JavaScript Form Validation Review

Validating Forms with JavaScript

- **Constraint Validation API**: Certain HTML elements, such as the textarea and input elements, expose a constraint validation API. This API allows you to assert that the user's provided value for that element passes any HTML-level validation you have written, such as minimum length or pattern matching.
- **checkValidity() method**: This method returns true if the element matches all HTML validation (based on its attributes), and false if it fails.

```
const input = document.querySelector("input");


input.addEventListener("input", (e) => {
  if (!e.target.checkValidity()) {
    e.target.setCustomValidity("You must use a .com email.")
  }
})
```

- **reportValidity() Method**: This method tells the browser that the input is invalid.

```
const input = document.querySelector("input");


input.addEventListener("input", (e) => {
  if (!e.target.checkValidity()) {
    e.target.reportValidity();
  }
})
```

- **validity Property**: This property is used to get or set the validity state of form controls (like <input>, <select>, etc.) and provides information about whether the user input meets the constraints defined for that element (e.g., required fields, pattern constraints, maximum length, etc.).

```
const input = document.querySelector("input");


input.addEventListener("input", (e) => {
  console.log(e.target.validity);
})
```

- **patternMismatch Property**: This will be true if the value doesn't match the specified regular expression pattern.

preventDefault() Method

- **Definition**: Every event that triggers in the DOM has some sort of default behavior. The click event on a checkbox toggles the state of that checkbox, by default. Pressing the space bar on a focused button activates the button.
  The preventDefault() method on these Event objects stops that behavior from happening.

```
button.addEventListener('click', (event) => {
  // Prevent the default button click behavior
  event.preventDefault();
  alert('Button click prevented!');
});
```

Submitting Forms

- **Definition**: There are three ways a form can be submitted. The first is when the user clicks a button in the form which has the type attribute set to submit. The second is when the user presses the Enter key on any editable input field in the form. The third is through a JavaScript call to the requestSubmit() or submit() methods of the form element.
- **action Attribute**: The action attribute should contain either a URL or a relative path for the current domain. This value determines where the form attempts to send data - if you do not set an action attribute, the form will send data to the current page's URL.

```
<form action="https://freecodecamp.org">
  <input
  type="number"
  id="input"
  placeholder="Enter a number"
  name="number"
 />
 <button type="submit">Submit</button>
</form>
```

- **method Attribute**: This attribute accepts a standard HTTP method, such as GET or POST, and uses that method when making the request to the action URL. When a method is not set, the form will default to a GET request. The data in the form will be URL encoded as name=value pairs and appended to the action URL as query parameters.

```
<form action="/data" method="POST">
 <input
  type="number"
  id="input"
  placeholder="Enter a number"
  name="number"
 />
 <button type="submit">Submit</button>
</form>
```

- **enctype Attribute**: The form element accepts an enctype attribute, which represents the encoding type to use for the data. This attribute only accepts three values: application/x-www-form-urlencoded (which is the default, sending the data as a URL-encoded form body), text/plain (which sends the data in plaintext form, in name=value pairs separated by new lines), or multipart/form-data, which is specifically for handling forms with a file upload.