Using Regular Expressions for Form Validation in JavaScript

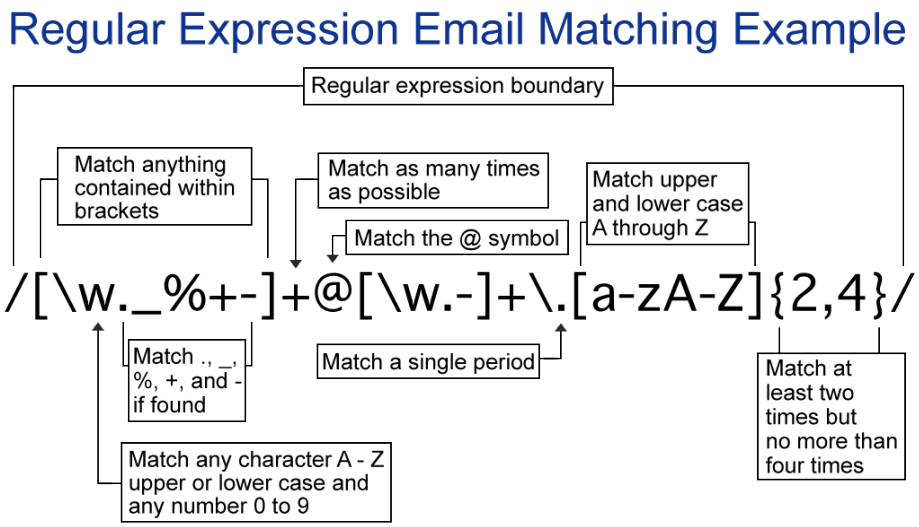
By Chad Jordan – February 28th, 2009

In this guide you will learn:

1. A thorough understanding of the what, how, and why of regular expressions
2. The advantages and uses of implementing regex into JavaScript
3. The fundamentals of creating a basic sign-up form using HTML and CSS
4. A complete implementation using JavaScript to build the functionality of an electronic account form for regex validation.

Introduction

Most people have come to know JavaScript as a front-end programming language for the web, but JavaScript is also a technology used for the back-end. If you read my previous guide on how to write a spam filter in Perl, you saw my brief explanation of using regular expressions for matching email addresses and trimming whitespace.



Regular Expressions *(Regex for short)* comes from mathematics and computer science theory, where it reflects a trait of mathematical expressions called regularity. The text patterns used by the earliest grep tools were regular expressions in the mathematical sense. In programming, the use of regex falls under multiple categories and it has a slightly different set of rules depending on which programming language you're using. Regular expressions are patterns used to match character combinations in strings. However, regardless of which technology you're using it for, most of the same rules apply when implementing it. A regular expression can be a single character, or a more complicated pattern. Regular expressions can be used to perform all types of text search and text replace operations. This is done by using special characters/symbols in very specific ways. Below are some of the most common symbols used in regex syntax.

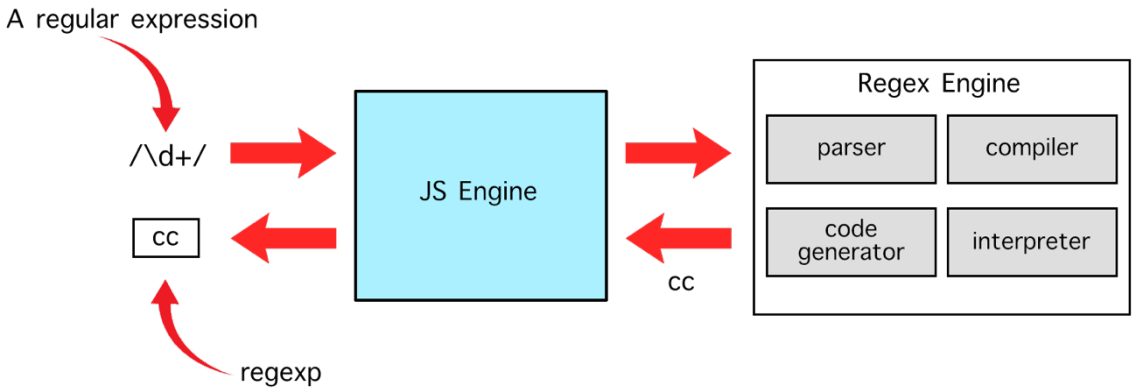
Characters:	**Descriptions:**
****	The backslash quotes the character after it.
.	The dot represents any single character.
*****	The asterisk can represent any character. However, whereas the dot can only represent a single character, the asterisk represents anywhere from zero to an infinite amount of characters.

