.

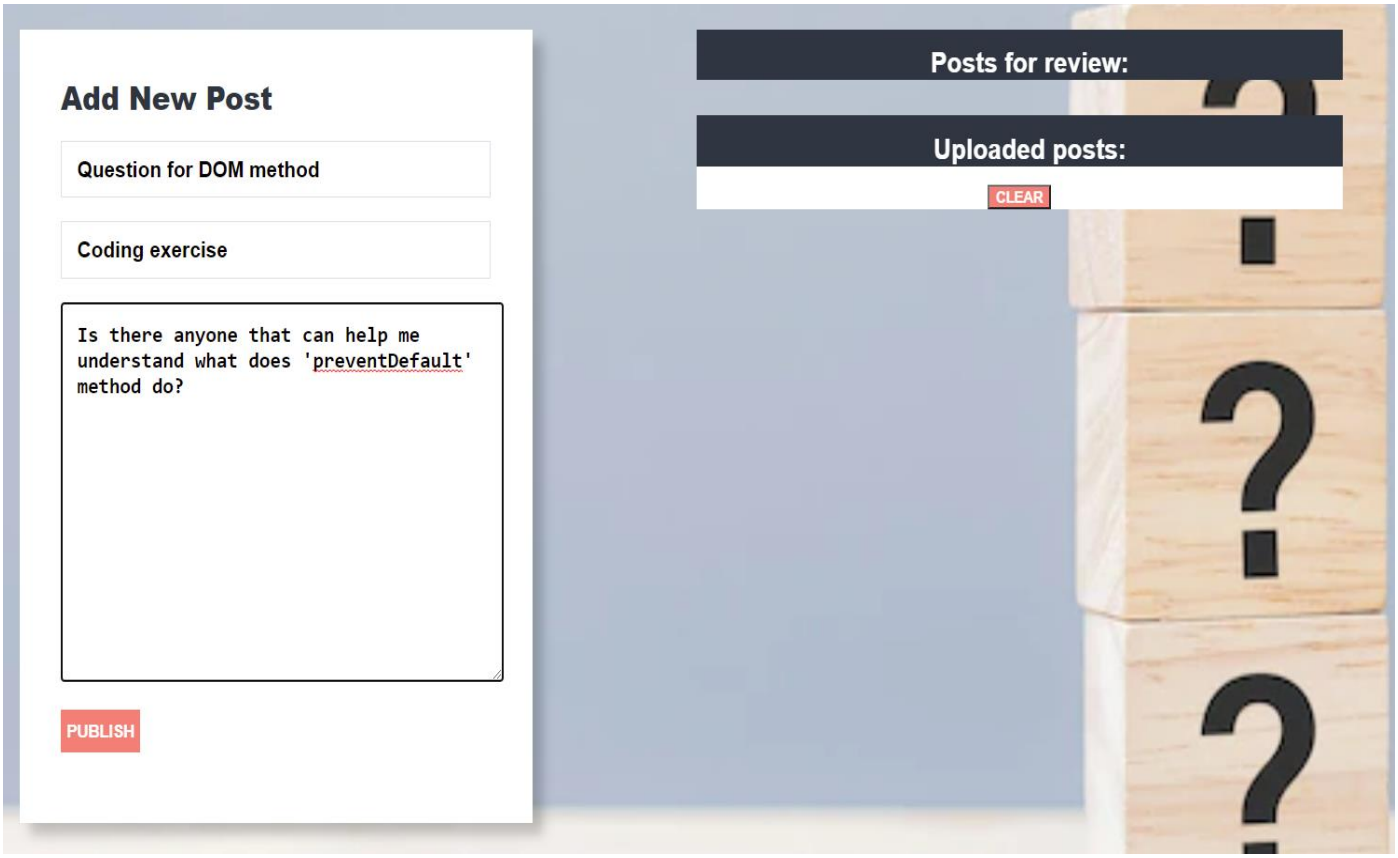
The HTML structure looks like this:

```
<ul id="review-list">
  <li class="rpost">
    <article>
      <h4>Question for DOM method</h4>

      <p>Category: Coding exercise</p>

      <p>Content: Is there anyone that can help me understand what does 'preventDefault' method do?</p>
    </article>
    <button class="action-btn edit">Edit</button>
    <button class="action-btn approve">Approve</button>
  </li>
</ul>
```

