Experiment No. 4

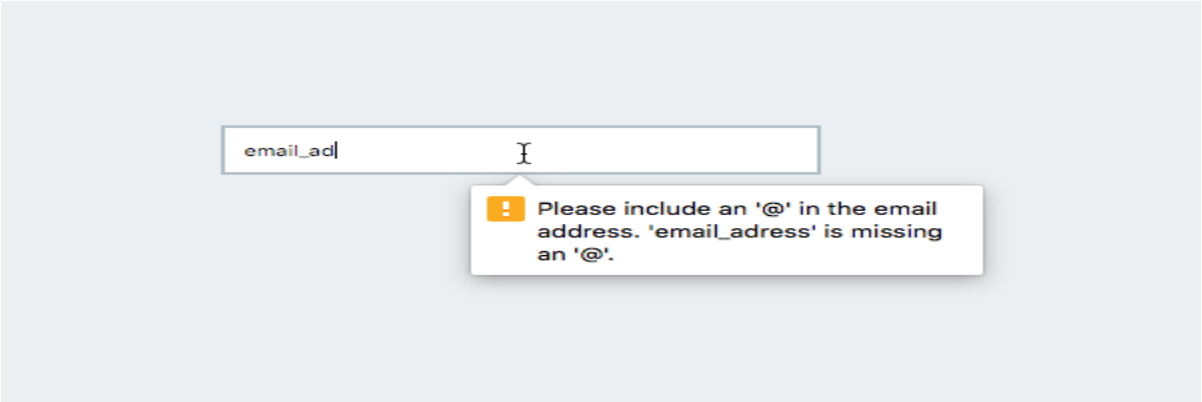
1. **Apply validations (not null, min, max, multiple of n, read only and disabled, Pattern length, domain,Overriding submit, formnovallidate) on form controls.**

Illustration:

Validation

Form validation is of vital importance to a web site's security as well as its usability. The validation process evaluates whether the input value is in the correct format before submitting it. For example, if we have an input field for an email address, the value must certainly contain a valid email address; it should start with a letter or a number, followed by the @ symbol, then end with a domain name.

The HTML5 specification has made validation that bit easier with the introduction of new input types such as email, url, and tel, and these also come packaged up with predefined validation. Whenever the value given is not met with the expected formatting, these input types will throw an error message thus preventing submission.



For Example- Invalid email address error message (Chrome)

Expecting every possible input scenario to be catered for is impractical, however. What if you have a username, zip code, or any special data types that are not specified as standard input types? How do we validate those inputs? This is where the attribute pattern comes into play.

1. Using the Pattern Attribute

The pattern attribute is only applicable to the input element. It allows us to define our own rule to validate the input value using Regular Expressions. Again, if the value does not match the specified pattern, the input will throw an error.

For example, say we have a username input in our form. There isn't a standard type for username, hence we use the regular text input type:

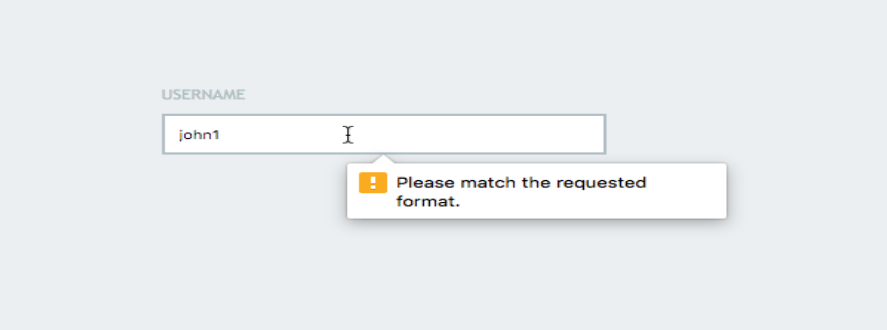
```
1<form action="somefile.php">
2    <input type="text" name="username" placeholder="Username">
3</form>
```

Let's define a rule to add using the pattern attribute. In this case, we'll specify that the username should only consist of lowercase letters; no capital letters, numbers or other special characters allowed. In addition, the username length shouldn't be more than 15 characters. In RegEx, this rule can be expressed as [a-z]{1,15}.