$	The dollar sign at the end of the regex signifies the end of the line.
^	A circumflex at the beginning means it is the beginning of a line, and any characters immediately following it must be located at the very beginning of the string.
[set]	A set of characters in square parentheses matches any single character from a set.

Aside from the conceptual, and implementation side of regex, this guide will not be too heavy of a read. Just like my previous guides, I'll be using the vim code editor in Linux for my code examples. Regexes are implemented for a lot of different uses, but in this guide, I'll be using them to write functions into validating user input into an account sign-up form.

Regex in JavaScript

In JavaScript, regular expressions are also objects. There are two ways to create a regular expression in Javascript. They can be created either with a RegExp constructor, or by using forward slashes (/) to enclose the pattern. A good example of this is the following: let re = /hi/; The easiest way to create a new RegExp object is to simply use the special regex syntax: myregexp = /regex/. If you have the regular expression in a string (e.g. because it was typed in by the user), you can use the RegExp constructor: myregexp = new RegExp(regexstring). Modifiers can be specified as a second parameter: myregexp = new RegExp(regexstring, "gim") In JavaScript source code, a regular expression is written in the form of /pattern/modifiers where "pattern" is the regular expression itself, and "modifiers" are a series of characters indicating various options. The "modifiers" part is optional, and this portion of the syntax is borrowed from Perl. An example of this is-as follows:
/g *enables "global" matching. When using the replace() method, specify this modifier to replace all matches, rather than only the first one.*
/i *makes the regex match case insensitive.*
/m *enables "multi-line mode". In this mode, the caret and dollar match before and after line breaks in the subject string.*
We need regexes because by using them we can express something that is a generalization rather than something specific. They allow us to unambiguously describe patterns that can be used to have software do things like search for more complex patterns in redundant data. A JavaScript engine is a software component that executes JavaScript code. The first JavaScript engines were mere interpreters, but all relevant modern engines use just-in-time compilation for improved performance. JavaScript engines are typically developed by web browser vendors, and every major browser has one. It outsources the understanding work to another engine called a Regex Engine. I have created a visual to illustrate what's happening:



Building the Sign-up Form

In order to allow the user to input information into a form, we have to first create a webpage with the embedded form in it. In my previous guide on coding a basic website from scratch, I presented the basic concepts of creating simple web pages with HTML and CSS. This process will be similar. I start by writing a container that will house the sign-up form for the user account. Starting on **line 14** I place a header tag for the largest text at the upper-left portion of the form. There has to be an action that takes the inputted information of the form, and feeds it to the CGI file to be stored on the department servers. This is done on **line 16** in the *form action* tag. On the next line, I begin by creating the content on the form with instructions for the *username* input area. Since this is a sign-up form for a user account, this will have multiple input tags of different types, separated by paragraph tags.

```
 1 <!DOCTYPE html
 2 PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 4 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 5
 6   <head>
 7         <link rel="stylesheet" type="text/css" href="style.css" />
 8         <title>Validation Form</title>
 9         <script type="text/javascript" src="load.js">
10         </script>
11   </head>
12   <body>
13   <div id="container">
14   <h1>Start A New Account</h1>
15   <hr/>
16   <form action="http://www.css.taylor.edu/~jgeisler/cos264/form.cgi" onsubmit="return allvalid();">
17                 <p>What is your desired username?</p>
18                 <p>(12 Character max, only letters, numbers and underscores are allowed)</p>
19                 <input type="text" id="username" name="username" onblur="checkusername();"/>
20                 <p>What is your first and last name?</p>
21                 <input type="text" id="fullname" name="fullname" onblur="checkfullname();"/>
22                 <p>Email Address</p>
23                 <input type="text" id="email" name="email_address" onblur="checkemail();"/>
24                 <p>Desired Password</p>
25                 <input type="password" name="password" id="password1"/>
26                 <p>Confirm Password</p>
27                 <input type="password" name="password_confirm" id="password2" onblur="checkpassword();"/>
28                 <p>Credit Card Number</p>
29                 <p>(separate numbers with dashes)</p>
30                 <input type="text" name="credit_card" id="creditcard" onblur="checkcreditcard();"/>
31                 <p>What is your Gender?"</p>
32                 <input type="radio" name="sex" value="male" id="male"/>Male
33                 <br/>
34                 <input type="radio" name="sex" value="female" id="female"/>Female
```

