Promise.Finally... Finally

Most languages that I've worked with have an additional tool for promises that covers items that we have to do at the end of a task, regardless of whether it succeeds or not.

Using this example we'll explore uses for `finally` and compare with using `async / await` for the same task.

`fetchAndDisplay` takes two parameters, a URL and the element you want to insert the content into.

The original version does the following:

- shows a loading spinner to indiccate the content is loading
- Fetches the URL
- If the fetch succeeds
    - Insert the text into the element
    - Hide the loading spinner
- If it fails
    - Insert an error message into the text element
    - Hide the loading spinner

This is a short example but it shows one of the reasons why we need a `finally` block. the `hideLoadingSpinner` function is called in muliple locations of `fetchAndDisplay`. In this case it's simple but you can imagine the potential damage if we were cleaning data after we complete a large transaction, whether success or failure.

```
const fetchAndDisplay = ({ url, element }) => {
  showLoadingSpinner();
  fetch(url)
    .then((response) => response.text())
    .then((text) => {
      element.textContent = text;
      hideLoadingSpinner();
    })
    .catch((error) => {
      element.textContent = error.message;
```

```
      hideLoadingSpinner();
    });
  };

fetchAndDisplay({
  url: someUrl,
  element: document.querySelector('#output')
});
```

The `finally` block takes care of the code dupplication. It will run once whether the promise fulfills or rejects. Since we want to hide the spinner regardless of whether it succeeds or fails we put it in the `finally` block. The code now looks like this.

```
const fetchAndDisplay = ({ url, element }) => {
  showLoadingSpinner();
  fetch(url)
    .then((response) => response.text())
    .then((text) => {
      element.textContent = text;
    })
    .catch((error) => {
      element.textContent = error.message;
    })
    .finally(() => {
      hideLoadingSpinner();
    });
};
```

We can also use the `async`/`await` to do the same thing with the full `try`/`catch`/`finally` blocks; taking into account that we still want to use `hideLoadingSpinner` only once.

```
const fetchAndDisplay = async ({ url, element }) => {
  showLoadingSpinner();
  try {
```

```
    const response = await fetch(url);
    const text = await response.text();
    element.textContent = text;
  } catch (error) {
    element.textContent = error.message;
  } finally {
    hideLoadingSpinner();
  }
};
```