JAVASCRIPT DOM AND EVENTS

EXERCISE ONE: ES6 LOOPS, STRING MANIPULATION, ARRAYS, FUNCTIONS, DOM, DEFAULT FUNCTION ARGUMENTS, CONDITIONS

`capitalize.js`

- Create a function named `capitalize` that takes two parameters:
    - `sentence`: The sentence to be capitalized.
    - `lowercaseArticles`: A boolean value indicating whether to lowercase articles in the
      middle of the sentence. Make this parameter optional and set its default value to `true`.
- Note: In the solutions there is special handling for the strings "JavaScript" and “HTML5." Test by including the file in any of the HTML files and changing the case of the `h2` tags.
- Hint: `function capitalize(sentence, lowercaseArticles = true) { ... }`
- Make the sentence lowercase and split into words: Inside the `capitalize` function, convert
  the `sentence` parameter to lowercase and split it into an array of words using the `split()` method
  with " " as the delimiter. Store the resulting array in a variable named `words`.
- Hint: `const words = sentence.toLowerCase().split(" ");`
- Define articles, words that will not get uppercased: Create an array called `articles` that contains all the common
  articles in English: "the", "a", "an", "and", "but", "or", "for", "nor", "so", "yet", "as", "at", "by", "for",

"in", "of", "off", "on", "per", "to", "up", and "via".

- Hint (you can copy): `const articles = ["the", "a", "an", "and", "but", "or", "for", "nor", "so", "yet", "as", "at", "by", "for", "in", "of", "off", "on", "per", "to", "up", "via"];`

- Capitalize words: Use the `map()` method to iterate over each word in the `words` array and apply a capitalization function. The capitalization function should:

  - Always capitalize the first word of the sentence (`index === 0`).

Only make lowercase articles in the middle of the sentence if `lowercaseArticles` is `true`. • Hint:

```
const capitalizedWords = words.map((word, index) => {

            if (index === 0) {

          return word.charAt(0).toUpperCase() + word.slice(1);
            }

        if (lowercaseArticles && articles.includes(word)) {
              return word;

}
        return word.charAt(0).toUpperCase() + word.slice(1);
          });
```

Join the capitalized words: Join the capitalized words back into a

sentence using the join()
method with " " as the delimiter.
`Hint: return capitalizedWords.join(" ");`

- Use the function: Select an element with the tag name `h1` using `document.querySelector("h1")` and store it in a variable (e.g., `h1Element`). Hint: `const h1Element = document.querySelector("h1");`

- Call the function with two different options:

  - To have lowercase articles in the middle of the sentence:
    `h1Element.textContent = capitalize(h1Element.textContent, true);`

  - To capitalize all words except the first word:
    `h1Element.textContent = capitalize(h1Element.textContent, false);`

  - Another way to have lowercase articles in the middle -default behavior: `h1Element.textContent = capitalize(h1Element.textContent);`

EXERCISE TWO: LOOPS, Array, querySelector, DOM, EVENTS, ARROW

`filter.js` for the table in `about.html` — examine the HTML tags

1. Start with the event listener:

- Attach an event listener to the element with the ID "search". Listen specifically for the "keyup" event, which fires each time the user releases a key after typing.
- The `() => { ... }` part is an anonymous function that gets executed whenever the "keyup" event occurs.

Hint:
`document.querySelector("#search").addEventListener("keyup", () => { ... })`

2. Get search text:

- Retrieve the current value of the element with the ID "search" (presumably an input field).
- Convert the value to uppercase using the `toUpperCase()` method. This ensures case-insensitive searching.
- The retrieved value is stored in the variable `filter`.

Hint:

```
const filter =
document.querySelector("#search").value.toUpperCase();
```

3. Get table rows:

- Retrieve all `<tr>` (table row) elements inside the first `<table>` (table) element on the page.
- Store them in the variable `tableRows`.

Hint:

```
const tableRows =
document.querySelector("table").querySelectorAll("tr");
```

4. Looping through rows:

- Iterate through each row in the `tableRows` collection.
- Uses the `forEach` method of the `Array.from()` function.
  - `Array.from()` converts the `tableRows` collection (which might not be a true array) into a proper array. This allows us to use array methods like `forEach`.
- Inside the `forEach` loop, the variable `row` will represent the current table row being processed.