The select tag of the region id input on **line 36** will automatically create a drop-down and once I add the option values, the user can choose from their regional location.

```
35                 <p>Select your Region</p>
36                 <select name="region" id="region" >
37                         <option value="west_coast">West Coast </option>
38                         <option value="middle_land">Middle Area-Pointy Land </option>
39                         <option value="plains">Great Plains </option>
40                         <option value="midwest">Midwest </option>
41                         <option value="gulf_coast">Gulf Coast </option>
42                         <option value="east_coast">East Coast </option>
43                 </select>
44                 <br/>
45                 <br/>
46                 <textarea rows="10" cols="50" readonly="readonly">
47                 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam commodo pulvinar odio. Ut lac
   inia congue tortor. Nam volutpat purus. Nulla ipsum ligula, consequat eu, pharetra quis, imperdiet eu, lib
   ero. Nullam molestie, risus sit amet feugiat consequat, ligula purus porta nisi, eget venenatis justo maur
   is id lectus. In in diam. Fusce pulvinar nisl eu augue. Proin et nisl ut risus tincidunt tincidunt. Donec
   turpis. Cras aliquet. Donec in elit at tellus dapibus fringilla.
48                 Curabitur ornare nisl. Etiam cursus quam vehicula libero. Vivamus arcu elit, fermentum in,
```

On **line 46** the textarea for *readonly* text contains Lorem ipsum text, but only for the sake of this assignment. This area is where you would place the organization's terms and conditions for the user account if this was a real-world project. After the *"terms and conditions"* text is complete, I add a closing tag for *textarea*, add a couple of break tags, and then on **line 55** I add a checkbox with some text for agreeing to the terms and conditions. Then a couple more break tags to add a small gap, followed by the last two remaining input tags on lines **58** and **59** for the option of resetting all input fields, and then the submit button. These are typical elements of any sign-up form. Once the user is ready to submit the information in the form, this data is sent

