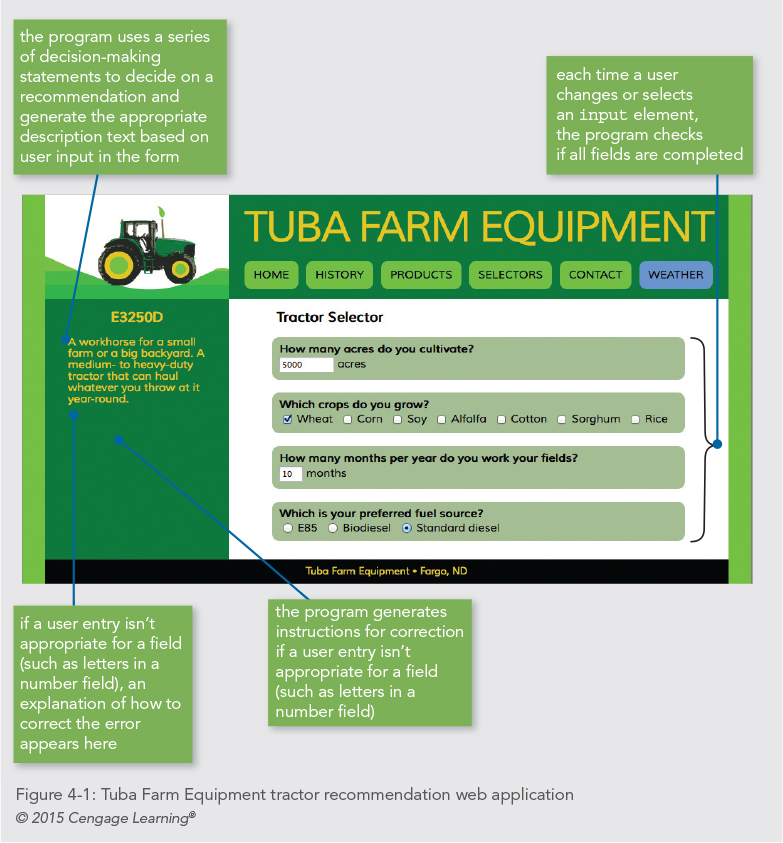
**Exercise 01\_04\_01 – Step 1**

The following web application has been coded, but it doesn’t work as expected. We will use debugging techniques to identify and fix the bugs in the program so it functions as designed.

