JS Advanced Exam-Retake

Problem 1. Gem Collection

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**.

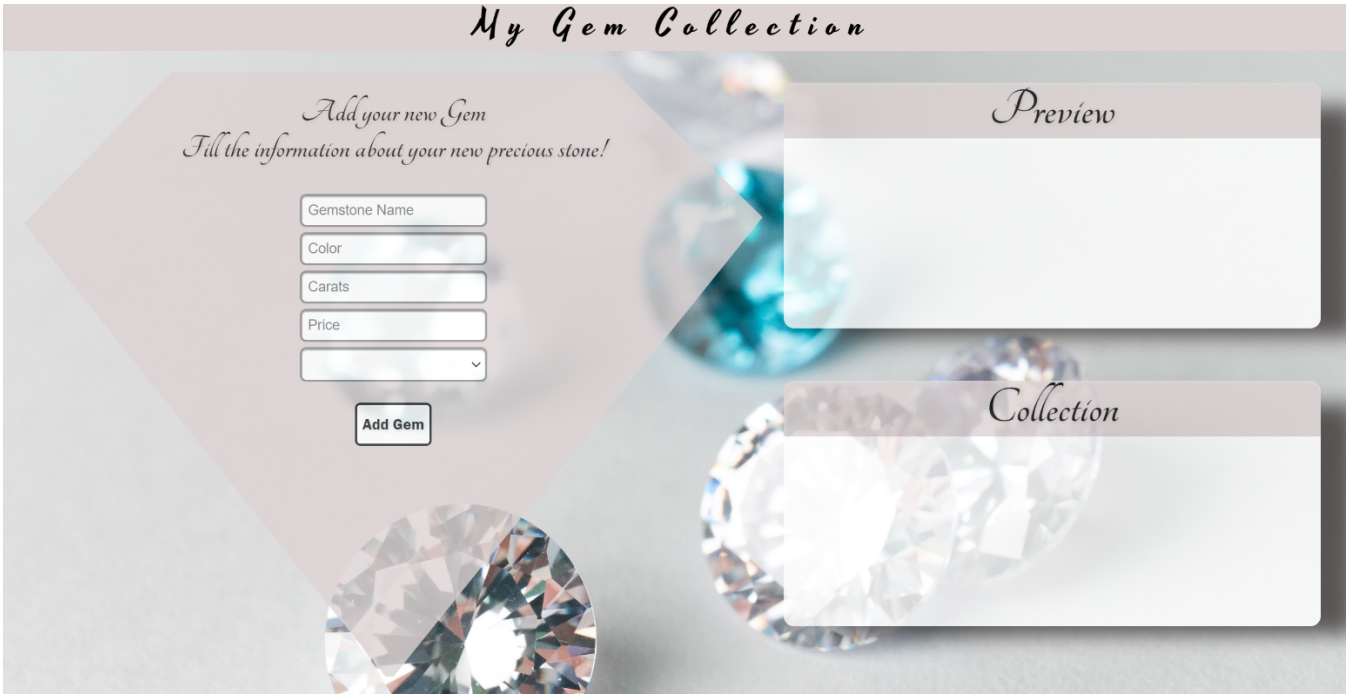
The following actions are **NOT** supported:

- `.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()`)
- `prepend()`
- `replaceWith()`
- `replaceAll()`
- `closest()`
- `replaceChildren()`
- Always turn the collection into a **JS array** (forEach, forOf, et.)

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.