Add [a-z] {1, 15} as the value of the pattern attribute in our username input:

```
1    <form action="somefile.php">
2        <input type="text" name="username" placeholder="Username" pattern="[a-z]{1,15}">
3    </form>
```

Now, since it only accepts lowercase letters, submitting any other value will throw an error message:



For example- The error message, stating the pattern is not matched

As you can see above, the error message says "Please match the requested format." So validation is working, but this message doesn't help users to understand what the requested format actually is. To understand requested format you put the domain values on the form. (Refer Point **10**)

The **pattern** attribute specifies a regular expression that the <input> element's value is checked against. The pattern attribute works with the following input types: text, search, url, tel, email, and password.

Note: Use the global **title** attribute to describe the pattern to help the user.

Note: Learn about **regular expressions** in JavaScript.

An input field that can contain only three letters (no numbers or special characters):

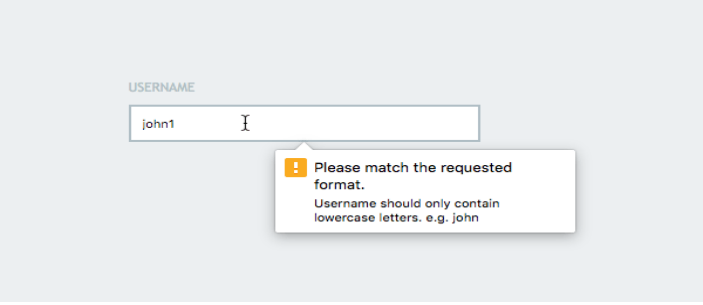
Country code: <input type="text" name="country_code" pattern="[A-Za-z]{3}" title="Three letter country code">

A. Customizing the Validation Message

Fortunately, we can customize the message to be more helpful, and we have a couple of ways to do so. The easiest approach is to specify a title attribute within our input element:

```
<form action="somefile.php"
  <input
    type="text"
    name="username"
    placeholder="Username"
    pattern="[a-z]{1,15}"
    title="Username should only contain lowercase letters. e.g. john">
</form>
```

Now the title is included along with the default message: Still, the popup message is inconsistent. If we compare it with the one thrown by the email input type shown earlier, the actual instructions could be more prominent.



For example-title with pop up message

The second approach will solve this.

B. Replacing the Default Validation Message

Let's now replace the default "Please match the requested format" with a completely customized message. We'll use a bit of JavaScript to do this.

Begin by adding an id to the input element, so as to be able to select it conveniently.

```
<form action="somefile.php">
  <input
    type="text"
    name="username"
    placeholder="Username"
    pattern="[a-z]{1,15}"
    id="username">
</form>
```

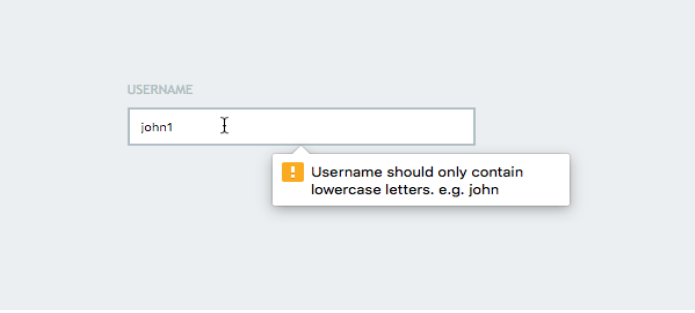
Now, we can select the input element using JavaScript and assign it to a variable (either between script tags on our page, in a separate JavaScript file, or in the JS pane on CodePen):

```
1  var input = document.getElementById('username');
```

Lastly, we specify the message used when the input shows its invalid state.

```
1  input.oninvalid = function(event) {
2      event.target.setCustomValidity('Username should only contain lowercase letters. e.g. john');
3  }
```

The oninvalid event inherits an event object which contains a few properties, including the target property (the invalid element) and the validationMessage which contains the error text message. In the example above, we have overidden the text message using the setCustomValidity() method. Find the custom message replacing the default.