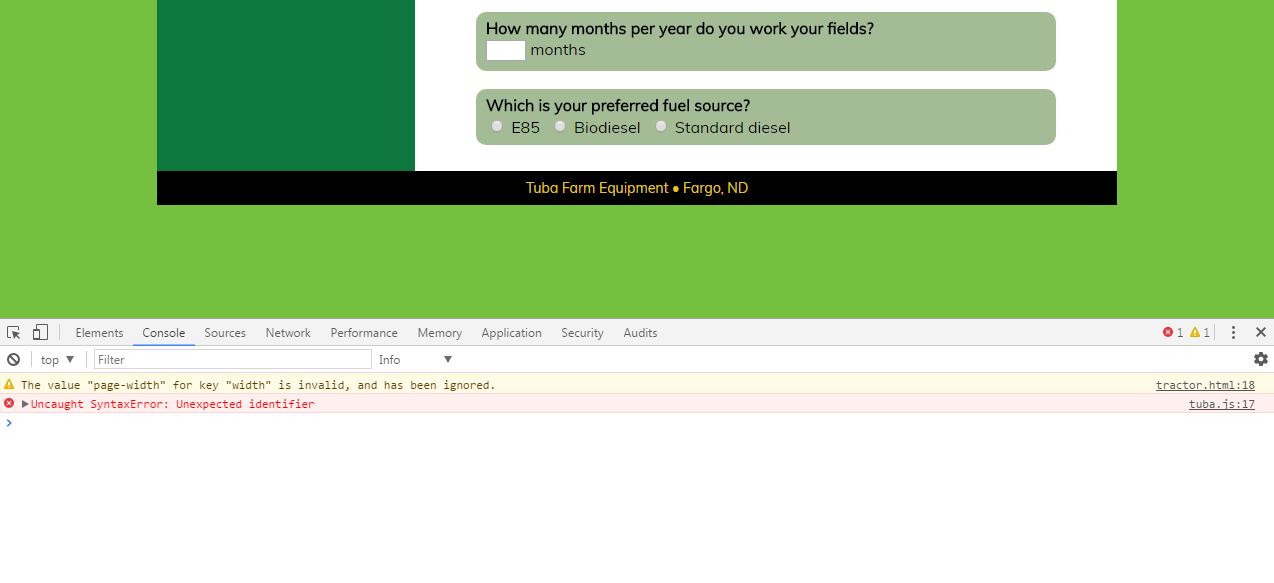
 

The application accepts user answers to four questions. Each time a user types a character or clicks a box, the program uses a series of decision-making statements to identify the tractor model that best fits the user’s answers. It then adds the model name and description to the sidebar. Inappropriate data entry causes the display of an explanation of how to correct the input.

The starting version does not work as planned. There are several bugs in it. the starting point is to open and test the application.

1. Open ***tractor.html***; fill in the appropriate documentation in the top comments. Scroll through the document to get familiarized with the content.
   1. In particular, look at the ***<aside>*** element near the bottom. It contains two empty elements, an ***<h2>*** and a ***<p>*** element. The program will populate them with either a warning or a recommendation
2. Open ***tuba.js***; fill in the appropriate documentation in the top comments. Scroll through the document to get familiarized with the content. For the moment, ***ignore any syntax errors*** you might spot in the code.
   1. There are twelve global variables; four of them hold one each of the four control ***<fieldset>*** elements.
   2. There are four functions which are designed to test the completeness of each of one of the ***<fieldset>*** groups. We will code them to do this, then they will generate the output for the ***<aside>*** elements.
   3. Notice at the bottom, that there is a function that creates the event listeners for the controls, and a command which calls it. both take into account the two different ***DOM Event Models***.
3. Open ***tractor.html*** in the Chrome browser. Place some input into each of the controls. Notice that no recommendations are displayed in the sidebar. Also notice that there are no error messages, even if more than 12 months are entered.

**Exercise 01\_04\_01 – Step 2**



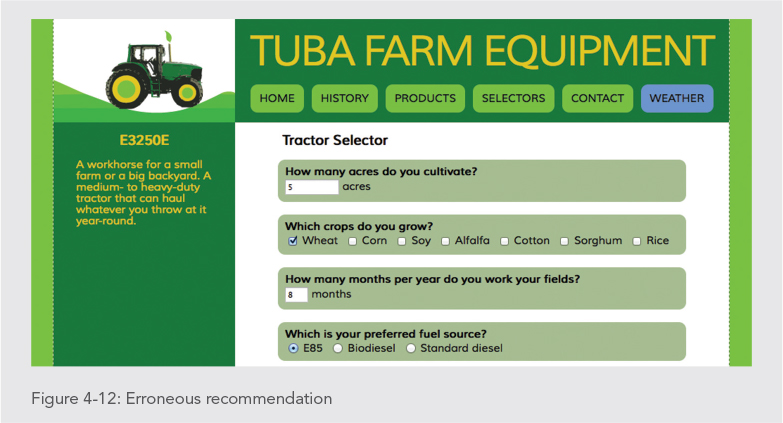
1. Make sure ***tractor.html*** is open in the browser. Use the Chrome ***Inspect*** option, then open up the ***Console*** tab. Ignore the warning. The error message indicates that there is a syntax error in ***tuba.js*** on line 17. Chrome specifies “Unexpected identifier” as the error, but it is really up to you to figure it out. JSLint can be much more helpful.
   1. Return to ***tuba.js*** in the IDE. At line 17, the declaration of ***fuelComplete*** is missing the letter ***v*** in the ***var*** keyword.
   2. Fix the error and save the file. If you are using a JS Lint package in your IDE, that will be a helpful indicator. Cycle the browser.
2. The Console now shows another error message. The error message indicates that there is a syntax error in ***tuba.js*** on line 26, once again “Unexpected Identifier” is Chrome’s reading on it.
   1. Return to ***tuba.js*** in the IDE. At line 26, the declaration of ***monthsFieldset*** is missing an assignment operator.
   2. Fix the error, save the file, and cycle the browser.
3. The Console now shows another error message. The error message indicates that there is a syntax error in ***tuba.js*** on line 27. This time “Invalid or unexpected token” appears. My Brackets JSLint is much more helpful.
   1. Return to ***tuba.js*** in the IDE. At line 27, we can see, not easily, that the parameter to the function call has an unclosed string.
   2. Fix the error, save the file, and cycle the browser.
4. The next error message indicates a syntax error on line 34. It is much more granular and helpful, this time giving us a clue to the problem being a token, a parentheses. JSLint is quite a bit better.
   1. We can see, more easily, that the parameter list is missing an opening parentheses.
   2. Fix the error, save the file, and cycle the browser.
5. The browser console indicates no more errors. Not so, JSLint, it is quite a bit pickier, which is a good thing.

