**1. How to validate form inputs in JavaScript?**

To validate form inputs in JavaScript, you typically attach a submit event listener to the form. Inside this event handler, you first call event.preventDefault() to stop the default submission. Then, you access the value of each input field, use if/else statements to check if the data meets your criteria (e.g., if it's not empty, if it's a number), and if any checks fail, you display an error message and prevent the form from proceeding.

**2. What is event.preventDefault()?**

event.preventDefault() is a JavaScript method that stops the browser's default action for a specific event. For a form, the default action is to submit the data to the URL in the action attribute and reload the page. By calling event.preventDefault(), you can run your own JavaScript validation and logic without the page reloading. This is crucial for providing immediate user feedback without disrupting their experience.

**3. How to check email format with regex?**

You can check an email's format using a **regular expression (regex)**, which is a powerful pattern-matching tool. A common regex pattern for emails looks for a sequence of characters, followed by an @ symbol, followed by more characters and a dot (.), and then a domain. In JavaScript, you define the pattern and then use the .test() method to see if a string matches it.

JavaScript

const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

const isValid = emailRegex.test('test@example.com'); // returns true

const isInvalid = emailRegex.test('test-at-example'); // returns false

**4. Difference between client-side and server-side validation?**

* **Client-side validation** happens in the user's browser, typically using JavaScript. It provides immediate feedback to the user, improving the user experience. However, it's not foolproof, as malicious users can bypass it.
* **Server-side validation** happens on the server after the form is submitted. It is the most important type of validation because it protects the database and application logic from malicious data.

Both are essential. Client-side validation is for a good user experience, while server-side validation is for robust security and data integrity.

**5. How to show error messages dynamically?**

To show error messages dynamically, you first create a hidden HTML element (like a <p> or <span>) for each potential error. In JavaScript, you can check if an input is invalid. If it is, you can remove a CSS class like hidden or display: none from the corresponding error element, making it visible. For valid inputs, you ensure the error element remains hidden.

**6. What is form submission?**

Form submission is the process where a browser sends data from a web form to a web server. This happens when a user clicks a submit button or presses Enter in a form. The data is packaged up and sent as an HTTP request (usually a POST or GET request) to the URL specified in the form's action attribute.

**7. How to improve form accessibility?**

Improving form accessibility makes your form usable for everyone, including people with disabilities who use assistive technologies like screen readers. Key practices include:

* Using <label> tags with a for attribute that matches the input's id. This links the label to the input, making it readable for screen readers.
* Providing clear and concise error messages.
* Ensuring the form can be fully navigated and submitted using only a keyboard (Tab key).
* Using appropriate HTML5 input types, like type="email", so browsers can offer better mobile keyboards and built-in validation.

**8. How to handle form reset?**

A form can be reset in two main ways:

1. Using a <button type="reset"> element, which, when clicked, clears all form inputs back to their initial values.
2. In JavaScript, you can programmatically reset a form by calling the .reset() method on the form element (e.g., myForm.reset()). This is useful for clearing the form after a successful submission.

**9. What are common security issues with forms?**

Common security issues with forms arise when user-submitted data is not properly validated or sanitized on the server-side. The most critical issues are:

* **Cross-Site Scripting (XSS):** Injecting malicious scripts into a form that other users might later execute.
* **SQL Injection:** Injecting malicious SQL commands to gain access to or modify a database.
* **Cross-Site Request Forgery (CSRF):** Tricking a user into submitting a form with malicious data from a different website.

These threats are primarily mitigated by **robust server-side validation and sanitization.**

**10. How does HTML5 built-in validation differ from JS validation?**

* **HTML5 built-in validation** is a declarative approach using attributes directly in the HTML, like required, type="email", minlength, and pattern. It's easy to implement but offers minimal control over the appearance and content of error messages.
* **JavaScript validation** is a programmatic approach that gives you complete control over the validation logic, the styling of error messages, and the overall user experience. It requires more code but provides maximum flexibility