For example- Pop-Message without title.

C. Styling

To complement the new input types and these methods for setting a custom validation message, the CSS3 spec brings a couple of useful pseudo-classes, :valid and :invalid. These enable us to apply styles depending on the input validity state, for example:

```
input:invalid {
   border-color: red;
}
input,
input:valid {
   border-color: #ccc;
}
```

HTML:

```
<form id="form">
   <div>
      <label for="username">Username</label>
      <input name="username" type="text" placeholder="Username" pattern="[a-z]{1,15}"
      id="username">
   </div>
</form>
```

CSS:

```
html {
      box-sizing: border-box;
   }   box-sizing: inherit;
   }

   body {
      font-family: "Trebuchet MS", "Lucida Grande", "Lucida Sans Unicode",
            "Lucida Sans", Tahoma, sans-serif;
      background-color: #ECEFF1;
   }
```

```
form {
   max-width: 300px;
   margin-top: 60px;
   margin-right: auto;
   margin-left: auto;
   text-align: left;
   position: relative;
   padding-top: 80px;
}
label,
input {
      display: block;
}
label {
   font-size: 12px;
   text-transform: uppercase;
   color: #B0BEC5;
   margin-bottom: 10px;
}
input {
      width: 100%;
      padding: 10px;
      outline: 0;
      border: 2px solid #B0BEC5;
}
input:invalid {
      border-color: #DD2C00;
}

#notify {
   margin-top: 5px;
```

```
    padding: 10px;
    font-size: 12px;
    width: 100%;
    color: #fff;
    display: block;
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
}
#notify.error {
    background-color: #DD2C00;    }
```

There are several things to keep in mind when using these pseudo-classes:

- First, the :valid is applied by default, even when the input value is empty. Thus, as you can see above, we set the border-color to #ccc; the default color given to our input element. The empty value is always considered valid, unless we have added the required attribute. In that case, the input is invalid and the red border color is given.
- The valid and the invalid styles apply *immediately* as the user is typing, even when the value is empty. Instant style changes may then throw the users into a panic.

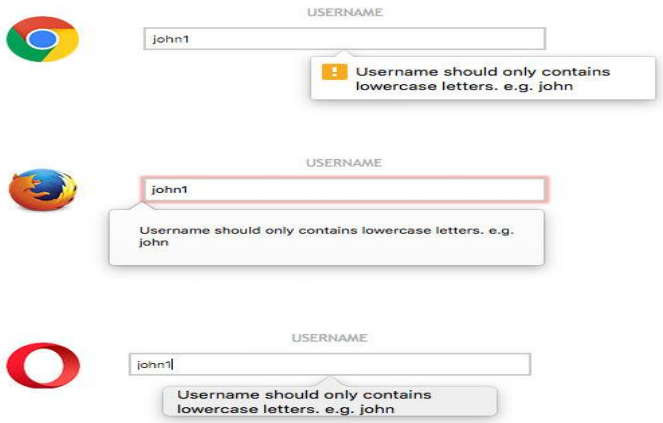


Fig.-Styling of Pop up Messages

C.1 Styling the Popup Message

Form validation has become a new standard as per the HTML5 specification, yet how the error popup appears is entirely up to the browser vendor. Expect different aesthetics in different browsers, which won't help the consistency of User Interface.

Apply Validations (for not null, min, max, multiple of n, read only and disabled, Pattern length, domain, Overriding submit, form no validate) on form controls.

2. Form Validate/No validate:

Google Chrome prevented the ability to customize the default popup styles a few years ago. If this is something you want to achieve, the only remaining option is to entirely override the popup message using JavaScript.

```
<!DOCTYPE html>
<html>
<body>
<form action="/action_page.php" novalidate> OR validate
E-mail: <input type="email" name="user_email">
<input type="submit">
</form>
<p><strong>Note:</strong> The novalidate attribute of the form tag is not supported in Internet Explorer 9 and earlier versions, or in Safari.</p>
</body>
</html>
```