Write the missing functionality of this user interface. The functionality is divided in the following steps:



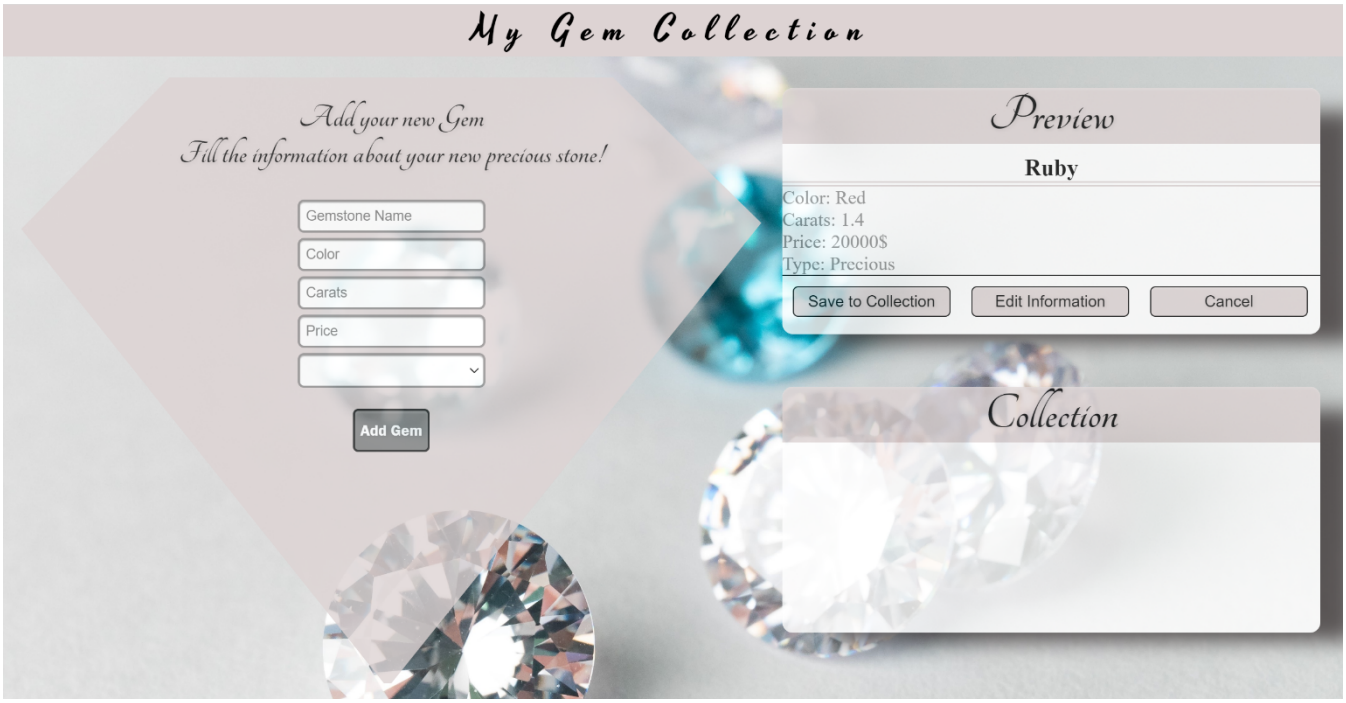
Your Task

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

All fields **(Gemstone Name, Color, Carats, Price,** and **Type)** are **filled with the correct input**

- **Gemstone Name, Color, Age, Carats, Price** and **Type** are **non-empty strings**. If any of them is empty, the program should not do anything.

1. Getting the information from the form



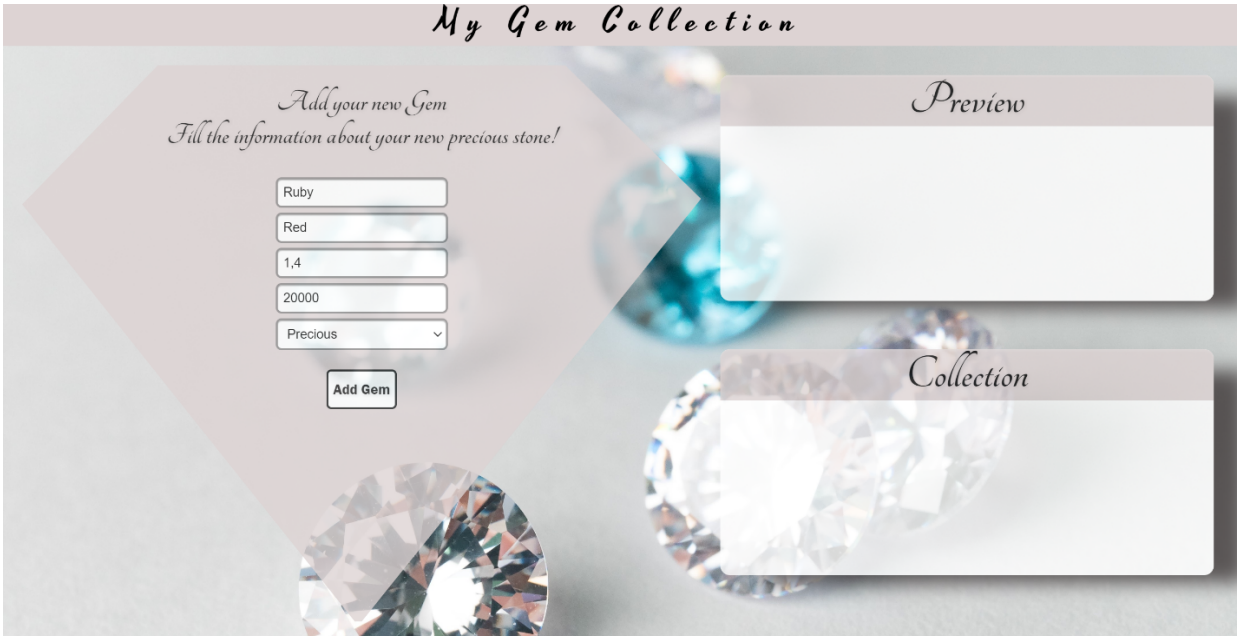
- On clicking the **["Add Gem"]** button the information from the input fields is listed in the **"preview"** section. To the unordered list with id **"preview-list"**, should be added a list item, containing the input information.
- The text format and order for each list item should be the same as in the picture below.
- When the button is clicked, the input fields must be cleared and the **["Add Gem"]** button must be **disabled**. At the same time the **"Save to Collection", "Edit Information"** and the **"Cancel"** buttons must be **added**.
  The HTML structure looks like this:

```
<h3>Preview</h3> == $0
<ul id="preview-list">
  <li class="gem-info">
    <article>
      <h4>Ruby</h4>
      <p>Color: Red</p>
      <p>Carats: 1.4</p>
      <p>Price: 20000$</p>
      <p>Type: Precious</p>
    </article>
    <button class="save-btn">Save to Collection</button>
    <button class="edit-btn">Edit Information</button>
    <button class="cancel-btn">Cancel</button>
  </li>
</ul>
```

2. Editing the information from the Preview

The functionality here is the following:

- When the **"Edit Information"** button is clicked, all of the **information** is **loaded** in the **input fields** from step 1 and all buttons in preview section are **removed** while the **["Add Gem"]** button is **enabled** again.



- The **list items** must be **removed** from the **"preview-list" and** all of the **information** must go back to the **input fields** again**.**

```
    <h3>Preview</h3>
    <ul id="preview-list"> </ul>
</div>
```

3. Saving the information from the Preview

- When the **"Save to Collection"** button is clicked, the gem information must be transferred from **"preview"** to **"collection",** for you that means a new list item should be added to the unordered list with id: **"collection",** having the information about gems.

The HTML structure looks like this:

```
<h3>Collection</h3>
<ul id="collection">
▼<li>
    <p class="collection-item">Ruby - Color: Red/
    Carats: 1.4/ Price: 20000$/ Type: Precious</p>
  </li>
```

- The **list item** from preview section must be **removed** while the **["Add Gem"]** button is **enabled** again.

4. Cancel the information from the Preview

- When the **"Cancel"** button is clicked, the **list item** must be **removed** from the **"preview-list"** and the **["Add Gem"]** button is **enabled** again**.**

```
<h3>Preview</h3>
<ul id="preview-list"> </ul>
</div>
```

Submission

Submit only yours **solution()** function.

GOOD LUCK... ☺