**Form Validation**

Form validation is one of the most common tasks performed using JavaScript. You have likely come across forms on the Web that have shown you a prompt when you have not entered a value into a field that requires one, or when you have entered the wrong kind of value; this is because the form has been *validated*. That is, a script has checked to see whether the text you have entered or choices you have made match some rules that the programmer has written into the page. For example, if you are expected to enter an e - mail address, these validation rules may check what you entered to ensure that it contains an @ symbol and at least one period or full stop. These kinds of rules help ensure that the data provided by users meets the requirements of the application before being submitted.

**When to Validate:**

Validation can happen in two places: in the browser using JavaScript, and on the server using one of several languages such as ASP.NET or PHP. In fact, applications that collect important information using a form (such as e - commerce orders) are usually validated both in the browser *and* on the server. You may wonder why forms are validated in the browser if they will only get checked again when they reach the server; the reason is that it helps the user enter the correct data required for the job without the form being sent to the server, being processed, and then being sent back again if there are any errors. This has two key advantages:

* It’ s quicker for the user because the form does not need to be sent to the server, processed, and returned to the user with any relevant error messages.
* It saves the load on the server because some errors will get caught before the form is submitted.

It is very important to validate on the server because you cannot guarantee that the user has JavaScript enabled in his or her browser, and if a user entered a wrong value into a database or other program it could prevent the entire application from running properly. (It is also possible for hackers to bypass JavaScript if they are intending to send some incorrect information.)

**What You Can Check For:**

When it comes to validating a form, you cannot always check whether users have given you the correct information, but you can check whether they have given you some information in the correct format. For example, you cannot ensure that the user has entered his or her correct phone number; the user could be entering anyone’s phone number, but you can check that it ’ s a number rather than letters or other characters, and you can check that the number contains a minimum number of digits. As another example, you can’ t ensures someone has entered a real e - mail address rather than a false address, but you can check that whatever was entered followed the general structure of an e - mail address (including an @ sign and a period, and that it is at least seven characters long). So, JavaScript form validation is a case of minimizing the possibility of user errors by validating form controls.

When it comes to form controls that allow users to indicate their choice from a selection of options (such as checkboxes, drop - down select boxes, and radio buttons), you can use JavaScript to check that a user has selected one of the options (for example, to check that a user has checked the terms and conditions).

**How to Check a Form:**

There are several ways in which you can check a form. Usually when the user presses the submit button on a form, it triggers the onsubmit event handler on the < form > element, which in turn calls a validation function stored either in a separate script or in the head of the document. The function must then return true in order for the form to be sent, or, if an error is encountered, the function returns false and the user’ s form will not be sent — at which point the form should indicate to the user where there is a problem with the information the user entered.

*If you use a validation function that is called by the* onsubmit *event handler, but the user ’ s browser does not support JavaScript, then the form will still be submitted without the validation checks taking place.*

In the validation functions you meet in this chapter, the first task will be to set a variable that can be returned to say whether the script found errors or not. At first, this is set to true (indicating that the form can be sent because problems were found); then as the script checks the values the user has entered, if the function finds an error this value can be turned to false to prevent the form from being submitted.

Some forms also check values as the user moves between form fields — in which case the values the user entered are passed to a function that checks that specific form control using the onblur event (which fires when that form control loses focus).

**Checking Text Fields:**

You have probably seen forms on web sites that ask you to provide a username and password, and then to re - enter the password to make sure you did not mistype something.

**For Example (Please refer to the code) it includes:**

* The username is of minimum length
* The password is of minimum length
* The two-password match

Note: All these is created using single function to check the form. In this case, the validate function will live in either <head> or end of <body> element, and will start by setting a variable and returning to true to allow login/submit and false to redirect to the same page i.e. to not allow to login and submit the form ; if no errors are found this will be the value that the function returns, which will in turn allow the form to be sent. If an error is met, the variable will be set to false, and the form will not send.