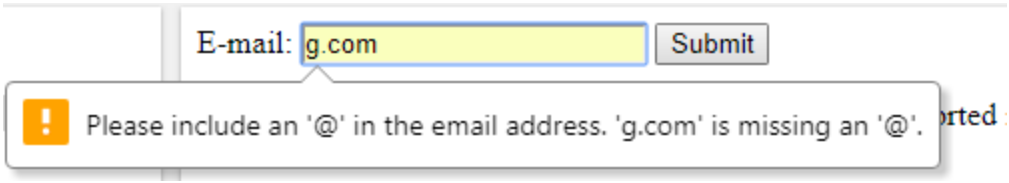


Fig. Form Validation

3. The maxlength Attribute

The **maxlength** attribute specifies the maximum allowed length for the input field:

```
<form action="">
First name:<br>
<input type="text" name="firstname" maxlength="10">
</form>
```

With a maxlength attribute, the input field will not accept more than the allowed number of characters.

The maxlength attribute does not provide any feedback. If you want to alert the user, you must write JavaScript code.

4. The disabled Attribute

The **disabled** attribute specifies that the input field is disabled.

A disabled input field is unusable and un-clickable, and its value will not be sent when submitting the form:

Example:

```
<form action="">
First  name:<br>
<input type="text" name="firstname" value="John" disabled>
</form>
```

5. The readonly Attribute

The **readonly** attribute specifies that the input field is read only (cannot be changed):

Example:

```
<form action="">
First name:<br>
<input type="text" name="firstname" value="John" readonly>
</form>
```

6. The min and max Attributes

The **min** and **max** attributes specify the minimum and maximum values for an <input> element.

The min and max attributes work with the following input types: number, range, date, datetime-local, month, time and week.

<input> elements with min and max values:

Enter a date before 1980-01-01:
<input type="date" name="bday" max="1979-12-31">

Enter a date after 2000-01-01:
<input type="date" name="bday" min="2000-01-02">

Quantity (between 1 and 5):
<input type="number" name="quantity" min="1" max="5">

7. The multiple Attribute

The **multiple** attribute specifies that the user is allowed to enter more than one value in the <input> element.

The multiple attribute works with the following input types: email, and file.

A file upload field that accepts multiple values:

Select images: <input type="file" name="img" multiple>

8. The step Attribute

The **step** attribute specifies the legal number intervals for an <input> element.

Example: if step="3", legal numbers could be -3, 0, 3, 6, etc.

Note: The step attribute can be used together with the max and min attributes to create a range of legal values.

The step attribute works with the following input types: number, range, date, datetime-local, month, time and week.

Example:

An input field with specified legal number intervals:

<input type="number" name="points" step="3">

9. The Non Empty field validation

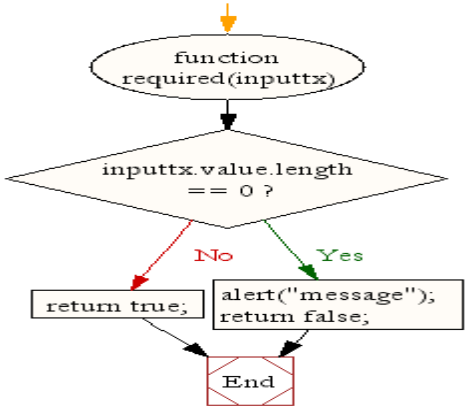


Figure. Flow graph non empty field validation

HTML Code

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JavaScript form validation - checking non-empty</title>
<link rel='stylesheet' href='form-style.css' type='text/css' />
</head>
<body>
<div class="mail">
<h2>Input your Name and Submit</h2>
<form name="form1" action="#" onsubmit="required()">
<ul>
<li><input type='text' name ='text1'/></li>
<li class="rq">*Required Field</li>
<li><input type="submit" name="submit" value="Submit" /></li>
</ul>
</form>
</div>
<script src="non-empty.js"></script>
```

```
</body>
</html>
```

JavaScript Code

```
function required()
{
var empt = document.forms["form1"]["text1"].value;
if (empt == "")
{
alert("Please input a Value");
return false;
}
else
{
alert('Code has accepted : you can try another');
return true;
}
}
```

CSS Code

