

Form Validation

In this lesson, we will go over the concept of form validation.

WE'LL COVER THE FOLLOWING




- Informing users
- Listing 4-7: Using `placeholder`

When you collect information on a web page, nothing prevents your users from providing something different than what you ask for. For example, they may leave a textbox empty, type names where you expect numbers, provide a phone number instead of an e-mail address, and so on. When they click the submit button, all data goes to the server as it is filled out.

If your server-side does not validate the data, it could get wrong information which leads to errors and confusion. It is trivial that the server-side checks all data coming from the browser, the browser—the server-side *must not trust* that the client-side sends valid data.

However, sending invalid data to the server-side consumes resources: the form data goes to the network, it reaches the server, the server carries out validation procedures, and it sends back an answer saying that something is wrong with the input.

Resources consumed at the server-side cost more than resources spent at the client (browser) side from the aspect of the service provider. If invalid data could be held back and not sent to the server, it would save resources. That is the idea behind client-side validation.

 **NOTE:** I should emphasize again that the server must validate all data, even if the client does it, too. So, when client-side validation is mentioned, it means validation in two places: at the client and at the

server.

Client-side validation is not a new idea. It existed before HTML5 and it required JavaScript. The creators of HTML5 recognized the importance of this topic, and they provided a way to free web developers from writing most of the validation scripts. They achieved this aim with HTML5 markup, and with a set of JavaScript functions.

Informing users

In many cases, users type invalid data because they do not know exactly what a specific form field expects. The `<input>` element has an attribute, `placeholder`, that specifies a short hint that describes the expected value of the field. By using a `placeholder`, you can help the user figure out what you expect.

Listing 4-7 shows you an example.

Listing 4-7: Using `placeholder`

```
<!DOCTYPE html>

<html>
<head>
  <title>Using placeholder</title>
  <style>
    ::-webkit-input-placeholder {
      font-style: italic;
    }

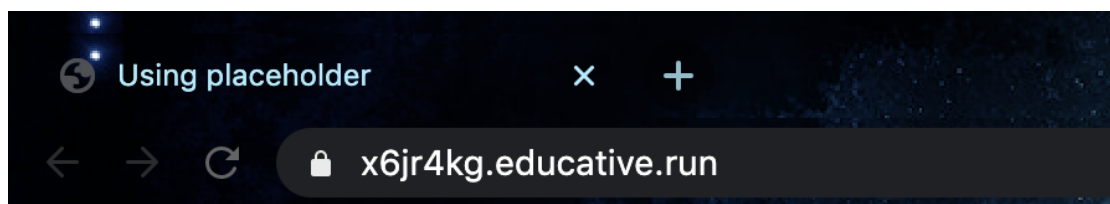
    :-moz-placeholder { /* Firefox 18- */
      font-style: italic;
    }

    :-moz-placeholder { /* Firefox 19+ */
      font-style: italic;
    }

    :-ms-input-placeholder {
      font-style: italic;
    }
  </style>
</head>
<body>
  <h2>Specify your booking info</h2>
  <form>
    <label>
      First name:
      <input type="text"
        placeholder="John"
```

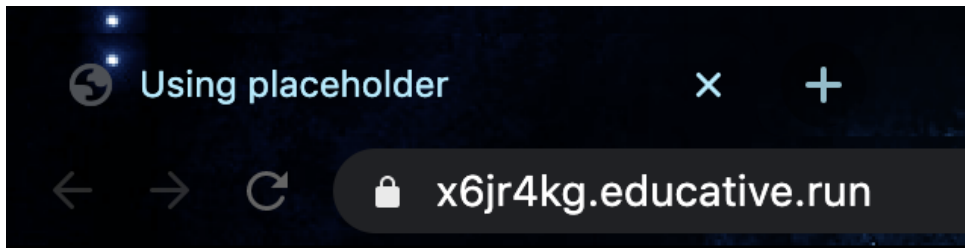
```
        autocomplete="on" />
    </label>
    <br />
    <label>
        Last name:
        <input type="text"
            placeholder="Smith" />
    </label>
    <br />
    <label>
        Membership ID:
        <input type="text"
            placeholder="ABC-123" />
    </label>
    <br />
    <input type="submit" value="Query" />
</form>
</body>
</html>
```

When the page is displayed, you can see that each input field shows the placeholder text (image below). The placeholder, often called a watermark, is dimmed to tell the user it is not the actual content of the field, only a hint.



Placeholders help users to guess out the content and format of a field.

When you type data into a field, the placeholder is removed, and the real content is shown (image below).



Specify your booking info

First name:

Last name:

Membership ID:

Placeholders are removed when you specify the value of a field

With properly designed placeholders you can even suggest the format of an input. For example, the Membership ID’s “ABC-123” placeholder implicitly tells the user that this field expects an ID that starts with three letters, goes on with a dash, and concludes with three digits.

In Listing 4-7, I made a little styling trick that italicizes the font of the placeholder, and added the following style definition to the `<head>` section:

Index.html

```
<style>
::-webkit-input-placeholder {
  font-style: italic;
}

:-moz-placeholder { /* Firefox 18- */
  font-style: italic;
}

::-moz-placeholder { /* Firefox 19+ */
  font-style: italic;
}

:-ms-input-placeholder {
  font-style: italic;
}
```

Later in this course you'll learn how it works.

NOTE: The `<input>` element has an attribute, `autocomplete`, that allows the browser to predict the value. When a user starts to type in a field, the browser can display options to fill in the field based on earlier typed values. Generally, you have to allow this option in your browser's settings.

Now that we have covered the basics of form validation, we will move on to study validation attributes in the *next lesson*.

Stay tuned! :)