**Regular Expressions:**

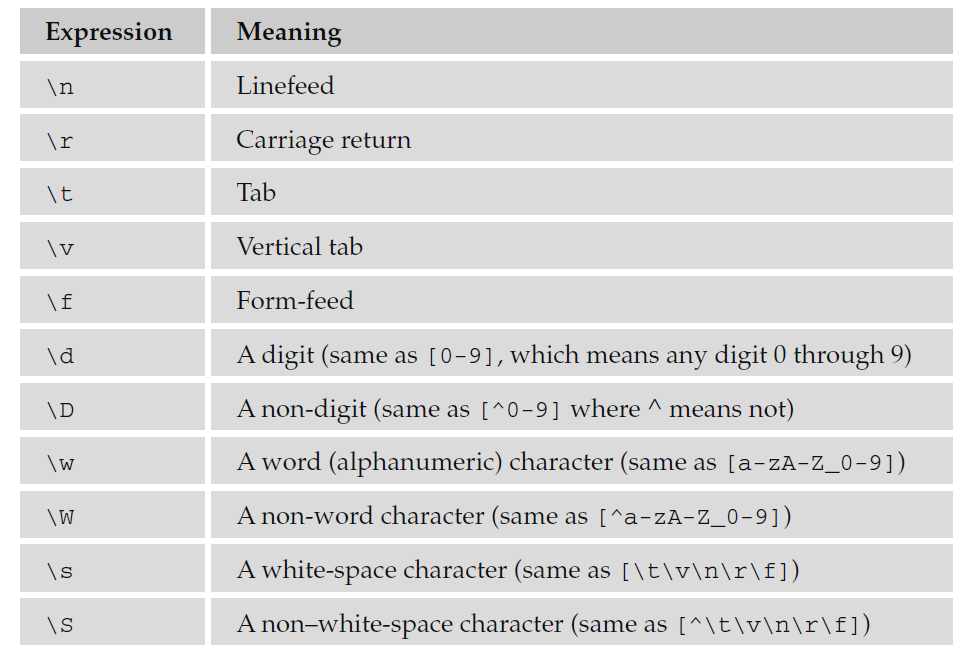
Regular Expressions provide a very powerful way to find a particular string, although the downside of this power is that they can become quite complicated.

Note that if you want to search for any of the following characters, they must be escaped because they have special meanings in Regular Expressions (remember “\.” Example discussed in class instead of “.”):

\ | ( ) [ { ^ $ \* + ? .

If you want to escape these characters, they must be preceded by a backslash (for example /\ \ / matches a backslash and /\$/ matches a dollar sign).

The table that follows lists some other interesting characters used in regular expressions.



**Testing Characters Using test() and Regular Expressions:**

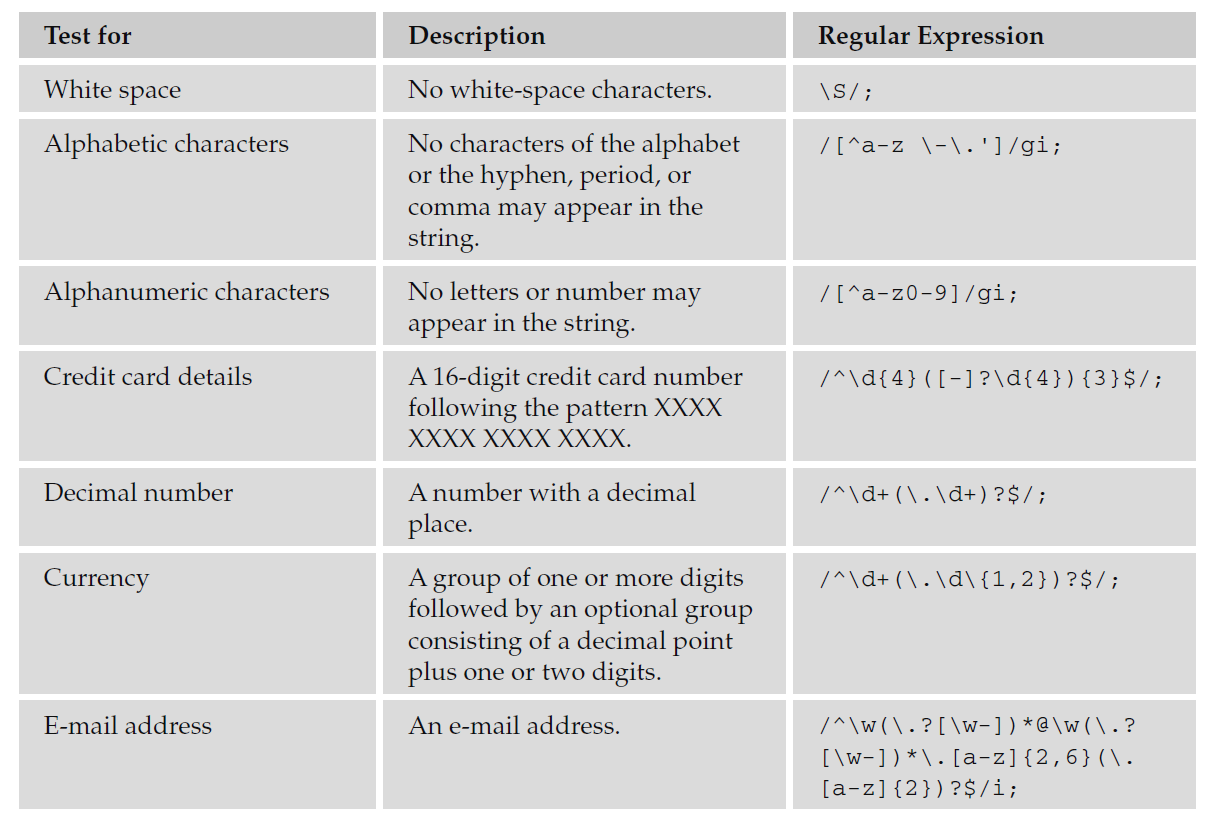
Regular Expressions really come into their own when you need to test whether strings entered by users conform to a pattern. For example, Regular Expressions can be used to test whether the string follows a pattern for e - mail addresses, for an amount of currency, or for a phone number.