```
li {list-style-type: none;
font-size: 16pt;
}
.mail {
margin: auto;
padding-top: 10px;
padding-bottom: 10px;
width: 400px;
background : #D8F1F8;
border: 1px soild silver;
}
```

```
.mail h2 {
margin-left: 38px;
}
input {
font-size: 20pt;
}
input:focus, textarea:focus{
background-color: lightyellow;
}
input submit {
font-size: 12pt;
}
.rq {
color: #FF0000;
font-size: 10pt;
}
```

Another function to check whether a field is empty

```
function Emptyvalidation(inputtxt)
    {
 if (inputtxt.value.length == 0)
    {
 document.inputtxt.style.background =   'Yellow';
    }
 else
    {
 document.inputtxt.style.background = 'White';
    }
 return error;
    }
```

In the above function background color of the input field will be yellow if the user input a blank field otherwise the background color will be white.

Flowchart:

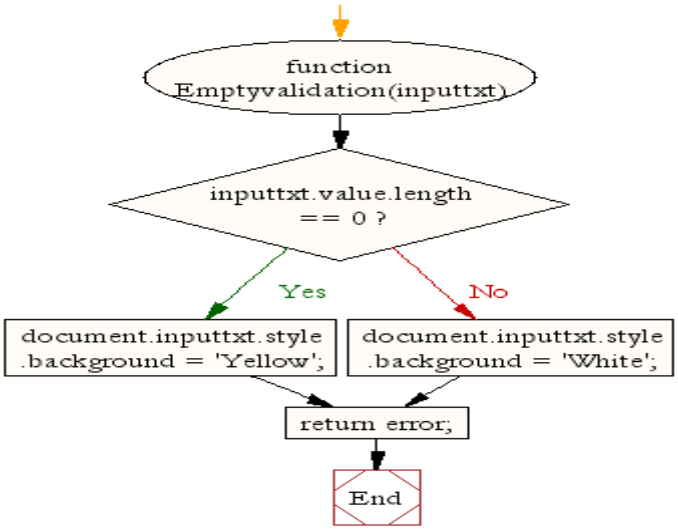


Figure. Flow graph non empty field validation with background style

10. Checking string length : Validation

```
Function lengthRange(inputtxt, minlength, maxlength)
{
  var userInput = inputtxt.value;
  if(userInput.length >= minlength && userInput.length <= maxlength)
    {
     return true;
    }
  else
    {
        alert("Please input between " +minlength+ " and " +maxlength+ " characters");
     return false;
    }
}
```

Flowchart:

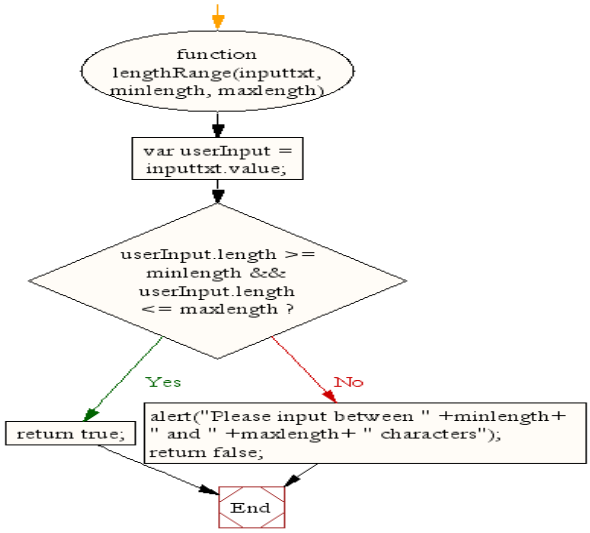


figure. Flow graph for checking string length

An example of the above function for a field that requires 6 to 8 characters to valid a usercode.

HTML Code

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JavaScript form validation - checking non empty</title>
<link rel='stylesheet' href='form-style.css' type='text/css' />
</head>
<body onload='document.form1.text1.focus()'>
<div class="mail">
<p>Enter Userid [between 6 to 8 characters] and Submit</p>
<form name="form1" action="#">
<ul>
<li>Username:<input type='text' name='text1'/></li>
<li> </li>
<li class="submit"it"><input type="submit" name="submit" value="Submit"
onclick="stringlength(document.form1.text1,6,8)"/></li>
```

```
<li> </li>
</form>
</ul>
</div>
<script src="string-lenght.js"></script>
</body>
</html>
```