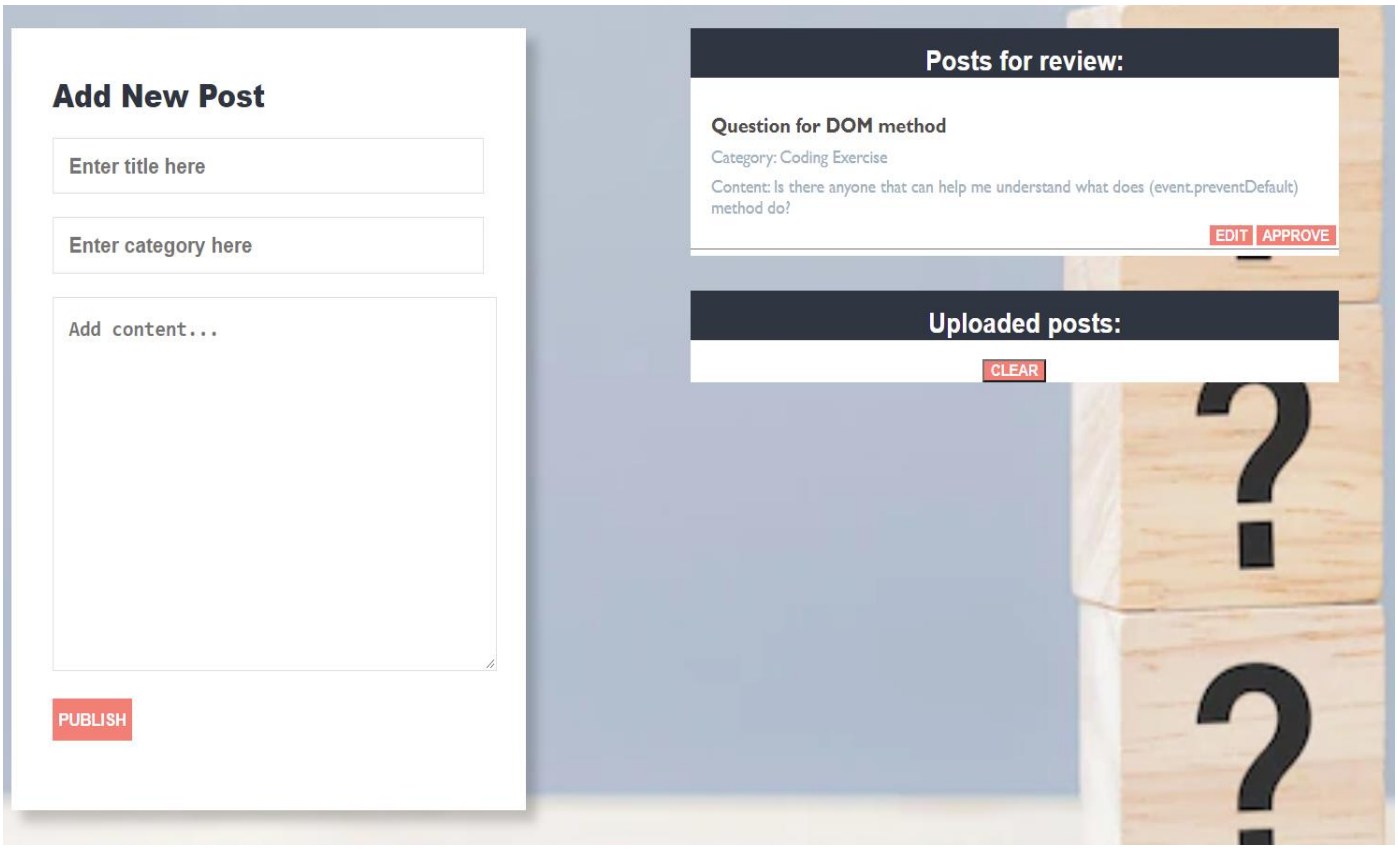


2. Edit information for posts

When the **["Edit"]** button is clicked, the information from the post must be sent to the input's fields and the record should be deleted from the **ul "review-list"**.