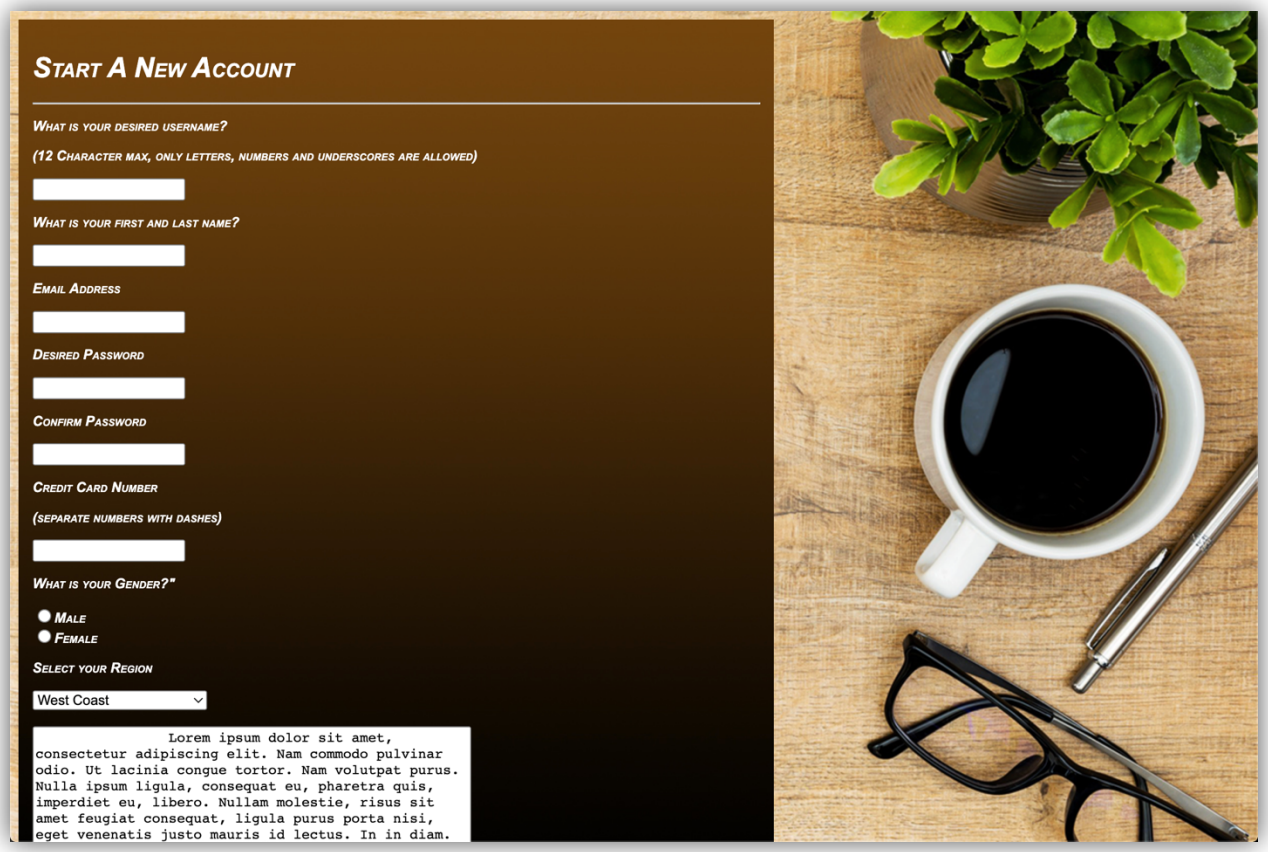
```
   n ante. Nulla facilisi. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus m
   us.
52                 </textarea>
53                 <br/>
54                 <br/>
55                 <input type="checkbox" name="agreement" id="checkbox"/>I Agree to the Terms and Conditions
56                 <br/>
57                 <br/>
58                 <input type="reset" name="reset"/>
59                 <input type="submit" name="submit" id="submit"/>
60                 <br/>
61    </form>
62    </div>
63 </body>
64 </html>
```

to the CGI file that the professor provided on the department server. This is the same file referenced earlier in the **<form action>** tag. With this code in place, my sign-up form is created but it has no style applied to it, and I want this to look pleasant on the webpage which means it's time to add some basic CSS to the page. The following CSS code is written into my style.css file and referenced in the HTML file.

```
 1 body {
 2         background-image: url(bg.jpg);
 3         background-attachment: fixed;
 4         font: italic small-caps 900 10pt arial;
 5         color: white
 6 }
 7 #container {
 8         background-image: url('BrownGradient.jpg');
 9         width: 700px;
10         margin-right: auto;
11         padding: 1em;
12 }
13 h1,h2,h3,h4,h5,h6 {
14         color: #ffffff;
15 }
16
17 .whitetext {
18         border: 3px solid black;
19         background-color: white;
20         color: black;
21 }
22
23 p.whitetext {
24         padding: 4px;
25 }
26
```

I chose this background image for the body of my webpage to hold the sign-up form. The container id holds the content of my form and floats it to the left side of the screen. I made a simple brown gradient background for the validation form in order to give it a little more color for the surrounding page. The brown gradient was just a simple effect I

created for the form. If you have the time to create a much more detailed, transparent design for your form, that can easily be added, and then use my CSS code or write your own to load a design into your form. With my HTML and CSS code in place, this is what my form currently looks like on a webpage. As the page stands, I can interact, scroll, and type within the input fields, but because there is no functionality built into the form, nothing will happen yet.