Hint:

```
Array.from(tableRows).forEach(row => { ... })
```

5. Search each row (Two Options):

A. Search all fields:

```
row.style.display = (cells.length &&
row.innerHTML.toUpperCase().includes(filter)) ? "" :
"none";`
```

Explanation: This line checks if there are any cells (<td>) within the current row (row.length). If there are cells, it then checks if the entire row's content (converted to uppercase using `toUpperCase()`) includes the search term (filter) using the includes() method. Based on the condition: If the search term exists and there are cells (cells.length && row.innerHTML.toUpperCase().includes(filter)), the row's display style is set to an empty string (""), making it visible. Otherwise, the row's display style is set to "none", effectively hiding it.

B. Search name field only: (Commented out in solutions.)

```
// row.style.display = (cells.length &&
cells[0].innerHTML.toUpperCase().includes(filter)) ? "" :
“none";
```

Explanation: This commented-out line demonstrates searching only the first cell (presumably containing the name) of each row. It follows the same logic as the uncommented line, but checks if the search term exists only within the content of the first cell (cells[0]).

EXERCISE THREE: KEY CODES AND DISABLING CUT, COPY AND PASTE: `contact.js` for the contact form

We will write functionality to disable cut, copy, and paste actions on all `<input>` elements on the page and provide visual feedback through an error message.

1. Select all `<input>` elements:

Use `document.querySelectorAll` to select all elements on the page with the tag name "input". Store these elements in a constant variable named `inputs`.

Hint:

```
const inputs = document.querySelectorAll("input");
```

2. Loop through each input element:

Use the `forEach` method on the `inputs` variable. The `forEach` method iterates through each element in the `inputs` array and executes the provided function (represented by the arrow function) on each element. The function takes a single argument, `input`, which represents the current `<input>` element being processed.

```
inputs.forEach(input => {
  // Code we are going to write to handle individual input
element
});
```

3. Disable cut, copy, and paste actions inside the loop:

- Create an array containing the events: `"cut"`, `"copy"`, and `"paste"`.
- Loop through this array using `forEach` again. For each event type (like `"cut"`), add an event listener to the current `input` element.
- When the specific event (e.g., "cut") occurs, the provided function is executed.
- Inside this function, use `event.preventDefault()` to prevent the default behavior of the event. This means the browser won't perform the cut, copy, or paste action by default.

```
["cut", "copy", "paste"].forEach(eventType => {
  input.addEventListener(eventType, (event) => {
    event.preventDefault();
    // Rest of the code handles error message
  });
});
```

4. Handle potential error messages inside the loop:

Set a variable `errorMessage` that initially holds `null`. This variable will store a reference to the error message element if one is created.

Inside other parts of the code, we'll have checks for `errorMessage`. If it's not `null`, it means an error message is currently displayed. In this case, the existing message is removed using `errorMessage.remove()`, and the variable is set back to `null` to prepare for a new message.

```
  let errorMessage = null;

  if (errorMessage) {
    errorMessage.remove();
    errorMessage = null;
  }
```

5. Handle Enter and ArrowRight key presses:

Here we set “hotkeys.” We will listen for the "keydown" event on the `input` element. When a key is pressed down, the function checks the pressed key using `event.key`. If it's either "Enter" or "ArrowRight", the function calls `this.blur()` on the `input` element. This removes the focus from the input field, allowing the error message to disappear if it was displayed.

```
input.addEventListener("keydown", (event) => {
  if (event.key === "Enter" || event.key === "ArrowRight")
{
    this.blur(); // Removes focus from the input
  }
});
```

6. Handle input events and error message:

Listen for the "input" event on the input field. This event fires whenever the user types or makes any changes to the input value.

- If an error message is currently displayed (`errorMessage` is not `null`), remove the message and clears the `errorMessage` variable.
- This is a way for the error message to disappear when the user corrects their input and the restriction is no longer needed.

```
input.addEventListener("input", () => {
  if (errorMessage) {
    errorMessage.remove();
    errorMessage = null;
  }
});
```

7. Show and style the error message that slides down (showError function):

```
function showError(input, eventType) {
  const errorMessage = document.createElement('div');
  errorMessage.textContent = `${eventType} is disabled in
this field`;
  errorMessage.classList.add('error-message');

  input.parentNode.insertBefore(errorMessage,
input.nextSibling);
```

```
  errorMessage.style.display = 'block';
  errorMessage.style.height = '0';
  errorMessage.style.color = 'red';
  errorMessage.style.transition = 'height 0.3s ease';

  setTimeout(() => {
    errorMessage.style.height = errorMessage.scrollHeight +
'px';
  }, 0);

  return errorMessage;
}
```