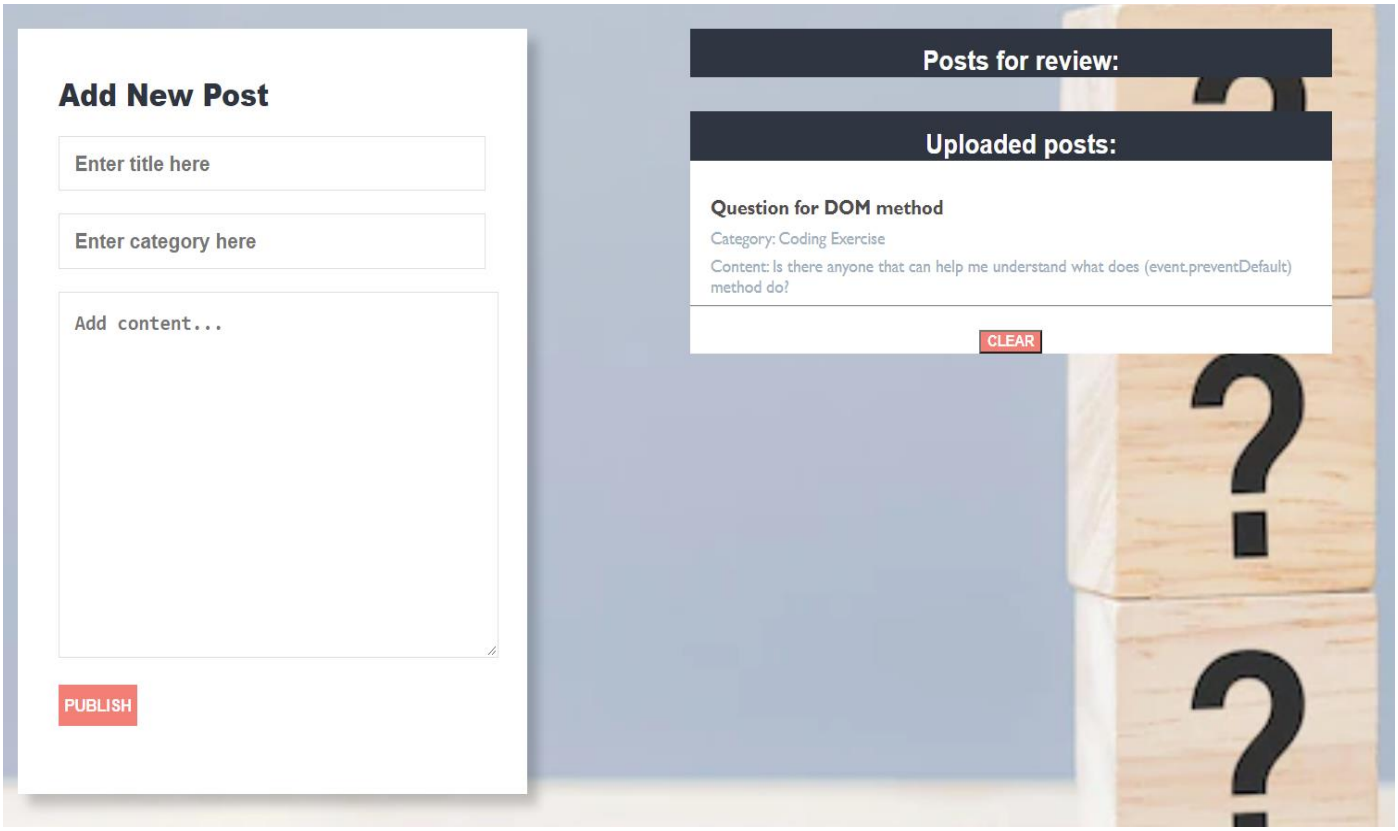
After editing the information make a new record to the **ul** with updated information.