Javascript code

```
function stringlength(inputtxt, minlength, maxlength)
{
var field = inputtxt.value;
var mnlen = minlength;
var mxlen = maxlength;

if(field.length<mnlen || field.length> mxlen)
{
alert("Please input the userid between " +mnlen+ " and " +mxlen+ " characters");
return false;
}
else
{
alert('Your userid have accepted.');
return true;
}
}
```

CSS Code

```
li {list-style-type: none;
font-size: 16pt;
```

```
}
.mail {
margin: auto;
padding-top: 10px;
padding-bottom: 10px;
width: 400px;
background : #D8F1F8;
border: 1px soild silver;
}
.mail h2 {
margin-left: 38px;
}
input {
font-size: 20pt;
}
input:focus, textarea:focus{
background-color: lightyellow;
}
input submit {
font-size: 12pt;
}
.rq {
color: #FF0000;
font-size: 10pt;
}
```

11. Override on Manual Submit

onSuccess(form) & onError(form)

These methods are called when the form validation is done.

They can return promises to wait on submit.

Define an object with `onSuccess(form)` or `onError(form)` Validation Hooks.

```
const hooks = {
  onSuccess(form) {
    alert('Form is valid! Send the request here.');
    // get field values
    console.log('Form Values!', form.values());
  },
  onError(form) {
    // get all form errors
    console.log('All form errors', form.errors());
    // invalidate the form with a custom error message
    form.invalidate('This is a generic error message!');
  },
};
```

Take the `form` instance as parameter in input.

Then pass the Validation Hooks as first argument to the `submit({ ... })` Action:

```
instance.submit({
  onSuccess: hooks.onSuccess,
  onError: hooks.onError,
})
```

or as second argument to the `onSubmit(e, { ... })` Event Handler:

```
<button
  type="submit"
  onClick={e => instance.onSubmit(e, {
    onSuccess: hooks.onSuccess,
    onError: hooks.onError,
  })}
>Submit</button>
```

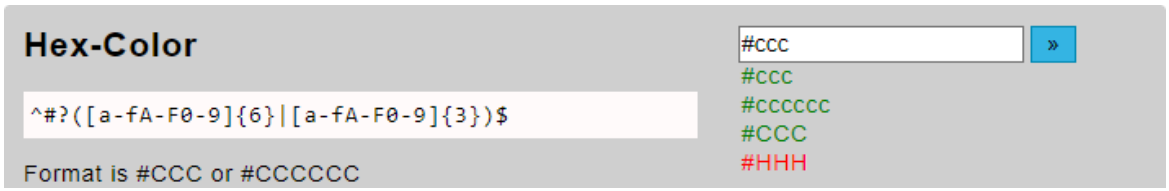
`instance` can be a **Form** or a **Field**

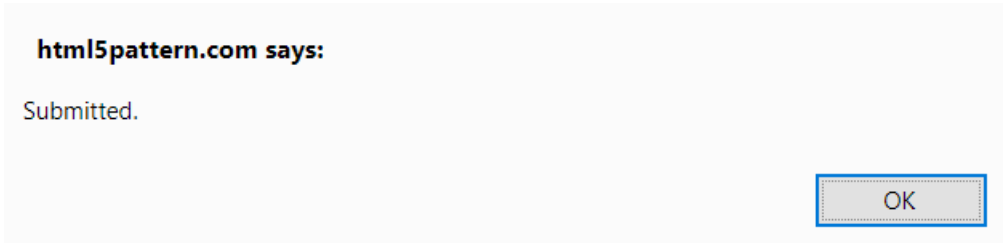
These methods can be called on **Nested Fields** as well for **Sub-Form Submitting**.

12. Domain Validation:

Apply domain validation using the pattern attribute as discussed earlier.

Example: Pattern for color





if input does not match with the domain: