Form Validation

Table of Contents

[User Experience](#_p1to3wvcpzi1)

[A Validation Gameplan](#_ko18pnl5mvat)

[preventDefault](#_cn3wh8xxkjvx)

[Common Checks](#_bj9zi25xbw)

[Anonymous Functions](#_2vgfqw878we3)

[Regular Expressions](#_9ee65q3fbc2r)

[Setting Focus to Inputs](#_bdhyk6ct8cmb)

## User Experience

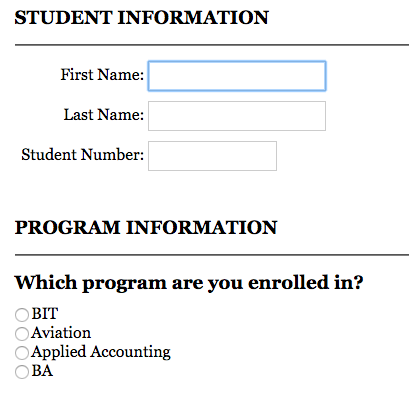
Have you ever had a bad experience filling a form out online? We all have. A lot of these frustrations can be eliminated with proper planning of how a form’s data is validated.

User experience is extremely important. Customers have been lost because of bad form validation strategies. They may have been close to making a purchase, only to stop and leave the website because form instructions were unclear, error messages were vague, and data had to be filled out again.

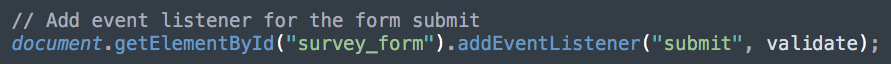
## A Validation Gameplan

Javascript allows us to do client-side validation. That means that before the form data is even sent anywhere, we can check to make sure it meets all of our requirements *before* the form is submitted. But we should allow the users freedom to fill out all the inputs, then check for validity once they click the submit button.

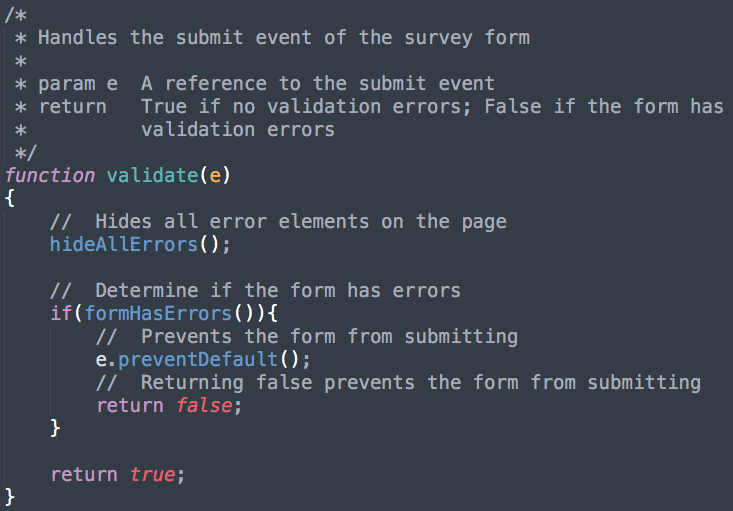
Consider this form below:



There are just three text inputs here, but there are things to consider. What data do we expect? Are all three similar types of data? Can we check all of them at once or do we have to do them separately? What happens when the form is first submitted?



First we have to handle submission of the form. Notice here this is not the *click* event of the button, but the *submit* event of the form. The user can also use the enter key to submit a form. Here we’re running a function called *validate()* when the form is submitted.



### preventDefault

The **preventDefault()** method tells the [user agent](https://developer.mozilla.org/en-US/docs/Glossary/user_agent) that if the event does not get explicitly handled, its default action should not be taken as it normally would be.

If there are errors in the user data, then we must prevent the form from submitting. If there are no errors, then we allow the form to submit. The *formHasErrors()* function will check all the inputs to see if there are any errors and return trueif there are errors, and falseif there aren’t.

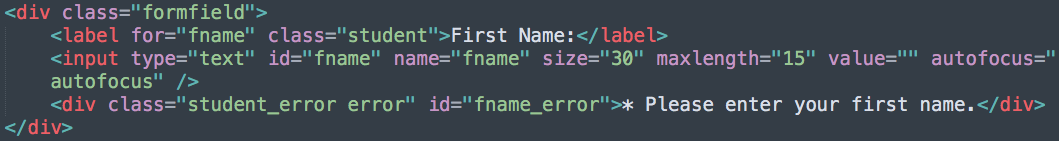
## Common Checks

The most common form-data-validation is checking a textbox for any number of characters.



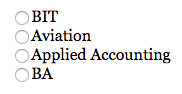
Here we’ve retrieved the value out of the textbox, and checked to see if there was any data entered. Now if there was an issue with the data, we have to display an appropriate error message to the user. This is where checking all validations on submit is an advantage.

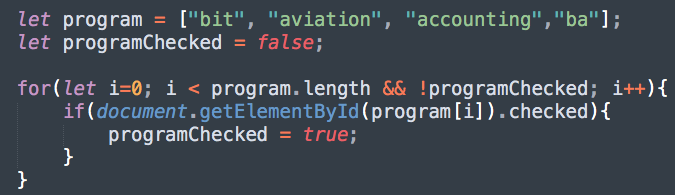
A common way to display error messages is to create them inside the markup, have them hidden with CSS, then display them as needed in Javascript.



Notice the id attribute. We’ve added “\_error” to the id of the input that this error message is tied to. This makes it easier in the Javascript to access the appropriate error message.



Radio buttons and checkboxes can be checked using the *check* property. Here we have four radio buttons, and created an array of their id attributes so we can use a loop to check all of them in one block of code.

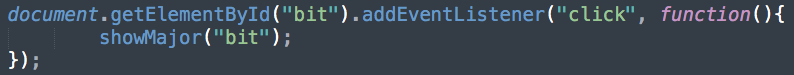


## Anonymous Functions

We’ve used event listeners, but one issue with them is that they are just references to functions; not calls of functions. This means that we can’t pass values to them. But what we can do is create an *anonymous function* that will be executed only when the event takes place. See the example below:

## 

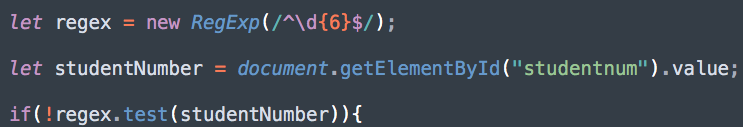
We have a function that should run when a radio button is clicked. But this function is expecting a value. So rather than just referencing this function in our addEventListener, we’ll create an anonymous function, and call the function in there.



Here’s an older but [still relevant article](https://www.javatpoint.com/javascript-anonymous-functions) about anonymous functions.

## Regular Expressions

Regular Expressions are used in validation to determine if data matches a pattern. For example, if we were to say that a student number had to be a 6 digit integer, we could create a regular expression pattern, and then test that pattern against what the user entered, like so:



You can see that regular expression syntax can be funky. If you need regular expressions, some research will show that many people have already created them. Use those in your code!

**NOTE:** In this course we want you to know what regular expressions are and what they can do, but you don’t have to write them yourselves.

## Setting Focus to Inputs

Setting focus to an input is a very simple, yet important feature to add to any form validation. As you can see with the code below, it’s as simple as calling the *focus()* method.



The *select()* will highlight any text within the textbox that may have been in error (an invalid postal code for example).