**Exercise 01\_04\_01 – Step 3**



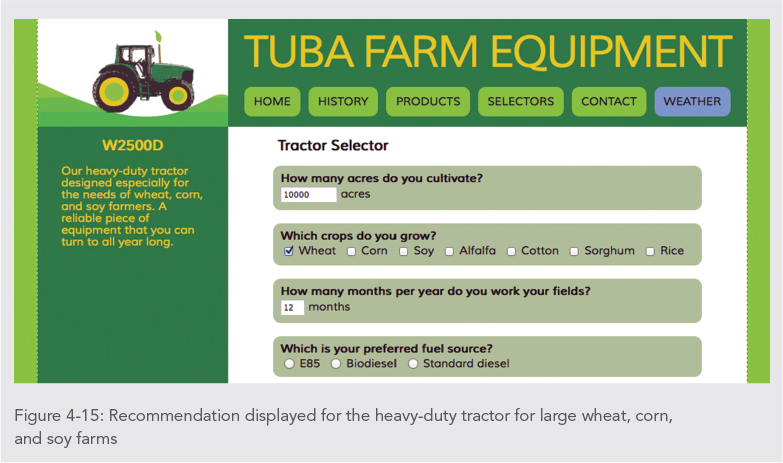
1. Make sure ***tractor.html*** is open in the browser. Enter ***5*** in the ***acres*** box, check the ***Wheat*** box, enter ***8*** in the ***months*** box, and click the ***E85*** button.
   1. The option which has been selected is the W1205E. We know this is incorrect, because the specs say that 5 acres is a small farm and it is picking a large farm model.
   2. We have a logic error which we need to fix.
2. Return to ***tuba.js*** in the IDE and locate the function that is named ***createRecommendation()***.
   1. Strategically place one ***alert()*** just below the first ***if*** statement, the one that is commented to look for acreage less than 5000.
   2. Place a second ***alert()*** just below its ***else*** clause, which is commented to look for more than 5000 acres.
   3. Save the file and run it in the browser. If we enter the same data, we see that its value is correct, but it executes on the ***else*** statement. That is not logically correct.
3. Return to the ***tuba.js*** file and examine the first ***if*** statement. After careful examination, we see that we have used an incorrect expression: ***acresBox.value >= 5000***. It should be corrected to use a ***<=*** operator. Correct it, save, and run it. Then, if it is working, remove the debug code.

**Exercise 01\_04\_01 – Step 4**



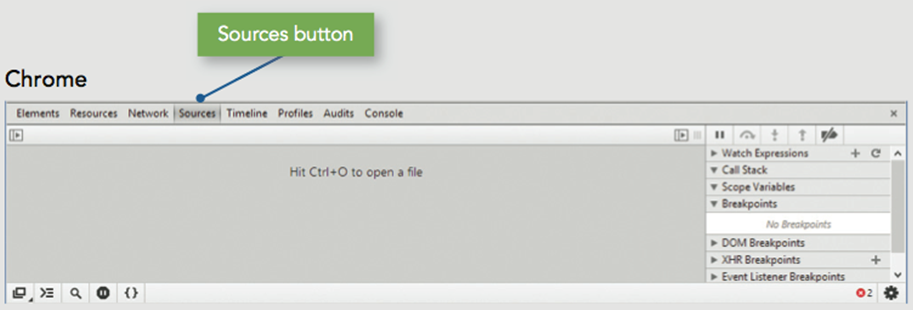
1. Make sure ***tractor.html*** is open in the browser. Enter ***5*** in the ***acres*** box, check the ***Wheat*** box, enter ***8*** in the ***months*** box, and click the ***E85*** button.
   1. The option which has been selected is the E3250E. We know this is incorrect, because the specs say that is a year-round model.
   2. We have a logic error which we need to fix.
2. Return to ***tuba.js*** in the IDE and locate the function that is named ***createRecommendation()***.
   1. Below the third line of the function, the ***if*** statement that checks the value in the months box, enter a ***console.log()*** statement. Enter another one below its matching ***else***.
   2. Save the work, recycle the browser, and make sure the console is open. Type a ***5*** in the acr***e***s box. A console message should appear, indicating the value of ***monthsBox*** is null. Good, it has not been entered yet.
   3. Click the ***months*** box and enter a ***12***. The console first shows a ***1***, followed by ***12***. That is because the event listener fires after each input. But notice that the first one fired on the ***if*** clause and the second fired on the ***else***. That is a logic error.
3. Return to the ***tuba.js*** file and examine the ***if*** statement. Once again, we see that we have used an incorrect relational expression. It should be corrected to use a ***<=*** operator. Correct it, save, and run it. Then, if it is working, remove the debug code.

**Exercise 01\_04\_01 – Step 5**



1. Make sure ***tractor.html*** is open in the browser. Enter ***10000*** in the ***acres*** box, enter ***12*** in the ***months*** box, check the ***Wheat*** box. The application correctly recommends a machine for wheat, corn, and soy.
2. Uncheck the ***Wheat*** box and check the ***Corn*** box. The app incorrectly recommends a general use machine. Rechecking the ***Wheat*** box with the ***Corn*** box is again correct. We have a logic error which we need to fix.
3. Return to the ***tuba.js*** file and locate the line:  
   ***if (document.getElementById("wheat").checked || document.getElementById("corn").checked && document.getElementById("soy").checked)***
4. Comment out as follows:  
   if (document.getElementById("wheat").checked /\* || document.getElementById("corn").checked && document.getElementById("soy").checked \*/) {  
   Save changes, and again enter ***10000*** in the ***acres*** box, enter ***12*** in the ***months*** box, check the ***Wheat*** box. The application correctly recommends a machine for wheat, corn, and soy.
5. Return to the ***tuba.js*** file and comment out as follows:  
   ***if (document.getElementById("wheat").checked || document.getElementById("corn").checked /\* && document.getElementById("soy").checked \*/) {***  
   Save changes and again enter ***10000*** in the ***acres*** box, enter ***12*** in the ***months*** box, check the ***Corn*** box. The application correctly recommends a machine for wheat, corn, and soy. This indicates the first two conditions are correct, the bug must be in the condition or operator still commented out.
6. Return to the ***tuba.js*** file and we can see that after the opening comment symbol, there is an ***&&*** operator. Replace it with the ***||*** operator, and remove the comment symbols.  
   Save changes and again enter ***10000*** in the ***acres*** box, enter ***12*** in the ***months*** box, check the ***Soy*** box. Correct. Check the ***Corn*** and ***Wheat*** boxes. Correct

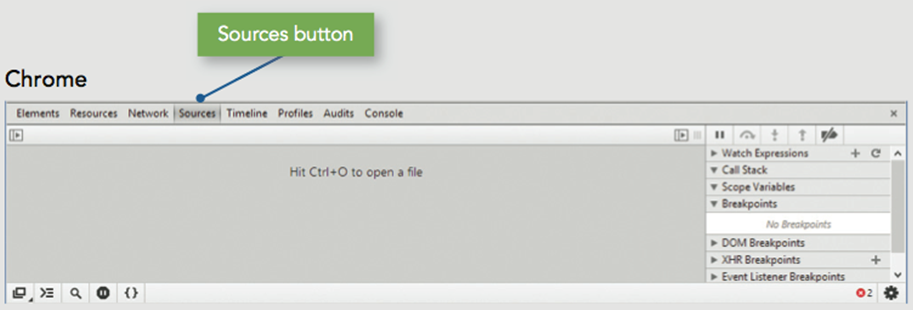
**Exercise 01\_04\_01 – Step 6**



**Setting Breakpoints**

1. Make sure ***tractor.html*** is open in the browser. In Chrome, open the Developer Tools and go to the Sources tab. Click on the ***tuba.js*** file in the list and it will be displayed.
2. Enter ***100*** in the ***acres*** box, check the ***Wheat*** box, enter ***12*** in the ***months*** box, and click ***E85***.
3. In the debugger tools, scroll down to line ***63***, and click it to set a breakpoint. Repeat for lines ***64***, ***67***, ***71***, ***72***, and ***75***. You can see the breakpoints in the list at the right of the screen.
4. In the web form, click at the end of the ***acres*** entry and add a zero to make it ***1000***. The event listener fires, and the first breakpoint is highlighted. The tools on the right indicate that the app ***Paused on breakpoint***.
5. In Chrome, click the ***blue arrow*** at the top to resume. Execution will pause at the next breakpoint. Hit resume again, the pause indicator disappears, as does the resume control. This indicates that execution has resumed and no other breakpoints were hit.
6. In the web form, click at the end of the ***acres*** entry and add a zero to make it ***10000***. The event listener fires, but the first breakpoint is now line ***71***. Resume stops at line ***75***, skipping the breakpoint at line ***72***. Continue until no more breakpoints are hit.
7. To remove a breakpoint, click the line number, such as line ***63***. Right click any breakpoint in the right hand list, and click ***Remove all breakpoints*** to clear the rest.

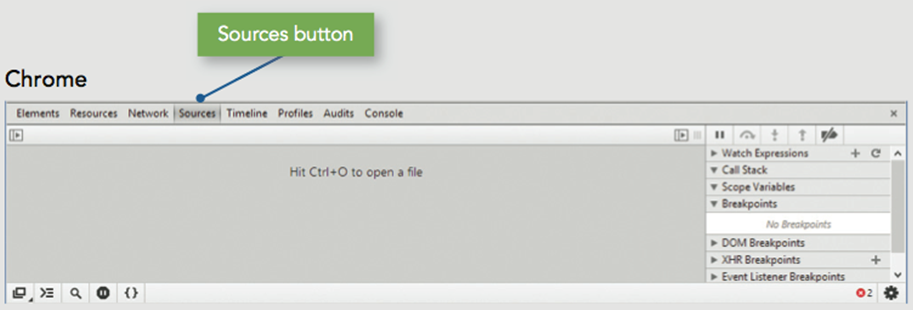
**Exercise 01\_04\_01 – Step 7**



**Stepping Through Scripts**

1. Make sure ***tractor.html*** is open in the browser. In Chrome, open the Developer Tools and go to the Sources tab. Click on the ***tuba.js*** file in the list and it will be displayed. Scroll to line ***50*** and set a breakpoint.
2. Click the ***E85*** option button, the event listener triggers, and the program stops at the breakpoint. Click the ***Step into*** tool on the top of the right sidebar. The highlight moves into the first statement of the function that was highlighted by the breakpoint.
3. Click ***Step into*** again, and the next line is highlighted, which is another function call. Click the ***Step over*** tool. The highlighted function is skipped over, highlighting goes to the end of the function it was in. Click ***Step into*** again, and the highlight goes to the end of the function set as the first breakpoint.
4. Click the ***Step out*** tool, and debugging comes to an end.

**Exercise 01\_04\_01 – Step 8**

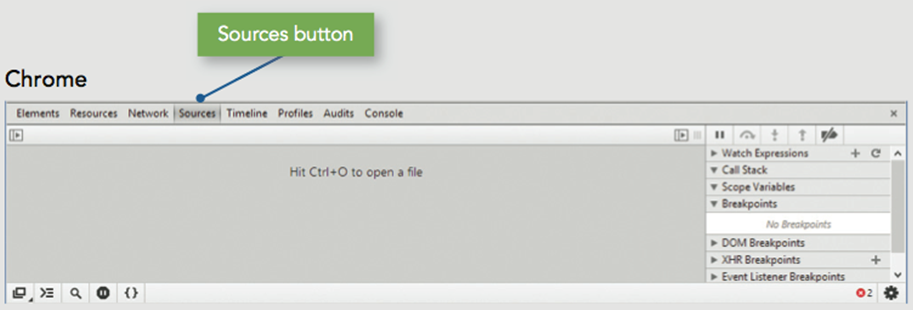


**Tracing Variables and Expressions**

1. Make sure ***tractor.html*** is open in the browser, with developer tools on and in the debugger. Make sure all breakpoints are cleared. Scroll to line ***84*** and set a breakpoint.
2. In line ***85***, highlight the text messageHeadElement.innerHTML. Copy it to the clipboard using ctrl-c or command-c. In chrome, expand the ***Watch*** section, hit the ***+*** symbol for add. Paste in the clipboard contents and hit ***enter***.
3. Click the ***E85*** option button, the event listener triggers, and the program stops at the breakpoint. Click the ***Step into*** tool and the highlight moves into the next line. Click the ***Step into*** again, and the highlighted expression is executed. Click the ***Resume execution*** tool, and script execution finishes.
4. Click the ***Biodiesel*** option button, then click ***Step into***. event listener triggers, and the program stops at the breakpoint. Click the ***Step into*** tool and the highlight moves into line ***86***, the ***else*** statement.
5. Click the ***Step into*** again, and line ***87*** is highlighted. Once more and the highlighted expression is executed. However, our web page shows a machine recommendation of just ***B***, indicating we have another bug.
6. Click ***Resume execution*** and script execution finishes.
7. Go back to tuba.js and look at line 87. The assignment statement ***=*** should be changed to ***+=*** compound assignment statement for concatenation. Save this and test by entering ***10000*** in the ***acres*** box, check the ***Wheat*** box, enter ***12*** in the ***months*** box, and click ***Biodiesel***.

**Exercise 01\_04\_01 – Step 9**

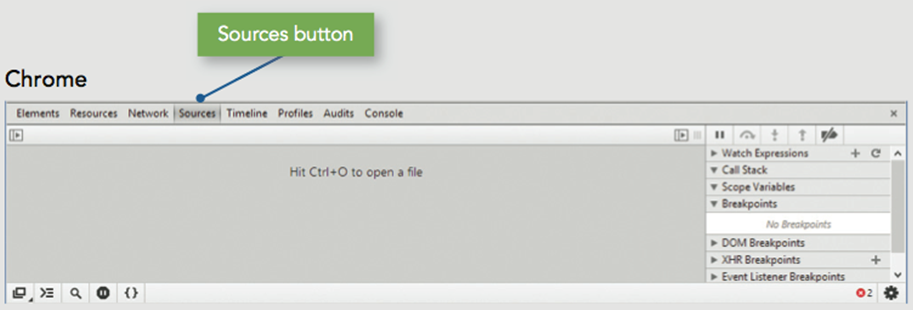
**Exercise 01\_04\_01 – Step 9**



**Examining the Call Stack**

1. Make sure all breakpoints are cleared. Scroll to line ***50*** and set a breakpoint. Click the ***E85*** button, the event listener triggers, and the program stops at the breakpoint. The ***verifyFuel()*** function is added to the call stack.
2. Click ***Step into***, ***testFormCompleteness()*** is added to the top of the call stack, and the highlight moves its first statement. The ***verifyFuel()*** function remains below it.
3. Click the ***Step into*** again until execution advances to line ***62***. The ***createRecommendation()*** function has executed and is placed on top of the call stack. We say it is ***pushed*** onto the stack.
4. Click ***Step into*** again until execution returns to about line ***56-58***. The ***createRecommendation()*** function has finished, and it is ***popped*** off the stack.
5. Click ***Step into*** again until execution returns to about line ***50-51***. The ***testFormCompleteness()*** function has finished, and it is ***popped*** off the stack.
6. Click ***Step into*** again until execution finishes. The ***verifyFuel()*** function has finished, and it is ***popped*** off the stack. Remove the breakpoint from line ***50***.

**Exercise 01\_04\_01 – Step 10**



**Exception Handling**

1. Run the app in the browser. Cycle the acres down to ***-1***, and there is nothing that stops this.
2. Return to ***tuba.js*** in the IDE. Locate the verifyAcres() function and delete its contents, so the function is empty. Substitute in a couple of variable declarations:  
    **var validity = true;  
    var messageText = "";**
3. Return to ***tuba.js*** in the IDE. Locate the verifyAcres() function and delete its contents, so the function is empty. Substitute in a couple of variable declarations
4. Below the variable declarations, enter a ***try*** clause with a ***throw*** statement:  
    **try {  
    if (!(acresBox.value > 0)) {  
    throw "Please enter a number of acres greater than 0.";  
    }  
    }**  
   Cycle this in the browser and turn on the Console. You should have an error message. A ***try*** clause must have at least one ***catch*** or ***finally*** clause to be valid.
5. Below the ***try*** statement, enter the following ***catch*** statement:  
    **catch(message) {  
    validity = false;  
    messageText = message;  
    // remove erroneous entry from input box  
    acresBox.value = "";  
    }**Cycle this in the browser and turn on the Console. The error message should have gone away now that there is at least one ***catch*** clause. When we test it we can enter positive numbers. However, it will not allow us to cycle down below ***1***.
6. We should be nice programmers and inform the users of their folly. Below the ***catch*** clause, let’s put in a ***finally*** block:  
    **finally {  
    acresComplete = validity;  
    // remove former recommendation  
    messageElement.innerHTML = messageText;  
    messageHeadElement.innerHTML = "";  
    testFormCompleteness();  
    }**Cycle this in the browser and give it a test.
7. Now let’s use the same technique to bulletproof the ***verifyMonths()*** function with the following code:  
   ***/\* verify months text box entry is between 1 and 12 \*/  
   function verifyMonths() {  
    var validity = true;  
    var messageText = "";  
    try {  
    if (!(monthsBox.value >= 1 && monthsBox.value <= 12)) {  
    throw "Please enter a number of months between 1 and 12.";  
    }  
    }  
    catch(message) {  
    validity = false;  
    messageText = message;  
    // remove erroneous entry from input box  
    monthsBox.value = "";  
    }  
    finally {  
    monthsComplete = validity;  
    // remove former recommendation  
    messageElement.innerHTML = messageText;  
    messageHeadElement.innerHTML = "";  
    testFormCompleteness();  
    }  
   }***Cycle this in the browser and give it a test.