To check whether a value a user has entered matches a regular expression, you can use the test() method, the if condition takes two parameters: the Regular Expression and the value the user entered. The test() method returns true if the value entered by the user matches the regular expression, and false if it does not.

Example:



Regular Expressions are not the easiest thing to learn to write, and there are entire books devoted to writing complex expressions. However, the table that follows lists some helpful ones that you can use to get you started. **Note: to ignore the headache you might have understanding the regular expression below follow the approach as discussed in class nevertheless, the below regex (regular expression works fine).**

****

**Select Box Options:**

When you want to work with a drop - down select box, the select object (which represents the

select box) has a very helpful property called selectedIndex , which tells you which option the user has selected.

Because this is an index, it will start at 0, so if the user has selected the first option, the selectedIndex property will have a value of 0. If the user selects the second option, the selectedIndex property will be given a value of 1, the third will be given a value of 2, and so on.

By default, if the user does not change the value that the control has when the page loads, the value will be 0 for a standard select box (because the first option is automatically selected when the form loads). In a multiple select box (which allows users to select more than one option from the list), the default value will be 1 if none of the options are selected (which indicates that the user has not selected any option).

If you wanted to access the value attribute on the selected option (rather than its index number) you would use the following syntax:

(same as code discussed in class but with options property)

var selectedOption = document.frmCards.selCards;

alert(selectedOption.options[selectedOption.selectedIndex].value);

This is because you need to look at which of the [option] elements were selected to get its value rather than just the index number of the selected element.

**Note: For example, please refer to the code.**

**Radio Buttons:**

A group of radio buttons is different from other form controls in that only one option from a group can be selected at a time, and all members of the group share a value for the name attribute. Scripts that interact with radio buttons usually want to either check that one of the options has been selected, or find out which of the options has been selected.

A set of radio buttons is represented as an array in JavaScript, and in order to find out which one was selected, you need to loop through the array, looking at the checked property of each radio button. If it is selected, the value will be true and false if not.

In order to loop through each of the radio buttons in the collection and see which one has a checked property, you will use a for loop.

Another way to ensure that one of the options is selected is to preselect an option when the page loads.

**Note: For example, please refer to the code.**

**Checkboxes:**

Checkboxes allow a user to select zero, one, or more items from a set of choices (they are not mutually exclusive as radio buttons are). As with radio buttons, when a group of checkboxes share the same name, they are made available in JavaScript as an array.

**Notes: There are two examples explained in code go through it properly**