Writing Regex with JavaScript

Now it's time to breathe some life into the form using JavaScript regular expressions. Basically, I'm going straight down the page of my form and addressing each input field in sequential order. The first is the username. You can see from **line 3** that I set up the regex to check the parameters of the required field, and if this is not met, an alert message pops up letting the user know the only types of characters allowed in the input field before moving on.

```
1  function checkusername(){
2          var usernametag=document.getElementById("username");
3          if(usernametag.value.search(/^[a-zA-Z0-9_]{1,12}$/) !=0){
4                  usernametag.style.color="red";
5                  alert("Username must contain only letters, numbers, or underscores with a maximum count
   of 12 characters.");
6                  return false;
7          }
8          else{
9                  usernametag.style.color="green";
10                 return true;
11         }
12 }
```

The next function on **line 13** handles the full-name element for the first and last name field. I start by writing a function called *checkfullname*, getting the *fullname* element accessing the data attribute of the document, and then assigns it to the variable *fullnametag*. Next, on **line 15** I use a conditional if statement to check for the value of the search, then passing the regex search parameters to the function. If the characters do not match the characters in the expression, then the characters are colored red, and the user is presented with a message on the screen letting them know that they have failed to input the proper information. Otherwise, if it's accepted, then it highlights the characters in green and allows the user to keep inputting more information in the next input fields. Essentially the same checks are performed for every function, just merely altering the regex fields according to the required user input.

```
13 function checkfullname(){
14         var fullnametag=document.getElementById("fullname");
15         if(fullnametag.value.search(/^[a-zA-Z]+ [a-zA-Z]+$/) !=0){
16                 fullnametag.style.color="red";
17                 alert("Full name must contain only letters with a space between the first and last.");
18                 return false;
19         }
20         else{
21                 fullnametag.style.color="green";
22                 return true;
23         }
24 }
25 function checkemail(){
26         var emailtag=document.getElementById("email");
27         if(emailtag.value.search(/^[a-zA-Z0-9_.]+@[a-zA-Z0-9_.]+\.(com|net|org|edu)$/) !=0){
28                 emailtag.style.color="red";
29                 alert("The correct form for the email address would be E.g. Username@domain.ext can contain
   only letters, numbers, or underscores with a maximum count of 12 characters.");
30                 return false;
31         }
32         else{
33                 emailtag.style.color="green";
34                 return true;
35         }
36 }
37 function checkpassword(){
38         var password1tag=document.getElementById("password1");
39         var password2tag=document.getElementById("password2");
40         if(password1tag.value==password2tag.value && password1tag.value.length > 0){
41                 password1tag.style.color="green";
42                 password2tag.style.color="green";
43                 return true;
44         }
45         else{
46                 password1tag.style.color="red";
47                 password2tag.style.color="red";
48                 alert("Passwords must match.");
49                 return false;
50         }
51 }
```

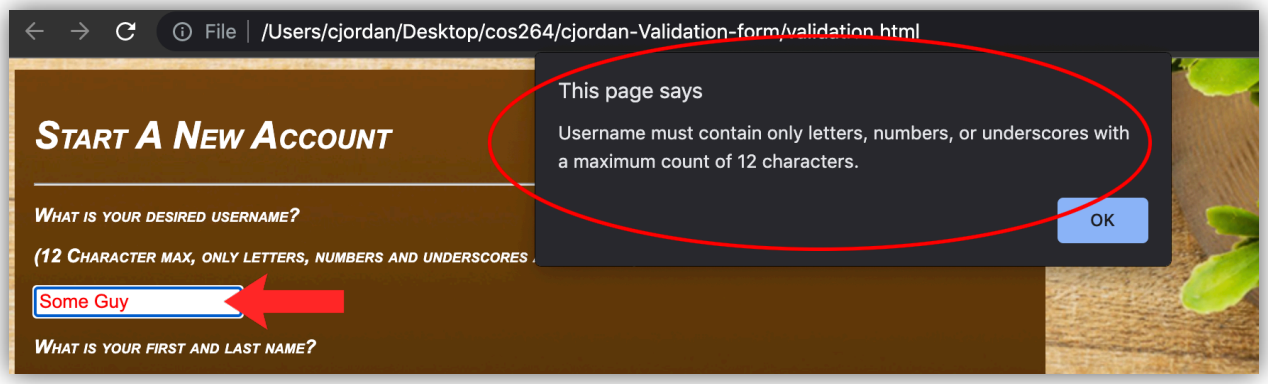
While the user is not expected to input real credit card information for the assignment, the program still has to meet the required credit card format. The regex on **line 54** ensures this.

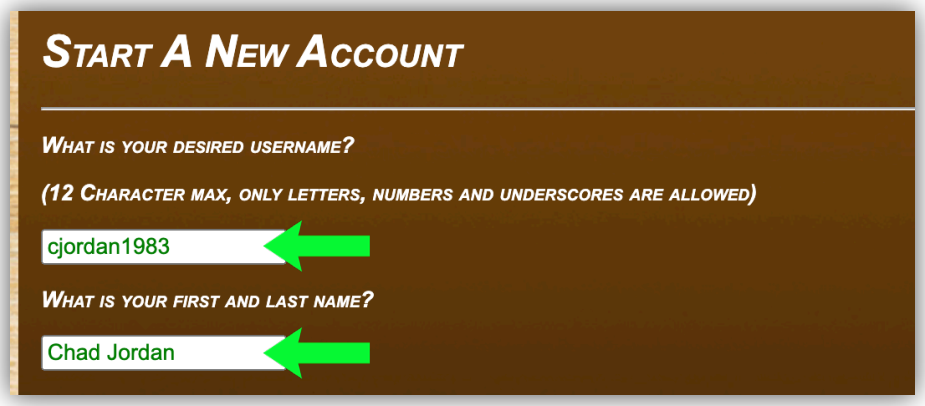
```
52 function checkcreditcard(){
53          var creditcardtag=document.getElementById("creditcard");
54          if(creditcardtag.value.search(/^[0-9]{4}-[0-9]{4}-[0-9]{4}-[0-9]{4}$/) !=0){
55                  creditcardtag.style.color="red";
56                  alert("Credit card must contain only numbers, with a count of 16. Be sure to include
   one hyphen between each set of four numbers.");
57                  return false;
58          }
59          else{
60                  creditcardtag.style.color="green";
61                  return true;
62          }
63 }
```

We know that credit cards have four sets of four numbers that are checked in a numeric range from 0 – 9 and this is exactly how we write the expression. This last function checks for all valid input and once the data attributes are verified, the form is successfully accepted and sent over to the servers.

```
64 function allvalid(){
65         if(!checkusername()) return false;
66         if(!checkfullname()) return false;
67         if(!checkemail()) return false;
68         if(!checkpassword()) return false;
69         if(!checkcreditcard()) return false;
70         var radiomaletag=document.getElementById("male");
71         var radiofemaletag=document.getElementById("female");
72         if(!(radiomaletag.checked || radiofemaletag.checked)){
73                 alert("Must choose one gender");
74                 return false;
75         }
76         var regiontag=document.getElementById("region");
77         var checkboxtag=document.getElementById("checkbox");
78         if(!checkboxtag.checked){
79                 alert("You must agree to our terms and conditions.");
80                 return false;
81         }
82
83         return true;
84 }
85
```

With all of this now in place, the form behaves exactly as it should. We can enter a random string of characters into the username field, not following the format, and sure enough, we get the following response from the form:





Yay! I confirmed the form properly submitted the data. This is a successfully completed regex validation form in JavaScript!

Conclusion

It should go without saying, of course, regexes can add more complexity, but if you consider the amount of time you can also save by using them for dynamically loading large amounts of XML data, it can help to alleviate unwanted redundancies. Regular Expressions are also used in programming higher efficiency in search engines as well as web search results. They are used in natural language processing and can be used in automated trading that analyzes newsfeeds that then make automated buy and sell decisions. There are many uses for implementing them, and while the process of implementing them can be both simplistic, or very complex, there is much validity in using them to remove unneeded redundancy in programming. If you can learn regexes, you can save a lot of time in programming tasks, and in web programming languages like JavaScript, and Perl. Regexes are frequently used for performing back-end procedures. All diagrams and code in this guide were created, written, and provided by Chad Jordan. For any possible inquiries such as general questions regarding this guide or other professional inquiries please feel free to email me at cjordan@wondercreationstudios.com

Resources Used:

- Sebesta, W. Robert - *Programming the World Wide Web – 4th Edition* – 2008
- W3schools.com
- iStock Photo