3. Approve posts

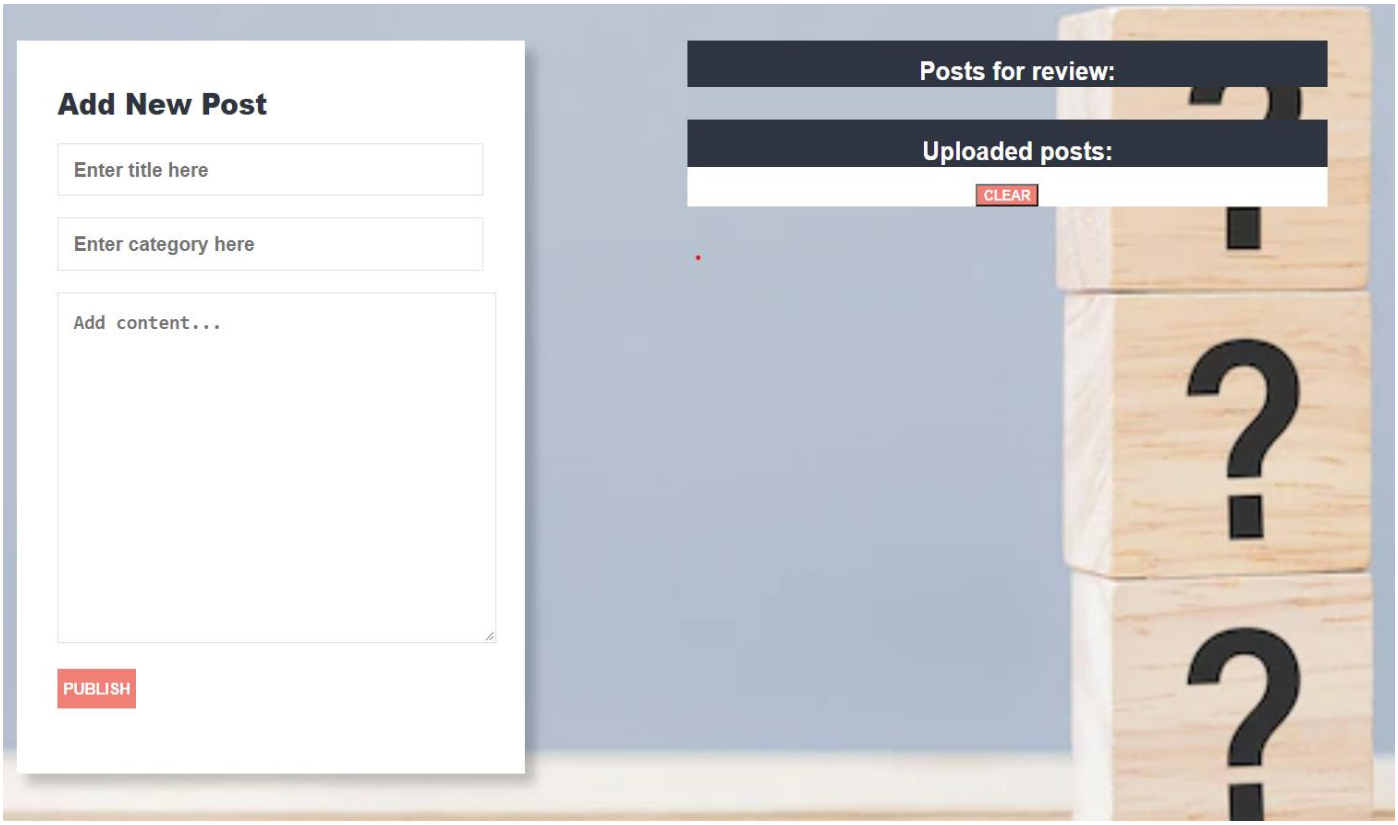
When you click the **["Approve"]** button, the record must be **deleted** from the **ul** with **id "review-list"**and appended to the **ul** with the **id "published-list"**
The **buttons [“Edit”]** and **[“Approve”]** should be removed from the **li** element.

```
<ul id="published-list">
  <li class="rpost">
    <article>
      <h4>Question for DOM method</h4>
      <p>Category: Coding exercise</p>
      <p>Content: Is there anyone that can help me understand what does 'preventDefault' do?</p>
    </article>
  </li>
</ul>
```



4. Clear posts

When you click the **["Clear"]** button, the record for all posts must be **deleted** from the **ul** with the **id "published-list".**



Problem 2. Garden

```
class Garden {

    //TODO Implement this class

}
```

Write a class **Garden**, which implements the following functionality:

Functionality

Constructor

Should have these **3** properties:

- **`spaceAvailable`** – Number
- **`plants`** – Array (empty)
- **`storage`** – Array (empty)

At the initialization of the Garden class, the **constructor** accepts the **spaceAvailable.**

`addPlant (plantName, spaceRequired)`

The **`plantName`** is of type **string**, while the **`spaceRequired`** is of type **number**.

- If there is **not enough space in the garden** for the new plant, **throw an Error**:

  `"Not enough space in the garden."`

- Otherwise, this function should add the plant with the properties: **plantName**, **spaceRequired**, **ripe**: **default false, quantity: default 0** to the plants array, reduce the space available with the space required by the plant, and **return**:

  `"The {plantName} has been successfully planted in the garden."`

NOTE: Plant names will be unique.

ripenPlant(plantName, quantity)

The **quantity** is of type **number**.

- If the plant is not found, **throw an Error**:

  `"There is no {plantName} in the garden."`

- If the plant is already ripe, **throw an Error**:

  `"The {plantName} is already ripe."`

- If the received quantity is less than or equal to 0, **throw an Error**:

  `"The quantity cannot be zero or negative."`

- Otherwise, this function should set the ripe property of the particular plant to true and add the quantity to the quantity property of the plant. If the quantity passed as a parameter is 1, **return:**

  `"{quantity} {plantName} has successfully ripened."`

  If the quantity parameter is greater than 1, **return:**

  `"{quantity} {plantName}s have successfully ripened."`

harvestPlant(plantName)

- If the plant is **not found**, throw **an Error:**

  `"There is no {plantName} in the garden."`

- If the plant **is not ripe**, throw **an Error**:

  `"The {plantName} cannot be harvested before it is ripe."`

- Otherwise, this function should **remove** the plant from the plants array, add it to storage with properties **plantName** and **quantity,** free up the total space that the plant required, and **return**:

  `"The {plantName} has been successfully harvested."`

generateReport()

This method should **return** the complete information about the garden:

- On the first line:

  `"The garden has { spaceAvailable } free space left."`

- On the second line list all plants that are in the garden **ordered alphabetically by plant name ascending** in the format:

  `"Plants in the garden: {plant1Name}, {plant2Name}, {…}"`

- On the third line add:

  If there are no plants in the storage, print:

  "Plants in storage: The storage is empty."

  If there are plants in the storage list them in the format:

  "Plants in storage: {plant1Name} ({plant1Quantity}), {plant2Name}, ({plant2Quantity}), {…}"

Examples

Input 1

`const myGarden = new Garden(250)`
 	
`console.log(myGarden.addPlant('apple', 20));`
 	
`console.log(myGarden.addPlant('orange', 200));`
 	
`console.log(myGarden.addPlant('olive', 50));`

Output 1
The apple has been successfully planted in the garden. The orange has been successfully planted in the garden. Uncaught Error Error: Not enough space in the garden.

Input 2

`const myGarden = new Garden(250)`
 	
`console.log(myGarden.addPlant('apple', 20));`
 	
`console.log(myGarden.addPlant('orange', 100));`
 	
`console.log(myGarden.addPlant('cucumber', 30));`
 	
`console.log(myGarden.ripenPlant('apple', 10));`
 	
`console.log(myGarden.ripenPlant('orange', 1));`
 	
`console.log(myGarden.ripenPlant('orange', 4));`

Output 2

```
The apple has been successfully planted in the garden.

The orange has been successfully planted in the garden.

The cucumber has been successfully planted in the garden.

10 apples have successfully ripened.

1 orange has successfully ripened.

Uncaught Error Error: The orange is already ripe.
```

```
const myGarden = new Garden(250)

console.log(myGarden.addPlant('apple', 20));

console.log(myGarden.addPlant('orange', 100));

console.log(myGarden.addPlant('cucumber', 30));

console.log(myGarden.ripenPlant('apple', 10));

console.log(myGarden.ripenPlant('orange', 1));

console.log(myGarden.ripenPlant('olive', 30));
```

Output 3

```
The apple has been successfully planted in the garden.

The orange has been successfully planted in the garden.

The cucumber has been successfully planted in the garden.

10 apples have successfully ripened.

1 orange has successfully ripened.

Uncaught Error Error: There is no olive in the garden.
```

Input 4

```
const myGarden = new Garden(250)

console.log(myGarden.addPlant('apple', 20));

console.log(myGarden.addPlant('orange', 100));

console.log(myGarden.addPlant('cucumber', 30));

console.log(myGarden.ripenPlant('apple', 10));

console.log(myGarden.ripenPlant('orange', 1));

console.log(myGarden.ripenPlant('cucumber', -5));
```

Output 4

```
The apple has been successfully planted in the garden.

The orange has been successfully planted in the garden.

The cucumber has been successfully planted in the garden.

10 apples have successfully ripened.

1 orange has successfully ripened.

Uncaught Error Error: The quantity cannot be zero or negative.
```

Input 5

```
const myGarden = new Garden(250)

console.log(myGarden.addPlant('apple', 20));

console.log(myGarden.addPlant('orange', 200));

console.log(myGarden.addPlant('raspberry', 10));

console.log(myGarden.ripenPlant('apple', 10));
```

```
console.log(myGarden.ripenPlant('orange', 1));

console.log(myGarden.harvestPlant('apple'));

console.log(myGarden.harvestPlant('olive'));
```

Output 5

```
The apple has been successfully planted in the garden.

The orange has been successfully planted in the garden.

The raspberry has been successfully planted in the garden.

10 apples have successfully ripened.

1 orange has successfully ripened.

The apple has been successfully harvested.

Uncaught Error Error: There is no olive in the garden.
```

```
const myGarden = new Garden(250)

console.log(myGarden.addPlant('apple', 20));

console.log(myGarden.addPlant('orange', 200));

console.log(myGarden.addPlant('raspberry', 10));

console.log(myGarden.ripenPlant('apple', 10));

console.log(myGarden.ripenPlant('orange', 1));

console.log(myGarden.harvestPlant('apple'));

console.log(myGarden.harvestPlant('raspberry'));
```

Output 6

```
The apple has been successfully planted in the garden.

The orange has been successfully planted in the garden.

The raspberry has been successfully planted in the garden.

10 apples have successfully ripened.

1 orange has successfully ripened.

The apple has been successfully harvested.

Uncaught Error Error: The raspberry cannot be harvested before it
is ripe.
```

Input 6

```
const myGarden = new Garden(250)

console.log(myGarden.addPlant('apple', 20));

console.log(myGarden.addPlant('orange', 200));

console.log(myGarden.addPlant('raspberry', 10));

console.log(myGarden.ripenPlant('apple', 10));

console.log(myGarden.ripenPlant('orange', 1));

console.log(myGarden.harvestPlant('orange'));

console.log(myGarden.generateReport());
```

Output 6

```
The apple has been successfully planted in the garden.

The orange has been successfully planted in the garden.

The raspberry has been successfully planted in the garden.

10 apples have successfully ripened.

1 orange has successfully ripened.
```

```
The orange has been successfully harvested.

The garden has 220 free space left.

Plants in the garden: apple, raspberry

Plants in storage: orange (1)
```

Problem 3. Book selection

Your Task

Using **Mocha** and **Chai** write **JS Unit Tests** to test a variable named **bookSelection**, 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:

`isGenreSuitable (genre, age)` - A function that accepts two parameters: **string** and **number**.

- If the value of the string **`genre`** is equal to "**`Thriller`**" or "**`Horror`**" and the value of **`age`** is less or equal to **12**,
  return: **`` `Books with ${genre} genre are not suitable for kids at ${age} age` ``**
- Otherwise, if the above conditions are not met, **return** the following message:

  `` `Those books are suitable` ``
- There is **no** need for **validation** for the **input**, you will always be given string and number.

- **`isItAffordable (price, budget)`** - A function that accepts two parameters: **number** and **number**.
  - You need to **calculate** if you can afford buying the book by **subtracting** the **`price`** of the book from your **`budget`**.
  - If the **result** is lower than **0,** return**:**

    `"You don't have enough money"`
  - Otherwise, if the above conditions are not met, **return** the following message:

    `` `Book bought. You have ${result}$ left` ``

- You need to validate the input, if the **`price`** and **`budget`** are not a **number**, **throw** an error: "**`Invalid input`**".

- **`suitableTitles(books, wantedGenre)`** - A function that accepts an array and string.
  - The **`books`** array will store the titles and the genre of its books ([{ **title**: "**The Da Vinci Code**", **genre**: "**Thriller**" }, ...])
  - You must **add** the **title** of the book that its genre is equal to the **`wantedGenre`**.
  - Finally, **return** the changed array of book titles.
  - There is a need for validation for the input, an **array** and **string** may not always be valid. In case of submitted **invalid** parameters, **throw** an error "**`Invalid input`**":
    - If passed **`books`** parameter is not an array.
    - If the **`wantedGenre`** is not a string.

JS Code

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

bookSelection.js

```
const bookSelection = {
  isGenreSuitable(genre, age) {
    if (age <= 12 && (genre === "Thriller" || genre === "Horror")) {
      return `Books with ${genre} genre are not suitable for kids at ${age} age`;
    } else {
      return `Those books are suitable`;
    }
  },
  isItAffordable(price, budget) {
    if (typeof price !== "number" || typeof budget !== "number") {
      throw new Error("Invalid input");
    }

    let result = budget - price;

    if (result < 0) {
      return "You don't have enough money";
    } else {
      return `Book bought. You have ${result}$ left`;
    }
  },
  suitableTitles(array, wantedGenre) {
```

```
    let resultArr = [];

    if (!Array.isArray(array) || typeof wantedGenre !== "string") {
      throw new Error("Invalid input");
    }
    array.map((obj) => {
      if (obj.genre === wantedGenre) {
        resultArr.push(obj.title);
      }
    });
    return resultArr;
  },
};
```